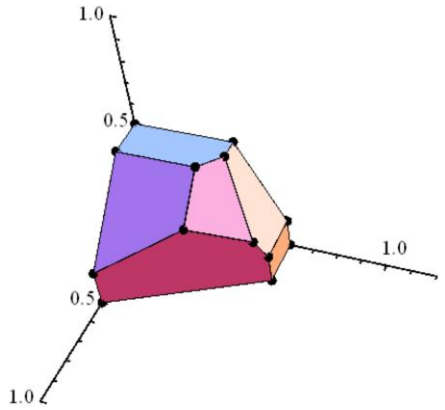
Univerzitet u Nišu
Prirodno-Matematički fakultet

Predrag S. Stanimirović
Nebojša V. Stojković
Marko D. Petković

MATEMATIČKO PROGRAMIRANJE

x	t_2	-1		
$\frac{1}{2}$	$-\frac{1}{2}$	$\frac{3}{2}$	=	$-t_1$
$-\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	=	-z
1	1	1	=	-y
$\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{3}{2}$	=	-F



ISBN: 86-83481-46-8

Univerzitet u Nišu
Prirodno-Matematički fakultet

Predrag S. Stanimirović
Nebojša V. Stojković
Marko D. Petković

MATEMATIČKO PROGRAMIRANJE

Niš 2007.

P.S. Stanimirović, N.V. Stojković,
M.D. Petković

MATEMATIČKO PROGRAMIRANJE

Univerzitet u Nišu
Prirodno-Matematički fakultet

Predrag S. Stanimirović
Nebojša V. Stojković
Marko D. Petković

MATEMATIČKO PROGRAMIRANJE

Niš 2007.

Dr Predrag S. Stanimirović, red.prof. Prirodno-matematičkog fakulteta u Nišu
Dr Nebojša V. Stojković, docent Ekonomskog fakulteta u Nišu
Marko D. Petković, istraživač-pripravnik Prirodno-matematičkog fakulteta u Nišu

MATEMATIČKO PROGRAMIRANJE

Izdavač: Prirodno-matematički fakultet u Nišu
Višegradska 33 Niš
http:/www.pmf.ni.ac.yu

Recenzenti: Dr Gradimir Milovanović
redovni profesor Elektronskog fakulteta u Nišu, dopisni član SANU
Dr Stojan Bogdanović
redovni profesor Ekonomskog fakulteta u Nišu

Odlukom Nastavno-naučnog veća Prirodno-matematičkog fakulteta u Nišu, br. 643/3-01 od 19.9.2007., ovaj rukopis je odobren za štampu kao monografija

CIP – Катаголизација у публикацији
Народна библиотека Србије, Београд

519.85

СТАНИМИРОВИЋ, Предраг С.
Matemaričko programiranje / Predrag S. Stanimirović, Nebojša V. Stojković, Marko D. Petković. – Niš : Prirodno – matematički fakultet, 2007 (Niš : Grafika Galeb).
IV, 415 str. : ilustr. ; 25 cm

Tiraž 100. – Bibliografija uz svako Poglavlje. – Registar.

ISBN 978-86-83481-46-0
1. Стојковић, Небојша В. 2. Петковић, Марко Д
а) Математичко програмирање
COBISS.SR – ID 143763980

ISBN 978-86-83481-46-0

Tehnička obrada: Autori

Tiraž: 100 primeraka

Štampa: Grafika Galeb Niš

Sadržaj

Glava 1

Linearno programiranje

U prvom poglavlju ove glave dajemo matematički model problema linearnog programiranja i opisujemo uobičajene pretpostavke pod kojima se problem rešava. Zatim ćemo proučiti definiciju i rešavanje problema linearnog programiranja. To najpre podrazumeva matematički model i osnovne definicije, kao i osnovne osobine skupa dopustivih rešenja. Zatim ćemo navesti najjednostavniji geometrijski metod za rešavanje problema linearnog programiranja i pokazati kako se on može implementirati.

U drugom poglavlju ćemo proučiti simpleks metod za rešavanje problema linearnog programiranja u opštem obliku i detaljno ćemo opisati sve njegove faze kao i dualni simpleks metod. Takođe ćemo razmotriti još jednu verziju simpleks metoda, revidirani simpleks metod, kojom se rešava problem numeričke stabilnosti klasičnog simpleks metoda. Na kraju, ukazaćemo na problem cikliranja kao i na dva načina da se taj problem prevaziđe. Takođe ćemo pokazati da je simpleks algoritam, i pored svojih odličnih osobina na praktičnim problemima, eksponencijalne složenosti.

Treće poglavlje predstavlja naše originalne rezultate preuzete iz radova [58, 78, 70], i bavi se modifikacijama i poboljšanjima pojedinih faza simpleks metoda. U radovima [70, 78] korišćen je taj algoritam, jer ne zahteva uvođenje veštačkih promenljivih. U tim radovima su uvedena dva algoritma za dobijanje početnog bazično dopustivog rešenja u fazi I dvofaznog simpleks algoritma opisanog u [51] i [79]. Opisano je novo pravilo za izbor bazičnih i nebazičnih promenljivih, tj. za izbor promenljive koja ulazi u bazu kao i promenljive koja napušta bazu. Na kraju ove glave detaljno ćemo opisati implementaciju simpleks i revidiranog simpleks metoda, kao i rezultate testiranja programima MarPlex i RevMarPlex koje su nastale kao rezultat implementacije modifikacija uvedenih u radovima [58, 78, 70]. Pokazaćemo da su uvedene modifikacije u praksi pokazale bolje performanse u odnosu na klasične algoritme.

U čevrtom poglavlju se izučava postoptimalna analiza simpleks metoda.

U petom poglavlju razmatramo direktne metode za rešavanje problema linearnog programiranja. Dajemo rezultate iz našeg rada [76] i rada [67] gde ćemo ukazati na

moguću primenu naših rezultata iz teorije generalisanih inverza [59, 60, 71, 61].

Na kraju ćemo dati kôd najvažnijih procedura programa MarPlex, RevMarPlex i GEOM uz objašnjenje implementacionih detalja.

1 Opšti zadatak linearnog programiranja

Zadatak linearnog programiranja je da odredi maksimum (minimum) linearne funkcije koja zavisi od više promenljivih pod uslovom da su ove promenljive nenegativne i da zadovoljavaju linearna ograničenja u obliku jednačina i/ili nejednačina. Linearna ciljna funkcija koja se optimizira naziva se *ciljna funkcija* ili funkcija cilja . Linearno programiranje je jedan od najefikasnijih pristupa formulisanju i rešavanju složenih problema donošenja odluka i kao takvo predstavlja osnovnu disciplinu operacionih istraživanja.

Linearno programiranje se javilo u oblasti rešavanja praktičnih zadataka kao što su planiranje ekonomskog razvoja kako na nivou radne organizacije tako i na širem regionalnom ili najširem društvenom planu.

Opšte prihvaćeno mišljenje je da je prvi rad koji pripada linearnom programiranju objavio L.V. Kantorovič 1939. godine [35]. U njemu se prvi put definiše transportni zadatak. Među prve radove iz ove oblasti spada i rad pukovnika Vlastimira Ivanovića iz 1940. godine, pod nazivom "Pravila za proračun potrebnog broja transportnih sredstava" [34]. U ovom radu, pukovnik Ivanović je pokazao kako se izračunava minimalan broj vozila za prevoz date količine materijala korišćenjem "Principa najveće ekonomije". Po nekim autorima se nastanak linearnog programiranja vezuje za članak [24] iz 1931. godine. Linearni modeli se koriste i u rešavanju određenih transportnih zadataka, koji su poznati u teoriji i praksi kao transportni problemi, kao i u drugim oblastima planiranja. Prvi algoritam za rešavanje transportnog problem razradio je F.F Hitchock u radu [31] 1941. godine. Na formulisanju transportnog problema i njegovom rešavanju radio je i T.C. Koopmans, koji je rezultate svojih istraživanja objavio 1947. godine. Najveći doprinos razvoju linearnog programiranja dao je G.B. Dantzig, koji je 1947. godine formulisao opšti problem linearnog programiranja i postavio simpleks metod. U radu [20] su izložene osnove simpleks metoda. Radovi John von Neumanna iz tog perioda su omogućili teorijsko formulisanje dualnog problema kao i pronalaženje veze između linearnog programiranja i teorije igara. Metod za otklanjanje degeneracije predložen je u radu [12]. Značajni su i radovi S. Gass-a i T. Saaty-a iz 1955. o parametarskom programiranju kao i radovi E. Beale-a i R. Gomory-a iz 1958. o celobrojnom programiranju.

Problemi matematičkog programiranja javljaju se u različitim disciplinama. Na primer, menadžer na berzi mora da odabere ulaganja koja će generisati najveći mogući profit a da pri tome rizik od velikih gubitaka bude na unapred zadatom nivou. Menadžer proizvodnje organizuje proizvodnju u fabrici tako da količina proizvoda i kvalitet budu maksimalni a utrošak materijala i vremena i škart minimalni, pri čemu ima na raspolaganju ograničene resurse (broj radnika, kapacitet mašina, radno vreme). Naučnik pravi matematički model fizičkog procesa koji najbolje opisuje određenu fizičku pojavu, a na raspolaganju ima konačni broj mernih

rezultata. Takođe, model ne sme biti suviše komplikovan.

U svim ovim situacijama možemo da indetifikujemo tri zajednička pojma [7]:

1. Postoji globalna veličina (cilj) koja se želi optimizovati (profit, razlika između predviđanja modela i eksperimentalnih podataka).

2. Uz globalni cilj obično su prisutni dodatni zahtevi ili ograničenja, koja moraju biti zadovoljena (ograničen rizik, resursi, kompleksnost modela).

3. Postoje određene veličine tako da ako se njihove vrednosti izaberu "dobro", zadovoljeni su i cilj i ograničenja. Te veličine se nazivaju optimizacione promenljive ili parametri.

Znači, da bi zadali problem matematičkog programiranja moramo:

1. odabrati jednu ili više optimizacionih promenljivih,

2. odabrati funkciju cilja i

3. formirati skup ograničenja.

Nakon toga, identifikuje se klasa problema kojoj dobijeni matematički model pripada i bira se metod za njegovo rešavanje.

Postoji više metoda za rešavanje problema linearnog programiranja [7, 10, 33, 73, 84]. Geometrijski metod je primenljiv na probleme kod kojih je broj promenljivih $n = 2$ ili kada je $n - m = 2$, gde je m broj ograničenja. U principu, zadatak linearnog programiranja se može rešiti geometrijski i u slučaju $n = 3$. Nedostatak geometrijskog metoda je u tome što ne rešava opšti zadatak linearnog programiranja već samo neke specijalne slučajeve. Do prvog opšteg metoda za rešavanje problema linearnog programiranja došao je američki matematičar Danzing 1947. godine [20]. Ovaj metod je poznat kao simpleks metod linearnog programiranja. On je takođe formulisao opšti oblik problema linearnog programiranja i dao algoritam za njegovo rešavanje, poznat kao ***simpleks metod***. Dantzig-ov rad je više godina kružio među stručnjacima i poslužio kao osnova svim narednim razmatranjima problema linearnog programiranja. Iako su u međuvremenu pronađeni i drugi metodi, ovaj metod se i danas koristi kroz brojne modifikacije (vidi radove [8, 25]). I geometrijski i simpleks metod traže maksimum (minimum) funkcije cilja na rubovima oblasti ograničenja.

U kasnijem periodu se pojavljuju i alternativni pristupi rešavanju problema linearnog programiranja. Pomenimo radove u kojima se koriste generilisani inverzi [63, 64, 65, 11], kao i radove Conna [14] i Daxa [22] koji ne koriste simpleks metod. Iako je praksa pokazala da je simpleks metod veoma efikasan, 1972. godine su V. Klee (Kli) i G.L. Minty (Minti) dokazali da on nije polinomijalan [38]. Oni su konstruisali jednostavan primer zadatka linearnog programiranja kod koga je dopustivi skup deformisana n-dimenzionalna kocka sa 2^n temena, za čije je rešavanje simpleks metodu sa standardnim izborom vodećeg elementa potrebno $2^n - 1$ iterativnih koraka. Prvi polinomijalni algoritam za rešavanje problema linearnog programiranja

je dao Hačijan u radu [37] 1979. godine. Time je napravljen veliki zaokret u razvoju linearnog programiranja. Hačijan je pokazao da njegov metod elipsoida rešava problem linearnog programiranja za $O(n^4L)$ elementarnih aritmetičkih operacija, gde je L broj bitova potrebnih za zapis svih parametara problema (matrice ograničenja, funkcije cilja i desne strane ograničenja. Hačijan je rešio vrlo važno teorijsko pitanje. Međutim, ispostavilo se da metod elipsoida nije primenljiv u praksi. Testiranja su pokazala da je simpleks metod daleko efikasniji, jer on "retko" dostiže gornju granicu složenosti, za razliku od metoda elipsoida, koji to često čini. Ovakav zaključak je inicirao da se pronađu novi metodi za rešavanje problema linearnog programiranja koji će osim polinomijalne složenosti imati i praktičnu primenljivost. Prvi takav metod predložio je 1984. godine N. Karmarkar u radu [36]. Karmarkarov algoritam je neke test primere rešavao i do 50 puta brže od simpleks metoda. Karmarkarov rezultat je izazvao pravu revoluciju u razvoju ove oblasti. Posle Karmarkarovog metoda pojavila se čitava familija metoda koji su poznati pod nazivom *unutrašnji metodi*. Danas su ***primal-dual*** metodi unutrašnje tačke dominantni za rešavanje problema linearnog programiranja [93]. Iako su otkriveni polinomijalni metodi, simpleks metod se i danas koristi i kroz brojne modifikacije još uvek živi. Simpleks metod je mnogo bolji od metoda unutrašnje tačke na tzv. loše uslovljenim problemima, zbog svoje spore, ali pouzdane konvergencije. Isto tako, u praksi se često javlja potreba za rešavanjem klase srodnih problema gde se optimalno rešenje jednog od njih može efikasno iskoristiti kao početna tačka za rešavanje ostalih problema. I u ovom slučaju je simpleks metod bolji izbor od metoda unutrašnje tačke.

Osim teorijskih rezultata pojavili su se i programski paketi za rešavanje problema linearnog programirana, i koji su bazirani na teorijskim rezultatima. Ovi programski paketi su našli široku primenu u praksi.

Unutrašnji metodi dele se na *primalne, dualne* i *primalno-dualne*. Primalni metodi rešavaju problem linearnog programiranja u standardnom obliku

$$\begin{aligned} \min \quad & c^T x, \\ \text{p.o.} \quad & Ax = b, \quad x \geq 0, \end{aligned}$$

gde je A matrica tipa $m \times n$, dok su a, b i c vektori odgovarajućih dimenzija. Dualni problemi rešavaju dualni problem oblika

$$\begin{aligned} \max \quad & b^T y, \\ \text{p.o.} \quad & A^T y + s = c, \quad s \geq 0. \end{aligned}$$

Primalno-dualni metodi rade simultano i sa primalnim i sa dualnim problemom (primal-dual metodi). Danas su primal-dual metodi dominantni. Ovi metodi su opisani u [93]. Izučavanje metoda unutrašnje tačke za rešavanje problema linearnog programiranja pripada savremenom i modernom trendu u svetu što potvrđuju već objavljene monografije o polinomijalnim metodima, kao što su [7, 40, 49, 81, 93]. Poslednjih desetak godina se objavljuje veliki broj radova iz ove oblasti tako da bibliografija broji više hiljada naslova.

Postoje tri alata u rešavanju problema linearnog programiranja.

1. Modeli: formulacija problema u detaljnim matematičkim terminima.
2. Algoritmi: tehnike za rešavanje modela.
3. Kompjuteri i softver: mašine za izvršavanje algoritamskih koraka.

Postoji veći broj programskih paketa koji su namenjeni rešavanju problema linearnog programiranja.

HOPDM je napisan u programskom jeziku FORTRAN [29].

LIPSOL je napisan u programskim jezicima MATLAB i FORTRAN. Deo koda koji se odnosi na **Sparse Cholesky** faktorizaciju i rešavanje linearnih sistema je napisan u FORTRAN-u, a ostatak koda u MATLAB-u. Algoritam je opisan u [94].

LOQO je napisan u programskom jeziku C, a opisan je u [83].

PCx je napisan u programskim jezicima C i FORTRAN. **Sparse Cholesky** kod je napisan u FORTRAN-u, a ostatak koda u C-u. Detaljan opis se može naći u [19].

MOSEK je napisan u programskom jeziku C++ i predstavlja jedan od najjačih solvera ne samo za probleme linearnog programiranja već uopšte i za probleme kvadratnog i nelinearnog programiranja. Za razliku od predhodno pomenutih programa, ovaj program je komercijalan.

LINDO - Linear Interactive and Discrete Optimizer - je interaktivni softverski paket koji se može koristiti za rešavanje problema linearnog programiranja. Razvijen je 1980. godine i od tada je prilagođen Windows okruženju i grafički orijentisanim programima. Softverski paket LINDO se koristi za rešavanje problema zadatih direktno sa tastature.

MarPlex je naš program. Napisan je u programskom jeziku Visual Basic 6.0 i koristi simpleks metod, odnosno modifikacije prezentovane u odeljku 3 ove glave. Izdvaja se od ostalih programa zbog svog jednostavnog interface-a kao i ne toliko obimnog i citljivog koda. Pošto je implementiran u Visual Basic-u, i koristi simpleks metod, zaostaje po pitanju brzine.

RevMarPlex je takođe naš program i predstavlja revidiranu verziju programa MarPlex napisanu u programskom jeziku MATHEMATICA.

Implementacija nekih metoda linearnog programiranja u programskom jeziku MATHEMATICA se može naći u [7].

1.1 Matematički model

Opšti oblik problema linearnog programiranja možemo izraziti na sledeći način. Odrediti vrednosti promenljivih $x_1, \ldots, x_k$ koje odgovaraju linearnim jednačinama

i nejednačinama

$$\begin{aligned} N_i^{(1)}: \quad & \sum_{j=1}^{k} a_{ij}x_j \;\leq\; b_i, \quad i \in I_1 \\ J_i: \quad & \sum_{j=1}^{k} a_{ij}x_j \;=\; b_i, \quad i \in I_2 \\ N_i^{(2)}: \quad & \sum_{j=1}^{k} a_{ij}x_j \;\geq\; b_i, \quad i \in I_3 \\ & x_j \;\geq\; 0, \quad j \in J \subseteq \{1,\ldots,k\}, \end{aligned} \tag{1.1.1}$$

gde je $I_1 \cup I_2 \cup I_3 = \{1,\ldots,m\}$, $I_1 \cap I_2 = \emptyset$, $I_1 \cap I_3 = \emptyset$, $I_2 \cap I_3 = \emptyset$, tako da linearna ciljna funkcija

$$z(x) = z(x_1,\ldots,x_k) = c_1x_1 + \cdots + c_kx_k \tag{1.1.2}$$

ima ekstremum, tj. minimum ili maksimum. Pri tome su a_{ij}, b_i, c_j poznati realni brojevi.

Po konvenciji, u opštem slučaju vektor $y \in \mathbb{R}^k$ predstavlja $k-$torku realnih brojeva $y = (y_1,\ldots,y_k)$, dok se u matričnim formulama podrazumeva da je y matrica tipa $k \times 1$ (vektor), tj.

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}, \quad y^T = [y_1 \ldots y_k].$$

Dalje, $y \geq 0$ označava $y_1 \geq 0, \ldots, y_k \geq 0$.

Rešenje $x = (x_1, x_2, \ldots, x_k)$ u primenama najčešće ima značenje plana ili programa (proizvodnje, prevoza), pa je otuda ovaj zadatak dobio naziv "programiranje", a naziv "linearno programiranje" označava da su ograničenja promenljivih (1.1.1), kao i funkcija cilja (1.1.2) linearni.

Proizvoljno rešenje sistema nejednačina (1.1.1) može da se zamisli i kao vektor $x = (x_1, x_2, \ldots, x_k)$, koji u geometrijskoj interpretaciji predstavlja tačku k-dimenzionalnog prostora $\mathbb{R}^k$. Svako nenegativno rešenje sistema nejednačina (1.1.1) nazivamo *dopustivim rešenjem* . Ako sa Ω_P označimo skup dopustivih rešenja, onda je $\Omega_P \subset R^k$. Za skup Ω_P pretpostavljamo da nije prazan i da sadrži najmanje jedan element (slučaj kada zadatak (1.1.1)-(1.1.2) ima rešenje). Optimalno rešenje $x^* = (x_1^*, \ldots, x_k^*) \in \Omega_P$ zadatka linearnog programiranja je ono dopustivo rešenje za koje funkcija cilja $z(x) = z(x_1,\ldots,x_k)$ dostiže maksimum (minimum). U slučaju maksimizacije funkcije cilja ispunjen je uslov

$$z(x^*) = \max_{x \in \Omega_P} z(x)$$

ili

$$z(x^*) \geq z(x) \quad \forall x \in \Omega_P.$$

Često se u praksi traži da optimalna vrednost funkcije cilja bude najmanja na skupu dopustivih rešenja Ω_P. U ovom slučaju optimalno rešenje $x^* \in \Omega_P$ je ono dopustivo rešenje za koje je ispunjeno

$$z(x^*) = \min_{x \in \Omega_P} z(x).$$

ili

$$z(x^*) \leq z(x) \quad \forall x \in \Omega_P.$$

U jednom konkretnom zadatku rešava se samo problem maksimizacije, odnosno problem minimizacije. Problem minimuma se može transformisati u problem maksimuma (i obratno) jednostavnim množenjem ciljne funkcije sa -1, koristeći sledeći rezultat.

Propozicija 1.1.1 *Dati kriterijum optimizacije može da se zameni suprotnim, pri čemu ta zamena ne utiče na optimalno rešenje, to jest ako je za $x^* \in \Omega_P$ ispunjeno*

$$z(x^*) = \max_{x \in \Omega_P} z(x)$$

onda je

$$-z(x^*) = \min_{x \in \Omega_P} [-z(x)]$$

i obrnuto.

Dokaz. Prema pretpostavci teoreme ispunjeno je

$$z(x^*) \geq z(x) \qquad \forall x \in \Omega_P.$$

Ako ovu nejednakost pomnožimo sa -1, dobija se

$$-z(x^*) \leq -z(x) \qquad \forall x \in \Omega_P,$$

čime je teorema dokazana. □

Zadatak linearnog programiranja ima rešenje ako veličina z_{max} (z_{min}) ima konačnu vrednost na skupu Ω_P dopustivih rešenja. Zadatak linearnog programiranja nema rešenja ako sistem nejednačina (1.1.1) nema nenegativnih rešenja ili ako veličina z_{max} (z_{min}) nema konačnu vrednost.

Ograničenja (1.1.1) određuju u k-dimenzionalnom prostoru konveksnu oblast Ω_P ograničenu skupom hiperravni

$$\sum_{j=1}^{k} a_{ij} x_j = b_i, \ i = 1, \ldots, m.$$

Nazovimo ovu oblast Ω_P-poliedrom, mada u izvesnim slučajevima ona može biti i beskonačna. Ta oblast se naziva *oblast dopustivih rešenja*. Kako su funkcije koje treba maksimizirati ili minimizirati linearne, klasični matematički metodi pokazuju da se maksimumi ili minimumi funkcije cilja $z(x) = z(x_1, \ldots, x_k)$ dostižu na

granicama oblasti Ω_P koja je određena datim ograničenjima. Ako hiperravan $z(x) = \text{const}$ nije paralelna ni sa jednom od pomenutih hiperravni, koje predstavljaju strane poliedra, onda će ciljna funkcija $z(x)$ dostići maksimum (minimum) u jednom od temena poliedra.

Neka je $A = [a_{ij}]_{m\times k}$ data matrica sa vrstama $V_1, \ldots, V_m$, neka su $b \in \mathbb{R}^m$ i $c \in \mathbb{R}^k$ dati vektori i neka je $x \in \mathbb{R}^k$ nepoznat vektor. U *matričnom obliku* problem (1.1.1)-(1.1.2) se može zapisati na sledeći način:

$$\begin{array}{lll} \min & c^T x, & \\ \text{p.o.} & V_i^T x \le b_i, & i \in I_1, \\ & V_i^T x = b_i, & i \in I_2, \\ & V_i^T x \ge b_i, & i \in I_3, \\ & x_j \ge 0, & j \in J \subseteq \{1, \ldots, k\}, \end{array} \tag{1.1.3}$$

gde je $I_1 \cup I_2 \cup I_3 = \{1, \ldots, m\}$, $I_1 \cap I_2 = \emptyset$, $I_1 \cap I_3 = \emptyset$, $I_2 \cap I_3 = \emptyset$.

Ukoliko je $I_2 = \{1, \ldots, m\}$ i $J = \{1, \ldots, k\}$, $A = [a_{ij}]_{m\times n}$, $c, x \in \mathbb{R}^n$, problem (1.1.3) se svodi na tzv. *standardni oblik* problema linearnog programiranja

$$\begin{array}{lll} \min & c^T x, & \\ \text{p.o.} & Ax = b, & \\ & x \ge 0. & \end{array} \tag{1.1.4}$$

U slučaju $I_3 = \{1, \ldots, m\}$ i $J = \{1, \ldots, k\}$ problem (1.1.3) se svodi na tzv. *simetrični oblik*

$$\begin{array}{lll} \min & c^T x, & \\ \text{p.o.} & Ax \ge b, & \\ & x \ge 0. & \end{array} \tag{1.1.5}$$

Bez narušavanja opštosti se može pretpostaviti da je $b_i \ge 0$ za svako $i = 1, \ldots, m$ (u suprotnom, odgovarajuća nejednačina se može pomnožiti sa -1).

Za vektor x koji zadovoljava uslove (1.1.1) kažemo da je *dopustivo rešenje* problema. Dopustivo rešenje x^* je *minimalno* ako je

$$c^T x^* \le c^T x$$

za svako dopustivo rešenje x.

Problem zadat u opštem obliku (1.1.1)-(1.1.2) je moguće transformisati u standardni ili simetrični oblik. Ako se uvedu nove nepoznate

$$\begin{aligned} x_{k+i} &= b_i - \sum_{j=1}^{k} a_{ij} x_j, \quad i \in I_1 \\ x_{k+i} &= \sum_{j=1}^{k} a_{ij} x_j - b_i, \quad i \in I_3. \end{aligned}$$

Tada uslovi (1.1.1) prelaze u sistem jednačina sa $k+q=n$ nepoznatih, gde je $q=|I_1|+|I_3|$.

$$\begin{aligned}\sum_{j=1}^{k} a_{ij}x_j + x_{k+i} &= b_i, \quad i \in I_1\\ \sum_{j=1}^{k} a_{ij}x_j &= b_i, \quad i \in I_2\\ \sum_{j=1}^{k} a_{ij}x_j - x_{k+i} &= b_i, \quad i \in I_3.\end{aligned} \tag{1.1.6}$$

Pri tome posmatramo funkciju

$$z = c_1x_1 + \cdots + c_kx_k + \cdots + c_nx_n \tag{1.1.7}$$

gde je $c_{k+1} = \ldots = c_n = 0$.

U sledećoj tabeli je ilustrovana veza između osnovnih oblika linearnog problema.

Ograničenje	Kanonska forma	Standardna forma
$a_i^T x \le b_i$	$-a_i^T x \ge -b_i$	$a_i^T x + s_i = b_i,\ s_i \ge 0$
$a_i^T x \ge b_i$		$a_i^T x - s_i = b_i,\ s_i \ge 0$
$a_i^T x = b_i$	$a_i^T x \ge b_i,\ -a_i^T x \ge -b_i$	

Promenljive s_i se nazivaju *slack promenljive* . Uvođenjem svake slack promenljive povećava se dimenzija problema n za 1, i matrica A se proširuje kolonom jedinične matrice I, dok se vektor ciljne funkcije c proširuje nultim elementom.

Promenljive iz skupa $\{1,\ldots,n\}\backslash J$, na koje nije nametnut uslov nenegativnosti, nazivaju se *slobodne promenljive* . Svaku slobodnu promenljivu $x_j,\ j \in \{1,\ldots,n\}\backslash J$ možemo zameniti u ciljnoj funkciji i svim ograničenjima izrazom $x_j^+ - x_j^-$, pri čemu je $x_j^+ \ge 0, x_j^- \ge 0$, i tako dolazimo do modela kod koga je na sve promenljive nametnut uslov nenegativnosti. Ovako postavljen problem je ekvivalentan problemu (1.1.1)-(1.1.2).

Primer 1.1.1 Razmotrimo problem linearnog programiranja

$$\begin{aligned}\min\quad & -5x_1 - 4x_2,\\ \text{p.o.}\quad & x_1 + x_2 \le 80,\\ & 3x_1 + x_2 \le 180,\\ & x_1 + 3x_2 \le 180,\\ & x_1 \ge 0,\ x_2 \ge 0.\end{aligned}$$

Standardni oblik ovog problema se dobija uvođenjem izravnavajućih promenljivih x_3, x_4 i x_5 :

$$\begin{aligned}\min\quad & -5x_1 - 4x_2,\\ \text{p.o.}\quad & x_1 + x_2 + x_3 = 80,\\ & 3x_1 + x_2 + x_4 = 180,\\ & x_1 + 3x_2 + x_5 = 180,\\ & x_i \ge 0,\ i = 1,\ldots,5.\end{aligned}$$

Da bi se problem u opštem obliku preveo na simetričan oblik potrebno je eliminisati ograničenja tipa jednačina. Jednačine $V_i^T x = b_i$, $i \in I_2$ se mogu ekvivalentno zameniti nejednačinama $V_i^T x \geq b_i$, $(-V_i)^T x \geq -b_i$, $i \in I_2$. Ograničenje oblika $(-V_i)^T x \geq -b_i$ dobijeno je množenjem ograničenja oblika $V_i^T x \leq b_i$ sa -1. Dalje, svaku slobodnu promenljivu $x_j, j \in \{1, \ldots, n\} \backslash J$ zamenimo u ciljnoj funkciji i svim ograničenjima kao ranije sa po dve nenegativne promenljive i time dobijamo problem tipa (1.1.5).

Napomena 1.1.1 *Razmotreno svođenje na simetričan oblik važi uz pretpostavku da je sistem jednačina $\{J_i\}_{i \in I_2}$ konzistentan i potpunog ranga. U suprotnom prvo bi trebalo eliminisati suvišne jednačine (na primer Gauss-ovim metodom eliminacije), odnosno formirati ekvivalentan sistem potpunog ranga.*

Iz ovih razmatranja sledi da se ne gubi na opštosti ako se posmatra linearni problem oblika (1.1.4) ili (1.1.5). U nastavku se uglavnom bavimo problemom linearnog programiranja u standardnom obliku jer je najpogodniji za teorijska razmatranja.

U vezi sa postavkom zadatka linearnog programiranja u standardnom obliku napomenimo sledeće. Sistem jednačina $Ax = b$ ima rešenje ako je rang matrice sistema jednačina

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}$$

jednak rangu proširene matrice $[A|b]$, koja se dobija dodavanjem kolone slobodnih članova $b_1, \ldots, b_m$ matrici A (teorema Kronecker-Capellija). Rang r matrice A naziva se rangom sistema i predstavlja broj linearno nezavisnih jednačina među datim ograničenjima. Ako taj uslov nije ispunjen, skup dopustivih rešenja Ω_P je prazan i problem nema rešenja.

Očigledno, rang sistema jednačina $Ax = b$ ne može biti veći od broja jednačina m, to jest uvek je ispunjeno $r \leq m$, ali broj r ne može biti veći ni od broja promenljivih n, pa je takođe ispunjeno $r \leq n$. Ako je $r < m$, tada primenom Gaussovog algoritma dolazimo do zaključka o nesaglasnosti sistema ili eliminišemo $m - r$ suvišnih jednačina u sistemu $Ax = b$. Zato pretpostavimo da je $r = m$, tj. da među ograničenjima (1.1.4) nema linearno zavisnih i da je sistem $Ax = b$ saglasan. U slučaju da je $r = m = n$, sistem ima jedinstveno rešenje $x = A^{-1}b$ pa ostaje samo da se proveri uslov $x \geq 0$. Nas će interesovati slučaj $r = m < n$, kada je broj linearno nezavisnih jednačina manji od broja promenljivih. Tada, ako je sistem jednačina $Ax = b$ saglasan, postoji beskonačno mnogo rešenja. Svako od tih rešenja dobija se tako što se za $n - r = s$ promenljivih izaberu proizvoljne vrednosti, a zatim se vrednosti preostalih r promenljivih izračunavaju iz sistema jednačina $Ax = b$. Promenljive veličine koje izračunavamo nazivamo *zavisnim* ili *bazičnim* (ima ih r), a promenljive veličine čije se vrednosti biraju proizvoljno nazivamo *nezavisnim* ili *slobodnim* promenljivim (njih ima $n - r = s$). Praktično se iz sistema jednačina r zavisnih promenljivih izražava pomoću $n - r = s$ nezavisnih promenljivih, pa se za nezavisne promenljive biraju proizvoljne vrednosti. Kako

se za nezavisne promenljive može birati beskonačno mnogo različitih vrednosti, to sistem jednačina $Ax = b$ ima beskonačno mnogo rešenja.

Definicija 1.1.1 *Nenegativno rešenje, koje se dobija tako što se za nezavisne promenljive izaberu vrednosti jednake nuli, naziva se bazično rešenje zadatka linearnog programiranja.*

Primer 1.1.2 Dat je zadatak linearnog programiranja u opštem obliku:

$$\begin{aligned} \max \quad & z(x) = 2x_1 + 5x_2 \\ \text{p.o.} \quad & x_1 + 4x_2 \leq 24 \\ & 2x_1 + x_2 \leq 21 \\ & x_1 + x_2 \leq 9 \\ & x_1 \geq 0,\ x_2 \geq 0. \end{aligned}$$

Transformisati ovaj zadatak na standardni oblik, naći broj zavisnih i nezavisnih promenljivih, izraziti zavisne promenljive pomoću nezavisnih i naći jedno bazično rešenje.

Levim stranama nejednačina dodajemo nenegativne promenljive x_3, x_4 i x_5 tako da one postanu jednačine:

$$\begin{aligned} x_1 + 4x_2 + x_3 &= 24 \\ 2x_1 + x_2 + x_4 &= 21 \\ x_1 + x_2 + x_5 &= 9. \end{aligned}$$

Sada se zadatak linearnog programiranja (u kanoničnom obliku sa maksimizacijom funkcije cilja) može formulisati na sledeći način: naći nenegativno rešenje $x = (x_1, x_2, x_3, x_4, x_5)$, $x_i \geq 0$, $i = 1, 2, 3, 4, 5$ koje zadovoljava poslednji sistem ograničenja i za koje ciljna funkcija $z(x) = 2x_1 + 5x_2 + 0 \cdot x_3 + 0 \cdot x_4 + 0 \cdot x_5$ dostiže maksimum.

Matrica sistema jedačina i proširena matrica jednake su (redom):

$$A = \begin{bmatrix} 1 & 4 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix} \qquad Ap = [A|b] = \begin{bmatrix} 1 & 4 & 1 & 0 & 0 & 24 \\ 2 & 1 & 0 & 1 & 0 & 21 \\ 1 & 1 & 0 & 0 & 1 & 9 \end{bmatrix}.$$

S obzirom da je rang matrice A jednak rangu matrice Ap ($r = r_A = r_{A_p} = 3$), imamo $r = 3$ zavisne promenljive i $k = n - r = 5 - 3 = 2$ nezavisne promenljive. Za nezavisne promenljive izaberimo x_1 i x_2, i pomoću njih izrazimo zavisne promenljive x_3, x_4, x_5:

$$\begin{aligned} x_3 &= 24 - x_1 - 4x_2 \\ x_4 &= 21 - 2x_1 - x_2 \\ x_5 &= 9 - x_1 - x_2 \end{aligned}$$

Bazično rešenje se sada dobija ako se stavi $x_1 = x_2 = 0$. Sledi, $x_3 = 24$, $x_4 = 21$ i $x_5 = 9$, to jest, dobili smo bazično rešenje $(0, 0, 24, 21, 9)$. Za ovo bazično rešenje ciljna funkcija ima vrednost $z = 0$. Naravno, možemo dobiti i druga bazična rešenja ako drugi par promenljivih odaberemo za nezavisne promenljive. Na primer, ako bismo za nezavisne promenljive odabrali x_2 i x_5, onda bi zavisne promenljive bile definisane na sledeći način:

$$\begin{aligned} x_1 &= 9 - x_2 - x_5 \\ x_3 &= 24 - (9 - x_2 - x_5) - 4x_2 = 15 - 3x_2 + x_5 \\ x_4 &= 21 - 2(9 - x_2 - x_5) - x_2 = 3 + x_2 + 2x_5. \end{aligned}$$

Za $x_2 = x_5 = 0$ dobijamo drugo bazično rešenje $(9, 0, 15, 3, 0)$. Ako funkciju cilja izrazimo pomoću x_2 i x_5:

$$z(x) = 2(9 - x_2 - x_5) + 5x_2 = 18 + 3x_2 - 2x_5$$

vidimo da ona dobija vrednost 18 za $x_2 = x_5 = 0$. Ova vrednost je veća od vrednosti funkcije cilja u prvom bazičnom rešenju, tj. bazično rešenje $(9, 0, 15, 3, 0)$ je bliže optimalnom rešenju od $(0, 0, 24, 21, 9)$. Maksimalna vrednost funkcije cilja iznosi $z_{max} = 33$. Ovu vrednost ciljna funkcija je dostigla u bazičnom rešenju $(4, 5, 0, 13, 0)$, tj. bazično rešenje $(4, 5, 0, 13, 0)$ je optimalno rešenje.

Primer 1.1.3 Fabrika proizvodi dve vrste artikala A_1 i A_2, i to na mašinama M_1 i M_2. Za artikal A_1 mašina M_1 radi 2^h, a mašina M_2 radi 4^h, i za vrstu A_2 mašina M_1 radi 4^h, a mašina M_2 radi 2^h. Fabrika dobija 3500 dinara po jedinici proizvoda A_1, a 4800 dinara po jedinici proizvoda A_2. Koliko treba proizvoditi artikala A_1 i A_2 i kako iskoristiti rad mašina M_1 i M_2 da dnevna dobit fabrike bude maksimalna?

Rešenje: Neka je x_1 broj proizvedenih artikala A_1, a x_2 broj proizvedenih artikala A_2 u toku dana. Tada je dnevna dobit fabrike:

$$z(x) = 3500x_1 + 4800x_2$$

uz uslove:

$$\begin{aligned} 2x_1 + 4x_2 &\leq 24 \\ 4x_1 + 2x_2 &\leq 24 \\ x_1 \geq 0, x_2 &\geq 0. \end{aligned}$$

Ovim smo dobili simetrični oblik. Ako sada uvedemo slack promenljive x_3 i x_4 dobijamo ekvivalentan problem u standardnom obliku:

$$\begin{aligned} \max \quad & 3500x_1 + 4800x_2 \\ \text{p.o.} \quad & 2x_1 + 4x_2 - 24 = -x_3 \\ & 4x_1 + 2x_2 - 24 = -x_4 \\ & x_1 \geq 0,\ x_2 \geq 0. \end{aligned}$$

odnosno u matričnoj formi (1.1.4):

$$A = \begin{bmatrix} 2 & 4 & 1 & 0 \\ 4 & 2 & 0 & 1 \end{bmatrix} \quad b = \begin{bmatrix} 24 \\ 24 \end{bmatrix} \quad c = \begin{bmatrix} 3500 \\ 4800 \end{bmatrix}$$

1.2 Osobine skupa ograničenja

Neka je problem linearnog programiranja zadat u standardnom obliku (1.1.4). Skup $\Omega_P = \{x | Ax = b, x \geq 0\}$ na kome je definisana funkcija z bitno utiče na ekstremne vrednosti i ima interesantne geometrijske osobine. U nastavku pretpostavljamo da je $r = m < n$, gde su m i n dimenzije matrice A, a r je njen rang. Tada sistem ima beskonačno mnogo rešenja pa ima smisla tražiti ekstremnu vrednost funkcije $z(x)$ definisane na skupu Ω_P. Označimo kolone matrice A sa $K_1, \ldots, K_n$.

Primetimo da su vektori $K_1, \ldots, K_n$ i b dimenzije m. Podsetimo da su vektori $x_1, \ldots, x_n$ linearno nezavisni ako iz jednačine $\alpha_1 x_1 + \cdots + \alpha_n x_n = 0$ sledi $\alpha_1 = \cdots = \alpha_n = 0$, a u suprotnom su linearno zavisni. Izraz oblika $a = \lambda a^1 + (1-\lambda)a^2$, $0 \leq \lambda \leq 1$ naziva se konveksna kombinacija vektora a^1 i a^2. Uopštena konveksna kombinacija vektora $a^1, \ldots, a^k$ je svaki vektor a oblika

$$a = \sum_{i=1}^{k} \lambda_i a^i, \quad \sum_{i=1}^{k} \lambda_i = 1, \quad (\lambda_1, \ldots, \lambda_k \geq 0.)$$

Skup vektora K je konveksan ako

$$(\forall a^1, a^2 \in K)(\lambda a^1 + (1-\lambda)a^2 \in K, \quad 0 \leq \lambda \leq 1).$$

Geometrijski, skup K je konveksan ako za svake dve tačke $a^1, a^2 \in K$ važi da je segment određen tim tačkama sadržan u K. Najmanji konveksan skup koji u

prostoru $\mathbb{R}^n$ sadrži $n+1$ različitih tačaka naziva se *simpleks* u $\mathbb{R}^n$. U prostoru $\mathbb{R}^2$ svaki trougao je simpleks, a u $\mathbb{R}^3$ svaki tetraedar je simpleks. Navodimo neka poznata tvrđenja i osnovne definicije koje su neophodne za izučavanje simpleks metoda kao i metoda unutrašnjih tačaka.

Teorema 1.2.1 *Skup* $\Omega_P = \{x | Ax = b, x \geq 0\}$ *je konveksan.*

Dokaz. Neka je $x^1, x^2 \in \Omega_P$. Tada za njihovu konveksnu kombinaciju

$$x = \lambda x^1 + (1-\lambda)x^2, \quad 0 \leq \lambda \leq 1$$

važi

$$Ax = \lambda Ax^1 + (1-\lambda)Ax^2 = \lambda b + b - \lambda b = b.$$

Kako je očigledno $x \geq 0$, to je $x \in \Omega_P$. □

Definicija 1.2.1 *Moguće rešenje* x *je osnovno (bazično) ako su u jednačini*

$$b = x_1 K_1 + \cdots + x_n K_n$$

vektori K_i *za koje je* $x_i > 0$, *linearno nezavisni.*

Primetimo da bazično rešenje može da ima najviše m koordinata većih od nule.

Definicija 1.2.2 *Bazično rešenje za koje je tačno m koordinata veće od nule naziva se nedegenerisano. Bazično rešenje za koje je manje od m koordinata veće od nule naziva se degenerisano.*

Primetimo da od n kolona matrice A tj. od vektora $K_1, \ldots, K_n$ možemo formirati $\binom{n}{m}$ podmatrica sa m kolona. Ako su odabrani vektori $K_{i_1}, \ldots, K_{i_m}$ linearno nezavisni, tada je matrica $B = [K_{i_1} \cdots K_{i_m}]$ regularna i postoji B^{-1}. Tada je vektor $x = B^{-1}b$ jedinstveno određen i ako je $x \geq 0$, onda je x osnovno rešenje. Napomenimo da je $x \in \mathbb{R}^n$ a $B^{-1}b \in \mathbb{R}^m$, i da $x = B^{-1}b$ po definiciji znači da su koordinate x_i za $i \notin \{i_1, \ldots, i_m\}$ jednake nuli.

Definicija 1.2.3 *Matrica* $A_B = [K_{i_1} \cdots K_{i_m}]$ *je osnovna (bazična) ako je regularna. Preostale kolone matrice A formiraju nebazičnu matricu* A_N. *Promenljive* $x_{i_1}, \ldots x_{i_m}$ *nazivamo bazičnim dok preostale promenljive nazivamo nebazičnim . Dve bazične matrice su susedne ako se razlikuju u jednoj koloni.*

Neka je A_B bazična matrica. Sada sistem iz (1.1.4) možemo napisati kao:

$$A_B x_B + A_N x_N = b \Longrightarrow x_B = A_B^{-1} b - A_B^{-1} A_N x_N$$

Ako sada stavimo $x_N = 0$ dobijamo jedno bazično rešenje ($x = (x_B, x_N) = (B^{-1}b, 0)$, posle odgovarajuće prenumeracije promenljivih). Obrnuto, svako bazično rešenje određuje odgovarajuću bazičnu matricu (ukoliko je nedegenerisano, onda je ta matrica jedinstvena a u suprotnom nije). Ovim smo opravdali naziv "bazična matrica" u Definiciji 1.2.3.

Iz prethodne definicije sledi da je maksimalan broj osnovnih rešenja jednak $\binom{n}{m}$.

Definicija 1.2.4 *Tačka x je ekstremna tačka konveksnog skupa Ω_P ako ispunjava uslove*

$$x = \lambda x^{(1)} + (1-\lambda)x^{(2)} \wedge \lambda \in [0,1] \Leftrightarrow x = x^{(1)} = x^{(2)}.$$

Definicija 1.2.5 *Bazično rešenje x sistema $Ax = b$ za koje važi i uslov $x \geq 0$ je bazično dopustivo rešenje.*

Primer 1.2.1 U primeru (1.1.1) je

$$K_1 = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}, \quad K_2 = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix}, \quad K_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad K_4 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad K_5 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Kako je $m = 3$ i $n = 5$, bazičnih matrica i bazičnih rešenja ima najviše $\binom{5}{3} = 10$. Odredimo bazične matrice, bazična rešenja i vrednosti ciljne funkcije u njima:

$$\begin{array}{lll} B_1 = [K_3\, K_4\, K_5], & x^1 = (0,0,80,180,180), & z^1 = 0, \\ B_2 = [K_1\, K_3\, K_5], & x^2 = (60,0,20,0,120), & z^2 = -300, \\ B_3 = [K_1\, K_1\, K_5], & x^3 = (50,30,0,0,40), & z^3 = -370, \\ B_4 = [K_1\, K_2\, K_4], & x^4 = (30,50,0,40,0), & z^4 = -350, \\ B_5 = [K_2\, K_3\, K_4], & x^5 = (0,60,20,120,0), & z^1 = -240. \end{array}$$

Dakle, postoji pet osnovnih rešenja. Za ostale baze rešenja nisu dopustiva:

$$\begin{array}{lll} B_6 = [K_1\, K_2\, K_3], & x^6 = (45,45,-10,0,0), & z^6 = -405, \\ B_7 = [K_1\, K_4\, K_5], & x^7 = (80,0,0,-60,100), & z^7 = -400, \\ B_8 = [K_1\, K_3\, K_4], & x^8 = (180,0,-100,-360,0), & z^8 = -900, \\ B_9 = [K_2\, K_3\, K_5], & x^9 = (0,180,-100,0,-360), & z^9 = -720, \\ B_{10} = [K_2\, K_4\, K_5], & x^{10} = (0,80,0,100,-60), & z^{10} = -320. \end{array}$$

Vrlo važna činjenica je da egzistencija osnovnog rešenja sledi iz postojanja dopustivog rešenja što sledi iz sledeće teoreme.

Teorema 1.2.2 *Ako je $\Omega_P = \{x | Ax = b, x \geq 0\} \neq \emptyset$, tada on sadrži bar jedno osnovno rešenje.*

Dokaz. Neka je $x = (x_1, \ldots, x_n) \in \Omega_P$. Ako je potrebno prenumerišimo kolone K_i i koordinate vektora x tako da je $x_i > 0$ za $i \leq p$ i $x_i = 0$ za $i > p$. Sada je

$$b = \sum_{i=1}^{n} x_i K_i = \sum_{i=1}^{p} x_i K_i.$$

Ako su vektori $K_1, \ldots, K_p$ linearno nezavisni, onda je $p \leq m$ i x je osnovno rešenje. Ako su vektori $K_1, \ldots, K_p$ linearno zavisni, tada postoje $\lambda_1, \ldots, \lambda_p \in \mathbb{R}$ tako da je

$$\sum_{i=1}^{p} \lambda_i K_i = 0$$

i postoji $\lambda_i \neq 0$, $1 \leq i \leq p$. Neka je $\lambda_k \neq 0$ i $\lambda_k > 0$. Tada je

$$K_k = -\sum_{j \neq k} \frac{\lambda_j}{\lambda_k} K_j \quad \text{i} \quad b = \sum_{j \neq k} \left(x_j - x_k \frac{\lambda_j}{\lambda_k} \right) K_j.$$

Dakle, vektor b je prikazan kao zbir $p-1$ kolona matrice A. Ako je pri tome još i $x_j - x_k\frac{\lambda_j}{\lambda_k} \geq 0$, $j \neq k$, tada je vektor

$$x^1 = (x_1 - x_k\frac{\lambda_1}{\lambda_k}, \ldots, x_{k-1} - x_k\frac{\lambda_{k-1}}{\lambda_k}, 0, x_{k+1} - x_k\frac{\lambda_{k+1}}{\lambda_k}, 0, \ldots, 0, x_p - x_k\frac{\lambda_p}{\lambda_k}, 0, \ldots, 0)$$

element skupa Ω_P (tj. rešenje sistema $Ax = b$). Ako je $\lambda_j \leq 0$, tada je očigledno i $x_j^1 \geq 0$. U suprotnom je $x_j^1 = x_j - x_k\frac{\lambda_j}{\lambda_k} \geq 0$, što je ekvivalentno sa $\frac{x_j}{\lambda_j} \geq \frac{x_k}{\lambda_k}$. Prema tome i u ovom slučaju važi $x_j^1 \geq 0$, pa zaključujemo da je $x_1 \in \Omega_P$. Međutim, x^1 ima najviše $p-1$ strogo pozitivnih koordinata, što je kontradikcija. Prema tome, vektori $K_1, \ldots, K_p$ su linearno nezavisni, i x je bazično dopustivo rešenje. □

Vezu između bazičnih rešenja i temena skupa Ω_P daje sledeća teorema.

Teorema 1.2.3 *Ako je $x \in \Omega_P$ bazično rešenje sistema $Ax = b, x \geq 0$, tada je x ekstremna tačka skupa Ω_P. Obratno, ako je x ekstremna tačka skupa Ω_P, tada je x bazično rešenje.*

Dokaz. Neka je x bazično rešenje i neka je novom numeracijom (ako je potrebno)

$$b = \sum_{j=1}^{m} x_j K_j, \quad x = (x_1, \ldots, x_m, 0, \ldots, 0).$$

Pretpostavimo da x nije ekstremna tačka skupa Ω_P, tj. postoje $x^1, x^2 \in \Omega_P$, $x^1 = (x_1^1, \ldots, x_n^1), x^2 = (x_1^2, \ldots, x_n^2)$, tako da je

$$x = \lambda x^1 + (1-\lambda)x^2, \quad 0 < \lambda < 1.$$

Kako je $x_i = \lambda x_i^1 + (1-\lambda)x_i^2$, $i = 1, \ldots, n$, to je $x_i^1 = x_i^2 = 0$ za $i > m$. Dakle, x^1 i x^2 su bazična rešenja za bazu $K_1, \ldots, K_m$, tj.

$$b = \sum_{j=1}^{m} x_j^1 K_j = \sum_{j=1}^{m} x_j^2 K_j = \sum_{j=1}^{m} x_j K_j.$$

Kako su $K_1, \ldots, K_m$ linearno nezavisni, to je $x = x^1 = x^2$ što znači da je x ekstremna tačka skupa Ω_P.

Dokažimo obratno tvrđenje. Neka je $x \in \Omega_P$ ekstremna tačka skupa Ω_P. Novom numeracijom (ako je potrebno) postižemo da je $x_i > 0$ za $i \leq p$ i $x_i = 0$ za $i \geq p$. Pretpostavimo da su odgovarajući vektori $K_1, \ldots, K_m$ iz $b = \sum_{j=1}^{p} x_j K_j$, linearno zavisni, tj. da postoji bar jedno $\lambda_j > 0$, $1 \leq j \leq p$, tako da važi

$$\sum_{j=1}^{p} \lambda_j K_j = 0.$$

Sada je

$$b = \sum_{j=1}^{p} x_j K_j \pm \sum_{j=1}^{p} \mu\lambda_j K_j = \sum_{j=1}^{p} (x_j \pm \mu\lambda_j) K_j$$

za svako $\mu \in \mathbb{R}$. Ako je $\mu = \frac{1}{2}\min\left\{\frac{x_j}{\lambda_j} | \, 1 \leq j \leq p\right\}$, tada je $x_j \pm \mu\lambda_j > 0$, $1 \leq j \leq p$, pa je

$$\begin{aligned} x^1 &= (x_1 + \mu\lambda_1, \ldots, x_p + \mu\lambda_p, 0, \ldots, 0) \in \Omega_P, \\ x^2 &= (x_1 - \mu\lambda_1, \ldots, x_p - \mu\lambda_p, 0, \ldots, 0) \in \Omega_P, \end{aligned}$$

i važi

$$x = \frac{1}{2}x^1 + \frac{1}{2}x^2,$$

što je nemoguće jer je x ekstremna tačka skupa Ω_P. □

Iz prethodnog sledi da je broj ekstremnih tačaka manji od $\binom{n}{m}$.

Posledica 1.2.1 *Skup Ω_P ima konačno mnogo ekstremnih tačaka.*

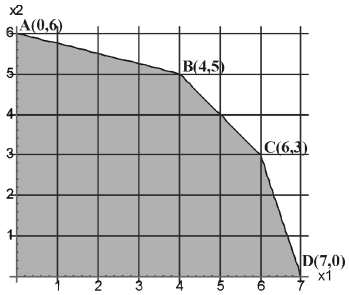
Dokaz. Svaka bazična matrica B određuje tačno jedno bazično rešenje (na osnovu Teoreme 1.2.3), i za svako bazično rešenje može naći odgovarajuća matrica B tako da važi Teorema 1.2.3. Zaključujemo da bazičnih matrica ima ne manje nego bazičnih rešenja, a na osnovu prethodne teoreme ekstemnih tačaka skupa Ω_P. Očigledno, bazičnih matrica ima ne više od $\binom{n}{m}$. □

Značaj prethodne teoreme i posledice je u tome što se broj potencijalnih ekstremuma funkcije $z(x)$ redukuje sa (u opštem slučaju) beskonačnog skupa Ω_P na konačan skup temena koji ima maksimalno $\binom{n}{m}$ elemenata. Ciljna funkcija $z(x)$ dostiže maksimum ili minimum u temenima konveksnog skupa Ω_p. Formalno, dovoljno je izračunati vrednosti ciljne funkcije u svim ekstremnim tačkama skupa Ω_P i odrediti onu tačku, ili tačke, u kojima je vrednost funkcije $z(x)$ ekstremna. Ovaj broj potencijalnih ekstremuma veoma brzo raste sa povećanjem vrednosti m i n.

Primer 1.2.2 Posmatramo sledeći problem

$$\begin{aligned} \max \quad & z(x) = 2x_1 + 5x_2 \\ \text{p.o.} \quad & x_1 + 4x_2 \leq 24, \; 3x_1 + x_2 \leq 21, \; x_1 + x_2 \leq 9. \end{aligned}$$

Oblast Ω_p sa ekstremnim tačkama A, B, C, D prikazana je na sledećoj slici.



Slika 1.2.1. Oblast dopustivih rešenja zadatog problema nacrtana u programu MATHEMATICA.

Posle izračunavanja vrednosti $z(A), z(B), z(C), z(D)$ možese uočiti da najveću vrednost ima $z(B) = z(4,5) = 33$, što i predstavlja rešenje problema.

Dokazaćemo kasnije u Teoremi 2.1.1 da se ekstemum funkcije cilja $z(x)$ (ako postoji) nalazi upravo u ekstremnim tačkama skupa Ω_P, a na osnovu Teoreme 1.2.3 u bazično dopustivim rešenjima sistema $Ax = b$.

Sledeća teorema daje kriterijum za detekciju neograničenosti skupa Ω_P.

Teorema 1.2.4 *Ako postoji bazična matrica $A_B = [K_{i_1} \cdots K_{i_m}]$ i kolona K_p matrice A tako da je $A_B^{-1}K_p \leq 0$, tada je Ω_P neograničen skup.*

Dokaz. Neka je $A_B = [K_1 \dots K_m]$ i $B^{-1}K_p \leq 0$ za neko p. Tada je

$$K_p = \lambda_1 K_1 + \cdots + \lambda_m K_m$$

i važi $\lambda_j \leq 0$, $1 \leq j \leq m$. Sada iz

$$b = \sum_{j=1}^{m} x_j K_j - \mu K_p + \mu K_p, \quad \mu > 0,$$

sledi

$$b = \sum_{j=1}^{m} (x_j - \mu\lambda_j) K_j + \mu K_p, \quad \mu > 0,$$

odakle je

$$x^\mu = (x_1 - \mu\lambda_1, \dots, x_m - \mu\lambda_m, 0, \dots, 0, \overset{(p)}{\rightarrow} \mu, 0, \dots, 0) \in \Omega_P$$

za svako $\mu > 0$, tj. Ω_P je neograničen skup. □

Ako je za svaku osnovnu matricu B i svaki vektor K_p vektor $B^{-1}K_p$ nenegativan, tada je skup Ω_P ograničen i svako $x \in \Omega_P$ je konveksna kombinacija osnovnih (bazičnih) rešenja. Znači, dovoljno je znati samo bazična rešenja. Ako je jedno bazično rešenje poznato, možemo odrediti drugo bazično rešenje. Neka je $B = [K_1 \cdots K_m]$ osnovna matrica i neka je $x = B^{-1}b = (x_1, \dots, x_m, 0, \dots, 0)$ poznato osnovno rešenje. Ako za neko K_r važi $B^{-1}K_r \leq 0$, skup Ω_P je neograničen pa zato pretpostavimo da je Ω_P ograničen, tj. $B^{-1}K_r$ nenegativno. Neka je

$$B^{-1}K_r = (\lambda_1, \dots, \lambda_m, 0, \dots, 0)$$

nenegativno i neka je $\lambda_k > 0$ za neko $k, 1 \leq k \leq m$. Tada iz

$$K_r = \lambda_1 K_1 + \cdots + \lambda_m K_m$$

sledi

$$K_k = \frac{1}{\lambda_k} K_r - \sum_{j \neq k} \frac{\lambda_j}{\lambda_k} K_j. \tag{1.2.1}$$

Zamenom (1.2.1) u

$$b = x_1 K_1 + \cdots + x_k K_k + \cdots + x_m K_m$$

dobijamo

$$b = x_1K_1 + \cdots + \frac{x_k}{\lambda_k}K_r - \sum_{j\neq k} x_k\frac{\lambda_j}{\lambda_k}K_j + \cdots + x_mK_m$$

to jest

$$b = \sum_{j\neq k}(x_j - x_k\frac{\lambda_j}{\lambda_k})K_j + \frac{x_k}{\lambda_k}K_r.$$

Odgovarajuće rešenje

$$x^1 = (x_1 - x_k\frac{\lambda_1}{\lambda_k}, \ldots, \overset{(k)}{\rightarrow} 0, \ldots, x_m - x_k\frac{\lambda_m}{\lambda_k}, 0, \ldots, 0, \overset{(r)}{\rightarrow} \frac{x_k}{\lambda_k}, 0, \ldots, 0)$$

je u skupu Ω_P ako je $x_j - x_k\frac{\lambda_j}{\lambda_k} \geq 0, j \neq k$, što je ispunjeno ako k izaberemo tako da je

$$\frac{x_k}{\lambda_k} = \min_j \left\{ \frac{x_j}{\lambda_j} |\ \lambda_j > 0 \right\}.$$

Bazično rešenje x^1 je različito od x ako je $x_k \neq 0$ što je ispunjeno ako je x nedegenerisano rešenje. Odgovarajuća osnovna matrica dobija se tako što kolonu K_k zamenimo kolonom K_r.

1.3 Geometrijski metod

Svakom uslovu nenegativnosti odgovara u vektorskom prostoru $\mathbb{R}^n$ poluprostor u kome je odgovarajuća promenljiva nenegativna. Svakoj uslovnoj jednačini u $\mathbb{R}^n$ odgovara jedna hiperravan. Svakoj uslovnoj nejednačini odgovara poluprostor omeđen hiperravni pridruženoj odgovarajućoj jednačini. Skup svih dopustivih vektora x je presek svih datih poluprostora i datih hiperravni, dakle čini jedan konveksni poliedar Ω_P. Jednačina $c^Tx = k$ za neko k predstavlja hiperravan paralelnu sa prostorom $\mathbb{R}^{n-1}$ koji je normalan na c. Projekcija poliedra Ω_P na pravac određen vektorom c je zatvoren skup $[l, L]$ realnih brojeva, gde je l minimum a L maksimum ciljne funkcije (1.1.2). Odgovarajuće hiperravni normalne na c su dodirne hiperravni poliedra Ω_P. Zajedničke tačke tih dodirnih hiperravni sa poliedrom Ω_P daju vrednosti u kojima funkcija (1.1.2) dostiže ekstremnu vrednost.

Geometrijski metod se može iskoristiti kod problema koji sadrže $n = 2$ promenljive, a najviše $n = 3$ promenljive. Zadatak linearnog programiranja dat u osnovnom obliku koji ispunjava uslov $n - m = 2$ (a najviše $n - m = 3$) takođe se može rešavati geometrijskim metodom. Geometrijski metod, iako ne baš pristupačan, koristi se zato što olakšva pristup opštoj algebarskoj metodi.

Neka je dat linearni problem u obliku

$$\begin{aligned} \max\ & z(x) = c_1x_1 + \ldots + c_nx_n \\ \text{p.o.}\ & a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leq b_1 \\ & \cdots\cdots\cdots \\ & a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leq b_m. \end{aligned}$$

Za dati sistem znamo da je svako rešenje sistema nejednačina jedna tačka prostora $\mathbb{R}^n$, a skup nenegativnih dopustivih rešenja Ω_p je podskup prostora $\mathbb{R}^n$. Svaka od nejednačina

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i,\ i = 1, 2, ..., m$$

određuje podskup $D_i \subset \mathbb{R}^n$, $i = 1, \ldots, m$ koji predstavlja skup tačaka s jedne strane hiperravni

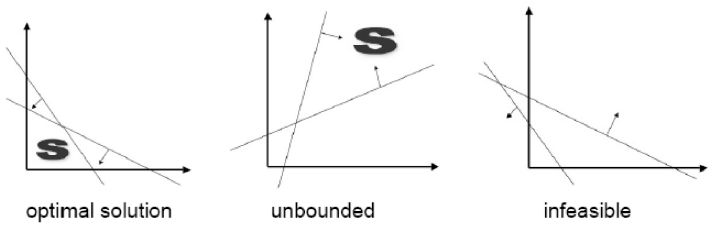
$$\sum_{j=1}^{n} a_{ij}x_j = b_i,$$

pa je oblast dopustivih rešenja (poliedar u $\mathbb{R}^n$) određena presekom skupova

$$\Omega_p = D_1 \cap D_2 \cap \cdots \cap D_m \cap D_{m+1} \cap \cdots \cap D_{m+n},$$

gde se podskupovi $D_{m+1}, ..., D_{m+n}$ dobijaju iz uslova nenegativnosti promenjivih $x_1 \geq 0, \ldots, x_n \geq 0$. Skup dopustivih rešenja geometrijski predstavlja poliedar (simplicijalni kompleks).

Različite mogućnosti pri rešavanju problema linearnog programiranja prikazane su na slici 1.3.1.



Slika 1.3.1. Različite mogućnosti pri rešavanju problema linearnog programiranja.

Skup tačaka u kojima funkcija cilja $z(x)$ ima vrednost d predstavlja takođe jednu hiperravan $\mathcal{H}_{z,d} = \{x \in \mathbb{R}^n \mid z(x) = d\}$, koju u zavisnosti od vrednosti d možemo translirati u pravcu vektora c. Sada se problem linearnog programiranja svodi na nalaženje maksimalne (minimalne) vrednosti za d tako da je $\mathcal{H}_{z,d} \cap \Omega_P \neq \emptyset$. Upravo na ovoj činjenici se zasniva geometrijski metod. Sada je jasno zašto je ovaj metod primenljiv samo u slučajevima $n = 2$ i $n = 3$. Primetimo još da se, na osnovu Teoreme 2.1.1, ekstremum funkcije cilja dostiže u ekstremnoj tački skupa Ω_P. Skup optimalnih tačaka Ω_P^* iz iste teoreme je konveksan i predstavlja k-dimenzionalni poliedar. Ciljna funkcija, koja se za $z(x) = d$ interpretira kao hiperravan, dostiže maksimum (minimum) u jednom temenu poliedra (slučaj jedinstvenog optimalnog rešenja) ili po jednoj strani poliedra ako je hiperravan $z(x) = d$ njoj paralelana (slučaj beskonačno mnogo optimalnih rešenja).

Geometrijski metod ilustrujemo na sledećem primeru, a ujedno izvlačimo zaključke koji će biti od značaja za opšti algebarski metod.

Primer 1.3.1 Fabrika proizvodi dve vrste artikala A_1 i A_2, i to na mašinama M_1 i M_2. Za artikal A_1 mašina M_1 radi 2^h, a mašina M_2 radi 4^h, dok za vrstu A_2 mašina M_1 radi 4^h, a mašina M_2

radi 2^h. Fabrika dobija 3500 dinara po jedinici proizvoda A_1, a 4800 dinara po jedinici proizvoda A_2. Koliko treba proizvoditi artikala A_1 i A_2 i kako iskoristiti rad mašina M_1 i M_2 da dnevna dobit fabrike bude maksimalna?

Rešenje: Neka je x_1 broj proizvedenih artikala A_1, a x_2 broj proizvedenih artikala A_2 u toku dana. Tada je dnevna dobit fabrike:

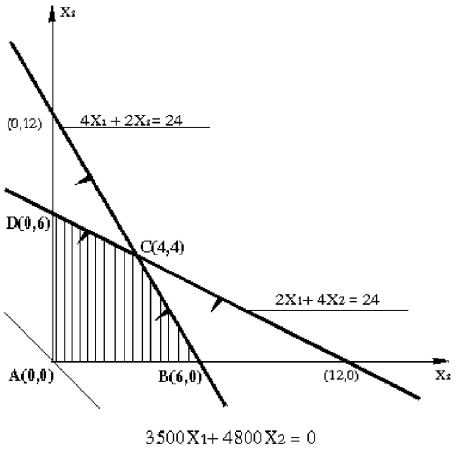
$$z(x) = 3500x_1 + 4800x_2$$

uz uslove:

$$\begin{aligned} 2x_1 + 4x_2 &\leq 24 \\ 4x_1 + 2x_2 &\leq 24 \\ x_1 \geq 0, x_2 &\geq 0. \end{aligned}$$

Ovaj model se sastoji od funkcije $z(x)$ i ograničenja (sistema nejednačina ili jednačina), i naziva se matematički model. Dnevna dobit fabrike je funkcija $z(x)$ koja zavisi od dve promenljive x_1 i x_2 i taj oblik zavisnosti je linearan. Zaista, vrednost jedinice A_1 na tržištu, tj. 3500 se množi sa brojem proizvedenih jedinica x_1 te vrste proizvoda, i tome se dodaje vrednost jedinice proizvoda A_2 na tržištu, tj. 4800 pomnožene sa brojem x_2 proizvedenih jediničnih artikala A_2. Ograničenja proizilaze iz činjenice da M_1 proizvodi jedinicu proizvoda A_1 za 2^h, a M_2 tu istu jedinicu za 4^h, dok dok se proizvod A_2 proizvodi na mašini M_1 za 4^h, a na M_2 za 2^h, kao i da mašine u idealnom stepenu iskorišćenja rade najviše 24^h. Otuda slede linearne nejednačine u navedenom matematičkom modelu, gde se prva odnosi na mogućnosti mašine M_1, a druga na mogućnosti mašine M_2. Ovim nejednakostima dodaje se priroda traženih promenjivih x_1 i x_2, koja se sastoji u tome da budu nenegativne veličine. Skup rešenja Ω_p je konveksan skup, jer je presek konačnog broja konveksnih skupova. Funkcija $z(x)$ dostiže maksimu ili minimum u temenima konveksnog skupa Ω_p.

Ako se ograničenja grafički predstave u koordinatnom sistemu x_1x_2 dobija se jedan četvorougao $ABCD$ koji je konveksan, u čijoj oblasti se nalaze moguća rešenja datog linearnog modela. Unutar četvorougla, kao i na tačkama ruba, (duž AB, BC, CD i DA) nalaze se moguća rešenja datog modela. Međutim, uočava se da je teme $C(4,4)$ sa koordinatama $x_1 = 4$ i $x_2 = 4$ takvo da odgovara optimalnom rešenju datog modela, jer je najudaljenije od prave $3500x_1 + 4800x_2 = 0$ dobijene iz funkcije cilja $z(x)$ za $z(x) = 0$. Zaista, odgovarajuća vrednost funkcije kriterijuma je max $z(x) = 3500 * 4 + 4800 * 4 = 14000 + 19200 = 33200$ dinara dnevne dobiti, koja je najveća moguća u datim uslovima.



Slika 1.3.2. Grafički prikaz ograničenja.

Primer 1.3.2 Dnevna količina stočne hrane treba da sadrži najmanje $0.4kg$ komponente A, $0.5kg$ komponente B, $2kg$ komponente C i $1.8kg$ komponente D. Na tržištu se mogu naći dve vrste

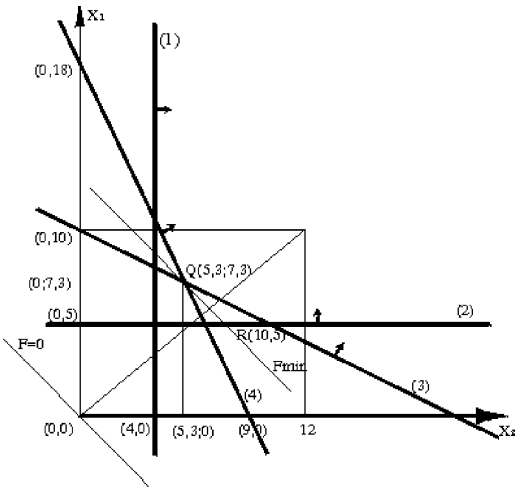
mešavina stočne hrane M_1 i M_2, pri čemu ove mešavine sadrže komponente A, B, C i D u sledećim količinama:

komponente	M_1	M_2	propisane količine za komponente
A	0.1	0	0.4
B	0	0.1	0.5
C	0.1	0.2	2.0
D	0.2	0.1	1.8
cena mešavina po kg	60	50	-

Koliku količinu svake od mešavina M_1 i M_2 treba kupiti da dnevni troškovi budu minimalni?

Rešenje: Potrebno je da se reši sledeći linearni program:

$$\begin{aligned} \min \quad & z(x) = 60x_1 + 50x_2 \\ \text{p.o.} \quad & 0.1x_1 \geq 0.4 \\ & 0.1x_2 \geq 0.5 \\ & 0.1x_1 + 0.2x_2 \geq 2.0 \\ & 0.2x_1 + 0.1x_2 \geq 1.8 \\ & x_1 \geq 0,\ x_2 \geq 0. \end{aligned}$$



Slika 1.3.3. Grafički prikaz ograničenja.

Optimalno rešenje problema je tačka $Q(5.3, 7.3)$.

Primer 1.3.3 Dva rudnika R_1 i R_2 snabdevaju ugljem tri grada A, B i C. Rudnik R_1 može dnevno da isporučuje 500 tona uglja, a rudnik R_2 800 tona. Troškovi prevoza jedne tone uglja u novčanim jedinicama od rudnika do gradova prikazani su u tabeli. Kako organizovati prevoz da ukupni troškovi prevoza budu najmanji?

Rudnici	A	B	C
R_1	8	5	5
R_2	4	6	8

Rešenje: Označimo sa x_1 broj tona uglja koji treba prevesti iz rudnika R_1 u grad A, a sa x_2 broj tona uglja iz R_1 u B. Uzimajući u obzir da se u gradovima potražuje dnevno isto onoliko

uglja koliko se dnevno može dobiti iz rudnika, možemo ostale količine uglja da izrazimo pomoću x_1 i x_2:

$$\begin{aligned} R_1 &\rightarrow C : 500 - x_1 - x_2 \\ R_2 &\rightarrow A : 500 - x_1 \\ R_2 &\rightarrow B : 400 - x_2 \\ R_2 &\rightarrow C : 800 - 500 + x_1 - 400 + x_2 = x_1 + x_2 - 100. \end{aligned}$$

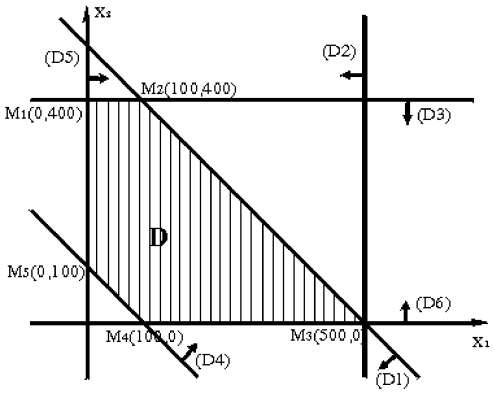
Ograničenja i uslovi nenegativnosti glase:

$$\begin{aligned} &(D_1) \quad 500 - x_1 - x_2 \geq 0 \ \Rightarrow \ x_1 + x_2 \leq 500 \\ &(D_2) \quad 500 - x_1 \geq 0 \ \Rightarrow \ x_1 \leq 500 \\ &(D_3) \quad 400 - x_2 \geq 0 \ \Rightarrow \ x_2 \leq 400 \\ &(D_4) \quad x_1 + x_2 - 100 \geq 0 \ \Rightarrow \ x_1 + x_2 \geq 100 \\ &(D_5) \quad x_1 \geq 0 \\ &(D_6) \quad x_2 \geq 0. \end{aligned}$$

Oblasti $D_1, D_2, \ldots, D_6$ su označene strelicama na datoj slici, a oblast dopustivih rešenja

$$D = \bigcap_{i=1}^{6} D_i$$

šrafirana je na slici.



Slika 1.3.4. Grafički prikaz ograničenja.

Ukupni troškovi prevoza jednaki su:

$$\begin{aligned} z =& 8x_1 + 5x_2 + 5(500 - x_1 - x_2) + 4(500 - x_1) + 6(400 - x_2) + 8(x_1 + x_2 - 100) \\ =& 7x_1 + 2x_2 + 6500. \end{aligned}$$

Veličine x_1 i x_2 treba sada odrediti tako da ukupni troškovi budu minimalni. Prava $7x_1 + 2x_2 + 6500 = \text{const}$ uzima najmanju vrednost u tačkama oblasti dopustivih vrednosti (D), koje su najbliže koordinatnom početku. Najjednostavnije je zbog toga izračunati vrednost funkcije cilja u tačkama oblasti dopustivih vrednosti (D), koje su najbliže koordinatnom početku. Najjednostavnije je zbog toga izračunati vrednost funkcije cilja u tačkama $M_4(100, 0)$ i $M_5(0, 100)$:

$$\begin{aligned} z(M_4) &= 7 * 100 + 2 * 0 + 6500 = 7200 \\ z(M_5) &= 7 * 0 + 2 * 100 + 6500 = 6700 = z_{min}. \end{aligned}$$

Na taj način minimalni troškovi prevoza od rudnika R_1 i R_2 do gradova A, B i C obezbeđeni su ako je plan prevoza sledeći $(x_1 = 0, x_2 = 100)$:

Rudnici	A	B	C
R_1 (500)	0	100	400
R_2 (800)	500	300	0

U ovoj tabeli je, dakle prikazan optimalan plan prevoza uglja. Ovakav zadatak je poznat pod nazivom transportni zadatak. Kao što vidimo, u slučaju kada imamo dve otpremne stanice (rudnici) i tri prijemne stanice (gradovi), transportni zadatak može da se reši geometrijski, jer se broj nepoznatih svodi na dve zahvaljujući uslovu da je količina ponuđenog uglja od strane rudnika R_1 i R_2 jednaka količini traženog uglja u gradovima A, B i C.

Primer 1.3.4 Rešiti problem linearnog programiranja grafičkim metodom

$$\begin{aligned} \max \quad & z(x) = 2x_1 + 3x_2 \\ \text{p.o.} \quad & x_1 + x_2 \leq 5 \\ & x_1 - 3x_2 \geq 0 \\ & 2x_1 + 3x_2 \geq 6 \\ & x_i \geq 0, \quad i = 1, 2. \end{aligned}$$

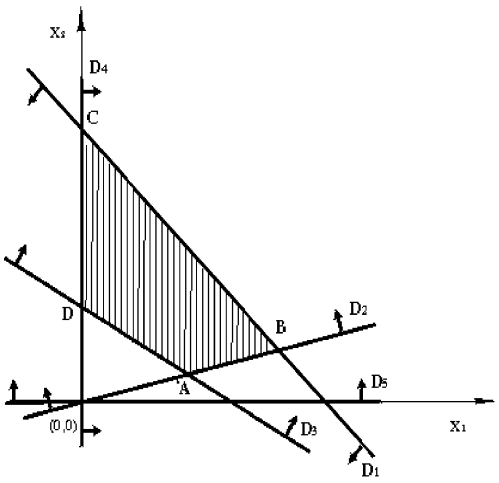
Označimo ograničenja i uslove negativnosti respektivno sa D_1, D_2, D_3, D_4, D_5:

$$\begin{aligned} (D_1) \quad & x_1 + x_2 \leq 5 \\ (D_2) \quad & x_1 - 3x_2 \geq 0 \\ (D_3) \quad & 2x_1 + 3x_2 \geq 6 \\ (D_4) \quad & x_1 \geq 0 \\ (D_5) \quad & x_2 \geq 0. \end{aligned}$$

Svako od ograničenja i uslova nenegativnosti predstavlja poluravan u prostoru $\mathbb{R}^2$. Jasno sva dopustiva rešenja će se naći u podskupu

$$D = D_1 \cap D_2 \cap D_3 \cap D_4 \cap D_5.$$

Prikažimo geometrijski oblast dopustivih rešenja određenu ravnima $D_i (i = 1, 2, ..., 6)$:



Slika 1.3.5. Grafički prikaz ograničenja.

Šrafirana površ predstavlja oblast dopustivih rešenja. Optimalno rešenje će biti u nekoj od tačaka A, B, C, D. Da bismo utvrdili koja tačka predstavlja optimalno rešenje, treba najpre nacrtati pravu

koja predstavlja funkciju cilja i to za $z = 0$. Ovako nacrtanu pravu treba paralelno pomerati u pravcu rasta nepoznatih X_1 i X_2, sve dok ne stignemo do poslednje tačke šrafirane površi, odnosno do poslednje tačke poligona dopustivih rešenja. U toj tački će ciljna funkcija imati maksimalnu vrednost tj. ta tačka će predstavljati optimalno rešenje. U našem primeru to će očigledno biti tačka B čije koordinate treba izračunati i uvrstiti u funkciju cilja. Iz sistema

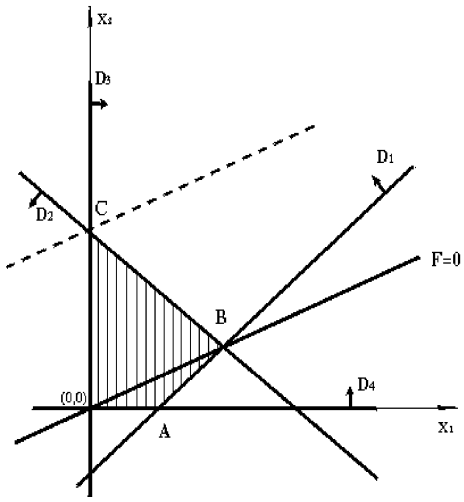
$$x_1 + x_2 = 5$$
$$x_1 - 3x_2 = 0$$

dobijamo $B(15/4, 5/4)$ i

$$z_{max} = z(B) = z(15/4, 5/4) = 2 * 15/4 + 3 * 5/4 = 45/4.$$

Primer 1.3.5 Naći minimum funkcije $z(x) = x_1 - 2x_2$ uz ograničenja

$$(D_1) \quad x_1 - x_2 \leq 1$$
$$(D_2) \quad x_1 + x_2 \leq 3$$
$$(D_3) \quad x_1 \geq 0$$
$$(D_4) \quad x_2 \geq 0$$



Slika 1.3.5. $z_{min} = z(C) = z(0, 3) = -6$.

Pravac pomeranja je u pravcu rašćenja x_2 jer se njenim povećanjem smanjuje vrednost funkcije cilja.

Sada možemo izvesti sledeće zaključke u vezi geometrijskog metoda:

I Ako je oblast dopustivih rešenja D konačna onda na njenoj granici postoji najmanje jedna tačka koja predstavlja optimalno rešenje.

Dokaz sledi iz teoreme Weierstrassa koja govori da neprekidna funkcija definisana na zatvorenom i ograničenom skupu dostiže bar u jednoj tački tog skupa najveću vrednost, i bar u jednoj tački najmanju vrednost. Kako je ciljna funkcija linearna i neprekidna to ova teorema važi i kad je u pitanju konveksna i ograničena oblast.

II U zadatku linearnog programiranja ne postoji optimalno rešenje kada je skup dopustivih rešenja D prazan ili kada je D neograničen i istovremeno na njemu ciljna funkcija, koja se minimizira (maksimizira), beskonačno opada (raste).

III Ako su dobijena dva optimalna rešenja onda su sve tačke duži između tih tačaka, optimalna rešenja. Dakle, ako su $x^{(1)} \in D$ i $x^{(2)} \in D$ optimalna rešenja, onda je svaki vektor oblika

$$x_\lambda^{(0)} = \lambda x^{(2)} + (1-\lambda)x^{(1)}, \quad 0 \leq \lambda \leq 1$$

optimalno rešenje.

U slučaju $n = 2$, ako se ograničenja grafički predstave u koordinatnom sistemu x_1x_2 dobija se konveksan poligon, na čijim se temenima (ekstremnim tačkama) nalaze moguća rešenja datog problema linearnog programiranja. Teme najudaljenije od prave određene funkcijom cilja predstavlja optimalno rešenje datog problema.

Geometrijski metod, za slučaj $n = 2$ smo implementirali u programskom jeziku MATHEMATICA [90]. Tako je nastao program GEOM [80], koji za unet problem linearnog programiranja u simetričnom obliku od dve promenljive pronalazi optimalno rešenje geometrijskim metodom i pri tome grafički prikazuje sve međukorake. Za grafičko prikazivanje skupa Ω_P dopustivih rešenja, koristili smo sledeće standardne funkcije programskog jezika MATHEMATICA: `InequalityPlot`, `InequalitySolve`, `FindInstance`, itd. Ove funkcije se nalaze u standardim paketima `Graphics‘InequalityGraphics` i `Algebra’InequalitySolve`. Kompletan kôd programa GEOM kao i detalji implementacije prikazani su u dodatku, a mogu se naći i u našem radu [80]. Razmotrimo sada rad programa GEOM na sledećem primeru:

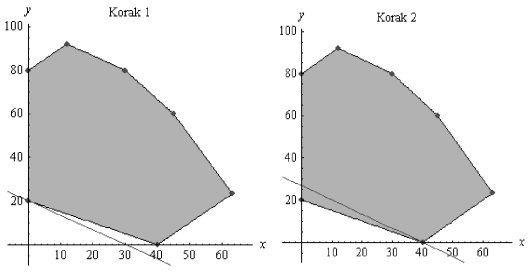
Primer 1.3.6 Rešiti problem linearnog programiranja:

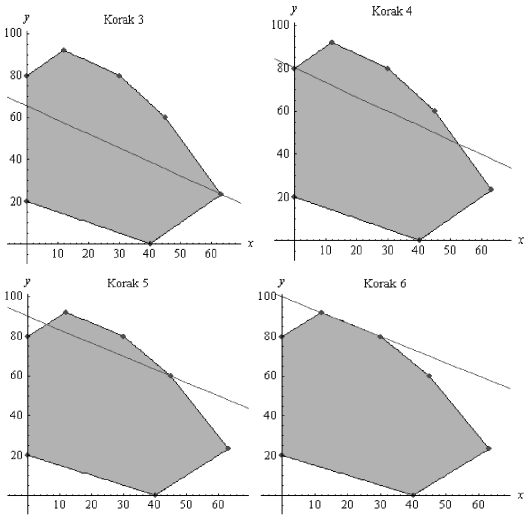
$$\begin{aligned} \max \quad & z(x,y) = 8x + 12y \\ \text{p.o.} \quad & 8x + 4y \leq 600 \\ & 2x + 3y \leq 300 \\ & 4x + 3y \leq 360 \\ & 5x + 10y \geq 600 \\ & x - y \geq -80 \\ & x - y \leq 40 \\ & x, y \geq 0 \end{aligned}$$

Problem rešavamo sledećom naredbom:

```
Geom[8x+12y, {8x+4y<=600, 2x+3y<=300, 4x+3y<=360, 5x+10y>=600,x>=0, y>=0}]
```

Program daje grafički prikaz skupa dopustivih rešenja Ω_P, kao i prave koja odgovara ciljnoj funkciji ($\mathcal{H}_{z,d}$). Pravu pomeramo "naviše", u pravcu vektora $c = \begin{bmatrix} 8 \\ 12 \end{bmatrix}$ sve dok postoji presek sa poligonom dopustivih rešenja. Na slici su prikazani položaji prave $\mathcal{H}_{z,d}$ kada prolazi kroz temena poligona (ekstremne tačke):





Slika 1.3.6. Vizuelizacija geometrijskog metoda.

Program takođe daje optimalno rešenje (ili izraz koji opisuje sva optimalna rešenja). U ovom slučaju to je:

$$x^* = \lambda \begin{bmatrix} \frac{190}{3} \\ \frac{70}{3} \end{bmatrix} + (1-\lambda) \begin{bmatrix} 45 \\ 60 \end{bmatrix},\ 0 \leq \lambda \leq 1$$

2 Simpleks metod

Simpleks metod se zasniva na tri bitna principa:

1. Postoji mogućnost određivanja bar jednog dopustivog rešenja (plana), koji se često naziva bazičnim planom ili dopustivim bazičnim planom.

2. Postoji mogućnost da se proveri da li je bazični dopustivi plan optimalan ili ne.

3. Postoji mogućnost, da se u slučaju da dopustivi plan nije optimalan, izabere novi, koji je bliži optimalnom.

Prema gore navedenom, simpleks metod se zasniva na sukcesivnom poboljšanju početnog dopustivog plana, sve dok se ne dobije optimalan plan. Algoritam simpleks metoda takođe omogućava da se ustanovi da li je zadatak rešiv ili ne, odnosno, da li postoji protivurečnost u ograničenjima.

2.1 Osobine simpleks metoda

Primetimo prvo da vektor x^* u kome ciljna funkcija $z(x)$ dostiže ekstremnu vrednost ne mora biti jedinstven. Sledeća teorema pokazuje da ciljna funkcija dostiže ekstremnu vrednost u nekoj od ekstremnih tačaka skupa Ω_P.

Teorema 2.1.1 *Ako je* $\Omega_P = \{x : Ax = b, x \geq 0\}$ *ograničen skup i* $z(x) = c_1x_1 + \cdots + c_nx_n$ *zadata linearna funkcija, tada postoji bar jedna ekstremna tačka* $x^* \in \Omega_p$

takva da je

$$\inf_{x\in\Omega_P} z(x) = z(x^*).$$

Skup $\{x|\, x \in \Omega_P,\, z(x) = z(x^*)\}$ *je konveksan.*

Dokaz. Neka su $x^1, \ldots, x^p$ ekstremne tačke skupa Ω_P i neka je x^* ekstremna tačka za koju je $z(x^i) \geq z(x^*)$, $i = 1, \ldots, p$. Kako je svako $x \in \Omega_P$ konveksna kombinacija ekstremnih tačaka, to postoje pozitivni skalari $\lambda_1, \ldots, \lambda_p$ takvi da je

$$x = \sum_{k=1}^{p} \lambda_k x^k, \quad \sum_{k=1}^{p} \lambda_k = 1.$$

Sada je

$$z(x) = z\left(\sum_{k=1}^{p} \lambda_k x^k\right) = \sum_{k=1}^{p} \lambda_k z(x^k) \geq \sum_{k=1}^{p} \lambda_k z(x^*) = z(x^*),$$

što dokazuje da z dostiže minimum u x^*.

Neka je $z(x^1) = z(x^2) = z(x^*)$. Tada je

$$z(\lambda x^1 + (1-\lambda)x^2) = \lambda z(x^1) + (1-\lambda) z(x^2) = z(x^*)$$

za svako $0 \leq \lambda \leq 1$. □

Značaj prethodne teoreme je u tome što se broj potencijalnih ekstremuma funkcije $z(x)$ redukuje sa (u opštem slučaju) beskonačnog skupa Ω_P na konačan skup temena koji ima maksimalno $\binom{n}{m}$ elemenata. Formalno, dovoljno je izračunati vrednosti ciljne funkcije u svim ekstremnim tačkama skupa Ω_P i odrediti onu tačku, ili tačke, u kojima je vrednost funkcije $z(x)$ ekstremna.

Neka su $j_1 < \cdots < j_m$ indeksi takvi da je $\mathcal{A}_B = \{K_{j_1}, \ldots, K_{j_m}\}$ jedna baza matrice A i neka je $B = \{j_1, \ldots, j_m\}$. Ako se drugačije ne naglasi, podrazumevamo da se bazi $\mathcal{A}_B$ pridružuje bazična podmatrica $A_B = [K_{j_1} \ldots K_{j_m}]$. Ukoliko je bazično rešenje dopustivo, bazu $\mathcal{A}_B$ nazivamo *dopustivom bazom.* Neka je

$$N = \{i_1, \ldots, i_{n-m}\} = \{1, \ldots, n\} \backslash B$$

rastuće uređen skup i neka je $A_N = \{K_{i_1}, \ldots, K_{i_{n-m}}\}$. Sada se sistem ograničenja $Ax = b$ može zapisati u obliku

$$A_B x_B + A_N x_N = b, \tag{2.1.1}$$

gde je x_B vektor bazičnih, a x_N vektor nebazičnih promenljivih. Sada iz (2.1.1) sledi $x_B = A_B^{-1} b - A_B^{-1} A_N x_N$, tako da je moguće eliminisati bazične promenljive iz ciljne funkcije. Analogno sa x_B i x_N definišemo c_B i c_N tako da je

$$c^T x = c_B^T x_B + c_N^T x_N = c_B^T A_B^{-1} b + (c_N^T - c_B^T A_B^{-1} A_N) x_N,$$

odakle je $c^T x^* = c_B^T A_B^{-1} b$ za bazično rešenje x^*. Uvedimo oznake

$$d' = c_B^T A_B^{-1} b,\ (c'_N)^T = c_N^T - c_B^T A_B^{-1} A_N,\ A'_N = A_B^{-1} A_N,\ b' = A_B^{-1} b.$$

Sada se standardni oblik problema (1.1.4) može zapisati u ekvivalentnom obliku

$$\begin{array}{lll} \min & d' & +(c'_N)^T x_N, \\ \text{p.o.} & x_B & +A'_N x_N = b', \\ & x_B & \geq 0,\ x_N \geq 0. \end{array} \tag{2.1.2}$$

Problem (2.1.1) je *kanonski oblik problema* (1.1.4) *u odnosu na bazu* $\mathcal{A}_B$. U kanonskom obliku matrica odgovarajućeg sistema jednačina uz bazične promenljive sadrži različite jedinične kolone, a u ciljnoj funkciji su koeficijenti uz bazične promenljive jednaki nuli.

Lema 2.1.1 *Vektor koeficijenata ciljne funkcije u kanonskom obliku problema linearnog programiranja je jednoznačno određen bazom* $\mathcal{A}_B$*, tj. ne zavisi od poretka kolona u matrici* $\mathcal{A}_B$*.*

Dokaz. Neka je bazi $\mathcal{A}_B$ pridružena matrica $\bar{A}_B$ sa poretkom kolona različitim od $j_1, \ldots, j_m$ tada je $\bar{A}_B = A_B P$, gde je P permutaciona matrica. Kako je $P^{-1} = P^T$, polazni problem se može zapisati u obliku

$$\begin{gathered} \min \bar{c}_B^T \bar{x}_B + c_N^T x_N, \\ \bar{A}_B \bar{x}_B + A_N x_N = b, \\ \bar{x}_B \geq 0, \quad x_N \geq 0, \end{gathered}$$

gde je $\bar{x}_B = P^{-1} x_B$, $\bar{c}_B = P^{-1} c_B$, odakle se dobija kanonski oblik

$$\begin{array}{lll} \min & \bar{c}_B^T & (\bar{A}_B)^{-1} b + (c_N^T - \bar{c}_B^T (\bar{A}_B)^{-1} A_N) x_N, \\ \text{p.o.} & \bar{x}_B & +(\bar{A}_B)^{-1} A_N x_N = (\bar{A}_B)^{-1} b, \\ & \bar{x}_B & \geq 0, \quad x_N \geq 0. \end{array} \tag{2.1.3}$$

S obzirom da je $\bar{c}_B^T (\bar{A}_B)^{-1} = c_B^T (P^{-1})^T (P^{-1}) A_B^{-1} = c_B^T A_B^{-1}$, to su slobodni članovi i vektori koeficijenata u ciljnim funkcijama problema (2.1.1) i (2.1.3) jednaki, dok se sistem jednačina u (2.1.3) može zapisati u obliku $P^{-1}(x_B + A_B^{-1} A_N x_N) = P^{-1} b$, tj. radi se o permutaciji jednačina sistema u (2.1.1). □

Koristićemo problem linearnog programiranja u simetričnom obliku:

$$\begin{array}{ll} \min & z(x) = \displaystyle\sum_{i=1}^{n} c_i x_i + d \\ \text{p.o} & \displaystyle\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, m, \\ & x_j \geq 0, \quad j = 1, \ldots, n. \end{array} \tag{2.1.4}$$

Definicija 2.1.1 *Problem* (2.1.4) *je bazično dopustiv ako je* $b_i \geq 0$ *za svako* $i = 1, \ldots, m$

Prethodna definicija je u bliskoj vezi sa pojmom bazično dopustivog rešenja. Naime, ako uvedemo dodatne slack promenljive, dobijamo:

$$\begin{aligned} \min \quad & z(x) = \sum_{i=1}^{n} c_i x_i + d \\ \text{p.o} \quad & \sum_{j=1}^{n} a_{ij} x_j - b_i = -x_{n+i}, \quad i = 1, \ldots, m. \end{aligned} \tag{2.1.5}$$

Oblik (2.1.5) se često naziva *kanonski* oblik problema linearnog programiranja. Promenljive na desnoj strani (u ovom slučaju $x_{n+1}, \ldots, x_{n+m}$) nazivamo *bazičnim* dok one na levoj strani jednačina (u ovom slučaju $x_1, \ldots, x_n$) *nebazičnim*. Bazične i nebazične promenljive nadalje ćemo označavati sa $x_{B,1}, \ldots, x_{B,m}$ i $x_{N,1}, \ldots, x_{N,n}$, redom. Ako sada stavimo $x_{N,1} = \ldots = x_{N,n} = 0$, tada je $x_{B,i} = b_i$ za $i = 1, \ldots, m$. Ovako dobijeno rešenje je bazično dopustivo, akko je $b_i \geq 0$ za svako $i = 1, \ldots, m$. Primetimo takođe da je matrica sistema u standardnom obliku (2.1.5) $A' = [A|I_m]$ gde je I_m jedinična matrica. I_m je bazična matrica koja odgovara upravo konstruisanom bazičnom rešenju.

Posledica 2.1.1 *Neka u bazično dopustivom kanonskom obliku* (2.1.5) *važi* $c_j \geq 0$ *za svako* $j = 1, \ldots, n$. *Tada je bazično rešenje* $x^* = (0, \ldots, 0, b_1, \ldots, b_m)$ *optimalno.*

Dokaz. S obzirom da je $x_B^* = b' \geq 0$ a $x_N^* = 0$, može se zaključiti da je x^* dopustivo rešenje. Ako je x proizvoljno dopustivo rešenje, tada iz $x \geq 0$ i $x_{N,j}^* = 0$ za $j = 1, \ldots, n$ sledi

$$z(x) = c_1 x_{N,1} + \ldots + c_n x_{N,n} + d \leq d = f(x^*)$$

pa je x^* optimalno rešenje. □

Fundamentalna osobina kanonskog oblika je da se na osnovu koeficijenata može odrediti da li je odgovarajuće bazično rešenje optimalno, kao i u kom slučaju je ciljna funkcija neograničena odozdo.

Teorema 2.1.2 *Neka je u kanonskom obliku* (2.1.3)

$$c_j' \geq 0, \ j \in N, \ b_i' \geq 0, \ i = 1, \ldots, m.$$

Tada je bazično rešenje koje odgovara bazi $\mathcal{A}_B$ *optimalno.*

Dokaz. Iz $x_N^* = 0$ sledi $x_B^* = b' \geq 0$, dakle x^* je dopustivo rešenje. Ako je x proizvoljno dopustivo rešenje, tada iz $x \geq 0$ i $x_j^* = 0$, $j \in N$ sledi

$$c^T x = d' + \sum_{j \in N} c_j' x_j \geq d' = d' + \sum_{j \in N} c_j' x_j^* = c^T x^*,$$

pa je x^* optimalno rešenje. □

Teorema 2.1.3 *Neka je u kanonskom obliku* (3.1.2) $b'_i \geq 0$, $i = 1, \ldots, m$ *i za neko* $k \in N$ *je ispunjeno* $c'_k < 0$ *i* $a'_{ik} \leq 0$, $i = 1, \ldots, m$. *Tada je ciljna funkcija na dopustivom skupu neograničena odozdo.*

Dokaz. Neka su za $t \geq 0$ koordinate tačke $x(t)$ definisane sa

$$x_{j_i}(t) = b'_i - a'_{ik}t,\ i = 1, \ldots, m, \quad x_j(t) = 0,\ j \in N\backslash\{k\}, \quad x_k(t) = t.$$

Direktno se proverava da je $x(t)$ dopustiva tačka problema (2.1.1) i važi

$$c^T x(t) = d' + \sum_{j \in N} c'_j x_j(t) = d' + c'_k t \to -\infty, \quad t \to \infty.$$

□

Ukoliko u kanonskom obliku nisu ispunjeni uslovi iz Teoreme 3.1.2 ili Teoreme 3.1.3 tada možemo preći na kanonski oblik koji odgovara nekoj novoj susednoj bazi ali tako da se vrednost ciljne funkcije u novom bazično dopustivom rešenju smanji ili bar ne poveća. Ta ideja je u osnovi simpleks metoda. Pri tome se koristi *tablični zapis* problema linearnog programiranja.

Elementarne transformacije LP-tablice su:
1) množenje $i-$te vrste sa $\lambda \neq 0$, za $i \in \{1, \ldots, m\}$,
2) dodavanje $i-$te vrste, za $i = 1, \ldots, m$, $j-$toj vrsti za $j \in \{0, 1, \ldots, m\}$.

Lema 2.1.2 *LP-tablica se primenom konačno mnogo elementarnih transformacija transformiše u LP-tablicu koja odgovara ekvivalentnom problemu linearnog programiranja.*

Dokaz. Elementarne transformacije sa $i-$tom vrstom za $i \in \{1, \ldots, m\}$ očigledno ne menjaju dopustivi skup. Dodavanje $i-$te vrste za $i \in \{1, \ldots, m\}$ nultoj vrsti transformiše je u vrstu

$$[-d + b_i \,|\, c_1 + a_{i1} \cdots c_n + a_{in}],$$

koja odgovara novoj ciljnoj funkciji

$$\begin{aligned} z &= d - b_i + (c_1 + a_{i1})x_1 + \cdots + (c_n + a_{in})x_n \\ &= d + c_1x_1 + \cdots + c_nx_n + (-b_i + a_{i1}x_1 + \cdots + a_{in}x_n) \\ &= d + c_1x_1 + \cdots + c_nx_n, \end{aligned}$$

tj. na dopustivom skupu vrednosti transformisane ciljne funkcije i polazne ciljne funkcije su jednake, pa su problemi ekvivalentni. □

Iz Leme 3.1.2 sledi da se postupkom koji je analogan Gauss-Jordan-om metodu dobijaju različite, međusobno ekvivalentne, formulacije problema.

U ovoj glavi prikazaćemo simpleks metod za rešavanje zadataka linearnog programiranja. Ovaj metod je razvio 1940. godine George B. Danzig. Najpre ćemo pokazati algebarski pristup, koji je pogodan kada se radi o zadatku manjih dimenzija, a zatim ćemo dati uprošćenje simpleks algoritma pomoću tzv. simpleks-tabela.

2.2 Algebarska suština simpleks metoda

Neka je zadat sledeći problem linearnog programiranja sa $r = m$ linearno nezavisnih ograničenja:

$$\begin{aligned} \min \quad & z(x) = c^T x = \sum_{i=1}^{n} c_i x_i \\ \text{p.o} \quad & \sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1, \ldots, m, \\ & x_j \geq 0, \quad j = 1, \ldots, n. \end{aligned}$$

Optimalno rešenje se nalazi u jednom od temena konveksnog poliedra, gde je najmanje $k = n-m$ promenljivih jednako nuli. Izaberimo proizvoljno k promenljivih za nezavisne (slobodne) promenljive i pomoću njih izrazimo zavisne promenljive. Neka su nezavisne promenljive $x_1, \ldots, x_k$ takve da kada pomoću njih izrazimo $m = n - k$ zavisnih promenljivih:

$$\begin{aligned} -x_{k+1} &= \alpha_{k+1,1}x_1 + \alpha_{k+1,2}x_2 + \cdots + \alpha_{k+1,k}x_k - \beta_{k+1} \\ \cdots &\quad \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots \\ -x_n &= \alpha_{n,1}x_1 + \alpha_{n,2}x_2 + \cdots + \alpha_{n,k}x_k - \beta_n \end{aligned} \tag{2.2.1}$$

koeficijenti $\beta_{k+1}, \ldots, \beta_n$ jesu pozitivni. Funkcija cilja u tački $x = (x_1, \ldots, x_k)$ dobija vrednost

$$z(x_1, \ldots, x_n) = z(x_1, \ldots, x_k) = \gamma_0 + \gamma_1 x_1 + \cdots + \gamma_k x_k. \tag{2.2.2}$$

Ako stavimo da je $x_1 = x_2 = \cdots = x_k = 0$ dobijamo bazično rešenje

$$(\underbrace{0, 0, \ldots, 0}_{k}, \beta_{k+1}, \beta_{k+2}, \ldots, \beta_n), \quad \beta_{k+1}, \ldots, \beta_n \geq 0$$

i minimalnu vrednost

$$z = \gamma_0.$$

Razmotrimo sada da li možemo da umanjimo vrednost ciljne funkcije z uvećavajući neku od promenljivih $x_1, x_2, \ldots, x_k$. Ako su svi koeficijenti $\gamma_1, \gamma_2, \ldots, \gamma_k$ uz promenljive $x_1, x_2, \ldots, x_k$ u funkciji cilja pozitivni, onda uvećavajući neku od ovih promenljivih (na vrednosti koje su veće od nule) ne možemo da umanjimo vrednost funkcije cilja. To znači da je dobijeno bazično rešenje i optimalno. Međutim, ako između koeficijenata $\gamma_1, \gamma_2, \ldots, \gamma_k$ ima i negativnih, onda uvećavajući vrednost promenljive ispred koje je koeficijent u funkciji cilja negativan, možemo da umanjimo vrednost funkcije cilja z, tj. možemo da dođemo do boljeg bazičnog rešenja. Neka je, na primer negativan koeficijent γ_j u izrazu (2.2.2). Sada ima smisla uvećati promenljivu x_j i na taj način preći od dobijenog bazičnog rešenja, gde je x_j bilo jednako nuli, do novog bazičnog rešenja gde će umesto x_j biti jednaka nuli neka od zavisnih promenljivih. Uvećavajući x_j smanjujemo vrednost funkcije cilja z ali moramo

da pazimo da nam pri tom neka od zavisnih promenljivih $x_{k+1}, x_{k+2}, \ldots, x_n$ ne postane negativna. One, očigledno, neće postati negativne, ako su u jednačinama (2.2.1) koeficijenti $\alpha_{k+1,j}, \ldots, \alpha_{nj}$ uz x_j negativni, pa se tada x_j može uvećati do beskonačnosti, za neko $j, 1 \leq j \leq k$. To znači da i ciljna funkcija nije ograničena sa donje strane (tj. $z_{\min} = -\infty$). U tom slučaju zadatak nema optimalno rešenje. Pretpostavimo sada da se između koeficijenata uz x_j u jednačinama (2.2.1) nalaze i pozitivni koeficijenti. Neka je, na primer, u jednačini kojom se izražava zavisna promenljiva x_i, $i \in \{k+1, \ldots, n\}$

$$-x_i = \alpha_{i1}x_1 + \cdots + \alpha_{ij}x_j + \cdots + \alpha_{ik}x_k - \beta_i$$

pozitivan koeficijent uz x_j, tj. $\alpha_{ij} > 0$. Ako stavimo $x_1 = \cdots = x_{j-1} = x_{j+1} = \cdots = x_k = 0$, dobijamo sistem

$$\begin{aligned} -x_{k+1} &= \alpha_{k+1,j}x_j - \beta_{k+1} \\ &\cdots \\ -x_n &= \alpha_{n,j}x_j - \beta_n. \end{aligned}$$

To znači da x_j možemo uvećavati do vrednosti

$$\frac{\beta_i}{\alpha_{ij}}, \quad \beta_i > 0, \alpha_{ij} > 0, \ i = k+1, \ldots, n$$

jer za tu vrednost promenljive x_j promenljiva x_i postaje jednaka nuli: $x_i = 0$. Pri daljem uvećavanju x_j promenljiva x_i bi postala negativna. Izaberimo sada između promenljivih $x_{k+1}, x_{k+2}, \ldots, x_n$ promenljivu x_p prema uslovu

$$\frac{\beta_p}{\alpha_{pj}} = \min\left\{\frac{\beta_i}{\alpha_{ij}}, \quad \alpha_{ij} > 0, \ k+1 \leq i \leq n\right\}.$$

Tada se veličina x_j izražava iz jednačine

$$-x_p = \alpha_{p1}x_1 + \cdots + \alpha_{p,j}x_j + \cdots + \alpha_{pk}x_k - \beta_p$$

i zamenjuje u ostalim jednačinama (2.2.1) i u funkciji cilja (2.2.2). Na taj način, umesto promenljive x_j među nezavisne promenljive je ušla promenljiva x_p, a promenljiva x_j, koja je u prethodnom bazičnom rešenju bila nezavisna, sada postaje zavisna promenljiva. Simbolično, tu promenu označavamo oznakom $x_p \leftrightarrow x_j$. Znači, sada su zavisne promenljive

$$x_j, x_{k+1}, \ldots, x_{p-1}, x_{p+1}, \ldots, x_n.$$

Ove zavisne promenljive se izražavaju pomoću nezavisnih promenljivih

$$x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k, x_p.$$

Takođe, i ciljna funkcija se izražava pomoću nezavisnih promenljivih.

Izvedeni postupak se sada ponavlja. Ako su svi koeficijenti uz nezavisne promenljive $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k, x_p$ u funkciji cilja pozitivni, onda je za

$$x_j = x_{k+1} = \cdots = x_{p-1} = x_{p+1} = \cdots = x_n = 0$$

dobijeno bazično rešenje i optimalno rešenje. Ukoliko ima i negativnih koeficijenata uz nezavisne promenljive postupak se ponavlja dok se ne dođe do optimalnog rešenja.

Neka je, u opštem slučaju zadat bazično dopustiv problem (2.1.5) kod koga je $c_j < 0$ za neko $j \in \{1, \ldots, n\}$ i $a_{ij} > 0$ za neko $i \in \{1, \ldots, m\}$. S obzirom na $c_j < 0$, ima smisla uvećati promenljivu $x_{N,j}$ i na taj način preći od dobijenog bazičnog rešenja, gde je $x_{N,j}$ bilo jednako nuli, do novog bazičnog rešenja gde će umesto $x_{N,j}$ biti jednaka nuli neka od zavisnih promenljivih. Uvećavajući $x_{N,j}$ smanjujemo vrednost funkcije cilja $z(x)$ ali moramo da pazimo da nam pri tom neka od bazičnih promenljivih $x_{B,1}, x_{B,2}, \ldots, x_{B,m}$ ne postane negativna. Ukoliko je $a_{sj} < 0$, promenljiva $x_{B,s}$ očigledno neće postati negativna. Posmatrajmo sada jednačinu:

$$-x_{B,i} = a_{i1}x_{N,1} + \ldots + a_{ij}x_{N,j} + \ldots + a_{in}x_{N,n} - b_i$$

Ako stavimo $x_{N,1} = \cdots = x_{N,j-1} = x_{N,j+1} = \cdots = x_{N,n} = 0$, dobijamo:

$$\begin{aligned} -x_{B,1} &= a_{1j}x_{N,j} - b_i \\ &\cdots \\ -x_{B,m} &= a_{nj}x_{N,j} - b_n. \end{aligned}$$

To znači da $x_{N,j}$ možemo uvećavati do vrednosti:

$$\frac{b_i}{a_{ij}}, \quad b_i > 0, a_{ij} > 0,$$

jer za tu vrednost promenljive $x_{N,j}$ promenljiva $x_{B,i}$ postaje jednaka nuli. Pri daljem uvećavanju $x_{N,j}$ promenljiva $x_{B,i}$ bi postala negativna. Izaberimo sada bazičnu promenljivu $x_{B,p}$ prema uslovu

$$\frac{b_p}{a_{pj}} = \min\left\{\frac{b_i}{a_{ij}}, \quad a_{ij} > 0,\ k+1 \leq i \leq n\right\}. \tag{2.2.3}$$

Izvršimo sada zamenu promenljivih $x_{B,p}$ i $x_{N,j}$, tj izrazimo sada promenljivu $x_{N,j}$ iz jednačine

$$-x_{B,p} = a_{p1}x_{N,1} + \cdots + a_{pj}x_{N,j} + \cdots + a_{pn}x_{N,n} - b_p$$

i zamenimo je u ostalim jednačinama i u funkciji cilja. Na taj način, umesto promenljive $x_{N,j}$ među nebazične promenljive je ušla promenljiva $x_{B,p}$, a promenljiva $x_{N,j}$ koja je u prethodnom bazičnom rešenju bila nezavisna, sada postaje zavisna promenljiva. Ovim smo dobili novo bazično dopustivo rešenje:

$$x^1 = (x_N^1, x_B^1) = \left((0, \ldots, 0, b_p^1, 0, \ldots, 0), (b_1^1, \ldots, b_{p-1}^1, 0, b_{p+1}^1, \ldots, b_n^1)\right)$$

gde je:

$$b_p^1 = \frac{b_p}{a_{pj}}, \quad b_l^1 = b_l - a_{lj}\frac{b_p}{a_{pj}}, \quad l \neq p$$

kao i ekvivalentan problem u kanonskom obliku. U tački x^1, funkcija cilja ima veću vrednost nego u polaznom bazično dopustivom rešenju. Ako sada nastavimo da primenjujemo isti postupak sa novim kanonskim oblikom, funkcija cilja će se povećavati i u jednom trenutku ćemo sigurno doći u situaciju da možemo da primenimo lemu (2.1.1) ili dolazimo do zaključka da je ciljna funkcija neograničena. Ovo intuitivno razmatranje je ključno za simpleks metod, i biće u sledećem odeljku formalnije sprovedeno. Takođe, uvešćemo pojam Tuckerove tabele koji u mnogome pojednostavljuje upravo opisan postupak konstruisanja novog bazičnog rešenja. Ovakav pristup se najčešće koristi prilikom implementacije simpleks metoda.

Primer 2.2.1 Rešiti sledeći problem

$$\begin{array}{lll} \min & z(x) & = 5x_1 - 2x_3 \\ \text{p.o.} & & -5x_1 - x_2 + 2x_3 \leq 2 \\ & & -x_1 \quad\quad + x_3 + \ x_4 \leq 5 \\ & & -3x_1 \quad\quad\quad + 5x_4 \leq 7 \\ & & x_i \geq 0, \quad i = 1, 2, 3, 4. \end{array}$$

Rešenje: Ograničenjima dodajemo dopunske promenljive tako da se zadatak transformiše na sledeći standardni oblik:

$$\begin{array}{lll} \multicolumn{3}{l}{\min z(x) = 5x_1 - 2x_3} \\ \text{p.o.} & -5x_1 - x_2 + 2x_3 \quad\quad + x_5 \quad\quad\quad = 2 \\ & -x_1 \quad\quad + x_3 + x_4 \quad\quad + x_6 \quad\quad = 5 \\ & -3x_1 \quad\quad\quad + 5x_4 \quad\quad\quad + x_7 = 7 \\ & x_i \geq 0, \quad i = 1, \ldots, 7. \end{array}$$

Kako je $r = m = 3$ i $k = n - m = 7 - 3 = 4$, znači da je broj nezavisnih promenljivih 4, a broj zavisnih 3. Neka su nezavisne promenljive x_1, x_2, x_3 i x_4. Iz sistema jednačina dobijamo:

$$\begin{array}{lll} -x_5 & = & -5x_1 - x_2 + 2x_3 - 2 \\ -x_6 & = & -x_1 \quad\quad + x_3 + x_4 - 5 \\ -x_7 & = & -3x_1 \quad\quad\quad + 5x_4 - 7 \\ z(x) & = & 5x_1 - 2x_3 \end{array}$$

Početno bazično rešenje za $x_1 = x_2 = x_3 = x_4 = 0$ jednako je

$$I : (0, 0, 0, 0, 2, 5, 7) \text{ i } z_1(x) = 0.$$

Kako je koeficijent uz x_3 u funkciji cilja negativan, vrednost funkcije cilja se može umanjiti ako x_3 uzme pozitivne vrednosti. Iz sistema jednačina se dobija za $x_1 = x_2 = x_4 = 0$:

$$x_5 = 0 \text{ za } x_3 = 1 = \frac{\beta_1}{\alpha_{13}}$$

$$x_6 = 0 \text{ za } x_3 = 5 = \frac{\beta_2}{\alpha_{23}}.$$

Takođe, rašćenje x_3 ne utiče na promenu vrednosti x_7, tj., u ovom slučaju može da poraste do $+\infty$. Kako je

$$\min\{1, 5, +\infty\} = 1.$$

To znači da promenljive x_3 i x_5 menjaju uloge u drugom bazičnom rešenju ($x_3 \leftrightarrow x_5$). Eliminacijom x_3 iz prvog ograničenja dobija se

$$\begin{array}{lll} -x_3 & = & -\frac{5}{2}x_1 - \frac{1}{2}x_2 + \frac{1}{2}x_5 - 1 \\ -x_6 & = & -x_1 + \left(1 + \frac{5}{2}x_1 + \frac{1}{2}x_2 - \frac{1}{2}x_5\right) + x_4 - 5 = \frac{3}{2}x_1 + \frac{1}{2}x_2 + x_4 - \frac{1}{2}x_5 - 4 \\ -x_7 & = & -3x_1 + 5x_4 - 7 \\ z(x) & = & -2 - x_2 + x_5. \end{array}$$

Drugo bazično rešenje je jednako

$$II : (0, 0, 1, 0, 0, 4, 7) \text{ i } z_2(x) = -2.$$

Ni ovo rešenje nije optimalno jer se, zbog negativnog koeficijenta uz x_2 u funkciji cilja, njena vrednost može umanjiti povećanjem vrednosti promenljive x_2. Povećanjem vrednosti x_2 nenegativnost promenljive x_3 se neće izgubiti, niti promenljive x_7. Međutim kod promenljive x_6 vidimo da x_2 može da uzme najveću vrednost

$$\frac{\beta_2}{\alpha_{25}} = \frac{4}{\frac{1}{2}} = 8,$$

kada x_6 postaje jednako nuli, to jest, imamo $x_2 \leftrightarrow x_6$. Sledi

$$\begin{aligned}
-x_2 &= 3x_1 + 2x_4 - x_5 + 2x_6 - 8 \\
-x_3 &= -\frac{5}{2}x_1 - \frac{1}{2}(8 - 3x_1 - 2x_4 + x_5 - 2x_6) + \frac{1}{2}x_5 - 1 = -x_1 + x_4 + \frac{1}{2}x_5 + x_6 - 5 \\
-x_7 &= -3x_1 + 5x_4 - 7 \\
z(x) &= -10 + 3x_1 + 2x_4 + 2x_6.
\end{aligned}$$

Treće bazično rešenje je jednako:

$$III : (0, 8, 5, 0, 0, 0, 7) \text{ i } z_3(x) = -10$$

Kako su sada svi koeficijenti uz promenljive u funkciji cilja pozitivni, to je ovo treće bazično rešenje i optimalno rešenje, a $z_{min} = -10$.

2.3 Pojam Tuckerove tabele i Simpleks metod za bazično dopustive kanonske oblike

Neka je zadat jedan kanonski oblik problema linearnog programiranja:

$$\begin{aligned}
&\max f(x) = c_1 x_{N,1} + \ldots + c_n x_{N,n} + d \\
&\text{p.o. } a_{11}x_{N,1} + a_{12}x_{N,2} + ... + a_{1n}x_{N,n} - b_1 = -x_{B,1} \\
&\quad a_{21}x_{N,1} + a_{22}x_{N,2} + ... + a_{2n}x_{N,n} - b_2 = -x_{B,2} \\
&\quad \ldots\ldots\ldots \\
&\quad a_{m1}x_{N,1} + a_{m2}x_{N,2} + ... + a_{mn}x_{N,n} - b_m = -x_{B,m}.
\end{aligned} \tag{2.3.1}$$

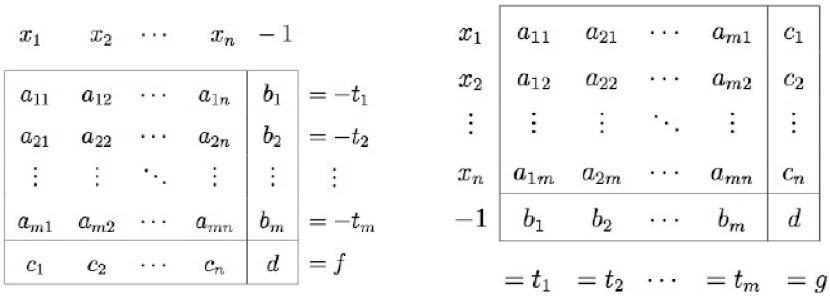
Problem možemo tabelarno prikazati na sledeći način:

$x_{N,1}$	$x_{N,2}$	$\cdots$	$x_{N,n}$	-1		
a_{11}	a_{12}	$\cdots$	a_{1n}	b_1	$=$	$-x_{B,1}$
a_{21}	a_{22}	$\cdots$	a_{2n}	b_2	$=$	$-x_{B,2}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$		$\vdots$
a_{m1}	a_{m2}	$\cdots$	a_{mn}	b_m	$=$	$-x_{B,m}$
c_1	c_2	$\cdots$	c_n	d	$=$	f

(2.3.2)

Ovu tabelu nazivamo ***Tuckerovom tabelom*** za problem (2.3.1). Uvedimo oznake $a_{m+1,j} = c_j$ za $j = 1, \ldots, n$ kao i $a_{i,n+1} = b_i$ za $i = 1, \ldots, n$. Takođe, neka je $a_{m+1,n+1} = d$. Sada nam je za opisivanje kanonskog oblika linearnog programiranja dovoljna *proširena Tuckerova tabela*: $\overline{A} = \{a_{ij}\}_{i=\overline{1,m+1}, j=\overline{1,n+1}}$. U nastavku ćemo često poistovećivati matrice $\overline{A}$ i A kad god nema opasnosti od zabune.

U literaturi se sreću dva oblika Tuckerovih tabela:



gde su $x_1, \ldots, x_n$ nebazične, a $t_1, \ldots, t_m$ bazične promenljive.

Na kraju prethodnog odeljka bilo je potrebno izraziti nebazičnu promenljivu $x_{N,j}$ iz p-te jednačine sistema (2.3.1) i zameniti je u ostalim jednačinama i u funkciji cilja. Na taj način, promenljiva $x_{B,p}$ postaje nebazična, dok $x_{N,j}$ postaje bazična.

Posmatrajmo sada šta se dešava sa matricom sistema A' odgovarajućeg standardnog oblika (2.1.2). U matrici A' kolone koje odgovaraju bazičnim promenljivima formiraju jediničnu matricu dok ostale formiraju Tuckerovu tabelu A. Posle zamene promenljivih kolona $K_{v_{N,j}}$ matrice A' postaje jednaka p-toj koloni jedinične matrice. Da bi to uradili, dovoljno je da za svako $i = 1, \ldots, m$, $i \neq p$ od i-te vrste matrice A' oduzmemo p-tu vrstu pomnoženu sa:

$$\frac{a'_{i,v_{n,j}}}{a'_{p,v_{n,j}}} = \frac{a_{ij}}{a_{pj}}, \quad A' = [a'_{ij}]_{i=\overline{1,m},j=\overline{1,m+n}}$$

dok p-tu vrstu samo podelimo sa $a_{pj} = a'_{p,v_{n,j}}$. Na isti način transformišemo vektor b' i funkciju cilja smatrajući ih redom $n+m+1$-om kolonom i $m+1$-om vrstom matrice A'. Ovu transformaciju opisujemo u vektorskom i skalarnom obliku na sledeći način:

$$V_q^1 = V_q - \frac{a_{qj}}{a_{pj}} V_p, \qquad (a')_{ql}^1 = a'_{ql} - \frac{a'_{pl} a'_{q,v_{B,j}}}{a'_{p,v_{B,j}}} \tag{2.3.3}$$

Gde je sa V_i označena i-ta vrsta $A'_{i\bullet}$ matrice A'. U tom slučaju nova Tuckerova tabela jednaka je:

$$A^1 = \begin{array}{ccccccccc}
x_{N,1} & \cdots & x_{N,j-1} & x_{B,p} & x_{N,j+1} & \cdots & x_{N,n} & -1 & \\
a^1_{11} & \cdots & a^1_{1,j-1} & a^1_{1j} & a^1_{1,j+1} & \cdots & a^1_{1n} & b^1_1 & = -x_{B,1} \\
a^1_{21} & \cdots & a^1_{2,j-1} & a^1_{2j} & a^1_{2,j+1} & \cdots & a^1_{2n} & b^1_2 & = -x_{B,2} \\
\vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\
a^1_{p-1,1} & \cdots & a^1_{p-1,j-1} & a^1_{p-1,j} & a^1_{p-1,j+1} & \cdots & a^1_{p-1,n} & b^1_{p-1} & = -x_{B,p-1} \\
a^1_{p1} & \cdots & a^1_{p,j-1} & a^1_{pj} & a^1_{p,j+1} & \cdots & a^1_{pn} & b^1_p & = -x_{B,j} \\
a^1_{p+1,1} & \cdots & a^1_{p+1,j-1} & a^1_{p+1,j} & a^1_{p+1,j+1} & \cdots & a^1_{p+1,n} & b^1_{p+1} & = -x_{B,p+1} \\
\vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\
a^1_{m1} & \cdots & a^1_{m,j-1} & a^1_{mj} & a^1_{m,j+1} & \cdots & a^1_{mn} & b^1_m & = -x_{B,m} \\
c^1_1 & \cdots & c^1_{j-1} & c^1_j & c^1_{j+1} & \cdots & c^1_n & d^1 & = f
\end{array} \tag{2.3.4}$$

gde su elementi a^1_{ql} dati pomoću izraza:

$$\begin{aligned} a^1_{pj} =& \frac{1}{a_{pj}}; \\ a^1_{pl} =& \frac{a_{pl}}{a_{pj}}, && l \neq j; \\ a^1_{qj} =& -\frac{a_{qj}}{a_{pj}}, && q \neq p; \\ a^1_{ql} =& a_{ql} - \frac{a_{pl}a_{qj}}{a_{pj}}, && q \neq p,\ l \neq j; \end{aligned} \tag{2.3.5}$$

Naravno, ovde podrazumevamo da je $q = 1, \ldots, m+1$ i $l = 1, \ldots n+1$. Izrazi (2.3.5) dobijaju se direktno iz izraza (2.3.3) ako se uzme u obzir struktura matrice A'.

2.4 Algoritam simpleks metoda

Posmatramo linearni program

$$\begin{aligned} \max \quad & z(x) = z(x_{N,1}, \ldots, x_{N,n_1}) = \sum_{i=1}^{n_1} c_i x_{N,i} - d \\ \text{p.o.} \quad & N_i^{(1)}: \quad \sum_{j=1}^{n_1} a_{ij} x_{N,j} \leq b_i, \quad i = 1, \ldots, r \\ & N_i^{(2)}: \quad \sum_{j=1}^{n_1} a_{ij} x_{N,j} \geq b_i, \quad i = r+1, \ldots, s \\ & J_i: \quad \sum_{j=1}^{n_1} a_{ij} x_{N,j} = b_i, \quad i = s+1, \ldots, m \\ & x_{N,j} \geq 0, \quad j = 1, \ldots, n_1. \end{aligned} \tag{2.4.1}$$

Pri tome su a_{ij}, b_i i c_j poznati realni brojevi. Svaka nejednakost oblika $N_i^{(1)}$ (LE ograničenja) se transformiše u odgovarajuću jednačinu dodajući *slack* (*izračunavajuće*) promenljive $x_{B,i}$:

$$N_i^{(1)}: \quad \sum_{j=1}^{n_1} a_{ij} x_{N,j} + x_{B,i} = b_i, \quad i = 1, \ldots, r.$$

Takođe, svaka nejednakost oblika $N_i^{(2)}$ (GE ograničenje) se transformiše u jednakost oduzimanjem *surplus* (*izravnjavajućih*) promenljivih $x_{B,i}$:

$$N_i^{(2)}: \quad \sum_{j=1}^{n_1} a_{ij} x_{N,j} - x_{B,i} = b_i, \quad i = r+1, \ldots, s.$$

Na taj način, dobijamo linearni program u standardnoj formi

$$\begin{array}{ll} \max & c_1x_1+\cdots+c_{n_1}x_{n_1}-d \\ \text{p.o.} & Ax=b, \\ & b=(b_1,\ldots,b_m),\ x=(x_{N,1},\ldots,x_{N,n_1},x_{B,1},\ldots,x_{B,m}) \\ & x_{N,j}\geq 0,\ j=1,\ldots,n_1, \\ & x_{B,i}\geq 0,\ i=1,\ldots,s,\ x_{B,i}=0,\ i=s+1,\ldots,m \end{array} \tag{2.4.2}$$

gde matrica A pripada skupu $\mathbb{R}^{m\times(n_1+s)}$.

U svakoj jednačini odaberemo jednu od promenljivih $x_{N,j}$ za koju važi $a_{p,j}\neq 0$ za bazičnu, i izvršimo odgovarajuće zamene u ostalim jednačinama. Može se koristiti algoritam za zamenu bazične promenljive $x_{B,p}=0$ i nebazične $x_{N,j}$. Posle zamene $n=n_1+s$, kanonička forma problema (2.4.2) može biti zapisana u sledećem tabličnom obliku T_0:

Početna tabela T_0

$x_{N,1}$	$x_{N,2}$	$\ldots$	$x_{N,n}$	-1	
a_{11}^0	a_{12}^0	$\ldots$	a_{1n}^0	b_1^0	=-$x_{B,1}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
a_{m1}^0	a_{m2}^0	$\ldots$	a_{mn}^0	b_m^0	=-$x_{B,m}$
c_1^0	c_2^0	$\ldots$	c_n^0	d^0	$=z^0$

(2.4.3)

gde su $x_{N,1},\ldots,x_{N,n}$ nebazične promenljive i $x_{B,1},\ldots,x_{B,m}$ bazične promenljive. U tabeli (2.4.3) su transformisani koeficijenti matrice A i vektora c označeni sa a_{ij}^0 i c_j^0, respektivno. Tabela (2.4.3) naziva se Tuckerova koja odgovara maksimizaciji funkcije cilja.

Definicija 2.4.1 *Rešenje zadatka linearnog programiranja kod koga su nezavisne promenljive jednake nuli naziva se bazično rešenje.*

Propozicija 2.4.1 *Neka je Tuckerova tabela osnovnog maksimizacionog zadatka linearnog programiranja predstavljena tabelom T_k oblika*

Tabela T_k u k-toj iteraciji.

$x_{N,1}$	$x_{N,2}$	$\ldots$	$x_{N,n}$	-1	
a_{11}^k	a_{12}^k	$\ldots$	a_{1n}^k	b_1^k	=-$x_{B,1}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
a_{m1}^k	a_{m2}^k	$\ldots$	a_{mn}^k	b_m^k	=-$x_{B,m}$
c_1^k	c_2^k	$\ldots$	c_n^k	d^k	$=z^k$

(2.4.4)

Ako je $b_1^k,\ldots,b_m^k\geq 0$, tada u bazično dopustivoj maksimizacionoj tabeli bazično rešenje jeste dopustivo rešenje.

Dokaz. Zaista, stavljajući da su sve nezavisne promenljive jednake nuli, sva glavna ograničenja svode se na jednačine $-b_i^k=-x_{B,i}$ ili $b_i^k=x_{B,i}$. Na taj način, svako rešenje zadovoljava sva ograničenja postavljenog zadatka, pa znači da je dopustivo. □

Sledi algoritam simpleks metoda za maksimizacionu bazično dopustivu tabelu T_k datu u (2.4.4).

Algoritam SimpleksBasicMax

(Simpleks metod za maksimizacionu bazično dopustivu tabelu).

k-ta iteracija simpleks metoda sastoji se od sledećih koraka:

Korak 1. Ako je $c_1^k, \ldots, c_n^k \leq 0$, algoritam staje. Bazično dopustivo rešenje koje odgovara simpleks tablici T_k je optimalno.

Korak 2. Izaberi proizvoljno $c_j^k > 0$. (Može se uzeti maksimalno $c_j^k > 0$. Selekcija prvog $c_j^k > 0$ rešava problem cikliranja.)

Korak 3. Za svako l za koje je $c_l^k > 0$, ispitati da li je $a_{il}^k \geq 0$ za svako $i = 1, \ldots, m$. Ako takvo l postoji algoritam staje. Ciljna funkcija je na dopustivom skupu neograničena odozgo, tj. maksimum je $+\infty$.

Ovaj korak je modifikacija odgovarajućeg koraka iz [84]. Modifikacija se sastoji u tome što se uslov iz Koraka 3 proverava za svako l za koje važi $c_l^k > 0$, a ne samo za $l = j$.

Korak 4. Izračunati

$$\min_{1 \leq i \leq m} \left\{ \frac{b_i^k}{a_{ij}^k}, \quad a_{ij}^k > 0 \right\} = \frac{b_p^k}{a_{pj}^k}$$

i zameniti nebazičnu promenljivu $x_{N,j}$ i bazičnu promenljivu $x_{B,p}$. Simbolički se ova transformacija zapisuje na sledeći način: $x_{N,j} \leftrightarrow x_{B,p}$.

Detaljnije je ovakva transformacija opisana u algoritmu **Replace**.

Algoritam Replace.

(Zamena bazične promenljive $x_{B,p}$ i nebazične promenljive $x_{N,j}$.)
Transformacija vodećeg elementa:

$$a_{pj}^{k+1} = \frac{1}{a_{pj}^k}. \tag{2.4.5}$$

Transformacija elementa u vodećoj vrsti, izuzimajući vodeći element:

$$a_{pl}^{k+1} = \frac{a_{pl}^k}{a_{pj}^k}, \quad l \neq j, \tag{2.4.6}$$

Transformacija elementa vektora b u vodećoj vrsti:

$$b_p^{k+1} = \frac{b_p^k}{a_{pj}^k}, \tag{2.4.7}$$

Transformacija elementa u vodećoj vkoloni, izuzev vodećeg elementa:

$$a_{qj}^{k+1} = -\frac{a_{qj}^k}{a_{pj}^k}, \quad q \neq p, \tag{2.4.8}$$

Transformacija elementa izvan vodeće vrste i vodeće kolone:

$$a_{ql}^{k+1} = a_{ql}^{k} - \frac{a_{pl}^{k} a_{qj}^{k}}{a_{pj}^{k}}, \quad q \neq p,\, l \neq j; \tag{2.4.9}$$

Transformacija elementa vektora b izvan vodeće vrste:

$$b_{q}^{k+1} = b_{q}^{k} - \frac{b_{p}^{k} a_{qj}^{k}}{a_{pj}^{k}}, \quad q \neq p \tag{2.4.10}$$

Transformacija elementa vektora c u vodećoj vrsti:

$$c_{j}^{k+1} = -\frac{c_{j}^{k}}{a_{pj}^{k}}, \tag{2.4.11}$$

Transformacija elementa vektora c izvan vodeće vrste:

$$c_{l}^{k+1} = c_{l}^{k} - \frac{c_{j}^{k} a_{pl}^{k}}{a_{pj}^{k}}, \quad l \neq j; \tag{2.4.12}$$

Transformacija slobodnog člana d:

$$d^{k+1} = d^{k} - \frac{b_{p}^{k} c_{j}^{k}}{a_{pj}^{k}}. \tag{2.4.13}$$

Korak 5. Zameniti k sa $k+1$ i ići na Korak 1.

Razmotrimo *Algoritam Replace.*

Jednačina kojom je izražena bazična promenljiva $x_{B,p}$ je oblika

$$\sum_{l=1}^{n} a_{pl}^{k} x_{N,l} - b_{p}^{k} = -x_{B,p}$$

odakle se dobija posle zamene

$$\sum_{l=1, l\neq j}^{n} \frac{a_{pl}^{k}}{a_{pj}^{k}} x_{N,l} + \frac{1}{a_{pj}^{k}} x_{B,p} - \frac{b_{p}^{k}}{a_{pj}^{k}} = -x_{N,j}. \tag{2.4.14}$$

Odatle se dobijaju jednakosti (2.4.5), (2.4.6) i (2.4.7). Za $q \neq p$ iz

$$\sum_{l=1}^{n} a_{ql}^{k} x_{N,l} - b_{q}^{k} = -x_{B,q}$$

i (2.4.14) dobijamo

$$\sum_{l=1, l\neq j}^{n} a_{ql}^{k} x_{N,l} - a_{qj}^{k} \left(\sum_{l=1, l\neq j}^{n} \frac{a_{pl}^{k}}{a_{pj}^{k}} x_{N,l} + \frac{1}{a_{pj}^{k}} x_{B,p} - \frac{b_{p}^{k}}{a_{pj}^{k}} \right) - b_{p}^{k} = -x_{B,q},$$

odnosno

$$\left(a_{q1}^k - \frac{a_{p1}^k a_{qj}^k}{a_{pj}^k}\right) x_{N,1} + \cdots + \left(a_{q,j-1}^k - \frac{a_{p,j-1}^k a_{qj}^k}{a_{pj}^k}\right) x_{N,j-1}$$
$$-\frac{a_{qj}^k}{a_{pj}^k} x_{B,p}$$
$$+\left(a_{q,j+1}^k - \frac{a_{p,j+1}^k a_{qj}^k}{a_{pj}^k}\right) x_{N,j+1} + \left(a_{qn}^k - \frac{a_{pn}^k a_{qj}^k}{a_{pj}^k}\right) x_{N,n}$$
$$-\left(b_q^k - \frac{b_p^k a_{qj}^k}{a_{pj}^k}\right) = -x_{B,q}.$$

Odatle proizilaze (2.4.8), (2.4.9) i (2.4.10). Konačno, iz

$$z^k = c_1^k x_{N,1} + \cdots + c_n^k x_{N,n} - d^k$$

i (2.4.14) dobijamo

$$z^k = \sum_{l=1, l\neq j}^{n} c_l^k x_{N,l} - c_j^k \left(\sum_{l=1, l\neq j} \frac{a_{pl}^k}{a_{pj}^k} x_{N,l} + \frac{1}{a_{pj}^k} x_{B,p} - \frac{b_p^k}{a_{pj}^k}\right) - d^k,$$

odnosno

$$\begin{aligned} z^k \;=\; & \left(c_1^k - \frac{c_j^k a_{p1}^k}{a_{pj}^k}\right) x_{N,1} + \cdots + \left(c_{j-1}^k - \frac{c_j^k a_{p,j-1}^k}{a_{pj}^k}\right) x_{N,j-1} \\ & -\frac{c_j^k}{a_{pj}^k} x_{B,p} \\ & + \left(c_{j+1}^k - \frac{c_j^k a_{p,j+1}^k}{a_{pj}^k}\right) x_{N,j+1} + \left(c_n^k - \frac{c_j^k a_{pn}^k}{a_{pj}^k}\right) x_{N,n} \\ & - \left(d^k - \frac{b_p^k c_j^k}{a_{pj}^k}\right). \end{aligned}$$

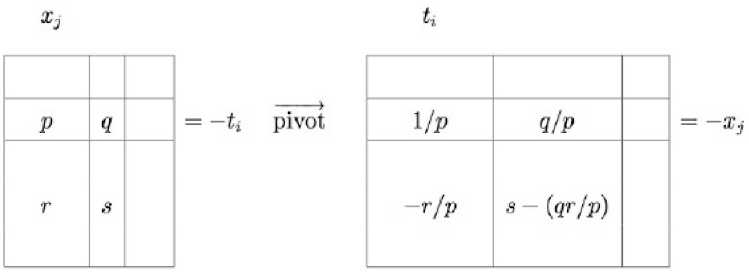
Odatle proizilaze (2.4.11), (2.4.12) i (2.4.13).

Ovakva transformacija je slična Gaussovom postupku eliminacije:

- odabrati $p = a_{ij} \neq 0$;
- razmeniti $x_{N,j}$ i $x_{B,i}$;
- zameniti p sa $1/p$;
- zameniti svaku vrednost q u $i$toj vrsti sa q/p;
- zameniti svaku vrednost r u koloni j sa $-r/p$;

- zameniti druge vrednosti s sa $s-(q\cdot r/p)$.

Ovaj proces se može prikazati sledećom slikom:



Slika 2.4.1. Grafička ilustracija transformacije u simpleks algoritmu

Napomena 2.4.1 *U opštem slučaju indeksi p i j u Koraku 3 nisu jednoznačno određeni, tj. može da postoji više kandidata koji ispunjavaju navedene uslove. Kada su p i j izabrani, oni određuju tzv. stožerni ili pivot element a_{pj} pomoću kojeg se obavljaju transformacije u Koraku 4. Prelazak sa tablice T_k na tablicu T_{k+1} naziva se pivotiranje. Niz elementarnih transformacija u Koraku 4 čini jednu složenu, tzv. stožernu transformaciju. U sledećoj teoremi se dokazuje da pivotiranje ne povećava ciljnu funkciju i da je T_{k+1} takođe simpleks tablica.*

Algoritam simpleks metoda za minimizacionu bazično dopustivu tabelu (2.4.4) sličan je algoritmu za maksimizacionu tabelu:

Algoritam SimpleksBasicMin

(Simpleks metod za minimizacionu bazično dopustivu tabelu, kao u [79, 84]).

k-ta iteracija simpleks metoda sastoji se od sledećih koraka:

Korak 1. Ako je $c_1^k,\dots,c_n^k\geq 0$, algoritam staje. Bazično dopustivo rešenje koje odgovara simpleks tablici T_k je optimalno.

Korak 2. Izaberi proizvoljno $c_j^k<0$. (Može se uzeti minimalno $c_j^k<0$ ili prvo $c_j^k<0$.)

Korak 3. Ispitati da li je $a_{ij}^k\leq 0$ za svako $i=1,\dots,m$. Ako je taj uslov ispunjen, algoritam staje. Ciljna funkcija je na dopustivom skupu neograničena odozdo, tj. minimum je $-\infty$.

Korak 4. Izračunati

$$\min_{1\leq i\leq m}\left\{\frac{b_i^k}{a_{ij}^k},\quad a_{ij}^k>0\right\}=\frac{b_p^k}{a_{pj}^k}$$

i zameniti nebazičnu promenljivu $x_{N,j}$ i bazičnu promenljivu $x_{B,p}$, koristeći algoritam *Replace*.

Korak 5. Zameniti k sa $k+1$ i ići na Korak 1.

Praktično to znači da se na tabeli T_k primenjuju sledeće elementarne transformacije:

- pomnožiti $p-$tu vrstu sa $-\frac{a_{qj}^k}{a_{pj}^k}$ i dodati vrstama $q = 0, \ldots, m$, $q \neq p$;
- podeliti p-tu vrstu sa a_{pj}^k.

Teorema 2.4.1 *Primenom elementarnih transformacija opisanih u Koraku 4 Algoritma SimpleksBasicMin se od simpleks tablice T_k dobija simpleks tablica T_{k+1} koja odgovara ekvivalentnom problemu linearnog programiranja. Pri tom je $d^{k+1} \geq d^k$.*

Dokaz. Na osnovu Leme 2.1.2 tablica T_{k+1} odgovara ekvivalentnom problemu linearnog programiranja. Pri tome iz bazične matrice izlazi kolona koja je imala jedinicu u $p-$toj vrsti a ulazi $j-$ta kolona. Na osnovu pretodnih razmatranja o prelasku na susednu bazu, iz uslova

$$\frac{b_p^k}{a_{pj}^k} = \min\left\{ \frac{b_i^k}{a_{ij}^k} | \, a_{ij}^k > 0 \right\}$$

sledi da je x^{k+1} bazično dopustivo rešenje, tj. $b_i^{k+1} \geq 0$. Dokaz sledi iz

$$b_i^{k+1} = b_i^k - \frac{b_p^k a_{ij}^k}{a_{pj}^k}$$

i

$$\frac{b_p^k}{a_{pj}^k} \leq \frac{b_i^k}{a_{ij}^k}.$$

Iz $c_j^k < 0$ sledi

$$-d^{k+1} = -d^k + \frac{c_j^k}{a_{pj}^k} b_p^k \leq -d^k,$$

tj. $d^{k+1} \geq d^k$. $\square$

Primenimo sada Algoritam *SimpleksBasicMin* na već razmatrani Primer 1.1.1:

Primer 2.4.1 Problemu

$$\begin{array}{ll} \min & -5x_1 - 4x_2, \\ \text{p.o.} & x_1 + x_2 \leq 80, \\ & 3x_1 + x_2 \leq 180, \\ & x_1 + 3x_2 \leq 180, \\ & x_1 \geq 0, \ x_2 \geq 0 \end{array}$$

odgovara simpleks tablice T_0 : (pivot elementi su zbog preglednosti uokvireni)

$T_0 =$

x_1	x_2	-1	
1	1	80	$=$-x_3
$\boxed{3}$	1	180	$=$-x_4
1	3	180	$=$-x_5
-5	-4	0	$=z$

Odgovarajuće bazično dopustivo rešenje je $x^0 = (0, 0, 80, 180, 180)$, a vrednost ciljne funkcije $c^T x^0 = 0$. Očigledno da testovi u koracima 1 i 2 nisu zadovoljeni. Izaberimo $j = 1$. Sada je

$$b_1^0/a_{11}^0 = 80,\ b_2^0/a_{21}^0 = 60,\ b_3^0/a_{31}^0 = 180,$$

odakle određujemo $p = 2$. Znači, potrebno je da se izvrši zamena $x_1 \leftrightarrow x_4$. Primenom Algoritma *Replace* dobija se sledeći niz transformacija kao i odgovarajuća tabela T_1:

$$a_{21}^1 = a_{pj}^1 = \frac{1}{a_{pj}^0} = 1,$$

$$a_{22}^1 = a_{pl}^1 = \frac{a_{pl}^0}{a_{pj}^0} = \frac{a_{22}^0}{a_{21}^0} = \frac{1}{3},$$

$$a_{11}^1 = a_{qj}^1 = -\frac{a_{qj}^0}{a_{pj}^0} = -\frac{a_{11}^0}{a_{21}^0} = -\frac{1}{3},$$

$$a_{31}^1 = a_{qj}^1 = -\frac{a_{qj}^0}{a_{pj}^0} = -\frac{a_{31}^0}{a_{21}^0} = -\frac{1}{3}a_{31}^0 - \frac{a_{21}^0 a_{31}^0}{a_{21}^0} = 1 - \frac{3\cdot 1}{3} = \frac{2}{3},$$

$$a_{12}^1 = a_{ql}^1 = a_{ql}^0 - \frac{a_{pl}^0 a_{qj}^0}{a_{pj}^0} = a_{12}^0 - \frac{a_{22}^0 a_{11}^0}{a_{21}^0} = 1 - \frac{1\cdot 1}{3} = \frac{2}{3},$$

$$a_{32}^1 = a_{ql}^1 = a_{ql}^0 - \frac{a_{pl}^0 a_{qj}^0}{a_{pj}^0} = a_{32}^0 - \frac{a_{22}^0 a_{31}^0}{a_{21}^0} = 3 - \frac{1\cdot 1}{3} = \frac{8}{3}.$$

$$b_1^1 = b_q^1 = b_q^0 - \frac{b_p^0 a_{qj}^0}{a_{pj}^0} = b_1^0 - \frac{b_2^0 a_{11}^0}{a_{21}^0} = 80 - \frac{80\cdot 1}{3} = 20,$$

$$b_2^1 = b_p^1 = \frac{b_p^0}{a_{pj}^0} = \frac{b_2^0}{a_{21}^0} = \frac{180}{3} = 60,$$

$$b_3^1 = b_q^1 = b_q^0 - \frac{b_p^0 a_{qj}^0}{a_{pj}^0} = b_3^0 - \frac{b_2^0 a_{31}^0}{a_{21}^0} = 180 - \frac{180\cdot 1}{3} = 120.$$

$$c_1^1 = c_j^1 = -\frac{c_j^0}{a_{pj}^0} = -\frac{c_1^0}{a_{21}^0} = -\frac{-5}{3} = \frac{5}{3},$$

$$c_2^1 = c_l^1 = c_l^0 - \frac{c_j^0 a_{pl}^0}{a_{pj}^0} = c_2^0 - \frac{c_1^0 a_{22}^0}{a_{21}^0} = -4 - \frac{-5\cdot 1}{3} = -\frac{7}{3},$$

$$d^1 = d^0 - \frac{b_2^0 c_1^0}{a_{21}^0} = 0 - \frac{180\cdot(-5)}{3} = -300.$$

$T_1 =$

x_4	x_2	-1	
$-\frac{1}{3}$	$\frac{2}{3}$	20	$=-x_3$
$\frac{1}{3}$	$\frac{1}{3}$	60	$=-x_1$
$-\frac{1}{3}$	$\frac{8}{3}$	120	$=-x_5$
$\frac{5}{3}$	$-\frac{7}{3}$	300	$=z$

Sada je bazično dopustivo rešenje $x^1 = (60, 0, 20, 0, 120)$, kome odgovara vrednost ciljne funkcije $c^T x^1 = -300$. Testovi u Koracima 1 i 2 nisu zadovoljeni. Sada $j = 2$ i

$$b_1^1/a_{12}^1 = 30,\ b_2^1/a_{22}^1 = 180,\ b_3^1/a_{32}^1 = 45,$$

odakle proizilazi $p = 1$. Znači, potrebno je da se izvrši zamena $x_2 \leftrightarrow x_3$. Primenom Algoritma

Replace dobija se sledeći niz transformacija kao i odgovarajuća tabela T_2:

$$a_{12}^2 = a_{pj}^2 = 1,$$

$$a_{11}^2 = a_{pl}^2 = \frac{a_{pl}^1}{a_{pj}^1} = \frac{a_{11}^1}{a_{12}^1} = -\frac{1}{2},$$

$$a_{22}^2 = a_{qj}^2 = -\frac{a_{qj}^1}{a_{pj}^1} = -\frac{a_{22}^1}{a_{12}^1} = -\frac{1}{2},$$

$$a_{32}^2 = a_{qj}^2 = -\frac{a_{qj}^1}{a_{pj}^1} = -\frac{a_{32}^1}{a_{12}^1} = -4,$$

$$a_{21}^2 = a_{ql}^2 = a_{ql}^1 - \frac{a_{pl}^1 a_{qj}^1}{a_{pj}^1} = a_{21}^1 - \frac{a_{11}^1 a_{22}^1}{a_{12}^1} = \frac{7}{6},$$

$$a_{31}^2 = a_{ql}^2 = a_{ql}^1 - \frac{a_{pl}^1 a_{qj}^1}{a_{pj}^1} = a_{31}^1 - \frac{a_{11}^1 a_{32}^1}{a_{12}^1} = 1.$$

$$b_1^2 = b_p^2 = \frac{b_p^1}{a_{pj}^1} = \frac{b_1^1}{a_{12}^1} = \frac{20}{2/3} = 30,$$

$$b_2^2 = b_q^2 = b_q^1 - \frac{b_p^1 a_{qj}^1}{a_{pj}^1} = b_2^1 - \frac{b_1^1 a_{22}^1}{a_{12}^1} = 60 - \frac{20 \cdot 1/3}{2/3} = 50,$$

$$b_3^2 = b_q^2 = b_q^1 - \frac{b_p^1 a_{qj}^1}{a_{pj}^1} = b_3^1 - \frac{b_1^1 a_{32}^1}{a_{12}^1} = 120 - \frac{20 \cdot 8/3}{2/3} = 40.$$

$$c_2^2 = c_j^2 = -\frac{c_j^1}{a_{pj}^1} = -\frac{c_2^1}{a_{11}^1} = \frac{7}{2},$$

$$c_1^2 = c_l^2 = c_l^1 - \frac{c_j^1 a_{pl}^1}{a_{pj}^1} = c_1^1 - \frac{c_2^1 a_{11}^1}{a_{12}^1} = \frac{5}{3} - \frac{-7/3 \cdot (-1/3)}{2/3} = \frac{1}{2},$$

$$d^2 = d^1 - \frac{b_1^1 c_2^1}{a_{12}^1} = 300 - \frac{20 \cdot (-7/3)}{2/3} = 370.$$

$T_2 =$

x_4	x_3	-1	
$-\frac{1}{2}$	$\frac{3}{2}$	30	$=-x_2$
$\frac{1}{2}$	$-\frac{1}{2}$	50	$=-x_1$
1	-4	40	$=-x_5$
$\frac{1}{2}$	$\frac{7}{2}$	370	$=z$

Bazično dopustivo rešenje je $x^2 = (50, 30, 0, 0, 40)$, a ciljna funkcija uzima vrednost $c^T x^2 = -370$. Kako je test u Koraku 1 zadovoljen, to je x^2 optimalno rešenje, a vrednost ciljne funkcije je $z = -370$.

Primetimo da svakoj simpleks tablici odgovara jedna ekstremna tačka skupa Ω_P, tako da simpleks metod predstavlja "šetnju" po temenima skupa Ω_P. Kako je $d^{k+1} < d^k$, to se nijedan vrh ne može ponoviti. Kako je broj vrhova konačan, sledi da posle konačno mnogo iteracija dolazimo ili do optimalnog rešenja ili do zaključka da ciljna funkcija nije ograničena odozdo.

Napomena 2.4.2 *Može se dogoditi više od jedne vrednosti p za koju je razlomak $\frac{b_p^k}{a_{pj}^k}$ najmanji. Tada se može izabrati proizvoljno p. U sledećim poglavljima reći ćemo nešto više o ovome.*

Teorema 2.4.2 *Ako je* $c_1^k, \ldots, c_n^k \leq 0$ *ciljna funkcija ima ekstremum u tački* $x_{N,i} = 0$, $i = 1, \ldots, n$.

Dokaz. Neka su $x_{N,1}, \ldots, x_{N,n}$ proizvoljni nenegativni brojevi. Tada važi

$$z(x) = \sum_{i=1}^{n} c_i^k x_{N,i} - d^k \leq -d^k$$

jer je $c_i^k x_{N,i} \leq 0$ za svako $i = 1, \ldots, n$. Jednakost važi ako i samo ako je $x_{N,i} = 0$, $i = 1, \ldots, N$. □

Teorema 2.4.3 *Neka je za neko* $j \in \{1, \ldots, n\}$ *ispunjeno* $c_j^k > 0$ *i* $a_{ij}^k \leq 0$ *za svako* $i = 1, \ldots, m$, *tada je ciljna funkcija neograničena.*

Dokaz. Ako je $a_{ij}^k \leq 0$ za svako $i = 1, \ldots, m$, s obzirom na ograničenja

$$a_{i1}^k x_{N,1} + \cdots + a_{ij}^k x_{N,j} + \cdots + a_{in}^k x_{N,N} - b_i^k = -x_{B,i},\ i = 1, \ldots, m,$$

promenljiva $x_{N,j}$ se može uvećati do beskonačnosti, a da nijedna zavisna promenljiva $x_{B,i}$ ne bude negativna. Kako je $c_j^k > 0$, to se sa povećanjem promenljive x_j povećava i vrednost ciljne funkcije. To znači da je ciljna funkcija neograničena. □

U opštem slučaju je $c_j^k > 0$ za neko j i $a_{ij}^k > 0$ za neko i. Pokazaćemo da tabela (2.4.4) može da se transformiše u ekvivalentnu, pri čemu su jedna nezavisna i zavisna promenljiva zamenile mesta. Izaberimo p tako da bude:

$$\frac{b_p^k}{a_{pj}^k} = \min_{1 \leq i \leq m} \left\{ \frac{b_i^k}{a_{ij}^k} |\ a_{ij}^k > 0 \right\}. \tag{2.4.15}$$

Promenljivu $x_{N,j}$ možemo zameniti iz jednačine

$$x_{B,p} = b_p^k - (a_{p1}^k x_{N,1} + \cdots + a_{pj}^k x_{N,j} + \cdots + a_{pn}^k x_{N,n})$$

i zameniti je u svim ostalim jednačinama i u funkciji cilja. Na ovaj način dobijamo sistem ekvivalentan sistemu (2.4.2) pri čemu je sada $x_{B,p}$ nezavisna a $x_{N,j}$ zavisna promenljiva. I dalje su slobodni članovi b_i u svim jednačinama pozitivni dok c_j^k postaje nepozitivno. Dokaz ovog tvrđenja (sledeća teorema) kao i algoritam za transformaciju sistema jednačina (2.4.2) (za zamenu promenljivih $x_{B,p}$ i $x_{N,j}$) biće pokazani u nastavku. Sada primenjujemo isti postupak sve dok je i jedan od brojeva c_j pozitivan. Kada su svi c_j^k nepozitivni, dobijeno dopustivo rešenje $x_{N,1} = \cdots = x_{N,n} = 0$ je optimalno, tj ciljna funkcija dostiže ekstremum.

Teorema 2.4.4 *Ako je* $b_1^k, \ldots, b_m^k \geq 0$ *i ako je indeks* p *odabran prema kriterijumu* (3.3.15) *i* $c_j^k > 0$, *tada i posle zamene mesta promenljivih* $x_{B,p}$ *i* $x_{N,j}$ *važi* $b_1^{k+1}, \ldots, b_n^{k+1} \geq 0$. *Takođe je i* $c_j^{k+1} < 0$.

Dokaz. Prema algoritmu za zamenu promenljivih $x_{B,p}$ i $x_{N,j}$ posle transformacije je

$$b_q^{k+1} = \frac{a_{pj}^k b_q^k - a_{qj}^k b_p^k}{a_{pj}^k}.$$

Ako je $a_{qj}^k \geq 0$, s obzirom na $\frac{b_p^k}{a_{pj}^k} \leq \frac{b_q^k}{a_{qj}^k}$ dobijamo

$$b_q^{k+1} = \frac{a_{pj}^k b_q^k - a_{qj}^k b_p^k}{a_{pj}^k} \geq \frac{a_{pj}^k b_q^k - a_{pj}^k b_q^k}{a_{pj}^k} = 0.$$

Pretpostavimo sada da je $a_{qj}^k < 0$. S obzirom na $-\frac{a_{qj}^k b_p^k}{a_{pj}^k} > 0$, dobijamo

$$b_q^{k+1} = \frac{a_{pj}^k b_q^k - a_{qj}^k b_p^k}{a_{pj}^k} > b_q^k > 0.$$

Posle smene dobijamo

$$c_j^{k+1} = -\frac{c_j^k}{a_{pj}^k} < 0,$$

čime je teorema dokazana. □

Element a_{pj} nazvaćemo ključnim (pivot) elementom a zamenu promenljivih $x_{B,p}$ i $x_{N,j}$ nazvaćemo zamenom pomoću ključnog elementa a_{pj}^k.

Primer 2.4.2 Rešiti sledeći linearni problem:

$$\begin{array}{ll} \max & L = 7x_1 + 5x_2 \\ \text{p.o.} & 2x_1 + 3x_2 + x_3 = 19 \\ & 2x_1 + x_2 + x_4 = 13 \\ & 3x_2 + x_5 = 15 \\ & 3x_1 + x_6 = 18. \end{array}$$

Rešenje. Rang matrice sistema i proširene matrice

$$\left[\begin{array}{ccccccc} 2 & 3 & 1 & 0 & 0 & 0 & 19 \\ 2 & 1 & 0 & 1 & 0 & 0 & 13 \\ 0 & 3 & 0 & 0 & 1 & 0 & 15 \\ 3 & 0 & 0 & 0 & 0 & 1 & 18 \end{array}\right]$$

jednak je 4. To znači da možemo uzeti 4 promenljive za bazne (na primer x_3, x_4, x_5, x_6), a dve promenljive (x_1, x_2) za nezavisne (slobodne).

$$\begin{aligned} x_3 &= 19 - 2x_1 - 3x_2 \\ x_4 &= 13 - 2x_1 - x_2 \\ x_5 &= 15 - 3x_2 \\ x_6 &= 18 - 3x_1 \end{aligned}$$

Osim toga, na osnovu ciljne funkcije zaključujemo

$$L - 7x_1 - 5x_2 = 0.$$

Početna simpleks tabela ima oblik

$x_{N,1}$	$x_{N,2}$	$x_{B,1}$	$x_{B,2}$	$x_{B,3}$	$x_{B,4}$	b_i	$x_{B,i}$
x_1	x_2	x_3	x_4	x_5	x_6		
2	3	1	0	0	0	19	x_3
2	1	0	1	0	0	13	x_4
0	3	0	0	1	0	15	x_5
3	0	0	0	0	1	18	x_6
-7	-5	0	0	0	0	0	L

Tabela 2.4.1.

U poslednjoj vrsti su negativni koeficijenti –7 i –5. Uzmimo najmanji negativni koeficijent, –7. Zatim u koloni za x_1 uočimo tri pozitivna elementa: 2,2,3. Podelimo tim brojevima odgovarajuće slobodne članove. Minimum tih količnika je

$$\min\left\{\frac{19}{2}, \frac{13}{2}, \frac{18}{3}\right\} = \frac{18}{3}.$$

U preseku vrste za x_6 i kolone za x_1 nalazi se broj 3. Znači, promenljive x_1 i x_6 zamenjuju uloge. Novu bazu čine x_3, x_4, x_5, x_1, dok su nezavisne promenljive x_2, x_6. Da bismo na mestu preseka dobili 1, podelićemo posmatranu vrstu brojem 3. Ostalim vrstama dodajemo podeljenu vrstu, prethodno umnoženu brojem tako da u koloni koja odgovara x_2, ispod i iznad vodećeg elementa dobijemo nule.

$x_{B,4}$	$x_{N,2}$	$x_{B,1}$	$x_{B,2}$	$x_{B,3}$	$x_{N,1}$	b_i	$x_{B,i}$
x_1	x_2	x_3	x_4	x_5	x_6		
0	3	1	0	0	-2/3	7	x_3
0	1	0	1	0	-2/3	1	x_4
0	3	0	0	1	0	15	x_5
1	0	0	0	0	1/3	6	x_1
0	-5	0	0	0	7/3	42	L

Tabela 2.4.2.

Isti postupak se nastavlja na novoj tabeli. U poslednjoj vrsti negativan je koeficijent –5. U koloni za x_2 uočimo dva pozitivna elementa: 3,1,3. Podelimo tim brojevima odgovarajuće slobodne članove. Minimum tih količnika je

$$\min\left\{\frac{7}{3}, \frac{1}{1}, \frac{15}{3}\right\} = \frac{1}{1}.$$

U preseku vrste za x_4 i kolone za x_2 nalazi se broj 1. Sada promenljive x_2 i x_4 zamenjuju uloge. Novu bazu čine x_3, x_2, x_5, x_1, dok su nezavisne promenljive x_4, x_6. Ostalim vrstama dodajemo drugu vrstu, prethodno umnoženu brojem tako da u koloni x_2 ispod i iznad vodećeg elementa dobijemo nule.

$x_{B,4}$	$x_{B,2}$	$x_{B,1}$	$x_{N,2}$	$x_{B,3}$	$x_{N,1}$	b_i	$x_{B,i}$
x_1	x_2	x_3	x_4	x_5	x_6		
0	0	1	-3	0	4/3	4	x_3
0	1	0	1	0	-2/3	1	x_2
0	0	0	-3	1	2	12	x_5
1	0	0	0	0	1/3	6	x_1
0	0	0	5	0	-1	47	L

Tabela 2.4.3.

Postupak se ponavlja na poslednjoj tabeli.

$$\min\left\{\frac{4}{4/3}, \frac{12}{2}, \frac{6}{1/3}\right\} = \frac{4}{4/3}$$

Sada promenljive x_6 i x_3 zamenjuju uloge. Novu bazu čine x_6, x_2, x_5, x_1, dok su nezavisne promenljive x_3, x_4.

$x_{B,4}$	$x_{B,2}$	$x_{N,1}$	$x_{N,2}$	$x_{B,2}$	$x_{B,4}$	b_i	$x_{B,i}$
x_1	x_2	x_3	x_4	x_5	x_6		
0	0	3/4	-9/4	0	1	3	x_6
0	1	1/2	-1/2	0	0	3	x_2
0	0	-2/3	3/2	1	0	6	x_5
1	0	-1/4	3/4	0	0	5	x_1
0	0	3/4	11/4	0	0	50	L

Tabela 2.4.4.

Nova vrsta za L nema negativnih koeficijenata. Maksimalna vrednost ciljne funkcije je $L_{max}=50$, a dobija se za bazu $\{5, 3, 0, 0, 6, 3\}$.

Napomena 2.4.3 *Ukoliko u u nekoj iteraciji postoji više jednakih minimalnih količnika b_i/a_{ip}, potrebno je izabrati onu vrstu za koju je odnos elemenata sledeće kolone u odnosu na elemente one kolone koju koristimo za rešavanje najmanji. Ovaj postupak se ponavlja dok ne dođemo do jednoznačnog minimalnog elementa b_j/a_{jp}.*

Još jedan oblik simpleks tabela

Simpleks model problema linearnog programiranja tipa maksimuma je sistem nejednačina oblika:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m &\leq b_1 \\ a_{21}x_2 + a_{22}x_2 + \cdots + a_{2m}x_m &\leq b_2 \\ &\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m &\leq b_n \end{aligned}$$

Ovaj sistem nejednačina prevodimo u sistem jednačina, uvođenjem dopunskih promenljivih, na sledeći način:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m + x_{m+1} &= b_1 \\ a_{21}x_2 + a_{22}x_2 + \cdots + a_{2m}x_m + x_{m+2} &= b_2 \\ &\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m + x_{m+n} &= b_n \end{aligned}$$

Funkcija kriterijuma ima oblik:

$$F(x) = \sum_{j=1}^{n} c_j x_j + 0 \cdot (c_{n+1}x_{n+1} + \cdots + c_{n+m}x_{n+m}) .$$

Na osnovu ovakvog modela, postavljamo nultu simpleks tabelu oblika:

max ST - 0

Bazne promenljive			Slobodne Promenljive								
C			c_1	c_2	...	c_n	0	0	0	0	b_j/a_{ij}
C_b	X_b	B	X_1	X_2	...	X_n	X_{n+1}	X_{n+2}	...	X_{n+m}	
0	x_{n+1}	b_1	a_{11}	a_{12}	...	a_{1n}	1	0	...	0	
0	x_{n+2}	b_2	a_{21}	a_{22}	...	a_{2n}	0	1	...	0	
...	...	...	...	...	...	...	0	0	...	0	
0	x_{n+m}	b_m	a_{m1}	a_{m2}	...	a_{mn}	0	0	...	1	
$F_j - c_j$		0	F_1-c_1	F_2-c_2		F_n-c_n	0	0	0	0	$\theta(c_j - F_j)$

gde su:

C - vektor koeficijenata c_i uz promenljive x_i funkcije kriterijuma, $i = 1, \ldots, n$.

C_b - vektor koeficijenata u funkciji kriterijuma uz promenljive koje sačinjavaju bazično dopustivo rešenje. Kod max ST - 0 vrednost ovih koeficijenata je 0,

X_b - vektor promenljivih bazično dopustivog rešenja.

B - vektor vrednosti promenljivih bazno dopustivog rešenja za posmatranu iteraciju,

X_j - množitelji vektora baze.

$F_j - c_j$ - kriterijum optimalnosti. Na osnovu ovog kriterijuma se odlučuje da li je dobijeno rešenje maksimalno. Rešenje je maksimalno ako je ispunjen uslov

$$(\forall j)\ F_j - c_j \geq 0.$$

U slučaju da rešenje nije optimalno, ovim kriterijumom se određuje vektor koji ulazi u bazu, a to je vektor x_j koji ispunjava uslov:

$$\min_j \{F_j - c_j |\ F_j - c_j < 0\}.$$

S druge stane, na osnovu "teta" kriterijuma se određuje koji vektor izlazi iz baze, tj:

$$\theta = \min_i \left\{ \frac{b_i}{a_{ij}}, \quad a_{ij}^k > 0 \right\} = \frac{b_p}{a_{pj}}.$$

Procedura za nalaženje optimalnog rešenja simpleks metodom

Posmatra se opšti problem linearnog programiranja. Potrebno je da se pronađu nenegativne vrednosti promenljivih $x_1, \ldots, x_k$ koje će ispunjavati sistem ograničenja dat linearnim nejednačinama i jednačinama 1.1.1, i obezbediti maksimalnu vrednost kriterijumske funkcije

$$z_0 = \sum_{j=1}^{k} c_j x_j.$$

Ovako formulisani problem linearnog programiranja rešavamo simpleks metodom prema sledećoj proceduri:

1) Sve nejednačine sistema prevodimo u jednačine. Ako je nejednačina tipa "$\leq$", onda dodajemo odgovarajuću promenljivu; ako je nejednačina tipa "$\geq$", onda odgovarajuću izravnavajuću promenljivu oduzimamo od leve strane nejednačine. Uz izravnavajuće promenljive u funkciji kriterijuma uvek stoje nule.

Veštačke promenljive se dodaju svim jednačinama i nejednačinama tipa "$\geq$". Uz ove promenljive u funkciji kriterijuma stavljamo koeficijente M.

2) Od koeficijenata prilagođenog modela sastavljamo početnu simpleks tabelu. Bazu vektorskog prostora za početno rešenje čine jedinični vektori. To su vektori koji odgovaraju "dodatim" promenljivim (dodate izravnavajuće i sve veštačke).

3) Za dodatni red simpleks tabele računamo koeficijente $F_j - c_j$ za svako j. Simpleks tabela sadrži početno rešenje.

4) Proverimo koeficijente reda $(F_j - c_j)$:
– Ako je $F_j - c_j \geq 0$ za svako j, onda tvrdimo da je pronađeno optimalno rešenje.
– Ako postoji $F_j - c_j < 0$, onda biramo

$$F_s - c_s = \left[\min_j (F_j - c_j)\right] < 0 \ .$$

Koeficijent $(F_s - c_s)$ odgovara vektoru A_s, pa određujemo da vektor A_s uđe u novu bazu vektora.

5) Koordinate vektora B delimo odgovarajućim pozitivnim koordinatama vektora A_s i određujemo

$$\theta = \frac{b_r}{a_{rs}} = \min_i \frac{b_i}{a_{is}} \quad (a_{is} > 0) \ .$$

Najmanja vrednost od ovih količnika odgovara vektoru A_r, pa određujemo da vektor A_r treba da izađe iz početne baze.

6) Koeficijente za novu simpleks tabelu izračunavamo na osnovu koeficijenata iz početne simpleks tabele prema opisanim pravilima transformacije.

7) Vraćamo se na tačku 4) iz ove procedure. Svaka naredna iteracija počinje tačkom 4) i ponavlja postupak do tačke 7), sve dok se u tački 4) ne utvrdi da su sve razlike $F_j - c_j \geq 0$. Tada se postupak završava, pronađeno je optimalno rešenje linearnog problema.

Primer 2.4.3 Rešiti sistem nejednačina primenom simpleks metode, ako su date sledeća funkcija kriterijuma i ograničenja:

$$\begin{array}{ll} \max & F(X) = 10x_1 + 12x_2 + 10x_3 \\ \text{p.o.} & 4x_1 + 5x_2 + 4x_3 \leq 4200 \\ & 2x_1 = 2x_2 + x_3 \leq 1500 \\ & 2x_1 + 3x_2 + 4x_3 \leq 2400 \end{array}$$

Na osnovu ovog modela formiraćemo polaznu simpleks tabelu, max ST-0.

$$\begin{aligned} \max \quad & F(X) = 10x_1 + 12x_2 + 10x_3 \\ \text{p.o.} \quad & 4x_1 + 5x_2 + 4x_3 + x_4 = 4200 \\ & 2x_1 = 2x_2 + x_3 + x_5 = 1500 \\ & 2x_1 + 3x_2 + 4x_3 + x_6 = 2400 \end{aligned}$$

max ST - 0

C			10	12	10	0	0	0	b_j / a_{i2}
C_b	X_b	B	X_1	X_2	X_3	X_4	X_5	X_6	
0	x_4	4200	4	5	4	1	0	0	840
0	x_5	1500	2	2	1	0	1	0	750
0	x_6	2400	2	3	4	0	0	1	800
$F_j - c_j$		0	-10	-12	-10	0	0	0	9000

max ST - 1

C			10	12	10	0	0	0	b_j / a_{i2}
C_b	X_b	B	X_1	X_2	X_3	X_4	X_5	X_6	
0	x_4	450	-1	0	1.5	1	-2.5	0	300
12	x_2	750	1	1	0.5	0	0.5	0	1500
0	x_6	150	-1	0	2.5	0	-1.5	1	60
$F_j - c_j$		9000	2	0	-4	0	6	0	240

max ST - 2

C			10	12	10	0	0	0
C_b	X_b	B	X_1	X_2	X_3	X_4	X_5	X_6
0	x_4	360	-0.4	0	0	1	-1.6	-0.6
12	x_2	720	1.2	1	0	0	0.8	-0.2
10	x_3	60	-0.4	0	1	0	-0.6	0.4
$F_j - c_j$		9240	0.4	0	0	0	3.6	1.6

Ova tabela sadrži optimalno rešenje, koje iznosi:

$$X^* = [0, 720, 60, 360, 0, 0]^T, \quad F(X^*) = F_{\max} = 9240.$$

2.5 Određivanje početnog bazično dopustivog rešenja

Primena Algoritma *SimpleksbasicMax* podrazumeva da je poznato jedno bazično dopustivo rešenje. To je ispunjeno ako su sva ograničenja tipa $\leq$ i ako je $b_i \geq 0$, $i = 1, \ldots, m$, odnosno ako su sva ograničenja tipa $\geq$ i ako je $b_i \geq 0$, $i = 1, \ldots, m$. Dodavanjem izravnavajućih promenljivih odmah formiramo i bazično dopustivo rešenje. Međutim, u opštem slučaju bazično dopustivo rešenje nije poznato. Dokazaćemo teoremu na osnovu koje je moguće odrediti bazično dopustivo rešenje ukoliko je skup dopustivih rešenja problema linearog programiranja neprazan.

Do sada je opisan algoritam (Algoritam *SimpleksBasicMax*/*SimpleksBasicMin*) za nalaženje optimalnog rešenja, ako postoji, u slučajevima kada se u zadacima

osnovnog oblika traži maksimum/minimum funkcije cilja i kada su početne maksimizacione/minimizacione tabele bazično dopustive. Šta učiniti ako početna tabela osnovnog zadatka linearnog programiranja nije bazično dopustiva? Pre nego što možemo da primenimo simpleks algoritam moramo najpre tabelu maksimizacije da transformišemo na maksimizacionu tabelu sa bazično dopustivim rešenjem. Tako je u opisanom algoritmu potrebno prethodno uvesti korake ovakve transformacije. Razmotrimo na kraju simpleks algoritam za maksimizacionu tabelu u opštem slučaju.

Definicija 2.5.1 *Osnovni zadatak linearnog programiranja sa maksimizacijom je nedopustiv ako nema dopustiva rešenja.*

Teorema 2.5.1 *Ako je* $a_{i1}^k, \ldots, a_{in}^k \geq 0$ *i* $b_i^k < 0$ *tada i-ta jednačina u sistemu* (2.3.2) *nema rešenja, tj. sistem je nedopustiv.*

Dokaz. Uz navedene pretpostavke dobijamo

$$-x_{B,i} = a_{i1}^k x_{N,1} + \cdots a_{in}^k x_{B,N} - b_i^k \geq -b_i^k > 0,$$

odnosno $x_{B,i} < 0$, što predstavlja kontradikciju sa polaznom predpostavkom da su sve promenljive nenegativne. □

U ovom odeljku razmotrićemo problem pronalaženja prvog bazično dopustivog rešenja (kanonskog oblika). Najčešće su u upotrebi dve vrste metoda kojima se rešava ovaj problem. Jedna grupa metoda se nazivaju *dvofazni simpleks metodi* (two-phase simplex method), a drugu vrstu čine *BigM metodi* (BigM methods). Ovde su opisana tri pristupa. Dva pristupa su tzv. dvofazni simpleks metodi. Jedan metod zahteva uvođenje veštačkih promenljivih i samim tim povećava dimenzije problema, dok drugi ne zahteva primenu veštačkih promenljivih. Treći metod kombinuje ove dve faze u jednu, i naziva se *BigM* metod.

2.6 Dvofazni simpleks metodi

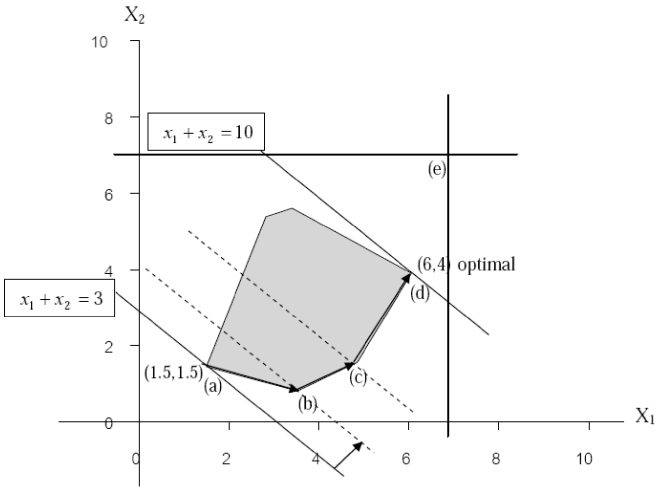
Dvofazni simpleks metod se sastoji iz dve faze, faze I i faze II. Faza I pokušava da pronađe neko početno bazično dopustivo rešenje (initial basic feasible solution). Kada je početno bazično dopustivo rešenje pronađeno, tada se primenjuje faza II da bi se pronašlo optimalno rešenje. Simpleks metod je iterativna procedura čija se svaka iteracija karakteriše određivanjem m bazičnih promenljivih $x_{B,1}, \ldots, x_{B,m}$ i n nebazičnih promenljivih $x_{N,1}, \ldots, x_{N,n}$.

Geometrijski, simpleks metod se kreće iz jedne ekstremne tačke (ugla) skupa dopustivih rešenja u drugu i pri tome poboljšava vrednosti ciljne funkcije u svakoj iteraciji. Dvofazni simpleks metod prolazi kroz dve faze, faza I i faza II. Faza I pokušava sa nađe inicijalnu ekstremnu tačku. Kada se inicijalna ekstremna tačka jedanput pronađe, primenjuje se faza II da bi se rešio originalni LP.

Primer 2.6.1 Za sledeći problem

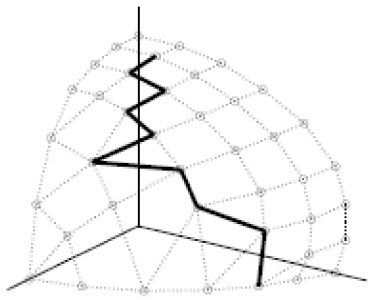
$$\begin{aligned} \min \quad & -x_1 - x_2 \\ \text{p.o.} \quad & 2x_1 + 3x_2 \leq 24,\ 2x_1 - x_2 \leq 8,\ x_1 - 2x_2 \leq 2, \\ & -x_1 + 2x_2 \leq 8,\ x_1 + 3x_2 \geq 6, 3x_1 - x_2 \geq 3, \\ & 0 \leq x_1 \leq 7,\ 0 \leq x_2 \leq 7 \end{aligned}$$

faza I simpleks algoritma se završava u ekstremnoj tački koja je označena sa (a) na sledećoj slici. Tada faza II sledi sekvencu ekstremnih tačaka koje su označene strelicama duž ivica skupa dopustivih rešenja. Optimalna ekstremna tačka je označena sa (d).



Slika 2.6.1. Geometrija simpleks metoda

Na sledećoj slici je prikazan trodimenzionalni slučaj.



Slika 2.6.2. Temena i simpleks metod u R^3

Ako je $\mathbf{b} \geq 0$ i ako su sve nebazične promenljive $x_{N,1}, \ldots, x_{N,n}$ jednake nuli, tada je $x_{B,1} = b_1, \ldots, x_{B,m} = b_m$ bazično dopustivo rešenje. Ako uslov $\mathbf{b} \geq 0$ nije ispunjen, neophodno je da se nađe početno bazično dopustivo rešenje ili da se odredi da ono ne postoji. Postoji više strategija za fazu I.

Dvofazni simpleks metod koji koristi veštačke promenljive

Klasični pristup se sastoji u tome da se linearnom programu u standardnom obliku pridruži tzv. *prošireni problem* [5, 17, 33].

Neka je dat problem linearog programiranja u standardnom obliku

$$\begin{array}{ll} \min & c^T x, \\ \text{p.o.} & Ax = b, \\ & x \geq 0. \end{array} \tag{2.6.1}$$

Jasno je da bez umanjenja opštosti možemo pretpostaviti da je u standardnom obliku $b \geq 0$ (u suprotnom pomnožimo odgovarajuće jednačine sa -1). Problemu (2.6.1) pridružimo pomoćni problem linearnog programiranja

$$\begin{array}{ll} \min & e^T w, \\ \text{p.o.} & Ax + w = b, \\ & x \geq 0,\ w \geq 0, \end{array} \tag{2.6.2}$$

gde je $e = (1, \ldots, 1) \in \mathbb{R}^m$ i $w \in \mathbb{R}^m$ je vektor tzv. *veštačkih* promenljivih. Važna činjenica je da je skup dopustivih rešenja problema (2.6.2) neprazan jer mu sigurno pripada tačka ($x = 0$, $w = b$). Jasno je takođe da je ciljna funkcija na tom skupu odozdo ograničena sa nulom. Dopustivu bazu čine kolone koje odgovaraju promenljivim $w_1, \ldots, w_m$, a kanonski oblik problema (2.6.2) se dobija eliminacijom w iz ciljne funkcije pomoću jednačine $w = b - Ax$. Problemi (2.6.1) i (2.6.2) su povezani sledećom teoremom:

Teorema 2.6.1 *Skup dopustivih rešenja problema* (2.6.1) *je neprazan ako i samo ako je optimalna vrednost ciljne funkcije problema* (2.6.2) *jednaka nuli.*

Dokaz. Neka je $\bar{x}$ dopustivo rešenje problema (2.6.1). Tada je $(\bar{x}, 0)$ dopustivo rešenje problema (2.6.2), pri čemu je vrednost ciljne funkcije jednaka nuli. S obzirom da je nula donja granica za ciljnu funkciju problema (2.6.2), sledi da je $(\bar{x}, 0)$ optimalno rešenje i da je nula optimalna vrednost ciljne funkcije tog problema.

Pretpostavimo sada da je $(\bar{x}, \bar{w})$ optimalno rešenje problema (2.6.2) i neka je $e^T \bar{w} = 0$. Iz $e > 0$, $\bar{w} \geq 0$ sledi da je $\bar{w} = 0$, pa je $A\bar{x} = b$, tj. $\bar{x}$ je dopustivo rešenje problema (2.6.2). □

Na prethodnoj teoremi se zasniva tzv. *dvofazna modifikacija simpleks metoda.*

Algoritam 2. (Dvofazna modifikacija simpleks metoda).

I Faza:

Za problem (2.6.1) se formira problem (2.6.2) kome se pridružuje LP-tablica. Eliminacijom jedinica iz nulte vrste, koje odgovaraju veštačkim promenljivim, LP-tablica se svodi na simpleks tablicu, a zatim se primenjuje Algoritam *SimpleksBasic*. Kako je ciljna funkcija ograničena odozdo, moguća su dva slučaja:

1) Optimalna vrednost je veća od nule. Po Teoremi 2.6.1 problem (2.6.1) nema dopustivih rešenja i postupak se završava.

2) Optimalna vrednost je nula. Tada su u optimalnom bazično dopustivom rešenju sve veštačke promenljive jednake nuli. Prelazi se na drugu fazu.

II Faza:

Korak 1. Iz poslednje simpleks tablice uklanjaju se sve nebazične kolone koje odgovaraju veštačkim promenljivim, nulta vrsta se zamenjuje vrstom $[0|\, c_1 \ldots c_n\, 0 \ldots 0]$ koja ima $n + k + 1$ element, pri čemu je k broj bazičnih veštačkih promenljivih. Dobijena LP-tablica se svodi na simpleks tablicu eliminacijom onih $c_j \neq 0$ koji odgovaraju bazičnim promenljivim.

Korak 2. Ukoliko je $k = 0$ ići na Korak 3. Ako je $k > 0$ u simpleks tablici postoje bazične kolone koje odgovaraju veštačkim promenljivim. Uočimo bazičnu kolonu koja odgovara veštačkoj promenljivoj w_s. Neka ta kolona sadrži jedinicu u vrsti v. Moguća su dva slučaja:

1) Svi elementi vrste v osim bazične jedinice su jednaki nuli. Tada se vrsta v i kolona koja odgovara promenljivoj w_s izostavljaju iz simpleks tablice.

2) Neka je osim bazične jedinice, recimo, $r-$ti element vrsta v različit od nule. Očigledno je $r > 0$ jer se u nultoj koloni nalazi veštačka promenljiva w_s koja je jednaka nuli, i da $r-$ta kolona ne odgovara veštačkoj promenljivoj, jer su izostavljene sve kolone koje odgovaraju nebazičnim promenljivim. Stožernom transformacijom sa stožernim elementom a_{vs}, učiniti $r-$tu kolonu bazičnom i zatim izostaviti kolonu koja odgovara promenljivoj w_s jer je sada to nebazična kolona koja odgovara veštačkoj promenljivoj. Zameniti k sa $k - 1$ i ponoviti korak 2.

Korak 3. Dobijena simpleks tablica sadrži samo kolone koje odgovaraju promenljivim iz problema (2.6.1). Primeniti Algoritam 1.

Primetimo da se Algoritam 2 očigledno završava u konačnom broju koraka.

Primer za ilustraciju Algoritma 2 je preuzet iz [17].

Primer 2.6.2 Odrediti rešenje sledećeg problema:

$$\begin{aligned} \min \quad & x_1 - x_2 + x_3, \\ \text{p.o.} \quad & x_1 + x_2 + 2x_3 + x_4 = 3, \\ & -x_2 - x_3 + x_4 = 3, \\ & x_1 - x_2 + 3x_4 = 9, \\ & x \geq 0. \end{aligned}$$

Kako nije poznato bazično dopustivo rešenje, primenićemo dvofaznu modifikaciju simpleks algoritma.

I Faza.

Pridruženi problem je:

$$\begin{array}{lrrrrrrrrr} \min & w_1 + w_2 + w_3, & & & & & & & & \\ & x_1 & +x_2 & +2x_3 & +x_4 & +w_1 & & & = & 3, \\ \text{p.o.} & & -x_2 & -x_3 & +x_4 & & +w_2 & & = & 3, \\ & x_1 & -x_2 & & +3x_4 & & & +w_3 & = & 9, \\ & x \geq 0, & w \geq 0, & & & & & & & \end{array}$$

i odgovara mu sledeća LP-tablica:

$$\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 3 & 1 & 1 & 2 & 1 & 1 & 0 & 0 \\ 3 & 0 & -1 & -1 & 1 & 0 & 1 & 0 \\ 9 & 1 & -1 & 0 & 3 & 0 & 0 & 1 \end{array}$$

odakle se eliminacijom jedinica iz nulte vrste (oduzimanjem svih ostalih vrsta od nulte) dobija simpleks tablica (u nastavku su pivot elementi uokvireni):

$$\begin{array}{cccccccc} -15 & -2 & 1 & -1 & -5 & 0 & 0 & 0 \\ 3 & \boxed{1} & 1 & 2 & 1 & 1 & 0 & 0 \\ 3 & 0 & -1 & -1 & 1 & 0 & 1 & 0 \\ 9 & 1 & -1 & 0 & 3 & 0 & 0 & 1 \end{array}$$

Primenom simpleks algoritma dobijamo novu simpleks tablicu

$$\begin{array}{cccccccc} -9 & 0 & 3 & 3 & -3 & -2 & 0 & 0 \\ 3 & 1 & 1 & 2 & 1 & 1 & 0 & 0 \\ 3 & 0 & -1 & -1 & 1 & 0 & 1 & 0 \\ 6 & 0 & -2 & -2 & \boxed{2} & -1 & 0 & 1 \end{array}$$

Kako u nultoj koloni postoje negativni elementi, ponovo primenjujemo simpleks algoritam:

$$\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 3/2 \\ 0 & 1 & 2 & 3 & 0 & 2 & 0 & -1/2 \\ 0 & 0 & 0 & 0 & 0 & -1/2 & 1 & -1/2 \\ 3 & 0 & -1 & -1 & 1 & -1/2 & 0 & 1/2 \end{array}$$

Kako je optimalna vrednost pomoćnog problema $\min e^T w = 0$, prelazimo na drugu fazu.

II Faza. Uklanjanjem nebazičnih kolona koje odgovaraju veštačkim promenljivim (peta i sedma) i zamenom koeficijenata nulte vrste odgovarajućim koeficijentima polaznog problema dobija se LP-tablica

$$\begin{array}{cccccc} 0 & 1 & -1 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 3 & 0 & -1 & -1 & 1 & 0 \end{array}$$

odakle se, posle eliminacije jedinice u nultoj vrsti prve kolone, dobija simpleks tablica

$$\begin{array}{cccccc} 0 & 0 & -3 & -2 & 0 & 0 \\ 0 & 1 & \boxed{2} & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 3 & 0 & -1 & -1 & 1 & 0 \end{array}$$

Sada eliminišemo drugu vrstu jer su u njoj svi elementi (osim koeficijenta koji odgovara veštačkoj promenljivoj) jednaki nuli i eliminišemo petu kolonu jer su svi elementi jednaki nuli. Dobijamo simpleks tablicu iz koje su eliminisane sve veštačke promenljive:

$$\begin{array}{ccccc} 0 & 0 & -3 & -2 & 0 \\ 0 & 1 & \boxed{2} & 3 & 0 \\ 3 & 0 & -1 & -1 & 1 \end{array}$$

Sada su ispunjeni svi uslovi za primenu Algorima 1. U sledećem koraku se dobija optimalno rešenje

$$\begin{array}{ccccc} 0 & 3/2 & 0 & 5/2 & 0 \\ 0 & 1/2 & 1 & 3/2 & 0 \\ 3 & 1/2 & 0 & 1/2 & 1 \end{array}$$

Druga varijanta dvofaznog simpleks metoda je opisana u [51] i [79]. Ovaj algoritam ima prednost u odnosu na prošireni problem zato što ne koristi veštačke promenljive.

Dvofazni simpleks metod bez veštačkih promenljivih

Izložićemo sada jednu verziju algoritma za određivanje početnog bazično dopustivog rešenja koja ne zahteva uvođenje veštačkih promenljivih. Osnovni nedostatak prethodnog metoda jeste povećanje dimenzije linearnog problema. Razmatramo standardni oblik problema linearnog programiranja bez ograničenja o znaku koeficijenata b_i u kome nije poznato bazično dopustivo rešenje. Egzistencija dopustivog rešenja sledi iz sledeće leme:

Lema 2.6.1 *Neka je za neko i koeficijent $b_i = a_{i0}$ negativan. Ako je $a_{ij} \geq 0$, $j = 1, \ldots, n$, tada je skup dopustivih rešenja prazan.*

Dokaz. S obzirom na uslove iz leme, u skupu ograničenja postoji jednačina u kojoj je zbir proizvoda nenegativnih bojeva negativan broj, što je nemoguće. □

Neka je skup rešenja problema linearnog programiranja dopustiv i neka je x bazično nedopustivo rešenje sa q negativnih koordinata. Neka je (novom numeracijom ako je potrebno) postignuto $x = (x_1, \ldots, x_m, 0, \ldots, 0)$ bazično nedopustivo rešenje takvo da je $x_1, \ldots, x_q < 0$, $q \leq m$, i $x_p \geq 0$ za $p > q$. Odgovarajuća baza je $K_1, \ldots, K_m$.

Ako je $q = m$, izaberemo za stožerni element $a_{ms} < 0$ (takvo postoji po Lemi (2.6.1) i primenom simpleks transformacije odmah dobijamo novo rešenje u kome je bar jedna koordinata pozitivna.

Ako je $q < m$, posmatramo $a_{rs} < 0$ za $r = 1, \ldots, q$ (postoje po Lemi (2.6.1). Ako postoje $r \in \{1, \ldots, q\}$ i $s \in \{1, \ldots, n\}$ takvi da je

$$\min_{h>q} \left\{ \frac{x_h}{a_{hs}} | \, a_{hs} > 0 \right\} \geq \frac{x_r}{a_{rs}}, \tag{2.6.1}$$

izaberemo a_{rs} za stožerni element. Tada je

$$x^1 = \sum_{j \neq r} \left(x_j - \frac{x_r}{a_{rs}} a_{js} \right) K_j + \frac{x_r}{a_{rs}} K_s.$$

Za $j > q$ i $a_{js} < 0$ tivijalno važi $x_j - \dfrac{x_r}{a_{rs}} a_{js} \geq 0$. Za $a_{js} \geq 0$ i $j > q$ je

$$\frac{x_j}{a_{js}} \geq \frac{x_r}{a_{rs}} a_{js} \Leftrightarrow x_j - \frac{x_r}{a_{rs}} a_{js} \geq 0$$

i kako je $\dfrac{x_r}{a_{rs}} > 0$, očigledno je da x^1 ima najviše $q - 1$ negativnih koordinata.

Ako ne postoji r takvo da je uslov (2.6.1) ispunjen, neka su sada $r \notin \{1, \ldots, q\}$ i $s \in \{1, \ldots, n\}$ takvi da je

$$\min_{h>q} \left\{ \frac{x_h}{a_{hs}} | \, a_{hs} > 0 \right\} = \frac{x_r}{a_{rs}}. \tag{2.6.2}$$

Za tako određeno r i s postavimo a_{rs} za stožerni element. Primenom simpleks transformacije dobijamo novo rešenje x^1 u kome se dokazuje analogno prethodnom da su koordinate $x^1_{q+1}, \ldots, x^1_m$ ostale nenegativne. Kako je skup bazičnih rešenja konačan, (primenjujući pravila protiv cikliranja ukoliko je potrebno), posle konačno mnogo transformacija će biti ispunjen uslov (2.6.1). Na osnovu ovih razmatranja sledi konačnost sledećeg algoritma.

Algoritam 3.

(Rešavanje problema linearnog programiranja bez uvođenja veštačkih promenljivih).

Korak 1. Formiramo početnu LP-tablicu oblika

$$\begin{array}{rrcr} -d & c_1 & \cdots & c_n \\ b_1 & a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & & \vdots \\ b_m & a_{m1} & \cdots & a_{mn} \end{array}$$

Ukoliko je $b_i \geq 0$, $i = 1, \ldots, m$, primenimo Algoritam 1.

Korak 2. Neka je $b_{i_1}, , \ldots, b_{i_q} < 0$. Ukoliko je $a_{ij} \geq 0$ za neko $i \in \{i_1, \ldots, i_q\}$ i svako $j = 1, \ldots, n$, algoritam staje jer je skup ograničenja nedopustiv. U suprotnom nastavljamo.

Korak 3. Ukoliko je $q = m$ biramo $a_{ms} < 0$ za stožerni element.

Ako je $q < m$ i ako postoje $r \in \{i_1, \ldots, i_q\} = I$ i s takvi da je ispunjen uslov

$$\min_{h \notin I} \left\{ \frac{b_h}{a_{hs}} | \, a_{hs} > 0 \right\} \geq \frac{b_r}{a_{rs}}, \quad a_{rs} < 0,$$

izaberemo a_{rs} za stožerni element.

Ukoliko ne postoji r takvo da je prethodni uslov ispunjen, neka je sada r takvo da je

$$\min_{h \notin I} \left\{ \frac{b_h}{a_{hs}} | \, a_{hs} > 0 \right\} = \frac{b_r}{a_{rs}}.$$

Za tako određeno r i s postavimo a_{rs} za stožerni element.

Korak 4. Primenimo simpleks transformaciju i prelazimo na Korak 1.

Primer 2.6.3 Neka je početna LP-tablica

$$\begin{array}{rrcccc} 720 & 7 & 0 & 0 & 5 & 0 \\ -100 & -2 & 0 & 1 & -1 & 0 \\ -360 & \boxed{-8} & 0 & 0 & -3 & 1 \\ 180 & 3 & 1 & 0 & 1 & 0 \end{array}$$

Kako u nultoj koloni postoje negativni koeficijenti, primenimo Algoritam 3. Uslov (2.6.1) je ispunjen u prvoj koloni (stožerni elementi su uokvireni), i primenom simpleks transformacije dobijamo

$$\begin{array}{rccccr} 405 & 0 & 0 & 0 & 11/8 & 7/8 \\ -10 & 0 & 0 & 1 & -1/4 & \boxed{-1/4} \\ 45 & 1 & 0 & 0 & 3/8 & -1/8 \\ 45 & 0 & 1 & 0 & -1/8 & 3/8 \end{array}$$

U sledećem koraku ponovo primenimo Algoritam 3. jer je $a_{01} < 0$, i odmah dobijamo optimalno rešenje

$$\begin{array}{rrrrrr} 370 & 0 & 0 & 3/2 & 1/2 & 0 \\ 40 & 0 & 0 & -4 & 1 & 1 \\ 50 & 1 & 0 & -1/2 & 1/2 & 0 \\ 30 & 0 & 1 & 3/2 & -1/2 & 0 \end{array}$$

Dokazali smo da ako je $\Omega_P \neq \emptyset$ tada postoji bazično dopustivo rešenje. Uslov za egzistenciju optimalnog bazično dopustivog rešenja da je sledeća lema:

Lema 2.6.2 *Ako je dopustivi skup problema* (1.1.4) *neprazan i ako je ciljna funkcija na njemu ograničena odozdo, onda problem* (1.1.4) *ima bazično dopustivo optimalno rešenje.*

Dokaz. Kako je ciljna funkcija problema (1.1.4) ograničena odozdo, primena simpleks algoritma na početno bazično dopustivo rešenje se završava posle konačno mnogo koraka i to bazično dopustivim optimalnim rešenjem. □

Direktna posledica Leme 2.6.2 je da egzistencija optimalnog rešenja povalači egzistenciju bazično dopustivog optimalnog rešenja.

Algoritam SimpleksMax/SimpleksMin.

Algoritam simpleks metoda za maksimizacione/minimizacione tabele koje nisu bazično dopustive. Tekuća tabela je oblika (2.3.4)

Korak 1. Ako su $b_1^k, b_2^k, \ldots, b_m^k \geq 0$, prelazimo na Korak 5.
U suprotnom nastavljamo.

Korak 2. Biramo $b_i^k < 0$ (Na primer poslednje).

Korak 3. Ako $a_{i1}^k, a_{i2}^k, \ldots, a_{in}^k \geq 0$, STOP:
maksimizacioni zadatak je nedopustiv. (Ovaj slučaj ćemo detaljnije razmotriti malo kasnije).
U suprotnom nastavljamo.

Korak 4. Ako je $i = m$, biramo $a_{mj}^k < 0$, uzimamo za ključni element a_{mj}^k i prelazimo na Korak 1. Ako je $i < m$, biramo $a_{ij}^k < 0$ i izračunavamo

$$\min_{l>i} \left(\left\{ \frac{b_i^k}{a_{ij}^k} \right\} \bigcup \left\{ \frac{b_l^k}{a_{lj}^k}; a_{lj}^k > 0 \right\} \right) = \frac{b_p^k}{a_{pj}^k}$$

pa za ključni element biramo a_{pj}^k (što odgovara zameni bazične promenljive $x_{B,p}$ i nebazične promenljive $x_{N,j}$). Preći na Korak 1.

Korak 5. Primeniti simpleks algoritam za bazično dopustivu maksimizacionu (minimizacionu) tabelu (Algoritam *SimpleksBasicMax*/*SimpleksBasicMin*).

Napomena 2.6.1 *Kao i u Algoritmu SimpleksBasicMax može postojati više od jedne vrednosti za p za koju je* $\frac{b_p^k}{a_{pj}^k}$ *najmanje. Može se odabrati proizvoljno p.*

Implementacija simpleks algoritma koji ne koristi veštačke promenljive pomoću funkcionalnih mogućnosti u jeziku MATHEMATICA opisana je u monografiji [72].

Primer 2.6.4 Ovim primerom se ilustruje opšti simpleks algoritam (Algoritam *SimpleksMax*) na tabelu maksimizacije

x_1	x_2	-1	
-1	-2	-3	$=-t_1$
1	1	3	$=-t_2$
1	1	2	$=-t_3$
-2	4	0	$=z$

Rešenje:
(1) Početna tabela je očigledno tabela maksimizacije.
(2) Prelazimo na korak (2) jer je $b_1 = -3$ negativno.
(3) Biramo $b_1 = -3$.
(4) Prelazimo na korak (4), jer su oba koeficijenta $a_{11} = -1$ i $a_{12} = -2$ negativna.
(5) Možemo da izaberemo $a_{11} = -1$ ili $a_{12} = -2$. Radi određenosti biramo $a_{11} = -1$. Kako je $1 = i < m = 3$ izračunavamo

$$\min\left(\left\{\frac{b_1}{a_{11}} = \frac{-3}{-1}\right\} \bigcup \left\{\frac{b_2}{a_{21}} = \frac{3}{1}, \frac{b_3}{a_{31}} = \frac{2}{1}\right\}\right) = \frac{b_3}{a_{31}}$$

Za ključni element uzimamo a_{31}:

x_1	x_2	-1	
-1	-2	-3	$=-t_1$
1*	1	3	$=-t_2$
1	1	2	$=-t_3$
-2	4	0	$=z$

$\Rightarrow$

t_3	x_2	-1	
-1	-1	-1	$=-t_1$
-1	0	1	$=-t_2$
1	1	2	$=-x_1$
2	6	4	$=z$

Prelazimo na korak (1).
(1) Očigledno!
(2) Prelazimo na korak (2), jer je $b_2 = -1$ negativno.
(3) Moramo da izaberemo $b_2 = -1$.
(4) Prelazimo na korak (4), jer je $a_{12} = -1$ negativno.
(5) Moramo da izaberemo $a_{12} = -1$. Kako je $1 = i < m = 3$, izračunavamo

$$\min\left(\left\{\frac{b_1}{a_{12}} = \frac{-1}{-1}\right\} \bigcup \left\{\frac{b_3}{a_{32}} = \frac{2}{1}\right\}\right) = \frac{b_1}{a_{12}}$$

Ključni element je a_{12}:

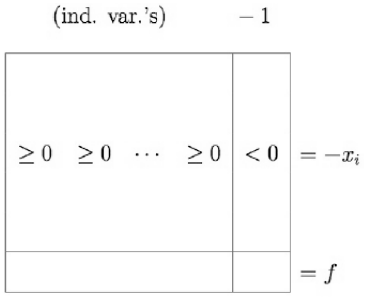
t_3	x_2	-1	
-1	-1	-1	$=-t_1$
-1	0	1	$=-t_2$
1	1	2	$=-x_1$
2	6	4	$=z$

$\Rightarrow$

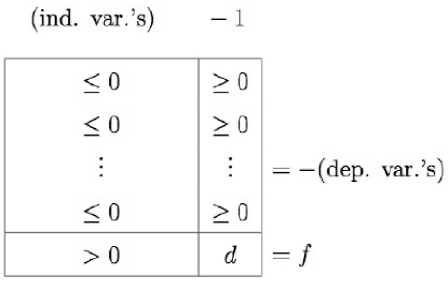
t_3	t_1	-1	
-1	-1	1	$=-x_2$
-1	0	1	$=-t_2$
2	1	1	$=-x_1$
8	6	-2	$=z$

Prelazimo na korak (1).
(1) Očigledno!
(2) $b_1, b_2, b_3 \geq 0$, to jest, tabela je maksimizaciona bazično dopustiva. Prelazimo na korak(5).
(5) Primeniti simpleks algoritam za maksimizacione tabele sa bazično dopustivim rešenjima (prepustimo čitaocu da sam završi zadatak).

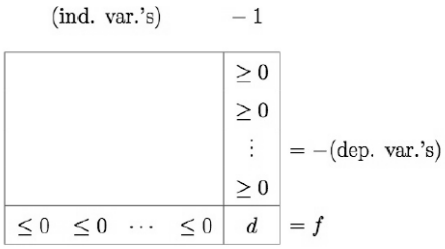
Mogući završeci u simpleks algoritmu prikazani su na sledećim slikama:



Slika 2.6.1. Tuckerova tabela kada optimalno rešenje ne postoji.



Slika 2.6.2. Tuckerova tabela kada je optimalno rešenje neograničeno.



Slika 2.6.3. Tuckerova tabela kada optimalno rešenje postoji.

2.7 BigM metod

Kombinovanjem I i II faze dobijen je algoritam kojim je moguće rešiti proizvoljan problem linearnog programiranja. Napomenimo na kraju da je suština ideje dvofazne modifikacije simpleks metoda u sledećem: ograničenja polaznog problema (2.5.1) uvođenjem veštačkih promenljivih $w_1, \ldots, w_m$ dopunimo do bazično dopustivog rešenja, posmatrajući pri tome proširenu ciljnu funkciju

$$z^w = c^T x + Mw_1 + \cdots + Mw_m,$$

gde je M proizvoljno veliki koeficijent. Sve dok se veštačke promenljive javljaju u bazično dopustivom rešenju, optimalno rešenje nije pronađeno zbog proizvoljnosti koeficjenta M. Dakle, cilj je da se primenom simpleks metoda sve veštačke

promenljive eliminišu iz bazično dopustivog rešenja i time izjednače sa nulom. Kada se formira bazično dopustivo rešenje bez veštačkih promenljivih, one nam nisu više potrebne jer su u redukovanom problemu ispunjeni uslovi za primenu Algoritma 1. Smisao uvođenja veštačkih promenljivih je samo u tome da se kanališe redosled izvođenja elementarnih transformacija.

U jednoj varijanti ovog metoda [17, 73], pomoćni problem se formira na sledeći način:

$$\begin{aligned} \min \quad & c^T x + Me^T w, \\ \text{p.o.} \quad & Ax + w = b, \\ & x \geq 0,\ w \geq 0, \end{aligned} \tag{2.7.1}$$

gde je M dovoljno velika konstanta. Početno bazično rešenje ovog problema je $(0, b)$. Ukoliko se, primenom simpleks metoda dobije optimalno rešenje kod koga je neka od promenljivih w_i bazična, polazni problem je nedopustiv (zbog proizvoljnosti konstante M). U suprotnom, dobijeno rešenje je optimalno i za polazni problem. Zbog konstante M ovaj metod se često naziva i ***BigM*** metod. Glavni nedostatak ovog metoda je povećanje dimenzije problema i neodređenost oko izbora konstante M.

Primer 2.7.1 Razmotrimo sada problem u kome se traži minimalna vrednost funkcije kriterijuma primenom ***BigM*** metoda:

$$\begin{aligned} \min \quad & 160x_1 + 100x_2 + 120x_3, \\ \text{p.o.} \quad & 2x_1 + x_2 + 2x_3 \geq 350, \\ & x_1 + x_2 + x_3 \geq 300, \\ & 4x_1 + x_2 + 2x_3 \geq 400, \\ & x_1 \geq 0\ ,\ x_2 \geq 0\ ,\ x_3 \geq 0. \end{aligned} \tag{2.7.2}$$

Uvođenjem izravnavajućih promenljivih, sistem (2.7.2) ćemo izraziti u obliku jednačina. Pošto je leva strana nejednačina ovog sistema veća od desne, oduzimamo izravnavajuće promenljive i dobijamo sledeći sistem jednačina:

$$\begin{aligned} & 2x_1 + x_2 + 2x_3 - x_4 = 350\ , \\ & x_1 + x_2 + x_3 - x_5 = 300\ , \\ & 4x_1 + x_2 + 2x_3 - x_6 = 400\ . \end{aligned} \tag{2.7.3}$$

Koeficijenti uz izravnavajuće promenljive su negativni. To znači da ne možemo odrediti početno nenegativno bazično rešenje u prvoj simpleks tabeli. Zbog toga uvodimo nove promenljive u model, tzv. veštačke promenljive. Ove promenljive ne pripadaju sistemu ograničenja i nemaju konkretno ekonomsko značenje, osim što služe kao kalkulativno sredstvo. U postupku rešavanja problema one se eliminišu iz rešenja, tako što se u optimalnom rešenju ne može pojaviti ni jedna veštačka promenljiva. Ako nenegativno bazično rešenje problema sadrži bar jednu pozitivnu veštačku promenljivu (koje se ne možemo osloboditi), onda ne postoji moguće rešenje problema.

Da se veštačke promenljive ne bi pojavile u optimalnom rešenju, u funkciji kriterijuma uz ove promenljive uvodimo koeficijente obeležene sa M, gde je M relativno veliki pozitivan broj.[1] Tako smo, posle proširenja problema sa veštačkim promenljivim, dobili sledeći model:

$$(\min);\ z_0 = 160x_1 + 100x_2 + 120x_3 + 0x_4 + 0x_5 + +0x_6 + Mx_7 + Mx_8 + Mx_9\ ,$$

[1] Kada tražimo maksimalnu vrednost funkcije kriterijuma, uz veštačke promenljive uvodimo koeficijent $-M$.

$$\begin{aligned} 2x_1 + x_2 + 2x_3 - x_4 \qquad\quad + x_7 \qquad\quad &= 350\ , \\ x_1 + x_2 + \ x_3 \qquad - x_5 \qquad + x_8 \qquad &= 300\ , \\ 4x_1 + x_2 + 2x_3 \qquad - x_6 \qquad + x_9 \quad &= 400\ , \end{aligned}$$

u kome sve promenljive moraju biti nenegativne.

Sada se može popuniti početna simpleks tabela i odrediti početno bazično rešenje. Njega će sačinjavati veštačke promenljive.

			160	100	120	0	0	0	M	M	M	
C	B	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	
M	X_7	350	2	1	2	-1	0	0	1	0	0	
M	X_8	300	1	1	1	0	-1	0	0	1	0	
M	X_9	400	$\boxed{4}$	1	2	0	0	-1	0	0	1	$\rightarrow$
$z_j - c_j$		0	-160	-100	-120	0	0	0	0	0	0	
		1050	7	3	5	-1	-1	-1	0	0	0	
			$\uparrow$									

Tabela 2.7.1.

Simpleks tabela je popunjena na uobičajeni način. Jedino su koeficijenti u redu $(z_j - c_j)$ podeljeni na dva dela. U prvi deo reda unose se koeficijenti uz koje se ne javlja M, a u drugi deo koeficijenti uz koje se javlja M. Tako je, na primer, koeficijent $(z_1 - c_1) = 7M - 160$. Dobijen je množenjem kolone C i kolone X_1, proizvodi su sabrani i od njih je oduzet koeficijent iz zaglavlja. Na isti način utvrđeni su i koeficijenti za ostale kolone. Ova podela koeficijenata iz reda $(z_j - c_j)$ na dva dela izvršena je samo iz praktičnih razloga. Kako je M veoma veliki broj, to će, sve dok su i veštačke promenljive u rešenju, drugi deo reda određivati koja promenljiva ulazi u naredno rešenje.

Da li je u prvoj simpleks tabeli pronađeno optimalno rešenje i, ako nije, koju promenljivu treba izabrati da uđe u naredno rešenje? Ovde je odgovor sasvim očigledan: početno rešenje čine veštačke promenljive, pa je sigurno da ono ne može biti optimalno. Sve dok u redu $(z_j - c_j)$ postoje pozitivni koeficijenti (a tražimo minimalnu vrednost funkcije), nije pronađeno optimalno rešenje. Dodajmo ovome i drugi kriterijum: u naredno rešenje treba da uđe ona promenljiva kojoj u simpleks tabeli odgovara najveća (pozitivna) vrednost koeficijenta iz reda $(z_j - c_j)$.

Vratimo se prvoj tabeli i konstatujmo da nije pronađeno optimalno rešenje, te da u naredno rešenje treba da uđe promenljiva x_1. Na osnovu količnika između kolone X_0 i kolone X_1, iz rešenja izlazi promenljiva x_9. U prvoj simpleks tabeli označena je kolona X_1, treći red u kome je promenljiva x_9 i zaokružen je karakterističan koeficijent. Na osnovu tako pripremljene prve simpleks tabele, dobijamo drugu simpleks tabelu.

Bazično rešenje u drugoj tabeli sačinjavaju promenljive

$$x_1 = 100\ , \quad x_7 = 150\ , \quad x_8 = 200\ ,$$

a vrednost funkcije kriterijuma iznosi $z_0 = 16000 + 350M$. Optimalno rešenje nije pronađeno. U naredno rešenje treba da uđe promenljiva x_3, a iz rešenja će izaći promenljiva x_7. Zbog toga smo u drugoj tabeli označili kolonu X_3, prvi red u kome je promenljiva x_7 i zaokružili karakterističan koeficijent.

			160	100	120	0	0	0	M	M	M	
C	B	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	
M	X_7	150	0	1/2	$\boxed{1}$	-1	0	1/2	1	0	$-1/2$	$\rightarrow$
M	X_8	200	0	3/4	1/2	0	-1	1/4	0	1	$-1/4$	
160	X_1	100	1	1/4	1/2	0	0	$-1/4$	0	0	1/4	
$z_j - c_j$		16000	0	$-$ 60	$-$ 40	0	0	$-$ 40	0	0	40	
		350	0	5/4	3/2	-1	-1	3/4	0	0	$-7/4$	
					$\uparrow$							

Tabela 2.7.2.

Sastavljamo treću simpleks tabelu.

			160	100	120	0	0	0	M	M	M	
C	B	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	
120	X_3	150	0	1/2	1	− 1	0	1/2	1	0	−1/2	
M	X_8	125	0	1/2	0	1/2	−1	0	−1/2	1	0	→
160	X_1	25	1	0	0	1/2	0	−1/2	−1/2	0	1/2	
$z_j - c_j$		22000	0	− 40	0	− 40	0	− 20	40	0	20	
		125	0	1/2	0	1/2	−1	0	−3/2	0	− 1	
				↑								

Tabela 2.7.3.

I u trećoj simpleks tabeli nije pronađeno optimalno rešenje. U rešenju se nalazi još jedna veštačka promenljiva ($x_8 = 125$), a u drugom delu reda $(z_j - c_j)$ postoje pozitivni koeficijenti. Treba odrediti koja promenljiva ulazi u naredno rešenje. Vidimo da u trećoj simpleks tabeli dve promenljive (x_2 i x_4) ravnopravno konkurišu za ulazak u naredno rešenje jer imaju potpuno iste koeficijente. Prema dopunskom kriterijumu prednost će dobiti ona promenljiva koja (ukoliko bude izabrana da uđe u rešenje) dobija veću vrednost. Zato najpre delimo kolonu X_0 kolonom X_2 i određujemo da će promenljiva x_2, ukoliko uđe u naredno rešenje, dobiti vrednost 250; zatim delimo kolonu X_0 kolonom X_4 i određujemo da će promenljiva x_4, ukoliko uđe u rešenje, dobiti vrednost 50.

Prema tome, u naredno rešenje ulazi promenljiva koja dobija veću vrednost, tj. promenljiva x_2. U trećoj tabeli označili smo kolonu X_2, zatim drugi red (jer promenljiva x_8 izlazi iz rešenja) i zaokružili karakterističan koeficijent.

Sastavljamo četvrtu simpleks tabelu.

			160	100	120	0	0	0	M	M	M
C	B	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
120	X_3	25	0	0	1	−3/2	1	1/2	3/2	− 1	−1/2
100	X_2	250	0	1	0	1	− 2	0	− 1	2	0
160	X_1	25	1	0	0	1/2	0	−1/2	−1/2	0	1/2
$z_j - c_j$		32000	0	0	0	0	−80	− 20	0	80	20
		0	0	0	0	0	0	0	− 1	− 1	− 1

Tabela 2.7.4.

U četvrtoj simpleks tabeli nema veštačkih promenljivih u rešenju. To se može zaključiti prema koloni B, ali i prema drugom delu reda $(z_j - c_j)$. Pošto su veštačke promenljive izašle iz rešenja, u ovom delu reda $(z_j - c_j)$ ispod svih kolona su nule, osim ispod kolona veštačkih promenljivih gde su jedinice sa znakom minus.

Problem se dalje može rešavati bez kolona koje odgovaraju veštačkim promenljivim (ove promenljive kad jednom izađu iz rešenja, ne mogu se više vratiti u njega) i bez drugog dela reda $(z_j - c_j)$. Ove kolone se mogu zanemarivati i odmah pošto veštačka promenljiva izađe iz rešenja. Postoji, međutim, razlog da se zadrže i ove kolone do kraja postupka, jer se u tom delu simpleks tabele nalaze podaci koji se mogu korisno upotrebiti za analizu optimalnog rešenja.

Rešavanje problema simpleks tabelom treba nastaviti tako što se za ocenu optimalnosti rešenja i za izbor promenljive koja će ući u rešenje uzima u obzir prvi deo reda $(z_j - c_j)$. Posmatranjem ovog reda u četvrtoj simpleks tabeli, vidimo da ne postoje pozitivni koeficijenti, pa možemo zaključiti da je rešenje iz ove tabele optimalno. Njega sačinjavaju promenljive:

$$x_1 = 25\ , \quad x_2 = 250\ , \quad x_3 = 25\ ,$$

a minimalna vrednost funkcije kriterijuma iznosi $z_0 = 32000$.

Rekli smo da je u četvrtoj simpleks tabeli pronađeno optimalno rešenje. Međutim, u redu $(z_j - c_j)$ kolone X_4 (kolona nebazične promenljive x_4) nalazi se koeficijent nula. To znači da

možemo odrediti da promenljiva x_4 uđe u naredno rešenje i da odredimo još jedno rešenje sa istom vrednošću funkcije kriterijuma. Drugim rečima, možemo odrediti još jedno optimalno rešenje, pa je potrebno da sastavimo još jednu simpleks tabelu. Pošto promenljiva x_4 ulazi u naredno rešenje, u četvrtoj tabeli označili smo kolonu X_4. Iz rešenja izlazi promenljiva x_1, pa smo označili i treći red i zaokružili karakterističan koeficijent. Sada možemo odrediti petu simpleks tabelu.

			160	100	120	0	0	0	M	M	M
C	B	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
120	X_3	100	3	0	1	0	1	− 1	0	− 1	1
100	X_2	200	−2	1	0	0	− 2	1	0	2	− 1
0	X_4	50	2	0	0	1	0	− 1	−1	0	1
$z_j - c_j$		32000	0	0	0	0	−80	−20	0	80	20
		0	0	0	0	0	0	0	−1	− 1	− 1

Tabela 2.7.5.

U petoj simpleks tabeli dobili smo još jedno optimalno rešenje. Njega sačinjavaju promenljive

$$x_2 = 200\ , \quad x_3 = 100\ , \quad x_4 = 50\ ,$$

a minimalna vrednost funkcije kriterijuma, $z_0 = 32000$, ista je kao i kod rešenja iz četvrte simpleks tabele.

Napomenimo samo da se konveksnom kombinacijom ova dva optimalna rešenja mogu dobiti i druga, takođe, optimalna rešenja. Rešenja koja su pronađena pomoću simpleks tabela bazična su, a sva rešenja koja se mogu dobiti kao konveksna kombinacija bazičnih rešenja nisu bazična.

Obeležimo optimalno rešenje iz četvrte simpleks tabele sa X_0^1, a rešenje iz pete tabele sa X_0^2, pa ćemo dobiti sledeći izraz za konveksnu kombinaciju:

$$X_0^3 = qX_0^1 + (1-q)X_0^2\ , \quad 0 < q < 1\ ,$$

gde X_0^3 predstavlja novo optimalno rešenje.

Za problem minimuma koji smo rešavali bilo je potrebno da se u svakoj jednačini doda veštačka promenljiva. To je problem u kome je sistem ograničenja imao sve nejednačine smera "$\geq$", pa je u svim nejednačinama oduzeta izravnavajuća promenljiva i dodata veštačka. Za ovakve probleme, kada je broj veštačkih promenljivih jednak broju ograničenja, kažemo da imaju potpunu veštačku bazu. Razume se da ne mora i neće uvek tako biti. Ukoliko problem ima manje veštačkih promenljivih nego što je broj ograničenja, onda imamo nepotpunu veštačku bazu. U postupku rešavanja ovih problema nema nikakvih razlika u odnosu na već razmatrani postupak.

Razmotrimo sada metod iz [84] kod koga nema povećanja dimenzija problema. Posmatraćemo jedan kanonski oblik (2.3.1) problema linearnog programiranja:

$$\begin{aligned} \max\ & z(x) = c_1x_{N,1} + \ldots + c_nx_{N,n} + d \\ \text{p.o.}\ & a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n - b_1 = -x_{B,1} \\ & a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n - b_2 = -x_{B,2} \\ & \quad \ldots\ldots\ldots \\ & a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n - b_m = -x_{B,m}. \end{aligned} \tag{2.7.4}$$

pri čemu nisu svi $b_i \geq 0$, tj postoji neko $i \in \{1, \ldots, m\}$ takvo da je $b_i < 0$ ali koeficijenti c_i ispunjavaju uslov optimalnosti. Sada odgovarajuće bazično rešenje $x = (0, \ldots, 0, b_1, \ldots, b_n)$ nije dopustivo, tj ne pripada skupu Ω_P. Zadatak koji rešavamo je nalaženje ekvivalentnog bazično dopustivog problema (2.7.4), odnosno nalaženje jednog bazično dopustivog rešenja. Za to nam je potreban dualni simpleks metod.

2.8 Dualnost u linearnom programiranju

U određenim slučajevima matematički model osnovnog zadatka LP ne može da posluži za nalaženje optimalnog plana primenom opisanog simpleks metoda. U ovakvim slučajevima se pribegava preformulaciji primalnog zadatka LP u dualni zadatak, pomoću koga je moguće naći traženo rešenje. Između primalnog i dualnog zadatka postoje sledeće korespodencije:

1. Dualni model ima onoliko promenljivih koliko primalni zadatak ograničenja, i ograničenja koliko primalni zadatak promenljivih;

2. Slobodni članovi u ograničenjima primalnog zadatka postaju koeficijenti uz promenljive funkcije kriterijuma (cilja) dualnog modela, a koeficijenti uz promenljive funkcije kriterijuma primalnog modela postaju slobodni članovi u ograničenjima dualnog modela;

3. Smer nejednačina dualnog modela je suprotan smeru nejednačina primalnog modela;

4. Ako je u primernom modelu tražen maksimum funkcije kriterijuma, u dualnom se traži minimum, i obrnuto;

5. Dopunskoj promenljivoj x_{n+1} primala koja je u optimalnom baznom rešenju odgovara promenljiva y_j u dualnom modelu, pri čemu je $x_{n+1}y_j = 0,\ j = 1, \ldots, m$.

6. Realnoj promenljivoj x_j primala iz bazičnog dopustivog rešenja odgovara dopunska promenljiva y_{m+j} dualnog modela sa nultom vrednošću $x_j y_{m+j} = 0,\ j = 1, \ldots, n$.

7. Matrica ograničenja u primalnom modelu jednaka je transponovanoj matrici ograničenja u dualnom modelu.

Razmatramo problem linearnog programiranja u standardnom obliku

$$\begin{aligned} \min \quad & c^T x, \\ \text{p.o.} \quad & Ax = b, \\ & x \geq 0, \end{aligned} \tag{2.8.1}$$

gde je A matrica tipa $m \times n$ sa uobičajenom pretpostavkom $\text{rang}(A) = m$. Problemu (2.8.1) pridružimo tzv. *dualni problem* oblika

$$\begin{aligned} \max \quad & b^T y, \\ \text{p.o.} \quad & A^T y \leq c \end{aligned} \tag{2.8.2}$$

u kome su slobodni koeficijenti problema (2.8.1) postali koeficijenti u ciljnoj funkciji, a koeficijenti iz ciljne funkcije problema (2.8.1) postali slobodni koeficijenti u sistemu ograničenja. Primetimo da za vektor y nije nametnut uslov nenegativnosti. Problem (2.8.1) se u kontekstu teorije dualnosti naziva *primalni problem.* Dodavanjem izravnavajućih promenljivih problem (2.8.2) se svodi na

$$\begin{aligned} \max \quad & b^T y, \\ \text{p.o.} \quad & A^T y + s = c, \\ & xs \geq 0. \end{aligned} \tag{2.8.3}$$

Primer 2.8.1 Neka je dat primalni problem linearnog programiranja

$$\begin{array}{lrrrrrl} \min & -x_1 - x_2, & & & & & \\ \text{p.o.} & x_1 & -x_2 & -x_3 & & = 1, \\ & -x_1 & +x_2 & & -x_4 & = 1, \\ & x \geq 0. & & & & & \end{array}$$

Ovom problemu, dualni problem je

$$\begin{array}{lrrrrrrrr} \max & y_1 + y_2, & & & & & & & \\ \text{p.o.} & y_1 & -y_2 & +s_1 & & & & = & -1, \\ & -y_1 & +y_2 & & +s_2 & & & = & -1, \\ & -y_1 & & & & +s_3 & & = & 0, \\ & & -y_2 & & & & +s_4 & = & 0, \\ & s \geq 0. & & & & & & & \end{array}$$

Neka su Ω_P i Ω_D dopustivi skupovi problema (2.8.1) i (2.8.3), respektivno. Između ciljnih funkcija primala i duala postoji sledeća veza:

Teorema 2.8.1 (Slaba dualnost). *Ako su Ω_P i Ω_D neprazni skupovi i ako je $x \in \Omega_P$ i $(y, s) \in \Omega_D$, tada je $c^T x \geq b^T y$.*

Dokaz. S obzirom da važi $x \geq 0$ i $s \geq 0$ dobija se

$$0 \leq s^T x = (c - A^T y)^T x = c^T x - y^T (Ax) = c^T x - b^T y.$$

Dokaz je završen. □

Sledeća teorema je fundamentalna u teoriji dualnosti.

Teorema 2.8.2 (Jaka dualnost) *Ako problem* (2.8.1) *ima optimalno rešenje onda i problem* (2.8.3) *ima optimalno rešenje i pri tome* $\min c^T x = \max b^T y$.

Dokaz. Kako problem (2.8.1) po pretposavci teoreme ima optimalno rešenje, to se primena Algoritma 2 na (2.8.1) završava u konačnom broju iteracija sa optimalnim bazično dopustivim rešenjem x^*. Neka su $j_1 < \cdots < j_m$ indeksi bazičnih, a $i_1 < \cdots < i_{n-m}$ indeksi nebazičnih kolona u odgovarajućoj simpleks tablici i neka je $A_B = [K_{j_1} \dots K_{j_m}]$, $A_N = [K_{i_1} \dots K_{j_{n-m}}]$, $c_B = (c_{j_1}, \dots, c_{j_m})$, $c_N = (c_{i_1}, \dots, c_{j_{n-m}})$. Na osnovu prethodnih razmatranja kanonskog oblika linearnog problema, nulta vrsta na nultoj poziciji sadrži $-cTx^* = -c_B^T A_B^{-1} b$, na pozicijama $j_1, \dots, j_m$ sadrži nule i na pozicijama $i_1, \dots, j_{n-m}$ sadrži vektor $c_N - A_N^T (A_B^{-1})^T c_B$ koji je nenegativan zbog uslova optimalnosti.

Neka je $y^* = (A_B^{-1})^T c_B$ i neka je s^* definisano sa $s_B^* = 0$, $s_N^* = c_N - A_N^T (A_B^{-1})^T c_B$. Tada je $s^* \geq 0$ i

$$A_B^T y^* + s_B^* = c_B, \quad A_N^T y^* + s_N^* = c_N,$$

tj. rešenje (y^*, s^*) je dopustivo. Pored toga je $c^T x^* = c_B^T A_B^{-1} b = (y^*)^T b = b^T y^*$. Kako po Teoremi 2.8.1 važi $b^T y \leq c^T x^* = b^T y^*$, sledi da je (y^*, s^*) optimalno rešenje problema (2.8.3). □

Iz sledeće teoreme slede Karash-Kuhn-Tuckerovi uslovi optimalnosti.

Teorema 2.8.3 *(i)* $x^* \in \Omega_P$ *je optimalno rešenje problema* (2.8.1) *ako i samo ako postoji* $(y^*, s^*) \in \Omega_D$ *tako da je* $c^T x^* = b^T y^*$.

(ii) $x^* \in \Omega_P$ *je optimalno rešenje problema* (2.8.1) *ako i samo ako postoji* $(y^*, s^*) \in \Omega_D$ *tako da je* $(x^*)^T s^* = 0$.

Dokaz. (i) Neka je $x^* \in \Omega_P$ optimalno rešenje problema (2.8.1). Tada po Teoremi 2.8.2 problem (2.8.3) ima optimalno rešenje $(y^*, s^*) \in \Omega_D$ i važi $c^T x^* = b^T y^*$.

Ako postoji $(y^*, s^*) \in \Omega_D$ tako da je $c^T x^* = b^T y^*$, na osnovu Teoreme 2.8.1 sledi da za svako $x \in \Omega_P$ važi $c^T x \geq c^T x^*$. Kako je $x^* \in \Omega_P$, to je x^* optimalno rešenje problema (2.8.1).

(ii) Neka je $x^* \in \Omega_P$ optimalno rešenje problema (2.8.1). Tada prema (i) postoji $(y^*, s^*) \in \Omega_D$ i važi $c^T x^* = b^T y^*$. Kada ovu jednakost transponujemo i od nje oduzmemo jednakost $(x^*)^T A^T y^* = (y^*)^T A x^*$ dobijamo

$$(x^*)^T (c - A^T y^*) = (y^*)^T (b - A x^*).$$

Sada iz $s^* = c - A^T y^*$ i $b - A x^* = 0$ sledi $(x^*)^T s^* = 0$.

Ako je $(y^*, s^*) \in \Omega_D$ takvo da je $(x^*)^T s^* = 0$, zbog dopustivosti x^* i (y^*, s^*) sledi

$$(x^*)^T s^* = (x^*)^T (c - A^T y^*) = (x^*)^T c - (x^*)^T A^T y^* = (x^*)^T c - b^T y^* = 0,$$

pa na osnovu (i) sledi da je x^* optimalno rešenje problema (2.8.1). □

Značaj Teoreme 2.8.3 je u tome što se uspostavlja sledeća ekvivalencija: vektor $x^* \in \mathbb{R}^n$ je optimalno rešenje problema (2.8.1) ako i samo ako postoje vektori $s^* \in \mathbb{R}^n$ i $y^* \in \mathbb{R}^m$ tako da važe sledeći uslovi

$$\begin{aligned} A^T y^* + s^* &= c, \\ A x^* &= b, \\ x_i^* s_i^* &= 0, \quad i = 1, \ldots, n, \\ (x^*, s^*) &\geq 0. \end{aligned} \tag{2.8.4}$$

Očigledno je da je (2.8.4) takođe potreban i dovoljan uslov da (y^*, s^*) bude optimalno rešenje problema (2.8.3). Ovi uslovi igraju važnu ulogu u takozvanim primal-dual metodima unutrašnje tačke.

Pitanje egzistencije dopustivih rešenja primalnog i dualnog problema rešava sledeća teorema:

Teorema 2.8.4 (i) $\Omega_P \neq \emptyset$ *i* $\Omega_D \neq \emptyset$ *ako i samo ako problemi* (2.8.1) *i* (2.8.3) *imaju optimalna rešenja.*

(ii) *Neka je* $\Omega_P \neq \emptyset$. *Tada je* $\Omega_D = \emptyset$ *ako i samo ako je* $\inf\limits_{x \in \Omega_P} c^T x = -\infty$.

(iii) *Neka je* $\Omega_D \neq \emptyset$. *Tada je* $\Omega_P = \emptyset$ *ako i samo ako je* $\sup\limits_{(y,s) x \in \Omega_D} b^T y = \infty$.

Dokaz. (i) Ako problemi (2.8.1) i (2.8.3) imaju optimalna rešenja, trivijalno je $\Omega_P \neq \emptyset$ i $\Omega_D \neq \emptyset$.

Neka je $\Omega_P \neq \emptyset$ i $\Omega_D \neq \emptyset$ i uočimo proizvoljno $(y, s) \in \Omega_D$. Tada je po Teoremi 2.8.1 ciljna funkcija problema (2.8.1) ograničena odozdo i stoga problem (2.8.1) ima optimalno rešenje. Na osnovu Teoreme 2.8.2 sledi da i problem (2.8.3) ima optimalno rešenje.

(ii) Neka je $\Omega_D = \emptyset$ i pretpostavimo suprotno, tj. da je $c^T y \geq M$ za $x \in \Omega_P$. Tada problem (2.8.1) ima optimalno rešenje i na osnovu Teoreme 2.8.2 sledi da i problem (2.8.3) ima optimalno, stoga i dopustivo rešenje, što je kontradikcija.

Neka je $\inf_{x \in \Omega_P} c^T x = -\infty$ i pretpostavimo da postoji $(y, s) \in \Omega_D$. Tada je prema Teoremi 2.8.1 $c^T x \geq c^T y$ za svako $x \in \Omega_P$, što je nemoguće.

(iii) Uvođenjem smene $y = y^+ - y^-$, $y^+ \geq 0$, $y^- \geq 0$, problem (2.8.3) je ekvivalentan minimizacionom problemu

$$\begin{array}{ll} \min & (-b)^T (y^+ - y^-), \\ \text{p.o.} & A^T (y^+ - y^-) + s = c, \\ & y^+ \geq 0,\ y^- \geq 0,\ s \geq 0, \end{array} \tag{2.8.5}$$

čiji je dual

$$\begin{array}{ll} \max & c^T u, \\ \text{p.o.} & Au \leq -b, \quad -Au \leq b, \quad u \leq 0. \end{array} \tag{2.8.6}$$

Smenom $x = -u$ dual (2.8.6) postaje

$$\begin{array}{ll} \max & (-c)^T x, \\ \text{p.o.} & Ax = b, \\ & x \geq 0. \end{array} \tag{2.8.7}$$

Primenom (ii) na dualni par (2.8.5) i (2.8.7) direktno sledi da je $\Omega_P = \emptyset$ ako i samo ako je $\inf\limits_{(y,s) x \in \Omega_D} (-b)^T y = -\infty$, tj. $\sup\limits_{(y,s) x \in \Omega_D} b^T y = \infty$. $\square$

Primetimo da iz dokaza Teoreme 2.8.4 (iii) sledi zaključak da je dual dualnog problema jednak primalnom problemu.

Napomena 2.8.1 *U Primeru 2.6.1 je $\Omega_P = \emptyset$ i $\Omega_D = \emptyset$, što pokazuje da je moguć i taj slučaj.*

Neka je $I_1 \cup I_2 = \{1, \ldots, m\}$, $I_1 \cap I_2 = \emptyset$, $J_1 \cup J_2 = \{1, \ldots, n\}$, $J_1 \cap J_2 = \emptyset$, i neka su $V_1, \ldots, V_m$ vrste $K_1, \ldots, K_n$ kolone matrice A. Tada problemu

$$\begin{array}{ll} \min & c^T x, \\ \text{p.o.} & V_i^T x = b_i, \quad i \in I_1, \\ & V_i^T x \geq b_i, \quad i \in I_2, \\ & x_j \geq 0, \qquad j \in J_1, \\ & x_j \text{ neograničeno po znaku}, \ j \in J_2, \end{array} \tag{2.8.8}$$

odgovara dualni problem

$$\begin{array}{ll} \max & b^T y, \\ \text{p.o.} & y_i \text{ neograničeno po znaku, } i \in I_1, \\ & y_i \geq 0, \quad i \in I_2, \\ & K_j^T y \geq c_j, \quad j \in J_1, \\ & K_j^T y = c_j, \quad j \in J_2, \end{array} \tag{2.8.9}$$

i važe analogoni Teorema 2.8.1 i 2.8.2. Dual simetričnog problema linearnog programiranja

$$\begin{array}{ll} \min & c^T x, \\ \text{p.o.} & Ax \geq b, \\ & x \geq 0, \end{array}$$

dat je sa

$$\begin{array}{ll} \max & b^T y, \\ \text{p.o.} & A^T y \leq c, \\ & y \geq 0. \end{array}$$

Iz (2.8.8) i (2.8.9) sledi da je problem dualan sa (1.1.3) dat sa

$$\begin{array}{ll} \max & b^T y, \\ \text{p.o.} & y_i \text{ neograničeno po znaku, } i \in I_1, \\ & y_i \geq 0, \quad i \in I_2, \\ & y_i \leq 0, \quad i \in I_3, \\ & K_j^T y \leq c_j, \quad j \in J, \\ & K_j^T y = c_j, \quad j \in \{1, \ldots, n\} \backslash J. \end{array}$$

2.9 Dualni simpleks metod

Razmotrimo problem linearnog programiranja kome odgovara LP-tablica

$$\begin{array}{cccc} -d & c_1 & \cdots & c_n \\ b_1 & a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & & \vdots \\ b_m & a_{m1} & \cdots & a_{mn} \end{array}$$

koju nazivamo *dualnom simpleks tablicom* ako ona sadrži m različitih bazičnih kolona i važi $c_1 \geq 0, \ldots, c_n \geq 0$. Ako je $b_1 \geq 0, \ldots, b_m \geq 0$, tada je dualna simpleks tablica ujedno i simpleks tablica kojoj odgovara optimalno bazično rešenje. Ako postoji $k \in \{1, \ldots, m\}$ tako da je $b_k < 0$ i $a_{k1} \geq 0, \ldots, a_{kn} \geq 0$, pokazali smo da

je skup dopustivih rešenja prazan. Ako postoji $a_{ki} < 0$ moguće je primeniti dualni simpleks metod.

Neka je dat problem linearnog programiranja

$$\begin{array}{ll} \min & d + c^T x, \\ \text{p.o.} & Ax = b, \\ & x \geq 0, \end{array}$$

kome odgovara početna dualna simpleks tablica DT_0 :

$$\begin{array}{cccc} -d^0 & c_1^0 & \cdots & c_n^0 \\ b_1^0 & a_{11}^0 & \cdots & a_{1n}^0 \\ \vdots & \vdots & & \vdots \\ b_m^0 & a_{m1}^0 & \cdots & a_{mn}^0 \end{array}$$

Sada su ispunjeni uslovi za primenu dualnog simpleks metoda po sledećem algoritmu:

Algoritam 4. (Dualni simpleks metod).
Staviti $k = 0$; $k-$ta iteracija se satoji opd sledećih koraka

Korak 1. Ako je $b_i^k \geq 0$ za sve $i = 1, \ldots, m$, algoritam staje, jer je bazično dopustivo rešenje optimalno.

Korak 2. Za svako i za koje je $b_i^k < 0$ ispitati da li je $a_{ij}^k \geq 0$ za sve $j = 1, \ldots, n$. Ako takvo j postoji, algoritam staje jer je dopustivi skup prazan.

Korak 3. Odrediti $s \in \{1, \ldots, m\}$ za koje je $b_s^k < 0$.
Odrediti $r \in \{1, \ldots, m\}$ takvo da je $\dfrac{c_r^k}{a_{sr}^k} = \max\left\{ \dfrac{c_j^k}{a_{sj}^k} | a_{sj}^k < 0 \right\}$.

Korak 4. Primeniti na DT_k sledeće elementarne transformacije:
- pomnožiti $s-$tu vrstu sa $-a_{ir}^k/a_{sr}^k$ i dodati vrstama $i = 0, \ldots, m$, $i \neq s$;
- podeliti $s-$tu vrstu sa a_{sr}^k.

Korak 5. Zameniti k sa $k + 1$ i ići na Korak 1.

Dualni simpleks algoritam je maksimizacioni za ciljnu funkciju, što sledi iz sledeće teoreme:

Teorema 2.9.1 *Primenom elementarnih transformacija iz Koraka 4 Algoritma 4 se od dualne simpleks tablice DT_k dobija dualna simpleks tablica DT_{k+1} koja odgovara ekvivalentnom problemu linearnog programiranja. Pri tome je $d^{k+1} \geq d^k$.*

Dokaz. Očigledno je da se primenom elementarnih transformacija dobija LP-tablica DT_{k+1} koja odgovara ekvivalentnom problemu linearnog programiranja. Pri tome sve bazične kolone koje su u $s-$toj vrsti imale nule ostaju bazične, a $r-$ta

kolona postaje bazična umesto one koja je u $s-$toj vrsti imala jedinicu. Za $j = 1, \ldots, n$ je

$$c_j^{k+1} = c_j^k - \frac{a_{sj}^k}{a_{sr}^k} c_r^k.$$

Ako je $a_{sj}^k \geq 0$, iz $c_r^k \geq 0$ i $a_{sr}^k < 0$ sledi $c_j^{k+1} \geq c_j^k \geq 0$. Ako je $a_{sj}^k < 0$, na osnovu izbora r u Koraku 3 sledi

$$c_j^{k+1} = a_{sj}^k \left(\frac{c_j^k}{a_{sj}^k} - \frac{c_r^k}{a_{sr}^k} \right) \geq 0.$$

Stoga je DT_{k+1} dualna simpleks tablica. Iz $c_r^k b_s^k / a_{sr}^k \geq 0$, sledi

$$-d^{k+1} = -d^k - \frac{b_s^k}{a_{sr}^k} c_r^k \leq -d^k \quad \Leftrightarrow \quad d^{k+1} \geq d^k$$

čime je dokaz kompletiran. □

Primer 2.9.1 Neka je dat problem

$$\begin{array}{lrrrrrrrr}
\min & 3x_1 + 2x_2, & & & & & & & \\
 & -x_1 & +3x_2 & +x_3 & & & & = & -1, \\
 & -2x_1 & -10x_2 & & +x_4 & & & = & -10, \\
\text{p.o.} & 2x_1 & +4x_2 & & & +x_5 & & = & 8, \\
 & 3x_1 & -5x_2 & & & & +x_6 & = & 6, \\
 & x \geq 0. & & & & & & &
\end{array}$$

kome odgovara dualna simpleks tablica

$$\begin{array}{rrrrrrr}
0 & 3 & 2 & 0 & 0 & 0 & 0 \\
-1 & \boxed{-1} & 3 & 1 & 0 & 0 & 0 \\
-10 & -2 & -10 & 0 & 1 & 0 & 0 \\
8 & 2 & 4 & 0 & 0 & 1 & 0 \\
6 & 3 & -5 & 0 & 0 & 0 & 1
\end{array}$$

Primenom Algoritma 4 sa uokvirenim pivot elementima, dobijamo

$$\begin{array}{rrrrrrr}
-3 & 0 & 11 & 3 & 0 & 0 & 0 \\
1 & 1 & -3 & -1 & 0 & 0 & 0 \\
-8 & 0 & \boxed{-16} & -2 & 1 & 0 & 0 \\
6 & 0 & 10 & 2 & 0 & 1 & 0 \\
3 & 0 & 4 & 3 & 0 & 0 & 1
\end{array}$$

odakle odmah sledi dualna simpleks tablica

$$\begin{array}{rrrrrrr}
-17/2 & 0 & 0 & 13/8 & 11/16 & 0 & 0 \\
5/2 & 1 & 0 & -5/8 & -3/16 & 0 & 0 \\
1/2 & 0 & 1 & 1/8 & -1/16 & 0 & 0 \\
1 & 0 & 0 & 3/4 & 10/16 & 1 & 0 \\
1 & 0 & 0 & 5/2 & 1/4 & 0 & 1
\end{array}$$

Odgovarajuće bazično dopustivo rešenje je $(5/2, 1/2, 0, 0, 1, 1)$ je optimalno i optimalna vrednost ciljne funkcije je $17/2$. Primetimo da su u ovom slučaju koraci dualnog algoritma identični sa Algoritmom 3.

Napomenimo da je Algoritam 4 moguće primeniti ako je poznata početna dualna simpleks tablica. Ukoliko to nije slučaj, dualnu simpleks tablicu možemo konstruisati primenom sledećeg algoritma:

Algoritam 5. (Dualni algoritam za proizvoljnu simpleks tablicu).

Korak 1. Formiramo početnu LP-tablicu oblika

$$\begin{array}{cccc} -d & c_1 & \cdots & c_n \\ b_1 & a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & & \vdots \\ b_m & a_{m1} & \cdots & a_{mn} \end{array}$$

Ukoliko je $c_1, \ldots, c_n \geq 0$, primenimo Algoritam 4.

Korak 2. Ako je $c_j < 0$ za neko j i $a_{1j}, \ldots, a_{mj} \leq 0$, algoritam staje jer je skup ograničenja nedopustiv.

Korak 3. Neka je $c_{j_1}, \ldots, c_{j_q} < 0$.

Ukoliko je $q = n$ biramo $a_{in} < 0$ za stožerni elemet, primenimo simpleks transformaciju i prelazimo na Korak 1.

Ako je $q < n$ i ako postoji $p \in \{j_1, \ldots, j_q\}$ takvo da je ispunjen sledeći uslov

$$\min_k \left\{ \frac{c_k}{a_{ik}} | \, a_{ik} > 0, \, c_k \geq 0 \right\} \geq \frac{c_p}{a_{ip}} \geq 0, \tag{2.9.1}$$

biramo $a_{ip} < 0$ za stožerni element, primenimo simpleks transformaciju i prelazimo na Korak 1. Ukoliko postoji više elemenata $a_{ip} < 0$ koji zadovoljvaju uslov (2.9.1) biramo onaj za koji eventualno važi

$$\min \left\{ \frac{b_h}{a_{hp}} | \, b_h \geq 0, \, a_{hp} > 0 \right\} \geq \frac{b_i}{a_{ip}}, \quad b_i < 0.$$

Ukoliko uslov 2.9.1 nije ispunjen neka je

$$\min_k \left\{ \frac{c_k}{a_{ik}} | \, c_k \geq 0, \, a_{ik} > 0 \right\} = \frac{c_r}{a_{ir}}.$$

Za tako određeno r postavimo a_{ir} za stožerni element, primenimo simpleks transformaciju i prelazimo na Korak 1.

Napomenimo da nijednoj simpleks tablici generisanoj dualnim alogoritmom ne odgovara dopustivo rešenje primalnog problema osim poslednjoj koja je istovremeno simpleks tablica. Pokazaćemo da svakoj dualnoj simpleks tablici odgovara dopustivo rešenje dualnog problema. Neka je u dualnoj simpleks tablici DT_k skup indeksa bazičnih kolona označen sa B i neka je odgovarajuća baza $\mathcal{A}_B$. Prenumeracijom kolona u pridruženoj matrici A_B možemo postići da DT_k odgovara sledećem problemu linearnog programiranja

$$\begin{array}{ll} \min & c_B^T A_B^{-1} b + (c_N^T - c_B^T A_B^{-1} A_N) x_N, \\ \text{p.o.} & x_B + A_B^{-1} A_N x_N = A_B^{-1} b, \\ & x_B \geq 0, \quad x_N \geq 0. \end{array} \tag{2.9.2}$$

Problem (2.9.2) je kanonski oblik standardnog problema linearnog programiranja u odnosu na bazu $\mathcal{A}_B$ za koju važi $c_N^T - c_B^T A_B^{-1} A_N \geq 0$. Definišemo (y^*, s^*) sa $y^* = (A_B^{-1})^T c_B$, $s_B^* = 0$, $s_N^* = c_N - A_N^T y^*$. Tada je

$$\begin{aligned} A_B^T y^* + s^* &= c_B, \qquad & A_N^T y^* + s^* &= c_N \\ s_B^* &\geq 0, & s_N^* &\geq 0, \end{aligned}$$

tj. (y^*, s^*) je dopustivo rešenje dualnog problema. Dakle, svakoj dualnoj simpleks tablici odgovara dopustivo rešenje duala. Koordinate vektora s^* su u tablici date eksplicitno a vektor y^* se računa iz sistema $A_B^T y^* = c_B$.

2.10 Eliminacija jednačina i slobodnih promenljivih

Još na početku, kada smo definisali oblike problema linearnog programiranja, pokazali smo kako se opšti oblik problema linearnog programiranja svodi na standardni, odnosno simetrični oblik. Ovde ćemo pokazati kako se mogu iskoristiti Tuckerove tabele i zamena promenljivih (algoritam **Replace**) za svođenje opšteg problema linearnog programiranja na kanonski oblik. Znači, posmatrajmo sada opšti oblik problema linearnog programiranja

$$\begin{aligned} \max\ & z(x) = c_1 x_1 + \ldots + c_n x_n + d \\ \text{p.o.}\ & N_i^{(1)}: \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, p \\ & N_i^{(2)}: \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = p+1, \ldots, q \\ & J_i: \quad \sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = q+1, \ldots, m \\ & x_j c \geq 0,\ j \in \mathcal{J} = \{1, \ldots, s\}, \quad s \leq n. \end{aligned}$$

Pomnožimo sve nejednačine tipa $N^{(2)}$ sa -1 i uvedimo dodatne promenljive $x_{n+1}, \ldots, x_{n+q}$ kao kod svođenja na standardni oblik. Formalno, označimo promenljive sa leve strane tabele sa $x_{B,i}$ a one sa desne $x_{N,j}$. Ovim smo dobili problem u obliku koji podseća na kanonski.

$$\begin{aligned} \max\ & z(x) = c_1 x_{N,1} + \ldots + c_n x_{N,n} + d \\ \text{p.o.}\ & a_{11} x_{N,1} + a_{12} x_{N,2} + \ldots + a_{1n} x_{N,n} - b_1 = -x_{B,1} \\ & a_{21} x_{N,1} + a_{22} x_{N,2} + \ldots + a_{2n} x_{N,n} - b_2 = -x_{B,2} \\ & \ldots\ldots\ldots \\ & a_{q1} x_{N,1} + a_{q2} x_{N,2} + \ldots + a_{qn} x_{N,n} - b_m = -x_{B,q} \\ & a_{q+1,1} x_{N,1} + a_{q+1,2} x_{N,2} + \ldots + a_{q+1,n} x_{N,n} - b_{q+1} = -0 \\ & \ldots\ldots\ldots \\ & a_{m1} x_{N,1} + a_{m2} x_{N,2} + \ldots + a_{mn} x_{N,n} - b_m = -0 \end{aligned} \tag{2.10.1}$$

Razlika je samo u jednačinama $q+1,\ldots,m$. Ako pretpostavimo da su njima dodeljene dodatne promenljive koje su identički jednake nuli, dobijamo problem u kanonskom obliku. Ekvivalentna Tuckerova tabela ima sledeći izgled:

$$\begin{array}{ccccccc} x_{N,1} & x_{N,2} & \cdots & x_{N,n} & -1 & & \\ a_{11} & a_{12} & \cdots & a_{1n} & b_1 & = & -x_{B,1} \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 & = & -x_{B,2} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{q1} & a_{q2} & \cdots & a_{qn} & b_q & = & -x_{B,q} \\ a_{q+1,1} & a_{q+1,2} & \cdots & a_{q+1,n} & b_{q+1} & = & -x_{B,q+1} \equiv 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m & = & -x_{B,m} \equiv 0 \\ c_1 & c_2 & \cdots & c_n & d & = & z \end{array} \tag{2.10.2}$$

Posmatrajmo sada jednu takvu jednačinu (npr. poslednju).

$$a_{m1}x_{N,1} + a_{m2}x_{N,2} + ... + a_{mn}x_{N,n} - b_m = -x_{B,m} = -0$$

Ukoliko su svi $a_{q+1,j} = 0$ onda je ova jednačina ili nemoguća, pa je problem nedopustiv, ili identitet, pa se može jednostavno izbaciti iz problema. Neka je sada $a_{mj} \neq 0$ za neko j. Ukoliko zamenimo promenljive $x_{B,m}$ i $x_{N,j}$ dobijamo:

$$\begin{array}{ccccccc} x_{N,1} & \cdots & x_{B,m} \equiv 0 & \cdots & x_{N,n} & -1 & & \\ a_{11} & \cdots & a_{1j} & \cdots & a_{1n} & b_1 & = & -x_{B,1} \\ \vdots & & \vdots & & \vdots & \vdots & & \vdots \\ a_{m1} & \cdots & a_{mj} & \cdots & a_{mn} & b_m & = & -x_{N,j} \\ c_1 & \cdots & c_j & \cdots & c_n & d & = & z \end{array} \tag{2.10.3}$$

Očigledno, sada nezavisna promenljiva $x_{B,m} \equiv 0$ ne utiče ni na funkciju cilja ni na uslove. Sledi da j-tu kolonu tabele (2.10.3) možemo izbaciti. Time smo broj nebazičnih promenljivih smanjili za 1. Ovaj postupak ponavljamo sve dok bazične promenljive koje su identične nuli ne eliminišemo u potpunosti. Može se opisati sledeći algoritam.

Algoritam ElJed (Eliminacija jednačina)

Korak 1. Ako ne postoji nijedna bazična promenljiva identički jednaka nuli, primeniti algoritam **NoBasicMax**.

Korak 2. Neka je $x_{B,i} \equiv 0$. Nađimo $a_{ij} \neq 0$. Ako takvo ne postoji, i ako je $b_i = 0$, izbaciti i-tu jednačinu i preći na korak 1, u suprotnom STOP. Problem je nedopustiv.

Korak 3. Zameniti promenljive $x_{B,i}$ i $x_{N,j}$, izbaciti j-tu kolonu, smanjiti n za 1, i preći na korak 1.

Na sličan način vršimo eliminaciju slobodnih promenljivih. Ako je neka od bazičnih promenljivih slobodna, npr. $x_{B,1}$, njenu vrednost uvek možemo izračunati iz jednačine:

$$a_{11}x_{N,1} + a_{12}x_{N,2} + ... + a_{1n}x_{N,n} - b_1 = -x_{B,1}$$

za proizvoljne vrednosti nebazičnih promenljivih. Znači, ova jednačina je uvek zadovoljena, te može biti izbačena iz problema.

Pretpostavimo da smo na ovaj način eliminisali sve bazične slobodne promenljive. Takođe, pretpostavimo da je neka nebazična promenljiva (npr. $x_{N,j}$) slobodna. Neka je najpre $a_{ij} = 0$ za svako $i = 1, \ldots, m$. Ako je $c_j \neq 0$ onda je funckija cilja neograničena ako je problem dopustiv. U suprotnom, promenljiva $x_{N,j}$ nema uticaja ni na funkciju cilja, pa može biti izbačena.

Neka je sada $a_{ij} \neq 0$ za neko $i \in \{1, \ldots, m\}$. Izvršimo zamenu promenljivih $x_{B,i}$ i $x_{N,j}$. Posle transformacije, promenljiva $x_{N,j}$ postaje bazična, pa se eliminiše kao u prethodnom slučaju. Prema tome imamo:

Algoritam ElSl (Eliminacija slobodnih promenljivih)

Korak 1. Ukoliko nema slobodnih promenljivih, primeniti algoritam **ElJed**.

Korak 2. Neka je promenljiva $x_{B,i}$ slobodna. Izbaciti i-tu jednačinu (vrstu) i preći na korak 1.

Korak 3. Neka je promenljiva $x_{N,j}$ slobodna. Ako je $a_{ij} = 0$ za svako $i = 1, \ldots, m$, preći na korak 2'. U suprotnom izabrati $a_{ij} \neq 0$ i preći na korak 4.

Korak 3'. Ako je $c_j = 0$ izbaciti j-tu kolonu, smanjiti n za 1 i preći na korak 1. U suprotnom, izbaciti j-tu kolonu, preći na korak 1 i nastaviti sa algoritmom sve dok se ne utvrdi da li je problem dopustiv ili nije. Ako je dopustiv, STOP. Funkcija cilja je neograničena.

Korak 4. Zameniti promenljive $x_{B,i}$ i $x_{N,j}$. Sada $x_{N,j}$ postaje bazična slobodna promenljiva, eliminisati je kao u koraku 2.

Sada je uveden kompletan algoritam koji rešava polazni problem linearnog programiranja u opštem obliku. Na sledećem primeru ćemo pokazati kako se istovremeno eliminišu jednačine i slobodne promenljive.

Primer 2.10.1 Neka je data sledeća tabela:

$$T_0 = \begin{array}{cccccc} x_1 & \boxed{x_2} & x_3 & -1 & & \\ 1 & 1 & 1 & 6 & = & -0 \\ 1 & 1 & 0 & 1 & = & -x_4 \\ 1 & 2 & 1 & 0 & = & z \end{array}$$

Uokvirena promenljiva je slobodna. Odaberimo a_{12} za pivot element. Posle zamene promenljivih dobija se tabela T_1. Izbacivanjem vrste koja odgovara slobodnoj promenljivoj i kolone koja odgovara nuli, dobija se ekvivalentna tabela T_1'.

$$T_1 = \begin{array}{cccccc} x_1 & 0 & x_3 & -1 & & \\ 1 & 1 & 1 & 6 & = & -\boxed{x_2} \\ 0 & -1 & -1 & -5 & = & -x_4 \\ -1 & -2 & -1 & -12 & = & z \end{array} \qquad T_1' = \begin{array}{cccccc} x_1 & x_3 & -1 & & \\ 0 & -1 & -5 & = & -x_4 \\ -1 & -1 & -12 & = & z \end{array}$$

Primetimo da ako sada izaberemo (prema algoritmu **NoBasicMax**) a_{12} za pivot, dobijamo optimalno rešenje.

2.11 Revidirani Simpleks metod

Simpleks metod radi tako što u svakom koraku vrši zamenu promenljivih sa ciljem povećanja (smanjenja) funkcije cilja ili dobijanja dopustivog rešenja. U svakoj iteraciji vrši se zamena promenljivih, pri čemu se računaju nove vrednosti elemenata Tuckerove tabele. Prilikom te popravke, vrši se niz deljenja, što dovodi do nagomilavanja numeričke greške. Zbog ovoga simpleks metod greši u primerima u kojima je potreban veliki broj iteracija. Da bi se to izbeglo, potrebno je da se elementi Tuckerove tabele računaju pomoću polazne matrice problema. To se postiže revidiranim simpleks metodom.

Posmatramo problem linearnog programiranja u standardnom obliku (1.1.4). Neka je dato jedno bazično rešenje (dobijeno rešavanjem sistema $Ax = b$ ili metodom eliminacije jednačina) x^*. Posmatrajmo sada polazni problem u kanonskom obliku:

$$\begin{array}{ll} \max & (c^*)^T x_N - d \\ p.o. & Tx_N - b^* = -x_B \\ & x = (x_B, x_N) \geq 0 \end{array} \tag{2.11.1}$$

Označimo sada sa n i m redom broj nebazičnih i bazičnih promenljivih a sa T matricu tipa $m \times n$ koja predstavlja Tuckerovu tabelu. Vektori b^* i c^* su odgovarajući slobodni vektor i vektor funkcije cilja u kanonskom obliku. Sistem jednačina u (2.11.1) može se drugačije napisati kao $[T|I_m] \begin{bmatrix} x_N \\ x_B \end{bmatrix} = b^*$. Napišimo sada sistem $Ax = b$ u obliku $A_B x_B + A_N x_N = b$. Iz ovog uslova i (2.11.1) dobijamo direkno da važi:

$$T = A_B^{-1} A_N \qquad b^* = A_B^{-1} b. \tag{2.11.2}$$

Znači, ako znamo indekse bazičnih promenljivih $v_{B,1}, \ldots v_{B,m}$ ($x_{B,i} = x_{v_{B,i}}$) možemo da rekonstruišemo Tuckerovu tabelu pomoću formula (2.11.2). Pri tome je ispunjeno $A_B = [K_{v_{B,1}} \cdots K_{v_{B,m}}]$.

Funkciju cilja $f(x) = c^T x$ ovde možemo tretirati kao jednačinu, pri čemu moramo uvesti dodatnu promenljivu x_{n+1} tako da važi $c^T x + x_{n+1} = 0$. Sada matrica sistema ima oblik $A_c = \begin{bmatrix} A & 0 \\ c^T & 1 \end{bmatrix}$ pri čemu je $x_c = (x_1, \ldots, x_{n+1})$. Takođe je

$$(A_c)_B = \begin{bmatrix} A_B & 0 \\ c_B & 1 \end{bmatrix}.$$

Znači, ako sada primenimo formule (2.11.2) za prošireni sistem

$$A_c x_c = \begin{bmatrix} b^* \\ d \end{bmatrix},$$

u potpunosti možemo da rekonstruišemo Tuckerovu tabelu.

Kao što možemo videti, kod upravo izloženog metoda rekonstrukcije Tuckerove tabele nema nagomilavanja numeričkih grešaka, pa je ovaj metod stabilniji od

klasičnog simpleks metoda. Međutim, primetimo da u svakom koraku moramo da računamo inverznu matricu, što čini da jedna iteracija revidiranog simpleks metoda bude znatno algoritamski kompleksnija od odgovarajuće iteracije klasičnog simpleksa.

Kod većine test primera (a i problema u praksi koji se svode na linearno programiranje) matrice sistema su veoma retke (*sparse*), tj. mali je broj nenula elemenata. Na žalost, inverzne matrice *sparse* matrica u opštem slučaju ne moraju biti *sparse*. Moguće je čak konstruisati primer *sparse* matrice relativno malih dimenzija čiji inverz nema ni jednu nulu, a u većini slučajeva broj nula je veoma mali. Metodi za inverziju matrica rade veoma dugo, a inverzna matrica zahteva mnogo prostora za pamćenje.

Takođe, u algoritmima simpleks metode nije potrebno da znamo celu Tuckerovu tabelu. Dovoljno je da znamo (da rekonstruišemo) pojedine redove ili kolone Tuckerove tabele. U nastavku ćemo opisati metod kod koga smo izračunavanje potrebnog dela Tuckerove tabele sveli samo na rešavanje sistema linearnih jednačina. Označimo j-tu kolonu matrice A sa $A_{\bullet j}$ a i-ti red sa $A_{i\bullet}$. Da bi rekonstruisali j-tu kolonu dovoljno je da odgovarajuću kolonu polazne matrice pomnožimo sa A_B^{-1} odnosno treba rešiti sistem $A_B T_{\bullet j} = K_{v_{N,j}}$. Ovde moramo rešiti onoliko sistema koliko je potrebno kolona rekonstruisati. U algoritmu **NoBasicMax** potrebna nam je samo jedna kolona i vektor b. Za rekonstrukciju i-tog reda potrebno nam je da znamo i-ti red matrice A_B^{-1} odnosno [7, 58, 73]:

$$T_{i\bullet} = (A_B^{-1})_{i\bullet} A_N \tag{2.11.3}$$

Vektor $(A_B^{-1})_{i\bullet}$ određujemo iz sledeće jednačine:

$$(A_B^{-1})_{i\bullet} A_B = (I_m)_{i\bullet} \Longrightarrow A_B^T (A_B^{-1})_{i\bullet}^T = (I_m)_{i\bullet}^T \tag{2.11.4}$$

što takođe predstavlja sistem linearnih jednačina koji treba rešiti.

Napomenimo da ako za rešavanje sistema linearnih jednačina (2.11.3) i (2.11.4) koristimo metode kao što su Gausova eliminacija, LR faktorizacija, itd. [48] dovoljno je jednom da odradimo potrebne transformacije matrice A_B a zatim samo da rekonstruišemo rešenja za sve potrebne RHS vektore. U algoritmu **BasicMax** potrebno nam je da rekonstruišemo samo zadnji red (funkciju cilja), što znači da je u svakoj iteraciji algortitma **BasicMax** potrebno da rešimo 3 sistema linearnih jednačina. U algoritmu **NoBasicMax** u svakoj iteraciji potrebno je rekonstruisati jedan red i dve kolone. Kao što možemo videti, osim velike uštede u memoriji (rekonstrukciju pomoću inverzne matrice zbog pomenutog problema nema smisla implementirati pomoću *sparse* reprezentacije matrica) dobija se i na vremenu.

Formulišimo sada algoritme **BasicMax** i **NoBasicMax** "jezikom" revidiranog simpleks metoda.

Algoritam RevBasicMax (Revidirani simpleks metod za bazično dopustive kanonske oblike [73]) Formirati matricu $(A_c)_B = \begin{bmatrix} A_B & 0 \\ c_B & 1 \end{bmatrix}$ gde je c_B vektor koji se sastoji od koeficijenata funkcije cilja uz bazične promenljive.

Korak 1. Rešiti sisteme $(A_B)_c b^* = b$ i $(A_B)_c^T((A_B)_c^{-1})_{m+1\bullet}^T = (I_{m+1})_{m+1\bullet}^T$, rekonstruisati $n+1$-vi red Tuckerove tabele $c^* = ((A_c)_B^{-1})_{n+1\bullet} N_c$.

Korak 2. Ako je $c^* \leq 0$ funkcija cilja ima ekstremum. U suprotnom neka je $c_j^* > 0$

Korak 3. Rekonstruisati j-tu kolonu matrice T_c , tj rešiti sistem $A_B T_{\bullet j} = K_{v_{N,j}}$. Ako je $T_{\bullet j} \leq 0$, STOP. Funkcija cilja je neograničena.

Korak 4. Izračunati:

$$\min_{1\leq i\leq m}\left\{\frac{b_i^*}{T_{ij}} \mid T_{ij} > 0\right\} = \frac{b_p^*}{T_{pj}}$$

Korak 5. Izbaciti iz baze promenljivu $x_{B,j}$ (odnosno vektor $K_{v_{B,j}}$) a ubaciti promenljivu $x_{N,j}$ (tj vektor $K_{v_{B,j}}$). Drugim rečima, izvršiti zamenu vrednosti $v_{B,p}$ i $v_{N,j}$. Preći na korak 2.

Sledeći algoritam nije pronađen u literaturi, tako da je delimično i originalan. Međutim, korišćenje predhodnog alogritma bez sledećeg je besmisleno, jer se već u algoritmu **NoBasicMax** numeričke greške nagomilavaju.

Algoritam RevNoBasicMax (Revidirani simpleks metod za kanonske oblike koji nisu bazično dopustivi)

Korak 1. Neka je početna bazična matrica B.

Korak 2. Rešiti sistem $(A_B)_c b^* = b$. Ako je $b^* \geq 0$, primeniti algoritam **RevBasicMax**. U suprotnom izabrati $b_i^* < 0$ tako da je i maksimalno.

Korak 3. Rekonstruisati i-tu vrstu $T_{i\bullet}$ matrice T (Tuckerove tabele). Ako je $T_{i\bullet} \leq 0$, STOP. Problem linearnog programiranja je nedopustiv. U suprotnom izabrati $T_{ij} < 0$.

Korak 4. Ako je $i = m$ zameniti promenljive $x_{B,i}$ i $x_{N,j}$ i preći na korak 2.

Korak 5. U rekonstruisati j-tu kolonu matrice T, tj rešiti sistem $A_B T_{\bullet j} = K_{v_{N,j}}$

Korak 6. Izračunati

$$\min_{l>i}\left(\left\{\frac{b_i^*}{T_{ij}}\right\}\bigcup\left\{\frac{b_l^*}{T_{lj}} \mid a_{lj} > 0\right\}\right) = \frac{b_p^*}{T_{pj}}$$

Zameniti promenljive $x_{B,p}$ i $x_{N,j}$ i preći na Korak 2.

Analogno se mogu opisati i algoritmi za eliminaciju jednačina i slobodnih promenljivih. Na kraju, još jednom pomenimo prednosti i nedostatke revidiranog simpleks metoda.

Prednosti:

- Elementi Tuckerove tabele T računaju se direkno na osnovu matrice A čime se izbegava nagomilavanje greške računskih operacija.
- Pošto nam u algoritmima simpleks metode nije potrebna cela matrica T već samo pojedini redovi i kolone, revidiranom simpleks metodom mi rekonstruišemo samo te redove i kolone a ne celu matricu T.

Nedostaci:

- S obzirom da u svakom koraku moramo ili da tražimo inverznu matricu ili da rešavamo nekoliko (maksimalno 3) sistema linearnih jednačina, iteracija revidiranog simpleksa je znatno sporija od iteracije običnog.
- Kod običnog simpleksa smo koristili jednostavan algoritam za zamenu promenljivih. Ovde je potrebno implementirati kompleksne algoritme za rešavanje sistema linearnih jednačina ili inverziju matrica.

2.12 Pojam cikliranja i anticiklična pravila

U Teoremi 2.4.1 pokazali smo da posle svake iteracije algoritma **BasicMax** vrednost funkcije cilja se ili povećava ili ostaje ista. Na taj način smo "dokazali" da se isti algoritam završava u konačno mnogo iteracija (pošto ima konačno mnogo bazično dopustivih rešenja). Pri tome, nismo razmatrali mogućnost da se vrednost funkcije cilja ne menja tokom rada algoritma **BasicMax**. U tom slučaju, nemamo garancije da će se algoritam **BasicMax** završiti u konačnom vremenu. Da je to stvarno moguće pokazuje sledeći primer [84]:

Primer 2.12.1 Posmatrajmo sledeći problem linearnog programiranja u simetričnom obliku:

$$\begin{aligned} \max\ & z = \frac{3}{4}x_1 - 20x_2 + \frac{1}{2}x_3 - 6x_4 \\ \text{p.o.}\ & \frac{1}{4}x_1 - 8x_2 - x_3 + 9x_4 \leq 0 \\ & \frac{1}{2}x_1 - 12x_2 - \frac{1}{2}x_3 + 3x_4 \leq 0 \\ & x_3 \leq 1 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned} \tag{2.12.1}$$

Formirajmo odgovarajući kanonski oblik, Tuckerovu tabelu i primenimo algoritam **BasicMax** 7 puta:

$$\begin{array}{cccccccc} x_1 & x_2 & x_3 & x_4 & -1 & & \\ \frac{1}{4} & -8 & -1 & 9 & 0 & = & -x_5 \\ \frac{1}{2} & -12 & -\frac{1}{2} & 3 & 0 & = & -x_6 \\ 0 & 0 & 1 & 0 & 1 & = & -x_7 \\ \frac{3}{4} & -20 & \frac{1}{2} & -6 & 0 & = & z \end{array} \tag{2.12.2}$$

Ukoliko bi nastavili sa primenom algoritma **BasicMax**, dobijenih 6 tabela bi se stalno ponavljale i nikad ne bi došli do optimalnog rešenja. Primetimo da u koracima 2,3 i 4 izbor elementa ne mora biti jedinstven.

Upravo ovo je glavni razlog pojave cikliranja. Cikliranje se najčešće tretira kao retka pojava koja se javlja samo u veštački konstruisanim primerima [84, 79]. Međutim još 1977. godine su Kotiah i Steinberg [41] otkrili čitavu klasu "nevештačkih" problema koji cikliraju. Takođe, kako ćemo kasnije videti, prilikom testiranja naših programa uočili smo pojavu cikliranja na više primera.

U ovom odeljku ćemo proučiti pravila kojima se sprečava cikliranje. Ta pravila se nazivaju ***anticiklična pravila***. Obradićemo dve vrste anticikličnih pravila: ***Leksikografski metod*** i ***Blandova pravila***.

Definišimo ***leksikografsko pravilo*** protiv cikliranja.

Za vektor $x \in \mathbb{R}^n$, $x \neq 0$ kažemo da je *leksikografski pozitivan* ($x \overset{lex}{>} 0$) ako je njegova prva koordinata različita od nule pozitivna. Ako je $x = 0$ kažemo da je x *leksikografska nula* ($x \overset{lex}{=} 0$), a za x kažemo da je *leksikografski negativan* ($x \overset{lex}{<} 0$) ako je $-x \overset{lex}{>} 0$. Za vektor $y \in \mathbb{R}^n$ kažemo da je *leksikografski veći* od x ako je $y - x \overset{lex}{>} 0$. Analogno se uvode pojmovi *leksikografski manji od* i *leksikografski jednak*. Oznaka $\hat{x} = \text{lex-min}\, X$ znači $\hat{x} \in Z$ i $\hat{x} \overset{lex}{\leq} x$, $x \in X$. Analogno se uvodi i oznaka lex-max.

Kako je u Tuckerovoj tabeli $b_i = a_{i,n+1}$ i $c_j = a_{m+1,j}$, Korak 3 u simpleks metodu (algoritmu **BasicMax**) može se zapisati i na ovakav način:

- *Korak 3. Odrediti* $j \in \{1, \ldots, n\}$ *za koje je* $c_j < 0$. *Odrediti* $p \in \{1, \ldots, m\}$ *takvo da je*
$$\frac{a_{p,n+1}}{a_{pj}} = \min\{\frac{a_{i,n+1}}{a_{ij}} \mid a_{ij} > 0\}.$$

Posmatrajmo sada ponovo odgovarajući standardni oblik 2.1.2, pri čemu ćemo matrici A' dodati vektor b' kao nultu kolonu. Cikliranje se izbegava ako se u Koraku 3 odabere p tako da se umesto minimuma postiže leksikografski minimum. Modifikovani korak je:

- *Korak 3'. Odrediti* $j \in \{1, \ldots, n\}$ *za koje je* $c_j < 0$. *Odrediti* $p \in \{1, \ldots, m\}$ *takvo da je*
$$\frac{V_p}{a_{pj}} = \textit{lex-min}\left\{\frac{V_i}{a_{ij}} \mid a_{ij} > 0\right\},$$
gde je, podsetimo se, sa V_i *označena* i*-ta vrsta* $A'_{i\bullet}$ *matrice* A'.

Primer 2.12.2 *U slučaju Tuckerove tabele iz primera sa početka ovog odeljka, uz isto pravilo za izbor indeksa* j *sledi da je* $j = 1$, $a_{ij} > 0$ *za* $i = 1, 2$, i

$$\frac{V_1}{a_{1j}} = [0, 1, -32, -4, 36, 4, 0, 0], \quad \frac{V_2}{a_{2j}} = [0, 1, -24, -1, 6, 0, 2, 0],$$

pa se lex-min postiže za $p = 1$, *pa je prvi pivot element* a_{11}. *Primenom leksikografskog pravila se dalje dobijaju pivot elementi* $a^1_{22}, a^2_{23}, a^3_{34}$, *koji vode do Tuckerove tabele* T_4 *sa novom vrednošću ciljne funkcije i cikliranje se izbegava.*

Napomena 2.12.1 *Važno je primetiti da je lex-min uvek jedinstven jer iz pretpostavke da je* $\text{rang}A' = m$ *sledi da ne postoje dve proporcionalne vrste.*

Dokažimo sada korektnost leksikografskog pravila.

Teorema 2.12.1 *Neka su u početnoj matrici* A' *sve vrste* $V_1, \ldots, V_m$ *leksikografski pozitivne i neka je u algoritmu* **BasicMax** *korak 3 zamenjen korakom 3'. Tada je cikliranje eliminisano, tj. simpleks metod u konačnom broju iteracija dolazi ili do optimalnog rešenja ili do zaključka da ciljna funkcija nije ograničena odozdo.*

Dokaz. Pokazaćemo prvo da vrste V_i^k, $i = 1, \ldots, m$, matrice A' ostaju leksikografski pozitivne. Za $i = p$ je $V_p^1 = \frac{V_p}{a_{pj}}$ pa $a_{pj} > 0$ i $V_p \overset{lex}{>} 0$ povlači $V_p \overset{lex}{>} 0$. Za $i \neq p$ je

$$V_i^1 = V_i - \frac{a_{ij}}{a_{pj}} V_p = a_{ij} \left(\frac{V_i}{a_{ij}} - \frac{V_p}{a_{pj}} \right) \overset{lex}{>} 0,$$

gde stroga nejednakost važi na osnovu prethodne napomene. Za $i \neq p$ i $a_{ij}^k \leq 0$ je $V_i^1 = V_i + \frac{|a_{ij}|}{a_{pj}} V_p \overset{lex}{\geq} V_i \overset{lex}{>} 0$. Za $m+1$-vu vrstu važi:

$$V_{m+1}^1 = V_{m+1} - \frac{a_{m+1,j}}{a_{pj}} V_p = V_{m+1} + \frac{|a_{m+1,j}|}{a_{pj}} V_p \overset{lex}{>} V_{m+1}$$

zbog

$$a_{m+1,j} < 0,\ a_{pj} > 0 \quad \text{i} \quad V_p \overset{lex}{>} 0.$$

Dakle, $m+1$-va vrsta strogo leksikografski raste i ne može se ponoviti. □

Napomena 2.12.2 *Pretpostavka o leksikografskoj pozitivnosti vrsta matrice A' u prethodnoj teoremi nije ograničavajuća jer se to može postići u svakom kanonskom obliku linearnog programiranja. Dovoljno je, na početku, izvršiti prenumeraciju promenljivih i dovesti kolone jedinične matrice na pozicije $1, \ldots, m$.*

Ako u Algoritmu 4 umesto Koraka 3 zapisanog u obliku:

Korak 3. Odrediti $s \in \{1, \ldots, m\}$ za koje je $a_{s0}^k < 0$. Odrediti $r \in \{1, \ldots, m\}$ takvo da je $\frac{a_{0r}^k}{a_{sr}^k} = \max \left\{ \frac{a_{0j}^k}{a_{sj}^k} | a_{sj}^k < 0 \right\}$;

primenimo sledeći korak:

Korak 3'. Odrediti $s \in \{1, \ldots, m\}$ za koje je $b_s^k < 0$. Odrediti $r \in \{1, \ldots, m\}$ takvo da je $\frac{K_r^k}{a_{sr}^k} = \text{lex-max} \left\{ \frac{K_j^k}{a_{sj}^k} | a_{sj}^k < 0 \right\}$, gde je K_j^k $j-$ta kolona $k-$te dualne simpleks tablice.

Možemo dokazati konačnost dualnog simpleks metoda.

Teorema 2.12.2 *Neka su u dualnoj simpleks tablici DT_0 kolone $K_1^0, \ldots, K_n^0$ leksikografski pozitivne i neka je u Algoritmu 4 Korak 3 zamenjen Korakom 3'. Tada niz nultih kolona (K_0^k) strogo leksikografski opada i dualni simpleks metod u konačnom broju iteracija dolazi ili do optimalnog rešenja ili do zaključka da je dopustivi skup prazan.*

Razmotrimo sada ***Blandova pravila***. Po Blandu [9], treba primenjivati sledeće dve modifikacije koraka 2 i 4:

- *Korak 2'. Izabrati $c_j > 0$ tako da je indeks odgovarajuće nebazične promenljive $x_{N,j}$ najmanji.*

- *Korak 4'. Izračunati*

$$\min_{1\le i\le m}\left\{\frac{b_i}{a_{ij}},\quad a_{ij}>0\right\}=\frac{b_p}{a_{pj}}$$

U slučaju jednakih vrednosti izabrati ono p takvo da je indeks odgovarajuće bazične promenljive $x_{B,p}$ najmanji. Zameniti nebazičnu promenljivu $x_{N,j}$ i bazičnu promenljivu $x_{B,p}$ i preći na Korak 1.

Dokažimo sada ispravnost ovih pravila. U dokazu će matrica A predstavljati matricu sistema, vektori b i c RHS vektor i vektor funkcije cilja u odgovarajućem standardnom obliku problema, a matrica T proširenu Tuckerovu tabelu. Ove oznake su iste kao u odeljku 2.11.

Teorema 2.12.3 *Blandova pravila eliminišu cikliranje kod simpleks metoda.*

Dokaz. Pretpostavimo suprotno. Neka je τ skup indeksa j takvih da promenljiva x_j tokom ciklusa **postaje** bazična. Neka je $q=\max\tau$. Neka je T' tabela u kojoj je promenljiva x_q **postaje** bazična a T'' tabela gde x_q **postaje** nebazična (u samim tabelama T' i T'' promenljiva x_q je redom nebazična i bazična). Označimo sa t pivot kolonu transformacije posle koje dobijamo tabelu T''.

Definišimo vektore $y=(y_1,\dots,y_n,y_{n+1})$ i $v=(v_1,\dots,v_n,v_{n+1})$ na sledeći način:

$$y_i=\begin{cases}1 & i=n+1\\ T'_{n+1,j} & i=v'_{N,j}\\ 0 & \text{u suprotnom}\end{cases}\qquad v_i=\begin{cases}-1 & i=v''_{N,t}\\ T''_{it} & i=v''_{B,i}\\ 0 & \text{u suprotnom}\end{cases}$$

Na osnovu **prvog Blandovog pravila** važi $y_1,\dots,y_{q-1}\ge 0$ i $y_q<0$. Pošto je $T=(A_c)_B^{-1}(A_c)_N$. Ako preuredimo kolone matrica A_c i vektora y i v možemo pisati $A_c=\begin{bmatrix}(A_c)_B & (A_c)_N\end{bmatrix}$ kao i $v=\begin{bmatrix}T''_{\bullet t} & -(I_m)_{\bullet t}\end{bmatrix}^T$ i $y=\begin{bmatrix}T'_{n+1\bullet}0\end{bmatrix}^T$. Sada imamo da je:

$$A_cv=[(A_c)_B\ (A_c)_N]\begin{bmatrix}T''_{\bullet t}\\ -(I_m)_{\bullet t}\end{bmatrix}=((A_c)_N)_{\bullet t}-((A_c)_N)_{\bullet t}=0$$

Znači v pripada jezgru $\mathcal{N}(A_c)$ matrice A_c. Primetimo takođe da je vektor y baš $n+1$-va vrsta matrice $(A_c)_B^{-1}A_c$. Odavde sledi da važi $y^Tv=0$. Primetimo da je $v_{n+1}=T''_{m+1,t}<0$, pa je $y_{n+1}v_{n+1}<0$. Znači, postoji j, tako da je $y_jv_j>0$.

Pošto je $y_j\neq 0$, promenljiva x_j je bazična u T' a pošto je $v_j\neq 0$, ili je x_j nebazična u T'' ili je $j=t$. U svakom slučaju je $j\in\tau$, pa je $j\le q$. Takođe, pošto je $v_q=T''_{p''t}>0$ ($q=v''_{B,p''}$, prema koraku 4 algoritma **BasicMax**) a $y_q=T'_{m+1,p'}<0$ ($q=v'_{N,p'}$, prema koraku 2 algoritma **BasicMax**), imamo da je $y_qv_q<0$ pa je $1\ge j<q$. Pošto je $y_j\ge 0$ sledi da je i $v_j\ge 0$.

Zaključujemo da je promenljiva x_j bazična u tabeli T''. Zato neka je $j=v_{B,k}$. Sada je $T''_{kt}=v_j>0$. Primetimo takođe da se tokom cikliranja poslednja kolona

Tuckerove tabele T (vektor b^*) ne menja. Naime, pošto se vrednost funkcija cilja $T_{m+1,n+1}$ ne menja, na osnovu formula (2.3.5) zaključujemo da je $b^*_p = 0$ (p je indeks pivot vrste), pa se na osnovu istih formula ne menja ni b^*.

Vrednosti svih promenljivih iz τ u odgovarajućim bazičnim rešenjima su 0. Zaista, ako je $z \in \tau$ i x_z bazična promenljiva, uočimo trenutak kada je ona postala bazična. Tada, ako je pivot element T''_{ql}, pri čemu je $v_{N,l} = z$ i $b^*_q = 0$, posle transformacije imamo da je $v_{B,q} = z$ a b^*_q ostaje 0.

Prema tome $(b^*)''_k = T''_{k,n+1} = 0$. Znači, sada imamo da je $(b^*)''_k = 0$, $T''_{kt} > 0$, $v''_{B,k} = j$ a $v''_{B,p''} = q$ gde je p'' odgovarajuća pivot vrsta (u sledećoj iteraciji x_q postaje nebazična). Na osnovu **drugog Blandovog pravila** zaključujemo da je $q = v''_{B,p''} \leq v''_{B,k} = j$ što je kontradikcija. Ovim je teorema dokazana. □

Primer kako se pomoću ovih pravila izbegava cikliranje kod prethodnog primera može se naći u [84].

Pomenimo na kraju dva nedostatka Blandovih pravila. Primena ovih pravila može da rezultuje takvim izborom pivot elemenata da su promene vrednosti funkcije cilja male, što često uzrokuje veći broj iteracija simpleks metoda. Takođe postoji opasnost od izbora pivot elemenata koji su bliski nuli i koji prouzrokuju velike numeričke greške.

2.13 Složenost simpleks metoda i Minty-Klee poliedri

U uvodu smo napomenuli da i pored dobrih osobina koje je pokazao u praksi, simpleks algoritam nije polinomijalan. To tvrđenje su prvi dokazali Minty i Klee u radu [38], još 1970. godine uz pretpostavku da se za pivot kolonu uzima prva kolona kod koje je $c_j < 0$. Kasnije je dokazano [39] da za skoro svako determinističko pravilo izbora pivot kolone postoji klasa primera problema linearnog programiranja tako da broj iteracija simpleks metoda zavisi eksponencijalno od dimenzije problema.

Definicija 2.13.1 *Posmatrajmo sledeći problem linearnog programiranja zadat u kanonskom obliku i preko Tuckerove tabele:*

$$\begin{aligned}
\min \epsilon^{n-1}x_1 + \epsilon^{n-2}x_2 + \ldots + \epsilon x_{n-1} + x_n & \\
x_1 & \leq t \\
2\epsilon x_1 + x_2 & \leq t^2 \\
2\epsilon^2 x_1 + 2\epsilon x_2 & \leq t^3 \\
\vdots & \vdots \\
2\epsilon^{n-1}x_1 + 2\epsilon^{n-2}x_2 + \ldots + 2\epsilon x_{n-1} + x_n & \leq t^n \\
x \geq 0 &
\end{aligned}$$

$$\begin{array}{ccccccc} x_1 & x_2 & \cdots & x_n & -1 & \\ 1 & 0 & \cdots & 0 & t & = -x_{n+1} \\ 2\epsilon & 1 & \cdots & 0 & t^2 & = -x_{n+2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 2\epsilon^{n-1} & 2\epsilon^{n-2} & \cdots & 1 & t^m & = -x_{n+m} \\ \epsilon^n & \epsilon^{n-1} & \cdots & 1 & 0 & = f \end{array}$$

Označimo ovaj problem sa $\mathcal{P}_n(\epsilon, t)$ i nazovimo ga uopštenim problemom Minty-Kleea dimenzije n.

Minty i Klee su u svom radu [38] posmatrali problem $\mathcal{P}_n(2, 5)$. Očigledno, za $t > 0$, ovaj problem je bazično dopustiv pa možemo odmah primeniti algoritam **BasicMax**. Dokažimo sada da algoritam **BasicMax** posle $2^n - 1$ iteracija dolazi do optimalnog rešenja $x^* = (0, \ldots, 0, t^n)$ $(x^* \in \mathbb{R}^n)$. Upravo to tvrdi glavna teorema ovog odeljka:

Teorema 2.13.1 *Neka je $\epsilon, t > 0$ i $\frac{\epsilon}{t} > \frac{1}{2}$. Algoritam* **BasicMax**, *primenjen na problem $\mathcal{P}(\epsilon, t)$, prolazi $2^n - 1$ iteracija do optimalnog rešenja $x^* = (0, \ldots, 0, t^n)$.*

Pre nego što damo dokaz, razmotrimo neke osobine uopštenog Minty-Klee problema.

Lema 2.13.1 *Neka važe uslovi teoreme 2.13.1. Ako primenimo algoritam za zamenu promenljivih k puta, pri čemu su pivot elementi $a^i_{p_i p_i} = 1$ gde su brojevi $p_i \in \{1, \ldots, n\}$, $i = 0, \ldots, k-1$ a sa a^l_{ij} je označen element (i, j) Tuckerove tabele posle l transformacija, dobijamo Tuckerovu tabelu T_k čiji su elementi jednaki $t^k_{ij} = (-1)^{c^k_{ij}} t^0_{ij}$, za $j < n + 1$. Sa c^l_{ij} smo označili broj pivot elemenata p_s za koje važi $j \leq p_s < i$ i $1 \leq s \leq l$.*

Dokaz. Dokaz ćemo izvesti matematičkom indukcijom. Za $l = 0$ tvrđenje trivijalno važi, sobzirom da je $c^0_{ij} = 0$ za svako $(i, j) \in \{1, \ldots, m + 1\} \times \{1, \ldots, n\}$. Pretpostavimo da tvrđenje važi za sve brojeve manje ili jednake k i dokažimo ga za $k+1$. Neka je $p = p_{k+1}$. Prema indukcijskoj hipotezi, u pivot vrsti i pivot koloni p, redom su različiti od nule elementi t^k_{ip} i t^k_{pj} tako da je $i \geq p$ a $p \geq j$. Prema tome, ako bar jedan od ova dva uslova ne važi, imamo da je $t^{k+1}_{ij} = t^k_{ij}$ i $c^{k+1}_{ij} = c^k_{ij}$, jer $p \notin [j, i)$ pa tvrđenje leme važi.

Za $i = p$ važi

$$t^{k+1}_{pj} = \frac{t^k_{pj}}{t^k_{pp}} = t^k_{pj} = (-1)^{c^k_{pj}} t^0_{pj} = (-1)^{c^{k+1}_{pj}} t^0_{pj}$$

jer je $c^{k+1}_{pj} = c^k_{pj}$, za $j \geq p$. Za $j = p$ imamo da je

$$t^{k+1}_{ip} = -\frac{t^k_{ip}}{t^k_{pp}} = -t^k_{ip} = -(-1)^{c^k_{ip}} t^0_{ip} = (-1)^{c^{k+1}_{ip}} t^0_{ip}$$

jer je $c_{ip}^{k+1} = c_{ip}^k + 1$ za $i \leq p$.

Za $j < p$ i $p < i$ dobija se

$$t_{ij}^{k+1} = t_{ij}^k - t_{pj}^k t_{ip}^k = (-1)^{c_{ij}^k} t_{ij}^0 - (-1)^{c_{ip}^k + c_{pj}^k} t_{ip}^0 t_{pj}^0$$

Kako je $t_{ij}^0 = 2\epsilon^{i-j}$ i $c_{ip}^k + c_{pj}^k = c_{ij}^k$ imamo

$$\begin{aligned} t_{ij}^{k+1} &= (-1)^{c_{ij}^k} \left(t_{ij}^0 - t_{ip}^0 t_{pj}^0\right) = (-1)^{c_{ij}^k} \left(2\epsilon^{i-j} - 4\epsilon^{i-p+p-j}\right) = -(-1)^{c_{ij}^k} 2\epsilon^{i-j} \\ &= (-1)^{c_{ij}^{k+1}} t_{ij}^0. \end{aligned}$$

U poslednjoj jednakosti smo iskoristili da je $c_{ij}^{k+1} = c_{ij}^k + 1$ jer je $i > p$ i $p > j$. Ovim je lema dokazana. □

Iz dokaza prethodne leme sledi da ako za pivot vrstu uvek biramo $p = j$ (pivot kolonu biramo kao u algoritmu **BasicMax**) posle $2^n - 1$ koraka dolazimo do optimalnog rešenja. Da bi to dokazali, pretpostavimo da smo u k-tom koraku odabrali j-tu kolonu za ključnu. Tada je $T_{m+1,1}^k, \ldots, T_{m+1,j-1}^k < 0$ a $T_{m+1,j}^k > 0$. Sada izborom elementa T_{jj}^k za ključni, na osnovu dokaza prethodne leme zaključujemo da će posle transformacije važiti $T_{m+1,1}^{k+1}, \ldots, T_{m+1,j-1}^{k+1} > 0$ a $T_{m+1,j}^{k+1} < 0$. Pridružimo sada svakoj Tuckerovoj tabeli T^k jedan prirodan broj $\tau(T^k)$ na sledeći način. Cifra na l-toj poziciji u binarnom zapisu broja $\tau(T^k)$ jednaka je 0 ako je $T_{m+1,l}^k$ pozitivan, u suprotnom je jednaka 1. Upravo smo dokazali da važi $\tau(T^{k+1}) = \tau(T^k) + 1$, odnosno $\tau(T^k) = k$. Algoritam staje kada su sve cifre broja $\tau(T^k)$ jednake jedinici, odnosno kada je $k = 2^n - 1$.

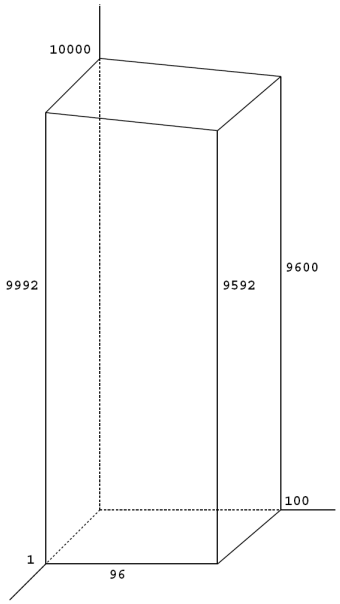
Sledeća lema, koju nećemo dokazivati završava dokaz teoreme 2.13.1:

Lema 2.13.2 *U svakoj iteraciji algoritma* **BasicMax**, *primenjenog na problem* $\mathcal{P}_n(\epsilon, t)$ *važiće* $p = j$, *tj. pivot element biće na glavnoj dijagonali Tuckerove tabele.*

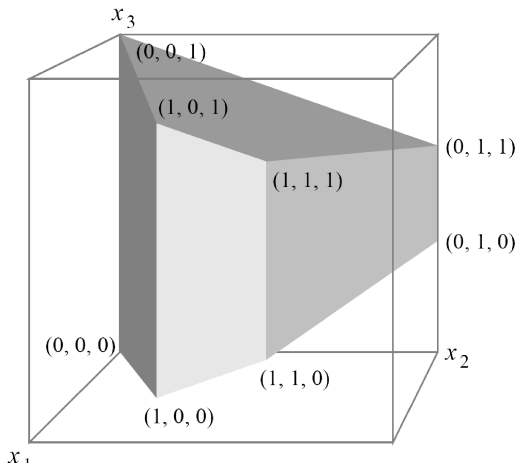
Poliedar dopustivih rešenja Ω_P za problem $\mathcal{P}(\epsilon, t)$ nazivamo *Minty-Klee poliedar*. Napomenimo, da ako izvršimo odgovarajuće smene promenljivih, Minty-Klee poliedar možemo opisati i sledećim sistemom nejednačina

$$\begin{array}{lrl} \min & x_n & \\ \text{p.o.} & & x_1 \leq 1 \\ & \epsilon x_1 \leq & x_2 \leq 1 - \epsilon x_1 \\ & \epsilon x_2 \leq & x_3 \leq 1 - \epsilon x_2 \\ & \vdots & \\ & \epsilon x_{n-1} \leq & x_n \leq 1 - \epsilon x_{n-1} \\ & x \geq 0 & \end{array}$$

Geometrijski, poslednji sistem predstavlja deformisanu n-dimenzionalnu jediničnu hiperkocku. Na slikama 2.13.1 i 2.13.2 su prikazani Minty-Klee poliedar za problem $\mathcal{P}_3(10, 100)$ kao i poliedar koji odgovara sistemu (2.13.1). Primetimo da putanja koju prolazi simpleks metod odgovara Hamiltonovom putu u odgovarajućem grafu poliedra.



Slika 2.13.1. Skup dopustivih rešenja za primer $\mathcal{P}_3(10, 100)$.



Slika 2.13.2. Minty-Klee poliedar za $\epsilon = \frac{1}{3}$.

Minty-Klee poliedri i njihove osobine proučavani od strane velikog broja autora. Uz višestruko ponavljanje odgovarajućih nejednakosti, u radu [23] je pokazano da određena klasa interior-point metoda takođe ima eksponencijalnu složenost u najgorem slučaju. U radu [28] razmatrana je varijanta simpleks metoda u kojoj se i pivot kolona i pivot vrsta biraju slučajno. U tom slučaju, izračunat je očekivani broj iteracija za primer $\mathcal{P}_n(\epsilon, t)$ koji iznosi:

$$F_n(\bar{x}) = n + 2\sum_{k=1}^{n} \frac{(-1)^{k+1}}{k+2} \binom{n-k}{2} \approx \left(\frac{\pi}{4} - \frac{1}{2}\right) n^2.$$

3 Modifikacije simpleks metoda i implementacija

U ovoj glavi pokazaćemo nekoliko originalnih modifikacija i poboljšanja pojedinih faza simpleks metoda [2] (algoritama **Replace**, **NoBasicMax**, **ElJed**, **ElSl**). Ukazaćemo na nekoliko nedostataka odgovarajućih algoritama i predložićemo modifikacije kod kojih nema tih nedostataka.

3.1 Dva poboljšanja simpleks metoda

U Algoritmu **Replace** opisan je postupak kojim se Tuckerova tabela transformiše u ekvivalentnu tabelu zamenom promenljivih. U praksi, Tuckerove tabele mnogih problema linearnog programiranja imaju dosta nula (oko 80%) u sebi. Broj operacija koje izvrši algoritam je $(m+1)(n+1)$ i ne zavisi od strukture same tabele. Primetimo sada da, pri transformaciji, mali broj elemenata promeni vrednost, pošto su mnogi elementi u Tuckerovoj tabeli jednaki nuli. Pošto se za ključni element a_{pj} uvek bira broj različit od nule, to ovaj elemenat uvek menja vrednost (osim kad je jednak 1). Elementi koji se nalaze u istoj vrsti ili istoj koloni sa ključnim elementom posle transformacije redom postaju jednaki:

$$\begin{aligned} a^1_{pl} &= \frac{a_{pl}}{a_{pj}}, \quad l \neq j; \\ a^1_{qj} &= -\frac{a_{qj}}{a_{pj}}, \quad q \neq p; \end{aligned} \tag{3.1.1}$$

Odavde se vidi da će oni menjati vrednosti ako i samo ako su različiti od nule. Takođe, svaki elemenat koji nije ni u istoj vrsti ni u istoj koloni sa ključnim postaje jednak:

$$a^1_{ql} = a_{ql} - \frac{a_{pl}a_{qj}}{a_{pj}}, \quad q \neq p,\ l \neq j$$

On će menjati vrednost ako i samo ako su projekcije na ključnu vrstu a_{pl} i a_{qj} različite od 0.

Znači, bilo koji element iz Tuckerove tabele se menja akko su obe njegove projekcije različite od 0. Konstruišimo sada skupove V i K na sledeći način:

$$\begin{aligned} V &= \{l \mid a_{pl} \neq 0,\ l = 1, \ldots, n+1\}, \\ K &= \{q \mid a_{qj} \neq 0,\ q = 1, \ldots, m+1\}. \end{aligned} \tag{3.1.2}$$

[2] Rezultati izloženi u ovoj glavi su preuzeti iz naših radova [78, 58, 70]

Znači, bilo koji element iz Tuckerove tabele se menja akko su mu koordinate redom u skupovima V i K.

Algoritam ModReplace (Poboljšanje algoritma Replace)

Korak 1. Formirati skupove V i K.

Korak 2. Sve elemente a_{ql} takve da $q \in K$ i $l \in V$ transformisati prema algoritmu **Replace**.

Ovim smo postigli da se broj operacija smanji od $(m+1)(n+1)$ na $|V||K|$ operacija. Ako u matrici ima mnogo nula, tada i skupovi V i K imaju mali broj elemenata, pa se u tom slučaju znatno redukovao broj operacija.

Razmotrimo sada jedno poboljšanje algoritma **ElJed** za eliminaciju jednačina.

U Algoritmu **ElJed** u svakoj iteraciji mi izbacujemo po jednu kolonu matrice sistema. Za izbacivanje elementa iz matrice potrebno je izvršiti približno mn operacija. Pošto se izbacivanje vrši u svakoj iteraciji to sledi da je broj operacija koji se izvrši za izbacivanje kolona iz matrice približno jednak mnJ gde je J broj jednačina. Taj broj se drastično redukuje predloženom modifikacijom. Naime, umesto izbacivanja, kolonu možemo samo markirati. Za to koristimo logički niz *outc*. Analogno, formiramo niz *outr* kojim markiramo izbačene vrste matrice sistema. Na kraju, izbacujemo sve markirane vrste i kolone i za to nam je ukupno potrebno mn operacija što predstavlja poboljšanje za ceo red veličine. Ovaj metod se može upotrebiti i kod algoritma za eliminaciju slobodnih promenljivih **ElSl**.

Pošto su modifikovani algoritmi veoma slični originalnim, nećemo ih eksplicitno formulisati, već cemo samo napomenuti da je u koracima 2 i 3 algoritma **ElJed**, odnosno 2 i 3' algoritma **ElSl** potrebno zameniti izbacivanje vrste, odnosno kolone sa $outc(j) = \top$, odnosno $outr(j) = \top$. Na kraju algoritma treba dodati još jedan korak kojim se izbacuju sve označene vrste.

3.2 Modifikacija Simpleks metoda za probleme koji nisu bazično dopustivi

Posmatramo problem linearnog programiranja (the linear programming (LP) problem) u standardnoj formi:

$$\begin{aligned} \max \quad & \mathbf{c}^T\mathbf{x} - d, \\ \text{p.o.} \quad & A\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \geq 0, \end{aligned} \tag{3.2.1}$$

gde je $A \in \mathbb{R}^{m\times(m+n)}$ matrica potpunog ranga vrsta ($\text{rank}(A) = m$), $\mathbf{c} \in \mathbb{R}^{n+m}$ i sistem $A\mathbf{x} = \mathbf{b}$ je definisan sa $\mathbf{x} \in \mathbb{R}^{m+n}$, $\mathbf{b} \in \mathbb{R}^m$. Pretpostavlja se da je (i,j)-ti element u A označen sa a_{ij}, $\mathbf{b} = (b_1, \ldots, b_m)^T$, $d \in \mathbb{R}$, $\mathbf{x}^T = (x_1, \ldots, x_{n+m})$ i $\mathbf{c}^T = (c_1, \ldots, c_{n+m})$.

Dvofazni simpleks metod se odvija u dve faze, koje se nazivaju faza I i faza II (phase I and phase II). Faza I pokušava da pronađe jedno inicijalno bazično dopustivo rešenje (initial basic feasible solution). Kada se jedno inicijalno bazično

dopustivo rešenje pronađe, primenjuje sa faza II da bi se pronašlo optimalno rešenje. Simpleks metod čini iteracije kroz skup bazičnih rešenja (dopustiva u fazi II) problema linearnog programiranja. Svako bazično rešenje se karakteriše skupom od m basičnih promenljivih (basic variables) $x_{B,1}, \ldots, x_{B,m}$. Preostale n promenljive se nazivaju nebazične promenljive (nonbasic variables), i označavaju se $x_{N,1}, \ldots, x_{N,n}$.

Ako je $\mathbf{b} \geq 0$ i ako su sve nebazične promenljive $x_{N,1}, \ldots, x_{N,n}$ jednake nuli, tada $x_{B,1} = b_1, \ldots, x_{B,m} = b_m$ jeste bazično dopustivo rešenje. Ako uslov $\mathbf{b} \geq 0$ nije ispunjen, neophodno je da se pronađe jedno bazično dopustivo rešenje ili da se odredi da ono ne postoji. Postoji veći broj strategija za fazu I. Klasični pristup se sastoji u tome da se sa standardnim linearnim problemom asocira sledeći prošireni problem:

$$\begin{array}{ll} \min & \mathbf{ew}, \\ \text{p.o.} & A\mathbf{x} + \mathbf{w} = \mathbf{b}, \\ & \mathbf{x} \geq 0,\ \mathbf{w} \geq 0, \end{array} \tag{3.2.2}$$

gde je $\mathbf{e} = (1, \ldots, 1) \in \mathbb{R}^m$ i $\mathbf{w} \in \mathbb{R}^m$ je vektor veštačkih promenljivih (artificial variables). Poznato je da ako je $(\mathbf{x}^*, \mathbf{w}^*)$ optimalno rešenje problema (3.2.2), tada neophodan i dovoljan uslov da (3.2.1) ima dopustivo rešenje jeste $w_i^* = 0, \quad i = 1, \ldots, m$ [5, 33]. Tada nema veštačkih promenlivih u finalnoj bazi, veštačke promenljive i odgovarajuće kolone se eliminišu, i dopustivo rešenje početnog LP problema je generisano. Nedostatak ovog pristupa jeste upotreba veštačkih promenljivih, jer se u fazi I dodaje m novih veštačkih promenljivih $w = (w_1, \ldots, w_m)$, po jedna za svako ograničenje.

Druga varijanta dvofaznog simpleks metoda (two-phase simplex method) je definisana u [51] i [79], i opisana u odeljcima 2.3, 2.4 i 2.5. U radovima [70], [78] korišćen je taj algoritam, jer ne zahteva uvođenje veštačkih promenljivih. U tim radovima su uvedena dva algoritma za dobijanje početnog bazično dopustivog rešenja u fazi I dvofaznog simpleks algoritma opisanog u [51] i [79]. Opisano je novo pravilo za izbor bazičnih i nebazičnih promenljivih, tj. za izbor promenljive koja ulazi u bazu kao i promenljive koja napušta bazu. Naš konačni cilj je da se minimizira ukupan broj iteracija u simpleks algoritmu. Međutim, taj cilj je nepristupačan. Prema tome, naš cilj je da se optimizira tekući simpleks korak. Na taj način je poboljšana efikasnost simpleks algoritma, što je potvrđeno numeričkim primerima.

Posmatraćemo kanonički oblik problema linearnog programiranja (kao i u odeljku 2.3) zadatog pomoću Tuckerove tabele.

$x_{N,1}$	$x_{N,2}$	$\ldots$	$x_{N,n}$	-1	
a_{11}	a_{12}	$\ldots$	a_{1n}	b_1	$= -x_{B,1}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
a_{m1}	a_{m2}	$\ldots$	a_{mn}	b_m	$= -x_{B,m}$
c_1	c_2	$\ldots$	c_n	d	$= z$

(3.2.3)

U ovom odeljku razmotrićemo modifikaciju metoda za nalaženje početnog bazično

dopustivog rešenja (algoritma **NoBasicMax**). Ovi rezultati su preuzeti iz naših radova [70, 78]. Možemo uočiti dva nedostatka algoritma **NoBasicMax**:

1. Ako je $p = i$ i ako postoji indeks $t < i = p$ tako da je

$$\frac{b_t}{a_{tj}} < \frac{b_p}{a_{pj}}, \quad b_t > 0,\ -a_{tj} < 0$$

u sledećoj iteraciji promenljiva $x_{B,t}$ postaje negativna:

$$x_{B,t} = b_t^1 = b_t - \frac{b_i}{a_{ij}} a_{tj} < 0.$$

2. Ako je $p > i$, i u sledećoj iteraciji slobodni koeficijent b_i^1 je negativan, ali može da postoji $b_t < 0$, $t < i$ tako da je

$$\min_{k>t} \left(\left\{ \frac{b_t}{a_{tj}},\ -a_{tj} > 0 \right\} \cup \left\{ \frac{b_k}{a_{kj}} \mid -a_{kj} < 0,\ b_k > 0 \right\} \right) = \frac{b_t}{a_{tj}}.$$

U tom slučaju je moguće izabrati a_{tj} za pivot element i dobijamo

$$x_{B,t} = b_t^1 = \frac{b_t}{a_{tj}} \geq 0.$$

Takođe, kako je

$$\frac{b_t}{a_{tj}} \leq \frac{b_k}{a_{kj}},$$

svaki $b_k > 0$ ostaje pogodan za bazično dopustivo rešenje:

$$x_{B,k} = b_k^1 = b_k - \frac{b_t}{a_{tj}} a_{kj} \geq 0.$$

Iz tih razloga predlažemo modifikaciju koraka 4. Glavna ideja je sadržana u sledećoj lemi:

Lema 3.2.1 *Neka je problem 3.2.3 dopustiv, neka je* $b_{i_1}, \ldots, b_{i_q} < 0$ *i neka je* $I = \{i_1, \ldots, i_q\}$. *U sledeća dva slučaja:*

a) $q = m$,

b) $q < m$ *i postoji* $r \in I$ *i* $s \in \{1, \ldots, n\}$ *tako da važi:*

$$\min_{h \notin I} \left\{ \frac{b_h}{a_{hs}} \mid -a_{hs} < 0 \right\} \geq \frac{b_r}{a_{rs}},\ -a_{rs} > 0, \tag{3.2.4}$$

moguće je dobiti novo bazično rešenje $x^1 = \{x_{B,1}^1, \ldots, x_{B,m}^1\}$ *sa najviše* $q - 1$ *negativnih koordinata u samo jednom iterativnom koraku simpleks metoda, ako se* a_{rs} *izabere za pivot element, tj. ako zamenimo nebazičnu promenljivu* $x_{N,s}$ *bazičnom promenljivom* $x_{B,r}$.

Dokaz. a) Ako je $q = m$, izaberemo proizvoljan koeficijent $-a_{ms} > 0$ za pivot element. Primenom simpleks metoda dobijamo novo rešenje sa najmanje jednom koordinatom pozitivnom. Na primer, imamo

$$x^1_{B,m} = b^1_m = \frac{b_m}{a_{ms}} \geq 0.$$

b) Preptostavimo sada da su uslovi $q < m$ i (3.2.4) zadovoljeni. Izaberemo a_{rs} za pivot element. U tom slučaju koordinate novog bazičnog rešenja su

$$x^1_{B,i} = b^1_i = b_i - \frac{b_r}{a_{rs}} a_{is}, \quad i \neq r,$$
$$x^1_{B,r} = b^1_r = \frac{b_r}{a_{rs}}.$$

Za $k \neq r$, $k \notin I$ i $a_{ks} < 0$ očigledno je da je

$$x^1_{B,k} = b_k - \frac{b_r}{a_{rs}} a_{ks} \geq 0.$$

Za $a_{ks} > 0$ i $k \notin I$, iz

$$\frac{b_k}{a_{ks}} \geq \frac{b_r}{a_{rs}}$$

odmah sledi

$$x^1_{B,k} = b^1_k = b_k - \frac{b_r}{a_{rs}} a_{ks} \geq 0$$

što je i trebalo pokazati. □

U skladu sa rezultatom iz Leme 3.2.1 predlažemo sledeću modifikaciju koraka 4 koja smanjuje broj negativnih koordinata u novom bazičnom rešenju za najmanje jedan u svakoj iteraciji, ukoliko izaberemo pivot element a_{rs} tako da b_r i a_{rs} zadovoljavaju uslov (3.2.4).

- *Korak 4'. Neka je $b_{i_1}, \ldots, b_{i_q} < 0$. Ako je $q = m$, izabrati proizvoljan koeficijent $a_{ms} < 0$ za pivot element.*

  Ako je $q < m$, razmotriti $a_{rs} < 0$ za $r \in \{i_1, \ldots, i_q\} = I$. Ako postoje $r \in I$ i $s \in \{1, \ldots, n\}$ tako da je uslov (3.2.4) *zadovoljen, tada izabrati a_{rs} za pivot element.*

Napomena 3.2.1 Ako ne postoji r ako da važi uslov (3.2.4), neka su $r \notin I$ i $s \in \{1, \ldots, n\}$ takvi da je

$$\min_{h \notin I} \left\{ \frac{b_h}{a_{hs}} \mid -a_{hs} < 0 \right\} = \frac{b_r}{a_{rs}}.$$

Za takvo r i s izabrati a_{rs} za pivot element. Primenom simpleks transformacije dobija se novo rešenje x^1 sa nenegativnim koordinatama

$$x^1_{B,i} = b^1_i = b_i - \frac{b_r}{a_{rs}} a_{is}, \; i \notin I.$$

Prema tome, broj negativnih koordinata u novom bazičnom rešenju (broj negativnih vrednosti b_i) u opštem slučaju ostaje isti.

Na osnovu predhodne napomene možemo zaključiti da i ovde postoji opasnost od cikliranja. Sada ćemo pokazati jedno anticiklično pravilo koje u ovom slučaju možemo primeniti [70].

S obzirom da je i_s fiksirano u koracima 4, 5 i 6, algoritam **ModNoBasicMax** može da ciklira samo ako je $b_{i_s}^1 = b_{i_s}$. Ako sada primenimo Blandova pravila pri čemu i_s-ti red smatramo funkcijom cilja, dobićemo da će posle konačno mnogo iteracija ili biti ispunjen uslov Leme 3.2.1 ili da će svi $a_{i_s,j}$ biti pozitivni, odnosno problem će biti nedopustiv.

U skladu sa ovim razmatranjima, predlažemo sledeće poboljšanje algoritma **NoBasicMax**.

Algoritam ModNoBasicMax (Modifikacija algoritma **NoBasicMax**)

Korak 1. Ako je $b_1, \ldots, b_m \geq 0$ preći na korak 7.

Korak 2. Konstruisati skup

$$B = \{b_{i_1}, \ldots, b_{i_e}\} = \{b_{i_k} \mid b_{i_k} < 0,\ k = 1, \ldots, e\}.$$

Korak 3. Izabrati proizvoljno $b_{i_s} < 0$.

Korak 4. Ako je $a_{i_s,1}, \ldots, a_{i_s,n} \geq 0$ tada STOP. Problem linearnog programiranja nema rešenja. U suprotnom, konstruisati skup

$$Q = \{a_{i_s,j_p} < 0,\ k = 1, \ldots, t\},$$

i postaviti $p = 1$.

Korak 5. Naći minimum

$$\min_{1 \leq k \leq m} \left\{ \frac{b_k}{a_{k,j_p}} \mid a_{k,j_p} > 0,\ b_k > 0 \right\} = \frac{b_u}{a_{u,j_p}}.$$

Ako je

$$\frac{b_{i_s}}{a_{i_s,j_p}} \leq \frac{b_u}{a_{u,j_p}}$$

zameniti nebazičnu i bazičnu promenljivu x_{N,j_p} i x_{B,i_s} i preći na korak 2. Vrednost b_{i_s} postaje pozitivna.

Korak 6. Ako je $p \leq t$ staviti $p = p+1$ i preći na korak 5. U suprotnom zameniti promenljive x_{N,j_p} i $x_{B,u}$ i preći na korak 4. Vrednost b_{i_s} je i dalje negativna.

Korak 7. Primeniti simpleks algoritam za bazično dopustive probleme, algoritam **BasicMax**.

Prednost algoritma **ModNoBasicMax** u odnosu na algoritam **NoBasicMax** je u tome što se kod algoritma **ModNoBasicMax** broj negativnih vrednosti b_i ne povećava prilikom iteracija, dok kod **NoBasicMax** to ne mora da bude slučaj. Mada, kako se pokazalo u praksi (videti odeljak 3.5) ova modifikacija ponekad može da ima kontraefekat: da veoma brzo dođe do bazično dopustivog problema, ali da posle algoritam **BasicMax** napravi mnogo više iteracija nego kada se primeni algoritam **NoBasicMax**.

3.3 Poboljšana verzija modifikacije za probleme koji nisu bazično dopustivi

U našem radu [70], uočili smo da algoritam **ModNoBasicMax** možemo još malo poboljšati. Naime, u algoritmu **ModNoBasicMax** vrednost i_s je fiksirana. Zato može da se dogodi da uslov Leme 3.2.1 nije zadovoljen za $i = i_s$ ali da postoji neko drugo $b_i < 0$ tako da je $a_{ij_p} < 0$ i da je uslov leme zadovoljen. Tada je bolje izabrati a_{ij_p} za pivot jer onda b_i postaje pozitivno.

Posmatrajmo šta se dešava sa vrednostima b_l posle zamene promenljivih $x_{B,i}$ i $x_{N,j}$:

$$b_l^1 = b_l - \frac{a_{lj}b_i}{a_{ij}} = a_{lj}\left(\frac{b_l}{a_{lj}} - \frac{b_i}{a_{ij}}\right).$$

Neka je sad $\frac{b_i}{a_{ij}} \geq 0$ (b_i i a_{ij} su istog znaka). Ako je $b_l > 0$, da bi ono ostalo pozitivno posle iteracije mora biti ili $a_{lj} < 0$ ili $a_{lj} > 0$ i $\frac{b_l}{a_{lj}} \geq \frac{b_i}{a_{ij}}$. Međutim, ako je $b_l < 0$, ono postaje pozitivno ako su a_{lj} i $\frac{b_l}{a_{lj}} - \frac{b_i}{a_{ij}}$ istog znaka. Ovo razmatranje nas navodi na sledeću strategiju izbora pivot elementa:

- Najpre moramo obezbediti da elementi b_l koji su pozitivni takvi i ostanu. Zato mora da bude

$$\frac{b_i}{a_{ij}} \leq \min\left\{\frac{b_k}{a_{kj}} \mid b_k > 0, a_{kj} > 0\right\}.$$

- Da bi negativan b_l postao pozitivan mora da je ili $a_{lj} > 0$ ili

$$\frac{b_l}{a_{lj}} \leq \frac{b_i}{a_{ij}}.$$

Na osnovu ovoga konstruišemo sledeći algoritam [70]:

Algoritam AdvModNoBasicMax

(Poboljšana verzija algoritma **ModNoBasicMax**)

Korak 1. Ako je $b_1, \ldots, b_m \geq 0$ preći na korak 5.

Korak 2. Konstruisati skup

$$B = \{b_{i_1}, \ldots, b_{i_e}\} = \{b_{i_k} \mid b_{i_k} < 0,\ k = 1, \ldots, e\}.$$

Korak 3. Neka je $s = 1$.

Korak 3.1 Ako je $a_{i_s,1}, \ldots, a_{i_s,n} \geq 0$ onda STOP. Problem je nedopustiv. U suprotnom konstruisati skup

$$Q = \{a_{i_s,j_p} < 0,\ p = 1, \ldots, t\},$$

i postaviti $p = 1$.

Korak 3.2 Naći minimume

$$M(j_p) = \min\left\{\frac{b_k}{a_{k,j_p}} \,|\, b_k > 0, a_{k,j_p} > 0\right\}.$$
$$p' = \operatorname{argmin}\left\{\frac{b_k}{a_{k,j_p}} \,|\, b_k < 0, a_{k,j_p} < 0\right\}.$$

Ako je $\frac{b_k}{a_{k,j_p}} \leq M(j_p)$ odabrati a_{p',j_p} za pivot element, izvršiti zamenu promenljivih x_{B,j_p} i $x_{N,p'}$ i preći na korak 1. (U sledećoj iteraciji b_k postaje pozitivno).

Korak 3.3 Ako je $p < t$ onda staviti $p = p + 1$ i preći na korak 3.2.

Korak 3.4 Ako je $s < e$ onda staviti $s = s + 1$ i preći na korak 3.1.

Korak 4. (Uslov leme 3.2.1 nije zadovoljen ni za jedno i_s i j_p) Označimo sa $v_{N,j}$ indeks bazične promenljive $x_{N,j}$ (odnosno takvu vrednost da važi $x_{v_{N,j}} = x_{N,j}$). Određujemo pivot prema Blandovim pravilima.

Korak 4.1 Naći $j_0 = \operatorname{argmin}\left\{v_{N,l} \,|\, a_{i_q,l} < 0\right\}$.

Korak 4.2 Naći

$$p'' = \operatorname{argmin}\left\{v_{B,p} \,|\, \frac{b_p}{a_{p,j_0}} = M(j_0)\right\}.$$

Korak 4.3 Izvršiti zamenu promenljivih $x_{B,j}$ i $x_{N,p''}$ i preći na korak 3.

Korak 5. Primeniti simpleks algoritam za bazično dopustive probleme, algoritam **BasicMax**.

Primetimo da u koraku 3.2 minimumi koje računamo zavise samo od j_p (ne zavise od i_s). Takođe, može se desiti da za nekoliko različitih vrednosti i_s nepotrebno računamo iste vrednosti za isto j_p. U tom slučaju, treba jednostavno preskočiti ponovljenu vrednost j_p i preći na sledeću (odnosno na korak 3.3).

3.4 Modifikacija revidiranog simpleks metoda

Kao i u odeljku 2.11 posmatramo problem linearnog programiranja u standardnom obliku:

$$\begin{aligned} &\min c^T x + d, \\ &p.o. Ax = b, \\ &x \geq 0, \end{aligned} \tag{3.4.1}$$

U odeljcima 3.2 i 3.3 razmotrili smo modifikaciju algoritma za nalaženje bazično dopustivog rešenja. Nedostaci koje smo uočili kod algoritma **NoBasicMax** važe i ovde i odnose se na algoritam **RevNoBasicMax**. Pri čemu, ovde moramo uzeti u obzir da nemamo celu Tuckerovu tabelu na raspolaganju već samo jedan njen deo. U ovom odeljku ćemo opisati modifikaciju algoritma **RevNoBasicMax** baziranu na već obrađenim modifikacijama kod klasičnog simpleks metoda.

Ovde je glavna ideja da nađemo minimum:

$$\frac{b_p^*}{T_{pj}} = \min\left\{\frac{b_k^*}{T_{kj}} \mid b_k^* < 0, T_{kj} < 0, k = 1, \ldots, m\right\}$$

i ako je relacija (3.4.2) zadovoljena, onda zamenom promenljivih $x_{B,p}$ i $x_{N,j}$ dobijamo novo bazično rešenje sa manjim brojem negativnih koordinata $(b^*)_i^1$.

Zato predlažemo modifikaciju koraka 6. algoritma **RevNoBasicMax**.

Lema 3.4.1 *Neka je problem* (3.4.1) *dopustiv i neka je x bazično nedopustivo rešenje sa q negativnih koordinata. Tada postoji $T_{ij} < 0$. Takođe, u sledeća dva slučaja:*

a) $q = m$,

b) $q < m$ *and*

$$\begin{aligned} Neg = &\left\{\frac{b_k^*}{T_{kj}} \mid b_k^* < 0, T_{kj} < 0, k = 1, \ldots, m\right\}, \\ Pos = &\left\{\frac{b_k^*}{T_{kj}} \mid b_k^* > 0, T_{kj} > 0, k = 1, \ldots, m\right\}, \\ &\min\left(Neg \cup Pos\right) = \frac{b_r^*}{T_{rj}} \in Neg \end{aligned} \tag{3.4.2}$$

moguće je naći novo bazično rešenje sa najviše $q-1$ negativnih koordinata u samo jednoj iteraciji revidiranog simpleks metoda, ako odaberemo T_{rj} za pivot element, tj. zamenimo promenljive $x_{N,j}$ i $x_{B,r}$.

Dokaz. Prvi deo teoreme koji je vezan za postojanje elementa $T_{ij} < 0$ je ekvivalentan lemi 3.2.1.

Dokažimo drugi deo teoreme.

a) Ako je $q = m$, za proizvoljni pivot elemenat $T_{ij} < 0$ dobijamo novo bazično rešenje sa bar jednom pozitivnom koordinatom:

$$(b^*)_i^1 = \frac{b_i^*}{T_{ij}} > 0.$$

b) Neka sada važi $q < m$ i (3.4.2). Odaberimo T_{rj} za pivot elemenat.

Za $b_k^* > 0$ i $T_{kj} < 0$ je trivijalno

$$(b^*)_k^1 = b_k^* - \frac{b_r^*}{T_{rj}} T_{kj} > b_k^* \geq 0.$$

Za $b_k^* > 0$ i $T_{kj} > 0$, korišćenjem $\frac{b_k^*}{T_{kj}} \geq \frac{b_r^*}{T_{rj}}$, odmah dobijamo

$$(b^*)_k^1 = b_k^* - \frac{b_r^*}{T_{rj}} T_{kj} \geq 0.$$

Znači svi pozitivni b_k^* ostaju pozitivni. Za $b_r^* < 0$ dobijamo

$$(b^*)_r^1 = \frac{b_r^*}{T_{rj}} \geq 0.$$

Ovim je dokaz završen. □

Napomena 3.4.1 Neka uslov (3.4.2) ne važi, tj. važi $\min(Neg \cup Pos) = \frac{b_r^*}{T_{rj}} \in Pos$. Ako odaberemo T_{rj} za pivot elemenat imamo

$$(b^*)_k^1 = b_k^* - \frac{b_r^*}{T_{kj}} T_{rj} \geq 0,$$

za $(b^*)_k > 0$ i T_{rj} bilo pozitivno ili negativno. Ali za negativno $b_k^* > 0$ možemo na sličan način dokazati

$$(b^*)_k^1 = b_k^* - \frac{b_r^*}{T_{kj}} T_{rj} < 0.$$

za T_{rj} bilo pozitivno ili negativno. Znači, naše novo rešenje ima isti broj negativnih koordinata q kao prethodno. I u ovom slučaju primenom odgovarajućih anticikličnih pravila kao u odeljku 3.3 omogućavamo da algoritam završi rad u konačnom vremenu.

Na osnovu razmatrane teorije konstruišemo sledeći algoritam.

Algoritam ModRevNoBasicMax (Modifikacija algoritma **RevNoBasicMax**)

Korak 1. Neka su A_B i A_N bazična i nebazična matrica. Rekonstruisati vektor $b^* = A_B^{-1} b$.

Korak 2. Konstruisati skup

$$B = \{b_{i_1}^*, \ldots, b_{i_q}^*\} = \{b_{i_k}^* \mid b_{i_k}^* < 0,\ k = 1, \ldots, q\}.$$

Korak 3. Izabrati proizvoljno $b_{i_s}^* < 0$.

Korak 4. Rekonstruisati i_s-tu vrstu $T_{i_s \bullet}$. Ako je $T_{i_s \bullet} \geq 0$ onda STOP. Problem je nedopustiv. U suprotnom, odabrati $T_{i_s, j} \leq 0$.

Korak 5. Rekonstruisati j-tu kolonu $T_{\bullet j}$. Izračunati

$$\min_{1 \leq i \leq n} \left\{ \frac{b_i^*}{T_{ij}} \mid b_i^* T_{ij} > 0, i = 1, \ldots, m \right\} = \frac{b_p^*}{T_{pj}},$$

zameniti promenljive $x_{N,j}$ i $x_{B,p}$ i preći na korak 2.

3.5 Implementacija Simpleks metoda i rezultati testiranja programa

Simpleks algoritam (algoritmi **Replace**, **BasicMax**, **NoBasicMax**, **ElJed**, **ElSl**) kao i modifikacije (algoritmi **ModNoBasicMax**, **AdvModNoBasicMax**) implementirani su u programskom jeziku Visual Basic 6.0 [47]. Tako je nastao program

MarPlex koji, kako ćemo videti, vrlo uspešno rešava široku klasu problema linearnog programiranja.

Ulazni podaci u MarPlex-u se zadataju u MPS fajlu. Ovaj tip fajla predstavljaju svetski standard za zadavanje problema linearnog programiranja u vidu simpleks matrica (tablica). Format je dat sledećom "tabelom":

```
------------------------------------------------------------------
Polje:    1          2            3        4        5       6
Kolone :  2-3       5-12         15-22    25-36    40-47   50-61
          NAME   naziv problema
          ROWS
           tip     ime
          COLUMNS
                  ime           ime    vrednost   ime    vrednost
                  kolone        vrste             vrste
          RHS
                  ime           ime    vrednost   ime    vrednost
                  rhs           vrste             vrste
          RANGES
                  ime           ime    vrednost   ime    vrednost
                  oblasti       vrste             vrste
          BOUNDS
            tip   ime           ime
                  ogranicenja   kolone vrednost
          ENDATA
------------------------------------------------------------------
```

U sekciji `ROWS` (vrste), svaka vrsta mora da ima tip i neko simboličko ime. U sekciji `COLUMNS` (kolone) data su simbolička imena kolona sa simboličkim imenima vrsta i vrednostima (`VALUES`). RHS je deo MPS formata koji opisuje slobodne članove nejednačina posmatranog problema. Postoje još i delovi `RANGES` i `BOUNDS` kojima se zadaje opseg vrednosti koji može da uzme svaka promenljiva (ograničenja u kojima učestvuje samo jedna promenljiva. Više o samom MPS formatu može se naći na internetu, npr [52]. MarPlex je besplatan program i dostupan je na internetu na adresi

`http://tesla.pmf.ni.ac.yu/people/dexter/Software/MarPlex.zip`

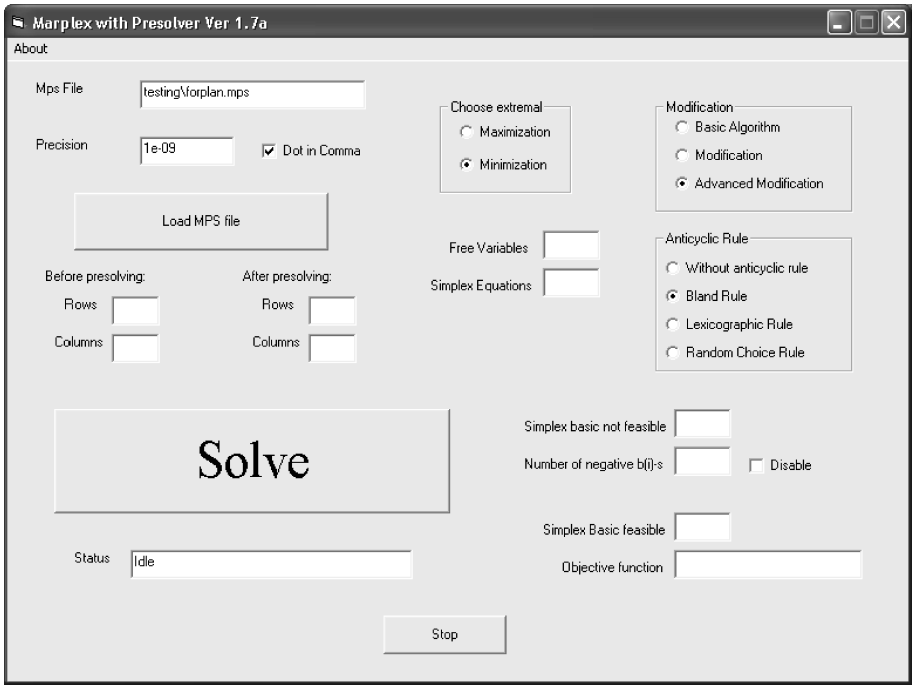
Interface MarPlex-a je prvenstveno prilagođen korisniku i omogućava udoban rad i za razliku od drugih sličnih programa ne opterećuje korisnika brojnim opcijama koje se mahom ne koriste.

Od opcija MarPlex poseduje:

- **Preciznost.** Ovo je jedna od najvažnijih opcija. Ona omogućava korisniku da zada preciznost sa kojom će program raditi. Sve vrednosti koje su po apsolutnoj vrednosti manje od broja ϵ koji se zadaje, MarPlex će tretirati kao da su jednake nuli. Time se sprečava opasan izbor malog pivot elementa u

simpleks metodu. Međutim, ukoliko je ova vrednost isuviše velika, može doći do generisanja pogrešnih rezultata.

- **Maksimizacija/Minimizacija.** Omogućava korisniku da izabere da li se rešava problem maksimizacije ili minimizacije funkcije cilja. Napominjemo da MPS fajl NE SADRŽI ovakvu specifikaciju. Prilikom testiranja uvek se radilo o minimizaciji funkcije cilja.



Slika 3.5.1. Interface programa MarPlex

- **Modifikacija.** Omogućava izbor algoritma za nalaženje početnog bazičnog rešenja. Postoje 3 opcije: Osnovni algoritam, Modifikacija ili Poboljšana modifikacija.
- **Anticiklična pravila.** Omogućava izbor anticikličnog pravila koje će se koristiti. Ponuđene su opcije: Bez pravila, Blandova pravila, Leksikografska metoda i Metoda slučajnog izbora. Kod ove poslednje, pivot se bira na slučajan način pri čemu su sve mogućnosti jednako verovatne.
- **Praćenje toka algoritma.** MarPlex ima mogućnost da se prilikom rada prate nekoliko parametara samog algoritma. To su:

  Trenutni (ukupan) broj iteracija svake faze algoritma.

  Trenutna vrednost funkcije cilja.

  Preostali broj negativnih vrednosti RHS vektora.

  Trenutni status programa (koji se algoritam trenutno izvršava).

Prilikom učitavanja podataka potrebno je naznačiti putanju do MPS fajla kao i opciju preciznost. Nakon toga, pritiskom na taster `Load MPS` vrši se učitavanje MPS fajla i presolving. Presolving je postupak u kome se problem što je moguće više uprošćava, eliminisanjem suvišnih podataka. Detaljnije o presolvingu u linearnom programiranju moše se naći u [93] gde je opisan presolver koji koristi program PCx. Presolver koji smo implementirali u MarPlex-u koristi modifikovanu verziju onog iz [93]. MarPlex ima opciju prikazivanja broja vrsta i kolona problema pre i posle presolvera. Pritiskom na taster `Solve` pokreće se solver i pristupa se rešavanju problema linearnog programiranja.

Program MarPlex smo testirali na referentnim svetskim *Netlib* test problemima. U sledećoj tabeli su prikazani rezultati testiranja. Za svaki problem smo rezervisali tri reda u tabeli: u prvom redu su rezultati postignuti primenom poboljšane modifikacije (algoritam **AdvModNoBasicMax**), zatim klasičnog algoritma **NoBasicMax** i na kraju modifikacije (algoritam **ModNoBasicMax**). Crtica u tabeli ukazuje da je progam dao pogrešan rezultat kao posledicu nagomilavanja računskih grešaka.

Brojevi iteracija za nalaženje bazično dopustivog rešenja, za nalaženje optimalnog rešenja (algoritam **BasicMax**) i ukupan broj iteracija dati su u kolonama označenim redom sa Bf., Sim. and Sum. U zadnjoj koloni su rezultati dobijeni programom PCx [93]. Napomenimo još jednom da je PCx baziran na primal-dual interior point metodu i predstavlja jedan od najjačih i najrobustnijih solvera za problem linearnog programiranja.

Name	Bf.	Sim.	Bf.+Sim.	Objective value	PCx
adlittle	57	44	101	225494.963162364	2.25494963e+005
	77	38	115	225494.96316238	
	21	54	76	225494.963162379	
afiro	4	8	12	−464.753142857143	−4.64753143e+002
	17	5	22	−464.753142857143	
	2	9	11	−464.753142857143	
agg	67	22	89	−35991767.2865765	−3.59917673e+007
	84	31	115	−35991767.2865765	
	38	25	63	−35991767.2865765	
agg2	40	69	109	−20239252.3559771	−2.02392521e+007
	52	64	118	−20239252.3559771	
	31	123	154	−20239252.3559771	
agg3	71	77	148	10312115.9293083	1.03121159e+007
	141	81	222	10312115.7307162	
	51	143	194	10312115.9372015	
bandm	273	159	432	−158.628018177046	−1.58628018e+002
	3128	171	3299	–	
	1495	127	1622	–	
beaconfd	1	33	34	33591.8961121999	3.35924858e+004
	1	33	34	33591.8961121999	
	1	33	34	33591.8961121999	
	1	732	733	−30.769485006264	

Name	Bf.	Sim.	Bf.+Sim.	Objective value	PCx
blend	1	732	733	−30.769485006264	−3.08121498e+001
	1	732	733	−30.769485006264	
brandy	1276	72	1348	1518.50982913344	1.51851054e+003
	2248	81	2329	–	
	624	90	714	1518.50992977114	
capri	251	120	371	2690.01291380796	2.69001291e+003
	214	138	352	2690.01291380796	
	1316	163	1479	2691.57274856721	
czprob	6933	591	7524	2185196.69885648	2.18519682e+006
	11261	635	11886	2185196.69882955	
	6824	648	7472	2185196.69885615	
e226	215	318	533	−18.7519290765415	−1.87519291e+001
	5663	567	6230	–	
	395	364	759	–	
etamacro	191	257	448	−755.715233369051	−7.55715223e+002
	185	176	361	−755.715233352295	
	162	215	377	−755.715233346024	
finnis	141	375	516	172791.065595611	1.72791066e+005
	276	308	584	172791.065595611	
	809	225	1034	172791.03306592	
fit1d	1	834	835	−9146.3780989634	−9.14637809e+003
	1	834	835	−9146.3780989634	
	1	834	835	−9146.3780989634	
ganges	1	420	421	−109585.736129308	−1.09585736e+005
	1	420	421	−109585.736129308	
	1	420	421	−109585.736129308	
gfrd-pnc	229	305	534	6902235.99954881	6.90223600e+006
	240	337	577	6902235.99954882	
	126	311	437	6902235.99954882	
grow15	1	879	880	−106870942.285325	−1.06870941e+008
	1	879	880	−106870942.285325	
	1	879	880	−106870942.285325	
grow22	1	3569	3570	−160871482.230788	−1.60834336e+008
	1	3569	3570	−160871482.230788	
	1	3569	3570	−160871482.230788	
grow7	1	240	241	−47787811.8605706	−4.77878118e+007
	1	240	241	−47787811.8605706	
	1	240	241	−47787811.8605706	
israel	2	157	159	−896644.821863043	−8.96644817e+005
	2	157	159	−896644.821863043	
	2	157	159	−896644.821863043	
kb2	1	50	51	−1749.9001299062	−1.74990013e+003
	1	50	51	−1749.9001299062	
	1	50	51	−1749.9001299062	
lotfi	76	128	204	−25.2647060618762	−2.52647061e+001
	339	158	397	−25.2647060618632	
	111	137	248	−25.2647060618773	

Name	Bf.	Sim.	Bf.+Sim.	Objective value	PCx
recipe	9	30	39	−266.616	−2.66616000e+002
	8	29	37	−266.616	
	9	28	37	−266.616	
sc105	1	56	57	−52.2020612117073	−5.22020612e+001
	1	56	57	−52.2020612117073	
	1	56	57	−52.2020612117073	
sc205	1	135	136	−52.2020612117073	−5.22020612e+001
	1	135	136	−52.2020612117073	
	1	135	136	−52.2020612117073	
sc50a	1	26	27	−64.5750770585645	−6.45750771e+001
	1	26	27	−64.5750770585645	
	1	26	27	−64.5750770585645	
sc50b	1	29	28	−70	−7.00000000e+001
	1	29	28	−70	
	1	29	28	−70	
scagr7	81	41	122	−2331389.82433099	−2.33138982e+006
	90	35	125	−2331389.82433097	
	69	26	95	−2331389.82433098	
scfxm1	1133	97	1220	18417.3255500362	1.84167590e+004
	2478	200	2878	–	
	311	123	434	18416.7590283489	
scorpion	90	38	128	1878.12482273811	1.87812482e+003
	114	37	151	1878.12482273811	
	70	70	140	1878.12482273811	
sctap1	320	8	328	1412.25	1.41225000e+003
	496	57	553	1412.24999999998	
	131	137	268	1412.25	
sctap2	739	195	934	1724.80714285713	1.72480714e+003
	739	195	934	1724.80714285713	
	739	195	934	1724.80714285713	
sctap3	469	252	721	1424	1.42400000e+003
	618	247	865	1424	
	369	909	1278	1424	
seba	79	32	111	15711.6000000006	1.57116000e+004
	90	40	130	15711.5999999923	
	–	–	–	–	
share1b	89	65	154	−76589.3185791853	−7.65893186e+004
	366	69	435	−76589.3224159041	
	368	63	431	−76589.3185791526	
share2b	135	38	173	−415.73224074142	−4.15732241e+002
	123	46	169	−415.732240741419	
	92	25	117	−415.732240741416	
shell	41	276	317	1208825346	1.20882535e+009
	55	279	334	1208825346	
	78	278	356	1208825346	
ship04l	8	251	259	1793324.53797036	1.79332454e+006
	450	124	574	1793324.53797036	

Name	Bf.	Sim.	Bf.+Sim.	Objective value	PCx
	100	374	474	1793324.53797035	
	14	172	186	1798714.70044539	
ship04s	116	185	301	1798714.70044539	1.79871471e+006
	57	194	251	1798714.70044539	
	320	530	850	1909055.21138913	
ship08l	461	364	825	1909055.21138913	1.90905521e+006
	144	631	775	1909055.21138913	
	54	258	312	1920098.21053462	
ship08s	169	239	408	1920098.21053462	1.92009821e+006
	67	272	339	1920098.21053462	
	49	1019	1068	1470187.91932926	
ship12l	938	1908	2846	1470187.91932926	1.47018797e+006
	232	1711	1943	1470187.91932926	
	55	439	494	1489236.13440613	
ship12s	429	556	985	1489236.13440613	1.48923613e+006
	166	486	632	1489236.13440613	
	75	326	401	15394362.1836319	
sierra	65	325	490	15381546.3836319	1.53943622e+007
	82	310	392	15394362.1836319	
	2450	44	2494	−251.26695098074	
stair	686	33	719	−251.266951192317	−2.51266951e+002
	11066	168	11234	−	
	21	138	159	1257.6995	
standata	76	98	174	1257.6995	1.25769951e+003
	146	116	262	1257.69949999999	
	131	109	240	1406.0175	
standmps	260	155	415	1406.01749999996	1.40601750e+003
	752	72	824	1406.0175	
	1	17	18	−41131.9762194364	
stocfor1	1	17	18	−41131.9762194364	−4.11319762e+004
	1	17	18	−41131.9762194364	
	69	55	124	129831.462637412	
vtp.base	179	71	250	129831.462461362	1.29831463e+005
	430	47	477	129831.464051472	

Tabela 3.5.1.

Analizirajući tabelu, možemo primetiti da su na skoro svim primerima modifikacije dale bolje rezultate od klasičnog simpleksa. Čak i ako se posmatra ukupan broj iteracija (kolona Sum.), ponovo su modifikacije bolje. Primetimo takođe da su modifikacija i poboljšana modifikacija skoro izjednačene. Upravo ova činjenica opravdava razmatranje originalne verzije modifikacije iz rada [78] u ovom radu, a i takođe dodatno zakomplikuje situaciju oko izbora algoritma za određivanje početnog bazično dopustivog rešenja. Međutim, ova originalna verzija je na 5 primera dala pogrešno rešenje. Klasični algoritam je na 3 primera dao pogrešan rezultat, dok je poboljšana modifikacija uspešno rešila sve primere iz ove klase. Na osnovu svega rečenog, možemo zaključiti da su se obe modifikacije dobro pokazale

u praksi, kao i da odgovor na pitanje koja je od njih bolja u mnogome zavisi od same strukture i prirode ulaznih podataka.

Revidirani simpleks metod (algoritmi **RevBasicMax** i **RevNoBasicMax**) kao i modifikacija (algoritam **ModRevNoBasicMax**) implementirani su u programskom jeziku MATHEMATICA. Tako je nastao program RevMarPlex, varijanta MarPlex-a koja koristi revidirani simpleks metod. Za razliku od MarPlex-a koji je implementiran u proceduralnom programskom jeziku, kod RevMarPlex-a smo se odlučili za paket MATHEMATICA prvenstveno zbog integrisanog solvera za sisteme linearnih jednačina (funkcija `LinearSolve`, [90]). Već smo napomenuli da je za implementaciju revidiranog simpleksa najpre neophodan brz i robustan (otporan na numeričke greške) solver za sisteme linearnih jednačina. Pored toga, kod RevMarPlex-a koristili smo prednosti MATHEMATICA-e po pitanju ulaznih i izlaznih podataka (ovde se funkcija cilja i ograničenja zadaju u svom prirodnom obliku).

Glavni nedostatak programa RevMarPlex je brzina rada. Dok MarPlex skoro sve probleme rešava u trenutku, RevMarPlex-u je često trebalo dosta vremena da reši problem. Da bi ovo objasnili, prvo moramo uzeti u obzir da se programi napisani u MATHEMATICA-i izvršavaju znatno sporije od odgovarajućih programa u proceduralnim jezicima. Međutim, to nije glavni razlog zašto je RevMarPlex znatno sporiji od MarPlex-a. Naime, prethodno smo već spomenuli da je iteracija revidiranog simpleks metoda algoritamski znatno kompleksnija od odgovarajuće iteracije simpleksa zato što je kod revidiranog simpleksa potrebno rešiti nekoliko sistema jednačina. Takođe, MarPlex koristi poboljšanje algoritma **Replace**, tj koristi sparse strukturu Tuckerove tabele dok kod RevMarPlex-a nemamo poboljšanje tog tipa.

U sledećoj tabeli su prikazani rezultati testiranja programa RevMarPlex na primerima manjih dimenzija

Problem	PCx	RevMarPlex	Mod.	Klas. alg.
Adlittle	2.25494963×10^{5}	225494.963162	32	39
Afiro	-4.64753143×10^{2}	-464.753142	2	17
Agg	3.59917673×10^{7}	-35991767.286576	151	151
Agg2	-2.0239251×10^{7}	-20239252.3559776	75	129
Blend	-3.08121498×10^{1}	-30.812150	-	-
Sc105	$-5.2202061212\times10^{1}$	-52.202061	-	-
Sc205	-5.22020612×10^{1}	-52.202061	-	-
Sc50a	$-6.4575077059\times10^{1}$	-64.575077	-	-
Sc50b	-7.000000000×10^{1}	-70	-	-
Scagr25	-1.47534331×10^{7}	-14753433.060769	520	> 1500
Scagr7	-2.33138982×10^{6}	-2331389.824330	55	74
Stocfor1	$-4.1131976219\times10^{4}$	-41131.976219	14	17
LitVera	1.999992×10^{-2}	0	-	-
Kb2	-1.7499×10^{3}	-1749.9001299062	-	-
Recipe	-2.66616×10^{2}	-266.6160	13	15
Share1B	-7.65893186×10^{2}	76589.3185791859	498	> 1500

Tabela 3.5.2.

Kao što možemo videti i u ovom slučaju se modifikacija pokazala bolja od klasičnog algoritma.

Oba programa smo testirali na još jednoj klasi ekstemno loše uslovljenih primera *KBAPAH* preuzetih iz [4], [42]. Ovi primeri se mogu naći i na internet stranici `www.psmtmath.s5.com`. Program PCx nije uspeo da reši ove primere. U sledećoj tabeli su prikazane optimalne vrednosti funkcije cilja izračunate pomoću MarPlex-a i RevMarPlex-a. Primeri su konstruisani tako da su optimalne vrednosti funkcija cilja jednaki 0.

Problem	RevMarPlex	MarPlex
07-20-02	0	0
15-30-03	7.2345×10^{-9}	2.16968×10^{-5}
15-30-04	2.16968×10^{-5}	1.3984×10^{-3}
15-30-06	4.90359×10^{-6}	1.671×10^{-3}
15-30-07	3.2807×10^{-4}	1.749×10^{-3}
15-60-07	0	-6.29305×10^{-7}
15-60-09	-6.29305×10^{-7}	-1.8353×10^{-6}
20-40-05	0	1.63948×10^{-6}
30-60-05	0	1.64567×10^{-11}
30-60-08	0	5.22527×10^{-5}
LitVera	0	0

Tabela 3.5.3.

Primećujemo da je u svim primerima RevMarPlex postigao bolje rezultate od MarPlex-a. Napominjemo da se kondicioni brojevi ($k = \|A\|\|A^{-1}\|$) kod ovih primera kreću u intervalu $(10^{15}, 10^{20})$.

3.6 Postoptimalna analiza

Često se javlja potreba da treba rešiti više problema linearnog programiranja koji su blisko povezani. Postoji više razloga zbog kojih se dolazi do takvih problema. Na primer, podaci koji definišu problem mogu biti nesigurni ili dati sa približnim vrednostima. Tada je poželjno razmotriti različite mogućnosti za polazne podatke. I u slučaju kada su polazni podaci dati sa tačnim vrednostima, može se desiti da se ti podaci menjaju svakodnevno, tako da se problem mora nanovo rešavati. U oba slučaja situacija je slična. Dakle, pitanje je da li prethodno dobijeno optimalno rešenje možemo iskoristiti da bi smo brže dobili rešenje novog sličnog problema. Naravno, odgovor je potvrdan u većini slučajeva.

Razmotrimo problem zadat u matričnom obliku

$$\begin{aligned} \max \quad & c^T x, \\ \text{p.o.} \quad & Ax = b, \\ & x \geq 0. \end{aligned}$$

Odgovarajuću simpleks tabelu koristeći ranije uvedene oznake možemo zapisati u obliku

$$d = c_B^T A_B^{-1} b - (c_B^T A_B^{-1} A_N - c_N^T) x_N,$$
$$x_B = A_B^{-1} b - A_B^{-1} A_N x_N,$$

ili korišćenjem oznaka

$$d' = c_B^T A_B^{-1} b, \quad (c'_N)^T = c_B^T A_B^{-1} A_N - c_N^T, \quad b' = A_B^{-1} b,$$

u obliku

$$d = d' - (c'_N)^T x_N, \tag{3.6.1}$$
$$x_B = b' - A_B^{-1} A_N x_N.$$

Neka je rešenje dato simpleks tabelom 3.6.1 optimalno. Pretpostavimo da se koeficijenti ciljne funkcije menjaju sa c na c'. Primetimo da se pri tome b' ne menja, tj. novo rešenje je takođe bazično dopustivo, dakle može se upotrebiti kao početno rešenje. Ako je razlika između c i c' mala, možemo očekivati da se novo optimalno rešenje dobije u relativno malom broju koraka.

Pretpostavimo sada da se c ne menja ali da se menja vektor b. U tom slučaju se $(c'_N)^T$ ne menja. Dakle, nova tabela je dualno dopustiva i primenom dualnog simpleks metoda dobijamo novo optimalno rešenje u malom broju koraka.

Dakle, promena vektora c ili b ne utiče na dopustivost optimalnog rešenja (3.6.1). Razmotrimo sada slučaj kada se menjaju vrednosti svih polaznih podataka. U tom slučaju se menjaju d', $(c'_N)^T$, b' ali se mogu promeniti i vrednosti matrica A_B i A_N. Ako je nova bazična matrica A_B nesingularna, možemo i dalje zadržati staru podelu na bazične i nebazične promenljive. Nova simpleks tabela najčešće neće biti ni primalno ni dualno dopustiva, ali ako je perturbacija polaznih podataka relativno mala, možemo očekivati da ta polazna tabela dovede do optimalnog rešenja u manjem broju iteracija u odnosu na klasično rešavanje. Iako ne postoji dokaz da bilo koji od tih takozvanih vrućih startova smanjuje broj potrebnih iteracija, iz empirijskih podataka je očigledno da ova procedura često dovodi do značajnog poboljšanja. Ponekad se vrućim startom problem reši i sto puta brže u odnosu na originalni simpleks metod.

Jedan od čestih problema postoptimalne analize se sastoji u sledećem: u kojim granicama se mogu menjati koeficijenti ciljne funkcije a da se pri tome ne naruši podela na bazične i nebazične promenljive. Pretpostavimo da se vektor c menja za $t\Delta c$, gde je t realan broj i Δc dat vektor (pri čemu je najčešće samo jedna koordinata različita od nule, ali diskusija važi i opštem slučaju). Tada se c'_N povećava za $t\Delta c'_N$, gde je

$$\Delta c'_N = \Delta c_B^T A_B^{-1} A_N - \Delta c_N^T.$$

Dakle, bazične promenljive se ne menjaju sve dok je

$$c'_N + t\Delta c'_N \geq 0.$$

Za $t > 0$ nejednakost važi ako je

$$t \leq \left(\max_{j \in N} -\frac{\Delta c'_j}{c'_j} \right)^{-1}.$$

Analogno, za $t < 0$, donja granica je

$$t \geq \left(\min_{j \in N} -\frac{\Delta c'_j}{c'_j} \right)^{-1}.$$

Iz ove dve nejednakosti dobijamo interval kome mora da pripada t

$$\left(\min_{j \in N} -\frac{\Delta c'_j}{c'_j} \right)^{-1} \leq t \leq \left(\max_{j \in N} -\frac{\Delta c'_j}{c'_j} \right)^{-1}.$$

Primer 3.6.1 [82] Razmotrimo problem linearnog programiranja

$$\begin{array}{rrcl} \max & 5x_1 + 4x_2 + x_3 & & \\ \text{p.o.} & 2x_1 + 3x_2 + x_3 & \leq & 5 \\ & 4x_1 + x_2 + 2x_3 & \leq & 11 \\ & 3x_1 + 4x_2 + 2x_3 & \leq & 8 \\ & x_1, x_2, x_3 & \geq & 0. \end{array}$$

Optimalna tabela za ovaj problem je

$$\begin{aligned} d &= 13 - 3x_2 - x_4 - x_6 \\ x_3 &= 1 + x_2 + 3x_4 - 2x_6 \\ x_1 &= 2 - 2x_2 - 2x_4 + x_6 \\ x_5 &= 1 + 5x_2 + 2x_4. \end{aligned}$$

Optimalna baza je $\{3, 1, 5\}$. Ispitajmo u kojim granicama se može promeniti koeficijent ciljne funkcije uz x_1 a da se optimalna baza ne promeni. Kako je $c = (5, 4, 3, 0, 0, 0)$, stavićemo $\Delta c = (1, 0, 0, 0, 0, 0)$. Sada je

$$\Delta c_B = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{i} \quad \Delta c_N = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Kako je

$$\Delta c'_N = \Delta c_B^T A_B^{-1} A_N - \Delta c_N^T,$$

i iz optimalne tabele je

$$A_B^{-1} A_N = \begin{bmatrix} -1 & -3 & 2 \\ 2 & 2 & -1 \\ -5 & -2 & 0 \end{bmatrix},$$

to je

$$\Delta c'_N = (2, 2, -1).$$

Dakle, granice za t određujemo iz nejednačina

$$3 + 2t \geq 0, \quad 1 + 2t \geq 0, \quad \text{i} \quad 1 - t \geq 0.$$

Odatle je $-1/2 \leq t \leq 1$, pa se koeficijent ciljne funkcije koji stoji uz x_1 može kretati u granicama od $4,5$ do 6.

Pretpostavimo sada da se menjaju vrednosti koordinata vektora b, tj. da se b menja za $t\Delta b$. Sada se c'_N ne menja ali se b' promeni za $t\Delta x_B$ gde je

$$\Delta x_B = A_B^{-1}\Delta b.$$

Dakle, baza ostaje optimalna sve dok t zadovoljava nejednačinu

$$\left(\min_{i\in B} -\frac{\Delta x_i}{b'_i}\right)^{-1} \leq t \leq \left(\max_{i\in B} -\frac{\Delta x_i}{b'_i}\right)^{-1}.$$

4 Tri direktna metoda u linearnom programiranju

Osim klasičnog simpleks metoda postoje i alternativni metodi za rešavanje problema linearnog programiranja koji se zasnivaju ili na geometrijskim osobinama skupa ograničenja [14, 20] ili na uopštenim inverzima [67, 11, 64, 63, 65]. Pomenimo i radove koji se odnose na Banachove prostore [43, 44]. Rezultate sličnog tipa opisujemo u ovom poglavlju i uvodimo tri direktna metoda za rešavanje problema linearnog programiranja koji zadovoljavaju izvesne uslove. Prvi metod pronalazi početnu tačku koja predstavlja ili ekstremnu tačku ili tačku koja je susedna ekstremnoj tački. Drugi metod se zasniva na teoriji igara a treći na uopštenim inverzima. U slučaju da uslovi za direktno nalaženje ekstremne tačke nisu zadovoljeni, ovi metodi se mogu primeniti za konstrukciju početne tačke za klasičan simpleks metod. Efikasnost poboljšanog simpleks metoda, sa početnom tačkom dobijenom primenom novih metoda, upoređena je sa originalnim simpleks metodom kao i sa metodima unutrašnje tačke, što je ilustrovano sa nekoliko karakterističnih primera. Osim toga, razmatra se i eliminacija suvišnih ograničenja u problemu linearnog programiranja.

Do kraja ovog odeljka sledimo radove [75, 76, 77] i [67].

4.1 Osnovni pojmovi

Razmotrimo problem linearnog programiranja u kome su ograničenja nejednakosti. Odrediti maksimum linearne ciljne funkcije

$$z(x) = \sum_{j=1}^{n} c_j x_j = cx \tag{4.1.1}$$

u odnosu na linearna ograničenja

$$\begin{aligned} \sum_{j=1}^{n} a_{ij}x_j &= r_i x \leq b_i, \quad i = 1,\ldots,m \\ x_j &\geq 0,\ j = 1,\ldots,n \end{aligned} \tag{4.1.2}$$

gde je $r_i x$, $i=1,\ldots,m$ skalarni proizvod vektora r_i i x, i

$$c=(c_1,\ldots,c_n),\ x=(x_1,\ldots,x_n),\ r_i=(a_{i1},\ldots,a_{in}),\quad i=1,\ldots,m.$$

Razmatramo sledeći problem: generisati pogodnu početnu tačku za simpleks metod, sa ciljem da se redukuje potreban broj iterativnih koraka za pronalaženje optimalnog rešenja. Dobijen je metod koji je primenljiv na probleme linearnog programiranja koji su definisani bez suvišnih uslova. Pokazuje se da je isti metod ponekad primenljiv i na probleme linearnog programiranja kod kojih postoje suvišni uslovi.

U nastavku uvodimo metod (takozvani *metod minimalnih uglova*), za ubrzanje simpleks metoda kod nekih klasa problema linearnog programiranja. Metod u najgorem slučaju daje pogodnu tačku za start standardnog simpleks algoritma. Ta tačka je rešenje linearnog sistema koji se dobija posle zamene $l \leq n$ izabrnih ograničenja sa odgovarajućim jednačinama pri čemu su $n - l$ promenljivih izjednačene sa nulom. Za neke probleme linearnog programiranja simpleks metod samo potvrđuje da je izabrana tačka optimalno rešenje i algoritam se završava u samo jednom koraku. U najgorem slučaju, metod minimalnih uglova daje teme konveksnog skupa koje je u susedstvu sa ekstremnom tačkom. Na taj način metod minimalnih uglova obezbeđuje u izvesnim slučajevima značajnu redukciju broja iterativnih koraka u odnosu na klasičan simpleks metod. Osim toga, pored ubrzanja konvergencija, za određene tipove problema se smanjuje i dimenzija problema.

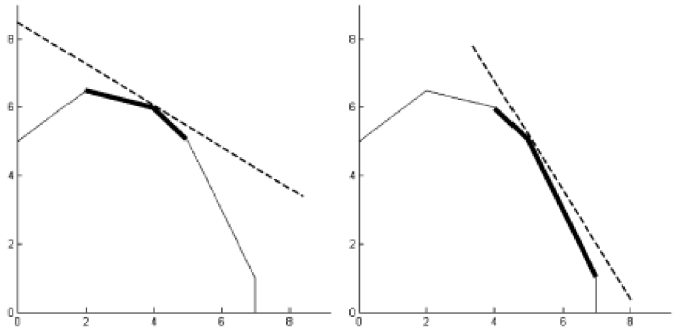
Osnovni preduslov za uspešnu primenu metoda minimalnih uglova je da je dati problem bez suvišnih ograničenja. Iz tih razloga, razmatramo i problem eliminacije suvišnih ograničenja. Takođe uvodi se jedan direktan metod koji se zasniva na teoriji igara. Kod nekih problema linearnog programiranja rešenje se može dobiti neposredno, dok se kod nekih problema njegovom primenom eliminišu suvišna ograničenja.

U četvrtom odeljku su razvijeni odgovarajući algoritmi za uvedene metode. Takođe su opisani najvažniji detalji implementacije metoda minimalnih uglova u programskom paketu MATHEMATICA.

Efikasnost predstavljenih metoda je pokazana na nekoliko ilustrativnih primera. Takođe je na tim primerima izvršeno upoređivanje sa standardnom simpleks metodom kao i sa metodima unutrašnje tačke.

4.2 Metod minimalnih uglova

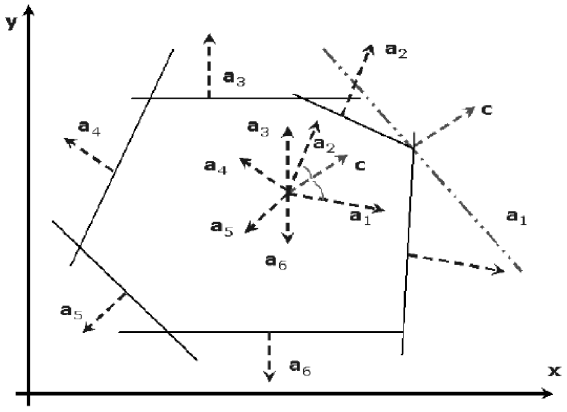
Glavna ideja u radu [76] jeste da se poboljša izbor inicijalne baza. Poznato je da se optimalno teme formira kao presek n ograničenja, gde je n broj promenljivih u LP. Ovakvih n ograničenja koja formiraju optimalno teme trebalo bi da zahvataju najmanje uglove sa ciljnom funkcijom. Ovakvo tvrđenje je ilustrovano na Slici 4.2.1, na kojoj isprekidana linija predstavlja ciljnu funkciju a boldovane linije predstavljaju ograničenja koja su najbliža prema zahvaćenom uglu sa ciljnom funkcijom.



Slika 4.2.1. Ilustracija ideje minimalnih uglova.

Primer 4.2.1 Posmatra se linearni problem

$$\begin{array}{ll} \max & z = x + y \\ \text{p.o.} & 3x - y \leq 70 \\ & x + 4y \leq 100 \\ & y \leq 20 \\ & -10x + 3y \leq -20 \\ & -5x - 3y \leq -55 \\ & -y \leq 5 \end{array}$$



Slika 4.2.2. Ilustarcija metoda minimalnih uglova.

Jedinstveno rešenje ovog problema je $x^* = (29.23, 17.69)$. Prvo i drugo ograničenje u ovom problemu jesu jedina aktivna ograničenja u x^*. Na slici 4.2.2 se vidi da su uglovi između gradijenata za funkcije date prvim i drugim ograničenjem i gradijenta ciljne funkcije c minimalni između uglova svih ograničenja i c.

Za opis metoda minimalnih uglova potrebna je sledeća definicija.

Definicija 4.2.1 *Neka je $P \subseteq \mathbb{R}^n$ poliedar (konveksan skup ili konus) definisan bez suvišnih nejednakosti sa:*

$$P: \quad \sum_{j=1}^{n} a_{ij} x_j = r_i x \leq b_i, \quad i = 1, \ldots, m.$$

Tangencijalni poliedar P^0 poliedra P je definisan sledećim skupom nejednakosti

$$P^0: \quad \sum_{j=1}^{n} a_{ij}x_j = r_i x \le |r_i|, \quad |r_i| = \sqrt{a_{i1}^2 + \cdots + a_{in}^2},\ i = 1, \ldots, m.$$

Metod predstavljen u sledećoj teoremi je direktan i za pogodno definisane probleme linearnog programiranja (4.1.1)-(4.1.2) daje optimalnu tačku u samo jednom koraku. Opisaćemo sada glavnu ideju tog metoda. U slučaju da je problem linearnog programiranja sa samo dve promenljive, možemo primeniti grafički postupak za rešavanje [66, 84]. Ako su ograničavajući uslovi dati nejednakostima, svaka od odgovarajućih pravih deli ravan na skup dopustivih i skup nedopustivih tačaka za date uslove. Dopustive tačke se nalaze u oblasti P koja zadovoljava sve date uslove. Optimalno rešenje je sada moguće odrediti crtanjem grafika modifikovane ciljne funkcije $z(x_1, x_2) = 0$ i njenim paralenim pomeranjem u smeru gradijentnog vektora (c_1, c_2). Optimalno rešenje je jedinstveno ako prava $z(x_1, x_2) = z_{max}$ prolazi kroz samo jedno teme (ekstremnu tačku) dopustive oblasti. U slučaju minimizacionog problema, datu pravu pomeramo paralelno u suprotnom smeru.

Poznato je da u n-dimenzionalnom slučaju teme dopustivog poliedra određujemo rešavanjem odgovarajućeg sistema sa n jednačina datih ograničavajućim uslovima. U sledećoj teoremi uvodimo kriterijum za pogodan izbor tih jednačina, koji se bazira na generalizaciji i formalizaciji grafičkog metoda. Time dobijamo glavni rezultat ovog odeljka.

Teorema 4.2.1 [*76*] *Neka je dat maksimizacioni problem linearnog programiranja* (4.1.1)-(4.1.2) *bez suvišnih nejednakosti. Neka je $P \subseteq \mathbb{R}^n$ poliedar definisan sa* (4.1.2). *Sa $c = (c_1, \ldots, c_n)$ označavamo gradijent vektor ciljne funkcije. Vektori r_i su definisani sa $r_i = (a_{i1}, \ldots, a_{in})$, $i = 1, \ldots, m$. Razmotrimo skup*

$$V = \left\{ v_i = \frac{cr_i}{|r_i|},\ |r_i| = \sqrt{a_{i1}^2 + \cdots + a_{in}^2}, \quad i = 1, \ldots, m \right\}. \tag{4.2.1}$$

Pretpostavimo da skup V sadrži l pozitivnih elemenata, označenih sa $v_{i_1}, \ldots v_{i_l}$.

Razlikujemo sledeće slučajeve:

(a) *Ako je P prazan skup, ne postoji rešenje datog problema.*

(b) *U slučaju da je $l = 0$, tada je $z_{\max} = +\infty$, gde $z_{\max}$ označava maksimalnu vrednost ciljne funkcije $z(x)$.*

(c) *U slučaju da je $l \ge n$, neka je x_0 rešenje sledećeg sistema jednačina:*

$$\begin{aligned} &a_{i_1,1}x_1 + \cdots + a_{i_1,n}x_n = r_{i_1}x = b_{i_1} \\ &\cdots \qquad\qquad \cdots \\ &a_{i_n,1}x_1 + \cdots + a_{i_n,n}x_n = r_{i_n}x = b_{i_n}, \end{aligned} \tag{4.2.2}$$

pri čemu indeksi $i_1, \ldots, i_n$ odgovaraju skupu od n maksimalnih i pozitivnih vrednosti izabranih iz skupa V. Ako optimalnu tačku u P označimo sa x_P, mogu da nastupe sledeći slučajevi:

(i) $x_0 = x_P$, *ili*

(ii) x_0 *i* x_P *pripadaju istoj graničnoj hiperravni poliedra* P.

(d) *U slučaju da je* $0 < l < n$, *razmotrimo sledeći sistem*

$$\begin{aligned} &a_{i_1,1}x_1 + \cdots + a_{i_1,n}x_n = r_{i_1}x = b_{i_1} \\ &\cdots \qquad\qquad \cdots \\ &a_{i_l,1}x_1 + \cdots + a_{i_l,n}x_n = r_{i_l}x = b_{i_l}, \end{aligned} \tag{4.2.3}$$

gde indeksi $i_1, \ldots, i_l$ *odgovaraju pozitivnim vrednostima* $v_{i_1}, \ldots, v_{i_l}$ *iz skupa* V. *Bazično rešenje* x_0 *problema* (4.1.1)-(4.1.2) *dobija se izjednačavanjem* $n - l$ *promenljivih sa nulom i rešavanjem* l *jednačina iz* (4.2.3) *Tada* x_0 *i* x_P *pripadaju istoj hiperravni poliedra* P.

Primer 4.2.2 Da bismo ilustrovali geometrijski smisao Teoreme 4.2.1, razmotrimo sledeći problem:

$$\begin{aligned} \max\ & z(x_1, x_2) = x_1 + x_2 \\ \text{p.o.}\ & (C_1) \qquad x_1/3 + x_2 \le 1 \\ & (C_2) \qquad x_1 + x_2 \le 2 \\ & (C_3) \qquad x_1 + x_2/3 \le 1. \end{aligned}$$

Minimalan ugao je zahvaćen između gradijentnog vektora $c = (1, 1)$ ciljne funkcie i prave $H_2 \equiv x_1 + x_2 = 2$. Uglovi između vektora c i pravih $H_1 \equiv x_1/3 + x_2 = 1$ i $H_3 \equiv x_1 + x_2/3 = 1$ su identični. Ako uzmemo dva ugla između vektora c i hiperravni H_2 i H_3, tada rešenje $x_0 = H_2 \cap H_3$ zadovoljava $x_0 = (1/2, 3/2) \notin P$. Slično, ako uzmemo hiperravni H_2 i H_1, dobijamo $x_0 = H_2 \cap H_1 = (3/2, 1/2) \notin P$. Optimalno rešenje x_P je u temenu $x_P = (3/4, 3/4)$. Međutim, x_0 i x_P su zajednički elementi na pravoj H_2. Uslov (C_2) je suvišan. Ako se uslov (C_2) eliminiše, dobijamo $x_0 = x_P = (3/4, 3/4)$.

Napomena 4.2.1 Ako je poliedar P definisan bez suvišnih ograničenja, tada Teorema 4.2.1 može biti uspešno primenjena. Ako sistem nejednačina (4.1.2) sadrži nekoliko suvišnih ograničenja, moguće je da su neki od minimalnih uglova u (4.2.1) određeni suvišnim ograničenjima. U tom slučaju može da se desi da $x_0 \notin P$, tako da tačke x_0 i x_P nisu elementi iste hiperravni od P. Naravno, i u tom slučaju tačka x_0 može biti iskorišćena kao početna tačka za simpleks metod, ali ubrzanje konvergencije nije garantovano u tom slučaju. Ustvari, za uspešnu primenu Teoreme 4.2.1, dovoljno je da nijedan od l minimalnih uglova nije zahvaćen sa nekim suvišnim ograničenjem. To daje motivaciju za eliminaciju suvišnih ograničenja, što se razmatra u narednom odeljku.

Napomena 4.2.2 Metod koji koristimo u slučaju $0 < l < n$ je takođe moguće primeniti i u slučaju $m < n$. U tom slučaju, predloženi metod je modifikacija pozantog metoda za konstrukciju bazičnog rešenja iz [10]. Primetimo da je u [10] bazično rešenje problema (4.1.1)-(4.1.2) za koji važi $m < n$, dobijeno izjednačavanjem $n - m$ promenljivih sa nulom i rešavanjem m jednačina.

Napomena 4.2.3 Metod minimalnih uglova daje bazično rešenje koje pripada istoj hiperravni kao i ekstremna tačka. Osim ove značajne osobine, naglasimo sledeće

značajno svojstvo ovog metoda. U simpleks metodu sva ograničenja se koriste u svakom koraku kao i dodate izravnavajuće promenljive. Kod metoda minimalnih uglova broj aktivnih ograničenja je manji u odnosu na standardni simpleks algoritam. Još više, kod metoda minimalnih uglova, izravnavajuće promenljive se ne koriste. Prema tome, dimenzija problema, razmatrana kod metoda minimalnih uglova, je značajno manja u odnosu na dimenziju odgovarajućeg problema kod primene simpleks metoda. Dakle, zamena nekoliko iteracija simpleks metoda sa samo jednom primenom metoda minimalnih uglova, obično značajno smanjuje broj potrebnih operacija i procesorsko vreme.

Napomena 4.2.4 U radu [88] dat je kontraprimer gde Teorema 4.2.1 ne važi. Kasnije je u svojoj magistarkoj tezi Wang [85] pokazao da primer iz [88] nije korektan jer sadrži suvišna ograničenja i zatim dao koretan kontraprimer. I pored toga, na osnovu radova [53, 54, 55, 56], Teorema 4.2.1 može da se koristi kao uspešna heuristika za pronalaženje početne tačke simpleks algoritma ili, u pogodnim uslovima, optimalne tačke linearnog problema. Pomenimo još i to da je rad [75] osim u [88, 85] citiran i u radovima [2, 3, 13, 15, 16, 26, 27, 30, 32, 45, 57, 68, 86].

4.3 Suvišna ograničenja i primena teorije igara

U početku ovog odeljka uvodimo nekoliko pravila za eliminaciju suvišnih ograničenja. Ta pravila su korisna i u slučaju transformacije problema (4.1.1)-(4.1.2), na oblik pogodan za primenu simpleks metoda. Ovaj odeljak je baziran na radovima [75] i [76].

Poznato je da Gausova eliminacija ili QR faktorizacija može biti primenjena za eliminaciju suvičnih ograničenja [18, 84]. Međutim, pri tome se javljaju sledeće poteškoće.

A. Pre primene Gausove eleiminacije, nejednakosti moraju biti transformisane u odgovarajuće jednakosti. Na taj način se dimenzija sistema na koji se primenjuje Gausova eliminacija obično značajno povećava.

B. Još više, greške zaokruživanja i potreban broj aritmetičkih operacija su značajni u mnogim slučajevima.

C. Postupak Gausove eliminacije prepoznaje samo linearno zavisna ograničenja tipa jednakosti. Primena simpleks algoritma obično zahteva uvođenje dopunskih promenljivih. U tom ekvivalentnom obliku, matrica ograničenja A je obično potpunog ranga, tako da problem nepotpunog ranga nije od velikog značaja u praksi [93].

Implementacija metoda unutrašnje tačke za probleme linearnog programiranja velikih dimenzija obično sadrži pripremnu fazu za eliminaciju suvišnih promenljivih i suvišnih ograničenja [1]. Na primer, pripremna faza programa PCx izvršava nekoliko tipova eliminacija suvišnih promenljivih i suvišnih ograničenja [19].

Pripremna faza u programu PCx radi sa problemima linearnog programiranja oblika

$$\min_{x\in\mathbb{R}^n} c^T x \quad \text{u odnosu na} \quad Ax = b,$$
$$\begin{cases} 0 \le x_i, & i \in \mathcal{N} \\ 0 \le x_i \le u_i, & i \in \mathcal{U} \\ x_i \text{ slobodno}, & i \in \mathcal{F}, \end{cases}$$

gde je $\mathcal{N}\cup\mathcal{U}\cup\mathcal{F}$ particija indeksnog skupa $\{1,\ldots,n\}$ [19]. Pripremna faza proverava ulazne podatke u odnosu na sledeća eliminaciona pravila [19].

Prazne vrste. Ako matrica A ima nula vrstu i odgovarajuću nula koordinatu u vektoru b, ta vrsta se može izostaviti iz razmatranja.

Duplirane vrste. Kada je vrsta matrice A (i odgovarajući element iz vektora ograničenja b) proporcionalna nekoj drugoj vrsti, može biti izostavljena.

Duplirane kolone. Kada je kolona matrice A proporcinalna nekoj drugoj koloni, te dve kolone je moguće objediniti. Ulazne promenljive objedinjenih kolona su ili normalne ili slobodne, što zavisi od toga da li je faktor proporcionalnosti kolona pozitivan ili negativan.

Fiksirane promenljive. Ako promenljiva ima nulu za gornje i donje ograničenje, očigledno je da je moguće tu promenljivu izjednačiti sa nulom i izostaviti iz problema.

Jednoelementne vrste. Ako i-ta vrsta matrice A sadrži samo jedan element A_{ij} različit od nule, jasno je da je $x_j = b_i/A_{ij}$, tako da ta promenljiva može biti eliminisana iz problema. Takođe se i-ta vrsta matrice A može izostaviti.

Jednoelementne kolone. Kada je A_{ij} jedini element različit od nule u koloni j matrice A, i x_j je slobodna promenljiva, tada je promenljivu x_j moguće izraziti preko drugih promenljivih koje se javljaju u i-toj vrsti matrice A i eliminisati je iz problema. Čak i ako promenljiva nije slobodna, x_j može biti eliminisana ako su njena ograničenja slabija od ograničenja koja slede iz domena ostalih elemenata koji figurišu u vrsti A_i. Slična tehnika je upotrebljena u [1]. Međutim, neka od suvišnih ograničenja nisu obuhvaćenja razmatranjima opisanim u [1] i [19]. Kako je napomenuto u [1], u slučaju problema linearnog programiranja velikih dimenzija nije moguće otkloniti sve suvišne nejednakosti manuelno. Dakle, analiza pripremne faze ima za cilj poboljšanja formulacije problema.

U tom cilju, u ovom odeljku predlažemo nekoliko dodatnih pravila za eliminaciju suvišnih ograničenja. Prvo pravilo se bazira na skoro očiglenim geometrijskim osobinama nekih suvišnih ograničenja.

Razmotrimo sledeća ograničenja u odnosu na (4.1.2):

$$\begin{aligned} \frac{1}{b_i} r_i x = \sum e_{ij} x_j &\le 1, \quad i = 1,\ldots,m \\ x_j &\ge 0, \quad j = 1,\ldots,n. \end{aligned} \tag{4.3.1}$$

gde je $e_{ij} = \dfrac{a_{ij}}{b_i},\ b_i \neq 0,\ i = 1, \ldots, m,\ j = 1, \ldots n.$ Ako je uslov

$$(\exists p, q)(1 \leq p, q \leq m)\ e_{ps} \geq e_{qs}, \quad s = 1, \ldots, n, \tag{4.3.2}$$

zadovoljen, tada q-to ograničenje iz (4.3.1) (respektivno (4.1.2)) može biti izostavljeno.

Međutim, ovo pravilo eliminacije nije univerzalno. Na primer, suvišno ograničenje (C_2) iz problema problema (4.2.2) ne zadovoljava uslov (4.3.2).

U sledećoj teoremi je primenjen poznat rezultat iz teorije igara, i definisano je nekoliko dodatnih pravila za eliminaciju, kao i direktan metod za rešavanje nekih tipova problema linearnog programiranja.

Teorema 4.3.1 [76] *Razmotrimo sledeći problem linearnog programiranja: odrediti maksimum ciljne funkcije*

$$z(x) = \sum_{j=1}^{n} c_j x_j = cx, \quad c_j > 0,\ j = 1, \ldots, n$$

u odnosu na ograničenja

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_j = r_i x \quad &\leq \quad b_i,\ b_i > 0, \quad i = 1, \ldots, m, \\ x_j \quad &\geq \quad 0, \quad j = 1, \ldots, n. \end{aligned} \tag{4.3.3}$$

Neka je $d_{ij} = \dfrac{a_{ij}}{b_i c_j}, \quad i = 1, \ldots, m,\ j = 1, \ldots, n.$ *Tada važe sledeća tvrđenja:*

(a) *U slučaju*

$$\max_{1 \leq i \leq m} \min_{1 \leq j \leq n} d_{ij} = \min_{1 \leq j \leq n} \max_{1 \leq i \leq m} d_{ij} = d_{kl}, \tag{4.3.4}$$

optimalno rešešenje problema (4.1.1)-(4.1.2) *je*

$$x_j = \begin{cases} \dfrac{b_k}{a_{kl}}, & j = l, \\ 0, & j \neq l. \end{cases} \tag{4.3.5}$$

(b) *Ako je*

$$\max_{1 \leq i \leq m} \min_{1 \leq j \leq n} d_{ij} \neq \min_{1 \leq j \leq n} \max_{1 \leq i \leq m} d_{ij}$$

i ukoliko postoje k, l *takvi da važi* $d_{ik} \geq d_{il},\ i = 1, \ldots, m,$ *tada to povlači da je* $x_k = 0$*, tako da možemo izostaviti* k*-tu kolonu.*

Tkođe, ako postoje k, l *takvi da važi* $d_{kj} \leq d_{lj},\ j = 1, \ldots, n,$ *tada možemo izostaviti* k*-tu vrstu.*

Dokaz. Neka je $y_j = c_j x_j,\ j = 1, \ldots, n$. Tada je problem (4.3.3) ekvivalentan problemu $z(y) = \sum_{j=1}^{n} y_j \to max$ sa ograničenjima

$$\sum_{j=1}^{n} d_{ij} y_j \leq 1, \quad i = 1, \ldots, m, \quad y_j \geq 0,\ \ j = 1, \ldots, n.$$

Primetimo da je ovaj problem linearnog programiranja ekvivalentan sa igrom dva igrača sa matricom plaćanja $D = ||d_{ij}||$ [50, 76]. Označimo optimalnu strategiju za drugog igrača vektorom $q = (q_1, \ldots, q_n)$.

(a) Ako je uslov (4.3.4) ispunjen, tada je optimalna strategija za drugog igrača čista strategija $q_l = 1$, $q_j = 0$, $j \neq l$. Kako je $c_j x_j = y_j = \dfrac{q_j}{v}$, gde je v vrednost igre, dobijamo [50] $x_j = 0$, $j \neq l$ i

$$x_l = \max_{1 \leq i \leq m} \left\{ \frac{a_{il}}{b_i} \right\} = \min_{1 \leq i \leq m} \left\{ \frac{b_i}{a_{il}} \right\} = \frac{b_k}{a_{kl}}.$$

(b) Ako postoje k, l tako da je $d_{ik} \geq d_{il}, i = 1, \ldots, m$, tada iz teorije igara sledi $q_k = 0$, što povlači $x_k = 0$ [50]. Dakle, možemo izostaviti k-tu kolonu.

Ako postoje k, l tako da je $d_{kj} \leq d_{lj}, j = 1, \ldots, n$, tada

$$\sum_{j=1}^{n} d_{lj} \xi_j \leq 1 \Rightarrow \sum_{j=1}^{n} d_{kj} \xi_j \leq 1, \ \text{ za proizvoljne } \xi_j > 0,\ \ j = 1, \ldots, n$$

tako da je k-to ograničenje suvišno. □

Uopštenja Teoreme 4.3.1 se mogu naći u radu [85].

Napomena 4.3.1 Ako su sva suvišna ograničenja eliminisana, tada je uslov $x_0 \in P$ ispunjen. Nažalost, predloženi algoritmi za eliminaciju suvišnih ograničenja ne garantuju kompletnu eliminaciju. Na primer, suvišno ograničenje (C_2) u problemu (4.2.2) ostaje posle svih eliminacija. U tom slučaju predlažemo da se bazično rešenje konstruisano primenom metode minimalnih uglova iskoristi kao početna tačka za primenu simpleks algoritma.

4.4 Algoritmi i implementcioni detalji

U skladu sa Teoremom 4.2.1 uvodimo dva algoritma, označena sa *Algoritam An* i *Algoritam Al*, za implementaciju metoda minimalnih uglova. Ovi algoritmi se mogu upotrebiti za maksimizaciju ciljne funkcije (4.1.1) pod ograničenjima (4.1.2), i moguće ih je primeniti u slučaju $l \geq n$.

Algoritam An

Korak 1. Eliminisati suvišna ograničenja, koristeći rezultate iz prethodnog odeljka i Gausovu eliminaciju.

Korak 2. Izračunati vrednosti $v_i = |c|\cos(c, r_i) = \dfrac{cr_i}{|r_i|}, \quad i = 1, \ldots, m.$

Korak 3. Odrediti n maksimalnih i pozitivnih vrednosti

$$v_{i_1} \geq \cdots \geq v_{i_n} > 0$$

iz skupa $\{v_1, \cdots v_m\}$.

Korak 4. Izračunati x_0 kao rešenje sistema jednačina (4.2.2).

Korak 5. Proveriti da li je x_0 bazično dopustivo rešenje ($x_1 \geq 0, \ldots, x_n \geq 0$), zbog primene simpleks metoda u sledećem koraku.

Korak 6. Ako je uslov iz prethodnog koraka ispunjen, primeniti algoritam simpleks metoda *SimMax* iz odeljka 1.2 za bazično dopustivo rešenje [84]). U suprotnom, primeniti pripremni algoritam simpleks metoda *PreSim Max* iz odeljka 1.2, koji generiše prvo bazično rešenje i završiti algoritmom za bazično dopustivo rešenje [84].

Neki softverski detalji implementacije *Algoritma An* su opisani u nastavku. Unutrašnji oblik problema postavljenog u (4.1.1) sadrži dva različita dela. Prvi deo je proizvoljna ciljna funkcija, određena sa odgovarajućim izrazom iz paketa MATHEMATICA, označena parametrom *objective*. Drugi deo je lista izabranih ograničenja koju nazivamo *constraints*. Prema unutrašnjoj formi linearnih programa, koji se zahtevaju u funkcijama *LinearProgramming*, *ConstrainedMin* i *ConstrainedMax* iz paketa MATHEMATICA [90], [92], izostavljamo listu promenljivih koje se koriste u ciljnoj funkciji i datim ograničenjima. Listu promenljivih izostavljamo zbog jednostavnije upotrebe programa. Tu listu je moguće rekonstruisati koristeći standardne funkcije *Variables* i *Union* (o tim funkcijama videti [91] i [92]). U tu svrhu, definišemo sledeću funkciju koja izdvaja listu promenljivih upotrebljenih u listi *lis_*.

```
SelectVariables[lis_] :=
  Module[{duz, l={}},
      duz = Length[lis];
      Do[l=Union[l, Variables[lis[[i]]]],{i,duz}];
      Return[l]
  ]
```

U glavnom programu možemo da pišemo

```
var=SelectVariables[Append[objective, constraints]].
```

Implementacija Koraka 2 i Koraka 3.

A. Matrica datih ograničenja, čiji su elementi jednaki a_{ij}, $i = 1, \ldots, m$, $j = 1, \ldots, n$, može biti formirana na sledeći način.

```
Formmatrix[constr_, var_]:=
  Module[{a={}, i,j,m,n, p, c=constr},
     m=Length[c];  n=Length[var];
     For[i=1, i<= m, i++,
         p={};
         For[j=1, j<= n, j++,
             AppendTo[p,Coefficient[c[[i,1]], var[[j]]]];
```

```
        ];
        AppendTo[a,p];
    ];
    Return[a];
]
```

B. Izbor l maksimalnih vrednosti $v_{i_q} > 0$, $q = 1, \ldots, l$ iz skupa V, definisanmog sa (4.2.1), može biti izvršen primenom funkcije `maximaln[mat_, ncf_, constr_]`. Formalni parametar $mat\_$ ove funkcije označava matricu datih ograničenja, i može biti konstruisan primenom funkcije $Formmatrix$. Parametar $ncf\_$ označava vektor $\{c_1, \ldots, c_n\}$, i može biti konstruisan na sledeći način:

```
For[i=1, i<= n, i++,
    AppendTo[nc, Coefficient[objective,var[[i]]]];
];
```

Konačno, parametar $constr\_$ označava listu datih ograničenja.

```
maximaln[mat_, ncf_, constr_]:=
  Module[{i,j,m,n, mt=mat,c=ncf, sp={},csp=constr, p},
     {m,n}=Dimensions[mt];
     For[i=1, i<= m, i++,
       AppendTo[sp, mt[[i]].c/Sqrt[mt[[i]].mt[[i]]] ]
     ];
     For[i=1, i<=m-1, i++,
         For[j=i+1, j<=m, j++,
            If[sp[[i]]<sp[[j]],
               p=sp[[i]]; sp[[i]]=sp[[j]]; sp[[j]]=p;
               p=csp[[i]]; csp[[i]]=csp[[j]]; csp[[j]]=p;
         ]  ]
     ];
     For[i=1; l=0, i<= m, i++,
        If[sp[[i]]>0, l++]
     ];
     Return[{Take[csp,l], Drop[csp,l]} ];
  ]
```

Implementacija Koraka 4 i Koraka 5.

A. Za svako $i = 1, \ldots, m$ transformišemo dato ograničenje oblika

$$\sum_{k=1}^{n} a_{ik}x_k + r_i \quad \begin{matrix} \leq \\ \geq \\ = \end{matrix} \quad \sum_{k=1}^{n} b_{ik}x_k + s_i$$

u oblik

$$\left\{ \sum_{k=1}^{n} a_{ik}x_k + r_i - \left(\sum_{k=1}^{n} b_{ik}x_k + s_i \right), \quad \begin{matrix} \leq \\ \geq \\ = \end{matrix} \right\}. \tag{4.4.1}$$

Nule na desnoj strani formule (4.4.1) se podrazumevaju. U tu svrhu, definišemo sledeću funkciju:

```
Transform[lis_] :=
  Module[{l={}, h, p},
     Do[h=Head[lis[[i]]];   p=lis[[i]]/.h->Subtract;  AppendTo[l, List[p,h]],
        {i,Length[lis]}
     ];
     Return[l]
  ]
```

Primenom izraza `lista =Transform[constraints]` lista datih ograničenja je transformisana u listu čiji elementi su oblika (4.4.1).

B. *Implementacija Koraka 4.*

B1. U sledećem kodu koristimo parametre $goal\_$ i $constr\_$, da bi predstavilii unutrašnji oblik postavljenog problema.

```
c=Transform[c]; var=SelectVariables[Append[c,g]];
n=Length[var];  m=Length[c];
For[i=1, i<= n, i++,
  AppendTo[nc, Coefficient[g,var[[i]]]];
];
mat=Formmatrix[c, var];  {l1,l2}=maximaln[mat,nc,c];
```

B2. Kostruisati i rešiti sistem (4.2.2). Sistem (4.2.2) se formira na sledeći način:

```
For[i=1, i<= n, i++,
   l11[[i]]=Equal[l1[[i,1]],0];
];
```

Rešiti sistem (4.2.2) i izračunati vrednost ciljne funkcije.

```
x=First[Solve[l11,var]];   x=x/.Rule->List;
For[i=1, i<= n, i++,
    g=g/.x[[i,1]]->x[[i,2]]
];
```

C. *Implementacija Koraka 5.* U ovom delu koda se proverava da li je konstruisano rešenje bazično dopustivo. Ovaj korak je takođe i priprema za simpleks metod.

```
test=True; i=1;
While[test && i<=n+m,
    If[x[[i,2]]<0, test=False, i++];
];
If[test,
   Print["Solution ",{x,g}, " is basic"],
   Print["Solution ", {x,g}, " is nonbasic"]
];
Return[{l1,l2,test}]
```

Implementacija Koraka 6.

Proizvoljno ograničenje zadato linearnom nejednakošću se može lako transformisati u odgovarajuće linearno ograničenje zadato jednakošću dodavanjem ili oduzimanjem pozitivne izravnavajuće promenljive. Taj cilj se postiže u sledećim koracima.

Korak S1. Transformišemo data ograničenja u oblik (4.4.1).

Korak S2. U sledećoj funkciji formalni parametar *lis* predstavlja ograničenje oblika (4.2.2) i *prom* je pozitivna izravnavajuća promenljiva. Rezultat je leva strana ekvivalentnih ograničenja zadatih jednakostima:

$$\sum_{k=1}^{n} a_{ik}x_k + r_i - \left(\sum_{k=1}^{n} b_{ik}x_k + s_i\right) \pm prom = 0.$$

```
AddVariable[lis_, prom_] :=
   Module[{},
      Switch[lis[[2]],
         LessEqual,     Return[Plus[lis[[1]],prom]],
         GreaterEqual,  Return[Substract[lis[[1]],prom]],
         Equal,         Return[lis[[1]]]
   ]  ]
```

Korak S3. Za svako $i = 1, \ldots, m$ u glavnoj funkciji sada transformišemo i-to dato ograničenje dodavanjem izravnavajuće promenljive $\$[i]$.

```
  ct=Transform[constraints];
  Do[ith=AddVariable[ct[[i]], $[i] ];
  AppendTo[system,Equal[ith,0]], {i,Length[constraints]} ];
```

Rešenje $x[[i,2]]$, $i = 1, \ldots, n$, dobijeno kao izlaz iz *Koraka 5*, može biti pripremljeno kao početna tačka $ksi[[i]]$, $i = 1, \ldots, m+n$ za simpleks metod na sledeći način:

```
  ksi=Table[i,{n+m}];
  For[i=1, i<= n, i++, ksi[[i]]=x[[i,2]] ];
  For[i=n+1,i<=2*n, i++, ksi[[i]]=0];
  For[i=1, i<= m-n, i++,
      ksi[[2*n+i]]=l2[[i,1]]
     ];
  For[i=1, i<= m-n, i++,
      For[j=1, j<= n, j++,
          ksi[[2*n+i]]=ksi[[2*n+i]]/.x[[j,1]]->x[[j,2]]
      ]
  ];
  For[i=2*n+1, i<= n+m, i++, ksi[[i]]=-ksi[[i]]];
```

Kada broj pozitivnih vrednosti $v_{i_1}, \ldots, v_{i_l}$, definisanih sa (4.2.1), zadovoljava $0 < l < n$, predlažemo sledeći algoritam:

Algoritam Al

Korak 1, *Korak 2.* Identični sa *Korakom 1* i *Korakom 2* of *Algoritma An*, respektivno.

Korak 3. Odrediti pozitivne vrednosti $v_{i_1}, \ldots v_{i_l}$ iz skupa $\{v_1, \ldots, v_m\}$.

Korak 4. Eliminisati proizvoljnih l nepoznatih, koristeći sistem (4.2.3). Pretpostavimo da je sistem (4.2.3) rešen u odnosu na promenljive $x_{i_1}, \ldots, x_{i_l}$. Neka je

bazično rešenje x_0 generisano izjednačavanjem ostali $n-l$ promenljivih sa nulom, i izračunavanjem promenljvih $x_{i_1}, \ldots, x_{i_l}$.

Korak 5. Primeniti opšti simpleks metod, koristeći bazično rešenje x_0.

Implementacija *Koraka 1*, *Koraka 2* i *Koraka 3* je opisana ranije. Sada opisujemo implementaciju *Koraka 4* i *Koraka 5*. Implementacija *Koraka 4* je opisana u nastavku. Sa *lvar* označavamo skup bazičnih promenljivih $x_{i_1}, \ldots, x_{i_l}$, dok *rvar* označava skup komplementarnih promenljivih.

```
x=First[Solve[l11,var]];
lvar={};
For[i=1, i<= l, i++,
    AppendTo[lvar, x[[i,1]] ]
];
rvar=Complement[var,lvar];
For[i=1, i<= n-l, i++,
    AppendTo[l11, Equal[rvar[[i]],0]]
];
x=First[Solve[l11,var]];
```

Rešenje $x[[i,2]]$, $i = 1, \ldots, n$, se može transformisati u početnu tačku $ksi[[i]]$, $i = 1, \ldots, m+n$ za simpleks metod primenom sledećeg kôda:

```
For[i=1, i<=n, i++,
    ksi[[i]]=x[[i,2]]
];
For[i=n+1, i<= n+l, i++, ksi[[i]]=0 ];
For[i=n+l+1, i<=n+m, i++, ksi[[i]]=l2[[i-n-l,1]] ];
For[i=n+l+1, i<=n+m, i++,
    For[j=1, j<=n, j++,
        ksi[[i]]=ksi[[i]]/.x[[j,1]]->x[[j,2]]
    ]
];
For[i=n+l+1, i<= n+m, i++, ksi[[i]]=-ksi[[i]]    ]
```

Konačno, prema Teoremi 4.3.1 predlažemo sledeći algoritam. Uslovi za primenu Teoreme 4.3.1 su pretpostavljeni.

Algoritam Ag

Korak 1. Izračunati

$$d_{ij} = \frac{a_{ij}}{b_i c_j}, \quad i = 1, \ldots, m, \ j = 1, \ldots, n.$$

Korak 2. Ako je uslov (4.3.4) ispunjen, iskoristiti vektor $(x_1, \ldots, x_n)$, definisan sa (4.3.5), za optimalnu tačku. Izlaz je vektor $(x_1, \ldots, x_n)$ i algoritam staje.

Korak 3. Ako uslov (4.3.4) nije zadovoljen, razmotriti sledeće slučajeve:

Slučaj 1. Ako postoje k, l tako da je $d_{ik} \geq d_{il}$, $i = 1, \ldots, m$, izbaciti k-tu kolonu iz početnog problema.

Slučaj 2. Ako postoje k, l tako da je $d_{kj} \leq d_{lj}$, $j = 1, \ldots, n$, izbaciti k-tu vrstu.

Slučaj 3. U suprotnom, eliminisati suvišna ograničenja primenom (4.3.1).

Posle svakog od ovih slučajeva, primeniti opšti algoritam simpleks metoda.

4.5 Numerički primeri

Geometrijske osobine Teoreme 4.2.1 *Algoritma An* su ilustrovane u sledeća četiri primera.

Primer 4.5.1 U ovom primeru maksimiziramo ciljnu funkciju $321x1+287x2$ pod sledećim skupom ograničenja:

$1.02x1+5x2 \leq 5103$, $1.032x1+4x2 \leq 4130.2$, $0.314x1+x2 \leq 1048.2$, $1.122x1+3x2 \leq 3202.8$,
$0.872x1+2x2 \leq 2182.2$, $0.504x1+x2 \leq 1119.8$, $0.577x1+x2 \leq 1154.7$,
$1.974x1+3x2 \leq 3592.2$, $3x1+4x2 \leq 5000$, $0.855x1+x2 \leq 1315.9$,
$0.98x1+x2 \leq 1400.2$, $4x1+3x2 \leq 5000$, $3.036x1+2x2 \leq 3636.6$,
$3.464x1+2x2 \leq 4000$, $1.134x1+x2 \leq 1511.8$, $1.984x1+x2 \leq 2222.2$,
$2.291x1+x2 \leq 2500$, $2.676x1+x2 \leq 2856.9$, $3.873x1+x2 \leq 4000$

Vrednosti v_i su, respektivno, sledeće:

345.3707765, 358.0919724, 369.9832633, 381.2617368, 391.3741373,
400.7613457, 409.0139395, 416.1997108, 422.2, 426.739932,
429.6561598, 429., 425.9470426, 421.495273, 430.5825196,
415.82367, 409.0075608, 401.1548539, 382.5567078

Transformišući nejednakosti koje odgovaraju maksimalnim v-ovima u odgovarajuće jednakosti, dobijamo sledeći linearni sistem:

$$\{-1511.8 + 1.134x1 + x2 = 0,\ \ -1400.2 + 0.98x1 + x2 = 0\}$$

Rešenje $x_0 = (724.675, 690.018)$ tog sistema je bazično dopustivo. Simpleks metod samo potvrđuje da je ta tačka ekstremna. Prema tome, u ovom slučaju je $x_0 = x_P$, i minimalni uglovi su zahvaćeni u ekstremnoj tački. Maksimalna vrednost ciljne funkcije je $z_{\max} = 430655.997402597545$.

Sa druge strane, originalni simpleks metod daje isto rešenje u 9 iteracija. Takođe, kanoničan oblik datog problema sadrži 21 promenljivu:

$$\begin{aligned}
-5103 + 1.02x1 + 5x2 + \$[1] &= 0, & -4130.2 + 1.032x1 + 4x2 + \$[2] &= 0,\\
-1048.2 + 0.314x1 + x2 + \$[3] &= 0, & -3202.8 + 1.122x1 + 3x2 + \$[4] &= 0,\\
-2182.2 + 0.872x1 + 2x2 + \$[5] &= 0, & -1119.8 + 0.504x1 + x2 + \$[6] &= 0,\\
-1154.7 + 0.577x1 + x2 + \$[7] &= 0, & -3592.2 + 1.974x1 + 3x2 + \$[8] &= 0,\\
4 - 5000 + 3x1 + 4x2 + \$[9] &= 0, & -1315.9 + 0.855x1 + x2 + \$[10] &= 0,\\
-1400.2 + 0.98x1 + x2 + \$[11] &= 0, & -5000 + 4x1 + 3x2 + \$[12] &= 0,\\
-3636.6 + 3.036x1 + 2x2 + \$[13] &= 0, & -4000 + 3.464x1 + 2x2 + \$[14] &= 0,\\
-1511.8 + 1.134x1 + x2 + \$[15] &= 0, & -2222.2 + 1.984x1 + x2 + \$[16] &= 0,\\
-2500 + 2.291x1 + x2 + \$[17] &= 0, & -2856.9 + 2.676x1 + x2 + \$[18] &= 0,\\
-4000 + 3.873x1 + x2 + \$[19] &= 0.
\end{aligned}$$

Napomenimo da se primenom programa PCx dobija isto rešenje u 6 iteracija.

Primer 4.5.2 U ovom primeru, *Algoritam An* daje bazično dopustivo rešenje koje pripada istoj hiperravni kao i ekstremna tačka. Posle toga se ekstremna tačka određuje u samo jednom koraku simpleks metoda. Razmotrimo sledeći maksimizacioni problem: maksimizirati $z(x) = x1 + x2$, u odnosu na ograničenja

$$\{x1/1000 + x2/9 \leq 1,\ \ x1/5 + x2/70 \leq 1,\ \ x1/4.9 + x2/111 \leq 1\}.$$

Primenom *Algoritma An*, dobijamo sledeće bazično dopustivo rešenje:

$$x_0 = (4.73821, 3.66509).$$

Međutim, tačka x_0 nije ekstremna. Primenom simpleks algoritma za bazično dopustivo rešenje, uzimajući za početnu tačku x_0, dobijamo u jednom dodatnom koraku simpleks metoda sledeću ekstremnu tačku

$$x_0 = (4.35995, 8.96076)$$

i maksimalnu vrednost 13.32071 ciljne funkcije.

Primetimo da tačke x_0 i x_P pripadaju hiperravni $x1/5 + x2/70 = 1$.

Primenom simpleks metoda, dobijamo identičan rezultat u tri koraka.

Primer 4.5.3 U ovom primeru, *Algoritam An* takođe daje bazično dopustivo rešenje koje pripada istoj hiperravni kao i ekstremna tačka. Međutim, ekstremna tačka se ne dobija u jednom koraku simpleks metoda. Maksimizirati $11x1 + 18x2 + 29x3 - 0.8x4$ u odnosu na uslove

$$x1 + 4x2 + 6x3 + 2x4 \leq 60,\ \ 4x1 + x2 + 5x3 + 9x4 \leq 91$$
$$6x1 + 5x2 + 8x3 + 5x4 \leq 102,\ \ 2x1 + 5x2 + 7x3 + 7x4 \leq 110,$$
$$-x1/2 + x2/5 - x3/3 - x4/2 \leq 1$$

Primenom *Algoritma An*, dobijamo bazično dopustivo rešenje

$$x_0 = \left(\frac{233}{157},\ \frac{2705}{314},\ \frac{227}{157},\ \frac{2415}{314}\right) \tag{4.5.1}$$

i odgovarajuću vrednost $\frac{67301}{157}$ ciljne funkcije. Kasnije, primenom simpleks metoda, startujući iz tačke (4.5.1), dobijamo maksimalnu vrednost $\frac{4467}{14}$ u 2 dodatne iteracije. Ukupno procesorsko vreme je 1.26 sekundi. Ekstremna tačka je

$$x_P = \left(\frac{33}{7},\ 0,\ \frac{129}{14},\ 0\right). \tag{4.5.2}$$

Primetimo da tačke x_0 i x_P, izražene sa (4.5.1) i (4.5.2), pripadaju istoj hiperravni $x1 + 4x2 + 6x3 + 2x4 = 60$.

Original simpleks metod daje rešenje u 7 iteracija i 1.59 sekundi.

Primer 4.5.4 U ovom primeru pokazujemo da je *Algoritam An* moguće primeniti i na linearne programe koji sadrže negativne koeficijente u ograničenjima. Maksimizirati $11x1 + 18x2 + 29x3 + 28x4$ pod uslovima

$$x1 + 4x2 + 6x3 + 2x4 \leq 60,\ \ 4x1 + x2 + 5x3 + 9x4 \leq 91,$$
$$6x1 + 5x2 + 8x3 + 5x4 \leq 102,\ \ 2x1 + 5x2 + 7x3 + 7x4 \leq 110,$$
$$-x1/2 + x2/5 - x3/3 - x4/2 \leq 1,\ \ x1/7 - x2/3 - x3/2.3 + x4/3.4 \leq 1$$

Primenom *Algoritma An*, dobijamo maksimalnu vrednost 428.668789808917161 i ekstremnu tačku

$$x_0 = x_P = (1.48408,\ 8.61465,\ 1.44586,\ 7.69108).$$

Napomenimo da simpleks metod daje ovo rešenje u 5 iteracija.

U sledećem primeru ilustrujemo primenu *Algoritma Al*.

Primer 4.5.5 Maksimizirati $z = x4$ pod ograničenjima

$$1/4 \leq x1,\ \ x1 \leq 1,\ \ 1/4 * x1 \leq x2,\ \ x2 \leq 1 - 1/4 * x1,$$
$$1/4 * x2 \leq x3,\ \ x3 \leq 1 - 1/4 * x2,\ \ 1/4 * x3 \leq x4,\ \ x4 \leq 1 - 1/4 * x3$$

Ovaj problem je poseban slučaj ($n = 4$) poznatog primera iz [38], koji demonstrira da simpleks algoritam nije polinomijalan. Simpleks metod konvergira u 7 iteracija i daje maksimalnu vrednost 255/256 i ekstremnu tačku

$$(1/4,\ 1/16,\ 1/64,\ 255/256). \tag{4.5.3}$$

Primenom *Algoritma Al*, dobijamo sledeće skalarne proizvode, respektivno:

$$\{0,\ 0,\ 0,\ 0,\ 0,\ 0,\ -0.9701425001,\ 0.9701425001\}.$$

Postoji samo jedan pozitivan skalarni proizvod, i *Algoritam An* nije u mogućnosti da da rešenje. Tada je primenjen *Algoritam Al*. Poslednji skalarni proizvod je pozitivan i pouzdano možemo upotrebiti samo jednačinu $x4 = 1 - 1/4 * x3$, koja nastaje iz poslednje nejednakosti. Koristeći tu jednačinu zajedno sa $x1 = x2 = x4 = 0$, dobijamo bazično rešenje

$$x_0 = (0,\ 0,\ 4,\ 0).$$

Sada, primenom simpleks metoda, startujući iz tačke x_0, dobijamo maksimalnu vrednost 255/256 i ekstremnu tačku (4.5.3) u 5 iteracija.

Sada ilustrujemo eliminaciju suvišnih ograničenja pod uslovima (4.3.2).

Primer 4.5.6 Razmotrimo maksimizaciju ciljne funkcije $2x1 + 3x2 + x3 + 5x4$, uz ograničenja

$$x1/10 + x2/15 + x3/12 + x4/15 <= 1,\ x1/12 + x2/16 + x3/14 + x4/20 <= 1,$$
$$x1/12 + x2/10 + x3/9 + x4/11 <= 1,\ x1/22 + x2/15 + x3/12 + x4/30 <= 1,$$
$$x1/8 + x2/6 + x3/5 + x4/9 <= 1,\ x1/3 - x2/4 + x3/2 - x4/5 >= 1.$$

Koristeći klasičan simpleks metod, dobijamo maksimalnu vrednost $\frac{1797}{67}$ i ekstremnu tačku

$$\left(\frac{336}{67},\ 0,\ 0,\ \frac{225}{67}\right)$$

u 9 iteracija. Kako je uslov (4.3.2) ispunjen za $p = 1$, $q = 2$, druga nejednačina je suvišna. Izostavljajući drugo ograničenje, simpleks metod daje identičan rezultat u 8 iteracija. Dakle, eliminacija suvišnih ograničenja ne samo da smanjuje dimenziju problema, već i u mnogim slučajevima redukuje broj iterativnih koraka.

U ovom primeru zbog postojanja suvišnog ograničenja ne važi uslov $x_0 \in P$, i rezultati Teoreme 4.2.1 nisu pouzdani u tom slučaju.

Primer 4.5.7 Razmotrimo problem (4.2.2). Kako je u Primeru 4.2.2 pokazano, metod minimalnih uglova daje tačku $x_0 = (1/2, 3/2) \notin P$. Međutim, tačka x_0 nije bazično dopustiva, i primenjujemo simpleks metod za nedopustivu startnu tačku. Tada simpleks metod konvergira u 3 iteracije. Originalni simpleks metod takođe daje rešenje u 3 iteracije. Dakle, računanje minimalnih uglova ne poboljšava konvergenciju simpleks metoda što znači da je uslov $x_0 \in P$ potreban.

U sledeća dva primera ilustrujemo primenu Teoreme 4.3.1.

Primer 4.5.8 Odrediti maksimum ciljne funkcije $x1 + x2 + 2x3 + 5x4 + 2x5$ pod uslovima:

$$30.4x1 + 12x2 + 16.5x3 + 30x4 + 16.3x5 \leq 2,$$
$$60x1 - 15.1x2 - 20x3 - 25.6x4 + 40x5 \leq 5,$$
$$48.7x3 + 24x2 + 80.2x3 + 40x4 + 96.5x5 \leq 8,$$
$$18.7x1 + 15x2 + 18.7x3 + 30x4 + 18.7x5 \leq 3.$$

Simpleks metod daje optimalnu vrednost 0.333333333333333303 i optimalnu tačku

$$(0,\ 0,\ 0,\ 0.0666667,\ 0)$$

u 9 iteracija. Primenom dela (a) Teoreme 4.3.1 dobijamo direktno to rešenje.

Primer 4.5.9 Maksimizirati $2x1 + 0.5x2 + 4x3 + x4 + 3x5 + 5x6$ pod uslovima:

$$20x1 + 6x2 + 32.3x3 - 6x4 + 24x5 + 60.5x6 \leq 2,$$
$$32x1 - 6x2 - 32x3 - 4x4 + 48.3x5 + 160.6x6 \leq 4,$$
$$20x1 + 7.5x2 + 100x3 + 5x4 + 90.7x5 + 50x6 \leq 5,$$
$$24x1 + 15x2 + 72.7x3 + 12x4 + 54x5 + 120.6x6 \leq 6.$$

Za 2.47 sekundi i 4 iteracije simpleks metoda dobijamo maksimalnu vrednost $\frac{1}{2}$ i sledeću ekstremnu tačku:

$$\left(\frac{3}{20},\ 0,\ 0,\ \frac{1}{5},\ 0,\ 0\right).$$

Posle eliminacije kolona 5 i 6 (na osnovu Teoreme 4.3.1), dobijamo isto rešenje u 1.32 sekundi i takođe u 4 iteracije.

Napomena 4.5.1 *Za mali broj ograničenja broj iteracija se smanjuje u odnosu na simpleks algoritam, ali može da se desi da je procesorsko vreme povećano zbog dodatnih izračunavanja pre primene simpleks metoda.*

U sledećem primeru dajemo jedan problem većih dimenzija sa 63 ograničenja i 32 promenljive.

Primer 4.5.10 Maksimizirati ciljnu funkciju

$$\begin{aligned}&5x11 + 4x12 + 2x13 + 3x14 + 3x15 + 9x16 + 2x17 + x18 + 0.5x19 + x20 + x21 + x22\\&+ 5x23 + x24 + 3x25 + x26 + x27 + x28 + x29 + 8x30 + x31 + 3x32 + x33\\&+ 5x34 + x35 + 7x36 + x37 + 0.5x38 + x39 + x40 + x41 + x42\end{aligned}$$

u odnosu na ograničenja

$10x13+8x17+0.1x24+0.5x37 \leq 220$, $5x11 \leq 9$, $6x17+0.1x21-0.2x29+0.1x31+0.7x37 \leq 90$, $4x12 \leq 6$ $-0.4x13 + 2x21 + 0.2x24 + 0.1x27 \leq 20$, $3x14 \leq 20$, $2x24 + 0.3x29 + 0.2x35 \leq 40$, $3x15 \leq 40$, $9x16 \leq 16$, $0.4x13 + 4x27 - 0.2x31 + 0.5x42 \leq 160$, $x18 \leq 120$, $-0.4x17 + 3x29 + 0.6x35 - 0.1x37 \leq 120$, $0.5x19 \leq 320$, $0.6x17 + 4x31 \leq 320$, $x20 \leq 200$, $-x21 - 2x22 \leq -1$, $-0.4x21 - 0.1x27 + 0.6x29 + 5x35 \leq 200$, $5x23 \leq 7$, $-0.1x24-0.3x31+6x37+0.1x42 \leq 120$, $x26 \leq 2$, $-0.2x13-0.2x21+0.6x27+4x42 \leq 40$, $3x25 \leq 8$, $x40 \leq 42$, $-2x22 - 15x23 \leq -2$, $-15x23 - 4x24 \leq -0.5$, $x39 \leq 14$, $-4x24 - 15x25 \leq -3$, $-15x25 - 6x26 \leq -2$, $0.5x38 \leq 70$, $-6x26 - 7x27 \leq -2$, $-7x27 - 8x28 \leq -5$, $7x36 \leq 11$, $-8x28 - 9x29 \leq -3$, $-9x29 - 8x30 \leq -4$, $5x34 \leq 6$, $-8x30 - x31 \leq -0.2$, $-x31 - 6x32 \leq -1$, $x33 \leq 30$, $-6x32 - 3x33 \leq -2$, $-3x33 - 20x34 \leq -5$, $3x32 \leq 20$, $-20x34 - 5x35 \leq -3$, $-5x35 - 42x36 \leq -4$, $8x30 \leq 16$, $-42x36 - 7x37 \leq -9$, $-7x37 - 4x38 \leq -7$, $x28 \leq 49$, $-4x38 - 9x39 \leq -5$, $-9x39 - x40 \leq -1$, $x22 \leq 40$, $-x40 - x41 \leq -0.1$, $-x41 - 2x42 \leq 2$, $-10x11 - 12x12 \leq -0.2$, $-16x12 - 10x13 \leq -2$, $x41 \leq 120$, $-6x13 - 3x14 \leq -0.3$, $-6x14 - 12x15 \leq -2$, $-24x15 - 54x16 \leq -3$, $-54x16 - 14x17 \leq -2$, $14x17 - 8x18 \leq -5$, $-8x18 - 4.5x19 \leq -3$, $-4.5x19 - x20 \leq -1$, $-x20 - x21 \leq -0.5$.

PCx rešava ovaj problem u 7 iteracija. Koristeći *Algoritam An* dobijamo maksimalnu vrednost 1657.32225559107115 i ekstremnu tačku

$$\begin{aligned}&(1.8,\ 1.5,\ 20.6827,\ 6.66667,\ 13.3333,\ 17.7778,\ 0.,\ 120.,\ 640.,\ 200.,\ 10.9497,\\&40.,\ 1.4,\ 11.0254,\ 2.66667,\ 2.,\ 41.2576,\ 49.,\ 33.2613,\ 2.,\ 80.,\ 6.66667,\\&30.,\ 1.2,\ 37.7098,\ 1.57143,\ 24.0974,\ 140.,\ 14.,\ 42.,\ 120.,\ 5.392980.)\end{aligned}$$

u samo dve iteracije.

Sledeći primer je iz [4]. Programi PCx i HOPDM nisu u mogućnosti da reše dati problem usled numeričke nestabilnosti. Sa druge strane, *Algoritam Al*, kao i originalni simpleks metod rešavaju ovaj problem posle samo 3 iteracije!

Primer 4.5.11 Maksimizirati

$$\begin{aligned}&- 8919x1 - 5981x2 - 9892x6 - 3x10 - 9800x11 - 9989x14\\&- 993x15 + 9978x17 - 9687x18 - 9993x19 + 9800x20\end{aligned}$$

pod uslovima

$\{8919x1 - 4788x2 - 2x3 - 9733x4 - 3993x5 - x7 - x8 - 9002x9 - 9789x10 + x11 - 3x12 - 9971x13 + x14 - x15 - x16 - 9978x17 + 9687x18 + 9993x19 - x20 = -9791$,

$-8919x1 - 4790x2 + 2x3 - 9733x4 - 3993x5 - 2x6 + x7 + x8 - 9002x9 + 9789x10 + x11 - 3x12 - 9971x13 + 4902x14 - x15 + x16 + 9978x17 - 9687x18 - 9993x19 - x20 = 9789$,

$8919x1 - x2 - 2x3 - 2x6 - x7 - x8 + 9789x10 - x11 + 4901x14 - x16 - 9978x17 + 9687x18 + 9993x19 + x20 = -9790$, $-4788x2 - 2x3 - x7 - x8 - 9789x10 + x14 - x15 - x16 = -9790$,

$8919x1 - 4789x2 + 2x3 + 9733x4 + 3993x5 + x7 + x8 + 9002x9 + 3x12 + 9971x13 + x14 - x15 + x16 - 9978x17 + 9687x18 + 9993x19 = 9791,$

$-8919x1 + 4789x2 - 2x3 + 9733x4 + 3993x5 + 2x6 - x7 - x8 + 9002x9 + x11 + 3x12 + 9971x13 - 4902x14 + x15 - x16 + 9978x17 - 9687x18 - 9993x19 - x20 = -9789,$

$4788x2 + 2x3 - 9733x4 - 3993x5 - 2x6 + x7 + x8 - 9002x9 + 9789x10 + x11 - 3x12 - 9971x13 + 4900x14 + x15 + x16 - x20 = 9789,$

$-x1 \leq -9872, \quad -x16 \leq -8790\}.$

U 3 iteracije dobijamo maksimalnu vrednost 0, i sledeću ekstremnu tačku:

$$(9872,\ 0,\ 500,\ 0,\ 0,\ 0\,0,\ 0,\ 0,\ 0,\ 0,\ 0,1/3,\ 0,\ 0,\ 0,8790,\ \frac{14674728}{1663},\ 0,\ 0,\ 0).$$

Dalje, PCx nije u mogućnosti da reši ovaj problem i staje posle 9 iteracija sa neodređenim statusom:

```
Problem 'KBAPAH ' terminated with UNKNOWN status
9 iterations
Terminated with status UNKNOWN (code 3)
Solution at iteration 8:
Primal Objective = -3.23998779e+003
Dual Objective = -4.40527272e+002
```

Slično, HOPDM staje posle 6 iteracija sa podoptimalnim statusom, nudeći maksimalnu vrednost $-7.1e + 1$.

U radu [4] je data familija degenerisanih problema sa ekstremnom vrednošću ciljne funkcije koja je jednaka nuli. Malom modifikacijom dobija se sledeći primer kod koga je ekstremna vrednost različita od nule. Tako dobijeni problem takođe nije rešiv primenom programa PCx.

Primer 4.5.12 U ovom primeru maksimiziramo ciljnu funkciju kao u Primeru 4.5.11. Ograničenja su kao u Primeru 4.5.11, osim prvog i sedmog ograničenja, koja su zamnenjena sa sledećim ograničenjima, respektivno:

$8919x1 - 4788x2 - 2x3 - 9733x4 - 3993x5 - x7 - x8 - 9002x9 - 9789x10 + x11 - 3x12 - 9971x13 + x14 - x15 - x16 - 9978x17 + 9687x18 + 9993x19 - x20 \leq -9791,$

$4788x2 + 2x3 - 9733x4 - 3993x5 - 2x6 + x7 + x8 - 9002x9 + 9789x10 + x11 - 3x12 - 9971x13 + 4900x14 + x15 + x16 - x20 = 9782.$

U 7 iteracija dobijamo maksimalnu vrednost -24196384, i ekstremnu tačku

$$(9872,\ 0,\ \frac{1001}{2},\ 0,\ 0,\ 2450,\ 0,\ 0,\ 0,\ 0,\ 0,\ 4/3,\ 0,\ 1,\ 0,\ 8790,\ \frac{88048373}{9978},\ 0,\ 0,\ 5).$$

Originalni simpleks metod rešava ovaj problem u 8 iteracija. Kao što smo napomenuli, PCx nije u mogućnosti da reši ovaj problem:

```
Problem 'NIS ' terminated with UNKNOWN status
10 iterations
Terminated with status UNKNOWN (code 3)
Solution at iteration 10:
Primal Objective = -7.80713398e+004
Dual Objective = -5.46606497e+004
```

U ovom odeljku smo razmotrili nekoliko važnih problema u linearnom programiranju i primeni simpleks metoda: konstrukciju pogodne početne tačke, izbor pivot elementa i eliminaciju suvišnih ograničenja. Dalje, uveli smo takozvani metod minimalnih uglova. Tom heuristikom biramo $l \leq n$ oganičenja, tako da hiperravan

generisana tim ograničenjima zahvata l minimalnih uglova iz intervala $\left[0, \frac{\pi}{2}\right)$ sa gradijent vektorom ciljne funkcije. Izabrana ograničenja, koja su data nejednakostima, transformišemo u odgovarajuće jednakosti. U slučaju kada je $l \geq n$, biramo n minimalnih uglova i dobijamo sistem od n linearnih jednačina sa n promenljivih. Rešenje tog linearnog sistema se koristi ako bazično rešenje za simpleks metod. Ako je broj l uglova iz intervala $\left[0, \frac{\pi}{2}\right)$ manji od n, eliminišemo $l < n$ nepoznatih i koristimo hiperravni koje odgovaraju pozitivnim vrednostima iz skupa V. Tada početno bazično rešenje za simpleks metod može biti generisano izjednačavanjem ostalih $n - l$ promenljivih sa nulom.

Za uspešnu primenu metoda minimalnih uglova, potrebno je da problem (4.1.1)-(4.1.2) bude postavljen bez suvišnih ograničenja. U nekim slučajevima, metod je primenljiv i na probleme sa suvišnim ograičenjima.

Međutim, problem otkrivanja suvišnih ograničenja je veoma kompleksan. Uveli smo nekoliko pravila eliminacije koja se zasnivaju na geometrijskim osobinama suvišnih ograničenja. Takođe su data eliminaciona pravila koja se zasnivaju na teoriji igara. U nekim slučajevima primena teorije igara generiše direktno rešenje problema.

Nažalost, svi predloženi algoritmi za eliminaciju suvišnih ograničenja ne garantuju kompletnu eliminaciju. U tom slučaju, metod minimalnih uglova se može primeniti za gnerisanje startne tačke za simpleks metod. Ako je problem postavljen bez suvišnih ograničenja, u mnogim slučajevima dobijamo značajno ubrzanje simpleks procedure.

Implementacija metoda minimalnih uglova je razvijena u programskom paketu MATHEMATICA. Efikasnost primenjenih metoda je upoređena sa standardnim simpleks algoritmom i metodima unutrašnje tačke. Generalno, PCx i HOPDM su brži od metoda minimalnih uglova. Na primer, metod minimalnih uglova rešava test problem $Afiro$ iz $Netlib$ biblioteke problema u 15 iteracija i PCx rešava isti problem u 8 iteracija. Međutim, postoji familija degenerisanih problema linearnog programiranja kod koje se javljaju numerički problemi kod primene metoda unutrašnje tačke, kao što su PCx i HOPDM. Ta familija je konstruisana u [4] i [42]. Ti problemi su rešivi metodom minimalnih uglova. Sa druge strane, postoje problemi koji ne mogu biti rešeni metodom minimalnih uglova. Na primer, test problem *Blend* sadrži loše uslovljenu matricu što prouzrokuje značajne numeričke greške, tako da metod minimalnih uglova ne može da ga reši. Iz tih razloga se u narednoj glavi razmatra implementacija i modifikacija primal-dual metoda sa ciljem da se poveća tačnost i numerička stabilnost iterativnog procesa.

4.6 Direktni heuristični algoritam sa uopštenim inverzima

U ovom delu dajemo algoritam iz rada [67]. Ovaj heuristički metod vrlo često daje ili optimalno, ili rešenje iz koga se, primenom simpleks metoda dobija optimalno rešenje primenom malog broja iteracija. Osnovna ideja ovog metoda je prikazivanje opšteg rešenja linearnog sistema $Ax = b$ pomoću *pseudoinverza* $A^{\dagger}$

(Moore-Penroseovog uopštenog inverza) matrice A. Više o Moore-Penroseovom inverzu kao i ostalim uopštenim (generalisanim) inverzima operatora i matrica može se naći u monografijama [6, 87]. Napomenimo samo da je matrica $A^\dagger$ jedinstveno rešenje sledećeg sistema matričnih jednačina:

$$AXA = A, \quad XAX = X, \quad (AX)^* = AX, \quad (XA)^* = XA$$

Na sličan način se definišu i ostali uopšteni inverzi matrice A. Postoji više metoda za izračunavanje Moore-Penroseovog i ostalih generalisanih inverza matrice. Metod koji je u literaturi poznat kao Leverrier-Faddev ili Souriau-Frame metod koristi karakteristični polinom matrice A. Ovaj metod je prevashodno namenjen za simboličko izračunavanje uopštenih inverza. U našim radovima [71, 62, 61] pokazana je modifikacija metoda Leverrier-Faddev ukoliko je matrica A polinomijalna matrica. Pomenimo još i metod pregradjivanja u kome se Moore-Penroseov inverz računa primenom odgovarajućih rekurentnih formula. Modifikacije ovog metoda za polinomijalne matrice jedne ili više promenljivih prikazane su u našim radovima [60, 59].

Posmatramo problem linearnog programiranja u standardnom obliku:

$$\begin{array}{ll} \min & c^T x, \\ \text{p.o.} & Ax = b, \\ & x \geq 0. \end{array} \tag{4.6.1}$$

Ulazni podaci algoritma su A, b, c dok su izlazni podaci vektor x, vrednost ciljne funkcije z i komentari na osnovu testa optimalnosti.

Algoritam DHALP.

Korak 1. Učitati m, n, $A = [a_{ij}]$, b, c.

Korak 2. Izračunati $d = A^\dagger b$, gde je $d = [d_i]$ vektor dimezije $n \times 1$ a $A^\dagger$ je Moore-Penrose inverz (ili p-inverz) i neka je $e = Ad$.

Ako je $e \neq b$, izlaz je "Problem je nedopustiv" i algoritam staje.

Korak 3. Izračunati

$$\begin{gathered} H = A^\dagger A, \\ c' = (I - H)c, \\ s_k = \min\left\{ \tfrac{d_i}{c'_i} \,|c'_i > 0 \right\}, \\ x = d - c' s_k, \end{gathered}$$

gde je $c' = [c'_i]$ $n \times 1$ vektor, I je $n \times n$ jedinična matrica, H je $n \times n$ matrica. Vektor pravca $c' s_k$ pokušava da vektor x iz skupa rešenja jednačine $Ax = b$ pomeri u skup dopustivih rešenja definisan sa $Ax = b$, $x \geq 0$ ukoliko on već nije u njemu. Rešenje će biti ekstremna tačka skupa dopustivih rešenja.

Korak 4. Ukloniti onu nepoznatu x_i koja se izjednačava sa nulom, ukloniti i-tu kolonu matrice A i odgovarajuću koordinatu c_i iz vektora c ciljne funkcije. Izračunati $d = A^\dagger b$.

Napomena 4.6.1 *Sledeća matrica $A^\dagger$ u Koraku 4 je izračunata iz prethodne $A^\dagger$. To izračunavanje zahteva $O(mn)$ operacija. Moguće je izračunati A^+ iz A, ali to zahteva $O(mn^2)$ operacija. Ako se dve ili više koordinata x_i izjednačavaju sa nulom treba primeniti algoritam izbacujući jedno x_i i izračunati odgovarajući vektor x. Onaj vektor x koji daje minimum ciljne funkcije će biti traženo rešenje (izlaz) algoritma. Ne postoji odgovarajući kriterijum za odluku koje od dve (ili više) koordinata x_i koje se izjednačavaju sa nulom izbaciti iz baze. Međutim, takva situacija nije tako česta.*

Korak 5. Ponoviti Korake 3 i 4 sve dok je $s_k \neq 0$ ili ga nije moguće izračunati (tj. $c_i' \leq 0$ za svako i).

Korak 6. Izračunati vrednost ciljne funkcije $z = c^T x$ gde su x i c dobijeni u poslednjem koraku algoritma i izbaciti rezultat.

Test optimalnosti. Neka je x vektor koji smo dobili primenom algoritma, i neka su A, b i c iz (4.6.1) poznati. Neka je B bazična matrica koja se sastoji od kolona matrice A koje odgovaraju bazičnim koordinatama x_i vektora x, neka je c_B^T vektor koji se sastoji od koordinata vaktora c koje odgovaraju bazičnim koordinatama x_i vektora x, i neka je p_j $j$ta kolona nebazične kolone od A.

Korak 7. Izračunati $y^T = c_B^T B^{-1}$ (vektor vrsta).

Korak 8. Izračunati $z_j - c_j = y^T p_j - c_j$ za sve nebazične vektore p_j.

Korak 9. Ako je $z_j - c_j \leq 0$ za svako j, rešenje je optimalno; u suprotnom rešenje je neograničeno ili algoritam ne može da odredi optimalno rešenje. U tom slučaju se optimalno rešenje dobija primenom simpleks algoritma koristeći vektor x kao dopustivu početnu tačku.

Komentar algoritma DHALP.

U Koraku 1 učitavamo podatke. Korak 2 proverava saglasnost jednačina $Ax = b$. U slučaju $e = AA^\dagger b \neq b$, problem nema dopustivo rešenje. Opšte rešenje sistema $Ax = b$ je $x = A^\dagger b \pm Pz$, gde je $P = (I - AA^\dagger)$ operator ortogonalne projekcije koji proizvoljan vektor z projektuje ortogonalno na nul prostor matrice A. Izračunavamo tačku c' u nul prostoru matrice A u Koraku 3. Vektor $x = d - c's_k = A^\dagger b - (I - A^\dagger A)cs_k$ je oblika $A^\dagger b - Pz$, gde cs_k odgovara proizvoljnoj koloni vektora z. Skalar s_k se u Koraku 3 računa na taj način iz dva razloga:

(i) jedna (ili više) koordinata x_i vektora x se izjednačavaju sa nulom,

(ii) smanjuje se vrednost ciljne funkcije.

Osim toga vektor x se pomera u skup dopustivih rešenja, ukoliko već nije u njemu. Ponekad (prilično retko) se desi da se bazična koordinata izjednači sa nulom i označi kao nebazična. Otvoren je problem pod kojim se potrebnim i dovoljnim uslovima to ne može desiti.

U Koraku 4 se eliminišu promenljive x_i koje se izjednačavaju sa nulom. U numeričkim eksperimentima, u više od 95% problema su takve koordinate nebazične.

U Koraku 5 proveravamo kriterijum za kraj algoritma dok u Koraku 6 dajemo izlazne podatke. U Koracima 7, 8 i 9 proveravamo da li je rešenje optimalno.

Najčešće je matrica B kvadratna i nesigularna i u tom slučaju optimalnost rešenja proveravamo direktno.

Ako je matrica B pravougaona ($m < n$), tada dodajemo jednu (ili više) vrsta matrici B i jedan (ili više) odgovarajućih elemenata vektoru b tako da novodobijen matrica B i vektor b ne menjaju vektor x i da je pri tome B kvadratna nesingularna matrica. Nakon toga primenimo test optimalnosti kao u Koracima 7, 8 i 9. Ukoliko nije $z_j - c_j \leq 0$ za svako j, primenimo simpleks algoritam koristeći x kao početnu tačku.

5 Opis korišćenih programa

U ovoj sekciji su opisani neki od programa koji su korišćeni u prethodnim sekcijama. Program GEOM se može koristiti kao ilustracija za geometrijski metod rešavanja dvodimenzionog linearnog programiranja. Pomoću programa Marplex i RevMarplex implementirane su opisane modifikacije simpleks algoritma.

5.1 Program GEOM

U prvom delu funkcije GEOM traži se skup dopustivih rešenja, odnosno njegove ekstremne tačke [80].

Najpre se prikazuje skup ograničenja koristeći funkciju `InequaltyPlot` koja se nalazi u paketu `Graphics‘InequalityGraphics`. Rešavajući sistem nejednačina određujemo oblast dopustivih rešenja `h`. Koristimo funkciju `InequalitySolve` iz paketa `Algebra‘InequalitySolve`. Ekstremne tačke pamtimo u `res` i dobijamo ih u preseku svake dve prave dobijene prevođenjem nejednačina iz liste ograničenja u jednačine. Sada sortiramo skup ekstremnh tačaka `res` prema udaljenosti od prave određene funkcijom cilja.

```
GEOM[f_, g_List] :=
    Module[{res = {}, res2 = {},var,p2,h,g1,i,j,res,res2,p1,mx,my,r,cf,n},

    (* Nalazenje skupa dopustivih resenja *)
      var = Variables[f];
      p2 = InequalityPlot[g, {var[[1]]}, {var[[2]]},
                          AspectRatio->1, DisplayFunction->Identity
                         ];
      h = g/.{List->And};
      h = InequalitySolve[h, var];
      If [h == False, Print["Problem is infeasible!!!!"]; Break[]; ];
      g1 = g/.{LessEqual->Equal, GreaterEqual->Equal, Less->Equal, Greater->Equal};
      For[i = 1, i=Length[g1]-1, i++,
        For[j= i+1, j = Length[g1], j++,
            t = FindInstance[g1[[i]] && g1[[j]] && h, var];
            If[t != {},
               AppendTo[res, {t[[1, 1, 2]], t[[1, 2, 2]]}];
               AppendTo[res2, {ReplaceAll[f, t[[1]]], t[[1]]}];
            ];
        ];
      ];
```

```
res2 = Union[res2];     res = Union[res];

(* Graficki prikaz *)
p1 = ListPlot[res,
 PlotStyle -> {PointSize[0.025], Hue[1]}, DisplayFunction -> Identity];
r = Solve[f == 0, var[[2]]][[1, 1, 2]];

mx = Max[Table[res[[i, 1]], {i, 1, Length[res]}]];
my = Max[Table[res[[i, 2]], {i, 1, Length[res]}]];
Print[res]; Print[res2];
For[i = 1, i = Length[res], i++,
  n = ReplaceAll[y - r, res2[[i, 2]]];
  cf[i] = Plot[r + n, {x, -mx/10, mx*1.1},
    PlotStyle -> {Thickness[0.007], Hue[1]}, DisplayFunction->Identity];
];

 ShowAnimation[Table[Show[p2, p1, cf[i], AspectRatio->1,
  PlotRange -> {{-mx/10, mx*1.1}, {-my/10, my*1.1}},
  TextStyle -> {FontFamily -> \"Times", FontSize->14},
  AxesLabel -> TraditionalForm /@ var,
  PlotLabel -> "Korak " <> ToString[i]],
  {i, 1, Length[res], 1}]
 ];
 If[res2[[Length[res2], 1]] == res2[[Length[res2] - 1, 1]],
      Print["Optimal solution is given by l*", res[[Length[res]]],
          "+(1-l)*", res[[Length[res] - 1]], ",    0=l=1"],
      Print["Optimal solution is: ", res[[Length[res]]]];
 ];
];
```

5.2 Program MarPlex

Razmotrimo najvažnije detalje implementacije programa MarPlex.

```
AdvModification FreeVariables MainModule, GlobalVariables, Output,
Modification, MPS, SimplexBazFeasible, SimplexBazNotFeasible,
SimplexMaximMinim, SimplexEquation, Input, Presolver Univerzal.
```

Ovde navodimo samo neke bitne procedure, jer je celokupan kôd suviše veliki. Napomenimo samo da modul `MainModule` služi za povezivanje drugih procedura u jedinstvenu celinu. U modulu `Output` se štampaju izlazni podaci a to je vrednost funkcije cilja (ili poruka ako je funkcija cilja neograničena ili ako je problem nedopustiv) kao i vrednosti promenljivih u optimalnoj tački. U modulu `GlobalVariables` deklarisane su globalne promenljive koje se koriste u drugim procedurama. To su:

1. Brojevi `m` i `n` koji predstavljaju dimenzije problema.
2. Matrica `a()()` koja predstavlja Tuckerovu tabelu.
3. Nizovi `nbasicv` i `basicv` koji predstavljaju prvu vrstu i zadnju kolonu matrice Tuckerove tabele.
4. Četiri konstante (`Infinity, Impossibile, Solution, Error`) jedna logička promenljiva `maksim` koja označava da li se radi o problemu maksimizacije ili minimizacije, kao i tačnost `eps`.

Najvažnije konstante i promenljive su definisane (deklarisane) na sledeći način:

```
Option Explicit

Public Const Infinity = 1 Public Const Impossible = 2 Public Const
Solution = 0 Public Const Error = -1

Public a() As Double Public eps As Double Public m As Integer
Public n As Integer Public basicv() As Integer Public nbasicv() As
Integer Public modif As Boolean

Public outk() As Boolean Public outv() As Boolean

Public maxim As Boolean
```

Svi nizovi i matrice su deklarisani bez dimenzija. Dimenzije svakog niza se posle učitavanja podataka postavljaju korišćenjem naredbe `ReDim`. Modul `MPS` služi za čitanje podataka iz MPS fajla. Ostali moduli implementiraju odgovarajuće faze simpleks algoritma. Od njih navodimo samo najbitnije.

Sledi kod procedure `SolveSystem` koja predstavlja implementaciju algoritma **Replace**.

```
Public Sub SolveSystem(pivrow As Integer,pivcol As Integer)
    Dim p As Double, nv As Integer, nk As Integer, h As Integer
        j As Integer, pomint As Integer

  p = a(pivrow, pivcol)
  nv = 0
  nk = 0
  For h=1 To n+1
    If (a(pivrow,h)<>0) And (h<>pivcol) And (outk(h)=True) Then
      nv=nv+1
      v(nv)=h
    End If
  Next h
  For h=1 To m+1
    If (a(h,pivcol)<>0) And (h<>pivrow) And (outv(h)=True) Then
      nk = nk+1
      k(nk) = h
    End If
  Next h
  For h=1 To nk
    For j=1 To nv
      a(k(h),v(j))=a(k(h),v(j))-a(pivrow,v(j))*a(k(h),pivcol)/p
      If Abs(a(k(h),v(j)))<epsil Then a(k(h),v(j))=0
    Next j
  Next h
  For h=1 To nk
    a(k(h),pivcol)=a(k(h),pivcol)/p
    If abs(a(k(h),pivcol)) <epsil Then a(k(h),pivcol)=0
  Next h
  For h=1 To nv
    a(pivrow,v(h))=a(pivrow,v(h))/p
    If Abs(a(pivrow,v(h))) <epsil Then a(pivrow,v(h))=0
  Next h
```

```
  a(pivrow,pivcol)=1/p
  If Abs(a(pivrow,pivcol))<epsil Then a(pivrow,pivcol)=0
  pomint = basicv(pivcol)
  basicv(pivcol) = nbasicv(pivrow)
  nbasicv(pivrow) = pomint
End Sub
```

Ovde su skupovi V i K predstavljeni redom kao nizovi `v` i `k`.
Moduli `SimplexBazNotFeasible`, `Modification`, `AdvModification` predstavljaju redom implementacije algoritma **NoBasicMax**, **ModNoBasicMax** i **AdvModNoBasicMax**.

Kôd funkcije `SimplexMod` je:

```
Public Function SimplexMod(res As Integer) As Double
  Dim i As Integer, j As Integer, p As Integer, from As Integer,
      from1 As Integer, found As Boolean

  from = 1
  frm_Marplex.Status.Text = "Finding first basic solution...."
  Do
    DoEvents
    If work = 1 Then Exit Function
    frm_Marplex.BazNed.Text = val(frm_Marplex.BazNed.Text) + 1
    i = FindI(from)
    If i = -1 Then Exit Do
    from = i
    from1 = 1
    found = False
    Do
      j = FindJ(from1, i)
      If j > 0 Then from1 = j + 1
      If j = -2 Then
        res = Impossible
        Exit Function
      End If
      If j = -1 Then Exit Do
      p = FindPivotRow(j)
      If p = -1 Then
        SolveSystem i, j
        found = True
      ElseIf a(p, n + 1) / a(p, j) >= a(i, n + 1) / a(i, j) Then
        SolveSystem i, j
        found = True
      End If
    Loop Until found
    j = from1 - 1
    If found = False Then
      SolveSystem p, j
    End If
  Loop
  SimplexMod = SolveBasicFeasible(res)
End Function
```

U implementaciji algoritma nisu konstruisani skupovi B i Q već se brojevi i_s i j_p (`i` i `j` u kodu) traže u ciklusu `Do-Loop`. Pri tome se pamte promenljive `from`

i `from1` kao predhodne vrednosti za `i` i `j`. Procedurama `FindI` i `FindJ` traže se brojevi i i j.

Poboljšana modifikacija (algoritam **AdvModNoBasicMax**) je implementirana u funkciji `SimplexAdvMod`. Sledi kod ove funkcije:

```
Public Function SimplexAdvMod(res As Integer) As Double
    Dim bliz As Integer, ii As Integer, i As Integer, j As Integer, k As Integer,
        l As Integer, p As Integer, niter As Integer, pivrow As Integer,
        pivcol As Integer, s() As Double, piv() As Integer, max As Double,
        maxpiv As Double, imaneg As Boolean, negbs As Integer
    frm_Marplex.Status.Text = "Finding first basic solution...."
    niter = 0
    ReDim s(n): ReDim piv(n)
    Do
        If work = 1 Then Exit Function
        DoEvents
        frm_Marplex.BazNed.Text = val(frm_Marplex.BazNed.Text) + 1
        pivrow = -1: maxpiv = 0: pivcol = -1
        For j = 1 To n: s(j) = -1: piv(j) = 0: Next j
        'Pokusavamo da nadjemo negativni pivot
        ii = -1:
        For i = 1 To m
            If a(i, n + 1) < 0 Then
                p = -1: ii = i: imaneg = False
                For j = 1 To n
                    If (a(i, j) < 0) Then
                      If s(j) = -1 Then
                          p = FindPivotRow(i, j)
                          piv(j) = p
                           s(j) = a(p, n + 1) / a(p, j)
                           If (a(p, n + 1) < 0) And (Abs(a(p, j)) > maxpiv) Then
                               pivrow = p
                               pivcol = j
                               maxpiv = a(pivrow, pivcol)
                               Exit For
                           End If
                       Else
                           p = piv(j)
                       End If
                    End If
                Next j
                If p = -1 Then
                    res = Impossible
                    Exit Function
                End If
                If pivrow > 0 Then Exit For
            End If
        Next i
        negbs = 0
        For i = 1 To m
            If a(i, n + 1) < 0 Then negbs = negbs + 1
        Next i
        frm_Marplex.negb.Text = negbs
        If ii = -1 Then Exit Do
        If pivrow > 0 Then
            SolveSystem pivrow, pivcol
```

```
        Else
            'If impossible, we are choosing the positive one
            'and applying anticyclic rules
            'Every time we are choosing last negative b_i
            bliz = 30000
            For j = 1 To n
                If (a(ii, j) < 0) And (nbasicv(j) < bliz) Then
                    bliz = nbasicv(j)
                    pivcol = j
                End If
            Next j
            bliz = 30000
            For l = 1 To m
                If (a(l, n + 1) >= 0) And (a(l, pivcol) > 0) Then
                    If (a(l, n + 1) / a(l, pivcol) = s(pivcol)) _
                    And (basicv(l) < bliz) Then
                        bliz = basicv(l)
                        pivrow = l
                    End If
                End If
            Next l
            SolveSystem pivrow, pivcol
        End If
    Loop
    SimplexAdvMod = SolveBasicFeasible(res)
End Function
```

Promenljiva `negbs` pamti trenutni broj negativnih elemenata b_i. Najpre pokušavamo da nađemo pivot element tako da je uslov Leme 3.2.1 zadovoljen. Ukoliko to nije moguće, biramo pozitivan pivot element i primenjujemo anticiklična pravila. Značenje ostalih promenljivih je isto kao i u prethodnoj proceduri.

Funkcija Simplex koja predstavlja implementaciju algoritma **NoBasicMax** data je sledećim kodom:

```
Public Function Simplex(res As Integer) As Double
    Dim i As Integer, pivcol As Integer, pivrow As Integer, h As Integer
    frm_Marplex.Status.Text = "Finding first basic solution...."
    Do
    If work = 1 Then Exit Function
    DoEvents
    frm_Marplex.BazNed.Text = val(frm_Marplex.BazNed.Text) + 1
    i = FindI
    If i = 0 Then
      Simplex = SolveBasicFeasible(res)
      Exit Do
    End If
    pivcol = FindPivotColNotFeasible(i)
    If pivcol = 0 Then
      res = Impossible
      Exit Do
    End If
    pivrow = FindPivotRowNotFeasible(i, pivcol)
    SolveSystem pivrow, pivcol
  Loop
End Function
```

Uloga pomoćnih procedura `FindI`, `FindPivotColNotFeasible` i `FindPivotRowNotFeasible` je ista kao i kod predhodne funkcije.

Obe ove funkcije posle svog izvršenja pozivaju funkciju `SolveBasicFeasible` kojom se rešava bazično dopustiva Tuckerova tabela.

5.3 Program RevMarPlex

Program RevMarPlex se sastoji od sledećih modula:

`DefaultFunctions Revised Simplex`

U modulu `DefaultFunctions` nalaze se pomoćne funkcije dok su u `RevisedSimplex` modulu funkcije: `SolveBasicFeasible`, `FindBasicNotFeasible`, `FindBasicFeasibleMod` i `RevisedSimplex`. Prve tri funkcije implementiraju odgovarajuće faze revidiranog simpleksa (poslednja implementira modifikaciju) dok je četvrta glavna funkcija i nju korisnik poziva. Njeni parametri su:

- `f_`: Linearna funkcija koju treba optimizovati u simboličkoj formi,
- `A_`: Lista ograničenja u simboličkoj formi,
- `max_`: Logička promenljiva. `True` za maksimizaciju, `False` za minimizaciju.

Funkcija `MakeMatrix` iz modula `DefaultFunctions` vrši konverziju problema iz simboličkog u matrični oblik. Parametri funkcije `FindBasicFeasible` su redom matrica sistema, RHS vektor, kao i indeksi kolona koje čine prvu bazu (prvo bazično rešenje).

Sledi kod funkcije `FindBasicFeasible`.

```
FindBasicFeasible[A_, b_, base1_] :=
    Block[{pom=0,iter=0,bs, bz1, bm, h=0,i=0,j=0,k=0,B={},N={},bz={},ik,row={},
      col={}, nbase={}, base={}, pr=0, pc=0, eps, min=0, negbi, radi},
      {m, n} = Dimensions[A];    base = base1;
      B = PrepareMatrix[A, base];
      For [i = 1, i <= m, i++, nbase = Join[nbase, {i}]];
      nbase = Complement[nbase, base];
      Nb = PrepareMatrix[A, nbase];  eps = 10^-6; bm = {};  radi = True;  k = 0;
      While[True,
        k++;     bs = Sort[base];     bm = Join[bm, {bs}];
        bz = Chop[LinearSolve[Transpose[B], b], eps];
        radi = False; negbi = 0;
        For [h = 1, h <= n, h++,
          If [bz[[h]] < 0,
              radi = True;  i = h; negbi++;
          ];
        ];

        If [Mod[k, 10] == 0,
            Print["Finished ", k, " iterations, ", negbi, " negative b_i-s"]];
        If [radi == False,
            Return[base]
        ];
```

```
      (* Reconstructing Row *)
      ik = Table[0, {n}]; ik[[i]] = 1;
      row = Chop[{LinearSolve[B, ik]}.Transpose[Nb], eps][[1]];

      pc = 0;     bliz = Infinity;
      For [h = 1, h <= m - n, h++,
        If [row[[h]] < 0,
            If [nbase[[h]] < bliz,
              pc = h;          bliz = nbase[[h]];
            ];
            Break[];
        ];
      ];

      If [pc == 0, Return[{}]];

      (*Reconstructing Column*)
      col = Chop[LinearSolve[Transpose[B], A[[nbase[[pc]]]]], eps];

      min = bz[[i]]/row[[pc]];    pr = i;   bliz = 0;
      For [h = 1, h <= n, h++,
        If [(col[[h]] > 0) ,
          If [(bz[[h]]/col[[h]]<min) ||
                   ((bz[[h]]/col[[h]]==min) && (bliz>base[[pr]])),
              min = bz[[h]]/col[[h]];  pr = h; bliz = base[[pr]];
          ];
        ];
      ];
      B[[pr]] = A[[nbase[[pc]]]]; Nb[[pc]] = A[[base[[pr]]]];
      pom = nbase[[pc]];  nbase[[pc]] = base[[pr]];  base[[pr]] = pom;
    ];
    Return[base];
  ];
```

Za rešavanje linearnih sistema jednačina, za rekonstrukciju redova i kolona Takerove tabele koristili smo funkciju `LinearSolve`. Promenljive `B` i `Nb` pamte redom bazičnu i nebazičnu matricu A_B i A_N.

Parametri funkcije `FindBasicFeasibleMod` su isti kao i parametri ranije opisane funkcije `FindBasicFeasible`. Sledi kod funkcije `FindBasicFeasibleMod`.

```
FindBasicFeasibleMod[A_, b_, base1_] :=
    Block[{pom=0, iter=0, bs, bz1, bm, h=0, i=0, j=0, k=0, B={}, N={}, bz={}, ik,
          row={}, col={}, nbase={}, base={}, pr=0, pc=0, eps, min=0, radi, negbi},
      {m, n} = Dimensions[A];     base = base1;
      B = PrepareMatrix[A, base];
      For [i = 1, i <= m, i++, nbase = Join[nbase, {i}]];
      nbase = Complement[nbase, base];
      Nb = PrepareMatrix[A, nbase];
      eps = 10^-6;     bm = {};    radi = True;     k = 0;

      While[True,
        k++;     bs = Sort[base];
        bm = Join[bm, {bs}];     bz = Chop[LinearSolve[Transpose[B], b], eps];
        radi = False;
        For [h = 1, h <= n, h++,
```

```
      If [bz[[h]] < 0,
          radi = True;      i = h;
        ];
      ];
      If [radi == False,  Return[base]      ];

      (* Reconstructing Row *)
      ik = Table[0, {n}]; ik[[i]] = 1;
      row = Chop[{LinearSolve[B, ik]}.Transpose[Nb], eps][[1]];

      pc = 0; negbi = 0;
      bliz = Infinity;
      For [h = 1, h <= m - n, h++,
        If [row[[h]] < 0,
            If [nbase[[h]] < bliz,
              pc = h; negbi++;        bliz = nbase[[h]];
            ];
            Break[];
        ];
      ];

      If [Mod[k, 10] == 0,
        Print["Finished ", k, " iterations, ", negbi, " negative b_i-s"]];

      If [pc == 0, Return[{}]];

      (*Reconstructing Column*)
      col = Chop[LinearSolve[Transpose[B], A[[nbase[[pc]]]]], eps];

      min = bz[[i]]/row[[pc]];
      pr = i;
      bliz = 0;
      For [h = pr + 1, h <= n, h++,
        If [(col[[h]]*bz[[h]] > 0) ,
            If[(bz[[h]]/col[[h]] < min) ||
               ((bz[[h]]/col[[h]] == min) && (bliz > base[[pr]])),
                  min = bz[[h]]/col[[h]]; pr = h; bliz = base[[pr]];
            ];
        ];
      ];
      B[[pr]] = A[[nbase[[pc]]]];    Nb[[pc]] = A[[base[[pr]]]];
      pom = nbase[[pc]];  nbase[[pc]] = base[[pr]];    base[[pr]] = pom;
    ];
    Return[base];
  ];
```

Literatura

[1] E.D. Andersen, J. Gondzio, C. Mészáros and X. Xu *Implementation of interior point methods for large scale linear programming*,Technical report, HEC, Université de Genève, 1996.

[2] Ángel Santos Palomo and Pablo Guerrero García, *Resolviendo una sucesión de sistemas compatibles dispersos*, XXVI Congreso National de Estadística e Investigación Opertiva Úbeda, 6–9 de noviembre de 2001.

[3] Ángel Santos Palomo and Pablo Guerrero García, *Solving a sequence of sparse compatibile Systems*, 19th Biennial Conf. Numerical Analysis, Dundee, Scotland, June 2001.

[4] M.D. Ašić and V.V. Kovačević-Vujčić,, *Ill-conditionedness and interior-point methods,* Univ. Beograd Publ. Elektrotehn. Fak., **11** (2000), 53–58.

[5] D. Bertsimas, J.N. Tsitsiklis, *Introduction to linear optimization*, Athena Scientific, Belmont, Massachusetts, 1997.

[6] A. Ben-Israel and T.N. E. Greville, *Generalized Inverses. Theory and Applications, Second edition*, CMS Books in Mathematics/Ouvrages de Mathmatiques de la SMC, 15. Springer-Verlag, New York, 2003.

[7] M.A. Bhatti, *Practical optimization with MATHEMATICA applications,* Springer Verlag Telos, 2000.

[8] R. Bixby, *Implementing the simplex method; the initial basis*, ORSA Journal on Computing **4** (1992), 267–284.

[9] R.G. Bland, *New finite pivoting rules for the simplex method*, Mathematics of Operations Research **2** (1977), 103–107.

[10] B.D. Bounday, *Basic linear programming*, Edvard Arnold, Baltimore, 1984.

[11] S.L. Campbell and C.D. Meyer, *Generalized inverses of linear transformations*, Pitman, London, San Francisco, Melbourne, 1979.

[12] A. Charnes, *Optimality and degeneracy in linear programming*, Econometrica **20** No.2 (1952)

[13] Richard J. Charon, Tim Traynor, Wang Cheng, *The minimal angles method of Stojkovic and Stanimirovic*, Technical report, University of Windsor, Ontario, Canada 2004.

[14] A.R. Conn, *Linear programming via a nondifferentiable penalty function*, SIAM J. NUMER.ANAL., Vol. **13**, No. 1 (1976), 145–154.

[15] H.W. Corley, Jay Rosenberger, Wei-Chang Yeh and T.K. Sung, *The cosine simplex algorithm*, The International Journal of Advanced Manufacturing Technology **27**, Numbers 9-10, (2006), 1047-1050.

[16] H.W. Corley, Jay Rosenberger and T.K. Sung, *Constraint Optimal Selection Techniques (COSTs) for Nonnegative Linear Programming Problems*, under review in European Journal of Operational Research.

[17] D. Cvetković, M. Čangalović, Đ. Dugošija, V. Kovačević-Vujčić, S. Simić, J. Vuleta, *Kombinatorna optimizacija-Matematička teorija i algoritmi* Društvo operacionih istraživača Jugoslavije DOPIS, Beograd 1996.

[18] Cvetković, D. *Diskretna matematika*, Prosveta, Niš, 1996.

[19] J. Czyzyk, S. Mehrotra and S.J. Wright, *PCx User Guide*, Optimization Thechnology Center, Technical Report 96/01 (1996) 1–21.

[20] G. B. Dantzig, *Programming of Interdependent Activities, Mathematical Model*, Econometrica 17, 1949, 200–211.

[21] G.B. Danzing, *Linear programming and Extensions*, Princeton University Press, Princeton, New Yersy 1963.

[22] A. Dax, *Linear programming via least squares*, Linear Algebra and its Applications **111** (1988), 313–324.

[23] A. Deza, E. Nematollahi, T. Terlaky, *How good are interior point methods? Klee-Minty cubes tighten iteration-complexity bounds.*, AdvOL-Report #2004/20 Advanced Optimization Laboratory, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, December 2004.

[24] J. Egerváry, *Matrixok combinatorius tulájdonsagairol*, Math.Fiz. Lopak, 1931.

[25] J.J. Forrest, D. Goldfarb, *Steepest-edge simplex algorithms for linear programming*, Mathematical Programming 57 (1992), 341–374.

[26] Y.-M. Gao, R.-C. Gan, *The Study of Criteria of Linear Programming with Ineffective Variables*, Journal of Taiyuan University of Technology, Vol.35 No.3 (2004), 371-374.

[27] Y.-M. Gao, R.-C. Gan, *The Study of Criteria of Linear Programming with Ineffective Constraints*, Journal of Taiyuan University of Technology, Vol.35 No.5 (2004), 633-636.

[28] B. Gartner, M. Henk, G.M. Ziegler, *Randomized Simplex Algorithms on Klee-Minty Cubes*, Combinatorica 18 (1998), 349–372.

[29] J. Gondzio, *HOPDM (Version 2.12), A fast LP solver based on a primal-dual interior point method*, European Journal of Operations Research **85**, (1995), 221–225.

[30] Hélcio Vieira Junior, Marcos Pereira Estellita Lins, *An improved intial basis for the simplex algorithm*, Computers and Operations Research Volume 32, Issue 8, (2005), 1983 - 1993.

[31] F. L. Hitchcock, *The distribution of a product from several sources to numerous localitites*, Journal Math. and Phys. 20, 1941, 224–230.

[32] Jian-Feng Hu, *A note on "An improved initial basis for the simplex algorithm"*, Computers and Operations Research 34, (2007), 3397–3401.

[33] J.P. Ignizio, *Linear programming in single-multiple-objective systems,* Englewood Cliffs: Prentice Hall, 1982.

[34] V. Ivanović, *Pravila za proračun potrebnog broja transportnih sredstava*, Vojnoizdavački glasnik, sveska 1-3, 1940, 1–10.

[35] L.V. Kantorovich *Matematicheskie metodi v organizacii i planirovanii proizvodstva*, Izd. LGU, 1939.

[36] N. Karmarkar,*A new polynomial-time algorithm for linear programming*, Combinatorica, 4, 1984, 373-395.

[37] L.G. Khachian, *A Polinomial Algorithm in Linear Programming*, Doklady Akademii Nauk SSSR, Vol. 244, No, 5, 1979, pp. 1093-1096.

[38] V. Klee, G.L. Minty, *How good is the simplex method?*, O. Shisha, ed., Inequalities III, Academic Press, New York, 1972, pp. 159–175.

[39] V. Klee, P. Kleinschmidtm, *The d-step conjecture and its relatives*, Math. Operations Research 12, 1987, pp. 718–755.

[40] M. Kojima, N. Megiddo, T. Noma and A. Yoshise, *A unified approach to interior-point algorithms for linear complementarity problems*, Lecture Notes in Computer Science 538, Springer-Verlag, Berlin, 1991.

[41] T.C.T. Kotiah, D.I. Steinberg, *Occurences of cycling and other phenomena arising in a class of linear programming models*, Communications of the ACM, 20, 1977, pp. 107–112.

[42] V. Kovačević-Vujčić, and M.D. Ašić, *Stabilization of interior-point methods for linear programming*, Computational Optimization and Applications, **14** (1999), 1–16.

[43] S.G. Kulkarni and K.C. Sivakumar, *Applications of generalized inverses to interval linear programs in Hilbert spaces*, Numer. Funct. Anal. and Optimiz. **16**(7 & 8) (1995), 965–973.

[44] S.G. Kulkarni and K.C. Sivakumar, *Explicit solutions of a special class of linear programming problems in Banach spaces*, Acta Sci. Math. (Szeged) **62** (1996), 457–465.

[45] Imran Maqsood, *Development of simulation-and optimization-based decision support methodologies for environmental systems management*, Doctoral thesis University of Regina, August 2004, pp 495

[46] K. Miettinen, *Nonlinear Multiobjective Optimization*, Kluwer Academic Publishers, Boston 1999.

[47] *Microsoft Developer Network (13)*, Microsoft Corporation Inc., 1998.

[48] G.V. Milovanović, *Numerička analiza I deo*, Naučna knjiga, Beograd 1991.

[49] Y. Nesterov and A. Nemirovskii, *Interior point polynomial methods in convex programming*, SIAM, Philadelphia, 1993.

[50] G. Owen, *Game theory*, W.B. Saunders Company, Philadelphia, London, Toronto, 1968.

[51] E. Nering, A. Tucker, *Linear Programs and Related Problems*, Academic Press, New York, 1993.

[52] *www.netlib.org*

[53] P.Q. Pan, *Practical finite pivoting rules for the simplex method*, OR Spektrum 12, (1990), 219–225.

[54] P.Q. Pan, *A simplex - like method with bisection for linear programming*, Optimization 22, (1991), 717–743.

[55] P.Q. Pan, *A variant of the dual pivoting rule in linear programming*, Journal of Information and Optimization Sciences, 15, No 3, (1994), 405–413.

[56] P.Q. Pan, *The most-obtuse-angle row pivot rule for achieving dual feasibility: a computational study*, Europ. J. of Oper.Res. **101** (1997), 167–176.

[57] S. Paulraj, C. Chellappan, T. R. Natesan, *A heuristic approach for identification of redundant constraints in linear programming models*, International Journal of Computer Mathematics, **83** (2006), 675–683.

[58] M.D. Petković, P.S. Stanimirović, N.V. Stojković, *Two modifications of revised simplex metod*, Matematicki Vesnik, 54 (2002), pp. 163–169.

[59] M.D. Petković, P.S. Stanimirović, *Partitioning method for two-variable rational and polynomial matrices*, Mathematica Balcanica vol. 19, 2005, pp. 185–194.

[60] M.D. Petković, P.S. Stanimirović, *Symbolic computation of the Moore-Penrose inverse using partitioning method*, International Journal of Computer Mathematics, 82(March 2005), pp. 355–367.

[61] M.D. Petković, P.S. Stanimirović, *Interpolation algorithm of Leverrier-Faddev type for polynomial matrices*, Numerical Algorithms **42** (2006), 345–361.

[62] M.D. Petković, P.S. Stanimirović, *Interpolation algorithm for computing Drazin inverse of polynomial matrices*, Linear Algebra and its applications, 422 (2007), 526-539.

[63] L.D. Pyle, *The weighted genalized inverse in nonlinear programming - active set selection using a variable-metric generalization of the simplex algorithm* Lecture Notes in Economics and Mathematical System, Austin, Texas, September 13-15, 1977.

[64] L.D. Pyle, *The genalized inverse in linear programming. Basic structure*, SIAM L. appl. Math. **22** (1972), 335–355.

[65] L.D. Pyle and R.E. Cline, *The genalized inverse in linear programming - interior gradient projection methods*, SIAM L. appl. Math. **24** (1973), 511–534.

[66] M. Sakarovitch, *Linear programming,* Springer-Verlag, New York, 1983.

[67] S.K. Sen and A. Ramful, *A direct heuristic algorithm for linear programming*, Proc. Indian Acad. Sci. (Math. Sci.), Vol. 110, No. 1, (2000), 79–101.

[68] Chanin Srisuwannapa, *Coordinate transformations for solving linear programming problems*, Doctoral thesis, Case Western Reserve University, Cleveland, 2007.

[69] W. Stadler (Ed.), *Multicriteria Optimization in Engineering and in the Sciences*, Plenum Press, New York, 1988.

[70] P.S. Stanimirović, N.V. Stojković, M.D. Petković, *Modification and implementation of two phases simplex method*, International Journal of Computer Mathematics, Prihvaćen za štampu.

[71] P.S. Stanimirović, M.D. Petković, *Computation of generalized inverses of polynomial matrices by interpolation*, Appl. Math. Comput., Vol 172/1 (January 2006), pp 508-523.

[72] P.S. Stanimirović, G.V. Milovanović, *Programski paket MATHEMATICA i primene*, Elektronski fakultet u Nišu, Edicija monografije, Niš, 2002.

[73] R. Stanojević, *Linearno Programiranje*, Institut za ekonomiku industrije, Beograd 1966.

[74] N.V. Stojković, *Primal-dual i simpleks metodi za rešavanje problema linearnog programiranja*, Doktorska disertacija, PMF Niš, 2001.

[75] N.V. Stojković and P.S. Stanimirović, *On elimination of excessive constraints in linear programming*, SYMOPIS, (1999), 207–210.

[76] N.V. Stojković and P.S. Stanimirović, *Two direct methods in linear programming*, Europ. J. of Oper.Res. **131(2)** (2001), 417–439.

[77] N.V. Stojković and P.S. Stanimirović, *A method for solving some classes of linear programming*, Proceedings of XIII Conference on Applied Mathematics, Igalo (1998), 153–160.

[78] N.V. Stojković, P.S. Stanimirović, M.D. Petković, *Several Modifications of Simplex Method*, Filomat, (2003), 169–176.

[79] J. Strayer, *Linear Programming and Its Applications*, Springer-Verlag 1989.

[80] M. Tasić, P.S. Stanimirović, I.P. Stanimirović, M.D. Petković, N.V. Stojković, *Some useful MATHEMATICA teaching examples*, Facta Universitatis(Niš) Series Electronics and Energetics, vol. 18, No. 2, August 2005, pp. 329-344.

[81] Ed. T. Terlaky, *Interior point methods of mathematical programming*, Kluwer Acad. Publ., Dodrecht, Boston, London , 1996.

[82] R. J. Vanderbei,*Linear Programming: Foundations and Extensions*, Princeton, N.J. 2001.

[83] R. J. Vanderbei,*LOQO: An interior-point code for quadratic programming, Technical Report SOR-94-15*, Department of Civil Engineering and Operations Research, Princeton University, Princeton, N.J. 1994.

[84] S. Vukadinović, S. Cvejić *Matematičko programiranje,* Univerzitet u Prištini, Priština, 1996.

[85] C.M.Wang, *Comments on Two Direct Methods in Linear Programming*, Master Thesis, Windsor, Ontario, Canada, 2004.

[86] Wang Cheng, *Comments on Two Direct Methods in Linear Programming*, under review in European Journal of Operational Research.

[87] G.Wang, Y.Wei and S. Qiao, *Generalized Inverses: Theory and Computations*, Science Press, Beijing, 2004.

[88] L. Wei, *A Note on Two Direct Methods in Linear Programming*, Europ. J. of Oper.Res. 158, (2004), 262–265.

[89] D.J. White, *A bibliography on the applications of mathematical programming multiple-objective methods*, Journal of the Operational Research Society, 41(8), 1990, pp. 669–691.

[90] S. Wolfram, *Mathematica: a system for doing mathematics by computer*, Addison-Wesley Publishing Co, Redwood City, California, 1991.

[91] S. Wolfram, *Mathematica Book, Version 3.0*, Wolfram Media and Cambridge University Press, 1996.

[92] S. Wolfram, *The Mathematica Book, 4th ed.*, Wolfram Media/Cambridge University Press, 1999.

[93] S.J Wright, *Primal-dual interior-point methods*, SIAM, Philadelphia, 1997.

[94] Y. Zang, *User's guide to LIPSOL*, Department of Mathematics and Statistics, University of Maryland, Baltimore County, Baltimore, Md., July 1995.

Glava 2

Primal-dual metodi unutrašnje tačke

1 Primal-dual algoritmi

Simpleks metod je od 1947. godine bio standardna procedura za rešavanje problema linearnog programiranja. Algoritam je u praktičnim primenama potvrdio svoju efikasnost tako da je Dantzig u [15] tvrdio da broj iteracija retko prelazi dvostruku dimenziju problema što se i eksperimentalno verifikuje na slučajno generisanim problemima, a što se potvrđuje sve do danas na realnim problemima. Uporedo sa razvojem linearnog programiranja, u oblasti računarstva razvija se teorija računske složenosti u okviru koje se definiše pojam polinomijalnog algoritma. Intuitivno, algoritam je polinomijalan ukoliko se broj elementarnih računskih operacija potrebnih za rešavanje proizvoljnog problema iz date klase može ograničiti polinomom po dimenziji problema. Tek 1972. godine je pokazano da postoje primeri [54] na kojima simpleks metod nije polinomijalan. U pomenutom radu je dat jednostavan primer kod koga je dopustivi skup deformisana n-dimenziona kocka sa 2^n temena za čije rešavanje simpleks metodu treba $2^n - 1$ iteracija. Prvi polinomijalan algoritam (metod elipsoida) za linearno programiranje je 1979. godine dao Kačijan [53] i pokazao da on rešava problem linearnog programiranja sa $O(nL)$ aritmetičkih operacija, gde je L broj bitova potrebnih za zapis svih parametara linearnog programa. Međutim, pokazalo se da metod elipsoida u praksi nije konkurentan simpleks metodu jer "često" dostiže gornju granicu složenosti dok to kod simpleks metoda nije slučaj. Štaviše, pokazalo se da pod izvesnim pretpostavkama o raspodeli ulaznih podataka simpleks metod jeste polinomijalan [12]. Prvi praktično primenljiv polinomijalan algoritam je 1984. godine dao Karmarkar [51]. Posle toga je razvoj unutrašnjih metopda veoma ubrzan. Pojavila se čitava familija unutrašnjih metoda, razvijeni su programski paketi koji su pokazali bolje performanse u odnosu na simpleks metod, naročito za probleme velikih dimenzija. Praksa je pokazala da se porastom dimenzije problema povećava superiornost unutrašnjih metoda. Opšte je

mišljenje da će praktičnoj upotrebi ostati oba pristupa linearnom programiranju. Pri rešavanju problema velikih dimenzija, u ranijim iteracijama će se primenjivati unutrašnje metode, a zatim će se postupak završiti simpleks metodom.

Detaljan osvrt na odnos simpleks metoda i unutrašnjih metoda, kao i pravci njihovog daljeg razvoja može se naći u radu Vere Kovačević-Vujčić [63]. Ovde treba napomenuti da su prve ideje o metodima unutrašnje tačke dali u svojim radovima von Neumann 1947. [77], Hoffman 1953. [49], Frish 1955. [35] i Dikin 1967. [18]. Međutim, računski problemi, numerička nestabilnost iteracija i loši eksperimentalni razultati su doveli do stanovišta da algoritmi unutrašnje tačke nisu konkurentni sa simpleks metodom. Polinomijalnost i računska složenost metoda unutrašnje tačke je razmatrana u radovima [31, 32, 37, 40, 50, 55, 76, 120]. Karmarkarov algoritam i metod elipsoida su detaljno opisani na našem jeziku u monografiji [13], tako da se na njima nećemo zadražavati. Takođe je u radu [63] dat opis različitih pristupa metodu unutrašnje tačke. Prema geometrijskoj interpretaciji, postojeći metodi unutrašnje tačke su klasifikovani u tri grupe:

• *Projektivni metodi* (Karmarkar [51], Anstreicher [4], De Chellinck i Vial [16], Todd i Ye [104], Ye [121], Yamashita [116], Gonzaga [41, 43]).

• *Afini metodi* (Dikin [18], Barnes [7], Gonzaga [42], Vanderbei, Monma i Morton [75], Monteiro, Adler i Resende [74], Ye [119]).

• *Metodi koje slede centralnu putanju* (Renegar [80], Gonzaga [44, 45], Roos i Vial [82, 83], Den Hertog, Roos i Terlaky [17], Renegar i Shub [81], Monteiro i Adler [73], Kojima, Mizuno i Yoshise [59]).

U zavisnosti od toga da li rešavaju primalni problem, dualni problem ili simultano oba problema, unutrašnji metodi su u pomenutom radu [63] klasifikovani na sledeći način:

• *Primalni metodi* (Karmarkar [51], Anstreicher [4], De Ghellinck i Vial [16], Ye [117], Gonzaga [44, 45], Den Hertog, Roos i Terlaky [17], Dikin [18], Barnes [7]), Vanderbei, Meketon i Freedman [106]).

• *Dualni metodi* (Yamashita [116], Gonzaga [39], Monma i Morton [75], Renegar [80], Renegar i Shub [81]).

• *Primal-dual metodi* (Monteiro i Adler [73], Mizuno, Tod i Ye [72], Gonzaga [42], Ye [119], Tanabe [102], Todd i Ye [104], Freund [28], Gonzaga i Todd [46], Kojima, Mizuno i Yoshise [56]).

Kako se primal-dual metod unutrašnje tačke u praksi potvrdio kao jedan od najefikasnijih, u ovoj glavi razmatramo teorijske osnove i različite algoritme primal-dual metoda, njihove modifikacije i implementacije. Osnovne ideje primal-dual metoda su razvijene između 1987. i 1991. godine. Za početak teorije primal-dual algoritama se smatra rad Megidda [66] iz 1987. godine. Motivisani tim radom, 1987. godine Kojima, Mizuno i Yoshise [59] razvijaju prvi polinomijalni primal-dual algoritam dugog koraka čija je kompleksnost $O(n \log 1/\varepsilon)$. Odmah zatim isti autori razvijaju primal-dual algoritam kratkog koraka [60] i poboljšavaju kompleksnost na $O(\sqrt{n} \log 1/\varepsilon)$. Slične rezultate dobijaju Monteiro i Adler [73], Mizuno, Tod i Ye [72], Gonzaga [42], Ye [119], Tanabe [102], Todd i Ye [104], Freund [28], Gonzaga

i Todd [46]. Većina savremenih programskih paketa koristi primal-dual algoritam koji je razvio Mehrotra 1992. godine u radu [70] na osnovu radova Megidda [66], Monteira, Adlera i Resendea [74] i Lustiga, Marstena i Shannoa [65].

Napomenimo da se primena primal-dual metoda može proširiti na monotone linearne komplementarne probleme, konveksno programiranje i semidefinitno programiranje, što je razmatrano u [25, 33, 68, 106].

1.1 Teorijske osnove primal-dual metoda

U analizi algoritama primal-dual metoda unutrašnje tačke pogodno je problem linearnog programiranja posmatrati u standardnom obliku:

$$\min c^T x \text{ u odnosu na uslove } Ax = b, \quad x \geq 0, \tag{1.1.1}$$

gde su $c, x \in R^n$, $b \in R^m$, i A je realna matrica tipa $m \times n$.

Dualni problem za (1.1.1) je

$$\max b^T y \text{ u odnosu na uslove } A^T y + s = c, \quad s \geq 0, \tag{1.1.2}$$

gde je $y \in R^m$ i $s \in R^n$. Iz teorije dualnosti je poznato da za dopustive vektore x i (y, s) problema (1.1.1) i (1.1.2) respektivno važi

$$b^T y \leq c^T x, \tag{1.1.3}$$

tj. ciljna funkcija duala je donje ograničenje ciljne funkcije primalnog problema i primal je gornje ograničenje za dual. Jednakost $b^T y^* = c^T x^*$ važi kada je x^* rešenje za (1.1.1) i (y^*, s^*) rešenje za (1.1.2) i zajedničku optimalnu vrednost označavamo sa z^*.

Centralno mesto u primeni metoda unutrašnje tačke imaju Karush-Kuhn-Tucker uslovi optimalnosti. Primena ovih uslova na probleme (1.1.1) i (1.1.2) daje sledeći rezultat:

Teorema 1.1.1 *Vektor $x^* \in R^n$ je rešenje za* (1.1.1) *ako i samo ako postoje vektori $s^* \in R^n$ i $y^* \in R^m$ tako da važi:*

$$\begin{aligned} A^T y^* + s^* &= c, && (1.1.4a)\\ Ax^* &= b, && (1.1.4b)\\ x_i^* s_i^* &= 0, i = 1, \ldots, n, && (1.1.4c)\\ (x^*, s^*) &\geq 0. && (1.1.4d) \end{aligned} \tag{1.1.4}$$

Ako tačka (x, y, s) zadovoljava uslove a, b i d iz prethodne teoreme, tada je $0 \leq s^T x = (c - A^T y)^T x = c^T x - y^T (Ax) = c^T x - b^T y$ odakle sledi (1.1.3). Uslov (1.1.4c) povlači da za svaki indeks $i = 1, \ldots, n$, jedna od vrednosti x_i^* ili s_i^* mora biti jednaka nuli. Taj uslov se naziva *uslov komplementarnosti*, jer povlači da se nenula komponente vektora x i s javljaju na komplementarnim koordinatama.

Važi i dualna teorema:

Teorema 1.1.2 *Vektor* $(y^*, s^*) \in R^m \times R^n$ *je rešenje za* (1.1.2) *ako i samo ako postoji vektor* $x^* \in R^n$ *tako da važe uslovi* (1.1.4).

Direktna posledica ovih teorema je da vektor (x^*, y^*, s^*) rešava sistem (1.1.4) ako i samo je x^* rešenje primalnog problema (1.1.1) i (y^*, s^*) je rešenje dualnog problema. Vektor (x^*, y^*, s^*) se naziva *primal-dual rešenje.* Sa Ω_P i Ω_D označimo skupove optimalnih rešenja primalnog i dualnog problema respektivno:

$$\begin{aligned} \Omega_P &= \{x^* | \, x^* \text{ je rešenje za (1.1.1)}\}, \\ \Omega_D &= \{(y^*, s^*) | \, (y^*, s^*) \text{ je rešenje za (1.1.2)}\}. \end{aligned}$$

Skup rešenja primal-dualnog problema je Dekartov proizvod

$$\Omega = \Omega_P \times \Omega_D = \{(x, y, s) | \text{ važi (1.1.4).}\}$$

Za svako rešenje (x^*, y^*, s^*), važi $x_j^* = 0$ i/ili $s_j^* = 0$ za svako $j = 1, \ldots, n$. Neka je

$$\begin{aligned} \mathcal{B} &= \{j \in \{1, \ldots, n\} \mid x_j^* \neq 0 \text{ za neko } x^* \in \Omega_P\}, \\ \mathcal{N} &= \{j \in \{1, \ldots, n\} \mid s_j^* \neq 0 \text{ za neko } (y^*, s^*) \in \Omega_D\} \end{aligned}$$

Jasno, $\mathcal{B} \cap \mathcal{N} = \emptyset$. Sledeći rezultat je poznat kao Goldman-Tuckerova teorema [36].

Teorema 1.1.3 *Postoji bar jedno primalno rešenje* $x^* \in \Omega_P$ *i jedno dualno rešenje* $(y^*, s^*) \in \Omega_D$ *tako da je* $x^* + s^* > 0$.

Primal-dual rešenje (x, y, s) sa osobinom $x + s > 0$ je *strogo komplementarno rešenje.*

Osnova primal-dual metoda je modifikacija Newtonove metode primenjena na prva tri uslova iz (1.1.4), pri čemu se traženi pravac i korak svake iteracije određuju tako da nejednakost $(x, s) \geq 0$ važi strogo u svakoj iteraciji. Ta stroga nejednakost upravo i dovodi do problema u pravljenju, analizi i primeni algoritama primal-dual metoda.

Uslove optimalnosti (1.1.4) razmatramo kao preslikavanje F iz R^{2n+m} u R^{2n+m}:

$$\begin{aligned} F(x, y, s) = & \begin{bmatrix} A^T y + s - c \\ Ax - b \\ XSe \end{bmatrix} = 0, && (1.1.5a) \\ & (x, s) \geq 0, && (1.1.5b) \end{aligned} \tag{1.1.5}$$

gde je

$$X = \text{diag}(x_1, \ldots, x_n), \quad S = \text{diag}(s_1, \ldots, s_n) \quad \text{i} \quad e = (1, \ldots, 1)^T.$$

Svi primal-dual metodi generišu iteracije (x^k, y^k, s^k) koje zadovoljavaju ograničenje (1.1.5b) striktno, tj. $x^k > 0$ i $s^k > 0$. Na osnovu te osobine se metode koje razmatramo i zovu *metode unutašnje tačke.* Većina metoda unutrašnje tačke zahteva da

iteracije budu *strogo dopustive*, tj. da (x^k, y^k, s^k) zadovoljava linearne jednačine iz primalnog i dualnog problema. Primal dual *dopustiv skup* $\mathcal{F}$ i *strogo dopustiv skup* $\mathcal{F}^0$ definisani su sa

$$\begin{aligned}\mathcal{F} = &\quad \{(x,y,s)|\, Ax = b, A^T y + s = c, (x,s) \geq 0\},\\ \mathcal{F}^0 = &\quad \{(x,y,s)|\, Ax = b, A^T y + s = c, (x,s) > 0\}.\end{aligned}$$

Rešavajući sistem Newtonovih linearnih jednačina

$$J(x,y,s)\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta s\end{bmatrix} = -F(x,y,s),$$

gde je J Jakobijan preslikavanja F, dobijamo traženi pravac $(\Delta x, \Delta y, \Delta s)$. Ako je $(x,y,s) \in \mathcal{F}^0$, Newtonove jednačine postaju

$$\begin{bmatrix}0 & A^T & I\\ A & 0 & 0\\ S & 0 & X\end{bmatrix}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta s\end{bmatrix} = \begin{bmatrix}0\\ 0\\ -XSe\end{bmatrix}. \tag{1.1.6}$$

Pun korak u tom pravcu obično nije dopustiv zbog uslova $(x,s) \geq 0$. Sledeću iteraciju određujemo sa

$$(x,y,s) + \alpha(\Delta x, \Delta y, \Delta s)$$

gde je $\alpha \in (0,1]$ određeno iz uslova pozitivnosti. Nažalost, vrednost parametra α je najčešće vrlo mala ($\alpha \ll 1$), tako da Newtonov pravac ne dozvoljava velike pomeraje ka rešenju. Primal-dual metode modifikuju Newtonovu metodu na dva načina:

1. Podešavaju traženi pravac ka unutrašnjosti nenegativnog kvadranta $(x,s) > 0$ tako je moguć duži korak duž dobijenog pravca pre nego što neka od komponenti (x,s) postane negativna.
2. Ne dozvoljavaju preveliko "približavanje" komponenti (x,s) ka granici nenegativnog kvadranta, jer se u tom slučaju dobija pravac duž koga je moguć samo mali pomeraj ka rešenju.

Centralna putanja

Centralna putanja $\mathcal{C}$ sastoji od strogo dopustivih tačaka i igra važnu ulogu u teoriji primal-dual algoritama. Karakteriše se KKT uslovima u kojima se umesto uslova komplementarnosti (1.1.4c) nameće uslov da proizvodi $x_i s_i$ imaju istu vrednost $\tau > 0$ za svako i, tj. $(x_\tau, y_\tau, s_\tau) \in \mathcal{C}$ ako je rešenje sistema:

$$\begin{aligned}A^T y + s &= c,\\ Ax &= b,\\ x_i s_i &= \tau, \quad i = 1, \ldots, n,\\ (x,s) &\geq 0.\end{aligned}$$

Analogna definicija putanje $\mathcal{C}$ data je sa

$$F(x_\tau, y_\tau, s_\tau) = \begin{bmatrix} 0 \\ 0 \\ \tau e \end{bmatrix}, \quad (x_\tau, s_\tau) > 0.$$

Pokazuje se da je (x_τ, y_τ, s_τ) definisano na jedinstven način za svako $\tau > 0$ ako i samo ako je $\mathcal{F}^0 \neq \emptyset$, tj. cela putnja $\mathcal{C}$ je dobro definisana. Za dokaz egzistencije centralne putanje koristi se sledeće tvrđenje:

Lema 1.1.1 *Pretpostavimo da je $\mathcal{F}^0 \neq \emptyset$. Tada je za svako $K \geq 0$ skup*

$$\{(x,s) | \, (x,y,s) \in \mathcal{F} \text{ za neko } y, \ i \ x^T s \leq K\}$$

ograničen.

Kada τ teži nuli, $\mathcal{C}$ konvergira ka primal-dual rešenju linearnog programa i to preko tačaka u kojima su proizvodi $x_i s_i$ strogo pozitivani i teže ka nuli istom brzinom. Većina primal-dual metoda umesto uobičajenog Newtonovog koraka za F uzima Newtonov korak ka tačkama na $\mathcal{C}$. To se postiže uvođenjem *parametra centriranja* $\sigma \in [0,1]$ i *mere dualnosti* μ definisane sa

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i s_i = \frac{1}{n} x^T s,$$

koja određuje srednju vrednost proizvoda $x_i s_i$. Jednačine za određivanje koraka iteracije su

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -XSe + \sigma \mu e \end{bmatrix}. \tag{1.1.7}$$

Sada je $(\Delta x, \Delta y, \Delta s)$ Newtonov korak ka tački $(x_{\sigma\mu}, y_{\sigma\mu}, s_{\sigma\mu}) \in \mathcal{C}$, u kojoj su proizvodi $x_i s_i$ jednaki $\sigma\mu$.

Za $\sigma = 1$, jednačine (1.1.7) definišu *centralni pravac,* Newtonov korak prema tački $(x_\mu, y_\mu, s_\mu) \in \mathcal{C}$, u kojoj su proizvodi $x_i s_i$ jednaki μ. Centralni pravac je obično jako usmeren ka unutrašnjosti nenegativnog kvadranta i dovodi do malog smanjenja parametra μ. Međutim, u sledećem koraku je tada moguće uzeti relativno duži korak iteracije. Druga ekstremna vrednost za $\sigma = 0$ daje standardni Newtonov korak (1.1.6) koji se naziva *afini pravac.* Većina algoritama koristi srednju vrednost parametra $\sigma \in (0,1)$, tako da istovremeno redukuje vrednost mere dualnosti μ i bira pravac ka centralnoj putanji.

1.2 Osnovna šema primal-dual algoritama

Osnovni primal-dual algoritam je opisan u [112].

Osnovni primal-dual algoritam

Data je tačka $(x^0, y^0, s^0) \in \mathcal{F}^0$

for $k = 0, 1, 2, \ldots$

solve

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix},$$

gde je $\sigma_k \in [0, 1]$ i $\mu_k = (x^k)^T s^k / n$;

postaviti

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + \alpha_k (\Delta x^k, \Delta y^k, \Delta s^k),$$

birajući α_k tako da je $(x^{k+1}, s^{k+1}) > 0$.

end(for).

Većina algoritama za početnu tačku zahteva strogo dopustivu tačku $(x^0, y^0, s^0) \in \mathcal{F}^0$ koja je takođe i u nekoj okolini centralne putanje $\mathcal{C}$. Međutim, često nije lako odrediti takvu tačku. Postoje i problemi kod kojih takva tačka uopšte ne postoji. U problemu

$$\min 2x_1 + x_2 \quad \text{u odnosu na } x_1 + x_2 + x_3 = 5,\ x_1 + x_3 = 5,\ x \geq 0,$$

je primal dopustiv skup $\{(\beta, 0, 5 - \beta) |\, \beta \in [0, 5]\}$. Kako je $x_2 = 0$, strogo dopustiv skup $\mathcal{F}^0$ je prazan. U opštem slučaju, linearni problemi koji se dobijaju svođenjem opšteg problema na standardni oblik često nemaju strogo dopustive tačke. Jedan od načina za prevazilaženje te teškoće je potapanje datog linearnog problema u nešto veći problem za koji je skup $\mathcal{F}^0$ neprazan.

Poznat je algoritam koji ne zahteva da početna tačka bude strogo dopustiva već se samo traži da komponente x i s budu strogo pozitivne. U opštem slučaju sve iteracije (x^k, y^k, s^k) generisane algoritmom su nedopustive, ali je granična tačka dopustiva i optimalna. Algoritam je moguće primeniti i kada je $\mathcal{F}^0 = \emptyset$. Ovakav pristup su dali Kojima [58], Zhang [122], Potra [79] i Wright [112].

Reziduume sistema jednačina

$$r_b - Ax - b, \quad r_c = A^T y + s - c$$

za $k-$tu iteraciju označimo sa r_b^k i r_c^k. Okolinu $\mathcal{N}_{-\infty}(\gamma)$ koja sadrži i nedopustive tačke definišmo sa

$$\begin{aligned} \mathcal{N}_{-\infty}(\gamma, \beta) = \quad & \{(x, y, s) |\, |(r_b, r_c)| \leq [|(r_b^0, r_c^0)|/\mu_0]\beta\mu, \\ & (x, s) > 0, x_i s_i \geq \gamma\mu,\ i = 1, \ldots, n\} \end{aligned}$$

gde su $\gamma \in (0, 1)$ i $\beta \geq 1$ dati parametri i vrednosti (r_b^0, r_c^0) i μ_0 izračunate u početnoj tački (x^0, y^0, s^0). Primetimo da je neophodan uslov

$$\beta \geq 1$$

da bi osigurali da početna tačka (x^0, y^0, s^0) pripada $\mathcal{N}_{-\infty}(\gamma, \beta)$.

Sada navodimo opšti primal-dual algoritam za nedopustivu tačku iz [112].

Algoritam NPD

Dato je $\gamma, \beta, \sigma_{\min}, \sigma_{\max}$, pri čemu je $\gamma \in (0,1)$, $\beta \geq 1$, i $0 < \sigma_{\min} < \sigma_{\max} \leq 0.5$;
izabrati (x^0, y^0, s^0) tako da je $(x^0, s^0) > 0$;

for $k = 0, 1, 2, \ldots$

izabrati $\sigma_k \in [\sigma_{\min}, \sigma_{\max}]$ i rešiti

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} -r_c^k \\ -r_b^k \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix},$$

izabrati za α_k najveću vrednost od $\alpha \in [0,1]$, tako da je

$$(x^k(\alpha), y^k(\alpha), s^k(\alpha)) \in \mathcal{N}_{-\infty}(\gamma, \beta)$$

i da važi sledeći uslov Armija:

$$\mu_k(\alpha) \leq (1 - 0.01\alpha)\mu_k.$$

postaviti

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k(\alpha_k), y^k(\alpha_k), s^k(\alpha_k));$$

end (for).

Neka je ν_k definisano sa

$$\nu_k = \prod_{j=0}^{k-1}(1 - \alpha_j) \quad (\nu_0 = 1).$$

Kako su prve dve komponente funkcije F linearne, sledi

$$(r_b^k, r_c^k) = (1 - \alpha_{k-1})(r_b^{k-1}, r_c^{k-1}) = \cdots = \nu_k(r_b^0, r_c^0),$$

i kako je $(x^k, y^k, s^k) \in \mathcal{N}_{-\infty}(\gamma, \beta)$, dobijamo

$$\nu_k |(r_b^0, r_c^0)|/\mu_k = |(r_b^k, r_c^k)|/\mu_k \leq \beta |(r_b^0, r_c^0)|/\mu_0.$$

Ako pretpostavimo da je $(r_b^0, r_c^0) \neq 0$, tada sledi

$$\nu_k \leq \beta \mu_k / \mu_0.$$

U dopustivom slučaju je $(r_b^0, r_c^0) = 0$ i sve iteracije (x^k, y^k, s^k) su strogo dopustive i tada se Algoritam NPD svodi algoritam za dopustivu tačku. U nastavku pretpostavljamo da je početna tačka (x^0, y^0, s^0) nedopustiva.

Sledeća teorema daje opšti rezultat o kompleksnosti metoda koji slede centralnu putanju.

Teorema 1.2.1 *Neka je* $\varepsilon \in (0,1)$ *dato. Pretpostavimo da algoritam generiše niz iteracija koje zadovoljavaju*

$$\mu_{k+1} \leq \left(1 - \frac{\delta}{n^{\omega}}\right)\mu_k, \quad k = 0, 1, 2, \ldots,$$

za neke pozitivne konstante δ *i* ω*. Pretpostavimo da početna tačka* (x^0, y^0, s^0) *zadovoljava*

$$\mu_0 \leq 1/\kappa$$

za meku pozitivnu konstantu κ*. Tada postoji indeks* K *pri čemu je*

$$K = O(n^{\omega} |\log \varepsilon|)$$

tako da važi

$$\mu_k \leq \varepsilon \quad \textit{za svako } k \geq K.$$

Dakle, ako smanjenje μ u svakoj iteraciji zavisi od n na neki način i ako početna mera dualnosti nije previše velika, algoritam ima polinomijalnu kompleksnost.

1.3 Potencijalno-redukcioni metodi unutrašnje tačke

Karmarkarov algoritam [51] je koristio logaritamsku potencijalnu funkciju oblika

$$\rho \log(c^T x - Z) - \sum_{i=1}^{n} \log x_i,$$

gde je $\rho = n + 1$ i Z donja granica optimalne vrednosti ciljne funkcije. Na razvoj potencijal-redukcionih algoritama posle 1988. godine, važan uticaj ima Tanabe-Todd-Ye [102, 104] potencijalna funkcija definisana sa

$$\Phi_\rho(x, s) = \rho \log x^T s - \sum_{i=1}^{n} \log x_i s_i$$

za neki parametar $\rho > n$. Opisaćemo algoritam koji su dali Kojima, Mizuno i Yoshise [60]. Taj algoritam ima najbolju poznatu granicu kompleksnosti jer zahteva $O(\sqrt{n}|\log \varepsilon|)$ iteracija da bi smanjio razliku između primalnog i dualnog rešenja ispod datog praga $\varepsilon > 0$. Algoritam PR je specijalan slučaj osnovnog primal-dual algoritma. Karakteriše ga izbor konstantne vrednosti $\sigma_k = n/\rho$ za parametar centriranja. Dužina koraka α_k se bira tako da minimizira $\Phi_\rho(\cdot)$ duž dobijenog pravca. Gornja granica dužine koraka od tačke $(x, y, s) \in \mathcal{F}^0$ duž izračunatog pravca $(\Delta x, \Delta y, \Delta s)$ je data sa

$$\alpha_{\max} = \sup\{\alpha \in [0,1] | \, (x, s) + \alpha(\Delta x, \Delta s) \geq 0\}.$$

Algoritam PR
Dato je $\rho > n$ **i tačka** $(x^0, y^0, s^0) \in \mathcal{F}^0$

for $k = 0, 1, 2, \ldots$

staviti $\sigma_k = n/\rho$ i rešiti

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix}, \tag{1.3.1}$$

gde je $\mu_k = (x^k)^T s^k/n$, da bi dobili $(\Delta x^k, \Delta y^k, \Delta s^k)$.

Izračunati $\alpha_{\max}$ iz (1.3.1) i izabrati dužinu koraka α_k iz

$$\alpha_k = \arg \min_{\alpha \in (0, \alpha_{max})} \Phi_\rho(x^k + \alpha \Delta x^k, s^k + \alpha \Delta s^k);$$

postaviti

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + \alpha_k(\Delta x^k, \Delta y^k, \Delta s^k);$$

end (for).

Prvo ćemo pokazati da kada $\Phi_\rho(x^k, s^k) \to -\infty$ tada $\mu_k \to 0$.

Lema 1.3.1 *Važe sledeće nejednakosti:*

(i) $\log(1 + t) \leq t$ *za svako* $t > -1$ *pri čemu jednakost važi samo za* $t = 0$.

(ii) Za svako $t \in R^n$ *za koje je* $\|t\|_\infty \leq \tau < 1$, *važi*

$$-\sum_{i=1}^{n} \log(1 + t_i) \leq -e^t t + \frac{\|t\|^2}{2(1 - \tau)}.$$

Predstavimo potencijalnu funkciju u obliku

$$\begin{aligned} \Phi_\rho(x, s) &= (\rho - n) \log x^T s + \Phi_n(x, s) \\ &= (\rho - n) \log x^T s - \sum_{i=1}^{n} \log \frac{x_i s_i}{x^T s/n} + n \log n. \end{aligned}$$

Sledeća lema pokazuje da je funkcija Φ_n ograničena odozdo.

Lema 1.3.2 *Za* $(x, s) > 0$ *važi*

$$\Phi_n(x, s) \geq n \log n,$$

pri čemu jednakost važi ako i samo ako je $XSe = (x^T s/n)e = \mu e$.

U sledećoj lemi se dokazuje ključni odnos između Φ_ρ i μ.

Lema 1.3.3 *(i) Funkcija* Φ_ρ *je neograničena odozdo na svom domenu.*

(ii) Za svako $(x, y, s) \in \mathcal{F}^0$ *važi*

$$\mu \leq \exp(\Phi_\rho(x, s)/(\rho - n)),$$

gde je $\mu = x^T s/n$.

Iz Leme 1.3.3(ii) sledi da ako generišemo niz iteracija $(x^k, y^k, s^k) \in \mathcal{F}^0$ za koje $\Phi_\rho(x^k, s^k) \to -\infty$, tada $\mu_k \to 0$. Pokazaćemo u nastavku da se Φ_ρ smanjuje za fiksiranu vrednost $\delta > 0$, nezavisno od n, u svakom koraku Algoritma PR, tj.

$$\Phi_\rho(x^{k+1}, s^{k+1}) \leq \Phi_\rho(x^k, s^k) - \delta \quad \text{za svako } k = 0, 1, 2, \ldots \tag{1.3.2}$$

Sledeća teorema daje opšti rezultat za kompleksnost algoritama koji postižu takvo smanjenje potencijalne funkcije.

Teorema 1.3.1 *Neka je data tačka $(x^0, y^0, s^0) \in \mathcal{F}^0$ i pretpostavimo da algoritam generiše niz $(x^k, y^k, s^k) \in \mathcal{F}^0$ koji zadovoljava* (1.3.2) *za neko $\delta > 0$. Tada za svako $\varepsilon \in (0, 1)$ postoji broj K definisan sa*

$$K = \left[\frac{\Phi_\rho(x^0, s^0)}{\delta} + \frac{\rho - n}{\delta} |\log \varepsilon| \right]$$

tako da je

$$\mu_k \leq \varepsilon \quad \text{za svako } k \geq K.$$

U nastavku pokazujemo egzistenciju kvadratne funkcije $q(\alpha)$ koja je gornje ograničenje za Φ_ρ i zavisi od dužine koraka α duž traženog pravca. Za bilo koji pravac $(\Delta x, \Delta y, \Delta s)$ važi

$$A\Delta x = 0, \quad A^T \Delta y + \Delta s = 0. \tag{1.3.3}$$

Odatle sledi

$$\Delta s^T \Delta x = -\Delta x^T A^T \Delta y = -(A\Delta x)^T \Delta y = 0. \tag{1.3.4}$$

Pretpostavili smo da je $\alpha \in (0, \alpha_{\max})$ ali kvadratna procena važi na kraćem intervalu $(0, \alpha_\tau]$ definisanom sa

$$\alpha_\tau \max(\|X^{-1}\Delta x\|_\infty, \|S^{-1}\Delta s\|_\infty) = \tau,$$

gde je $\tau \in (0, 1)$ konstanta koju definišemo kasnije. Iz (1.3.4) sledi

$$\begin{aligned}
&\Phi_\rho(x + \alpha\Delta x, s + \alpha\Delta s) - \Phi_\rho(x, s) \\
&= \rho \log \frac{(x + \alpha\Delta x)^T (s + \alpha\Delta s)}{x^T s} - \sum_{i=1}^n \log \frac{x_i + \alpha\Delta x_i}{x_i} - \sum_{i=1}^n \log \frac{s_i + \alpha\Delta s_i}{s_i} \\
&= \rho \log \left[1 - \alpha \frac{s^T \Delta x + x^T \Delta s}{x^T s} \right] - \sum_{i=1}^n \log \left[1 + \alpha \frac{\Delta x_i}{x_i} \right] - \sum_{i=1}^n \log \left[1 + \alpha \frac{\Delta s_i}{s_i} \right].
\end{aligned}$$

Primenom Leme 1.3.1 dobijamo

$$\begin{aligned}
&\Phi_\rho(x + \alpha\Delta x, s + \alpha\Delta s) \leq \Phi_\rho(x, s) + \rho\alpha \frac{s^T \Delta x + x^T \Delta s}{x^T s} \\
&-\alpha e^T (X^{-1}\Delta x + S^{-1}\Delta s) + \frac{\alpha^2}{2(1 - \tau)} (\|X^{-1}\Delta x\|^2 + \|S^{-1}\Delta s\|^2) \\
&= \Phi_\rho(x, s) + \alpha g_1 + \frac{1}{2}\alpha^2 g_2
\end{aligned} \tag{1.3.5}$$

za svako $\alpha \in (0, \alpha_\tau]$, gde je

$$\begin{aligned} g_1 &= \rho\frac{s^T\Delta x + x^T\Delta s}{x^T s} - e^T(X^{-1}\Delta x + S^{-1}\Delta s), \\ g_2 &= \frac{1}{(1-\tau)}(\|X^{-1}\Delta x\|^2 + \|S^{-1}\Delta s\|^2). \end{aligned}$$

Definišemo kvadratnu aproksimaciju sa

$$q(\alpha) = \Phi_\rho(x,s) + \alpha g_1 + \frac{1}{2}\alpha^2 g_2,$$

i iz (1.3.5) sledi

$$\Phi_\rho(x + \alpha\Delta x, s + \alpha\Delta s) \le q(\alpha) \quad \forall \alpha \in (0, \alpha_\tau].$$

Posmatramo sada pravac $(\Delta x, \Delta y, \Delta s)$ koji zadovoljava jednačinu

$$S\Delta x + X\Delta s = -XSe + \frac{n}{\rho}\mu e \tag{1.3.6}$$

kao i uslove (1.3.3). Za dato $(x, y, s) \in \mathcal{F}^0$ uvedimo oznake

$$\begin{aligned} V &= (XS)^{1/2}, \quad \upsilon = Ve = [(x_i s_i)^{1/2}]_{i=1}^n, \\ \upsilon_{\min} &= \min_i \upsilon_i, \quad r = -\upsilon + \frac{n}{\rho}\mu V^{-1}e, \end{aligned}$$

i neka je $D = X^{1/2}S^{-1/2}$. Važe sledeće jednakosti

$$\|\upsilon\|^2 = x^T s = n\mu, \quad X = VD, \quad S = VD^{-1}. \tag{1.3.7}$$

Uslove (1.3.6) možemo sada posmatrati u nekom od oblika

$$\begin{aligned} S\Delta x + X\Delta s &= Vr, \\ D^{-1}\Delta x + D\Delta s &= r, \\ X^{-1}\Delta x + S^{-1}\Delta s &= V^{-1}r. \end{aligned}$$

Iz (1.3.4) sledi

$$\|r\|^2 = \|D^{-1}\Delta x\|^2 + \|D\Delta s\|^2, \tag{1.3.8}$$

i stoga je

$$\|D^{-1}\Delta x\| \le \|r\|, \quad \|D\Delta s\| \le \|r\|.$$

Iz (1.3.7) i (1.3.8) sledi

$$\|X^{-1}\Delta x\|^2 + \|S^{-1}\Delta s\|^2 \le \frac{1}{\upsilon_{\min}^2}\|r\|^2,$$

odakle je

$$g_2 \leq \frac{1}{1-\tau}\frac{\|r\|^2}{\upsilon_{\min}^2}\|r\|^2.$$

Za koeficijent g_1 važi

$$g_1 = -\frac{\rho}{n\mu}\|r\|^2.$$

Ako je $r \neq 0$, tada je $g_1 < 0$ i kako je $g_2 > 0$, to je grafik funkcije q obrnuta parabola. Pokazaćemo da je njen minimum u intervalu $(0, \alpha_\tau]$.

Lema 1.3.4 *Za svako* $(x, y, s) \in \mathcal{F}^0$ *i za* $\rho > n + \sqrt{n}$, *važi*

$$\|r\|| \geq \frac{\sqrt{3}}{2\upsilon_{\min}}\frac{n\mu}{\rho}.$$

Odredićemo sada vrednost parametra α tako da se postiže konstantno smanjenje $q(\alpha)$ i Φ_ρ u svakoj iteraciji za fiksiranu vrenost $\tau = 0.5$ zbog jednostavnosti. Analogni rezultati se mogu pokazati za svako $\tau \in (0, 1)$.

Teorema 1.3.2 *Neka je* $\tau = 0.5$ *i neka je*

$$\bar{\alpha} = \frac{\upsilon_{\min}}{2\|r\|}.$$

Tada je $q(0) - q(\bar{\alpha}) \leq -0.15$, *i stoga* (1.3.2) *važi za* $\delta = 0.15$.

Napomenimo da se u praksi obično dobijaju veće vrednosti od ove donje granice.

Na osnovu 1.3.1 sledi rezultat o kompleksnosti algoritma.

Posledica 1.3.1 *Neka je* $\rho \geq n + \sqrt{n}$ *i* $\varepsilon > 0$, *i pretpostavimo da početna tačka* $(x^0, y^0, s^0) \in \mathcal{F}^0$ *zadovoljava*

$$\Phi_\rho(x^0, s^0) \leq \|(\rho - n)|\log\varepsilon|$$

za neko $\| > 0$ *nezavisno od n. Ako je K definisano sa*

$$K = \left[\frac{\kappa + 1}{0.15}(\rho - n)|\log\varepsilon|\right] = O((\rho - n)|\log\varepsilon|),$$

tada je

$$(x^k, y^k, s^k) \in \mathcal{F}^0, \quad \mu_k \leq \varepsilon,$$

za svako $k \geq K$.

Ovaj rezultat o kompleksnosti važi za svako ρ koje zadovoljava $\rho \geq n + \sqrt{n}$. Ako fiksiramo $\rho = n + \sqrt{n}$ dobija se najbolja procena kompleksnosti algoritma koja je jednaka $O(\sqrt{n}|\log\varepsilon|)$. Međutim, tada je

$$\sigma = \frac{n}{\rho} = \frac{n}{n + \sqrt{n}} \approx 1 - \frac{1}{\sqrt{n}} \quad \text{za veliko } n,$$

tj. parametar centriranja σ je blizu jedinice tako da nije moguć veliki korak u iteraciji. Manje vrednosti za σ se dobijaju birajući veće vrednosti za $\rho-$ obično je $\rho = 10n$ ili $\rho = n + n^{1.5}$. Time se kompleksnost povećava na $O(n|\log\varepsilon|)$ i $O(n^{1.5}|\log\varepsilon|)$ respektivno, ali se obično dobiju bolje numeričke performanse. Dalje, kako je Φ_ρ nelinearna funkcija višeg reda, nije pogodno izračunati tačnu vrednost njenog minimuma. Sa druge strane, ako je α i poznato, dobija se isuviše mali korak koji ne daje razumne numeričke performanse. U praksi su se bolje pokazale vrednosti $0.99\alpha_{\max}$ i $0.95\alpha_{\max}$.

1.4 Algoritmi koji slede centralnu putanju

Centralnu putanju $\mathcal{C}$ krakterišu KKT uslovi u kojima se umesto uslova komplementarnosti (1.1.4c) nameće uslov da proizvodi $x_i s_i$ imaju istu vrednost $\tau > 0$ za svako i, tj. $(x_\tau, y_\tau, s_\tau) \in \mathcal{C}$ ako je rešenje sistema:

$$\begin{aligned} A^T y + s &= c, && (1.4.1a)\\ Ax &= b, && (1.4.1b)\\ (x,s) &\geq 0. && (1.4.1c)\\ x_i s_i &= \tau, \quad i = 1,\ldots,n, && (1.4.1d) \end{aligned} \tag{1.4.1}$$

Pokazali smo da ovaj sistem ima jedinstveno rešenje (x_τ, y_τ, s_τ) za svako $\tau > 0$ kada je problem dopustiv (iako KKT uslovi mogu za $\tau = 0$ imati i višestruka rešenja). Algoritmi koji slede putanju podešavaju iteraciju duž $\mathcal{C}$ u pravcu opadanja τ ka skupu rešenja Ω. Oni generišu strogo dopustive iteracije (x^k, y^k, s^k) koje zadovoljavaju prva tri KKT uslova i odstupaju od centralne putanje $\mathcal{C}$ samo zato što proizvodi $x_i s_i$ u opštem slučaju nisu identični, tako da uslov (1.4.1d) nije zadovoljen u potpunosti. Odstupanje se meri upoređivanjem proizvoda sa njihovom srednjom vrednošću $\mu = x^T s/n$, koristeći, recimo, normu definisanu sa

$$\frac{1}{\mu}\|XSe - \mu e\|.$$

U literaturi se koriste i 2-norma i ∞-norma u ovoj definiciji. Za obe norme iz $(1/\mu)\|XSe - \mu e\| < 1$ sledi da su x i s strogo pozitivni. Podsetimo da je okolina $\mathcal{N}_2(\theta)$ putanje $\mathcal{C}$ definisana sa

$$\mathcal{N}_2(\theta) = \{(x,y,s) \in \mathcal{F}^0 |\, \|XSe - \mu e\|_2 \leq \theta\mu\}$$

pri čemu je odstupanje od $\mathcal{C}$ manje od $\theta \in (0,1)$, i okolina $\mathcal{N}_{-\infty}(\gamma)$ definisana sa

$$\mathcal{N}_{-\infty}(\gamma) = \{(x,y,s) \in \mathcal{F}^0 |\, x_i s_i \geq \gamma\mu \forall i = 1,\ldots,n\}$$

za neko $\gamma \in (0,1)$.

Algoritam kratkog koraka po putanji

Ovaj najjednostavniji algoritam su uveli Kojima, Mizuno i Yoshise [72] i Monteiro i Adler [73]. Početna tačka je u $\mathcal{N}_2(\theta)$ i koriste se konstantne vrednosti $\alpha_k = 1$ i $\sigma_k = \sigma$.

Algoritam KKP
Dato je $\theta = 0.4$, $\sigma = 1 - 0.4/\sqrt{n}$, i $(x^0, y^0, s^0) \in \mathcal{N}_2(\theta)$;
for $k = 0, 1, 2, \ldots$
staviti $\sigma_k = \sigma$ i rešiti

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix},$$

da bi dobili $(\Delta x^k, \Delta y^k, \Delta s^k)$;
postaviti

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + (\Delta x^k, \Delta y^k, \Delta s^k);$$

end (for).

Uvedimo oznake

$$\begin{aligned} (x(\alpha), y(\alpha), s(\alpha)) &= (x, y, s) + (\alpha)(\Delta x, \Delta y, \Delta s), \quad (1.4.5a) \\ \mu(\alpha) &= x(\alpha)^T s(\alpha)/n. \quad (1.4.5b) \end{aligned} \tag{1.4.2}$$

Lema 1.4.1 *Neka je korak* $(\Delta x, \Delta y, \Delta s)$ *definisan sa* (1.1.7). *Tada je*

$$\Delta x^T \Delta s = 0$$

i

$$\mu(\alpha) = (1 - \alpha(1 - \sigma))\mu.$$

Lema je iskazana u opštem slučaju jer se primenjuje na sve algoritme koji traženi pravac dobijaju iz sistema (1.1.7). Za konkretne vrednosti u algoritmu je

$$\mu_{k+1} = \sigma\mu = \left(1 - \frac{0.4}{\sqrt{n}}\right)\mu_k, \quad k = 0, 1, \ldots, \tag{1.4.3}$$

tako da je konvergencija linearna. Polinomijalna kompleksnost je direktna posledica (1.4.3) i Teoreme 1.2.1

Teorema 1.4.1 *Neka je* $\varepsilon > 0$ *i neka je* $(x^0, y^0, s^0) \in \mathcal{N}_2(0.4)$ *početna tačka Algoritma KKP za koju je* $\mu_0 \leq 1/\varepsilon^\kappa$ *za neku pozitivnu konstantu* κ. *Tada postoji indeks* K *pri čemu je* $K = O(\sqrt{n} \log 1/\varepsilon)$ *tako da je*

$$\mu_k \leq \varepsilon \quad \text{za svako } k \geq K.$$

Dokazaćemo sada da sve iteracije ostaju u istoj okolini.

Lema 1.4.2 *Neka su* u *i* v *vektori u* R^n *za koje važi* $u^T v \geq 0$. *Tada je*

$$\|UVe\| \leq 2^{-3/2}\|u + v\|^2,$$

gde je $U = diag(u_1, \ldots, u_n)$ *i* $V = diag(v_1, \ldots, v_n)$

U Lemi 1.4.1 je pokazano da je $\delta x^T \delta s = \sum \delta x_i \delta s_i = 0$, ali to ne znači da su svi sabirci jednaki nuli. U sledećoj lemi dajemo ograničenje za vektor proizvoda.

Lema 1.4.3 *Ako je* $(x, y, s) \in \mathcal{N}_2(\theta)$ *tada je*

$$\|\Delta X \Delta S e\| \leq \frac{\theta^2 + n(1-\sigma)^2}{2^{3/2}(1-\theta)}\mu.$$

Iz Leme 1.4.1 sledi da μ opada linearno kada se krećemo duž pravca $(\Delta x, \Delta y, \Delta s)$. Sledeća posledica Leme 1.4.3 pokazuje koliko tačka $(x(\alpha), y(\alpha), s(\alpha))$ odstupa od centralne putanje.

Lema 1.4.4 *Neka je* $(x, y, s) \in \mathcal{N}_2(\theta)$. *Tada je*

$$\begin{aligned}
&\|X(\alpha)S(\alpha)e - \mu(\alpha)e\| \\
&\leq |1-\alpha|\|XSe - \mu e\| + \alpha^2 \|\Delta X \Delta S e\| \\
&\leq |1-\alpha|\theta\mu + \alpha^2 \left[\frac{\theta^2 + n(1-\sigma)^2}{2^{3/2}(1-\theta)}\right]\mu.
\end{aligned}$$

Teorema 1.4.2 definiše odnos između θ i σ i pokazuje da i u slučaju punog koraka $\alpha = 1$ duž traženog pravca nova iteracija ostaje u okolini $\mathcal{N}_2(\theta)$.

Teorema 1.4.2 *Neka su parametri* $\theta \in (0,1)$ *i* $\sigma \in (0,1)$ *izabrani tako da zadovoljavaju jednakost*

$$\frac{\theta^2 + n(1-\sigma)^2}{2^{3/2}(1-\theta)} \leq \sigma\theta. \tag{1.4.4}$$

Tada za $(x, y, s) \in \mathcal{N}_2(\theta)$ *važi*

$$(x(\alpha), y(\alpha), s(\alpha)) \in \mathcal{N}_2(\theta)$$

za svako $\alpha \in [0, 1]$.

Lako se proverava da u Algoritmu KKP konkretan izbor parametara $\theta = 0.4$ i $\sigma = 1 - 0.4/\sqrt{n}$ zadovoljava uslove (1.4.4).

Prediktor-korektor metod

U Algoritmu KKP se σ bira strogo između 0 i 1. Time se u jednom koraku popravlja centralnost i smanjuje mera dualnosti μ. Prediktor-korektor metod, Algoritam PK, naizmenično bira dva tipa koraka

- *prediktor* korak sa $\sigma_k = 0$ koji smanjuje μ,
- *korektor* korak $\sigma_k = 1$ koji popravlja centralnost.

Razne varijante ovog algoritma su razmatrali Monteiro i Adler [73], Sonnevend, Stoer i Zhao [85], [86]. U nastavku sledimo jednostavan oblik algoritma koji su dali Mizuno, Todd i Ye [72]. Ovaj algoritam se ponekad naziva "Mizuno-Todd-Ye prediktor-korektor" da bi se razlikovao od prediktor-korektor algoritma Mehrotre koji se zasniva na drugim principima.

Algoritam PK karakteriše par $\mathcal{N}_2(\theta)$ okolina pri čemu je jedna unutar druge. Parne iteracije ostaju u unutrašnjoj okolini a neparne mogu biti i u spoljnoj okolini. Radi jednostavnosti, definisaćemo da je unutrašnja okolina $\mathcal{N}_2(0.25)$ i spoljna okolina $\mathcal{N}_2(0.5)$. Moguć je i drugi izbor parametara pod uslovom da okoline zadovoljavaju određene uslove.

Algoritam PK
Dato je $(x^0, y^0, s^0) \in \mathcal{N}_2(0.25)$;
for $k = 0, 1, 2, \ldots$
if k parno
(*prediktor korak*)
staviti $\sigma_k = 0$ i rešiti

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix},$$

da bi dobili $(\Delta x^k, \Delta y^k, \Delta s^k)$;
izabrati α_k kao najveću vrednost $\alpha \in [0, 1]$ za koju važi

$$(x^k(\alpha), y^k(\alpha), s^k(\alpha)) \in \mathcal{N}_2(0.5)$$

postaviti
$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k(\alpha_k), y^k(\alpha_k), s^k(\alpha_k))$;
else k neparno
(*korektor korak*)
rešiti prethodni sistem za $\sigma_k = 1$ da bi dobili $(\Delta x^k, \Delta y^k, \Delta s^k)$;
postaviti

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + (\Delta x^k, \Delta y^k, \Delta s^k);$$

end (if)
end (for).

Sledeća lema daje donju granicu za dužinu koraka i procenjuje smanjenje parametra dualnosti μ.

Lema 1.4.5 *Neka je* $(x, y, s) \in \mathcal{N}_2(0.25)$, *i neka je* $(\Delta x, \Delta y, \Delta s)$ *korak izračunat iz* (1.1.7) *za* $\sigma = 0$. *Tada je* $(x(\alpha), y(\alpha), s(\alpha)) \in \mathcal{N}_2(0.25)$ *za svako* $\alpha \in [0, \bar{\alpha}]$, *gde je*

$$\bar{\alpha} = \min\left(\frac{1}{2}, \left(\frac{\mu}{8\|\Delta X \Delta S e\|}\right)^{1/2}\right). \tag{1.4.5}$$

Dakle, prediktor korak je dužine minimum $\bar{\alpha}$, i nova vrednost mere dualnosti μ je najviše $(1-\bar{\alpha})\mu$.

Iz Leme 1.4.5 za $\theta = 0.25$ i $\sigma = 0$ sledi

$$\frac{\mu}{8\|\Delta X \Delta S e\|} \geq \frac{0.16}{n}$$

i iz (1.4.5) sledi procena

$$\bar{\alpha} \geq \min\left(\frac{1}{2}, \left(\frac{0.16}{n}\right)^{1/2}\right) = \frac{0.4}{\sqrt{n}}.$$

Odatle je za parne indekse

$$\mu_{k+1} \leq \left(1 = \frac{0.4}{\sqrt{n}}\right)\mu_k, \quad k = 0, 2, 4, \ldots \tag{1.4.6}$$

Sledeća lema pokazuje da korektor korak bilo koju tačku iz $\mathcal{N}_2(0.5)$ vraća u unutrašnju okolinu $\mathcal{N}_2(0.25)$ ne menjajući pri tome vrednost parametra μ.

Lema 1.4.6 *Neka je $(x, y, s) \in \mathcal{N}_2(0.5)$ i neka je $(\Delta x, \Delta y, \Delta s)$ korak izračunat iz (1.1.7) za $\sigma = 1$. Tada je*

$$(x(1), y(1), s(1)) \in \mathcal{N}_2(0.25), \quad \mu(1) = \mu.$$

Iako korektor korak ne menja vrednost μ zbog (1.4.6) se dobija polinomijalna kompleksnost kao i kod Algoritma KKP.

Teorema 1.4.3 *Neka je $\varepsilon > 0$ i neka je $(x^0, y^0, s^0) \in \mathcal{N}_2(0.25)$ početna tačka Algoritma PK za koju je $\mu_0 \leq 1/\varepsilon^\kappa$ za neku pozitivnu konstantu κ. Tada postoji indeks K pri čemu je $K = O(\sqrt{n}\log 1/\varepsilon)$ tako da je*

$$\mu_k \leq \varepsilon \quad \text{za svako } k \geq K.$$

Algoritam PK je poboljšanje Algoritma KKP jer u krajnjim iteracijama prediktor pravac postaje sve bolji, tako da je moguće uzeti dužinu koraka blizu 1. Ustvari, pokazuje se da je konvergencija mere dualnosti μ_k ka nuli superlinearna.

Algoritam dugog koraka po putanji

Algoritam DKP je u sličan prvom polinomijalnom primal-dual algoritmu koji su dali Kojima, Mizuno i Yoshise [59]. Praktično je primenljiviji od Algoritama KKP i PK ali mu je polinomijalnost lošija. Algoritam DKP generiše niz iteracija u okolini $\mathcal{N}_{-\infty}(\gamma)$, koja za male vrednosti γ zahvata veliki deo skupa strogo ostvarljivih tačaka $\mathcal{F}^0$.

Algoritam DKP
Dato je $\gamma, \sigma_{\min}, \sigma_{\max}$, pri čemu je $\gamma \in (0, 1)$, $0 < \sigma_{\min} < \sigma_{\max} < 1$,

i $(x^0, y^0, s^0) \in \mathcal{N}_{-\infty}(\gamma)$;
for $k = 0, 1, 2, \ldots$
izabrati $\sigma_k \in [\sigma_{\min}, \sigma_{\max}]$ i rešiti

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix},$$

da bi dobili $(\Delta x^k, \Delta y^k, \Delta s^k)$;
izabrati za α_k najveću vrednost od $\alpha \in [0, 1]$, tako da je

$$(x^k(\alpha), y^k(\alpha), s^k(\alpha)) \in \mathcal{N}_{-\infty}(\gamma);$$

postaviti

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k(\alpha_k), y^k(\alpha_k), s^k(\alpha_k));$$

end (for).

Sledeća dva tvrđenja daju donju granicu za α_k i odgovarajuću procenu za smanjenje parametra μ u svakoj iteraciji.

Lema 1.4.7 *Neka je* $(x, y, s) \in \mathcal{N}_{-\infty}(\gamma)$, *tada je*

$$\|\Delta X \Delta S e\| \leq 2^{-3/2} \left(1 + \frac{1}{\gamma}\right) n\mu.$$

Teorema 1.4.4 *Za date parametre* $\gamma, \sigma_{\min}$ *i* $\sigma_{\max}$ *u Algoritmu DKP postoji konstanta* δ *nezavisna od* n *tako da je*

$$\mu_{k+1} \leq \left(1 - \frac{\delta}{n}\right) \mu_k$$

za svako $k \geq 0$.

Polinomijalnost Algoritma DKP je direktna posledica prethodne teoreme.

Teorema 1.4.5 *Neka je* $\varepsilon > 0$ *i neka je* $\gamma \in (0, 1)$. *Neka je* $(x^0, y^0, s^0) \in \mathcal{N}_{-\infty}(\gamma)$ *početna tačka Algoritma DKP za koju je* $\mu_0 \leq 1/\varepsilon^\kappa$ *za neku pozitivnu konstantu* κ. *Tada postoji indeks* K *pri čemu je* $K = O(n \log 1/\varepsilon)$ *tako da je*

$$\mu_k \leq \varepsilon \quad \text{za svako } k \geq K.$$

Granične tačke iteracionog niza

Pokazali smo da u razmatranim algoritmima $\mu_k \to 0$, ne govoreći ništa o nizu $\{(x^k, y^k, s^k)\}$. Treba pokazati da niz $\{(x^k, s^k)\}$ ima graničnu vrednost. U slučaju da je $\mathcal{K}$ podniz za koji je

$$\lim_{k \in \mathcal{K}} (x^k, s^k) = (x^*, s^*),$$

tada je za svako $k \in \mathcal{K}$ ispunjeno

$$Ax^k = b, \quad c - s^k \in R(A^T), \quad (x^k, s^k) > 0.$$

Kako je $R(A^T)$ zatvoren i $\mu_k \to 0$, to za (x^*, s^*) važi

$$Ax^* = b, \quad c - s^* \in R(A^T), \quad (x^*, s^*) > 0, \quad (x^*)^T s^* = 0.$$

Dakle, $c - s^* = Ay^*$ za neko y^*. Upoređujući ove uslove sa KKT uslovima (1.4.1) zaključujemo $(x^*, y^*, s^*) \in \Omega$.

Güler i Ye [47] su dokazali da je niz $\{(x^k, s^k)\}$ ograničen i stoga ima bar jedan konvergentan podniz. Dalje, granična vrednost tih podnizova je strogo komplementarno rešenje.

Lema 1.4.8 *Neka je $\mu_0 > 0$ i $\gamma \in (0, 1)$. Tada za sve tačke (x, y, s) za koje važi*

$$(x, y, s) \in \mathcal{N}_{-\infty}(\gamma) \subset \mathcal{F}^0, \quad \mu \leq \mu_0$$

(gde je $\mu = x^T s/n$), postoje konstante C_0 i C_3 tako da je

$$\begin{aligned} \|(x, s)\| &\leq C_0 \\ 0 < x_i \leq \mu/C_3 \, (i \in \mathcal{N}), &\qquad 0 < s_i \leq \mu/C_3 \, (i \in \mathcal{B}), \\ s_i \geq C_3\gamma \, (i \in \mathcal{N}), &\qquad x_i \geq C_3\gamma \, (i \in \mathcal{B}). \end{aligned}$$

Teorema 1.4.6 *Neka je $\{(x^k, y^k, s^k)\}$ niz iteracija generisan Algoritmom KKP, PK ili DKP, i pretpostavimo da $\mu_k \to 0$ kada $k \to \infty$. Tada je niz $\{(x^k, s^k)\}$ ograničen i stoga ima bar jednu graničnu tačku. Svaka granična tačka odgovara strogo komplementarnom primal-dual rešenju.*

Posledica prethodne teoreme je da kada problem ima jedinstveno primal-dual rešenje (x^*, y^*, s^*), tada iteracioni niz bilo kog od razmatranih algoritama konvergira ka toj tački.

1.5 Algoritmi koji startuju iz nedopustive tačke

Prethodno razmatrani algoritmi za početnu tačku zahtevaju strogo dopustivu tačku $(x^0, y^0, s^0) \in \mathcal{F}^0$ koja je takođe i u nekoj okolini centralne putanje $\mathcal{C}$. Međutim, često nije lako odrediti takvu tačku. Postoje i problemi kod kojih takva tačka uopšte ne postoji. U problemu

$$\min 2x_1 + x_2 \quad \text{u odnosu na } x_1 + x_2 + x_3 = 5,\ x_1 + x_3 = 5,\ x \geq 0,$$

je primal dopustiv skup $\{(\beta, 0, 5 - \beta) | \, \beta \in [0, 5]\}$. Kako je $x_2 = 0$, strogo dopustiv skup $\mathcal{F}^0$ je prazan. U opštem slučaju, linearni problemi koji se dobijaju svođenjem opšteg problema na standardni oblik često nemaju strogo dopustive tačke. Jedan od

načina za prevazilaženje te teškoće je potapanje datog linearnog problema u nešto veći problem za koji je skup $\mathcal{F}^0$ neprazan.

Drugi način daje algoritam koji ne zahteva da početna tačka bude strogo dopustiva već se samo traži da komponente x i s budu strogo pozitivne. U opštem slučaju sve iteracije (x^k, y^k, s^k) generisane algoritmom su nedopustive, ali je granična tačka dopustiva i optimalna. Algoritam je moguće primeniti i kada je $\mathcal{F}^0 = \emptyset$. Ovakv pristup su dali Kojima, Megido i Mizuno [58], Zhang [122], Potra [79] i Wright [114].

Rezidume sistema jednačina

$$r_b - Ax - b, \quad r_c = A^T y + s - c$$

za $k-$tu iteraciju označimo sa r_b^k i r_c^k. Okolinu $\mathcal{N}_{-\infty}(\gamma)$ koja sadrži i nedopustive tačke definišmo sa

$$\begin{aligned}\mathcal{N}_{-\infty}(\gamma,\beta) = \quad & \{(x,y,s)|\, \|(r_b,r_c)\| \le [\|(r_b^0,r_c^0)\|/\mu_0]\beta\mu, \\ & (x,s) > 0, x_i s_i \ge \gamma\mu,\ i = 1,\dots,n\}\end{aligned}$$

gde su $\gamma \in (0,1)$ i $\beta \ge 1$ dati parametri i vrednosti (r_b^0, r_c^0) i μ_0 izračunate u početnoj tački (x^0, y^0, s^0). Primetimo da je neophodan uslov

$$\beta \ge 1$$

da bi osigurali da početna tačka (x^0, y^0, s^0) pripada $\mathcal{N}_{-\infty}(\gamma, \beta)$.

Algoritam NPD

Dato je $\gamma, \beta, \sigma_{\min}, \sigma_{\max}$, pri čemu je $\gamma \in (0,1)$, $\beta \ge 1$, i $0 < \sigma_{\min} < \sigma_{\max} \le 0.5$;

izabrati (x^0, y^0, s^0) tako da je $(x^0, s^0) > 0$;

for $k = 0, 1, 2, \dots$

izabrati $\sigma_k \in [\sigma_{\min}, \sigma_{\max}]$ i rešiti

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} -r_c^k \\ -r_b^k \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix},$$

izabrati za α_k najveću vrednost od $\alpha \in [0,1]$, tako da je

$$(x^k(\alpha), y^k(\alpha), s^k(\alpha)) \in \mathcal{N}_{-\infty}(\gamma, \beta) \tag{1.5.1}$$

i da važi sledeći uslov Armija:

$$\mu_k(\alpha) \le (1 - 0.01\alpha)\mu_k \tag{1.5.2}$$

postaviti

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k(\alpha_k), y^k(\alpha_k), s^k(\alpha_k));$$

end (for).

Neka je ν_k definisano sa

$$\nu_k = \prod_{j=0}^{k-1}(1-\alpha_j) \quad (\nu_0 = 1).$$

Kako su prve dve komponente funkcije F linearne, sledi

$$(r_b^k, r_c^k) = (1-\alpha_{k-1})(r_b^{k-1}, r_c^{k-1}) = \cdots = \nu_k(r_b^0, r_c^0), \tag{1.5.3}$$

i kako je $(x^k, y^k, s^k) \in \mathcal{N}_{-\infty}(\gamma, \beta)$, to je

$$\nu_k\|(r_b^0, r_c^0)\|/\mu_k = \|(r_b^k, r_c^k)\|/\mu_k \leq \beta\|(r_b^0, r_c^0)\|/\mu_0.$$

Ako pretpostavimo da je $(r_b^0, r_c^0) \neq 0$ tada je

$$\nu_k \leq \beta\mu_k/\mu_0.$$

U dopustivom slučaju je $(r_b^0, r_c^0) = 0$ i sve iteracije (x^k, y^k, s^k) su strogo dopustive i tada se Algoritam NPD svodi na Algoritam DKP. U nastavku pretpostavljamo da je početna tačka (x^0, y^0, s^0) nedopustiva.

Konvergencija algoritma sledi ako pokažemo egzistenciju konstante $\bar{\alpha}$ takve da je $\alpha_k \geq \bar{\alpha}$ za svako k. Iz uslova (1.5.2) sledi

$$\mu_{k+1} \leq (1 - 0.01\alpha_k) \leq (1 - 0.01\bar{\alpha})\mu_k \quad \text{za svako } k \geq 0,$$

i stoga niz $\{\mu_k\}$ konvergira Q-linearno ka nuli. Iz (1.5.3) takođe sledi

$$\|(r_b^{k+1}, r_c^{k+1})\| \leq (1-\bar{\alpha})\|(r_b^k, r_c^k)\|,$$

odakle sledi da niz rezidualnih normi teži nuli.

Teorema 1.5.1 *Niz $\{\mu_k\}$ generisan Algoritmom NPD konvergira Q-linearno ka nuli, i niz rezidualnih normi $\{\|(r_b^k, r_c^k)\|\}$ konvergira R-linearno ka nuli.*

Polinomijalna kompleksnost algoritma sledi iz činjenice da je donja granica $\bar{\alpha}$ dužine koraka inverzna polinomska funkcija od n ako se početna tačka izabere u obliku

$$(x^0, y^0, s^0) = (\zeta e, 0, \zeta e), \tag{1.5.4}$$

gde je ζ skalar za koji važi

$$\|(x^*, s^*)\|_\infty \leq \zeta, \tag{1.5.5}$$

za neko primal-dual rešenje (x^*, y^*, s^*). Iako se optimalno rešenje ne zna unapred, ovaj uslov je pogodan za praktičnu primenu jer dobro centrirana početna tačka za koju je količnik

$$\|(r_b^0, r_c^0)\|/\mu_0 \tag{1.5.6}$$

mali vodi bržoj konvergenciji nego tačka koja je mnogo bliža skupu rešenja Ω i slabo centrirana. Tačka (1.5.4) je centrirana i količnik (1.5.6) je približno $1/\zeta$.

Teorema 1.5.2 *Neka je $\varepsilon > 0$ dato i neka je data početna tačka $(x^0, y^0, s^0) = (\zeta e, 0, \zeta e)$, gde je ζ definisano u Lemi 1.5.2. Pretpostavimo da važi*

$$\zeta^2 \leq \frac{C}{\varepsilon^\kappa}$$

za neke pozitivne konstante C i κ. Tada postoji

$$K = O(n^2 |\log \varepsilon|)$$

tako da iteracije $\{(x^k, y^k, s^k)\}$ generisane Algoritmom NPD zadovoljavaju

$$\mu_k \leq \varepsilon \quad \text{za svako } k \geq K.$$

Dokazi Teorema 1.5.1 i 1.5.2 slede iz rezultata Kojime [57], Zhanga [122] i Wrighta [113]. Primetimo da iz

$$A\bar{x} = 0, \quad A^T\bar{y} + \bar{s} = 0,$$

sledi

$$\bar{x}^T\bar{s} = -\bar{x}^T A^T \bar{y} = 0.$$

Lema 1.5.1 *Postoji pozitivna konstanta C_1 tako da je*

$$\nu_k \|(x^k, s^k)\|_1 \leq C_1 \mu_k \quad \text{za svako } k \geq 0.$$

Lema 1.5.2 *Pretpostavimo da početna tačka zadovoljava (1.5.4) i (1.5.5). Tada za svaku iteraciju (x^k, y^k, s^k) važi*

$$\zeta \nu_k \|(x^k, s^k)\|_1 \leq 4\beta n \mu_k.$$

Lema 1.5.3 *Postoji pozitivna konstanta C_2 tako da je*

$$\|(D^k)^{-1}\Delta x^k\|_1 \leq C_2 \mu_k^{1/2}, \quad \|D^k \Delta s^k\| \leq C_2 \mu_k^{1/2}$$

za svako $k \geq 0$.

Lema 1.5.4 *Pretpostavimo da početna tačka zadovoljava (1.5.4) i (1.5.5). Tada postoji konstanta ω nezavisna od n tako da je*

$$\|(D^k)^{-1}\Delta x^k\|_1 \leq \omega \mu_k^{1/2}, \quad \|D^k \Delta s^k\| \leq \omega \mu_k^{1/2}. \tag{1.5.7}$$

Lema 1.5.5 *Postoji $\bar{\alpha} \in (0,1)$ tako da sledeća tri uslova važe za svako $\alpha \in [0, \bar{\alpha}]$ i svako $k \geq 0$:*

$$\begin{aligned}
&(x^k + \alpha\Delta x^k)^T (s^k + \alpha \Delta s^k) \geq (1-\alpha)(x^k)^T s^k, \\
&(x_i^k + \alpha \Delta x_i^k)(s_i^k + \alpha \Delta s_i^k) \geq (\gamma/n)(x^k + \alpha \Delta x^k)^T (s^k + \alpha \Delta s^k), \\
&(x^k + \alpha\Delta x^k)^T (s^k + \alpha \Delta s^k) \leq (1 - 0.01\alpha)(x^k)^T s^k.
\end{aligned}$$

Dakle, uslovi (1.5.1) i (1.5.2) su zadovoljeni za svako $\alpha \in [0, \bar{\alpha}]$ i svako $k \geq 0$.

Ako su zadovoljeni uslovi Leme 1.5.4, tada je

$$\bar{\alpha} \geq \bar{\delta}/n^2$$

za neki pozitivan broj $\bar{\delta}$ koji ne zavisi od n.

Teorema 1.5.3 *Neka je (x^k, y^k, s^k) niz generisan Algoritmom NPD. Tada postoji konstanta C_3 tako da za svako dovoljno veliko k važi*

$$0 < x_i^k \le \mu_k / C_3 \, (i \in \mathcal{N}), \qquad 0 < s_i^k \le \mu_k / C_3 \, (i \in \mathcal{B}),$$
$$s_i^k \ge C_3 \gamma \, (i \in \mathcal{N}), \qquad x_i^k \ge C_3 \gamma \, (i \in \mathcal{B}).$$

Dakle, bilo koja granična tačka niza $\{(x^k, s^k)\}$ može da posluži za konstrukciju striktno komplementarnog rešenja.

Teorema 1.5.4 *Neka je $\{(x^k, y^k, s^k)\}$ niz iteracija generisan Algoritmom NPD, i pretpostavimo da $\mathcal{F}^0 \neq \emptyset$. Tada je niz $\{(x^k, s^k)\}$ ograničen i stoga ima bar jednu graničnu taču.*

2 Simbolička implementacija Mehrotraovog algoritma

U ovom odeljku dajemo teorijske osnove primal-dual meotda i opisujemo simboličku implementaciju Mehrotraovog primal-dual metoda u programskom paketu MATHEMATICA. Dajemo i jednostavan algoritam za konstrukciju početne tačke iteracije. Kako je većina dostupnih test primera data u tzv. *MPS* formatu, razvijen je i softver za konverziju fajlova pisanih u *MPS* formatu u odgovarajući *NB* fajl. Implementacija je verifikovana na nekoliko ilustrativnih primera.

Simbolička implementacija primal-dual metoda podrazumeva mogućnost manipulacije formulama od strane kompjutera za vreme implementacije tih metoda. Poznati programski paketi bazirani na metodama unutrašnje tačke su razvijani u programskim jezicima FORTRAN, C i MATLAB. Vanderbeiov program LOQO [105] je pisan u programu C i dostupan je na internet adresi `http://www.sor.princeton.edu-/ rvdb/`. LIPSOL [123] je pisan u programu MATLABU i FORTRANU, i dostupan je na internet adresi `http://pc5.math.umbc.edu/ yzhang/`. Gondziov HOPDM je dostupan u izvornom kodu u FORTRANU [38]. Izvorni fajl je dostupan na `http://ecolu-info.unige.ch/ logilab/software/`. Jedan od popularnih i jakih kodova je PCx [13]. Veći deo PCx [14] koda je pisan u C-u, ali je glavni računski deo - Cholesky faktorizacija - pisan u FORTRANU 77 [2].

U ovom odeljku je razvijen eksperimentalni kod za primal-dual algoritam u programu MATHEMATICA. Cilj nije bio softver sa izuzetnim karakteristikama već predstavlja primenu mogućnosti simboličkog izračunavanja u programu MATHEMATICA. O programu MATHEMATICA videti [108] i [109]. U kodu je iskorišćena činjenica da MATHEMATICA poseduje velike mogućnosti za rešavanje sistema linearnih jednačina.

2.1 Opis Mehrotraovog algoritma

Mehrotraov algoritam generiše niz nedopustivih iteracija (x^k, y^k, s^k) za koje je $(x^k, s^k) > 0$. Traženi pravac se u svakoj iteraciji sastoji od tri komponente:

• afini "prediktor" pravac - Newtonov pravac za funkciju $F(x, y, s)$ definisanu sa (1.1.5),

• izraza za centriranje čija vrednost se određuje na osnovu podesivog izbora parametra za centriranje σ,

• "korektor" pravac koji kompenzuje izvesne nelinearnosti afinog pravca.

Osnovna odlika ovog algoritma je da se parametar centriranja ne određuje unapred kao kod opisanih algoritama. Ukoliko afini pravac obezbeđuje značajno smanjenje mere dualnosti μ pri čemu ostaje $(x, s) > 0$, zaključujemo da je potrebno malo centriranje u toj iteraciji, tako da se za σ uzima mala vrednost. Sa druge strane, ukoliko se u afinom pravcu dopušta samo kratak pomeraj pre nego što se naruši uslov $(x, s) > 0$, zaključujemo da je potrebna veća vrednost parametra za centriranje tako da uzimamo σ bliže jedinici.

Sada razmatramo Mehrotraov primal-dual algoritam [14, 112].

Algoritam MPD

Korak 1. Generisati početnu tačku (x^k, λ^k, s^k), $k = 0$.

Korak 2. Izračunati rezidume

$$r_b = Ax^k - b, \qquad r_c = A^T\lambda^k + s^k - c$$

i proverimo kriterijum za kraj algoritma

$$\frac{|r_b|}{1+|b|} \leq \epsilon, \quad \frac{|r_c|}{1+|c|} \leq \epsilon, \quad \frac{|c^Tx - b^T\lambda|}{1+|c^Tx|} \leq \epsilon.$$

U praksi je retkost da je treći uslov ispunjen i da istovremeno druga dva uslova ne važe. Prema tome, treći uslov je najvažniji i verovatno jedini uslov koji treba proveriti [2]. Najčešće se za granicu greške uzima vrednost $\epsilon = 10^{-8}$ [2].

Ako je kriterijum za kraj ispunjen, izlazni podatak je x^k; u suprotnom, ići na *Korak 3.*

Korak 3. Formirati matrice

$$S = \mathrm{diag}(s_1, \ldots, s_n), \; X = \mathrm{diag}(x_1, \ldots, x_n),$$

koje predstavljaju diajgonalne matrice sa vrednostima $s_1, \ldots, s_n$ i $x_1, \ldots, x_n$ na glavnoj dijagonali, respektivno. Konstruisati vektor e, definisan sa $e=(1, \ldots, 1)^T \in \mathbb{R}^n$.

Korak 4. Neka je $D = S^{-1/2}X^{1/2}$, i $r_{xs} = XSe$ i rešimo sledeći sistem u odnosu na $(\Delta x^{aff}, \Delta\lambda^{aff}, \Delta s^{aff})$:

$$\begin{aligned} rclAD^2A^T\Delta\lambda^{aff} &= -r_b - A(S^{-1}Xr_c - S^{-1}r_{xs}), && (2.1.1)\\ \Delta s^{aff} &= -r_c - A^T\Delta\lambda^{aff}, && (2.1.2)\\ \Delta x^{aff} &= -S^{-1}(r_{xs} + X\Delta s^{aff}). && (2.1.3) \end{aligned}$$

Napomenimo da je sistem ekvivalentan sistemu (1.1.7) pri čemu je parametar $\sigma = 0$. Time se dobija afini pravac iteracije.

Korak 5. Izračunati meru dualnosti $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i s_i$.

Korak 6. Izračunati uslov za nenegativnost iteracione tačke

$$\alpha_{aff}^{pri} = \max\{\alpha \in [0,1] \,:\, x^k + \alpha \Delta x^{aff} \geq 0\}$$
$$\alpha_{aff}^{dual} = \max\{\alpha \in [0,1] \,:\, s^k + \alpha \Delta s^{aff} \geq 0\}.$$

Korak 7. Izračunati $\mu_{aff} = \frac{1}{n}(x^k + \alpha_{aff}^{pri}\Delta x^{aff})(s^k + \alpha_{aff}^{dual}\Delta s^{aff})$ i $\sigma = (\frac{\mu_a ff}{\mu})^3$.

Ako je $\mu aff \ll \mu$, tada afini pravac obezbeđuje veliko smanjenje parametra μ tako da u *Koraku 8* treba izabrati parametar σ bliže nuli, tj. nastaviti iteraciju po afinom pravcu. Ako μaff nije značajno smanjeno u odnsu na μ, u *Koraku 8* treba izabrati parametar σ bliže jedinici. Time se iterativna tačka vraća bliže centralnoj putanji $\mathcal{C}$, tako da je algoritam u boljoj poziciji da sledećoj iteaciji značajno redukuje μ. Kriterijum za izbor parametra σ je maksimalno smanjenje μ. Međutim, kako je pri tome potreban veliiki broj aritmetičkih operacija, Mehrotra je predložio heuristiku $\sigma = (\frac{\mu_a ff}{\mu})^3$ koja se pokazala efikasnom na velikom broju test primera. Time se postiže velika ušteda u procesorskom vremenu potrebnom za izvršenje algoritma.

Korak 8. Izračunati $r_{xs} = -\sigma\mu e + \Delta X^{aff}\Delta S^{aff} e$, gde je

$$\Delta X^{aff} = \mathrm{diag}(\Delta x_1^{aff}, \ldots, \Delta x_n^{aff}), \quad \Delta S^{aff} = \mathrm{diag}(\Delta s_1^{aff}, \ldots, \Delta s_n^{aff}),$$

i rešiti sledeći sistem u odnosu na Δx^{cor}, $\Delta\lambda^{cor}$, i Δs^{cor}:

$$rclAD^2A^T\Delta\lambda^{cor} = AS^{-1}r_{xs}, \tag{2.1.4}$$
$$\Delta s^{cor} = -A^T\Delta\lambda^{cor}, \tag{2.1.5}$$
$$\Delta x^{cor} = -S^{-1}(r_{xs} + X\Delta s^{cor}). \tag{2.1.6}$$

Korak 9. Staviti

$$(\Delta x^k, \Delta\lambda^k, \Delta s^k) = (\Delta x^{aff}, \Delta\lambda^{aff}, \Delta s^{aff}) + (\Delta x^{cor}, \Delta\lambda^{cor}, \Delta s^{cor})$$

Korak 10. Izračunati

$$\alpha_{max}^{pri} = \max\{\alpha \geq 0 \,:\, x^k + \alpha\Delta x^k \geq 0\}$$
$$\alpha_{max}^{dual} = \max\{\alpha \geq 0 \,:\, s^k + \alpha\Delta s^k \geq 0\}.$$

Korak 11. Staviti

$$\alpha_k^{pri} = \min\{0.99\alpha_{max}^{pri}, 1\}$$
$$\alpha_k^{dual} = \min\{0.99\alpha_{max}^{dual}, 1\}.$$

Korak 12. Izračunati sledeću iteraciju

$$x^{k+1} = x^k + \alpha_k^{pri}\Delta x^k,$$
$$(\lambda^{k+1}, s^{k+1}) = (\lambda^k, s^k) + \alpha_k^{dual}(\Delta\lambda^k, \Delta s^k).$$

staviti $(x^k, \lambda^k, s^k) = (x^{k+1}, \lambda^{k+1}, s^{k+1})$, zameniti k sa $k+1$ i preći na *Korak 2.*

2.2 Implementacija Mehrotraovog algoritma

Unutrašnja reprezentacija problema

Sada dajemo opis implementacije Mehrotraovog algoritma u programu MATHEMATICA. Deo ovih rezultata je publikovan u radovima [91, 92]. Razmatraćemo problem linearnog programiranja u opštem obliku. Umesto ulaznog fajla u *MPS* formatu, koristimo *NB* fajl format koji podržava program MATHEMATICA. Na taj način je problem linearnog programiranja predstavljen u intuitivnijoj i prirodnijoj formi, koju zovemo unutrašnja reprezentacija i upisujemo je u obliku *NB* fajla. Unutrašanja reprezentacija problema sadrži dva dela: izraz koji predstavlja ciljnu funkciju i listu datih ograničenja.

Primer 2.2.1 Unutrašnja reprezentacija test problema koji je u literaturi poznat pod nazivom *Afiro* jednaka je

```
-.4x2-.32x13-.6x17-.48x29+10.x32,
{-x1+x2+x3==0,-1.06x1+x4==0,x1<=80.,-x2+1.4x13<=0,
   -x5-x6-x7-x8+x13+x14==0,-1.06x5-1.06x6-.96x7-.86x8+x15==0,
   x5-x9<=80.,x6-x10<=0,x7-x11<=0,x8-x12<=0,-x16+x17+x18+x19==0,
   -.43x16+x20==0,x16<=500.,-x17+1.4x29<=0,
   -.43x21-.43x22-.39x23-.37x24+x31==0,
   x21+x22+x23+x24-x29+x30+x32==44.,
   x21-x25<=500.,x22-x26<=0,x23-x27<=0,x24-x28<=0,
   2.364x9+2.386x10+2.408x11+2.429x12-x19+2.191x25+2.219x26+
        2.249x27+2.279x28<=0,
   -x3+.109x16<=0,-x14+.109x21+.108x22+.108x23+.107x24<=0,
   .301x1-x18<=0,.301x5+.313x6+.313x7+.326x8-x30<=0,
   x4+x20<=310.,x15+x31<=300.}
```

Kako je u programu MATHEMATICA moguća manpulacija formulama, nije veliki problem uzeti kompletnu informaciju o problemu iz takve reprezentacije. Pretpostavimo da je ciljna funkcija označena parametrom *objective*. Takođe, neka je lista ograničenja označena sa *constraints*.

Korak a) U ovom koraku se rešavaju sledeća dva glavna problema:

- generisanje matrice problema a, koja je da data u unutrašnjoj reprezentaciji *objective*, *constraints*, i

- generisanje liste koeficijenata koji predstavljaju ciljnu funkciju.

a1) Transformišemo listu datih ograničenja i podelimo je u tri dela. Deo *les* je generisan posle sledeće transformacije uslova tipa <=.

```
les=Cases[constraints,x_<=y_->x-y];
```

Slično, drugi i treći deo su generisani iz ograničenja tipa >= i ==, respektivno.

```
gre=Cases[constraints,x_>=y_->x-y];
equ=Cases[constraints,x_==y_->x-y];
```

Dužina generisanih listi je korisna informacija u implementaciji.

```
p=Length[les];  q=Length[gre];   m=Length[equ];
```

a2) Lista b datih promenljivih se generiše u sledećem kodu:

```
b={};
For[i=1,i<=p,i++,
   b=Union[b,Map[Variables,les,{1}][[i]] ]
];
For[i=1,i<=q,i++,
   b=Union[b,Map[Variables,gre,{1}][[i]] ]
];
For[i=1,i<=m,i++,
   b=Union[b,Map[Variables,equ,{1}][[i]] ]
];
```

a3) Lista koja sadrži uređeni par čiji prvi element čine transformisana ograničenja a drugi element je lista promenljivih, generiše se u sledećem kôdu:

```
For[i=1,i<=p,i++,
   a=Append[a,List[les[[i]],b]]
];
For[i=1,i<=q,i++,
   a=Append[a,List[gre[[i]],b]]
];
For[i=1,i<=m,i++,
    a=Append[a,List[equ[[i]],b]]
];
```

a4) Sada se matrica a standardnog oblika problema može generisati primenjujući funkciju *Apply* kao funkcional:

```
a=Apply[Coefficient,a,{1}];
```

Konačno, lista ce koja sadrži koeficijente ciljne funkcije se generiše na sledeći način:

```
ce=Coefficient[objective,b];   k=Length[ce];
```

Primer 2.2.2 Za unutrašnju reprezentaciju

```
-2x1+3x2, {3x1+4x2<=14, 3x1+4x2>=5, x1+2x2<=6, x1+3x2==3}
```

imamo $p = 2$, $q = 1$ i $a = \{\{3,4\},\{1,2\},\{3,4\},\{1,3\}\}$. U slučaju minimizacije je $ce = \{-2,3\}$, i $ce = \{2,-3\}$ u slučaju maksimizacije.

Korak b) Sada moramo transformisati problem u standardni oblik i generisati matricu tog problema a. Primenom funkcije $a = form[a,p,q]$ moguće je generisati matricu a standardnog oblika lineranog problema. To se može postići dodavanjem q-dimenzionog vektora oblika $\{0,\ldots,0,\alpha,0,\ldots,0\}$ i-toj vrsti matrice a, gde je $\alpha \in \{-1,0,1\}$ postavljeno na i-tu poziciju.

```
form[B_,p_,q_]:=
 Module[{i,L={},y},
    For[i=1, i<=First[Dimensions[B]], i++,
       If[i<=p,
           y=Join[B[[i]],ReplacePart[Table[0,{q}],1,i]],
           If[i<=q,
```

```
            y=Join[B[[i]],ReplacePart[Table[0,{q}],-1,i]],
            y=Join[B[[i]],Table[0,{q}]]
      ]   ];
      L=Append[L,y]
    ];
    Return[L]
]
```

Sada se matrica a standardnog oblika problema i proširena lista promenljivih u ciljnoj funkciji može generisati sledećim kodom:

```
q=p+q; m=q+m;
a=form[a,p,q];
ce=Join[ce,Table[0,{q}]];
```

Na taj način, prvih p vrsta matrice a, koje odgovaraju ograničenjima tipa $<=$, proširene su vektorima $\{0,\ldots,0,1,0,\ldots,0\}$; vrste matrice a koje odgovaraju ograničenjima tipa $>=$, indeksirane brojevima $p+1,\ldots,q$, proširene su vektorima $\{0,\ldots,0,-1,0,\ldots,0\}$; vrste matrice a koje odgovaraju ograničenjima tipa $==$, indeksirane brojevima $q+1,\ldots,m$, proširene su vektorima $\{0,\ldots,0,0,0,\ldots,0\}$;

Primer 2.2.3 Za skup ograničenja iz Primera 2.2.2 rezultat funkcije $form[a,p,q]$ je lista

$$a = \{\{3,4,1,0,0\},\{1,2,0,1,0\},\{3,4,0,0,-1\},\{1,3,0,0,0\}\}.$$

Takođe, ciljna funkcija je data listom $\{-2,3,0,0,0\}$ (za minimizaciju) ili $\{2,-3,0,0,0\}$ (za maksimizaciju).

Korak c) Generisati listu b slobodnih koeficijenata sa desne strane datih ograničenja.

```
b={};
For[i=1,i<=p,i++,
   If[NumberQ[First[Cases[les[[i]],x_]]],
      b=Append[b,-First[Cases[les[[i]],x_]]],
      b=Append[b,0]
]  ];
For[i=1,i<=q,i++,
    If[NumberQ[First[Cases[gre[[i]],x_]]],
       b=Append[b,-First[Cases[gre[[i]],x_]]],
       b=Append[b,0]
]   ];
For[i=1,i<=m,i++,
    If[NumberQ[First[Cases[equ[[i]],x_]]],
       b=Append[b,-First[Cases[equ[[i]],x_]]],
       b=Append[b,0]
]    ];
```

Kako je najveći broj test problema dostupan u MPS formatu, potreban nam je softver za transkripciju iz *MPS* formata u *NB* format. Taj softver je pisan u programskom jeziku DELPHI, i biće opisan u nastavku.

Primer 2.2.4 Razmotrimo poznati test problem pod nazivom *Blend*. Odgovarajući MPS fajl se transformiše u sledeću unutrašnju reprezentaciju:

```
3.2x1+2.87x2+.0044x8+.07x13+.0378x14+.155x15+.155x16+.155x17+.155x18+.0528x19+
.0528x20+.0528x21+.0528x22+.128x23+.118x24+.0924x36-5.36x37+.0924x44-5.08x45+
.0924x52-4.51x53-2.75x57-4.2x60-3.6x63-2.x70+.04x77+3.x79+.4x80+.0132x82+.01x83,
{-.2931x2+x4==0,-.537x1+x3==0,-.131x1-.117x2+x22+x62==0, -.1155x1-.0649x2+x17
+x21+x61==0, -.0365x1-.1233x2+x15+x19+x67==0, -.143x1-.2217x2+x16+x20+x66==0,
-.037x1+x64==0, -.18x2+x65==0, -.0277x3-.0112x4-.175x5-.271x6-.2836x7-.0571x24
-x25-x54+x55+x74==0, -.0563x3-.0378x4-.27x5-.3285x6-.3285x7-.0185x19-.0185x20
-.0184x21-.0184x22-.0114x24-x26-x27+x35+x43+x51+x56+x73-.5x79==0,
-.199x3-.1502x4-.028x5-.0255x6-.0241x7+x28+x38+x46+x58==0,
-.6873x3-.7953x4+x8+x59==0, -.017x3-.0099x4-.455x5-.2656x6-.2502x7-.0568x19
-.0568x20-.0564x21-.0564x22 +.76x23+.6571x24+x27+x72-.5x79==0,
-x8+x11+x12==0,-x9-x11+x13==0,-x10-x12+x14==0,x7-.0588x13==0,
x6-.0404x14==0,-.8145x13+x30+x40+x48==0,-.8564x14+x31+x41+x49==0,
x5-.3321x15-.3321x16-.2414x17-.2414x18==0,
x9+x10-.5875x15-.5875x16-.6627x17-.6627x18==0,
-.0091x13-.0069x14-.362x15-.362x16-.293x17-.293x18+x29+x39+x47==0,
-.0806x19-.0806x20-.078x21-.078x22+.5714x24+x26+x76==0,
-.0658x19-.0658x20-.0655x21-.0655x22+.5714x23+x25+x75==0,
-.0328x19-.0328x20-.0303x21-.0303x22+x54==0,
-.4934x19-.4934x20-.475x21-.475x22+x32+x42+x50==0,
x18-.2922x19-.2922x20-.305x21-.305x22+x69==0,
-.0096x19-.0096x20+x68==0,-x23+x33==0,-x24+x34==0,
-x28-x29-x30-x31-x32-x33-x34-x35+x37==0,-x38-x39-x40-x41-x42-x43+x45==0,
-x61-x62+x63==0,-x64-x65-x66-x67-x68-x69+x70==0,
-x46-x47-x48-x49-x50-x51+x53==0,-x55-x56+x57==0, -x58-x59+x60==0,
.0042x2+.0327x8-.8239x13-.7689x14+2.3x15+2.3x16+2.3x17+2.3x18+x71==0,
.003x1+.003x2+.01303x3+.01303x4+.01303x5+.01303x6+.01303x7+.0081x13+
.0063x14-.0654x19-.0654x20-.0654x21-.0654x22+.1869x23+.1724x24-x77+x78==0,
.0587x1+.1053x2+.0506x3+.0448x4+.0506x5+.0506x6+.0506x7+.094x8-.
2112x13-.156x14-.2049x15-.2049x16-.1531x17-.1531x18-.2535x19-.2535x20-
.2703x21-.2703x22+.2796x23+.2579x24-.325x71-4.153x72-4.316x73-3.814x74-
3.808x75-4.44x76+1.42x77-x80+x81==0,
.15x1+.185x2+.209x3+.185x4+.209x5+.209x6+.209x7+.045x8+.387x13+.297x14
+.826x15+.826x16+.826x17+.826x18+.632x19+.632x20+.632x21+.632x22+2.241x23
+2.067x24-x82==0,  .302x1+.384x2+.495x3+.721x4+.495x5+.495x6+.495x7
+.793x8+1.03x13+.792x14+14.61x15+14.61x16+14.61x17+14.61x18+.6807x19
+.6807x20+.6807x21+.6807x22+2.766x23+2.552x24-x83==0,
-7.95x28-8.84x29-9.43x30-9.03x31-9.23x32-9.4x33-9.74x34-9.74x35-.493x36
+10.03x37<=0,  -8.7x28-9.45x29-9.57x30-9.32x31-9.22x32-9.85x33-10.1x34
-9.9x35-.165x36+10.03x37<=0, -3.x28-3.x29-3.x30-3.x31-3.x32-3.x33-3.x34
-3.x35+x36<=0, 14.x28+12.x29+3.5x30+3.5x31+6.x32+2.5x33+3.3x34+66.x35
-9.5x37<=0, x28+x29+.233x30+.205x31+.381x32+.39x33+.233x34+x35-.5x37<=0,
-x28-x29-.358x30-.333x31-.509x32-.77x33-.58x34-x35+.5x37<=0,
-.00862x2-7.98x38-8.87x39-9.46x40-9.06x41-9.26x42-9.77x43-.435x44+9.65x45<=0,
-.00862x2-8.58x38-9.33x39-9.45x40-9.2x41-9.13x42-9.78x43-.208x44+9.65x45<=0,
-3.x38-3.x39-3.x40-3.x41-3.x42-3.x43+x44<=0,
14.x38+12.x39+3.5x40+3.5x41+6.x42+66.x43-9.5x45<=0,
x38+x39+.233x40+.205x41+.318x42+x43-.5x45<=0,
-x38-x39-.358x40-.333x41-.509x42-x43+.5x45<=0,
-.0101x2-7.99x46-8.88x47-9.47x48-9.07x49-9.27x50-9.78x51-.426x52+9.05x53<=0,
-.0101x2-8.59x46-9.34x47-9.46x48-9.21x49-9.14x50-9.79x51-.204x52+9.05x53<=0,
-3.x46-3.x47-3.x48-3.x49-3.x50-3.x51+x52<=0,
14.x46+12.x47+3.5x48+3.5x49+6.x50+66.x51-9.5x53<=0,
x46+x47+.233x48+.205x49+.318x50+x51-.5x53<=0,
-x46-x47-.358x48-.333x49-.509x50-x51+.5x53<=0,
```

```
10.1x64+12.63x65+8.05x66+6.9x67+8.05x68+4.4x69-10.1x70<=0,
-14.x58-.8x59+2.x60<=0,14.x58+.8x59-3.x60<=0,
x3+x4+x15+x17+x19+x21+x22+x63+x67<=23.26,
x16+x20+x66<=5.25, x1<=26.32,x2<=21.05,
1.3x13+x14<=13.45,x15+x16+x17+x18<=2.58,x19+x20+x21+x22<=10.,
x23+x24<=10.,.64x37-.36x45-.36x53<=0,.35x37+.35x45-.65x53<=0}
```

Početna tačka algortima

Prvi problem u implementaciji primal-dual algoritma je izbor početne tačke [2, 112, 118]. Poznato je da tačke koje su relativno blizu optimalnog rešenja ali nisu dobro centrirane, često dovode do numeričkih problema. Rezultat [112] Teorema 6.1, garantuje opštu konvergenciju primal-dual algoritma iz bilo koje startne tačke (x^0, λ^0, s^0) pri čemu je $(x^0, s^0) > 0$. Mehrotra [70] predlaže rešavanje jednog problema kvadratnog programiranja za dobijanje početne tačke. Heuristika za određivanje (x^0, λ^0, s^0) u prvom koraku računa (x^*, λ^*, s^*) kao rešenje problema:

$$\begin{aligned} &\min_{x} |x|^2 \quad \text{pod uslovom } Ax = b, \\ &\min_{(\lambda,s)} |s|^2 \quad \text{pod uslovom } A^T\lambda + s = c. \end{aligned} \tag{2.2.1}$$

To znači, x^* i s^* su vektori najmanje norme za koje su rezidumi r_b i r_c jednaki nuli. Startna tačka se zatim određuje sa

$$(x^0, \lambda^0, s^0) = (x^* + \delta_x e, \lambda^*, s^* + \delta_s e),$$

gde su skalari δ_x i δ_s određeni tako da je $(x^0, s^0) > 0$. Vreme potrebno za određivanje početne tačke na taj način je približno jednako vremenu potrebnom za jednu iteraciju metoda unutrašnje tačke [2].

Sada predlažemo dva jednostavna algoritma za određivanje početne tačke. Prvi algoritam određuje tačku koja je dobro centrirana i istovremeno zadovoljava bar jedno ograničenje, ukoliko je to moguće. Pored toga, predloženi algoritmi generišu početnu tačku desetak puta brže od jedne iteracije metoda unutrašnje tačke. Efikasnost predloženih heuristika je proverena na velikom broju test primera. Napomenimo da su rezultati iz ovog odeljka su objavljeni u radovima [75, 76]. Rezultati iz tih radova su citirani u [20, 21, 22] gde se napominje da je izbor početne tačke primenom naše heuristike dao bolje numeričke performanse od početne tačke generisane sa (2.2.1).

Algoritam PocTacka

Korak 1.1. Izračunati vrednost $aux = \max\{q_1, \ldots, q_m\}$, gde je

$$q_i = \begin{cases} \max\left\{\dfrac{b_i}{\sum_{j=1}^n a_{ij}}, 1\right\}, & \sum_{j=1}^n a_{ij} \neq 0; \\ 1, & \sum_{j=1}^n a_{ij} = 0,\ i = 1, \ldots, m. \end{cases} \tag{2.2.2}$$

Korak 1.1.a

Motivisani radom [52], generišemo alternativnu početnu tačku, koja se dobija na sličan način kao i tačka u *Koraku 1.1* i *Koraku 1.2*, koristeći vrednost

$aux = \max\{q_1, \ldots, q_m\}$, gde je

$$q_i = \begin{cases} \max\left\{ \dfrac{b_i}{\sqrt{\sum_{j=1}^{n} a_{ij}^2}}, 1 \right\}, & \sum_{j=1}^{n} a_{ij}^2 \neq 0; \\ 1, & \sum_{j=1}^{n} a_{ij}^2 = 0, \ i = 1, \ldots, m. \end{cases}$$

```
inicp[a_,b_]:=
  Module[{i,aux,k,m,pl={}},
    k=Last[Dimensions[a]];      m=First[Dimensions[a]];
    For[i=1,i<=m,i++,
        aux=N[Sum[a[[i,j]],{j,k}],20];
        If[aux !=0,
           AppendTo[pl,Max[b[[i]]/aux,20],1]]
    ]   ];
    If[pl=={}, aux=1, aux=Max[pl] ];
    Return[pom]
  ]
```

Korak 1.2. Generisati startne tačke x i s, čije su koordinate jednake aux, i startnu tačku l, čije su koordinate jednake nuli.

```
pi=inicp[a,b];
x=Table[pi,{k+q}];      s=x;     l=Table[0,{m}];
```

Pomoćne rutine

Korak 6 je implementiran u sledećoj funkciji *solv1*. Parametar x označava k-tu iteraciju x^k, i parametar dx označava vektor Δx^{aff}. Moguća su sledeća dva slučaja:

A. Ako je $dx[[i]] > 0$ za svako i, tada je $\alpha_{aff}^{pri} = 1$.

B. Ako postoji najmanje jedno i tako da je $dx[[i]] < 0$, stavimo

$$\alpha_{aff}^{pri} = \min\{Abs[x[i]/dx[i]], \ i = 1, \ldots, n\}.$$

```
solv1[x_,dx_]:=
  Module[{i,a=1,al={}},
    For[i=1,i<=First[Dimensions[x]],i++,
       If[dx[[i]]<0,
          al=Append[al,N[Abs[x[[i]]/dx[[i]]],20]]
    ]  ];
    If[Min[al]<1,a=Min[al]];
    Return[a]
  ]
```

Slična funkcija *solv2* može biti iskorišćena u implementaciji *Koraka 10.*

```
solv2[x_,dx_]:=
```

```
Module[{i,a=Abs[Max[]],al={}},
  For[i=1,i<=First[Dimensions[x]],i++,
      If[dx[[i]]<0,
         al=Append[al,Abs[x[[i]]/dx[[i]]]]
  ]   ];
  a=Min[al];    Return[a]
]
```

Glavni algoritam

Glavna funkcija koristi sledeće formalne parametre:
*objective_*: izraz predstavlja ciljnu funkciju.
*constraints_*: je lista datih ograničenja.
*eps_*: izabrana tačnost.
*iter_*: maksimalan broj iteracija.

Korak 1:

```
pi=inicp[a,b];
x=Table[pi,{k+q}];      s=x;  l=Table[0,{m}];
```

Korak 2:

```
at=Transpose[a];
rb=N[a.x-b];
rc=N[at.l+s-ce];
While[(Abs[ce.x-b.l]/(1+Abs[ce.x])>eps),
```

Korak 3:

```
  id=Table[1,{k+q}];
  Xmat=DiagonalMatrix[x];   Smat=DiagonalMatrix[s];
  rxs=Xmat.Smat.id;
```

Korak 4:
Koristimo sledeće pojednostavljenje:

$$S^{-1} = \mathrm{diag}(1/s[[1]], \ldots, 1/s[[k+q]]),$$
$$D^2 = \mathrm{diag}(x[[1]]/s[[1]], \ldots, x[[k+q]]/s[[k+q]]).$$

Sistem u *Koraku 4* može biti rešen korišćenjem MATHEMATICA/ funkcije *LinearSolve*.

```
  is=DiagonalMatrix[1/s];  d=DiagonalMatrix[x/s];
  Dlaff=LinearSolve[a.d.at, -rb-a.(is.Xmat.rc-is.rxs)ad ];
  Dsaff=N[-rc-at.Dlaff,20];
  Dxaff=N[-is.(rxs+Xmat.Dsaff),20];
```

Korak 5:

```
  mi=(x.s)/(k+q);
```

Korak 6:

```
AlfaPa=solv1[x,Dxaff];   AlfaDa=solv1[s,Dsaff];
```

Korak 7:

```
SigCent=N[((((x+AlfaPa*Dxaff).(s+AlfaDa*Dsaff))/(k+q))/mi)^3,20];
```

Korak 8:
Za rešavanje sistema u *Koraku 8* ponovo koristimo funkciju *LinearSolve*.

```
DXaff=DiagonalMatrix[Dxaff];   DSaff=DiagonalMatrix[Dsaff];
rxs=N[-mi*SigCent,20]*id+N[DXaff.DSaff,20].id;
Dlcc=LinearSolve[a.d.at,  a.is.rxs]  ];
Dscc=N[-at.Dlcc,20];
Dxcc=N[-is.(rxs+Xmat.Dscc),20];
```

Korak 9:

```
dx=Dxaff+Dxcc;  dl=Dlaff+Dlcc;   ds=Dsaff+Dscc;
```

Korak 10 i *Korak 11:*

```
AlfaPa=N[Min[0.99*solv2[x,dx],1],20];
AlfaDa=N[Min[0.99*solv2[s,ds],1],20];
```

Korak 12:

```
   x=N[x+AlfaPa*dx,20];l=N[l+AlfaDa*dl,20];s=N[s+AlfaDa*ds,20];
  (* Generate conditions for termination of the cycle:  *)
     rb=N[a.x-b,20];
     rc=N[at.l+s-ce,20];
  ]; (* End of the While cycle *)
  Return[{N[z,20],Take[N[x,20],k]}];
]
```

2.3 Translacija MPS fajla u NB fajl

Većina dostupnih test problema je data u MPS formatu čija je glavna karakteristika zapis matrice problema po kolonama. Cilj nam je da takav format prevedemo u mnogo prirodniji i korisnicima dostupniji NB format koji koristi MATHEMATICA i u kome se problem zapisuje na klasičan način tako da ne zahteva nikakvo posebno predznanje od korisnika. Za taj cilj koristimo programsko okruženje Delphi.

Sažet opis *MPS* formata je dostupan na *ftp://softlib.cs.rice.edu/pub/miplib*. U *MPS* fajl formatu postoji šest sekcija: *NAME, ROWS, COLUMNS, RHS, RANGES, BOUNDS* dok je sedma upravljačka reč *ENDATA*.

Sledi kratak opis tih sekcija:

NAME je prva sedkcija koja se sastoji od naredbe *NAME* i samog imena fajla koje počinje od pete kolone.

U sekciji *ROWS* dat je tip i ime svake vrste matrice ograničenja. Tip vrste (nejednakosti ili jednakosti) dat je u drugoj koloni a ime se nalazi između pete i dvanaeste kolone. Tip kolone je definisan na sledeći način:

tip	značenje	tip	značenje
E	jednakost	N	ciljna funkcija
L	manje ili jednako	N	bez ograničenja
G	veće ili jednako		

U sekciji *COLUMN*, definisana su imena promenljivih zajedno sa koeficijentima ciljne funkcije i nenula elementima matricwe ograničenja. Postoje sledeća polja:

polje	kolona	polje	kolona
ime kolone	5 - 12		
ime vrste	15 - 22	ime vrste	40 - 47
vrednost	25 - 36	vrednost	50 - 61

Poslednja dva polja su opciona.

U *RHS* sekciji su date informacije o slobodnim koeficijentima ograničenja datog problema.

polje	kolona	polje	kolona
rhs ime	5 - 12		
ime vrste	15 - 22	ime vrste	40 - 47
vrednost	25 - 36	vrednost	50 - 61

Poslednja dva polja su opcija kao i kod sekcije *COLUMNS*.

Sekcija *RANGES* je opciona i ređe se koristi, tako da se na njoj nećemo posebno zadržavati.

Sekcija *BOUNDS* je takođe opciona, ali je delimično opisujemo. U toj sekciji se opisuju ograničenja promenljivih.

polje	kolona
tip	2 - 3
ime ograničenja	5 - 12
ime kolone	15 - 22
vrednost	25 - 36

U slučaju da ograničenja nisu data, po definiciji se pretpostavlja($0 \leq x \leq \infty$).

Polje *type* označava tip ograničenja:

tip	značenje		tip	značenje	
LO	donja granica	$x >= b$	FR	slobodna promenljiva	
UP	gornja granica	$x <= b$	MI	donja granica -∞	-$\infty < x$
FX	fiksna promenljiva	$x = b$	BV	binarna promenljiva	$x = 0$ ili 1

U ovoj verziji programa konvertujemo *LO, UP* i *FX* tip ograničenja.

Dajemo *Afiro* problem zapisan u *MPS* formatu radi ilustracije (deo problema je izostavljen zbog veličine fajla):

```
NAME          AFIRO
ROWS
 E  R09
 E  R10
 L  X05
 .
 .
 .
 L  X50
 L  X51
 N  COST
COLUMNS
    X01       X48              .301   R09              -1.
    X01       R10             -1.06   X05               1.
    X02       X21              -1.    R09               1.
    X02       COST             -.4
    X03       X46              -1.    R09               1.
    .
    .
    .

    X36       COST            -.48
    X37       X49              -1.    R23               1.
    X38       X51               1.    R22               1.
    X39       R23               1.    COST             10.
RHS
    B         X50             310.    X51              300.
    B         X05              80.    X17               80.
    B         X27             500.    R23               44.
    B         X40             500.
ENDATA
```

Generalna ideja je: otvoriti *MPS* fajl i prevoditi ga red po red u odgovarajući *NB* format.

U programu se koriste sledeće konstante i promenljive:

```
const
      S0: PChar = 'NAME';     S1: PChar = 'ROWS';
      S2: PChar = 'COLUMNS';  S3: PChar = 'RHS';
      S4: PChar = 'RANGES';   S5: PChar = 'BOUNDS';
      S6: PChar = 'ENDATA';
var
      I, J, K, L, NumVar, NumActualVar: integer;
      S,Name,LinesType,ActualVar,ActualLin,ActualNum,BoundsType:string;
      E, F: TextFile;
      LinesName, Lines, Vars: array of string;
      BoundsM: boolean;
```

Najvažnije su: *LinesName, Lines, Vars*, promenljive tipa dinamički niz. Dinamički niz nema fiksiranu veličinu ili dužinu. Memorija za dinamički niz se dodeljuje kada je data vrednost niza ili se prelazi na proceduru *SetLength*. Kada se dinamički niz zapamti u memoriji, moguć je prelaz na standarne funkcije *Length, High*, and *Low*. Kada se otvara *MPS* fajl i čita red po red, nije nam poznato koliko tačno ograničenja i promenljivih imamo. Ako se ne koristi dinamički niz, potrebno je pročitati fajl nekoliko puta što bi znatno usporilo program.

Promenljiva *LinesName* predstavlja oblast imena vrsta u *ROWS* sekciji. String oblast *Lines* predstavlja ciljnu funkciju i ograničenja. Kada kompletiramo oblast *Lines*, ostaje samo da zapišemo string u novi fajl. Promenljiva *Vars* je niz imena kolona u sekciji *COLUMN*.

Sekcija *NAME* se jednostavno konvertuje sledećim kodom:

```
AssignFile(F, Edit1.Text); Reset(F);
Read(F,S); //  Read the problem name
Name := '(* '+S+' *)';
//  and incorporate it into the MATHEMATICA comment
Readln(F);
```

Zatim se iz *ROWS* sekcije izdvajaju promenljive *LinesType* i *LinesName*. *Lines Type* je string koji predstavlja tip vrste u *ROWS* sekciji. To se postiže sledećim kodom:

```
Read(F,S);
I := 1;   //  I denotes the number of rows (constraints)
while (StrLIComp(PChar(S), S2, 7) <> 0) do
 begin    //  Cycle terminates in the case S='COLUMNS'
   SetLength(LinesType, I);   SetLength(LinesName, I);
   LinesType[I] := S[2];  //  Remember row type
   //  Extract row name, deleting first 4 characters from S
   L := Length(S);
   for J := 1 to (L-4) do S[J] := S[J+4];
   SetLength(S,L-4);      LinesName[I-1] := S;
   Readln(F);  Read(F,S);
   I := I + 1;
 end;
I := I - 1;
SetLength(Lines, I);
```

Zatim se upisuje *COLUMNS* sekcija. Glavna petlja je:

```
Readln(F);  Read(F,S);
NumVar := 0;
while (StrLIComp(PChar(S), S3, 3) <> 0) do
  begin  //  Cycle terminates in the case S='RHS'
    J := 1;
    //  Analysis of the current string S
    Readln(F);   Read(F,S);
```

```
  end;
```

gde je *NumVar* broj promenljivih u *MPS* fajlu. Napomenimo da promenljive u *MPS* formatu mogu imati razna imena, ali ih mi menjamo u standarni oblik $x1, x2, \ldots$ Promenljiva J predstavlja trenutnu poziciju u stringu koji se razmatra. U sledeća četiri koraka opisujemo proces analize tekućeg stringa S.

Korak 1. Promenljiva *ActualVar* predstavlja ime tekuće kolone u *COLUMNS* sekciji. To ime izdvajamo iz tekuće kolone na sledeći način:

```
L := 1;
SetLength(ActualVar, Length(S));
while S[J] = ' ' do J := J + 1;
while S[J] <> ' ' do
  begin
    ActualVar[L] := S[J];     L := L + 1;  J := J + 1;
  end;
SetLength(ActualVar, L-1);
```

Korak 2. Ukoliko je promenljiva (ime kolone) *ActualVar* u *Vars* oblasti, moramo da nađemo njen indeks; ukoliko nije, moramo da je ubacimo u tu oblast. Napomenimo da u nastavku promenljiva *NumActualVar* predstavlja indeks promenljive u oblasti *Vars*.

```
K := 0;
while (K < NumVar) do
  if(StrLIComp(PChar(Vars[K]),PChar(ActualVar),Length(ActualVar))<>0)
      then K:= K + 1
   else break;
if K >= NumVar then
   begin  // Put the column name in array Vars
     NumVar := NumVar + 1;
     SetLength(Vars, NumVar);
     Vars[NumVar-1] := ActualVar;
   end;
NumActualVar := K;
```

Korak 3. Iz *COLUMNS* sekcije preuzimamo *row name* i *value* polja, plus opciono *row name* i *value* polja.

```
while (J < Length(S))do
 begin
   // Finding the content of the row name field
   // and putting it in the ActualLin variable
   while S[J] = ' ' do J := J + 1;
   SetLength(ActualLin, Length(S));  L:=1;
   while S[J] <> ' ' do
     begin
       ActualLin[L] := S[J];  L := L+1;  J := J+1;
     end;
   SetLength(ActualLin, L-1);
```

```
    // Finding the index K of ActualLin in the LinesName array
    K := 0;
    while(StrLIComp(PChar(LinesName[K]),PChar(ActualLin),
                    Length(ActualLin))<>0)
          do K := K + 1;
    // Finding the content of the value field
    // and putting it in the ActualNum variable
    while S[J] = ' ' do J := J + 1;
    SetLength(ActualNum, Length(S));   L:=1;
    while((S[J] <> ' ') and (J <= Length(S))) do
      begin
        ActualNum[L]:=S[J];  L:=L+1;  J:=J+1;
      end;
    SetLength(ActualNum, L-1);
    // Several simplifications
    if((ActualNum[1]<>'+') and (ActualNum[1]<>'-') and (Lines[K]<>''))
        then  ActualNum := '+' + ActualNum;
    if((ActualNum='1') or (ActualNum='1.')  or (ActualNum='1.0'))
        then ActualNum := '';
    if((ActualNum='+1') or (ActualNum='+1.')  or (ActualNum='+1.0'))
        then ActualNum := '+';
    if((ActualNum='-1') or (ActualNum='-1.')  or (ActualNum='-1.0'))
        then ActualNum := '-';
    // Finally we are forming the constraint
    Lines[K]:=Lines[K]+ActualNum+'x'+IntToStr(NumActualVar+1);
    // Skip whitespace characters in the end of line
    while ((S[J] = ' ') and (J <= Length(S))) do J:=J+1;
end;
```

Korak 4. Sekcija *COLUMNS* je ovim završena. Pre učitavanja *RHS* sekcije, dodajemo ==, <= i >= simbole na kraju linija koja predstavljaju ograničenja:

```
for I := 0 to High(Lines) do
  case LinesType[I+1] of
    'E': Lines[I] := Lines[I] + '==';
    'L': Lines[I] := Lines[I] + '<=';
    'G': Lines[I] := Lines[I] + '>=';
  end;
```

Sekcija *RHS* je vrlo slična sekciji *COLUMNS*. Ona ima *rhs name* polje na početku koje je za nas nebitno, a zatim slede četiri polja od kojih su dva opciona. To je analogno *COLUMNS* sekciji, tako da je kod skoro jednak. Potrebno je dodati vrednost desnoj strani ograničenja uz napomenu da, ako podatak nije dat, vrednost desne strane je jednaka nuli.

```
for I := 0 to High(Lines) do
  if Lines[I][Length(Lines[I])]= '='
      then Lines[I]:=Lines[I]+'0';
```

Sekcija *BOUNDS* je opciona i prevodimo je parcijalno. Prevodimo ograničenja tipa *LO, UP* i *FX*. Sledi kod:

```
readln(F);  read(F,S);
while (StrLIComp(PChar(S), S6, 6) <> 0) do
 begin //  Cycle terminates in the case S='ENDATA'
    //  Finding bound type field
    SetLength(BoundsType, 2);
    BoundsType[1] := S[2];   BoundsType[2] := S[3];
    J:=4;  //  Range name starts at fifth position
    //  Skipping bound name field
    while S[J] = ' ' do J := J + 1;

    //  Finding the content of the column name field
    //  and putting it in the ActualVar variable
    //  (like in the examples above)

    //  Finding the index K of the ActualVar in the Vars array
    K := 0;
    while(StrLIComp(PChar(Vars[K]),PChar(ActualVar),Length(ActualVar))<>0)
          do K := K + 1;

    //  Finding the content of the value field
    //  and putting it in the ActualNum variable
    //  (like in the examples above)

    // Finally we are forming the new constraint.
    ActualLin := '';
    if BoundsType='LO' then
       ActualLin:='x'+IntToStr(K+1)+'>='+ActualNum;
    if BoundsType='UP' then
       ActualLin:='x'+IntToStr(K+1)+'<='+ActualNum;
    if BoundsType='FX' then
]       ActualLin:='x'+IntToStr(K+1)+'=='+ActualNum;
    if (ActualLin <> '') then
       begin
         L := Length(Lines);
         SetLength(Lines, L+1);
         Lines[L] := ActualLin;
       end;
    Readln(F);       Read(F,S);
 end;
```

Time su formirani ciljna funkcija i sva ograničenja. Na kraju je potrebno zapamtiti oblast *Lines* kao novi fajl:

```
try
```

```
    AssignFile(E, Edit2.Text);   Rewrite(E);
    //  Objective function must be saved first
    K := 1;
    while LinesType[K] <> 'N' do K := K + 1;
    Writeln(E, WrapText(Lines[K-1]+',', #13#10, ['+','-'], 80));
    Writeln(E);   Writeln(E, Name);    Writeln(E);
    Write(E, '{');
    //  Saving all other element of the array Lines
    //  which represent the constraints
    if ((K-1) = High(Lines)) then
    begin   //  If selected line is last
      for I:=0 to (K-3) do
        Writeln(E,WrapText(Lines[I]+',',#13#10,['+','-'],80));
      Write(E,WrapText(Lines[K-2]+'}',#13#10,['+','-'],80));
    end
    else   //  Otherwise
    begin
      for I := 0 to (K-2) do
        Writeln(E,WrapText(Lines[I]+',',#13#10,['+','-'],80));
      for I := K to (High(Lines)-1) do
        Writeln(E,WrapText(Lines[I]+',',#13#10,['+','-'],80));
      Write(E,WrapText(Lines[High(Lines)]+'}',#13#10,['+','-'],80));
    end;
    CloseFile(E);
  end;
```

Napomenimo da neki *MPS* fajlovi mogu poticati pre 1995. godine. U tom slučaju program ne može da prepozna "Eoln" simbol koji predstavlja kraj linije u tekstualnom fajlu i program ne radi ništa. U tom slučaju je potrebno otvoriti fajl u nekom tekst editoru, NotePadu ili Wordu na primer, a zatim ponovo zapamtiti. Novi fajl tada ima korektne "Eoln" simbole i program sada radi ispravno.

2.4 Numerički primeri

U prvom primeru upoređujemo našu implementaciju sa nekoliko poznatih programa pisanih u različitim programskim jezicima, dok u drugom primeru ispitujemo uticaj perturbacije problema na stabilnost rešenja.

Primer 2.4.1 U ovom primeru pokazujemo visoke numeričke mogućnosti naše implementacije. Za test problem *Afiro* početna tačka ima sve koordinate jednake 500. Uobičajena tačnost u literaturi je 10^{-8}. U ovom slučaju koristimo tačnost 10^{-14}. Vrednosti mere dualnosti $x.s$ generisane tokom iteracija su

$2.27947460213494279\times10^{6}$, 296359., 3921.18, 613.909, 173.229, 42.61,
8.59576, 0.124154, 0.00124166, 0.0000124166, $1.24172039912096354\times10^{-7}$,
$1.24339209599189026\times10^{-9}$, $1.40714556006783997\times10^{-11}$, $1.39058129869647274\times10^{-12}$

i rešenje je

Out[1]= {-464.7531428571419,{$3.19975433444599843\times10^{-13}$,79.9999999999997424,

1.91244772496984012$\times 10^{-14}$,6.39852710910254973$\times 10^{-14}$,6.33933146702376237$\times 10^{-14}$,
5.77426057986956386$\times 10^{-15}$",18.2142857142854701,51.4971318189389215,
73.8941025852179755,25.5000000000002913,499.999999999995914,475.919999999998832,
24.0800000000000169,9.24725339363984843$\times 10^{-13}$,214.999999999999102,
147.740206132740929,3.5320429670772806$\times 10^{-14}$,54.4999999999994688,
3.51617861969817147$\times 10^{-14}$,3.49915810007669492$\times 10^{-14}$,2.05287088556072161$\times 10^{-14}$,
6.92504277492309583$\times 10^{-14}$,6.86331960286415032$\times 10^{-14}$,6.80075262615289499$\times 10^{-14}$,
339.942857142856658,236.202651010115571,63.5282886370786048,
5.32920502835706955$\times 10^{-15}$,84.7999999999996312,69.7114175332243757,
3.19791251290398559$\times 10^{-14}$,3.16007552845632666$\times 10^{-14}$,4.1377910630115311$\times 10^{-14}$

Za problem *Blend* smo postigli relativnu tačnost 10^{-13}, pri čemu su vrednosti mere dualnosti $x.s$ jednake

4329.02, 2295., 935.694, 73.9983, 12.7999, 3.18528,0.8102, 0.124219, 0.00416878,
0.0000455071, 4.55077$\times 10^{-7}$, 4.55079$\times 10^{-9}$, 4.55154$\times 10^{-11}$,4.55699$\times 10^{-13}$. a minimalna vrednost je -30.81214984582697.

Za problem *Adlittle* sa relativnom tačnošću 10^{-14} dobijamo minimalnu vrednost 225494.96316238158 i sledeće vrednosti za meru dualnosti $x.s$:

3.68314$\times 10^{6}$, 1.54606$\times 10^{7}$, 2.48261$\times 10^{7}$, 9.49634$\times 10^{6}$, 1.12582$\times 10^{6}$, 207828,
64916.2, 25548.8, 4728.11, 599.321,95.7551, 8.25261, 0.089176, 0.000891768.,
8.9178725141779207$\times 10^{-6}$, 8.92709050417012406$\times 10^{-8}$, 1.13837804301388501$\times 10^{-9}$,
3.77892830854748851$\times 10^{-10}$, 2.29911431090509088$\times 10^{-10}$,2.2951933148389334$\times 10^{-10}$,
2.0338064157382627$\times 10^{-10}$, 5.17798147396104369$\times 10^{-11}$.

Napomenimo da kod PCx daje minimalnu vrednost 2.25494963×10^5, koja sadrži samo tri decimalne cifre. U sledećoj tabeli dajemo rezultate dobijene primenom naše implementacije pod nazivom MPD na nekim $Netlib$ test problemima. Napomenimo da se rezultati podudaraju sa rezultatima iz [11] i slažu sa rezultatima iz [71] u skoro svim slučajevima.

Problem	*eps*	MPD optimalna vrednost	Br.it.	Najmanje $x.s$	Tačnost
Adlittle	10^{-14}	225494.96316238158	18	2.207476×10^{-11}	5.420766×10^{-15}
Afiro	10^{-14}	-464.753142857419	13	6.882627×10^{-12}	5.247988×10^{-15}
Agg	10^{-10}	-3.599176728426037$\times 10^{7}$	23	0.002223387	8.312113×10^{-11}
Agg2	10^{-12}	-2.0239252355976876$\times 10^{7}$	23	1.391321×10^{-6}	5.926817×10^{-14}
Agg3	10^{-12}	1.0312115935089154$\times 10^{7}$	21	2.253792×10^{-7}	5.599432×10^{-15}
Bandm	10^{-10}	-158.6280184446234	19	1.064918×10^{-8}	6.655994×10^{-11}
Blend	10^{-13}	-30.81214984582697	13	4.55918×10^{-13}	1.116779×10^{-14}
Degen2	10^{-8}	-1435.177999919273	11	6.52524$\times 10^{-7}$	1.81695$\times 10^{-8}$
Israel	10^{-12}	-896644.8218630119	25	3.582831×10^{-8}	5.699723×10^{-14}
Kb2	10^{-14}	-1749.900129906283	19	2.586536×10^{-13}	3.041345×10^{-13}
Lotfi	10^{-12}	-25.26470606187774	20	6.501900×10^{-12}	2.164251×10^{-13}
Recipe	10^{-12}	-266.6159999999579	14	7.790105×10^{-11}	2.480909×10^{-13}
Sc105	10^{-13}	-52.20206121170623	14	3.097975×10^{-12}	4.888131×10^{-14}
Sc205	10^{-13}	-52.20206121170889	18	1.101846×10^{-14}	3.111843×10^{-14}
Sc50a	10^{-13}	-64.5750770585606	13	1.58378×10^{-13}	6.024572×10^{-14}
Sc50b	10^{-14}	-69.9999999999994	12	1.637435×10^{-13}	1.601223×10^{-15}
Scagr7	10^{-13}	-2.3313898243313123$\times 10^{6}$	16	3.031844×10^{-8}	8.868252×10^{-14}
Scfxm1	10^{-8}	18416.759034153343	20	0.00001697	8.904402×10^{-10}
Sctap1	10^{-12}	1412.2500000000337	17	1.011259×10^{-13}	1.378802×10^{-13}
Share2b	10^{-12}	-415.73224071836	14	1.011259×10^{-13}	9.996957×10^{-13}
Stocfor1	10^{-13}	-41131.97621943486	17	1.436246×10^{-10}	4.050751×10^{-14}

Tabela 2.4.1

Najveće dimenzije matrice problema su 444×757 kod primera *Degen*2 i 516×758 kod primera *Agg*2 i *Agg*3.

U sledećoj tabeli dajemo odgovarajuće rezultate dobijene primenom poznatih programa PCx [14], LOQO [105], LIPSOL [123] i MATHEMATICA implementacijom iz [10]. U implementaciji iz [10] koristili smo samo definisanu relativnu tačnost 10^{-4}, zbog ekstremno velikog broja iteracija.

Problem	PCx	LIPSOL	LOQO	BHATTI
Adlittle	2.25494963×10^{5}	2.2549496316×10^{5}	2.2549496×10^{5}	223753
Afiro	-4.64753143×10^{2}	$-4.6475314219\times10^{2}$	-4.6475314×10^{2}	-464.713
Agg	3.59917673×10^{7}	-3.599176727×10^{7}	-3.5991767×10^{7}	-3.59892×10^{7}
Agg2	-2.02392521×10^{7}	$-2.0239252353\times10^{7}$	-2.0239252×10^{7}	$-2.02715e\times10^{7}$
Agg3	1.03121159×10^{7}	1.0312115935×10^{7}	1.0312116×10^{7}	1.02797×10^{7}
Bandm	-1.58628018×10^{2}	$-1.5862801844\times10^{2}$	-1.5862802×10^{2}	-158.628
Blend	-3.08121498×10^{1}	$-3.0812149846\times10^{1}$	-3.0812150×10^{1}	-30.7843
Degen2	-1.43517800×10^{3}	$-1.4351780000\times10^{3}$	-1.4351780×10^{3}	-1435.17
Israel	-8.96644817×10^{5}	$-8.9664482186\times10^{5}$	-8.9664465×10^{5}	-896617
Kb2	-1.74990013×10^{3}	$-1.7499001299\times10^{3}$	-1.7499001×10^{3}	-1749.89
Lotfi	$-2.5264706062\times10^{1}$	$-2.5264706056\times10^{1}$	-2.5264702×10^{1}	-25.2551
Recipe	-2.66616000×10^{2}	$-2.6661600000\times10^{2}$	-2.6661600×10^{2}	–
Sc105	$-5.2202061212\times10^{1}$	$-5.2202061212\times10^{1}$	-5.2202060×10^{1}	-52.1908
Sc205	-5.22020612×10^{1}	$-5.2202061119\times10^{1}$	-5.2202061×10^{1}	-52.1876
Sc50a	$-6.4575077059\times10^{1}$	$-6.4575077059\times10^{1}$	-6.4575077×10^{1}	-64.5683
Sc50b	-7.000000000×10^{1}	$-6.9999999976\times10^{1}$	-7.00000006×10^{1}	-69.993
Scagr7	-2.33138982×10^{6}	$-2.3313898243\times10^{6}$	-2.3313898×10^{6}	-2.33139×10^{6}
Scfxm1	1.84167598×10^{4}	1.8416759028×10^{4}	1.8416759×10^{4}	18416.7
Sctap1	1.41225000×10^{3}	1.4122500000×10^{3}	1.4122500×10^{3}	1412.26
Share2b	$-4.1573224074\times10^{2}$	$-4.1573224024\times10^{2}$	-4.1573224×10^{2}	-415.732
Stocfor1	$-4.1131976219\times10^{4}$	$-4.1131976219\times10^{4}$	-4.1131976×10^{4}	-41132

Tabela 2.4.2

Napomenimo da program PCx u praksi kod većine problema ne može da postigne preciznost veću od 10^{-11}.

U sedećoj tabeli upoređujemo broj potrebnih iteracija u slučaju kada je zadata tačnost 10^{-8} u svim implementacijama, osim kod implementacije iz [10] gde se koristi tačnost 10^{-4} iz već navedenih razloga. Evidentno je da sve implementacije primal dual algoritma imaju približno isti broj iteracija, osim implementacije iz [10] koja i pored znatno manje preciznosti zahteva daleko veći broj iteracija.

Problem	MPD	PCx	LIPSOL	LOQO	BHATI
Adlittle	13	12	12	16	50
Afiro	9	8	7	13	14
Agg	22	19	18	23	96
Agg2	21	23	16	33	50
Agg3	19	22	16	37	60
Bandm	16	17	17	20	86
Blend	11	10	12	17	14
Degen2	11	12	14	18	16
Israel	23	21	22	33	657
Kb2	15	13	15	18	65
Lotfi	18	15	18	21	82

Problem	MPD	PCx	LIPSOL	LOQO	BHATI
Recipe	10	9	9	14	–
Sc105	12	10	10	15	38
Sc205	15	11	11	17	72
Sc50a	10	8	10	15	22
Sc50b	9	6	7	13	25
Scagr7	14	14	12	18	105
Scfxm1	20	17	18	26	172
Sctap1	15	16	17	22	41
Share2b	11	17	12	16	33
Stocfor1	14	12	17	19	44

Tabela 2.4.3

3 Smanjenje dimenzije u Mehrotraovom algoritmu

U ovom odeljku opisujemo modifikaciju Mehrotraovog primal-dual algoritma. Ta modifikacija smanjuje dimenziju problema, poboljšava stabilnost metoda i eliminiše potrebu za svođenjem rešenja na tačno. Takođe je realizovana implementacija modifikacije u programu MATHEMATICA i dati su numerički primeri za poređenje sa originalnim algoritmom. Ovi rezultati su publikovani u radovima [93, 95, 96, 97].

3.1 Primal-dual algortimi i bazično rešenje

Primal-dual algoritmi razmatraju problem linearnog programiranja sa njegovim dualnim problemom (1.1.1) – (1.1.2). Pri tome se generiše niz koji konvergira ka optimalnom rešenju tako da je potrebno konačno rešenje svesti posebnim postupkom na tačno. Napomenimo da se često u praktičnoj primeni linearnog programiranja rešava niz sličnih problema. Pri tome je prethodno optimalno rešenje kod simpleks algoritma moguće upotrebiti za početno rešenje i tako dobiti novo optimalno rešenje brže i jednostavnije. Za unutrašnje metode takva opšta procedura još uvek ne postoji mada postoje takvi pokušaji [27, 29, 30, 2]. Trenutno usvojen pristup je da se niz sličnih problema prvo reši primenom metoda unutrašnje tačke, a zatim se prelazi na simleks metod. Pri tome se koriste prednosti oba metoda. Algoritam za generisanje optimalnog bazičnog rešenja je dat u [67]. On konstruiše optimalno bazično rešenje u manje od n iteracija startujući iz nekog komplementarnog rešenja; dakle, Megiddov algoritam podrazumeva da je poznato tačno optimalno rešenje. Ta pretpostavka u praksi nikad nije ispunjena jer primal-dual algoritam samo generiše niz koji konvergira ka optimalnom rešenju. Kako se rešenje uvek dobija sa unapred zadatom tačnošću, rešenje nije ni tačno dopustivo ni komplementarno. Postupak konačnog završavanja koji su predložili Andersen i Ye [3], [117] se zasniva na prelasku sa iteracije koja prati centralnu putanju na tačno primal-dual rešenje. Algoritam je uspešan ako je iteracija dovoljno bliska optimalnom rešenju tako da indeksni skupovi $\mathcal{B}$ i $\mathcal{N}$ mogu biti tačno određeni. U suprotnom, algoritam daje tačku

koja ne zadovoljva neko od ograničenja u (1.1.1), (1.1.2) i u tom slučaju se algoritam vraća u primal-dual algoritam da bi se kroz nekoliko koraka ponovo pokušalo sa konačnim završavanjem. Smanjenjem dimenzije problema moguće je pojednostaviti postupak konačnog završavanja. Napomenimo da je naglasak modifikacije na tome da se implemtira tako da se dobije bazično optimalno rešenje. Postupak konačnog završavanja je vrlo detaljno razmatran u velikom broju radova [3, 69, 107, 112, 117].

3.2 Redukcija dimenzija problema linearnog programiranja

Deo rezultata iz ovog poglavlja publikovan je u [93].

U nastavku razmatramo problem linearnog programiranja u opštem obliku

$$\begin{aligned} \text{minimizirati} \quad & c_1x_1 + \cdots + c_kx_k \\ \text{pod uslovima} \quad & \sum_{j=1}^{k} a_{ij}x_j \le b_i, \quad i = 1, \ldots, q, \\ & \sum_{j=1}^{k} a_{ij}x_j = b_i, \quad i = q+1, \ldots, m, \\ & x_i \ge 0, \quad i = 1, \ldots, k. \end{aligned} \tag{3.2.1}$$

Standardna forma za (3.2.1) je (sa $k + q = n$)

$$\begin{aligned} \text{minimizirati} \quad & c_1x_1 + \cdots + c_kx_k + c_{k+1}x_{k+1} + \cdots + c_nx_n \\ \text{pod uslovima} \quad & \sum_{j=1}^{k} a_{ij}x_j + x_{k+i} = b_i, \quad i = 1, \ldots, q, \\ & \sum_{j=1}^{k} a_{ij}x_j \qquad\quad = b_i, \quad i = q+1, \ldots, m, \\ & x_i \ge 0, \quad i = 1, \ldots, n. \end{aligned} \tag{3.2.2}$$

Problem (3.2.2) je oblika (1.1.1), gde je $n = k + q$ i

$$c_{k+1} = \cdots = c_{k+q} = 0, \quad A = \begin{bmatrix} a_{11} & \ldots & a_{1k} & 1 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{q1} & \ldots & a_{qk} & 0 & \ldots & 1 \\ a_{q+1,1} & \ldots & a_{q+1,k} & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & \ldots & a_{mk} & 0 & \ldots & 0 \end{bmatrix}. \tag{3.2.3}$$

Dualni problem za (3.2.2) je

$$\begin{aligned} &\text{maksimizirati} \;\; b_1\lambda_1 + \cdots + b_m\lambda_m \\ &\text{pod uslovima} \;\; \sum_{j=1}^{m} a_{ji}\lambda_j = c_i, \;\; i = 1, \ldots, k \\ &\qquad\qquad \lambda_j + s_{k+j} = 0, \;\; j = 1, \ldots, q \\ &\qquad\qquad\qquad\qquad s_i \geq 0, \;\; i = 1, \ldots, k+q. \end{aligned} \tag{3.2.4}$$

Primetimo da ako je $x_i = 0$ za neko i, tada to x_i nema uticaja na konačno rešenje, tako da i-ta kolona iz matrice A može biti izostavljena. Takođe, ako je $s_{k+j} = 0$ za neko j, tada iz (3.2.4) sledi da je $\lambda_j = 0$, tako da j-ta kolona i $(k+j)$-ta vrsta iz matrice A^T može biti izostavljena. Te činjenice povlače sledeća dva tvrđenja. Zbog jednostavnosti koristimo sledeće oznake. Neka su $\alpha = \{\alpha_1, \ldots, \alpha_p\}$ i $\beta = \{\beta_1, \ldots, \beta_q\}$ podskupovi od $\{1, \ldots, m\}$ i $\{1, \ldots, n\}$, respektivno, za neke cele brojeve $1 \leq p \leq m$ i $1 \leq q \leq n$. Sa A^α označavamo $p \times n$ podmatricu od A određenu vrstama koje su indeksirane skupom α. Slično, sa A_β označavamo $m \times q$ podmatricu od A određenu sa kolonama koje su indeksirane skupom β.

Lema 3.2.1 [*93*] *Razmotrimo problem linearnog programiranja* (3.2.2-3.2.4). *Pretpostavimo da je* $x_{i_1} = \cdots = x_{i_d} = 0$, *i razmotrimo skup indeksa* $I = \{i_1, \ldots, i_d\} \subseteq \{1, \ldots, n\}$. *Označimo sa* $\hat{A} \in \mathbb{R}^{m \times (n-d)}$ *matricu* $A_{\{1,\ldots,n\}\setminus I}$, *i neka su* $\hat{x}, \hat{c}, \hat{s} \in \mathbb{R}^{n-d}$, *respektivno, vektori* x, c *i* s *bez* $i-te$ $(i \in I)$ *koordinate. Sa* $\varphi_i(\lambda)$ *označimo linearnu funkciju*

$$\varphi_i(\lambda) = c_i - (a_{1i}\lambda_1 + \cdots + a_{mi}\lambda_m), \quad i \in I.$$

Tada je primal-dualni problem (3.2.2-3.2.4) *ekvivalentan problemu*

$$\begin{aligned} &\textit{minimizirati} \;\; \hat{c}^T\hat{x} \;\; \textit{pod uslovima} \;\; \hat{A}\hat{x} = b, \quad \hat{x} \geq 0, \;\; x_i = 0, \;\; i \in I, \\ &\textit{maksimizirati} \;\; b^T\lambda \;\; \textit{pod uslovima} \;\; \hat{A}^T\lambda + \hat{s} = \hat{c}, \quad \hat{s} \geq 0, \;\; s_i = \varphi_i(\lambda), \;\; i \in I, \end{aligned}$$

gde je $\hat{A}^T$ *transponovana matrica matrice* $\hat{A}$.

Dokaz. Dovoljno je dokazati Lemu za jedan indeks $i \in I$. Pretpostavimo sada da je $x_i = 0$. Koristimo sledeće oznake:

$$\begin{aligned} \hat{A}^i &= [A_1 \ldots A_{i-1} A_{i+1} \ldots A_n], \\ \hat{x}^i &= (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n), \\ \hat{c}^i &= (c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n), \\ \hat{s}^i &= (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n). \end{aligned}$$

Sada je (3.2.2) ekvivalentno sa $x_i = 0$ i

$$\min(\hat{c}^i)^T\hat{x}^i \text{ pod uslovima } \hat{A}^i\hat{x}^i = b, \quad \hat{x}^i \geq 0, \tag{3.2.5}$$

Dualni problem za (3.2.5) je

$$\max b^T y \text{ pod uslovima } (\hat{A}^i)^T y + \hat{s}^i = \hat{c}^i, \quad \hat{s}^i \geq 0. \tag{3.2.6}$$

Sada je (3.2.6) i

$$s_i = c_i - (a_{1i}y_1 + \cdots + a_{mi}y_m) = \varphi_i(y)$$

ekvivalentno sa (3.2.4). □

Lema 3.2.2 [93] *Pretpostavimo da u primal-dual problemu* (3.2.2)-(3.2.4) *važe jednakosti* $s_{k+j_1} = \cdots = s_{k+j_h} = 0$. *Razmotrimo skupove* $J = \{j_1, \ldots, j_h\} \subseteq \{1, \ldots, q\}$ *i* $J^k = \{k + j | \, j \in J\}$. *Označimo sa* $\hat{A}^T \in \mathbb{R}^{(n-h)\times(m-h)}$ *matricu*

$$(A^T)^{\{1,\ldots,n\}\setminus J^k}_{\{1,\ldots,m\}\setminus J} = \left(A^{\{1,\ldots,m\}\setminus J}_{\{1,\ldots,n\}\setminus J^k}\right)^T.$$

Neka su $\hat{x}, \hat{c}, \hat{s} \in \mathbb{R}^{n-h}$, *respektivno, vektori* x, c, s *bez* $(k + j)$*-te* $(j \in J)$ *koordinate, i neka su* $\hat{\lambda}, \hat{b} \in \mathbb{R}^{m-h}$, *respektivno, vektori* λ, b *bez* j*-te* $(j \in J)$ *koordinate. Sa* $\psi_j(\hat{x})$ *označimo linearnu funkciju*

$$\psi_j(\hat{x}) = b_j - (a_{j1}x_1 + \cdots + a_{jn}x_n), \quad j \in J.$$

Tada je primal-dual problem (3.2.2)-(3.2.4) *ekvivalentan sa*

$$\begin{aligned}
&\textit{minimizirati } \hat{c}^T\hat{x} \textit{ pod uslovima } \hat{A}\hat{x} = \hat{b}, \quad \hat{x} \geq 0, \; x_{k+j} = \psi_j(\hat{x}), \; j \in J,\\
&\textit{maksimizirati } \hat{b}^T\hat{\lambda} \textit{ pod uslovima } \hat{A}^T\hat{\lambda} + \hat{s} = \hat{c}, \quad \hat{s} \geq 0, \; s_{k+j} = 0, \; j \in J,
\end{aligned}$$

gde je $\hat{A} = A^{\{1,\ldots,m\}\setminus J}_{\{1,\ldots,n\}\setminus J^k}$.

Dokaz. Dovoljno je dokazati Lemu za jedno $j \in J$. Pretpostavimo sada da je $s_{l+j} = 0$ za neko $1 \leq j \leq m$. Zbog (3.2.3) je $y_j = 0$, i (3.2.4) je ekvivalentno sa

$$\max \hat{b}^T\hat{y} \text{ pod uslovima } \hat{A}^T\hat{y} + \hat{s} = \hat{c}, \quad \hat{s} \geq 0, \tag{3.2.7}$$

gde je

$$\begin{aligned}
(\hat{A}^j)^T &= [A_1^T \cdots A_{j-1}^T A_{j+1}^T \cdots A_m^T],\\
\hat{y}^j &= (y_1, \ldots, y_{j-1}, y_{j+1}, \ldots, y_m),\\
\hat{b}^j &= (b_1, \ldots, b_{j-1}, b_{j+1}, \ldots, b_m),\\
\hat{s}^j &= (s_1, \ldots, s_{l+j-1}, s_{l+j+1}, \ldots, s_n),\\
\hat{c}^j &= (c_1, \ldots, c_{l+j-1}, c_{l+j+1}, \ldots, c_n).
\end{aligned}$$

Primalni problem za (3.2.7) je

$$\min(\hat{c}^j)^T\hat{x}^j \text{ pod uslovima } \hat{A}^j\hat{x}^j = b, \quad \hat{x}^j \geq 0, \tag{3.2.8}$$

gde je

$$\hat{x}^j = (x_1, \ldots, x_{l+j-1}, x_{l+j+1}, \ldots, x_n).$$

Kako je (3.2.2) ekvivalentno sa (3.2.8) i

$$x_{l+j} = b_j - (a_{j1}x_1 + \cdots + a_{jl}x_l) = \psi_j(\hat{x}),$$

dokaz sledi. □

Sledeće tvrđenje objedinjuje prethodne dve leme.

Teorema 3.2.1 [*93*] *Neka su* $x_{i_1} = \cdots = x_{i_d} = 0$ *i* $s_{k+j_1} = \cdots = s_{k+j_h} = 0$, *gde su skupovi* $I = \{i_1, \ldots, i_d\}$ *and* $J = \{j_1, \ldots, j_h\}$ *definisani kao u Lemi 3.2.1 i Lemi 3.2.2, respektivno. Sa oznakama iz Leme 3.2.1 i Leme 3.2.2 primal-dual problem* (3.2.2-3.2.4) *je ekvivalentan sa*

$$\begin{aligned} &\textit{minimizirati} \;\; \hat{c}^T\hat{x} \;\; \textit{pod uslovima} \\ &\qquad \hat{A}\hat{x} = \hat{b}, \hat{x} \geq 0,\ x_i = 0,\ i \in I,\ x_{k+j} = \psi_j(\hat{x}),\ j \in J, \\ &\textit{maksimizirati} \;\; \hat{b}^T\hat{\lambda} \;\; \textit{pod uslovima} \\ &\qquad \hat{A}^T\hat{\lambda} + \hat{s} = \hat{c},\ \hat{s} \geq 0,\ s_{k+j} = 0,\ j \in J,\ s_i = \varphi_i(\hat{\lambda}),\ i \in I, \end{aligned}$$

gde je $\hat{A} \in \mathbb{R}^{(m-h)\times(n-h-d)}$, $\hat{x}, \hat{c}, \hat{s} \in \mathbb{R}^{n-h-d}$, $\hat{\lambda}, \hat{b} \in \mathbb{R}^{m-h}$.

Dokaz. Kako je $I \cap J^k = \emptyset$, za striktno komplementarno rešenje, dokaz sledi direktno iz Leme 3.2.1 i Leme 3.2.2. □

Napomena 3.2.1 *Teorema* 3.2.1 *je uopštenje Teoreme 3.1 iz rada* [*94*], *deo (b).*

Napomenimo da za proizvoljnu tačku $(x, y, s) \in \mathcal{N}_{-\infty}(\gamma)$, možemo proceniti skupove $\mathcal{B}$ i $\mathcal{N}$ na sledeći način [112]:

$$\begin{aligned} \mathcal{B}(x,s) &= \{i \in \{1, \ldots, n\} |\, x_i \geq s_i\}, \\ \mathcal{N}(x,s) &= \{1, \ldots, n\} \backslash \mathcal{B}(x,s), \end{aligned} \tag{3.2.9}$$

gde je

$$\mathcal{N}_{-\infty}(\gamma) = \{(x, y, s) \in \mathcal{F}^0 |\, x_i s_i \geq \gamma\mu \text{ za svako } i = 1, \ldots, n\}$$

pri čemu je $\gamma \in (0, 1)$.

Sledeći algoritam konačnog završavanja (Algotritam FT) je predložio Ye [117].

Algoritam FT
Dato je $\gamma \in (0, 1)$ i $(x, y, s) \in \mathcal{N}_{-\infty}(\gamma)$:
Odrediti skupove $\mathcal{B}(x, s)$ i $\mathcal{N}(x, s)$;
Rešiti sledeći problem:

$$\begin{aligned} \min_{(x^*, y^*, s^*)} &\frac{1}{2}|x^* - x|^2 + \frac{1}{2}|s^* - s|^2, \\ &Ax^* = b, \quad A^T y^* + s^* = c, \\ x_i^* = 0 \text{ for } i \in \mathcal{N}(x,s), &\quad s_i^* = 0 \text{ for } i \in \mathcal{B}(x,s). \end{aligned} \tag{3.2.10}$$

If $x_{\mathcal{B}}^* > 0$ i $s_{\mathcal{N}}^* > 0$

zaokruživanje je uspešno: (x^*, y^*, s^*) je striktno komplementarno rešenje;
else
zaokruživanje je neuspešno i vraćamo se u primal-dual algoritam.

Sledeći rezultat iz [112] pokazuje da je uspešnost Algoritma FT garantovana kada je μ dovoljno malo.

Teorema 3.2.2 *Neka je $\gamma \in (0,1)$ dato. Tada postoji granična vrednost $\bar{\mu}$ tako da za sve (x, y, s) koje zadovoljavaju*

$$(x, y, s) \in \mathcal{N}_{-\infty}(\gamma), \quad 0 < \mu = x^T s/n \leq \bar{\mu},$$

važi

(i) $\mathcal{B}(x, s) = \mathcal{B}$ i $\mathcal{N}(x, s) = \mathcal{N}$; što znači da su konkretni skupovi $\mathcal{B}$ i $\mathcal{N}$ korektno procenjeni postupkom (3.2.9)*;*

(ii) procedura (3.2.10) *daje striktno komplementarno rešenje (x^*, y^*, s^*).*

Sada predlažemo sledeću modifikaciju Algoritma FT. Neka je $\varepsilon > 0$ dato. Koristeći (3.2.9) procenimo skupove $\mathcal{B}$ i $\mathcal{N}$. Stavimo

$$I = \{\, i \,|\, x_i < \varepsilon,\ i \in \mathcal{N}\}, \quad J = \{\, j \,|\, s_{l+j} < \varepsilon,\ l + j \in \mathcal{B}\}.$$

u Teoremi 3.2.1 i ako je

$$x_{l+j} = \psi_j(\hat{x}) > 0,\ j \in J, \text{ i } s_i = \varphi_i(\hat{y}) > 0,\ i \in I,$$

primenimo Algoritam FT na primal-dual problem

$$\min \hat{c}^T \hat{x} \text{ pod uslovom } \hat{A}\hat{x} = \hat{b},$$
$$\max \hat{b}^T \hat{y} \text{ pod uslovom } \hat{A}^T \hat{y} + \hat{s} = \hat{c}.$$

Primetimo da u ovoj varijanti algoritma redukujemo primal-dual problem sa matricom $A_{m\times n}$ na primal-dual problem sa matricom $\hat{A}_{(m-r)\times(n-q)}$. Istu ideju ćemo u nastavku sprovesti u toku same implementacije primal-dual algoritma.

3.3 Redukcija dimenzije i potencijalna funkcija

U ovom odeljku pokazujemo uticaj redukcije dimenzije na primal-dual potencijalnu funkciju. Potencijal-redukcioni metodi koriste logaritamsku potencijanu funkciju kao kriterijum za svaku tačku iz $\mathcal{F}^0$ i pri tome u svakom koraku sledeću iteraciju biraju tako da se potencijalna funkcija smanji za datu fiksiranu vrednost [104]. Primer primal-dual potencijalne funkcije je Tanabe-Todd-Ye potencijalna funkcija

$$\Phi_\rho(x, s) = \rho \log x^T s - \sum_{i=1}^{n} \log x_i s_i \tag{3.3.11}$$

za neki parametar $\rho > n$. Potencijal-redukcioni algoritmi traženi pravac dobijaju rešavanjem (3.3.11) za neko $\sigma_k \in (0,1)$. Dužina koraka se bira tako da aproksimativno minimizira Φ_ρ duž dobijenog pravca.

U sledećoj teoremi ispitujemo uticaj redukcije problema pod uslovima Teoreme 3.2.1 na funkciju Φ_ρ.

Teorema 3.3.1 *Pretpostavimo da iterativni niz ne prati centralnu putanju i pretpostavimo da je $x_j s_j < \varepsilon < 1$ i $x_i s_i \geq \varepsilon$, $i \neq j$. Ako je moguće eliminisati x_j, za proizvoljno j (ili s_j, za neko $j > k$) i eliminisati j-tu kolonu iz matrice A (j-tu vrstu i $(j-k)$-tu kolonu iz matrice A^T), tada potencijal-redukciona funkcija se približno smanjuje za* $\log\left(\dfrac{n}{n-1}\right)^\rho - \log x_j s_j > 0$.

Dokaz. Dokaz Primenom Leme 3.2.1 (Leme 3.2.2) možemo eliminisati x_j, (s_j) i j-tu kolonu iz matrice A (j-tu vrstu i $(j-k)$-tu kolonu iz matrice A^T). Neka je

$$\hat{\Phi}_\rho(\hat{x}, \hat{s}) = \rho \log \hat{x}^T \hat{s} - \sum_{i \neq j} \log x_i s_i$$

potencijal-redukciona funkcija redukovanog problema, gde $\hat{x}$ i $\hat{s}$ označavaju vektore x i s bez j-tih elemenata, respektivno. Tada važi

$$\begin{aligned}
\Phi_\rho(x,s) - \hat{\Phi}_\rho(\hat{x},\hat{s}) &= \rho(\log x^T s - \log \hat{x}^T \hat{s}) - \log x_j s_j \\
&= \rho \log\left(1 + \frac{x_j s_j}{\sum_{i \neq j} x_i s_i}\right) - \log x_j s_j \\
&\geq \rho \log\left(1 + \frac{\varepsilon}{(n-1)\varepsilon}\right) - \log x_j s_j \\
&\geq \log\left(\frac{n}{n-1}\right)^\rho - \log x_j s_j > \log\left(\frac{n}{n-1}\right)^\rho - \log \varepsilon > 0.
\end{aligned}$$

Dokaz je kompletiran. □

4 Modifikovani algoritam i implementacija

U ovom odeljku razmatramo smanjenje dimenzije problema u toku izvršenja Mehrotraovog primal-dual algoritma. Predlažemo sledeće modifikacije *Koraka 12* i *Koraka 2*. Modifikacija *Koraka 12* se bazira na eliminaciji promenljivih x_i i s_i koje konvergiraju ka nuli, koja se zasniva na Teoremi 3.2.1.

4.1 Opis algoritma

Za tačku $(x, \lambda, s) \in \mathcal{N}_{-\infty}(\gamma)$, možemo proceniti skupove $\mathcal{B}$ i $\mathcal{N}$ kada je μ dovoljno malo na osnovu (3.2.9). Takođe je poznato da postoji granična vrednost $\overline{\mu}$ tako da za svako (x, λ, s) koje zadovoljava

$$(x, \lambda, s) \in \mathcal{N}_{-\infty}(\gamma) \subset \mathcal{F}^0, \quad 0 < \mu = x^T s/n \leq \overline{\mu}, \tag{4.1.1}$$

važi $\mathcal{B}(x,s)=\mathcal{B}$ and $\mathcal{N}(x,s)=\mathcal{N}$, tj. indeksni skupovi $\mathcal{B}$ i $\mathcal{N}$ su korektno procenjeni sa (3.2.9) [112].

Potrebni su nam mali realni brojevi $eps1$, $eps2$ and $eps3$ kao kriterijum za primenu redukcije predložene u Teoremi 3.2.1. Motivacija za primenu tih krirterijuma je u proceni (3.2.9) i (4.1.1).

Koristimo vektor *xzero* za pamćenje indeksa eliminisanih elemenata vektora x. Prazna lista je početna vrednost za vektor *xzero*.

Korak 12M. Izračunati sledeću iteraciju

$$x=x^{t+1}=x^t+\alpha_t^{pri}\Delta x^t,$$
$$(l,s)=(\lambda^{t+1},s^{t+1})=(\lambda^t,s^t)+\alpha_t^{dual}(\Delta\lambda^t,\Delta s^t).$$

Ako je uslov $xs=x^Ts<eps3$ zadovoljen, izvršiti dve petlje, *Petlju 1* i *Petlju 2*, koje su definisane na sledeći način.

Petlja 1. Za svako $i=1,\ldots,k$, ako je uslov

$$x_i=x_i^{t+1}<eps1,\quad x_i\le s_i=s_i^{t+1} \tag{4.1.2}$$

zadovoljen, izvršiti sledeće korake 1.1-1.4.

1.1. Obrisati i-ti element iz vektora x, c i s, i obrisati i-tu kolonu iz matrice A.

1.2. Postaviti $k=k-1$.

1.3. Ubaciti indeks i na početak vektora *xzero*.

1.4. Postaviti $i=i-1$.

Odgovarajući kod je:

```
For[i=1,i<=k,i++,
   cut=(x[[i]]<eps1) && (x[[i]]<s[[i]]);
   If[cut,
     x=Delete[x,i]; ce=Delete[ce,i]; s=Delete[s,i];
     a=Transpose[Delete[Transpose[a],i]];
     k--;
     xzero=Prepend[xzero,i];
     i--;
];
```

Petlja 2. za svako $i=k+1,\ldots,k+q=n$ izvršiti korake 2.1-2.4.

2.1. Postaviti $False$ kao vrednost logičke promenljive lg. Ta promenljiva se koristi kao indikator da je neki element iz x obrisan.

2.2. Pod uslovima (4.1.2) izvršiti sledeće:

2.2.1. Postaviti $lg=True$.
2.2.2. Obrisati i-ti element iz x, c, s i i-tu kolonu iz matrice A.
2.2.3. Postaviti $q=q-1$.
2.2.4. Zamenimo $(i-k)$-ti i poslednji element u A, x i $l=\lambda$.

2.3. U slučaju da je $i \leq Length[s]$, ako je uslov

$$lg = False, \quad s_i = s_i^{t+1} < eps2, \quad s_i \leq x_i = x_i^{t+1} \tag{4.1.3}$$

zadovoljen, izvršiti sledeće korake:

2.3.1. Obrisati i-ti element iz x, c i s, obrisati $(i-k)$-ti element iz b i $l = \lambda$, i obrisati $(i-k)$-tu vrstu i i-tu kolonu iz matrice A.

2.3.2. Postaviti $q = q - 1$.

2.4. U slučju da je $lg = True$ postaviti $i = i - 1$.

Konačno, kao i u klasičnom metodu, postaviti $(x^t, \lambda^t, s^t) = (x^{t+1}, \lambda^{t+1}, s^{t+1})$ i preći na *Korak 2M*, modifikaciju *Koraka 2*.

Odgovarajući kôd je:

```
For[i=k+1,i<=k+q,i++,
  lg=False;
  cut=(x[[i]]<eps1) && (x[[i]]<s[[i]]) && (i<=Length[x]);
    If[cut,
      lg=True;
      x=Delete[x,i]; ce=Delete[ce,i]; s=Delete[s,i];
      a=Transpose[Delete[Transpose[a],i]];
      q--;
      xzero=Prepend[xzero,i];
      pp=a[[i-k]]; a=Delete[a,i-k]; AppendTo[a,pp];
      pp=b[[i-k]]; b=Delete[b,i-k]; AppendTo[b,pp];
      pp=l[[i-k]]; l=Delete[l,i-k]; AppendTo[l,pp];
    ];
    If[i<=Length[s],
      cut=(s[[i]]<eps2) && (s[[i]]<x[[i]]);
      If[!lg && cut,
       lg=True;
        x=Delete[x,i]; ce=Delete[ce,i]; s=Delete[s,i];
        a=Delete[a,i-k]; b=Delete[b,i-k]; l=Delete[l,i-k];
        a=Transpose[Delete[Transpose[a],i]];
        xzero=Prepend[xzero,i];
        q--;
    ];];
    If[lg, i--];
];
```

Korak 2M. Proveriti kriterijum za zaustavljanje kao u *Koraku 2*. Ako taj kriterijum nije ispunjen, preći na *Korak 3*. Kada je kriterijum zadovoljen, izvršimo rekonstrukciju vektora x^t, umetanjem nula na odgovarajućim pozicijama vektora x^t, gde su odgovarajući elementi obrisani. Za tu svrhu koristimo listu $xzero$ i primenimo sledeću For petlju:

Za svako $i = 1, \ldots, n = Length[xzero]$ postaviti nulu na i-tu poziciju vectora x^t.

Nakon toga vratimo x kao izlaz.

To se postiže sledećim kodom

```
For[i=1, i<=Length[xzero],i++,
      x=Insert[x,0,xzero[[i]]];
];
```

Napomena 4.1.1 Važan praktičan problem je izbor indikatora $\mathcal{B}(x,s)$ za optimalnu particiju $\mathcal{B}$. Osim kriterijuma (3.2.9) i (4.1.1), moguće je primeniti i druge kriterijume za procenu indeksnih skupova.

1. Indikator (3.2.9) nije invarijantan u odnosu na skaliranje kolona. Alternativni indikator je definisan u [24]:

$$\mathcal{B}(x,s) = \{i \in \{1,\ldots,n\} | \, (|\Delta x_i^{aff}|/x_i) \leq (|\Delta s_i^{aff}|/s_i)\},$$

gde je $(\Delta x_i^{aff}, \Delta s_i^{aff})$ afini pravac. Taj indikator je invarijantan u odnosu na skaliranje kolona [24, 2].

Jedina modifikacija u prethodnom *Koraku 12M* je sledeća. U *Petlji 1* i u *Koraku 2.2 Petlje 2* zamenimo uslov (4.1.2) uslovom

$$\left|\Delta x_i^{aff}\right|/x_i > \left|\Delta s_i^{aff}\right|/s_i, \quad x_i < eps1,$$

što u programu glasi

$$Abs[Dxaff[[i]]]/x[[i]] > Abs[Dsaff[[i]]]/s[[i]].$$

Takođe, u *Koraku 2.3 Petlje 2* zamenimo uslov (4.1.3) sa

$$\left|\Delta x_i^{aff}\right|/x_i < \left|\Delta s_i^{aff}\right|/s_i, \quad s_i < eps2.$$

Primena ovih kriterijuma daje numeričke rezultate koji su vrlo slični rezultatima koji se dobijaju posle primene uslova (4.1.2) i (4.1.3).

2. Umesto kriterijuma $xs = x^T s < eps3$, koji dozvoljava redukciju u *Koraku 12M*, takođe smo koristili alternativni kriterijum $x^T s/x1^T s1 = r < eps3$, gde su $x1$ i $s1$ vrednosti za x i s u prethodnom iterativnom koraku. Naša numerička iskustva pokazuju da većina testiranih problema u poslednjim koracima iterativnog procesa uzima vrednost $r \approx 0.01$. To znači da je superlinearna konvergencija dostignuta za vrednost $eps3 = 0.01$. Ali, neki od tih problema poseduju sasvim zadovoljavajuću konvergenciju čak i za vrednosti $eps3 = 0.1$. Posle primene tog kriterijuma dobili smo gotovo iste rezultate kao i sa prethodnim kriterijumom.

Napomena 4.1.2 Primenom Teoreme 3.2.1 možemo da redukujemo dimenziju primal-dualnog linearnog problema (3.2.2) – (3.2.4). Takođe, time se poboljšava stabilnost i centralnost iterativnog niza. To sledi iz sledećih razmatranja. U *Koraku 4* svake primal-dual iteracije neophodno je rešiti linearni sistem (2.1.1). Takođe,

u *Koraku 8* je neophodno rešiti sistem (2.1.2). Primena Teoreme 3.2.1 redukuje sistem u (2.1.1) sa matricom tipa $A_{m\times n}$ u ekvialentan sistem sa matricom manje dimenzije $\hat{A}_{(m-h)\times(n-h-d)}$. Smanjenje dimenzije problema nam daje pogodnosti u izračunavanjima i potrebnom memorijskom prostoru. Jasno je da je modifikacija posebno efikasna za probleme linearnog programiranja velikih dimanzija. Takođe, poseban problem stabilnosti u (2.1.1) nastaje zbog prisustva veoma malih (i respektivno veoma velikih) dijagonalnih elemenata u D^2 i D^{-2}. U skladu sa Teoremom 3.2.1 moguće je eliminisati neke od tih elemenata i poboljšati stabilnost i centralnost iterativnog niza.

Napomena 4.1.3 Postavljanje $x_i = 0$, $i \in I$, $s_{k+j} = 0$, $j \in J$ u *Koraku 12M* je analogno postavljanju $x_i^* = 0$, $i \in \mathcal{N}(x,s)$, $s_i^* = 0$, $i \in \mathcal{B}(x,s)$ što je poznato iz algoritma konačnog završavanja. Naša strategija se razlikuje od klasičnog algoritma konačnog završavanja jer se ne procenjuju svi elementi iz skupova $\mathcal{N}$ i $\mathcal{B}$. Pored toga, mi primenjujemo tu strategiju sukscesivno tokom iteracija, vršeći pri tome samo jednostavnu verifikaciju.

4.2 Numerički primeri

U prvom primeru upoređujemo odnos procesorskih vremena za modifikovani i originalni metod dok u drugom primeru pokazujemo da postoje problemi malih dimenzija koje modifikacija uspešno rešava za razliku od originalnog metoda koji te probleme ne može da reši sa zadatom tačnošću.

Primer 4.2.1 U ovom primeru razmatramo nekoliko poznatih test problema iz literature i nekoliko slučajno generisanih primera. U Tabeli 4.1.1 su redom dati: naziv problema, zadate tačnosti, broj iteracija MPD, rezultati dobijeni modifikovanim algoritmom $MMPD$, broj iteracija $MMPD$ i odnos vremena izvršenja ova dva algoritma. Napomenimo da je $eps3 = 1$ u svim primerima. Najveće dimenzije matrice problema su 390×740 kod primera $Test12$, 444×757 kod primera $Degen2$ i 516×758 kod primera $Agg2$ i $Agg3$.

Problem	*eps*	$eps1, 2$	Br.it. MPD	MMPD opt. vred.	Br. it.	$TMod/T$
Afiro	10^{-12}	10^{-1}	13	-464.7531428571451	13	0.7135
Blend	10^{-12}	10^{-2}	11	-30.81214984583144	12	0.8846
KbM	10^{-12}	10^{-3}	12	0.	9	0.5599
Kb2	10^{-12}	10^{-3}	19	-1749.900129941934	20	0.7956
Adlittle	10^{-12}	10^{-2}	16	225494.963162381	16	0.7739
Sc50a	10^{-12}	10^{-3}	13	-64.57507705855755	13	0.8283
Sc50b	10^{-12}	10^{-3}	12	-70.00000000000004	12	0.8682
Sc105	10^{-12}	10^{-3}	15	-52.20206121170699	15	0.8792
Sc205	10^{-12}	10^{-4}	18	-52.20206121170724	17	0.8838
Share2b	10^{-12}	10^{-3}	14	-415.7322407415902	14	0.8414
Stocfor1	10^{-12}	10^{-3}	17	-41131.97621943501	17	0.7763
Lotfi	10^{-8}	10^{-4}	21	-25.2647060618722685	22	0.8105
Agg2	10^{-8}	10^{-4}	22	$-2.023925235068718\times10^{7}$	22	0.8974
Agg3	10^{-10}	10^{-4}	23	$1.031211593574227\times10^{7}$	21	0.7579
Test1	10^{-12}	10^{-4}	30	86.9999999999999928	29	0.8547
Test2	10^{-12}	10^{-4}	19	0.799724607645447882	18	0.8730
Test3	10^{-12}	10^{-4}	21	0.790546594605269792	19	0.6855
Test4	10^{-12}	10^{-4}	19	2.91137275484501465	18	0.7162
Test5	10^{-12}	10^{-4}	19	3.05735620828120602	18	0.6792
Test6	10^{-12}	10^{-4}	22	2.24385948689881997	21	0.7172
Test7	10^{-12}	10^{-4}	22	2.24385948689881997	21	0.8139
Test8	10^{-12}	10^{-4}	23	486.128054336368009	22	0.7257
Test9	10^{-12}	10^{-4}	23	495.742433638785229	22	0.7381
Test10	10^{-12}	10^{-4}	24	495.628752528977134	24	0.8684
Test11	10^{-12}	10^{-4}	25	466.272008865721332	24	0.7779
Test12	10^{-8}	10^{-4}	13	44.51259834897959	13	0.6959

Tabela 4.1.1

Napomenimo da *eps* označava kriterijum za zaustavljanje algoritma. Takođe, problem *KbM* se dobija brisanjem gornjih ograničenja iz poznatog problema *Kb2*. Napomenimo da vrednosti promenljivih *eps1*, *eps2* i *eps3* moraju biti pažljivo izabrane. Na primer, koristeći $eps1 = eps2 = 10^{-3}$ u problemu *Blend*, dobijamo loše uslovljen linearni sistem, ali modifikacija i pored toga daje optimalnu vrednost. Međutim, modifikacija ne daje rešenje za izbor $eps1 = eps2 = 10^{-1}$.

Takođe ilustrujemo tvrđenje da je algoritam konačnog završavanja nepotreban u našem algoritmu. Razmotrimo, na primer, test problem *Afiro.* Optimalna tačka generisana modifikovanim algoritmom ima nula koordinate na pozicijama koje su određene sledećim vektorom $xzero$

$$\{17,17,17,12,12,12,12,12,12,11,9,2,2,2,2,1,28\}.$$

Optimalna tačka xm jednaka je

$$\{0,\ 80.00000000000023,0,0,0,0,18.21428571428548,$$
$$51.49692904472464,73.89388764455072,25.49999999999969,$$
$$500.0000000000107,475.9200000000041,24.08000000000006,$$
$$0,214.9999999999961,147.786872899196,0,$$
$$54.50000000000029,0,0,0,0,0,0,339.9428571428572,$$
$$236.1559842436612,63.54835534665428,0,84.79999999999951,$$
$$69.71121475901005,0,0,0\}.$$

Sa druge strane, obični Mehrotraov metod daje optimalnu tačku x, jednaku vektoru

$\{$2.215848062666221$\times 10^{-12}$, 79.99999999999929,
1.312660664489544$\times 10^{-13}$, 4.394145729655261$\times 10^{-13}$,
4.353491369592836$\times 10^{-13}$, 5.774260579869571$\times 10^{-13}$,
18.21428571428262, 51.50061507109443, 73.89779483249926,
25.5000000000001, 500.00000000001, 475.9199999999925,
24.08000000000016, 7.65658212305429$\times 10^{-12}$,
214.9999999999957, 147.7683249031915,
2.425875538820393$\times 10^{-13}$, 54.49999999999952,
2.415068530966149$\times 10^{-13}$, 2.403460673212315$\times 10^{-13}$,
1.409009221948509$\times 10^{-13}$, 4.755857873948911$\times 10^{-13}$,
4.713557464415677$\times 10^{-13}$, 4.670668078581763$\times 10^{-13}$,
339.9428571428506, 236.1745322396589, 63.54037970837263,
3.686831036665461$\times 10^{-14}$, 84.79999999999963,
69.71490078537609, 2.196141063804613$\times 10^{-13}$,
2.170131429848782$\times 10^{-13}$, 1.773714715268836$\times 10^{-13}\}$.

Sličan oblik rešenja je generisan programom PCx.

Mali elementi u x, koji mogu da generišu loše uslovljenu matricu D^{-2}, su eliminisani tokom primene modifikovanog metoda. Slično, mali elementi u s koji mogu da generišu loše uslovljenu matricu D^2, se takođe izbegavaju u izračunavanjima modifikovanog algoritma. Iz tih razloga, modifikovani algoritam daje bolje rešenje u odnosu na originalni algoritam.

Primer 4.2.2 Na sledeća dva mala test problema je ilustrovana je veća numerička stabilnost modifikovanog metoda. Razmotrimo problem

$$\begin{array}{lrcl} \min & 3x_3 + 2x_4 + x_5 & & \\ \text{p.o.} & x_1 + 2x_2 + x_3 & = & 1000.02, \\ & x_1 + x_2 + x_4 & = & 1000.01, \\ & x_1 - x_2 + x_5 & = & 999.99, \\ & x_1,\, x_2,\, x_3,\, x_4,\, x_5 & \geq & 0. \end{array} \tag{4.2.1}$$

Primenom modifikovanog algoritma sa $eps = 10^{-12}$, $eps1 = 10^{-4}$, $eps2 = 10^{-3}$ i $eps3 = 10^{-1}$ dobijamo optimalnu vrednost 0 i ekstremnu tačku

$$\{999.9999999998636, 0.01000000039754838, 0, 0, 0\}.$$

Sa druge strane, običan algoritam nije u stanju da reši problem sa istom preciznošću 10^{-12}. Već smo pokazali da PCx takođe ne može da reši ovaj problem. Napomenimo da je u radu [64] dat interesantan primer dimenzija 18×18, baziran na prethodnom primenru, koji ne rešavaju PCx, HOPDM ni naša modifikacija.

Drugi primer je sledeći test problem

$$\begin{aligned} \max\ & 30x_1 + 60x_2 + 50x_3 \\ \text{p.o.}\ & 3x_1 + 4x_2 + 2x_3 \leq 60, \\ & x_1 + 2x_2 + 2x_3 \leq 30, \\ & 2x_1 + x_2 + 2x_3 \leq 40, \\ & x_1, x_2, x_3 \geq 0. \end{aligned} \tag{4.2.2}$$

Modifikovani algoritam sa $eps = 10^{-12}$, $eps1 = 10^{-3}$, $eps2 = 10^{-3}$ i $x.s < eps3 = 1$ daje optimalnu vrednost 900.0000000000002 i optimalnu tačku $\{0, 15., 0\}$ posle 9 iteracija. Originalni method posle pet iteracija dolazi do loše uslovljene matrice i staje posle sedam iteracija sa sistemom linearnih jednačina koji nema rešenje.

4.3 Poređenja algoritama i zaključne napomene

U ovom odeljku je opisano poboljšanje primal-dual metoda koje se uglavnom zasniva na eliminaciji vrsta i kolona iz matrice problema. Slična istraživanja su započeta u radu [65] gde je testirana samo eliminacija kolona. Takođe je razvijen eksperimentalni kôd u programu MATHEMATICA za implementaciju predložene modifikacije. Naš glavni cilj je da tim kodom uporedimo Mehrotraov primal-dual metod sa našom modifikacijom koja je opisana u odeljku 2.3. Posle primene predložene heuristike u Mehrotraovom algoritmu imamo sledeće prednosti u odnosu na originalni algoritam, koje su pokazane u prethodnim primerima:

- Modifikovani metod daje bolju preciznost

- Modifikacija poboljšava stabilnost i centralnost iterativnog niza.

  Na primer, tokom rešavanja problema *Afiro* modifikovanim metodom, loše uslovljene matrice koje mogu da dovedu do značajnih numeričkih grešaka se ne pojavljuju. Međutim, tokom primene običnog metoda, posle devetog koraka se javlja loše uslovljena matrica. Slična je situacija sa problemom *Blend*, gde običan metod prouzrokuje loše uslovljenu matricu posle deset koraka. Kao što je pokazano u Primeru 4.2.1, moguće pojavljivanje loše uslovljene matrice se može izbeći pažljivim izborom vrednosti za parametre $eps1$, $eps2$ i $eps3$. Takođe, dva ilustrativna test problema su data u Primeru 4.2.2

- Modifikovani metod je brži u odnosu na originalni, iako sadrži dodatne eliminacije nekih vrsta i kolona i na kraju primenjuje rutinu za rekonstrukciju rešenja. Tu činjenicu potvrđuje poslednja kolona u Tabeli 4.3.1.

  Takođe, primetimo da modifikacija redukuje procesorsko vreme potrebno za rešavanje problema *Blend* i *Kb2*, uprkos dodatnog iterativnog koraka koji zahteva modifikacija.

- Modifikacija smanjuje dimenziju problema i zahtev za memorijskim prostorom.

  U sledećoj tabeli sa *Dim[AS]* označavamo startnu dimenziju matrice A i *Dim[AF]* označava dimenziju finalne matrice A. Takođe, maksimalan brojn bajtova memorije koja se koristi za pamćenje inicijalne matrice A je označen sa *ByteCount[AS]*, dok *ByteCount[AF]* označava maksimalan broj bajtova memorije koja se koristi za pamćenje matrice A u finalnom iterativnom koraku.

Problem	Dim[AS]	Dim[AF]	ByteCount[AS]	ByteCount[AF]
Afiro	27×52	21×16	28864	7336
Blend	74×114	60×56	170816	68904
KbM	43×68	27×6	59712	3808
Kb2	52×77	33×33	81768	22864
Adlittle	56×138	46×61	156152	57616
Sc50a	27×52	21×16	28864	7336
Sc50b	74×114	60×56	170816	68904
Sc105	43×68	27×6	59712	3808
Share2b	52×77	33×33	81768	22864
Stockfor1	56×138	46×61	156152	57616

Tabela 4.3.1

Pored toga, navedimo i dodatne prednosti modifikovanog metoda:

- U modifikovanom algoritmu algoritam konačnog zaokruženja je nepotreban, kao što je pokazano u Primeru 4.2.1.
- Modifikaciju je moguće primeniti u bilo kom primal-dual algoritmu.
- Opisana modifikacija je primenljiva na već postojeće primal-dual kodove.
- Kako modifikacija redukuje dimenziju problema za vreme iterativnih koraka, realno je očekivati da će biti posebno efikasna za velike probleme linearnog programiranja (primeri *Agg2, Agg3, T12*).

5 Prošireni i normalni sistem jednačina

U ovom poglavlju upoređujemo dve varijante Mehrotraovog primal dual algoritma koje se baziraju na proširenom i normalnom sistemu jednačina, respektivno razmatrane su u radu [87]. Implementacija odgovarajućih algoritama u programu MATHEMATICA je iskorišćena za poređenje. Na kraju dajemo nekoliko ilustrativnih numeričkih primera.

5.1 Primal-dual algoritam sa proširenim sistemom

Razmatramo opšti problem linearnog programiranja (1.1.1). Poznato je da linearni sistem koji se rešava u svakoj primal-dual iteraciji možemo formulisati na tri ekvivalentna načina. Neredukovan oblik za nedopustiv algoritam unutrašnje tačke je

$$\begin{bmatrix} 0 & A & 0 \\ A^T & 0 & I \\ 0 & S & X \end{bmatrix} \begin{bmatrix} \Delta\lambda \\ \Delta x \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_b \\ -r_c \\ -XSe + \sigma\mu e \end{bmatrix}, \tag{5.1.1}$$

gde je $\mu = x^T s/n$ and $r_b = Ax - b$, $r_c = A^T\lambda + s - c$.

Eliminacijom Δs iz (5.1.1) i koristeći notaciju $D = S^{-1/2}X^{1/2}$, dobijamo prošireni sistem i posle eliminacije Δx iz proširenog sistema dobijamo normalni sistem jednačina. Detalji će biti dati u opisu algoritma. Pomenuti sistemi linearnih jednačina se rešavaju u *Koraku 4* i *Koraku 8*, tako da ostale korake koji su identični odgovarajućim koracima u klasičnom Mehrotrovom algoritmu ne navodimo.

Korak 4. Izračunati $D = S^{-1/2}X^{1/2}$, $r_{xs} = XSe$ i rešiti jedan od sledeća dva sistema u odnosu na $(\Delta x^{aff}, \Delta\lambda^{aff}, \Delta s^{aff})$. U slučaju normalnog sistema rešiti sistem

$$\begin{aligned} AD^2A^T\Delta\lambda^{aff} &= -r_b - A(S^{-1}Xr_c - S^{-1}r_{xs}),\\ \Delta s^{aff} &= -r_c - A^T\Delta\lambda^{aff},\\ \Delta x^{aff} &= -S^{-1}(r_{xs} + X\Delta s^{aff}), \end{aligned}$$

gde je

$$S^{-1} = \mathrm{diag}(1/s_1, \ldots, 1/s_{k+q}), \quad D^2 = \mathrm{diag}(x_1/s_1, \ldots, x_{k+q}/s_{k+q}).$$

U slučaju proširenog sistema rešiti

$$\begin{aligned} \begin{bmatrix} 0 & A \\ A^T & -D^{-2} \end{bmatrix} \begin{bmatrix} \Delta\lambda^{aff} \\ \Delta x^{aff} \end{bmatrix} &= \begin{bmatrix} -r_b \\ -r_c + X^{-1}r_{xs} \end{bmatrix},\\ \Delta s^{aff} &= -r_c - A^T\Delta\lambda^{aff}. \end{aligned} \tag{5.1.2}$$

Napomenimo da je u [112] preporučena sledeća formula za Δs^{aff}:

$$\Delta s^{aff} = -X^{-1}(r_{xs} - A^TS\Delta x).$$

Ali, u tom slučaju radimo sa matricom X^{-1}, koja je loše uslovljena u finalnim iterativnim koracima. Iz tog razloga prednost dajemo reprezentaciji oblika (5.1.2).

Korak 8. Izračunati $r_{xs} = -\sigma\mu e + \Delta X^{aff}\Delta S^{aff}e$, gde je

$$\Delta X^{aff} = \mathrm{diag}(\Delta x_1^{aff}, \ldots, \Delta x_n^{aff}),\ \Delta S^{aff} = \mathrm{diag}(\Delta s_1^{aff}, \ldots, \Delta s_n^{aff}),$$

i rešiti jedan od sledeća dva sistema u odnosu na $(\Delta x^{cor}, \Delta\lambda^{cor}, \Delta s^{cor})$. U slučaju normalnog sistema rešiti sistem

$$\begin{aligned} AD^2A^T\Delta\lambda^{cor} &= AS^{-1}r_{xs},\\ \Delta s^{cor} &= -A^T\Delta\lambda^{cor},\\ \Delta x^{cor} &= -S^{-1}(r_{xs} + X\Delta s^{cor}), \end{aligned}$$

ili u slučaju proširenog sistema rešiti

$$\begin{aligned} \begin{bmatrix} 0 & A \\ A^T & -D^{-2} \end{bmatrix} \begin{bmatrix} \Delta\lambda^{cor} \\ \Delta x^{cor} \end{bmatrix} &= \begin{bmatrix} -r_b \\ -r_c + X^{-1}r_{xs} \end{bmatrix},\\ \Delta s^{cor} &= -A^T\Delta\lambda^{cor}. \end{aligned} \tag{5.1.3}$$

5.2 Implementacija algoritma sa proširenim sistemom

Za generisanje matrice proširenog sistema (5.1.2) i (5.1.3) koristimo sledeću rutinu $FormAug$:

```
LinearAlgebra`MatrixManipulation`

FormAug[a_,d_]:=
   Module[{rez={},z={},dim},
     dim=Dimensions[a][[1]];
     z=ZeroMatrix[dim];
     rez=AppendColumns[AppendRows[z,a],AppendRows[Transpose[a],-d]];
     Return[rez];
   ]
```

Korak 4:

Implementacija rešavanja normalnog sistem u *Koraku 4* je objašnjena ranije. Implementacija koja odgovara proširenom sistemu je data sledećim kodom:

```
dd= DiagonalMatrix[N[s/x,20]];
AugA=FormAug[a,dd];
ls=LinearSolve[Xmat,rxs];       (* Compute X^-1rxs *)
lx=LinearSolve[AugA,Join[-rb,-rc+ls]]; (* Solve the system (2.2A) *)
Dlaff=Take[lx,Dimensions[a][[1]]];
Dxaff=Drop[lx,Dimensions[a][[1]]];
Dsaff=N[-rc-Transpose[a].Dlaff,20];
```

Napomenimo da vrednost $ls = X^{-1}rxs$ može biti izračunata sa $ls = Inverse[Xmat].rxs$. Međutim, izraz $Inverse[Xmat]$ prouzrokuje numeričke probleme zbog malih vrednosti nekih koordinata vektora x koje se nalaza na glavnoj dijagonali matrice $Xmat$.

Korak 8:

Implementacija rešavanja normalnog sistem u *Koraku 8* je objašnjena ranije. Implementacija za prošireni sistem (5.1.3) je data sledećim kôdom:

```
DXaff=DiagonalMatrix[Dxaff];     DSaff=DiagonalMatrix[Dsaff];
rxs=N[-mi*SigCent,20]*Table[1,{k+q}]+N[DXaff.DSaff,20].Table[1,{k+q}];
rb=Table[0,{Dimensions[a][[1]]}];
ls=LinearSolve[Xmat,rxs];  (* Compute ls = X^-1rxs *)
lx=LinearSolve[AugA,Join[-rb,ls]]; (* Solve the system *)
Dlcc=Take[lx,Dimensions[a][[1]]];
Dxcc=Drop[lx,Dimensions[a][[1]]];
Dscc=N[-Transpose[a].Dlcc,20];
```

5.3 Numerički primeri

Primer 5.3.1 Primenili smo obe verzije algoritma sa tačnošću $eps = 10^{-8}$ na poznate test probleme. Koristimo oznake $TimeA$ i $TimeN$ za procesorsko vreme potrebno u slučaju proširenog i normalnog sistema, respektivno. U sledećoj tabeli date su dimenzije problema i količnik potrebnih vremena za rešavanje problema.

Problem	Dimenzije	$TimeA/TimeN$
Adlittle	56×138	0.76
Afiro	27×51	0.95
Agg	488×615	0.52
Agg2	516×758	0.46
Agg3	516×758	0.57
Bandm	305×472	0.63
Blend	74×114	0.75
Israel	174×316	0.69
Kb2	43×68	0.69
Lotfi	153×366	0.64
Sc105	105×163	0.51
Sc205	205×203	0.55
Sc50b	50×78	0.72
Sc50a	50×78	0.66
Scagr7	129×185	0.61
Sctap1	300×660	0.78
Share2b	96×162	0.66
Stocfor1	117×165	0.60

Tabela 5.3.1.

Primetimo da je implementacija proširenog sistema brža u svim primerima iz Tabele 5.3.1. Takođe se može videti da količnik između procesorskih vremena potrebnih za prošireni i normalni sistem opada približno sa porastom dimenzije problema.

Primer 5.3.2 Za primer (4.2.2) primenom proširenog sistema dobijamo optimalnu vrednost `899.999999999` za obe početne tačke. Primenom normalnog sistema dobijamo istu optimalnu vrednost koristeći (2.2.3) startnu tačku, ali koristeći (2.2.2) za startnu tačku posle pet iteracija matrica postaje loše uslovljena.

Primer 5.3.3 U slučaju test primera *Degen2* početna matrica je loše uslovljena i kod proširenog sistema se javlja fenomen oscilovanja vrednosti xs. U prvih 15 iteracija dobijene su sledeće vrednosti:

```
{2944.86, 2936.47, 2941.46, 2948.47, 2946.99, 2954.17, 2960.11, 3014.65,
3027.11, 3026.22, 2965.13, 2954.41, 2055.41, 2970.52, 3015.56}.
```

Sa druge strane normalni sistem uspešno rešava primer *Degen2*. Suprotna situacija je kod test primera *Scfxm1* koji prošireni sistem rešava a normalni sistem ne može da reši.

U ovom poglavlju smo uporedili dve varijante Mehrotraovog primal dual algoritma koje se baziraju na proširenom i normalnom sistemu jednačina. Na primerima je pokazano da je prošireni sistem u opštem slučaju brži u odnosu na normalni sistem. Kako je tačnost slična u oba pristupa, to primena proširenog sistema ima bolje karakteristike. Ovde moramo napomenuti da ovi rezultati zavise od specifičnih metoda za rešavanje proširenog i normalnog sistema u programu MATHEMATICA. U velikom broju poznatih kodova Cholesky faktorizacija [78, 84] se koristi za rešavanje normalnog sistema jer se u mnogim primerima javljaju retke matrice [98]. Napomenimo da postoje i radovi u kojima se za rešavanje sistema predlažu iterativne metode [34, 8, 48]. Kako je napomenuto u [2, 112] prošireni sistem je numerički stabilniji i može se lako prilagoditi za rešavanje problema kvadratnog programiranja. Posebno, problemi koji sadrže slobodne promenljive se lakše rešavaju primenom proširenog sistema, dok primena normalnih jednačina zahteva nezgodnu

reformulaciju problema. Ali, primena proširenog sistema ima i izvesne nedostatke [112]:

- Algoritmi i softver za rešvanje simetričnih neodređenih sistema sa retkom matricom nisu tako dobro razvijeni i široko dostupni kao Cholesky kodovi za sisteme sa retkom matricom.

- U opštem slučaju zahteva više procesorskog vremena (u proseku, 50% -100% više) za izračunavanje iterativnog koraka u odnosu na Cholesky algoritm.

Prva primedba je od manje važnosti jer se veoma brzo razvija softver u toj oblasti [23, 5, 26]. Druga primedba ostaje i pored značajnog napretka u razvoju softvera. Kao što je pokazano, primena proširenog sistema u programu MATHEMATICA je čak i brža, posebno za probleme većih dimenzija. Na kraju napomenimo da program HOPDM u zavisnosti od osobina postavljenog problema određuje koji će se pristup primeniti. Primer 5.3.3 pokazuje da je u nekim slučajevima struktura problema presudna za uspešnu primenu izabranog pristupa.

6 Stabilizacija Mehrotraovog algoritma

U ovom poglavlju primenjujemo stabilizacionu proceduru na Mehrotraov primal dual algoritma, predloženu u [64] i [62]. Implementacija je izvršena u programu MATHEMATICA i dato je nekoliko test primera za komparaciju stabilizacione procedure i originalnog metoda. Rezultati iz ovog poglavlja su objavljeni u radovima [88, 89, 90, 100].

6.1 Teorijski uzroci numeričke nestabilnosti

Ranije smo pokazali da metodi unutrašnje tačke u t-toj iteraciji rešavaju sistem linearnih jednačina sa matricom oblika

$$AD_t^2A^T \text{ ili } \begin{bmatrix} 0 & A \\ A^T & -D_t^{-2} \end{bmatrix}.$$

Jedan od najozbiljnijih izvora numeričkih poteškoća je činjenica da u slučaju kada je skup optimalnih rešenja X^* degenerisan matrica teži da postane ekstremno loše uslovljena kako se približavamo optimalnom rešenju. Postoji više radova u kojima je proučavan sistem oblika

$$(AD_t^2A^T)u = AD_td.$$

Njegovo rešenje $u = (AD_t^2A^T)^{-1}AD_td$ ima zanimljivu osobinu da je uniformno ograničeno i pored moguće loše uslovljenosti matrice $AD_t^2A^T$. Prvi rezultat ovog tipa je dobio Dikin [19], koji je pokazao da je

$$\|u\| \leq \max_{J\in\Omega(A)} \|(A_J^T)^{-1}d_J\|$$

gde je $\Omega(A)$ skup indeksa kolona svih nesingularnih $m \times m$ podmatrica od A i A_J je podmatrica od A sa kolonama čiji indeksi su u J. Isti rezultat su dobili i Stewart [101] i Ben-Tal, Teboulle [9]. Stewart je još pokazao da je

$$\|A^T(AD_t^2A^T)^{-1}AD_t\|$$

uniformno ograničeno u odnosu na t i odredio eksplicitno ograničenje u zavisnosti od singularnih vrednosti matrice A. I pored toga, činjenica da je rešenje ograničeno ne povlači da će izračunavanje tog rešenja biti numerički stabilno. U primal-dual agoritmima se obično za rešavanje sistema koristi Cholesky dekompozicija. Ona se često stabilizuje heuristikom po kojoj se pivot elementi manji od granične vrednosti ε izostavljaju. Pod izvesnim pretpostavkama Wright [115] je pokazao da ta heuristika ima teorijsku osnovu, tj. da je rastojanje između tačnog i približnog rešenja ograničeno sa $\sqrt{\varepsilon}$. Međutim, u radovima [6, 64] je pokazano da u slučaju degeneracije metod unutrašnje tačke može da postane loše uslovljen, čak i u najjačim implementacijama, kao što su PCx [14, 70, 112] i HOPDM [38]. Takođe, u radovima [6, 64] je data stabilizaciona tehnika koja se zasniva na Gaussovoj eliminaciji i prevodi originalni problem u njemu ekvivalentan oblik pri čemu je loša uslovljenost izbegnuta. U nastavku opisujujemo implementaciju stabilizovanog algoritma koja rešava klasu loše uslovljenih test problema iz [6, 64] koje programi PCx i HOPDM nisu uspeli da reše.

Neka je $\{(x^t, \lambda^t, s^t)\}$ niz generisan primal-dual algoritmom i neka je $\{AD_t^2A^T\}$ odgovarajući niz matrica definisanih u *Koraku* 4. Prema Propoziciji 2.1 iz [64], u slučaju kada je optimalna površ X^* degenerisana sledi da $\text{cond}(AD_t^2A^T) \to \infty,\ t \to \infty$. To znači da matrice sistema (2.1.1) i (2.1.2) teže da postanu ekstremno loše uslovljene i da neka stabilizaciona procedura mora biti primenjena. Stabilizaciona procedura koju želimo da primenimo koristi neke pretpostavke o uniformnosti niza $\{(x^t, \lambda^t, s^t)\}$.

Predpostavka 6.1.1 *Neka je* $(x^t, \lambda^t, s^t), t = 1, 2, \ldots$ *niz generisan primal-dual metodom unutrašnje tačke, sa osobinom da sve tačke nagomilavanja od* $\{(x^t, \lambda^t, s^t)\}$ *pripadaju* $riX^* \times riY^*$. *Tada postoje pozitivne konstante* u *i* v *tako da je*

$$u \le (x_i^t/s_i^t)/(x_j^t/s_j^t) \le 1/u, \quad i, j \in \mathcal{B} \quad \textit{and}$$
$$v \le (x_i^t/s_i^t)/(x_j^t/s_j^t) \le 1/v, \quad i, j \in \mathcal{N}.$$

Pretpostavka 6.1.1 je blisko povezana sa pojmom maksimalnog komplementarnog niza (MCS) koji su uveli Guller i Ye u radu [47]. Svaki MCS niz zadovoljava Pretpostavku 6.1.1 [64] tako da to važi za većinu metoda unutrašnje tačke.

Glavna ideja sledeće procedure, preuzete iz [64], je u transformaciji originalnog problema u ekvivalentni oblik u kome se problemi stabilnosti ne javljaju ukoliko se primeni pravilna strategija za izračunavanje traženog pravca.

Procedura 6.1.1

Korak 1. Za dato t neka su $j(1), \ldots, j(n)$ indeksi koji zadovoljavaju uslov $(x_{j(1)}^t/s_{j(1)}^t) \ge \cdots \ge (x_{j(n)}^t/s_{j(n)}^t)$, i neka je $\hat{h} \in \{1, \ldots, n-1\}$ najmanji indeks tako da je $(x_{j(\hat{h}+1)}^t/s_{j(\hat{h}+1)}^t) < 1$.

Korak 2. Urediti kolone matrice A i c^T tako da $j(p)$-ta kolona dođe na p-tu poziciju, $p = 1, \ldots, n$. Set $p = 1, q = 1$.

Korak 3. Odrediti $|a_{i(p),p}| = \max\{|a_{ip}|, i = 1, \ldots, m\}$. Ako je $|a_{i(p),p}| \neq 0$ preći na *Korak 5.*

Korak4. Ako je $|a_{i(p),p}| = 0$ i $p = \hat{h}$ staviti $\hat{r} = q - 1$ i RETURN. U suprotnom zameniti p sa $p + 1$ i preći na *Korak 3.*

Korak 5. Zameniti $i(p)$-tu i q-tu vrstu u A i b. Koristeći a_{qp} kao pivot element eliminisati $a_{ip}, i = q + 1, \ldots, m$ (ako postoje) i c_p.

Korak 6. Ako je $q = \min\{\hat{h}, m\}$ ili $p = \hat{h}$ staviti $\hat{r} = q$ i RETURN. U suprotnom zameniti p sa $p + 1$, q sa $q + 1$ i preći na *Korak 3.*

Ako se Procedura 6.1.1 primenjuje u $\hat{t}$-tom koraku metoda unutrašnje tačke i ako je $\hat{r} < m$, parametare problema transformiše u oblik

$$\hat{A} = \begin{bmatrix} P & Q \\ O & R \end{bmatrix}, \hat{b}, \hat{c}$$

gde je P matrica tipa $\hat{r} \times \hat{h}$, $\text{rank}(P) = \hat{r}$, matrica O je nula matrica tipa $(m-\hat{r}) \times \hat{h}$, Q i R su matrice tipa $\hat{r} \times (n - \hat{r})$ i $(m - \hat{r}) \times (n - \hat{r})$ respektivno. Napomenimo da ova transformacija jednostavno permutuje komponente od x^t i s^t.

Neka je

$$AD_t^2 A^T = d^t \tag{6.1.1}$$

sistem koji treba rešiti u t-toj iteraciji. Za $t \geq \hat{t}$ koeficijenti matrice u (6.1.1) imaju oblik

$$\hat{A}\hat{D}_t^2 \hat{A}^T = \begin{bmatrix} F_t & G_t \\ G_t^T & H_t \end{bmatrix}$$

i F_t, G_t i H_t su respektivno $\hat{r} \times \hat{r}$,$\hat{r} \times (m - \hat{r})$ i $(m - \hat{r}) \times (m - \hat{r})$ matrice. Tada važi sledeće tvrđenje [64].

Teorema 6.1.1 *Pretpostavimo da metod unutrašnje tačke zadovoljava Pretpostavku 6.1.1 i da je Procedura 6.1.1 primenjena periodično, tj. u koracima $t=\hat{t}$, $t=2\hat{t}, \ldots$. Tada postoji konstanta C nezavisna od t tako da je*

$$cond(F_t) \leq C \text{ i } cond(H_t - G_t^T F_t^{-1} G_t) \leq C.$$

Prethodna teorema tvrdi da Procedura 6.1.1 rastavlja loše uslovljenu matricu u dobro uslovljene podmatrice. To nam osigurava stabilnost u daljim numeričkim izračunavanjima.

Rezultati Teoreme 6.1.1 se koriste u Algoritmu 6.1.1 na sledeći način. Za rešavanje sistema (2.1.1)

$$AD^2 A^T \Delta\lambda^{aff} = -r_b - A(S^{-1} X r_c - S^{-1} r_{xs})$$

koristimo sledeću dekompoziciju matrice AD^2A^T:

$$AD^2 A^T = \begin{bmatrix} F_t & G_t \\ G_t^T & H_t \end{bmatrix}. \tag{6.1.2}$$

Ako razbijemo svaki od vektora $\Delta\lambda^{aff}$ i $-r_b - A(S^{-1}Xr_c - S^{-1}r_{xs})$ u dva podesna bloka, označena sa δ_1, δ_2 i v_1, v_2, respektivno, u skladu sa blok dekompozicijom (6.1.2), sistem (2.1.1) može biti zamenjen ekvivalentnim sistemom:

$$\begin{aligned} \delta_1 + F_t^{-1}G_t\delta_2 &= F_t^{-1}v_2, \\ (H_t - G_t^T F_t^{-1}G_t)\delta_2 &= v_2 - G_t^T F_t^{-1}v_1. \end{aligned} \tag{6.1.3}$$

Prema Teoremi 6.1.1 sistem (6.1.3) sadrži samo dobro uslovljene matrice.

Na kraju navodimo teoremu koja daje oblik dualnog problema modifikovanog primalnog problema.

Teorema 6.1.2 *Pretpostavimo da matrica problema linearnog programiranja ima oblik*

$$A = \begin{bmatrix} a_{11} & \dots & a_{1,p-1} & a_{1p} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & a_{q-1,p-1} & a_{q-1,p} & \dots & a_{q-1,n} \\ 0 & \dots & 0 & a_{qp} & \dots & a_{qn} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & a_{mp} & \dots & a_{mn} \end{bmatrix}.$$

Ako upotrebimo a_{qp} kao pivot element za eliminaciju a_{ip} i saglasno transformišemo $b_i, i = q+1, \dots, m$, tada dual transformisanog problema

$$\min \ c^T x \quad \text{p.o. } A'x = b', \quad x \geq 0,$$

ima oblik

$$\max \ b'^T \lambda' \quad \text{p.o. } A'^T \lambda' + s = c, \quad s \geq 0,$$

gde je

$$\begin{aligned} \lambda'_i &= \lambda_i, \quad i \neq q, \\ \lambda'_q &= \lambda_q + \sum_{i=q+1}^{m} \frac{a_{ip}}{a_{qp}} \lambda_i. \end{aligned} \tag{6.1.4}$$

Dokaz. Treba pokazati $b'^T \lambda' = b^T \lambda$ i $A'^T \lambda' = A^T \lambda$. Kako je

$$\begin{aligned} b'_i &= b_i, \quad i = 1, \dots, q, \\ b'_i &= b_i - \frac{a_{ip}}{a_{qp}} b_q, \quad i = q+1, \dots, m, \end{aligned}$$

vrednost dualne ciljne funkcije je

$$\begin{aligned} {b'}^T\lambda' &= \sum_{i=1}^{m} b'_i\lambda'_i \\ &= \sum_{i=1}^{q-1} b_i\lambda_i + b_q\left(\lambda_q + \sum_{i=q+1}^{m}\frac{a_{ip}}{a_{qp}}\lambda_i\right) + \sum_{i=q+1}^{m}\left(b_i - \frac{a_{ip}}{a_{qp}}b_q\right)\lambda_i \quad (6.1.5) \\ &= \sum_{i=1}^{q} b_i\lambda_i + \sum_{i=q+1}^{m}\frac{a_{ip}}{a_{qp}}b_q\lambda_i + \sum_{i=q+1}^{m} b_i\lambda_i - \sum_{i=q+1}^{m}\frac{a_{ip}}{a_{qp}}b_q\lambda_i = b^T\lambda. \end{aligned}$$

Dalje, elementi ${A'}^T\lambda'$ imaju oblik $\sum_{i=1}^{m} a'_{ij}\lambda'_i, \quad j = 1,\ldots,n$. Sada, koristeći (6.1.4) i $a'_{ij} = a_{ij} - \frac{a_{qj}}{a_{qp}}a_{ip}, j = q+1,\ldots,m$, imamo

$$\begin{aligned} &\sum_{i=1}^{m} a'_{ij}\lambda'_i = \sum_{i=1}^{m} a_{ij}\lambda_i, \quad j = 1,\ldots,p-1, \\ &\sum_{i=1}^{m} a'_{ip}\lambda'_i = \sum_{i=1}^{q-1} a_{ip}\lambda_i + a_{qp}\left(\lambda_q + \sum_{i=q+1}^{m}\frac{a_{ip}}{a_{qp}}\lambda_i\right) = \sum_{i=1}^{m} a_{ip}\lambda_i, \\ &\sum_{i=1}^{m} a'_{ij}\lambda'_i = \sum_{i=1}^{q-1} a_{ij}\lambda_i + a_{qj}\left(\lambda_q + \sum_{i=q+1}^{m}\frac{a_{ip}}{a_{qp}}\lambda_i\right) + \sum_{i=q+1}^{m}\left(a_{ij} - \frac{a_{qj}}{a_{qp}}a_{ip}\right)\lambda_i \\ &\qquad = \sum_{i=1}^{m} a_{ij}\lambda_i, \quad j = p+1,\ldots,n. \end{aligned}$$

Prema tome, jednakost ${A'}^T\lambda' = A^T\lambda$ važi. □

6.2 Implementacija stabilizacione procedure

Sada opisujemo implementaciju Procedure 6.1.1 i odgovarajuću modifikaciju Mehrotraovog primal-dual metoda.

Formalni parametri se definišu sa:

$a\_, b\_, lam\_$: data matrica A i vektori b i λ, respektivno.
$xs\_$: vektor x/s čije koordinate su x_i/s_i, $i = 1,\ldots,k+n$.
$c\_, lam\_$: vektor ciljne funkcije i vektor λ u dualnom problemu.

Matrica na koju primenjujemo Gaussovu eliminaciju je oblika

$$[\,A\ b\,] = [\,a\ b\,].$$

```
<<LinearAlgebra‘MatrixManipulation‘
Elimination[a_, b_, xs_, c_, lam_] :=
Module[{an=a, xss={}, pom={}, indper={},n,m,p,q,h,L,maxi,cn=c,l=lam},
```

```
    {m,n}=Dimensions[a];    brelim++; inp = 1;
   (*  Step 1 *)
    indper = Sort[Range[n], xs[[#1]] > xs[[#2]] &];
    xss = xs[[indper]];
    h = 1;
     While[xss[[h]]>=1 && h<n, h++];
    h = h-1;  L = True;
    If[h==n, L=False; q=n];
     (* Step 2 *)
    an=Transpose[Append[Transpose[a][[indper]],b]];
    cn=cn[[indper]];
     p = 1; q=1;
     While[(L && (q<m)),
         (* Step 3 *)
         pom = Drop[pom, q-1];
         maxi = Position[pom, Max[pom]][[1,1]] + q-1;
         If[an[[maxi, p]]==0,
             (* Step 4 *)
             If[p==h,          (* Then *)
                 Return[List[an, indper, q-1, cn, l]],
                 p++;      (* Else *)
         ];
             ,                  (* Else *)
             (* Step 5 *)
             If[maxi != q,
                 pom=an[[maxi]];  an[[maxi]]=an[[q]];   an[[q]]=pom;
                 pom=l[[maxi]];   l[[maxi]]=l[[q]];     l[[q]]=pom;
             ];
            For[i=q+1, i<=m, i++,
                 l[[q]]=l[[q]]+N[ l[[i]]*an[[i,p]]/an[[q,p]], 20];
             ];
             (* Adaptation of the vector $\lambda $*)
             Do[an[[i]]=an[[i]]-N[an[[i, p]], 20]*N[an[[q]], 20]/N[an[[q, p]], 20],
                                {i,q+1,m}    (* Gaussian elimination *)
             ];
             (* Step 6 *)
            L = Not[(q == Min[h, m]) || (p==h)];
             If[L, p++; q++;];
         ];
     ];
     Return[List[an, indper, q, cn, l]];
];  }
```

Rezultat funkcije *Elimination*[] je lista čiji su elementi definisani u nastavku.

`an, cn, l`: matrica a i vektori c i l respektivno, posle stabilizacione procedure,

`indper`: skup indeksa $\{j_1, \ldots, j_n\}$ sortirani u skladu sa *Korakom* 1 Procedure 6.1.1.

Eliminacija se primenjuje u *Koraku* 4. Posle primene eleiminacije, transformisane matrice se koriste za rešavanje sistema u *Koraku* 4 i *Koraku* 8. Stabilizacija se prvi put primenjuje kada je matrica AD^2A^T loše uslovljena. U svim ostalim slučajevima, eliminacija se primenjuje kada je jedna od matrica F_t ili $H_t - G_t^T F_t^{-1} G_t$ loše uslovljena. Za izračunavanje faktora uslovljenosti matrice koristimo funkciju $MatrixConditionNumber[mat]$, koja izračunava faktor uslovljenosti u odnosu na

beskonačnu normu matrice *mat*. Koristimo parametar *eps*3 koji određuje graničnu vrednost od koje primenjujemo eliminaciju, koji je definisan sledećom heuristikom:

```
If[MatrixConditionNumber[N[a.d.Transpose[a]]]<=10^10,
     eps3=10*MatrixConditionNumber[N[a.d.Transpose[a]]],
     eps3=10^16
];
```

Modifikacija *Koraka 4*:
Da bi rešili sistem

$$AD^2A^T\Delta\lambda^{aff} = -r_b - A(S^{-1}Xr_c - S^{-1}r_{xs})$$

koristimo dekompoziciju matrice AD^2A^T na sledeći oblik:

$$AD^2A^T = \begin{bmatrix} F_t & G_t \\ G_t^T & H_t \end{bmatrix}.$$

Ako razdvojimo svaki od vektora $\Delta\lambda^{aff} = Dlaff$ i $-r_b - A(S^{-1}Xr_c - S^{-1}r_{xs}) = vRight$ u dva pogodna bloka, označena sa δ_1, δ_2 i v_1, v_2, respektivno, tada moramo rešiti sledeći linearni sistem:

$$\delta_1 + F_t^{-1}G_t\delta_2 = F_t^{-1}v_2, \tag{6.2.1}$$
$$(H_t - G_t^TF_t^{-1}G_t)\delta_2 = v_2 - G_t^TF_t^{-1}v_1. \tag{6.2.2}$$

Taj korak je implementiran u sledećem kodu. Vrednost $pok = 0$ parametra *pok* označava da eliminacija nije prethodno primenjivana. Sledeća rutina je upotrebljena za početak stabilizacije.

```
is=DiagonalMatrix[N[1/s,20]];   d= DiagonalMatrix[N[x/s,20]];
If[pok==0,
  If[( MatrixConditionNumber[N[a.d.Transpose[a]]]>eps3)&&(NumStep>1),
     pok=1;  (* $Cond(A)$ causes the elimination *)
      pom=Elimination[a,b,x/s,ce,l];
      vR=pom[[3]];  ce=pom[[4]]; l=pom[[5]];
      a=N[Transpose[Drop[Transpose[pom[[1]]],-1]]];
      b=N[Flatten[Take[Transpose[pom[[1]]],-1]]];
      pom=pom[[2]];
      permut=permut[[pom]];
      x=x[[pom]]; s=s[[pom]]; rc=rc[[pom]];
      rb=N[a.x-b,20];  rc=N[Transpose[a].l+s-ce,20];
      Xmat=DiagonalMatrix[x];   Smat=DiagonalMatrix[s];
      rxs=N[Xmat.Smat.Table[1,{n}],20];
      is= DiagonalMatrix[N[1/s,20]];   d= DiagonalMatrix[N[x/s,20]];
   ];,

  If[ ((vR<Dimensions[a][[1]])&&(MatrixConditionNumber[vH-Transpose[vG].vF.vG]>eps3))
        ||( MatrixConditionNumber[N[vF]]>eps3),
      (* $Cond(vF)$ or $cond(vH-vG^T VF^{-1}VG)$ causes the elimination *)
      pom=Elimination[a,b,x/s,ce,l];
      vR=pom[[3]];  ce=pom[[4]]; l=pom[[5]];
      a=N[Transpose[Drop[Transpose[pom[[1]]],-1]]];
```

```
      b=N[Flatten[Take[Transpose[pom[[1]]],-1]]];
      pom=pom[[2]];
      permut=permut[[pom]];
      x=x[[pom]]; s=s[[pom]];
      rb=N[a.x-b,20];   rc=N[Transpose[a].l+s-ce,20];
      Xmat=DiagonalMatrix[x];   Smat=DiagonalMatrix[s];
      rxs=N[Xmat.Smat.Table[1,{n}],20];
      is= DiagonalMatrix[N[1/s,20]];   d= DiagonalMatrix[N[x/s,20]];
   ];
]; (* Criteria for the stabilization starting are verified, *)
  (* and the elimination is possibly started *)  }
```

Sledeća rutina se koristi za proveru da li je rešavan sistem oblika (2.1.1) ili sistem oblika (6.2.1)-(6.2.2). Jednačine (2.1.2) i (2.1.3) ostaju iste u oba slučaja.

```
 If[pok==0,  (* The case when the stabilization is not already applied *)
  If[(vR<Dimensions[a][[1]])&&( MatrixConditionNumber[N[a.d.Transpose[a]]]>eps3)
                        &&(NumStep>1),
       vA=a.d.Transpose[a];
       (* Submatrix F *)   vF=SubMatrix[vA,{1,1},{vR,vR}];   vF=N[Inverse[vF],20];
       (* Submatrix G *)   vG=SubMatrix[vA,{1,vR+1},{vR,Dimensions[vA][[2]]-vR}];
       (* Submatrix H *)
       vH=SubMatrix[vA,{vR+1,vR+1},
                    {Dimensions[vA][[1]]-vR,Dimensions[vA][[2]]-vR}];
       vRight=N[-rb-a.(is.Xmat.rc-is.rxs),20];
       vRes=N[LinearSolve[vH-Transpose[vG].vF.vG,
                  Drop[vRight,vR]-Transpose[vG].vF.Take[vRight,vR]],20];
       Dlaff=N[vF.Take[vRight,vR]-vF.vG.vRes,20];
       Dlaff=Join[Dlaff,vRes];
       hhh=vH-Transpose[vG].vF.vG;
        ,
     Dlaff=N[LinearSolve[a.d.Transpose[a],-rb-a.(is.Xmat.rc-is.rxs)],20 ];
   (* The elimination is applied *);
        ];,  (*pok=1*)
(* The case when the stabilization is already applied *)
   vA=a.d.Transpose[a];
   (* Submatrix F *)  vF=SubMatrix[vA,{1,1},{vR,vR}];  vF=N[Inverse[vF],20];
   (* Submatrix G *)  vG=SubMatrix[vA,{1,vR+1},{vR,Dimensions[vA][[2]]-vR}];
   (* Submatrix H *)
    If[vR<Dimensions[a][[1]],
      vH=SubMatrix[vA,{vR+1,vR+1},{Dimensions[vA][[1]]-vR,Dimensions[vA][[2]]-vR}];
      vRight=N[-rb-a.(is.Xmat.rc-is.rxs),20];
     vRes=N[LinearSolve[vH-Transpose[vG].vF.vG,
                 Drop[vRight,vR]-Transpose[vG].vF.Take[vRight,vR]],10];
      Dlaff=N[vF.Take[vRight,vR]-vF.vG.vRes,20];
     Dlaff=Join[Dlaff,vRes];
      hhh=vH-Transpose[vG].vF.vG;
     ,
     (* Transformation is not applied *)
      Dlaff=N[LinearSolve[a.d.Transpose[a],
      ];
   ];
];
   (* Solve the systems (2.2) and (2.3) *)
Dsaff=N[-rc-Transpose[a].Dlaff,20];
Dxaff=N[-is.(rxs+Xmat.Dsaff),20];   }
```

Modifikacija *Koraka* 8:

Analogno kao u *Koraku* 4, linearan sistem oblika (6.2.1)-(6.2.2) treba rešiti umesto sistema (2.1.4) u *Koraku* 8.

Da bi rešili linearan sistem $AD^2A^T\Delta\lambda^{cor} = AS^{-1}r_{xs}$ koristimo razbijanje matrice AD^2A^T na oblik (6.2.2) kao i odgovarajuće razlaganje vektora $AS^{-1}r_{xs}$. Razlažemo svaki od vektora $\Delta\lambda^{aff} = Dlaff$ i $AS^{-1}r_{xs} = vRight$ u dva odgovarajuća bloka, označena sa δ_1, δ_2 i v_1, v_2, respektivno. Sledeća rutina se koristi za proveru da li je sistem oblika (2.1.4) ili sistem oblika (6.2.1)-(6.2.2) rešavan. Jednačine (2.1.5) i (2.1.6) ostaju iste u oba slučaja.

```
DXaff=DiagonalMatrix[Dxaff];     DSaff=DiagonalMatrix[Dsaff];
rxs=N[-mi*SigCent,20]*Table[1,{n}]+N[DXaff.DSaff,20].Table[1,{n}];
If[pok==0, (* The case when the stabilization is not already applied *)
  If[(vR<Dimensions[a][[1]])&&( MatrixConditionNumber[N[a.d.Transpose[a]]]>eps3)
                    &&(NumStep>1),
    (* Stabilization is used because of $cond(A)$ *)
     vRight=N[a.is.rxs,20];
    vRes=N[LinearSolve[vH-Transpose[vG].vF.vG,
               Drop[vRight,vR]-Transpose[vG].vF.Take[vRight,vR]],20];
     Dlcc=N[vF.Take[vRight,vR]-vF.vG.vRes,20];    Dlcc=Join[Dlcc,vRes];
    ,
     (* Modification is not applied *)
     Dlcc=N[LinearSolve[a.d.Transpose[a], a.is.rxs],20 ];
  ];
  ,(*pok=1*)
(* The case when the stabilization is already applied *)
   (* $Cond(vF)$ of $cond(vH-vG^T VF^{-1}VG$ causes the elimination *)
   If[vR<Dimensions[a][[1]],
      vRight=N[a.is.rxs,20];
      vRes=N[LinearSolve[vH-Transpose[vG].vF.vG,
                 Drop[vRight,vR]-Transpose[vG].vF.Take[vRight,vR]],20];
      Dlcc=N[vF.Take[vRight,vR]-vF.vG.vRes,20];
      Dlcc=Join[Dlcc,vRes];
      ,   (* Stabilization is not applied *)
      Dlcc=N[LinearSolve[a.d.Transpose[a],a.is.rxs],20];
   ];
];           (* Solve systems (2.5) and (2.6)  *)
Dscc=N[-Transpose[a].Dlcc,20];
Dxcc=N[-is.(rxs+Xmat.Dscc),20]; }
```

Takođe, *Korak* 12 je modifikovan prinudnim izlaskom iz while ciklusa u slučaju kada je uslov

```
x.s>x1.s1 && x1.s1<1
```

zadovoljen, gde $x1$ i $s1$ označavaju vrednosti vektora x i s u prethodnom iterativnom koraku.

U slučaju da je while ciklus završen na normalan način, rezultat se izračunava pomoću sledećeg koda.

```
   pom=Range[Length[permut]];
   pom=Sort[pom,permut[[#1]]<permut[[#2]]&];
   x= x[[pom]];      ce=ce[[pom]];
   z=-N[ce.x,20]
   Return[{N[z,20],Take[N[x,20],k1]}]; }
```

U slučaju kada je primenjen prinudan izlazak, koristimo jednostavnu korektivnu proceduru. Izlazna vrednost je tada generisana identičnim kodom, zamenjujući promenljive x i s njihovim prethodnim vrednostima $x1$ i $s1$, respektivno.

6.3 Numerički primeri

U ovom poglavlju dajemo numeričke rezultate primene naše implementacije Mehrotrinog primal-dual algoritma. Dve varijante algoritma su implementirane u programskom paketu MATHEMATICA [108, 109, 110, 111]. Prva verzija je direktna primena Algoritma 2.1. Modifikovana verzija je kombinacija Algoritma 2.1 i Procedure 6.1.1. Ove dve verzije su upoređene sa poznatim kodovima koji se zasnivaju na primal-dual algoritmu PCx [14], MOSEK [1] i LIPSOL [123]. Test primeri sadrže klasu degenerisanih test problema iz [6]. Da bi smo ilustrovali činjenicu da je simpleks metod numerički stabilan i na degenerisanim problemima, isti primeri su rešavani i kodom koji se zasniva na simpleks metodu. Implementacija Mehrotraovog metoda i stabilizacije je takođe primenjena na neke *Netlib* probleme.

Primer 6.3.1 Primenom modifikovanog metoda na problem (4.2.1) dobijamo optimalnu vrednost $4.689819293851613\times10^{-17}$, i optimalnu tačku

$\{1000.0000000000003, 0.009999999999990915, 4.465972084095311 \times 10^{-19},$
$9.290319146020935 \times 10^{-18}, 2.6977763021245666 \times 10^{-17}\}$.

Originalan Mehrotraov primal-dual metod ne može da reši ovaj problem: zaustavlja se posle 10 iteracija sa sistemom linearnih jednačina koji nema rešenja. Još više, interesantno je da PCx daje optimalnu vrednost $1.48461670 \times 10^{-10}$ i nedopustiv status.

Primer 6.3.2 Sledeći primer je test problem (4.2.2). Modifikovan metod, koristeći preciznost 10^{-15}, daje optimalnu vrdnost 899.9999999999999 i optimalnu tačku

$\{1.883068181443366\times10^{-14}, 14.999999999999979, 1.1048249711401044\times10^{-14}\}$

posle 14 iteracija. Originalni metod posle pet iteracija nailazi na loše uslovljenu matricu i staje posle sedam iteracija sa sistemom linearnih jednačina koji nema rešenja.

Primer 6.3.3 U radovima [6], [64], data je klasa loše uslovljenih test problema. U svim slučajevima (osim *Hager 1*) vrednost ciljne funkcije je jednaka nuli. U [6] neki od problema su rešavani sa PCx, HOPDM i solverom iz Microsoft Excel 8.0a. U Tabeli 6.3.1 dajemo vrednosti ciljne funkcije primalnog problema dobijene primenom PCx. U većini slučajeva rezultati nisu tačni. Pored toga, primetimo da rezultati zavise od procesora.

Problem	PCx (Pentium IV)	PCx (Pentium III)	Br. it.	Status
Little	1.48461670e-010	1.48461670e-010	6	Infeasible
Hager	7.75911235e-005	3.16756489e-003	13	Unknown
Hager1	7.85089570e-002	5.64193210e-002	13	Unknown
Hager2	5.54632489e-003	4.11184994e-003	13	Unknown
07-20-02	3.52750000e002	0	11	Infeasible
10-20-06	1.10673751e-014	6.18444502e-010	3	Unknown
10-20-11	-2.33440078e-005	-1.72408763e-005	10	Infeasible
10-20-12	3.73678721e-006	2.69583792e-003	12	Infeasible
10-20-13	2.04945591e-005	1.44417032e-007	12	Unknown
15-30-03	8.30517615e-009	6.22298978e-002	8	Unknown
15-30-04	-1.62595431e004	-1.07999616e004	11	Unknown
15-30-06	3.50522624e-002	1.01731404e-004	8	Unknown
15-30-07	9.72422879e003	2.93026691e003	10	Unknown
15-60-09	-1.20545931e004	-6.55982227e003	7	Unknown
20-40-05	4.13614433e003	4.19202481e000	9	Unknown
30-60-05	-2.58378496e-004	2.37138986e-001	5	Optimal

Tabela 6.3.1

Mnogo bolji rezultati su dobijeni korišćenjem kodova LIPSOL (Tabela 6.3.2) i MOSEK (Tabela 6.3.3). Naime, oba koda koriste rutine za generisanje bazičnog optimalnog rešenja koje se značajno razlikuje od rešenja koje je generisano primal-dual metodom. To se može videti u Tabeli 6.3.3, gde kolone MOSEK IP i MOSEK BAS daju optimalno prima-dual rešenje i vrednost bazičnog rešenja, respektivno. I pored toga, LIPSOL ne može da reši problem 07-20-02, i daje pogrešno rešenje za 10-20-11. MOSEK primal-dual rešenja su tačna samo za neke probleme. Sa druge strane, MOSEK bazično rešenje ima manju tačnost od rešenja iz Tabele 6.3.4 koja su dobijena primenom stabilizacione procedure.

Problem	LIPSOL	Status
Little	3.4177496495e-008	Converged
Hager	3.6578521551e-010	Converged
Hager1	8.0000200580e-002	Converged
Hager2	2.3283064365e-010	Converged
07-20-02	-	Not converged
10-20-06	1.5773489034e-010	Converged
10-20-11	-2.1375811612e-001	Converged
10-20-12	1.0585697416e-020	Likely converged but error $\geq$ tolerance
10-20-13	3.6362725826e-021	Likely converged but error $\geq$ tolerance
15-30-03	1.3295987516e-009	Converged
15-30-04	1.1530965567e-003	Converged
15-30-06	3.2699876586e-009	Converged
15-30-07	-5.9604644775e-007	Converged
15-60-09	-3.2091122637e-004	Converged
20-40-05	3.9942264557e-003	Converged
30-60-05	-5.2619215398e-007	Converged

Tabela 6.3.2

Problem	MOSEK IP	IP Status	MOSEK BAS
Little	0.00000000e+000	Feasible	0.00000000e+000
Hager	1.26560371e-005	Feasible	4.07935617e-012
Hager1	8.00000000e-002	Feasible	8.00000000e-002
Hager2	4.78001311e-007	Feasible	4.07453626e-010
07-20-02	-9.79658597e003	Unknown	-1.44988298e-005
10-20-06	1.09442797e-004	Feasible	3.09810931e-008
10-20-11	2.30185688e-005	Feasible	-4.53250825e-008
10-20-12	1.44992676e007	Unknown	-3.15392512e-004
10-20-13	2.2441842e004	Unknown	-3.00129e-004
15-30-03	2.95747574e-005	Feasible	-7.16417263e-018
15-30-04	1.72796838e004	Unknown	3.15442681e-004
15-30-06	1.478620e004	Unknown	-1.41331e-004
15-30-07	-3.11335992e007	Unknown	3.81857157e-004
15-60-09	-1.50742544e007	Unknown	-4.36755330e-004
20-40-05	-1.68255476e007	Unknown	3.66744919e-004
30-60-05	3.69986004e-005	Feasible	-8.01040996e-008

Tabela 6.3.3

Naša implementacija Mehrotraovog metoda takođe ne rešava ove probleme. Svi ti problemi su rešeni modifikovanim metodom primenom stabilizacione procedure. U Tabeli 6.3.4 dati su odgovarajući rezultati. Kolona *Primal-dual* sadrži rezultate generisane našom implementacijom Mehrotraovog algoritma, i kolona označena sa *Modification* sadrži rezultate dobijene sa stabilisanom verzijom Mehrotraovog algoritma. Kolona MarPlex sadrži rezultate dobijene primenom naše implementaije simpleks metoda u programskom jeziku Visual Basic [99]. Rezultati označeni sa * su dobijeni korišćenjem koda CPLEX.

Problem	Dimensions	Primal-dual	Modification	No. It.	MarPlex
Little	3×5	–	4.68981929e-017	14	0.00000000e000
Hager	18×27	0.01177870e000	2.31595656e-013	19	0.00000000e000
Hager1	18×27	0.04643150e000	0.08000000e000	19	0.08000000e000
Hager2	18×27	983.985462e000	-1.13927731e-012	17	0.00000000e000
07-20-02	9×22	-43326.7103e000	1.71702249e-004	13	-2.12028566e-008
10-20-06	10×20	–	5.21750702e-014	16	0.00000000e000
10-20-11	12×20	–	-1.65960575e-010	26	1.01339538e-007
10-20-12	12×20	0.98274344e000	6.06765507e-015	12	0.00000000e000
10-20-13	12×22	0.62207356e000	6.11841761e-015	12	0.00000000e000
15-30-03	17×32	0.98876311e000	5.43630863e-013	14	-9.41872713e-009
15-30-04	16×31	46.9218065e000	-4.44262662e-004	12	3.63477739e-007
15-30-06	16×31	0.00615304e000	2.02024065e-007	9	3.74187948e-007
15-30-07	17×32	-96148.2245e000	-5.73460485e-005	12	3.63477739e-007
15-60-09	16×61	279069.384e000	6.69293887e-009	15	-4.42993286e-005*
20-40-05	22×42	-89.0574426e000	-1.72587897e-006	11	1.05961369e-004*
30-60-05	30×59	2.65006827e007	-4.23620577e-009	15	0.00000000e000

Tabela 6.3.4

Primer 6.3.4 Naš kod je takođe primenjen na nekim *Netlib* test problemima. Originalni i modifikovani Mehrotraov algoritam daju skoro identične rezultate za skoro sve probleme. Primetimo da naša implementacija Mehrotraov primal-dual algoritma ne može da reši problem $Scfxm1$, ali sa modifikacijom problem $Scfxm1$ je rešen. Takođe, problem *Bandm* nije korektno rešen bez stabilizacije. Sa druge strane, primetili smo mala numerička odstupanja kada se stabilizacija primeni

na probleme *Agg*, *Agg*2, *Blend*, *Sctap*1 i *Share*2*b*. To je verovatno posledica grešaka koje indukuje Gaussova eliminacija. Zaključujemo da je primena stabilizacione procedure razumna samo na loše uslovljenim problemima.

Problem	Primal-dual	Modification
Adlittle	2.2549496316e005	2.2549496316e005
Afiro	-4.6475314286e002	-4.6475314286e002
Agg	-3.5991767284e007	-3.5991767295e007
Agg2	-2.0239252356e007	-2.0239252351e007
Agg3	1.0312115935e007	1.0312115935e007
Bandm	-1.5862801770e002	-1.5862801845e002
Blend	-3.0812149846e001	-3.0812149691e001
Capri	2.6900129138e003	2.6900129138e003
Degen2	-1.4351780000e003	-1.4351780000e003
Israel	-8.9664482186e005	-8.9664482186e005
Kb2	-1.7499001299e003	-1.7499001299e003
Lotfi	-2.5264706062e001	-2.5264706062e001
Recipe	-2.6661600000e002	-2.6661600000e002
Sc105	-5.2202061212e001	-5.2202061212e001
Sc205	-5.2202061212e001	-5.2202061212e001
Sc50a	-6.4575077059e001	-6.4575077059e001
Sc50b	-7.0000000000e001	-7.0000000000e001
Scagr7	-2.3313898243e006	-2.3313898243e006
Scfxm1	–	1.8416759028e004
Sctap1	1.4122500000e003	1.4122501183e003
Share2b	-4.1573224072e002	-4.1573224074e002
Stocfor1	-4.1131976219e004	-4.1131976219e004

Tabela 6.3.5

Na kraju napomenimo da, kako se metod bazira na Gaussovoj metodi eliminacije, česta primena stabilizacione procedure može da prouzrokuje akumulaciju grešaka zaokruživanja. Zbog toga je prikazanu proceduru preporučljivo koristiti periodično u problemima u kojima se javljaju ekstremno loše uslovljene matrice.

7 Modifikovani afini algoritam

U ovom poglavlju predlažemo jednostavan afini algoritam koji se odlikuje velikom numeričkom stabilnošću. Ideja za modifikaciju potiče iz rada [61]. Implementacijom ovog algoritma uspešno su rešeni svi navedeni loše uslovljeni problemi.

7.1 Afini algoritam i implementacija

Korak 1. Generisati početnu tačku x^0.

Korak 2. Izračunati rezidume

$$r_b = Ax^0 - b, \qquad r_c = A^T\lambda^0 + s^0 - c$$

i proveriti kriterijum za kraj algoritma $\frac{|c^T x - b^T \lambda|}{1+|c^T x|} \leq \epsilon$. Ako je kriterijum za kraj ispunjen, izlazni podatak je x^{k+1}; u suprotnom, ići na *Korak 3.*

Korak 3. Formirati matrice S, X i vektor e.

Korak 4. Neka je $D = S^{-1/2}X^{1/2}$, i $r_{xs} = XSe$ i rešimo sledeći sistem

$$\begin{aligned} AD^2A^T\Delta\lambda^{aff} &= -r_b - A(S^{-1}Xr_c - S^{-1}r_{xs}), \\ \Delta s^{aff} &= -r_c - A^T\Delta\lambda^{aff}, \\ \Delta x^{aff} &= -S^{-1}(r_{xs} + X\Delta s^{aff}). \end{aligned}$$

Korak 5. Izračunati uslov za nenegativnost iteracione tačke

$$\begin{aligned} \alpha_{aff}^{pri} &= \max\{\alpha \in [0,1] \,:\, x^k + \alpha\Delta x^{aff} \geq 0\} \\ \alpha_{aff}^{dual} &= \max\{\alpha \in [0,1] \,:\, s^k + \alpha\Delta s^{aff} \geq 0\}. \end{aligned}$$

Korak 6. Staviti $(\Delta x^k, \Delta\lambda^k, \Delta s^k) = (\Delta x^{aff}, \Delta\lambda^{aff}, \Delta s^{aff})$.

Korak 7. Izračunati

$$\begin{aligned} \alpha_{max}^{pri} &= \max\{\alpha \geq 0 \,:\, x^k + \alpha\Delta x^k \geq 0\} \\ \alpha_{max}^{dual} &= \max\{\alpha \geq 0 \,:\, s^k + \alpha\Delta s^k \geq 0\}. \end{aligned}$$

Korak 8. Staviti

$$\begin{aligned} \alpha_k^{pri} &= \min\{0.99\alpha_{max}^{pri}, 1\} \\ \alpha_k^{dual} &= \min\{0.99\alpha_{max}^{dual}, 1\}. \end{aligned}$$

Korak 9. Izračunati sledeću iteraciju

$$\begin{aligned} x^{k+1} &= x^k + \alpha_k^{pri}\Delta x^k, \\ (\lambda^{k+1}, s^{k+1}) &= (\lambda^k, s^k) + \alpha_k^{dual}(\Delta\lambda^k, \Delta s^k). \end{aligned}$$

Zatim primeniti transformaciju

$$x = \overline{x}^{t+1} = x^t + \alpha_k(x^{t+1} - x^t),$$

gde je α_k realan broj koji zadovoljava $0 < \alpha_k < 1$ i preći na *Korak 2.*

U *Koraku 9* primenimo zamenu

```
dx=Dxaff;  dl=Dlaff;   ds=Dsaff;
```

Ostali detalji implementacije su analogni odgovarajućim koracima u Mehrotraovom algoritmu.

7.2 Numerički primeri

Primer 7.2.1 U ovom primeru dajemo vrednosti mere dualnosti xs koje verifikuju numeričku stabilnost i konvergenciju nekih test primera.

Afiro:

{2.09131031484025964$\times 10^{6}$, 717870.2, 178650.3, 3178.164, 802.9825,
365.1796, 120.9447, 41.31498, 26.45259, 3.55117, 0.0674026✓,
0.00327542, 0.000120382, .85961511355211195$\times 10^{-5}$,
.36281499330046743$\times 10^{-6}$,.23982647832721707$\times 10^{-7}$,
1.05699567815953665$\times 10^{-9}$}

Blend:

{14442.41,5493.112, 3476.63, 1815.434, 734.1495, 142.1366, 42.09287,
15.52328, 9.826819, 6.14928, 3.02126, 1.43341, 0.950946, 0.499918,
0.167127, 0.0408819, 0.00494338, 0.000283267✓,
.44466400330290623$\times 10^{-5}$, .7795079018220674$\times 10^{-7}$,
1.71453952334643489$\times 10^{-9}$}

Kb2:

{379788.1, 218215.2, 88425.23, 50764.44, 26809.45, 14529.56, 9044.567,
8458.448, 7488.039, 911.929, 756.411, 705.527, 635.302, 643.668, 565.415,
500.207, 489.106, 427.418, 422.982, 346.793, 294.856, 291.866, 275.545,
222.275, 173.444, 106.912✓, 36.7028, 35.6593, 31.7892, 24.4521,
16.8164, 9.32001, 4.03164, 2.21231, 0.91458, 0.473405, 0.291879,
0.121496, 0.0169711, 0.00196154, 0.000650803, 7.74735416706757629$\times 10^{-6}$}$_{41}$

Adlittle:

{7.3076710216085603$\times 10^{6}$, 4.18579049157574267$\times 10^{6}$, 1.52389518770815923$\times 10^{6}$,
464181.4, 235050.5, 92340.6, 44810.17, 22558.18, 7628.979, 4468.44, 2964.28,
1368.54✓, 896.732, 684.051, 499.263, 398.103, 229.3717, 69.32518,
9.08286, 0.847247, 0.0447238, 0.000599578, 0.0000153751,2.90569345853342007$\times 10^{-7}$}

Sc50a: ($\alpha = 0.98$:)

{97873.41, 6501.212, 847.0143, 64.14744, 48.21975, 40.72946, 36.13577,
22.13138, 9.188619, 3.50789, 1.87901, 1.11681, 0.643128, 0.0786037,
0.0024431, 0.0000392868, 7.83566461175516693$\times 10^{-7}$✓,
3.50005943189752377$\times 10^{-8}$, 5.42850445053744934$\times 10^{-10}$}$_{18}$

Primetimo da je broj iteracija veći od klasičnog algoritma. Međutim, numerička stabilnost algoritma je povećana.

8 Zaključak

U ovoj glavi su razmatrani teorijski i praktični problemi implementacije Mehrotraovog primal dual algoritma. Realizovana je implementacija u programskom paketu MATHEMATICA i izvršeno je poređenje sa već postojećim kodovima. Pokazano je da u odnosu na kod $PrimalDualLP$ iz [10] koji je takođe realizovan u programskom paketu MATHEMATICA, implementacija MPD u svakom pogledu ima znatno bolje karakteristike. U poređenju sa jačim programima (PCx, LIPSOL, LOQO, HOPDM) implementacija MPD je u prednosti kada je u pitanju veća preciznost, rešavanje nekih slabo uslovljenih primera i unošenje podataka u klasičnom obliku. Sa druge strane, kod MPD je sporiji od pomenutih programa i ograničen je po pitanju dimenzije

problema.Većina dostupnih test problema je data u MPS formatu čija je glavna karakteristika zapis matrice problema po kolonama. Takav format prevedimo u mnogo prirodniji i korisnicima dostupniji NB format koji koristi MATHEMATICA i u kome se problem zapisuje na klasičan način tako da ne zahteva nikakvo posebno predznanje od korisnika. Za taj cilj smo koristili programsko okruženje Delphi.

Kod MPD je realizovan sa idejom da se na njemu izvrše sledeća istraživanja:

- uticaj smanjenje dimenzije problema eliminacijom vrsta i kolona iz matrice sistema;
- poređenje normalnog i pridruženog sistema jednačina;
- stabilizacija numeričkog procesa primenom procedure iz [64].

Posle eliminaciji vrsta i kolona iz matrice problema i primene predložene heuristike u Mehrotraovom algoritmu imamo sledeće prednosti u odnosu na originalni algoritam:

- modifikovani metod daje bolju preciznost, modifikacija poboljšava stabilnost i centralnost iterativnog niza,
- modifikovani metod je brži u odnosu na originalni, iako sadrži dodatne eliminacije nekih vrsta i kolona i na kraju primenjuje rutinu za rekonstrukciju rešenja.
- modifikacija redukuje procesorsko vreme potrebno za rešavanje problema *Blend* i *Kb2*, uprkos dodatnog iterativnog koraka koji zahteva modifikacija a takođe modifikacija smanjuje dimenziju problema i zahtev za memorijskim prostorom.

Na primerima je pokazano da je proširen sistem u opštem slučaju brži u odnosu na normalni sistem. Kako je tačnost slična u oba pristupa, to primena proširenog sistema ima bolje karakteristike. Posebno, problemi koji sadrže slobodne promenljive se lakše rešavaju primenom proširenog sistema, dok primena normalnih jednačina zahteva nezgodnu reformulaciju problema.

Primrnom stabilizacione procedure razvijen je kôd koji uspešno rešava loše uslovljene probleme, koje ni poznati LP solveri nisu mogli da reše. Stabilizovani kôd je primenjen na nekim *Netlib* test problemima. Originalni i modifikovani Mehrotraov algoritam daju skoro identične rezultate za skoro sve probleme. Primetimo da naša implementacija Mehrotraov primal-dual algoritma ne može da reši neke probleme, ali sa modifikacijom ti problemi su rešeni. Sa druge strane, primetili smo mala numerička odstupanja kada se stabilizacija primeni na neke probleme. To je verovatno posledica grešaka koje indukuje Gaussova eliminacija. Zaključujemo da je primena stabilizacione procedure razumna samo na loše uslovljenim problemima.

Literatura

[1] E.D. Andersen and K.D. Andersen, *The MOSEK interior-point optimizer for linear programming: an implementation of the homogeneous algoritm*, In H. Frenk, K. Roos, T. Terlaky and S. Zang, editors, *High Perfomance Optimization*, Kluwer Academic Publishers, (2000), 197–232.

[2] E.D. Andersen, J. Gondzio, C. Mészáros and X. Xu *Implementation of interior point methods for large scale linear programming*, Technical report, HEC, Université de Genève, 1996.

[3] E.D. Andersen and Y. Ye *Combining interior-point and pivoting algorithms for linear programming*, Technical report, Department of Menagement Sciences, The University of Iowa, 1994.

[4] K.M. Anstreicher, *A monotonic projective algorithm for fractional linear programming*, Algorithmica **1**, (1985) 483–498.

[5] C. Ashcraft, R.L. Grimes and J.G. Lewis, *Accurate symmetric indefinite linear equation solvers*, SIAM Journal on Matrix Analysis, **20**, Issue 2 (1999), 513 – 561.

[6] M.D. Ašić and V.V. Kovačević-Vujčić, *Ill-conditionedness and interior-point methods*, Univ. Beograd Publ. Elektrotehn. Fak. **11**, (2000), 53–58.

[7] E.R. Barnes, *A variation on Karmarkar's algorithm for solving linear programming problems*, Mathematical Programming **36** (1986), 174–182.

[8] V. Baryamureeba and T. Steihaug, *On the convergence of an inexact primal-dual interior point method for linear programming*, Technical report No.188, Department of informatics, Univesity of Bergen (2000), 1–13.

[9] A. Ben-Tal, M. Teboulle, *A geometric property of the least square solution of linear equations*, Linear Algebra and its Applications **139**, (1990), 165–170.

[10] M.A. Bhatti *Practical optimization methods*, Springer, New York, 2000.

[11] R. Bixby, *Implementing the simplex method; the initial basis*, ORSA Journal on Computing **4** (1992), 267–284.

[12] K.H. Borgwardt, *The simplex method: a probablistic analysis*, Springer, Berlin, 1987.

[13] D. Cvetković, M. Čangalović, Dj. Dugošija, V. Vujčić-Kovačević, S. Simić i J. Vuleta, *Kombinatorna optimizacija*, Društvo Operacionih Istraživača Jugoslavije - DOPIS, Beograd, 1996.

[14] J. Czyzyk, S. Mehrotra and S.J. Wright, *PCx User Guide,* Optimization Thechnology Center, Technical Report 96/01, (1996) 1–21.

[15] G.B. Dantzig, *Lineare Programmierung und Erweiterungen*, Springer-Verlag Berlin Heidelberg New York, 1966.

[16] G. De Ghellinck and J. Ph. Vial, *A polynomial Newton method for linear programming*, Algoritmica **1** (1986), 425–453.

[17] D. Den Hertog, C. Roos and T. Terlaky, *A potential reduction variant of Renegar's short-step path following method for linear programming*, Linear Algebra and its Applications **152** (1991), 43–68.

[18] I.I. Dikin, *Iterative solution of problems of linear and quadratic programming*, Soviet Mathematics Doklady **8** (1967), 674–675.

[19] I.I. Dikin, *On the speed of an iterative process*, Upravlyaemye Sistemy **12**, (1974), 54–60.

[20] Juan Dominguez and María D. Gonzáles-Lima, *A primal-dual interior-point algorithm for quadratic programming*, Universidad Simón Bolívar,(CESMa) Apdo 89000, Caracas 1080-A, Technical Report 2005-01. Venezuela

[21] Juan Dominguez and María D. Gonzáles-Lima, *A primal-dual interior-point algorithm for quadratic programming*, Numer Algor 42 (2006), 1–30.

[22] Juan Javier Dominguez Moreno, *Un algoritmo de puntos interiores para programación cuadrática*, Master thesis, Universidad Simón Bolívar, 2004.

[23] I.S. Duff, *The solution of augmented systems*, in Numerical Analysis, D.F. Griffiths and G.A. Watson, eds., Longman Scientific and Technical, Essex,U.K. (1993), 40–45.

[24] A.S. El-Bakry, R.A.Tapia and Y. Zhang, *A study of indicators for identifying zero variables in interior-point methods*, SIAM Rev. **36(1)** (1994), 45–72.

[25] A. Forsgren and G. Sporre, *On weighted linear least-squeares problems related to interior methods for convex quadratic programming*, Technical report TRITA-MAT-2000-OS11, Department of Mathematics, Royal Institute of Technology, 2000.

[26] R. Fourer and S. Mehrotra, *Solving symmetric indefinite systems in an interior-point method for linear programming*, Mathematical Programming **62** (1993), 15–39.

[27] R.M. Freund, *A potential-function reduction algorithm for solving a linear program directly from an infeasible "warm start"*, Mathematical Programming **52** (1991), 441-446.

[28] R. Freund, *Polynomial-time algorithms for linear programmnig based only on primal scaling and projected gradients of a potential function*, Mathematical Programming **51** (1991), 203-222.

[29] R. Freund, *Theoretical efficiency of a shifted-barier-function algorithm for linear programmnig*, Linear Algebra and its Applications **152** (1991), 19-41.

[30] R.M. Freund, *A potential-function reduction algorithm with user-specified phase I-phase II balance for solving a linear program directly from an infeasible "warm start"*, SIAM J. Optimization Vol. **5**, No. 2 (1995), 247–268.

[31] R.M. Freund, *Following a "balanced" trajectory from an infeasible point to an optimal linear programming solution with polynomial-time algorithm*, Mathematics of Operations Research Vol. **21**, No. 4 (1996), 839–859.

[32] R.M. Freund, *An infeasible-start algorithm for linear programming whose complexity depends on the distance from the starting point to the optimal solution*, Annals of Operations Research **62** (1996), 29–57.

[33] R.W. Freund and F. Jare, *A QMR-based interior-point algorithm for solving linear programs*, Mathematical Programming **76** (1996), 183–210.

[34] R.W. Freund, F. Jare and S. Mizuno, *Convergence of a class of inexact interior-point algorithms for linear programs* Mathematics of Operational Research Programming **24** (1999), 50–71.

[35] K.R. Frish, *The logarithmic potential method of convex programming*, Technical report, University Institute of Economics, Oslo, Norway (1955), 183–210.

[36] A.J. Goldman and A.W. Tucker, *Theory of linear programming*, in Linear Equalities and related Systems, H.W. Kuhn and A.W. Tucker, eds., Princeton University Press, Princeton, N.J. (1956), 53–97.

[37] J. Gondzio and T. Terlaky, *A computational view of interior-point methods for linear programming*, Cahiers de recherche, Technical Report, HEC, Université de Genève 1994.22 (1994).

[38] J. Gondzio, *HOPDM (Version 2.12): A fast LP solver based on a primal-dual interior point method*, European Journal of Operations Research **85** (1995), 221–225.

[39] C.C. Gonzaga, *An algorithm for solving linear programming problems in $O(n^3L)$ operations*, in Progress in Mathematical Programming: Interior-Point and Related Methods, N. Megiddo, ed., Springer-Verlag, New York (1988), 1–28.

[40] C.C. Gonzaga and J.F. Bonnans, *Fast convergence of the simplified largest step path following algorithm*, Mathematical Programming **76** (1996), 95–115.

[41] C.C. Gonzaga, *Conical projection algorithms for linear programming*, Mathematical Programming **43** (1989), 151–173.

[42] C.C. Gonzaga, *Polynomial affine algorithms for linear programming*, Mathematical Programming **49** (1990), 7–21.

[43] C.C. Gonzaga, *Interior point algorithms for linear programming with inequality constraints*, Mathematical Programming **52** (1991), 209–225.

[44] C.C. Gonzaga, *Large-step path following method for linear programming, Part I: barrier function method*, SIAM Journal on Optimization **1** (1991), 268–279.

[45] C.C. Gonzaga, *Large-step path following method for linear programming, Part II: potential reduction method*, SIAM Journal on Optimization **1** (1991), 280–292.

[46] C.C. Gonzaga and M.J. Todd, *An $O(\sqrt{n}L)$-iteration large-step primal-dual affine algorithm for linear programming*, SIAM Journal on Optimization **2** (1992), 349–359.

[47] O. Güler and Y. Ye, *Convergence behavior of interior-point algorithms*, Mathematical Programming **60** (1993), 215–228.

[48] N.I.M. Gould, *Iterative methods for ill-conditioned linear systems from optimization*, Technical report, Rutheford Appelton Laboratory, Oxfordshire (1998), 1–18.

[49] A.J. Hoffman, M. Mannos, D. Sokolowsky and N Wiegman, *Computational experience in solving linear programs*, Journal of the Society for Industrial and Applied Mathematics **1** (1953),17–33.

[50] B. Jansen, C. Ross, T. Terlaky and Y. Ye, *Improved complexity using higher-order correctors for primal-dual Dikin affine scaling*, Mathematical Programming **76** (1996), 117–130.

[51] N. Karmarkar, *A new polynomial-time algorithm for linear programming*, Combinatorica **4** (1984), 373–395.

[52] N.K. Karmarkar and K.G. Ramakrishnan, *Computational results of an interior point algorithm for large scale linear programming*, Mathematical Programming **52** (1991), 555–586.

[53] L.G. Khachiyan, *A polynomial algorithm in linear programming*, Soviet Mathematics Doklady **20** (1979), 191–194.

[54] V. Klee and G.L. Minty, *How good is the simplex method*, In: O. Shisha, ed., Inequalities III, Academic Press, New York (1972), 159–175.

[55] M. Kojima, S. Mizuno and A. Yoshise, *A polynomial-time algorithm for a class of linear complemntarity problems*, Mathematical Programming **44** (1989), 1–26.

[56] M. Kojima, S. Mizuno and A. Yoshise, *A primal-dual interior point algorithm for linear programming*, in Progress in Mathematical Programming: Interior-Point and Related Methods, N. Megiddo, ed., Springer-Verlag, New York **ch. 2** (1989), 29–47.

[57] M. Kojima, *Basic lemmas in polynomial-time infeasible-interior-point methods for linear programs*, Annals of Operations Research **62** (1996), 1–28.

[58] M. Kojima, *A primal-dual infeasible-interior-point algorithm for linear programming*, Mathematical Programming, Series A **61** (1993), 261–280.

[59] M. Kojima, S. Mizuno and A. Yoshise, *A polynomial-time algorithm for a class of linear complementarity problems*, Mathematical Programming **44** (1989), 1–26.

[60] M. Kojima, S. Mizuno and A. Yoshise, *An $O(\sqrt{n}L)$ iteration potential redusvtion algorithm for linear complementarity problems*, Mathematical Programming **50** (1991), 331–342.

[61] V.V. Kovačević-Vujčić, *Improving the rate of convergence of interior-point methods for linear programming*, Mathematical Programming **52** (1991), 467–479.

[62] V.V. Kovačević-Vujčić, *Stabilization of path-following interior-point methods for linear programming*, IX Conference on Applied Mathematics, D. Herceg, Lj. Cvetković, eds. Institute of Mathematics, Novi Sad (1995), 47–53.

[63] V.V. Kovačević-Vujčić, *New approaches to linear programming*, SYMOPIS, Beograd, November 4-6 (1993), 5–15.

[64] V.V. Kovačević-Vujčić and M.D. Ašić, *Stabilization of interior-point methods for linear programming*, Computational Optimization and Applications **14** (1999), 331–346.

[65] I.J. Lustig, R.E. Marsten and D.F. Shanno, *Computationial experience with a primal-dual interior point method for linear programming*, Linear Algebra and Its Applications **152** (1991), 191–222.

[66] N. Megiddo, *Pathways to the optimal set in linear programming*, in Progress in Mathematical Programming: Interior-Point and Related Methods, N. Megiddo, ed., Springer-Verlag, New York **ch. 8** (1989), 131–158.

[67] N. Megiddo, *On finding primal- and dual-optimal bases*, ORSA Journal on Computing, **3**(1991), 63–65.

[68] S. Mehrotra and Y. Ye, *Finding an interior point in the optimal face of linear programs*, Mathematical Programming **62** (1993), 497–515.

[69] S. Mehrotra, *On finding a vertex solution using interior point methods*, Linear Algebra and Its Applications **152** (1991), 233–253.

[70] S. Mehrotra, *On the implementation of a primal-dual interior point method* SIAM J. on Optimization **2** (1992), 575–601.

[71] S. Mehrotra and Y. Ye, *Finding an interior point in the optimal face of linear programs*, Mathematical Programming **62** (1993), 497–515.

[72] S. Mizuno, M. Todd and Y. Ye, *On adaptive step primal-dual interior-point algorithms for linear programming* Mathematics of Operations Research **18** (1993), 964–981.

[73] R.D.C. Monteiro and I. Adler *Interior path-following primal-dual algorithms. Part*I*: Linear Programming* Mathematical Programming **44** (1989), 27–41.

[74] R.D.C. Monteiro, I. Adler and M.G.C. Resende, *A polynomial-time primal-dual affine scaling algorithm for linear convex quadratic programming and its power series extension* Mathematics of Operations Research **15** (1990), 191–214.

[75] C.L. Monma and J. Morton, *Computational experience with a dual affine variant of Karmarkar's method for linear programming*, Operations Research Letters **6** (1987), 261–267.

[76] Y. Nesterov, *Long-step strategies in interior-point primal-dual methods*, Mathematical Programming **76** (1996), 47–94.

[77] J. von Neumann, *On a maximization problem*, Technical report, Institute for Advanced Study (Princeton, NJ, USA), (1947).

[78] E. Ng and B.W. Peyton, *A supernodal Cholesky factorization algorithm for shared-memory multiprocessors*, SIAM J. SCI.COMPUT.,Vol. **14**, No. 4, (1993), 761–769.

[79] F.A. Potra, *A quadratically convergent predictor-corrector method for solving linear programs from infeasible starting points*, Mathematical Programming **67** (1994), 383–406.

[80] J. Renegar, *A polynomial-time algorithm, based on Newton's method, for linear programming*, Mathematical Programming **40** (1988), 59–93.

[81] J. Renegar and M. Shub, *Unified complexity analysis for Newton LP metods*, Mathematical Programming **53** (1992), 1–16.

[82] C. Roos and J.Ph. Vial, *Long steps with the logarithmic penalty barrier function in linear programming*, in: J. Gabszewich, J.F. Richard and L. Wolsey,eds, Economic Decision-making: Games, Economics and Optimization. Contributions in Honour of Jacques H. Dreze (Elsevier Science Publisher, Amsterdam, 1990), 433–441.

[83] C. Roos and J.Ph. Vial, *A polynomial method of approximate centers for linear programming*, Mathematical Programming **54** (1992), 295-306.

[84] E. Rothberg, A. Gupta, E. Ng and B.W. Peyton, *Parallel sparse Cholesky factorization algorithms for shared-memory multiprocessors system*, Proceedings of the 7th IMACS International Conf. on Computer Methods for Partial Differential Equations

[85] G. Sonnevend, J. Stoer and G. Zhao, *On the complexity of following the central path of linear programs by linear extrapolation*, Methods of OPerations Research, **62** (1989), 19–31.

[86] G. Sonnevend, J. Stoer and G. Zhao, *On the complexity of following the central path of linear programs by linear extrapolation II*, Mathematical Programming, **52** (1991), 527–553.

[87] P. Stanimirović, N. Stojković, B. Momcilovic and Z. Jovanovic,*Augmented and Normal Equations System in Mehrotra's Primal-dual Algorithm*, Filomat **15**, Niš (2001), 285–292.

[88] P. Stanimirović, N. Stojković and Vera Kovačević Vujčić, *Stabilization of Mehrotra's primal-dual algorithm and its implementation*, SYMOPIS, Beograd (2001), 343–346.

[89] P. Stanimirović, N. Stojković and Vera V. Kovačević-Vujčić, *Stabilization of Mehrotra's primal-dual algorithm and its implementation,* European Journal of Operational Research, Volume bf 165, Issue 3, (2005), 598-609.

[90] P. Stanimirović, N. Stojković and Vera V. Kovačević-Vujčić, *Some implementation details of modified Mehrotra's primal-dual algorithm*, XVI Conference on Applied Mathematics, N. Krejic, Z. Luzanin, eds. Department of Mathematics and Informatics, Novi Sad, (2004),149–156.

[91] N. Stojković and P. Stanimirović, *About the starting point in primal-dual interior point method*, SYMOPIS, Beograd (2000), 261–264.

[92] N. Stojković and P. Stanimirović, *Initial point in primal-dual interior point method*, Facta Universitatis, Ser. Mech. Vol.3, No 11, (2001), 219–222.

[93] N. Stojković, *On the finite termination in the primal-dual algorithm for linear programing*, YUJOR **11** (2001), 31–40.

[94] N. Stojković and P. Stanimirović , *Two direct methods in linear programming*, European Journal of Operational Research **131**, (2001), 417–439.

[95] N. Stojković and P. Stanimirović, *Transformations of Dual Problem and Decreasing Dimensions in Linear Programming*, Matematičiki Vesnik **54**, (2002), 203–210.

[96] N. Stojković and P. Stanimirović, *Decreasing Dimensions in Mehrotra's Primal-Dual Algorithm*, Zbornik radova SYM-OP-IS 2003,(2003), 343–346.

[97] N. Stojković and P. Stanimirović, *Finite Termination and Decreasing Dimensions in Mehrotra's Primal-Dual Algorithm*, The Sixth International Symposium on Nonlinear Mechanics - Nonlinear Science and Applications - Niš 2003. 24 - 29. August, 2003

[98] N.V. Stojković and P.S. Stanimirović, *Inverse and characteristic polynomial of extended triangular matrix*, Publ. Elect. Fac. Beogr. **11** (2000), 71–78.

[99] N. V. Stojković, Predrag S. Stanimirović and Marko D. Petković, *Several Modifications of Simplex Method*, Filomat 17,(2003), 169–176.

[100] N.V. Stojković, *Primal-dual i simpleks metodi za rešavanje problema linearnog programiranja*, Doktorska disertacija, PMF Niš, 2001.

[101] G.W. Stewart, *On scaled projections and pseudoinverses*, Linear Algebra and its Applications **112**, (1989), 189–193.

[102] K. Tanabe, *Centered Newton method for mathematical programming*, in System Modeling and Optimization: Proceedings of the 13th IFIP conference, Lecture Notes in Control and Information Systems 113, Berlin, August/September 1987, Springer-Verlag, New York, (1988), 197–206.

[103] M.J. Todd, *Potential-reduction methods in linear programming*, Mathematical Programming **76** (1996), 3–45.

[104] M.J. Todd and Y.Ye, *A centered projective algorithm for linear programming*, Mathematics of Operations Research **15** (1990), 508–529.

[105] R.J. Vanderbei, *LOQO: an interior point code for quadratic programming*, Statistics and Operations Research, Princeton University, Technical Report SOR-94-15, 1998.

[106] R.J. Vanderbei, M.S. Meketon and B.A. Freedman, *A modification of Karmarkar's linear programming algorithm*, Algorithmica **1** (1986), 395-407.

[107] P.J. Williams, *Effective finite termination procedures in interior-point methods for linear programming*, Doctoral thesis, Houston, Texas, 1998.

[108] S. Wolfram, *Mathematica: a system for doing mathematics by computer*, Addison-Wesley Publishing Co, Redwood City, California, 1991.

[109] S. Wolfram, *Mathematica Book, Version 3.0*, Wolfram Media and Cambridge University Press, 1996.

[110] Wolfram Research, *Mathematica 4.0 standard add-on packages*, Wolfram Media, (1999).

[111] S. Wolfram, *Mathematica Book, 4th edition*, Wolfram Media/Cambridge University Press, (1999).

[112] S.J Wright, *Primal-dual interior-point methods*, SIAM, Philadelphia, 1997.

[113] S.J. Wright, *A path-following interior-point algorithm for linear and quadratic problems*, Annals of Operations Research **62** (1996), 103–130.

[114] S.J. Wright, *An infeasible-interior-point algorithm for linear complementarity problems*, Mathematical Programming, **67** (1994), 29–52.

[115] S.J Wright, *Modified Cholesky factorizations in interior-point algorithms for linear programming*, SIAM Journal on Optimization **9**, (1999), 1152–1191.

[116] H. Yamashita, *A polynomially and quadratically convergent method for linear programming*, Mathematical System Institute (Tokyo, Japan, 1986)

[117] Y. Ye, *On the finite convergence of interior -point algorithms for linear programming*, Mathematical Programming, **57** (1992), 325–336.

[118] Y. Ye, *On homogeneous and self-dual algorithms for LCP*, Mathematical Programming **76** (1996), 211–221.

[119] Y. Ye, *An $O(n^3L)$ potential reduction algorithm for linear programming*, Mathematical Programming **50** (1991), 239–258.

[120] Y. Ye, *On quadratic and $O(\sqrt{n}L)$ convergence of a predictor-corrector algorithm for LCP*, Mathematical Programming **62** (1993), 537–551.

[121] Y. Ye, *A class of projective transformations for linear programming*, SIAM Journal on Computing (1991), 537–551.

[122] Y. Zhang, *On the convergence of a class of infeasible-interior-point method for the horizontal linear complementarity problem*, SIAM J. on Optimization **4** (1994), 208–227.

[123] Y. Zhang, *Solving large-scale linear programs by interior-point methods under the MATLAB environment*, Optimization Methods and Software **10** (1998), 1–31.

Glava 3

Primena linearnog programiranja

1 Uvod

U okviru ove glave razmatraće se formulacija nekih optimizacionih zadataka koji imaju poseban praktični značaj, a svode se na primenu linearnog programiranja. Međutim, u praksi se najčešće postavlja pitanje: kako formulisati matematičke modele i pripremiti konkretne podatke iz raznih oblasti primene linearnog programiranja, tako da bi se mogli upotrebiti računari i gotove rutine (paketi programa) koje su razvijene specijalno za rešavanje ovih zadataka?

Od odgovora na ovo pitanje zavisi konkretizacija svega onoga što je poznato u primenjenoj matematici, a odnosi se na najrazličitije postupke rešavanja zadataka linearnog programiranja. Zbog toga će se dati formulacija nekih zadataka kod kojih primena linearnog programiranja igra značajnu ulogu sa stanovišta pripremanja i donošenja optimalnih upravljačkih odluka.

Matematički model određenog zadatka koji se svodi na linearno programiranje nikada ne može da obuhvati sve prirodne pojave i da bude verna slika stanja koje se sa njim želi prikazati, jer najčešće obuhvata samo važnije pojave. Zbog toga najkvalifikovaniji i najodgovorniji deo posla leži na onima koji formulišu zadatke. Od njih zavisi izbor karakterističnih pojava najvažnijih za dati problem, koje se kao takve uključuju u odgovarajući matematički model.

Kod formiranja upravljačkih zadataka posebnu pažnju trebalo bi posvetiti na izbor pokazatelja kvaliteta rešenja odgovarajućeg zadatka. Po pravilu, karakteristika koja se optimizira (funkcija cilja) predstavlja pokazatelj kvaliteta rešenja. Dopunske granice karakteristika rešenja određuju se konstrukcijom dodatnih ograničenja.

Sa gledišta izbegavanja većih teškoća u iznalaženju optimalnog plana, poželjno je izvršiti izbor upravljačkih promenljivih, tako da skup ograničenja bude što jednostavniji. To će se postići ako fizički i prirodni uslovi zadatka drugačije ne zahtevaju.

Postavka i rešavanje zadataka koji se svode na LP ne izvode se uvek samo u jednom potezu. To je najčešći slučaj sa zadacima kod kojih odgovarajući matematički model nije dobro formulisan, pa ga treba poboljšati i bolje približiti realnim uslovima. Zato se rešavanju zadataka koji se svode na linearno programiranje nikada ne sme pristupiti jednostrano. Na njima, kao i na drugim zadacima optimizacije, mora raditi tim specijalista koji dobro poznaje prirodu problema i matematičke metode iznalaženja odgovarajućeg optimalnog rešenja.

U prvom poglavlju su opisane neke primene linearnog programiranje koje su zasnovane na transportnom modelu, kao što su transportni zadatak na putnoj mreži, optimizacija železničkog transporta, minimizacija vremena transporta.

Naredno poglavlje izučava neke klasične primene linearnog programiranja, koje su često zastupljene u literaturi. Najpoznatije primene su: optimalni program proizvodnje, optimizacija utroška materijala, izbor sastava mešavine, problem ishrane, neke primene u poljoprivredi, proizvodnji i ishrani. Ove primene su opisane u knigama [5, 6, 7]. Za svaki matematički model opisan u ovom poglavlju data je njegova implementacija u programskom paketu MATHEMATICA.

Poslednja dva poglavlja opisuju neke nestandardne primene linearnog programiranja, koje su retko zastupljene u literaturi. Prva od takvih primena se odnosi na projektovanje teleskopa (problem optike), druga na projektovanje digitalnih FIR filtara koji imaju veliku ulogu u obradi digitalnih signala (problem telekomunikacija), a treća na linearnu regresiju. Modeli L_1 i L_∞ su implemenirani u programskom paketu MATHEMATICA.

2 Primene zasnovane na transportnom modelu

2.1 Transportni zadatak u putnoj mreži

Kod rešavanja određenih transportnih problema, često se za njih vezuje konkretna putna (transportna) mreža u okviru koje se definišu svi punktovi $P_1, \ldots, P_N$ od kojih su neki međusobno povezani komunikacijama K_{ij}. Postojanje komunikacije K_{ij} znači da se iz punkta P_i u punkt P_j može vršiti transport, ali ne i obrnuto, što znači da pojam komunikacije ima značenje jednosmernog kretanja. Dvosmerno kretanje između punktova P_i i P_j podrazumeva postojanje dve komunikacije K_{ij} i K_{ji}. Svaki punkt putne mreže može se okarakterisati brojem p_i $(i = 1, \ldots, N)$, koji ima značenje, na primer, za organizaciju transporta proizvedene ili uskladištene robe, obima proizvodnje ili obima uskladištene robe u punktu P_i. U zavisnosti od znaka veličine p_i, svi punktovi P_i $(i = 1, \ldots, N)$ dele se na punktove proizvodnje (skladištenja) i na punktove potrošnje. Za punktove proizvodnje uzima se da su vrednosti p_i pozitivne, a za punktove potrošnje da su negativne, dok za punktove koji su samo tranzitne stanice uzima se da $p_i = 0$. Svakoj komunikaciji K_{ij} pridružuju se dva karaktcristična broja:

d_{ij} - maksimalna propusna sposobnost komunikacije K_{ij} i

c_{ij} - cena prevoza jedinice mere proizvoda koji se transportuje iz punkta P_i u punkt P_j korišćenjem komunikacije K_{ij}.

Čak i u slučaju kada ne postoji komunikacija K_{ij} između punktova P_i i P_j, može se podrazumevati njeno postojanje, ali pod uslovom da se uzme u obzir da je njena propusna moć $d_{ij} = 0$.

Zadatak se sastoji u sledećem: potrebno je odrediti plan transporta, tj. skup vrednosti x_{ij} koje zadovoljavaju ograničenja:

$$\begin{aligned} \sum_{j=1}^{N} x_{ij} - \sum_{j=1}^{N} x_{ji} &= p_i, \ (i = 1, \ldots, N) \\ 0 \leq x_{ij} &\leq d_{ij}, \end{aligned} \tag{2.1.1}$$

za sve komunikacije, uzimajući u obzir i one koje ne postoje, za koje je $d_{ij} = 0$, a da pri tome funkcija cilja

$$F = \sum_{i=1}^{N} \sum_{j=1}^{N} c_{ij} x_{ij} \tag{2.1.2}$$

dobije minimalnu vrednost. Lako se može zapaziti da količina x_{ij} koju treba transportovati po bilo kojoj komunikaciji putne mreže K_{ij} ulazi u levu stranu izraza (2.1.1) dva puta: jedanput kao količina koju treba transportovati iz punkta P_i sa znakom plus, a drugi put kao količina koju treba transportovati iz punkta P_j sa znakom minus. Zbog toga je potrebno zadovoljenje ravnoteže proizvodnje (skladištenja) i potrošnje, a posledica toga je da imamo

$$\sum_{i=1}^{N} p_i = 0. \tag{2.1.3}$$

Prema tome, za egzistenciju transportnog plana neophodno je, pored uslova (2.1.1), zadovoljenje i uslova (2.1.3). Tako se i ovaj zadatak dovodi u neposrednu vezu sa transportnim zadatkom.

2.2 Optimizacija železničkog transporta

Ekonomično iskorišćenje vagona na železnici predstavlja izuzetno aktuelan i značajan zadatak, koji se sastoji u raspodeli vagona sa ciljem udovođenja određenim zahtevima za transport robe. Kada je reč o raspodeli vagona ne misli se samo na jedan tip, već na različite tipove vagona (zatvoreni, poluvagoni, platforme, itd., sa različitim brojem osovina).

Označimo sa:

x_{ij} - broj vagona j-tog tipa napunjenih i-tom robom;

a_{ij} - normu opterećenja jednog vagona j-tog tipa i-tom robom;

a_i - količinu robe koju treba transportovati u tonama;

b_j - raspoloživi broj vagona j-tog tipa;

c_{ij} - eksploatacione rashode koji nastaju usled transporta i-te robe na jednom vagonu j-tog tipa.

Zadatak optimalnog rasporeda vagona sastoji se u izračunavanju nepoznatih ,x_{ij} koje minimiziraju funkciju cilja

$$F = \sum_{j=1}^{n} \sum_{i=1}^{m} c_{ij} x_{ij},$$

pri ograničenjima

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_{ij} &= a_i, \ (i = 1, \ldots, m), \\ \sum_{i=1}^{n} x_{ij} &\leq b_j, \ (j = 1, \ldots, n), \\ x_{ij} &\leq 0, \ (i = 1, \ldots, n, j = 1, \ldots, n). \end{aligned}$$

Funkcija cilja F određuje ukupne troškove transporta. Prvi skup ograničenja ukazuje da sva roba mora biti transportovana, a drugi skup da je broj vagona pojedinih tipova ograničen. Na ovaj način mogu se rešavati slični problemi rečnog, morskog i drugih vrsta transporta kako robe, tako i putnika.

2.3 Minimizacija vremena transporta

U nizu transportnih zadataka kako vojnih, tako i civilnih, kvalitet organizacije transporta meri se utrošenim vremenom na njegovo sprovođenje (izlazak na vatreni položaj, kampanja pšenice, kukuruza, voća, povrća i drugih proizvoda).

Označimo sa t_{ij} vreme utrošeno na transport proizvoda iz i-tog punkta proizvodnje u j-ti punkt potrošnje. Neka su punktovi proizvodnje označeni sa A_i u kojima su proizvedene količine a_i $(i = 1, \ldots, m)$, a punktovi potrošnje sa B_j u kojima postoje potrebe za količinama b_j $(j = 1, \ldots, n)$, izraženim u istim jedinicama mere kao i vrednosti a_i.

Zadatak optimizacije se sastoji u određivanju plana transporta $\|x_{ij}\|$, tj. skupa vrednosti x_{ij} $(i = 1, 2, ..., m;\ j = 1, 2, ..., n)$ za koje će vreme $t(X)$ najdužeg trajanja prevoza

$$\begin{aligned} t(X) &= \max\ t_{ij}, \\ x_{ij} &> 0. \end{aligned} \tag{2.3.1}$$

postati minimalno pri skupu ograničenja

$$\begin{aligned} \sum_{j=1}^{n} x_{ij} &\leq a_i, \ (i = 1, \ldots, m), \\ \sum_{j=1}^{n} x_{ij} &= b_j, \ (j = 1, \ldots, n), \\ x_{ij} &\geq 0, \ \ (i = 1, \ldots, m, \ j = 1, \ldots, n). \end{aligned}$$

Funkcija cilja (2.3.1) koja se minimizira predstavlja najduže vreme iz skupa vremena t_{ij}- trajanje transporta iz bilo koga punkta A_i u punkt B_j gde su predviđene količine transporta ($x_{ij} > 0$). Ovako formulisani zadatak ne spada u okvire linearnog programiranja, obzirom da je funkcija cilja $t(X)$ nelinearna funkcija opromenljivih x_{ij}. Zadaci ove vrste rešavaju se svođenjem na sukcesivno rešavanje serije običnih transportnih zadataka, ili na rešavanje o maksimalnom protoku gde se koriste poznati, ali veoma složeni algoritmi Hitchcock-a, Forda, Fulkersona.

3 Klasična primena linearnog programiranja

3.1 Optimalni program proizvodnje

Sastaviti optimalni program proizvodnje znači izabrati, između velikog broja proizvoda, onaj asortiman proizvoda koji će obezbediti maksimalno ekonomske efekte iz ograničene količine proizvodnih resursa.

Pretpostavimo da preduzeće može proizvoditi n različitih tipova proizvoda: P_1, P_2, ..., P_n. Za proizvodnju preduzeće koristi m različitih vrsta mašina, r različitih kategorija radnika i g različitih vrsta sirovina u ograničenim količinama. Poznat je dohodak koji se ostvaruje po jedinici svakog proizvoda, pa je problem kako preduzeće da programira proizvodnju da bi ostvarilo najveći mogući dohodak u poslovanju.

Matematički model ćemo formulisati korišćenjem sledećih simbola:

x_j - količina j-tog proizvoda koju treba proizvesti prema optimalnom programu proizvodnje, a koju treba odrediti pomoću matematičkog modela;

c_j - dohodak po jedinici j-tog proizvoda;

a_{ij} - vreme koje je potrebno da se na i-toj mašini proizvede jedinica j-tog proizvoda;

a_{i0} - kapacitet i-te mašine izražen u vremenskim jedinicama;

b_{kj} - vreme potrebno radniku k-te kategorije da obradi, odnosno proizvede, jedinicu j-tog proizvoda;

b_{k0} - raspoloživi fond radnog vremena radnika k-te kategorije;

s_{vj} - količina v-te sirovine koja je potrebna za proizvodnju jedinice j-tog proizvoda;

s_{v0} - raspoloživa količina v-te vrste sirovine;

e_j - količina j-tog proizvoda koja se može prodati na trzištu.

Pomoću uvedenih simbola formulišemo matematički model. On ima funkciju kriterijuma

$$z_0 = \sum_{j=1}^{n} c_j x_j,$$

koja označava dohodak od celokupne proizvodnje, pa treba naći njenu maksimalnu vrednost uz sledeće ograničavajuće faktore:

$$\begin{aligned}
\sum_{j=1}^{n} a_{ij}x_j &\le a_{i0}\ ,\ i = 1, 2, \ldots, m,\\
\sum_{j=1}^{n} b_{kj}x_j &\le b_{k0}\ ,\ k = 1, 2, \ldots, r,\\
\sum_{j=1}^{n} s_{vj}x_j &\le s_{v0}\ ,\ v = 1, 2, \ldots, g,\\
0 \le x_j &\le e_j\ ,\ j = 1, 2, \ldots, n.
\end{aligned} \tag{3.1.1}$$

```
Dodaj[a_,b_]:=
  Module[{m=a,i,n=Length[b]},
      For[i=1,i<=n,i++,m=AppendTo[m,b[[i]] ]  ];
      Return[m]
  ]

OptimalniProgram[c_,a_,a0_,b_,b0_,s_,s0_,e_]:=
  Module[{m=-a,v=-a0,n=Length[c],i},
     m=Dodaj[m,-b];  m=Dodaj[m,-s]; m=Dodaj[m,-IdentityMatrix[n]];
     v=Dodaj[v,-b0]; v=Dodaj[v,-s0];
     For[i=1,i<=n,i++,v=AppendTo[v,-e[[i]]]];
     LinearProgramming[-c,m,v]
  ]
```

Primer 3.1.1 *U ovom primeru uočimo neke konkretne vrednosti za veličine koje su neophodne u matematičkom modelu* (3.1.1).

```
In[1]:= c = {2,6,2,3,5};
        a = {{1,3,2,1,4}, {3,1,2,4,5}, {3,4,5,1,2}};  a0 = {20,23,19};
        b = {{3,2,2,4,1}, {1,1,2,4,1}}; b0 = {12,14};
        s = {{3,3,1,2,3}, {2,4,4,1,5}}; s0 = {221,342};
        e = {23,33,56,47,45};

In[2]:=N[OptimalniProgram[c, a, a0, b, b0, s, s0, e],6]

Out[1]= {0, 3.45714, 0, 0.714286, 2.22857}
```

3.2 Optimizacija utroška materijala

Optimalni utrošak materijala je takav utrošak materijala koji obezbeđuje ostvarenje određene proizvodnje uz najmanji ukupni otpadak. Kod formiranja odgovarajućeg matematičkog modela polazi se od sledećih pretpostavki: preduzeće bi trebalo da proizvede m različitih delova u određenim količinama, za proizvodnju tih delova koristi se materijal istih dimenzija.

Broj varijanti za obradu materijala unapred je poznat, a isto tako poznata je količina otpadaka koja se javlja pri svakoj varijanti, kao i količina pojedinih delova koja se dobija obradom jedinice materijala po svakoj varijanti.

U matematičkom modelu se koriste sledeće oznake:

x_j - količina materijala (broj traka, tabli) koja će biti obrađena po j-toj varijanti, a koje se određuju kao nepoznate vrednosti;

a_j - količina otpadaka od jedinice datog materijala koji je obrađen po j-toj varijanti;

a_{ij} - količina delova i-te vrste koja se dobija od jedne jedinice materijala obrađenog po j-toj varijanti;

b_i - ukupna količina i-tog dela koju treba obezbediti za proizvodnju;

s - raspoloživa količina datog materijala.

Modelom će se odrediti količina materijala date dimenzije koja će biti obrađena po j-toj varijanti, ali tako da se željene količine delova proizvedu uz minimalni ukupni otpadak.

Funkcija kriterijuma modela

$$z_0 = \sum_{j=1}^{n} a_j x_j$$

označava ukupni otpadak pri obradi materijala po svim varijantama, pa je potrebno naći njenu minimalnu vrednost uz sledeći sistem ograničenja:

$$\begin{aligned} &\sum_{j=1}^{n} a_{ij} x_j = b_i \ , \ i = 1, 2, \ldots, m, \\ &\sum_{j=1}^{n} x_j \leq s, \\ &x_j \geq 0 \ , \ j = 1, 2, \ldots, n. \end{aligned}$$

Formirali smo model za slučaj kada preduzeće za proizvodnju koristi materijal istih dimenzija.

```
UtrosakMaterijala[a0_,a_,b_,s_]:=
   Module[{m=a,v=b,n=Length[a0],s0={},i},
      For[i=1,i<=n,i++,
          s0=AppendTo[s0,-1]
      ];
      m=Dodaj[m,{s0}];    v=Dodaj[v,{-s}];
      LinearProgramming[a0,m,v]
   ]
```

Primer 3.2.1 Neka su date sledeće ulazne veličine:

```
In[1]:=a0 = {3,7,9,6,5}; a = {{3,7,1,2,4}, {2,2,4,1,5}}; b = {35,40}; s = 200;
In[2]:=N[UtrosakMaterijala[a0, a, b, s],6]
Out[1]= {2.14286, 0, 0, 0, 7.14286}
```

Pretpostavićemo da preduzeće za proizvodnju koristi isti materijal u k raznih dimenzija. Svaka dimenzija materijala može se obraditi prema raznim varijantama.

Ako sa n_v označimo broj varijanti obrade v-tog materijala, onda će ukupan broj varijanti obrade materijala svih dimenzija biti jednak N, pri čemu je

$$N = \sum_{v=1}^{k} n_v.$$

Sa oznakama i parametrima, čije je značenje isto kao i u prethodnom modelu, formiramo sledeći model.

Potrebno je naći minimalnu vrednost funkcije kriterijuma

$$z_0 = \sum_{j=1}^{N} a_j x_j$$

uz zadovoljenje sledećeg sistema ograničenja:

$$\sum_{j=1}^{N} a_{ij} x_j = b_i \ , \ i = 1, 2, \ldots, m,$$
$$\sum_{j=1}^{n_v} x_j \leq s_v \ , \ v = 1, 2, \ldots, k,$$
$$x_j \geq 0 \ , \ j = 1, 2, \ldots, N.$$

```
UtrosakMaterijalaRazni[a0_,a_,b_,s_,n_]:=
   Module[{m=a,v=b,vn,k=Length[n],i,nula,j,l=Length[a0]},
       vn=Sum[n[[i]],{i,k}];   nula=Table[0,{k},{l}];
       For[i=1,i<=k,i++,
         For[j=1,j<=n[[i]],j++,nula[[i,j]]=-1]
       ];
       m=Dodaj[m,nula];     v=Dodaj[v,-s];
       LinearProgramming[a0,m,v]
   ]
```

3.3 Izbor sastava mešavine

Problem svodi na određivanje količina pojedinih sirovina koje će biti utrošene za proizvodnju gotovog proizvoda odgovarajućeg kvaliteta, ali tako da troškovi nabavke sirovina budu minimalni.

U prvom slučaju, u kome se od više sirovina prozvodi jedan proizvod, polazimo od sledćih pretpostavki: za dobijanje gotovog proizvoda može se koristiti n vrsta sirovina; gotov proizvod mora da sadrži m raznih elemenata u određenim količinama; treba proizvesti ukupno b jedinica gotovog proizvoda; poznate su nabavne cene pojedinih sirovina.

Matematički model ćemo formirati korišćenjem sledećih simbola:

x_j - količina j-te sirovine koja će se utrošiti za proizvodnju b jedinica gotovog proizvoda;

p_j - nabavna cena jedinice j-te sirovine;

a_{ij} - količina i-tog elementa u jedinici j-te sirovine;

a_{i0} - propisana (minimalna ili maksimalna) količina i-tog elementa u gotovom proizvodu;

b - količina gotovog proizvoda koju treba proizvesti;

s_j - dozvoljena količina j-te sirovine u gotovom proizvodu.

Model se sastoji od funkcije kriterijuma

$$\min \quad z_0 = \sum_{j=1}^{n} p_j x_j$$

i ograničavajućih faktora:

$$\sum_{j=1}^{n} a_{ij} x_j = a_{i0} \ , \ i = 1, 2, \ldots, m,$$
$$\sum_{j=1}^{n} x_j = b, \quad 0 \leq x_j \leq s_j \ , \ j = 1, 2, \ldots, n.$$

```
Mesavina[p_,a_,a0_,b_,s_]:=
  Module[{m=a,v=a0,n=Length[p],pom,i},
     pom={};
     For[i=1,i<=n,i++,pom=AppendTo[pom,1]];
     m=Dodaj[m,{pom}];     m=Dodaj[m,-IdentityMatrix[n]];
     v=Dodaj[v,{b}];    v=Dodaj[v,-s];
     LinearProgramming[p,m,v]
  ]

  MesavinaProcenti[p_,a_,a0_,s_]:=Mesavina[p,a,a0,1,s]
```

3.4 Primena u poljoprivredi

Pretpostavimo da poljoprivredno dobro raspolaže sa h hektara obradive površine, da na njoj može zasejati n vrsta poljoprivrednih kultura, da raspolaže sa m vrsta poljoprivrednih mašina i da se za proizvodnju koristi g vrsta semena, zaštitnih sredstava i đubriva. Pored toga, poznati su prosečni prinosi svih poljoprivrednih kultura po jednom hektaru, te prodajna cena po jedinici svake kulture i direktni troškovi obrade jednog hektara pod određenom kulturom.

Potrebno je da se odredi na kojoj površini je potrebno zasejati svaku od poljoprivrednih kultura da bi poljoprivredno dobro ostvarilo maksimalan čist prihod.

Matematički model ćemo formirati korišćenjem sledećih simbola:

x_j - broj hektara na kojima će biti zasejana j-ta poljoprivredna kultura;

q_j - prosečni prinos j-te kulture po jednom hektaru;

p_j - prodajna cena po jedinici j-te kulture;

t_j - direktni troškovi obrade jednog hektara pod j-tom kulturom;

a_{ij} - vreme potrebno i-toj poljoprivrednoj mašini za obradu jednog hektara pod j-tom kulturom;

a_{i0}^k - raspoloživo vreme rada i-te poljoprivredne mašine u k-toj sezoni;

b_j^k - broj radnika koje treba angažovati u k-toj sezoni po jednom hektaru pod j-tom kulturom;

b_0^k - broj radnika sa kojima raspolaže poljoprivredno dobro za k-tu sezonu;

s_{vj} - količina v-tog materijala (semena, zaštitnog sredstva, veštačkog đubriva) potrebna po jednom hektaru pod j-tom kulturom;

s_{v0} - raspoloživa količina v-tog materijala;

h - raspoloživa obradiva površina u hektarima;

$\underline{h}_j$ - minimalna površina pod j-tom kulturom;

$\overline{h}_j$ - maksimalna površina pod j-tom kulturom.

Model se sastoji od funkcije kriterijuma

$$z_0 = \sum_{j=1}^{n} (q_j p_j - t_j) x_j,$$

koja označava ukupan čist prihod poljoprivrednog dobra, pa je potrebno naći njenu maksimalnu vrednost uz sledeće ograničavajuće faktore:

$$\sum_{j=1}^{n} a_{ij} x_j \le a_{i0}^k \ , \ i = 1, 2, \ldots, m, \ k = 1, 2, \ldots, r,$$

$$\sum_{j=1}^{n} b_j^k x_j \le b_0^k \ , \ k = 1, 2, \ldots, r,$$

$$\sum_{j=1}^{n} s_{vj} x_j \le s_{v0} \ , \ v = 1, 2, \ldots, g,$$

$$\sum_{j=1}^{n} x_j = h,$$

$$\underline{h}_j \le x_j \le \overline{h}_j \ , \ j = 1, 2, \ldots, n.$$

```
Poljoprivreda[q_,p_,t_,a_,a0_,b_,b0_,s_,s0_,h_,hmin_,hmax_]:=
   Module[{r=Dimensions[a0][[1]],m={},v={},i,qpt,pom,n=Length[q],vn},
       vn=Dimensions[a0];
       For[i=1,i<=vn[[1]],i++,
           For[j=1,j<=vn[[2]],j++,v=AppendTo[v,-a0[[i,j]]]]];
       qpt=Table[q[[i]] p[[i]]-t[[i]],{i,n}];
       For[i=1,i<=r,i++,m=Dodaj[m,-a]];
       m=Dodaj[m,-b];     m=Dodaj[m,-s];
       pom={};
       For[i=1,i<=n,i++,pom=AppendTo[pom,1]];
       m=Dodaj[m,{pom}]; m=Dodaj[m,IdentityMatrix[n]]; m=Dodaj[m,-IdentityMatrix[n]];
```

```
    v=Dodaj[v,-b0];     v=Dodaj[v,-s0];  v=AppendTo[v,h];
    v=Dodaj[v,hmin];    v=Dodaj[v,-hmax];
    LinearProgramming[-qpt,m,v]
]
```

3.5 Primena u ishrani

Problemi ishrane daju široke mogućnosti primene linearnog programiranja. Pored rešavanja pitanja dijetalne ishrane, koja je predstavljala područje prve praktične primene linearnog programiranja, postoji niz drugih interesantnih pitanja, kao što su: minimizacija troškova ishrane, izbor optimalnih sastava ishrane, itd.

Razni prehrambeni proizvodi sadrže određene sastojke i vitamine u različitim odnosima. Minimalne potrebe ovih ili onih sastojaka u ishrani su poznate. Poznavajući raspoloživu količinu prehrambenih proizvoda i cenu po jedinici mere svakog od njih, može se postaviti pitanje: kako se mogu zadovoljiti potrebe ishrane, a da pri tome nastali troškovi budu minimalni?

Matematička formulacija ovog zadatka sastoji se u sledećem. Poznato je n različitih prehrambcnih proizvoda $P_1, Pz, \ldots, P_j, \ldots, P_n$. Označimo sa x_j nepoznate količine koje predstavljaju dnevne potrebe j-tog prehrambcnog proizvoda. U svakom prehrambenom proizvodu postoje određene količine prehrambenih sastojaka: belančevina, šećera, ugljenih hidrata, kalcijuma, gvožđa, vitamina i drugih, za život potrebnih komponenata koje ćemo označiti sa $K_1, \ldots, K_i, \ldots, K_m$. Neka veličina a_{ij} karakteriše sadržaj j-tog prehrambenog sastojka u i-tom prehrambenom proizvodu ($i = 1, \ldots, m;\ j = l, \ldots, n$). Tada je ukupna količina i-tog prehrambenog sastojka u ishrani jednaka

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n.$$

Ako se minimalna svakodnevna potreba organizma u i-tom prehrambenom sastojku označi sa b_i, može se formirati sistem ograničenja

$$\sum_{j=1}^{n} a_{ij}x_j \geq b_i,\ (i = 1, \ldots, m). \tag{3.5.1}$$

S druge strane, dnevna upotreba svakog prehrambenog prpizvoda može biti ograničena raspoloživim rezervama i mogućnostima njihove popune. Zbog toga postoji dopunski uslovi za nepoznate x_j, koji se mogu napisati u obliku

$$0 \leq x_j \leq a_j,\ (j = l, \ldots, n), \tag{3.5.2}$$

gde su a_j veličine koje se određuju na bazi raspoložive količine prehrambenog proizvoda P_j u određenom vremenskom periodu. Ako se sa c_j označi cena jedinice mere j-tog prehrambenog proizvoda, cena celokupne ishrane može se prikazati funkcijom cilja

$$F(X) = \sum_{j=1}^{n} c_j x_j. \tag{3.5.3}$$

Prema tome, optimalna ishrana biće takav sastav i svakodnevne upotrebe određenih prehrambenih proizvoda $X = (x_1, \ldots, x_n)$, koji daje minimum funkcije cilja (3.5.3), pri skupu ograničenja koja su zadata izrazima (3.5.1) i (3.5.2).

Nekada uslovi (3.5.1), u odnosu na neke prehrambene proizvode, mogu biti zadani sa dvostrukim ograničenjima, tako da umesto izraza (3.5.1) postoje ograničenja oblika

$$b_i^* \geq \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \ (i = 1, \ldots, m).$$

Radi potpunije ilustracije navedimo jedan numerički primer.

Primer 3.5.1 Potrebno je izračunati minimalne troškove ishrane sa tri proizvoda P_1, P_2 i P_3, koji sadrže određene procente pet komponenti K_1, K_2, K_3, K_4, K_5. Brojni podaci dati su u sledećoj tabeli. Sastaviti odgovarajući matematički model.

	K_1	K_2	K_3	K_4	K_5	Cena
P_1	0.05	0.10	0.00	0.15	0.20	6
P_2	0.13	0.18	0.10	0.00	0.25	4
P_3	0.10	0.00	0.15	0.07	0.00	9
Propisani minimun	0.05	0.10	0.20	0.14	0.17	

Tabela 3.6.1.

Iz opšte formulacije matematičkog modela ishrane proizilazi da su nepoznate x_1, x_2, x_3, a da predstavljaju broj kilograma proizvoda P_1, P_2 i P_3, koje treba uzimati u dnevnoj ishrani da bi ona udovoljila propisane uslove i da bi troškovi ishrane bili minimalni.

Prema tome, za dati brojni primer funkcija cilja ima oblik

$$F(x_1, x_2, x_3) = 6x1 + 4x2 + 9x3,$$

dok su ograničenja zadana sistemom nejednačina

$$\begin{aligned} 0.05x_1 + 0.13x_2 + 0.10x_3 &\geq 0.05, \\ 0.10x_1 + 0.18x_2 &\geq 0.10, \\ 0.10x_2 + 0.15x_3 &\geq 0.20, \\ 0.15x_1 + 0.07x_3 &\geq 0.14, \\ 0.20x_1 + 0.25x_2 &\geq 0.17. \end{aligned}$$

Zadatak se sastoji u nalaženju minimuma funkcije cilja pri zadanim ograničenjima.

Ne postoje teškoće da se analogija sa ovog malog primera prenese na opšti slučaj i konstruiše odgovarajuća tablica polaznih podataka oblika kakav ima prethodna tablica. Na taj način postiže se preglednost svih potrebnih polaznih podataka.

3.6 Transport proizvodnje

Ovde će se objasniti matematički model organizacije transporta poljoprivredne proizvodnje, pod pretpostavkom da postoji m - punktova proizvodnje (ekonomije) $A_1, \ldots, A_m$ i n - punktova potrošnje $B_1, \ldots, B_n$.

Neka poljoprivredno gazdinstvo raspolaže sa g_k prevoznih sredstava k-iog tipa ($k = 1, \ldots, s$). Tipovi transportnih sredstava razlikuju se po nosivosti, brzini

kretanja, eksploatacionim karakteristikama i troškovima prevoza po jedinici mere tereta.

U cilju formulacije odgovarajućeg matematičkog modela uvedimo sledeće oznake:

a_i - količina proizvoda izražena jedinicom mere koja se proizvede u punktu A_i $(i = 1, \ldots, m)$,

b_j - količina proizvoda izražena istom jedinicom mere koja je potrebna punktu B_j $(j = 1, \ldots, n)$,

d_k - količina tereta koja se može prevesti prevoznim sredstvom k-tog tipa $(k = 1, \ldots, s)$,

c_{ik} - troškovi koji nastaju usled dolaska praznog prevoznog sredstva k-tog tipa iz mesta njegovog stacioniranja u mesto proizvodnje A_i radi utovara,

c^*_{jk} - troškovi koji nastaju usled vraćanja praznog prevoznog sredstva k-tog tipa iz punkta B_j u mesto njegovog stalnog stacioniranja,

c^{**}_{ijk} - troškovi koji nastaju usled prevoza proizvoda iz punkta A_i u punkt B_j na jednoj mašini k-tog tipa,

x_{ijk} - broj prevoznih sredstava k-tog tipa upućenih u punkt A_i za prevoz tereta u punkt B_j.

Pretpostavimo da usled dužine transporta nije moguća upotreba nijednog prevoznog sredstva više od jedanput (što znači da svako prevozno sredstvo ostvaruje celu turu: mesto stacioniranja - mesto proizvodnje - mesto potrošnje).

Koristeći se uvedenim oznakama, formulisani zadatak svodi se na nalaženje promenljivih x_{ijk} posredstvom traženja minimuma funkcije cilja

$$F(X) = \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{s}(c_{ik} + c^*_{jk} + c^{**}_{ijk})x_{ijk},$$

pri uslovima ograničenja oblika.

$$\begin{aligned}
\sum_{j=1}^{n}\sum_{k=1}^{s} d_k x_{ijk} &\leq a_i,\ (i = 1, \ldots, m)\\
\sum_{i=1}^{m}\sum_{k=1}^{s} d_k x_{ijk} &= b_j,\ (j = 1, \ldots, n)\\
\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ijk} &\leq g_k,\ (k = 1, \ldots, s)\\
x_{ijk} &\geq 0 \quad (i = 1, \ldots, m; j = 1, \ldots, n; k = 1, \ldots, s).
\end{aligned}$$

Prvi skup ograničenja dat je na osnovu raspoloživih količina proizvodnje, drugi na osnovu potreba u punktovima potrošnje, a treći na osnovu ograničenja transportnih sredstava k-tog tipa $(k = 1, \ldots, s)$.

3.7 Primena u planiranju proizvodnje

Osnovni zadatak planiranja optimalne proizvodnje primenom linearnog programiranja sastoji se u pronalaženju takvih količina raznih artikala, koje jedno ili više

udruženih preduzeća mogu da proizvedu uz najpovoljnije korišćenje raspoloživih ili novih resursa (radne snage, mašina za rad, sirovina i materijala), pod uslovom da je obezbeđen plasman na tržištu celog asortimana proizvodnje. Ovde se razmatra više zadataka iz oblasti planiranja i programiranja proizvodnje koji se svode na linearno programiranje.

Izbor asortimana za slučaj ograničenja jedne kategorije resursa

Najpre ćemo razmotriti jedan uprošćen matematički model optimizacije proizvodnje, koji se sastoji u optimalnom korišćenju raspoloživog kapaciteta mašina. Pretpostavimo da se raspolaže sa n mašina $M_1, \ldots, M_j, \ldots, M_n$ na kojima bi trebalo proizvoditi m artikala $A_1, \ldots, A_i, \ldots, A_m$. Neka se svaki i-ti artikal obrađuje na svakoj j-toj mašini za vreme a_{ij}. Poznat je kapacitet svake mašine M_j izražen u određenim jedinicama vremena sa c_j $(j = 1, \ldots, n)$, kao i ukupno vreme obrade bilo kog i-tog artikla na svim mašinama, koje će se označiti sa

$$b_i = \sum_{j=1}^{n} a_{ij}, \ (i = 1, \ldots, m).$$

Odgovarajući matematički model koji obezbeđuje maksimizaciju proizvodnje, za koju je obezbeđen plasman na tržištu, sastoji se u nalaženju vrednosti

$$X = (x_1, \ldots, x_m),$$

tj. količina asortimana, za koje će funkcija cilja

$$F(X) = \sum_{i=1}^{m} b_i x_i, \tag{3.7.1}$$

dobiti maksimalnu vrednost, a da pri tome budu zadovoljena ograničenja

$$\begin{aligned} \sum_{i=1}^{m} a_{ij} x_i &\leq c_j, \ (j = 1, \ldots, n), \\ x_i &\geq 0, \ (i = 1, \ldots, m). \end{aligned} \tag{3.7.2}$$

Sličan zadatak, kao što je zadatak formulisan matematičkim modelom (3.7.1) - (3.7.2), može se formulisati sa ciljem obezbeđenja maksimalnog dohotka. Ako u tom smislu uvedemo dodatne oznake d_i $(i = 1, \ldots, m)$, koje imaju značenje dobiti po komadu i-tog artikla, zadržavajući ista značenja svih napred uvedenih oznaka, zadatak maksimizacije dobiti sastoji se u nalaženju količina x_i $(i = l, \ldots, m)$ pojedinh artikala za koje će funkcija cilja

$$F(X) = \sum_{i=1}^{m} d_i x_i,$$

dobiti maksimalnu vrednost pri istim ograničenjima oblika (3.7.2).

Izbor asortimana za slučaj ograničenja više kategorija resursa

U oba izložena zadatka optimizacije tretirana su ograničenja jedne kategorije resursa (ograničen kapacitet mašina za rad). Međutim, zadatak optimizacije uslova proizvodnje postaje nešto složeniji, ako se uzme u razmatranje ograničenje više različitih kategorija resursa.

Pretpostavimo da je u proizvodnju uključeno n mašina $M_1, \ldots, M_j, \ldots, M_n$, na kojima radi s kategorija radnika $R_1, \ldots, R_k, \ldots, R_s$ i koristi se r vrsta sirovina i materijala $S_1, \ldots, S_j, \ldots, S_r$. Neka se može proizvoditi m artikala $A_1, \ldots, A_i, ..., A_m$.

U cilju formulacije odgovarajućeg matematičkog modela uvedimo sledeće oznake:

x_i - količina i-tog artikla (proizvoda) koju treba odrediti pomoću matematičkog modela optimizacije ($i = 1, \ldots, m$),

c_i - dohodak po jedinici i-tog artikla, a_{ij} - vreme potrebno za obradu i-tog artikla na j-toj mašini ($j = 1, \ldots, n$), a_j - kapacitet j-te mašine izražen u vremenskim jedinica

b_{ki} - vreme potrebno za obradu i-tog artikla od strane radnika k-te kategorije specijalnosti ($k = 1, \ldots, s$),

b_k - raspoloživi fond vremena radnika k-te kategorije specijalnosti,

d_{iv} - količine v-te sirovine (materijala) koja je potrebna za proizvodnju i-tog artikla ($v = 1, \ldots, r$),

d_{iv} - raspoloživa količina v-te vrste sirovine (materijala),

e_i - količine i-tog artikla koja se može prodati na tržištu.

Koristeći se uvedenim oznakama, može se formulisati matematički model u kome se maksimizira funkcija cilja

$$F(X) = \sum_{i=1}^{m} c_i x_i$$

koja predstavlja ukupnu dobit od celokupne proizvodnje, tj. traže se one vrednosti $x_1, \ldots, x_m$ koje zadovoljavaju skup ograničenja

$$\begin{aligned}
\sum_{i=1}^{m} a_{ij} &\leq a_i, \ (j = 1, \ldots, n), \\
\sum_{i=1}^{m} b_{ki} x_i &\leq b_k, \ (k = 1, \ldots, s), \\
\sum_{i=1}^{m} d_{iv} x_i &\leq d_v, \ (v = 1, \ldots, r), \\
0 &\leq x_i \leq e_i, \ (i = 1, \ldots, m),
\end{aligned}$$

a da pri tome definisana funkcija cilja dobije maksimalnu vrednost. Slično kao u prethodnom slučaju i ovde se može formulisati funkcija cilja koja ima drugo značenje (maksimizacija proizvodnje, maksimizacija produktivnosti, tj. odnosa između vrednosti ukupne proizvodnje i angažovane radne snage za ostvarivanje te proizvodnje, itd.), a da pri tome ograničenja ostanu ista, kao što su napred formulisana.

3.8 Upravljanje zalihama

Ovde će se prikazati jedan matematički model upravljanja zalihama koji se svodi na primenu linearnog programiranja. Cilj koji se postavlja sastoji se u planiranju proizvodnje u zavisnosti od potreba tržišta, kako bi troškovi skladištenja gotove robe bili rainimalni.

Pretpostavimo da se proizvodi n artikala na m različitih mašina u s jednakih vremenskih intervala ($i = 1, \ldots, m; j = 1, \ldots, , n; k = 1, \ldots, s$). U cilju formulisanja odgovarajućeg modela, pomoću koga se vrši minimizacija troškova skladištenja, uvedimo sledeće oznake:

a_{ij} - vreme trajanja operacije na i-toj mašini za j-ti artikal, ($i = 1, \ldots, m; j = 1, \ldots, n$),

b_{ik} - kapacitet i-te mašine u k-tom vremenskom periodu ($k = 1, \ldots, s$),

c_{jk} - troškovi skladištenja jedinice j-tog artikla, koji su jednaki za bilo koji k-ti vremenski interval,

x_{jk} - obim proizvodnje j-tog artikla u k-tom vremenskom intervalu,

p_{jk} - tražnja j-tog artikla u k-tom vremenskom intervalu,

$X_{jk} = \sum_{v=1}^{k} x_{jv}$ - kumulativ proizvodnje j-tog artikla za prvih k vremenskih intervala,

$P_{jk} = \sum_{v=1}^{k} p_{jv}$ - kumulativ tražnje j-tog proizvoda za prvih k vremenskih intervala.

Prema tome, zadatak se sastoji u određivanju minimuma funkcije cilja

$$F = \sum_{j=1}^{n} \sum_{k=1}^{s} c_{jk}(X_{jk} - P_{jk}),$$

koja predstavlja troškove skladištenja za sve j-te artikle u svim vremenskim intervalima, pod uslovima da kumulativ proizvodnje nije manji od kumulativa tražnje, tj.

$$X_{jk} \geq P_{jk},$$

kao i da su zadovoljena ograničenja kapaciteta proizvodnje

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_{jk} &\leq b_{ik}, \ (i = 1, \ldots, m; k = 1, \ldots, s), \\ x_{jk} &\geq 0, \ (i = 1, \ldots, m; k = 1, \ldots, s). \end{aligned}$$

Koristeći se napred uvedenim oznakama za X_{jk} P_{jk}, formulisani matematički model može se prikazati u obliku funkcije cilja

$$F = \sum_{j=1}^{n} \sum_{k=1}^{s} c_{jk} \left[\sum_{v=1}^{k} (x_{jv} - p_{jv}) \right],$$

koju je potrebno minimizirati pri ograničenjima oblika

$$\begin{aligned}\sum_{v=1}^{k}(x_{jv}-p_{jv}) &\geq 0,\ (j=1,\ldots,n;k=1,\ldots,s),\\ \sum_{j=1}^{n}a_{ij}x_{jk} &\leq b_{ik},\ (i=1,\ldots,m;k=1,\ldots,s),\\ x_{jk} &\geq 0,\ (j=1,\ldots,n;k=1,\ldots,s).\end{aligned}$$

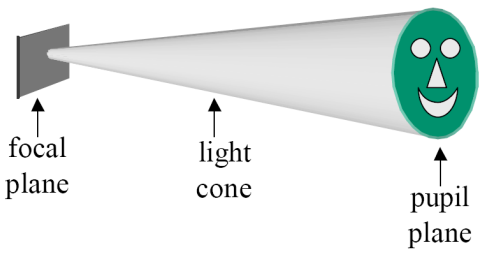
Na bazi rezultata koji se mogu dobiti na osnovu izloženog modela, može se vršiti upravljanje zalihama i proizvodnjom.

4 Primene u astronomiji i elektronici

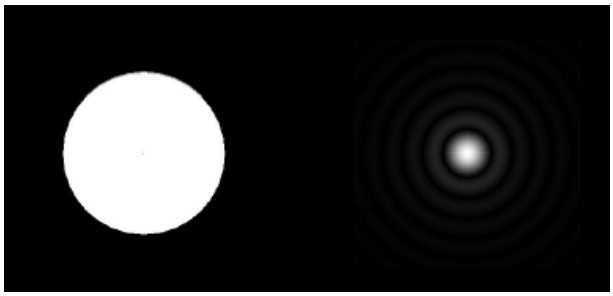
U ovoj glavi ćemo pokazati dve praktične primene prethodno izloženih metoda matematičkog programiranja (linearnog programiranja i višekriterijumske optimizacije), u astronomiji i u telekomunikacijama (elektronici). Linearno programiranje i višekriterijumska optimizacija, kao i ostale mnogobrojne metode matematičkog programiranja imaju primene u skoro svim oblastima nauke, tehnike i uopšte u svakodnevnom životu. Sledeća dva primera ilustruju tu činjenicu.

4.1 Izračunavanje optimalne maske teleskopa

U ovom odeljku razmotrićemo primenu linearnog programiranja na problem projektovanja teleskopa pomoću koga ćemo moći da utvrdimo da li oko neke zvezde kruže planete. Na slici 4.1.1 prikazan je uprošćeni model teleskopa.



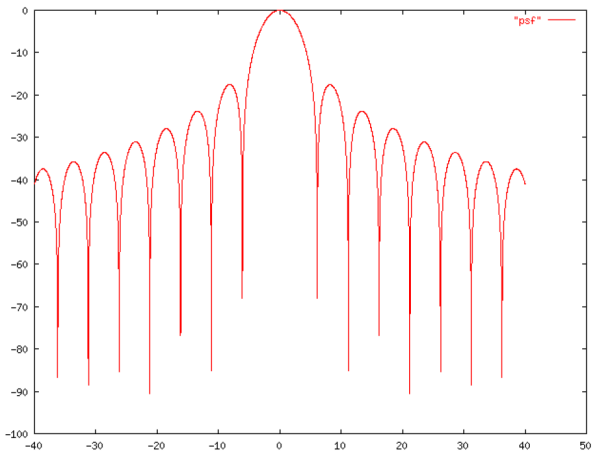
Slika 4.1.1. Principijelna šema teleskopa.



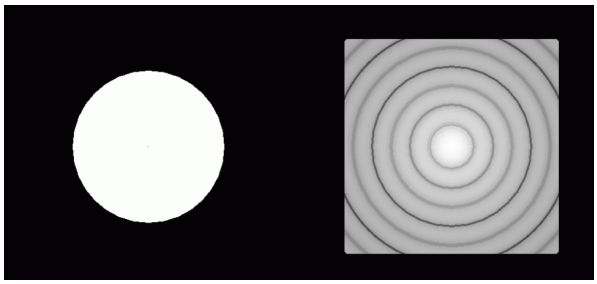
Slika 4.1.2. Profil objektiva i difrakciona slika.

Svetlost sa udaljene zvezde pada na ravan objektiva teleskopa (eng. *pupil plane*), prelama se kroz sočivo objektiva i pada na žižnu ravan (eng. *focal plane*). Svi svetlosni zraci koji pod istim uglom padnu na objektiv, posle prelamanja padaju u istu tačku na žižnoj ravni. Svetlosni zraci koji dolaze sa zvezde su skoro paralelni, pa bi prema tome lik zvezde trebao biti jedna tačka u žižnoj ravni (ako podesimo teleskop tako da se zvezda nalazi na glavnoj optičkoj osi objektiva, onda bi to bila tačka sa koordinatama $(0,0)$). Međutim, usled talasne prirode svetlosti, dolazi do difrakcije na otvoru objektiva i umesto jedne svetle tačke, vidimo mrlju konačne veličine koju okružuju prstenovi (slika 4.1.2).

Ovi prstenovi, iako su tamniji od centralne mrlje, ipak su intenzivniji od svetlosti bilo koje planete koja kruži oko zvezde. Na primer, za posmatrača sa strane, svetlost koja potče od Sunca je 10^{10} puta intenzivnija od svetlosti Zemlje. Postavljanjem maske na objektiv, menjamo jačinu i oblik prstenova na difrakcionoj slici. Treba naći takav oblik maske tako da intenzitet svetlosti u okolini svetle mrlje bude dovoljno mali. Na slici 4.1.3 je prikazan intenzitet svetlosti na koordinatnoj x osi žižne ravni. Intenzitet svetlosti je u decibelima $I[\text{dB}] = 10 \log \frac{I}{I_{ref}}$, pri čemu je za referentni intenzitet I_{ref} uzeta vrednost u koordinatnom početku. Na slici 4.1.4 je difrakciona slika, pri čemu je sada referentni nivo -110dB.



Slika 4.1.3. Intenzitet svetlosti na x koordinatnoj osi.



Slika 4.1.4. Profil objektiva i difrakciona slika, pri čemu je nula na −110dB.

Pretpostavimo da je profil maske iskazan funkcijom $A(x)$, tj da je otvoreni deo objektiva:

$$\left\{(x,y) \mid -\frac{1}{2} \le x \le \frac{1}{2},\ -A(x) \le y \le A(x)\right\} \tag{4.1.1}$$

Naravno, prethodno je izvršena odgovarajuća normalizacija dužina. Intenzitet svetlosti je direktno proporcionalan kvadratu jačine električnog polja E. Sa (x,y) označavaćemo koordinate tačke u ravni objektiva a sa (ξ,ζ) u žižnoj ravni. Ako usvojimo Fraunhoferovu aproksimaciju [3], imamo sledeći izraz za jačinu električnog polja u tački (ξ,ζ) u žižnoj ravni:

$$E(\xi,\zeta) = \int_{-1/2}^{1/2}\int_{-A(x)}^{A(x)} e^{i(x\xi+y\zeta)}dydx = 4\int_0^{1/2}\cos(x\xi)\frac{\sin(A(x)\zeta)}{\zeta}dx$$

Poslednja jednakost važi zbog simetrije oblika maske (4.1.1). Uslov koji ćemo sada nametnuti je da u prethodno zadatoj oblasti $\mathcal{O}$ važi sledeći uslov:

$$-\alpha E(0,0) \le E(\xi,\zeta) \le \alpha E(0,0), \quad (\xi,\zeta) \in \mathcal{O} \tag{4.1.2}$$

Pretpostavimo sada da su tačke skupa $\mathcal{O}$ na x osi (tj. da je $\zeta = 0$). Tada odnos $\frac{\sin(A(x)\zeta)}{\zeta}$ postaje samo $A(x)$ odnosno električno polje je jednako:

$$E(\xi,0) = 4\int_0^{1/2}\cos(x\xi)A(x)dx$$

Tada uslov (4.1.2) postaje:

$$\int_0^{1/2} A(x)\left[\cos(x\xi) - \alpha\right]dx \le 0$$
$$\int_0^{1/2} A(x)\left[\cos(x\xi) + \alpha\right]dx \le 0$$

Cilj nam je da vidljiva površina objektiva bude što je moguće veća, odnosno da maksimizujemo

$$\int_{-1/2}^{1/2} 2A(x)dx = 4\int_0^{1/2} A(x)dx$$

Znači, posmatramo sledeći optimizacioni problem:

$$\begin{aligned} &\max\ 4\int_0^{1/2} A(x)dx \\ &p.o. \int_0^{1/2} A(x)\left[\cos(x\xi)-\alpha\right]dx \le 0 \\ &\qquad \int_0^{1/2} A(x)\left[\cos(x\xi)+\alpha\right]dx \le 0 \\ &\qquad 0 \le A(x) \le \frac{1}{2}. \end{aligned} \tag{4.1.3}$$

Ako sada izvršimo odgovarajuću diskretizaciju integrala i skupa $\mathcal{O}$ dobijamo sledeći problem linearnog programiranja [4, 8]:

$$\begin{aligned} &\max\ 4\sum_{i=1}^{N_x} B_i A(x_i) \\ &p.o. \sum_{i=1}^{N_x} C_i A(x_i)\left[\cos(x_i\xi_j)-\alpha\right] \le 0, \quad j=1,\ldots,N_\xi \\ &\qquad \sum_{i=1}^{N_x} D_i A(x_i)\left[\cos(x_i\xi_j)-\alpha\right] \le 0 \quad j=1,\ldots,N_\xi \\ &\qquad 0 \le A(x_i) \le \frac{1}{2}. \end{aligned} \tag{4.1.4}$$

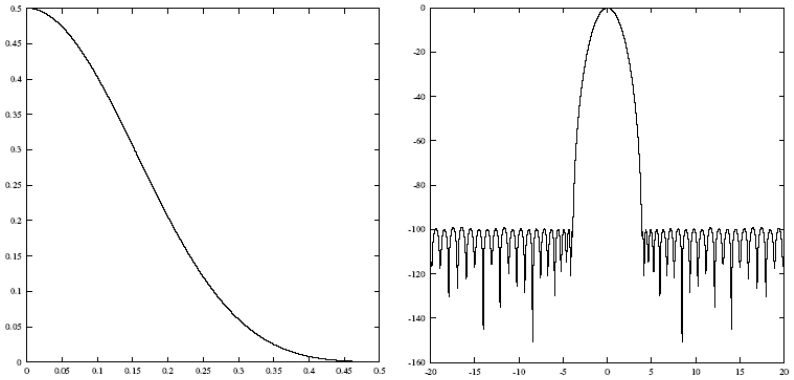
Ovde smo sa N_x i N_ξ označili broj diskretizacionih nivoa redom za promenljive x i ξ. ξ_i su vrednosti diskretizovane promenljive ξ. Ako je skup $\mathcal{O}$ zadat npr. kao interval

$$\mathcal{O} = \{(\xi, 0) \mid \xi_0 < \xi < \xi_1\}$$

onda možemo uzeti ekvidistantnu podelu $\xi_i = a + (i-1)\frac{b-a}{N_\xi - 1}$. Vrednosti x_i, B_i, C_i i D_i su koeficijenti odgovarajućih kvadraturnih formula kojima se aproksimiraju integrali u funkciji cilja i uslovima. U najprostijem slučaju možemo uzeti $B_i = C_i = D_i = \frac{1}{N_x}$ i $x_i = -\frac{1}{2} + \frac{i-1}{N_x - 1}$.

Problem (4.1.4) predstavlja problem linearnog programiranja u simetričnom obliku. Ulogu promenljivih ovde imaju vrednosti funkcije $A(x)$ u tačkama x_i ($A(x_i)$).

Ako je sada $\alpha = 10^{-5}$, $\xi_0 = 4$, $\xi_1 = 40$, $N_x = N_\xi = 1000$, i diskretizacija je uniformna, dobijamo rešenje sa slike 4.1.5 za funkciju $A(x)$ [4].



Slika 4.1.5. Profil optimalne maske $A(x)$ i intenzitet svetlosti na x osi ako se koristi optimalna maska.

Sa slike vidimo, da je intenzitet svetlosti za $\xi > 4$ manji ili jednak -110dB, tako da je u ovom pojasu moguće zapaziti lik neke planete koja kruži oko zvezde.

Slična analiza se može obaviti ukoliko skup $\mathcal{O}$ ima nešto drugačiji oblik, ili se formira više od jedne maske [4].

4.2 Projektovanje FIR filtara

U ovom odeljku razmotrićemo primenu linearnog programiranja u projektovanju digitalnih FIR filtara. FIR filtri, kao i uopšte digitalni filtri imaju veliku ulogu u obradi *digitalnih signala* [2]. Digitalni signali su za razliku od kontinualnih signala definisani samo u diskretnim vremenskim trenucima i predstavljaju se kao niz realnih vrednosti $x(t)$, $t = 1, 2, \ldots$. U analizi digitalnih signala najčešće se koriste diskretna Fouriereova transformacija i z-transformacija. One su date pomoću sledećeg izraza:

$$X(f) = \sum_{n=-\infty}^{+\infty} x(t)e^{2\pi f i n} \qquad X(z) = \sum_{n=-\infty}^{+\infty} x(n)z^{-n} \tag{4.2.1}$$

Fouriereova transformacija $X(f)$ se naziva i *spektar* signala $x(t)$. Napomenimo da je $X(f)$ periodična funkcija sa periodom 1. Nadalje ćemo $X(f)$ posmatrati samo na intervalu $(-\frac{1}{2}, \frac{1}{2})$. Digitalni filtri, kao i analogni, modifikuju ulazni signal tako što propuštaju na izlaz samo one komponente signala čija je frekvencija u tačno određenom opsegu. Frekvencijska karakteristika idealnog filtra može se prikazati pomoću izraza [2]:

$$|H(f)| = \left|\frac{Y(f)}{X(f)}\right| = \begin{cases} 1, & f \in \mathcal{O} \\ 0, & \text{U suprotnom} \end{cases}$$

gde su $Y(f)$, $X(f)$ redom spektri izlaznog i ulaznog signala a $\mathcal{O}$ skup frekvencija koje se propuštaju. Idealni filtar je nemoguće u praksi realizovati, pa se zato pristupa traženju realnog filtra (koji se može realizovati u praksi) koji što bolje aproksimira idealni. Neka je $h(t)$ inverzna diskretna Fouriereova transformacija funkcije $H(f)$, data pomoću:

$$h(t) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(f) e^{2\pi i f t} df.$$

Imamo da su signal na izlazu i signal na ulazu povezani sledećom relacijom:

$$y(t) = \sum_{s=-\infty}^{+\infty} h(s) x(t-s)$$

koji predstavlja *konvoluciju* nizova $x(t)$ i $h(t)$.

Za FIR (*Finite Impulse Response*) filtre važi da postoji broj n_0 takav da je $h(t) = 0$ ako je $|t| > n_0$. Broj n_0 nazivamo *red* filtra. U suprotnom, u pitanju je IIR filtar (*Infinite Impulse Response*). Uvešćemo još jednu pretpostavku u našu analizu, da je $h(t)$ simetrična funkcija po t. Ovaj uslov obezbeđuje linearnost faze kompleksne funkcije $H(f)$. Sada je $|H(f)|$, takođe, simetrična funkcija i važi $|H(f)| = |A(f)|$ gde je $A(f)$ definisana pomoću [2]:

$$A(f) = h(0) + 2 \sum_{t=1}^{n_0} h(t) \cos(2\pi f t). \tag{4.2.2}$$

Nadalje ćemo posmatrati samo vrednosti $H(f)$ (odnosno $A(f)$) za $f > 0$. Neka je $\mathcal{O} = [0, f_0]$ (propusnik niskih frekvencija). Znači, potrebno je minimizirati:

$$\begin{aligned} &\min \int_0^{f_0} |A(f) - 1| df \\ &\min \int_{f_0}^{1/2} |A(f)| df, \end{aligned} \tag{4.2.3}$$

Ovim smo dobili problem bezuslovne višekriterijumske nelinearne optimizacije. Promenljive su u ovom slučaju vrednosti $h(t)$, $t = 0, \ldots, n_0$. Uvedimo sada pomoćne funkcije $\tau(f)$ i $\eta(f)$. Problem (4.2.3) možemo ekvivalentno predstaviti u obliku:

$$\begin{aligned} &\min \int_0^{f_0} \tau(f) df \\ &\min \int_{f_0}^{1/2} \eta(f) df \\ &p.o. \ -\tau(f) \le A(f) - 1 \le \tau(f), \quad f \in \mathcal{O} \\ &\quad -\eta(f) \le A(f) \le \eta(f), \quad f \notin \mathcal{O}. \end{aligned} \tag{4.2.4}$$

Ako sada izvršimo diskretizaciju po frekvenciji, slično kao u prethodnom odeljku, dobijamo problem linearne višekriterijumske optimizacije. U praksi se najčešće

fiksira jedno odstupanje (npr. na skupu $\mathcal{O}^C$) i traži se minimum drugog. Sada problem (4.2.3) svodimo na sledeći problem:

$$\begin{aligned} \min \ & \int_0^{f_0} \tau(f)df \\ p.o. \ & -\tau(f) \le A(f) - 1 \le \tau(f), \quad f \in \mathcal{O} \\ & -\epsilon \le A(f) \le \epsilon, \quad f \notin \mathcal{O} \\ & \tau(f) \ge 0 \end{aligned} \tag{4.2.5}$$

a ako uvedemo diskretizaciju po f direktno dobijamo problem linearnog programiranja u simetričnom obliku:

$$\begin{aligned} \min \ & \sum_{j=1}^{N'_f} B_j \tau(f_j) \\ \text{p.o.} \ & -\tau(f_j) \le h(0) + 2\sum_{t=1}^{n_0} h(t)\cos(2\pi f_j t) - 1 \le \tau(f_j), \quad j = 1, \ldots, N'_f \\ & -\epsilon \le h(0) + 2\sum_{t=1}^{n_0} h(t)\cos(2\pi f_j t) \le \epsilon, \quad j = N'_f + 1, \ldots, N_f + N'_f \end{aligned} \tag{4.2.6}$$

Primetimo da je ovo svođenje isto kao kod metode ϵ ograničenja iz prethodne glave. Kao i u prethodnom odeljku, i ovde su f_j i B_j, $j = 1, \ldots, N'_f$ čvorovi i koeficijenti odgovarajuće kvadraturne formule. Za $j > N'_f$ vrednosti f_j su iz skupa $\mathcal{O}^C$. Promenljive su nam sada $\tau(f_j)$, $j = 1, \ldots, N'_f$ i $h(t)$, $t = 0, \ldots, n_0$.

U radu [1], razmatran je problem reprodukcije signala pomoću tri FIR filtra, pri čemu svaki propušta različiti opseg frekvencija. Imamo da je spektar paralelne veze ovih filtara:

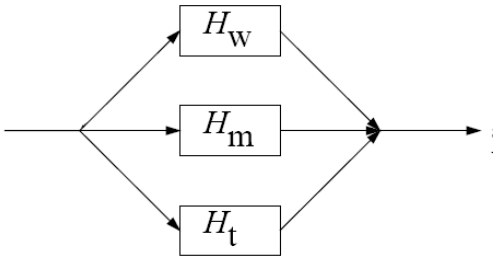
$$H(f) = H_w(f) + H_m(f) + H_t(f)$$

gde su $H_w(f)$, $H_m(f)$ i $H_t(f)$ gde su redom spektri svakog filtra (indeksi potiču od engleskih reči w-*woofer*, m-*midrange*, t-*tweetier*). Propusni opsezi ($\mathcal{O}$) za ove filtre su redom $\mathcal{O}_w = [0, f_w]$,$\mathcal{O}_m = [0, f_{m1}] \cup [f_{m2}, \frac{1}{2}]$ i $\mathcal{O}_t = [f_t, \frac{1}{2}]$. U ovom slučaju nam je cilj da funkcija $A(f)$ bude što bliža jedinici za $f \in [0, \frac{1}{2}]$. Prema tome, imamo sledeći problem:

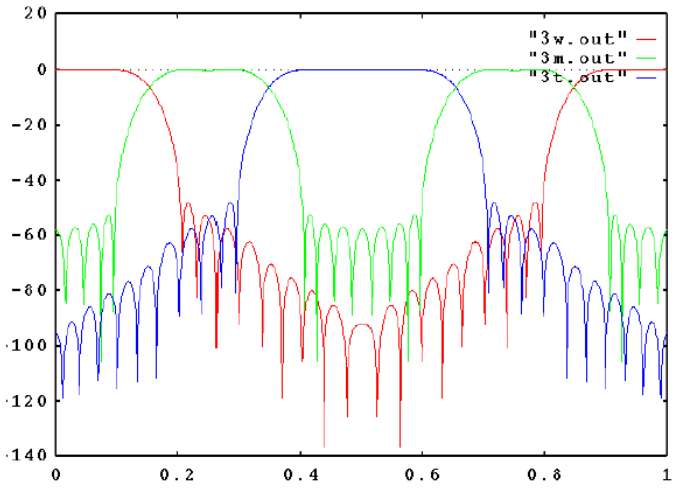
$$\begin{aligned} \min \ & \int_0^{1/2} |A_w(f) + A_m(f) + A_t(f) - 1| \\ p.o. \ & -\epsilon \le A_i(f) \le \epsilon, \quad f \in \mathcal{O}_i, \quad i = w, m, t \end{aligned} \tag{4.2.7}$$

koji na sličan način svodimo na problem linearnog programiranja.

Na slici 4.2.2 su prikazane funkcije $A_w(f)$, $A_m(f)$ i $A_t(f)$ za sledeće vrednosti parametara [1]: $N_f = 1000$, $f_w = 0.2$, $f_t = 0.7$, $f_{m1} = 0.1$ i $f_{m2} = 0.4$.



Slika 4.2.1. Blok šema paralelne veze tri filtra.



Slika 4.2.2. Grafici funkcija $A_w(f)$, $A_m(f)$ i $A_t(f)$.

5 Linearna regresija

U mnogim naukama veoma često postoji potreba određivanja direktne funkcionalne zavisnosti između dve ili više veličina. Oblast matematičke statistike koja se bavi utvrđivanjem i opisivanjem ovih zavisnosti naziva se *regresija*. Ukoliko se utvrđuje linearna zavisnost posmatranih obeležja, reč je o *linearnoj regresiji*. Postoje dva moguća pristupa ovom problemu.

Kod prvog pristupa se posmatra uticaj obeležja (slučajnih veličina) $X_1, \ldots, X_n$ na takođe slučajnu veličinu Y. Pri tom, treba odrediti funkciju $f(x_1, \ldots, x_n)$ takvu da slučajna veličina $f(X_1, \ldots, X_n)$ najbolje aproksimira Y. Kao mera odstupanja ovih dvaju veličina, najčešće se koristi srednjekvadratno odstupanje, tj uslov da je $E(Y - f(X))$ minimalno. Ovim tipom regresije nećemo se baviti.

U drugom pristupu, posmatra se uticaj određenog broja neslučajnih veličina, *kontrolnih faktora* na vrednost odgovarajuće slučajne veličine Y. Ova veza je de-

deterministička (neslučajna). Na veličinu Y takođe utiču i slučajni faktori, kao i drugi neslučajni faktori čiji se uticaj ne može efektivno sagledati. Pretpostavićemo da su ova dva uticaja međusobno nezavisna, aditivna (vrednost promenljive Y je zbir vrednosti determinističke i slučajne komponente) i da je očekivanje slučajne komponente jednako nuli.

U nastavku ćemo detaljnije opisati problem linearne regresije druge vrste i dati rešenja u slučaju L_2, L_1 i L_∞ norme. Pokazaćemo da se slučajevi L_1 i L_∞ norme svode na problem linearnog programiranja.

5.1 Formulacija problema linearne regresije druge vrste

Ukoliko je veza kontrolnih faktora i promenljive Y linearna, radi se o *linearnom modelu regresije druge vrste*. Označimo sa $a_1, \ldots, a_n$ vrednosti kontrolnih faktora, $x_1, \ldots, x_n$ vrednosti koeficijenata linearne veze, a sa ϵ slučajnu komponentu. Imamo:

$$Y = a_1 x_1 + \ldots, a_n x_n + \epsilon. \tag{5.1.1}$$

Potrebno je izvršiti odgovarajuću procenu koeficijenata $x_1, \ldots, x_n$ tako da se relacija (5.1.1) "najbolje" slaže sa dobijenim eksperimentalnim rezultatima. Pri tome se vrši niz eksperimenata sa različitim vrednostima kontrolnih faktora $a^k = (a_{k1}, \ldots, a_{kn})$ $k = 1, \ldots, m$ gde je m ukupan broj eksperimenata. Označimo sa Y_i i ϵ_i, $i = 1, \ldots, m$ dobijene vrednosti veličine Y i slučajne veličine ϵ u i-tom eksperimentu (ponavljanju). Dobija se:

$$Y_i = a_{i1} x_1 + \ldots, a_{in} x_n + \epsilon_i, \quad i = 1, \ldots, m. \tag{5.1.2}$$

Obeležimo sada $\mathbf{Y} = [Y_1 \ \cdots \ Y_m]^T$, $A = [a_{ij}]$, $x = [x_1 \ \cdots \ x_n]$ kao i $\epsilon = [\epsilon_1 \ \cdots \ \epsilon_m]$. Prethodni sistem možemo zapisati u matričnoj formi na sledeći način:

$$\mathbf{Y} = \mathbf{A}\mathbf{x} + \epsilon \tag{5.1.3}$$

Potrebno je, za zadate vrednosti matrice A i vektora $\mathbf{Y}$ naći vrednost vektora x takvu da je norma vektora ϵ minimalna.

$$\min \ \|\epsilon\| = \|\mathbf{Y} - \mathbf{A}\mathbf{x}\|. \tag{5.1.4}$$

Napomena 5.1.1 U izrazu (5.1.1) kontrolni faktori ne moraju biti funkcionalno nezavisni. Takođe, moguće je posmatrati opštiji model:

$$Y = x_1 f_1(a_1, \ldots, a_p) + \ldots + x_n f_n(a_1, \ldots, a_p) + \epsilon$$

gde su sada $a_1, \ldots, a_p$ kontrolni faktori (zavisni ili nezavisni), a $f_1, \ldots, f_n$ funkcije kojima se opisuje zavisnost veličine Y od faktora $a_1, \ldots, a_p$. U ovom slučaju, matricu A određujemo kao $A = [f_j(a_{i1}, \ldots, a_{ip})]$, $i = 1, \ldots, m$, $j = 1, \ldots, n$, a dalji postupak je potpuno isti.

U opštem slučaju, možemo posmatrati bilo koju normu u prethodnom izrazu. Međutim, najčešće su u upotrebi L_1 norma (suma apsolutnih vrednosti komponenata), L_2 (suma kvadrata) kao i L_∞ (maksimalna apsolutna vrednost komponente).

U zavisnosti od toga koja se funkcija norme koristi razlikujemo L_1 *linearnu regresiju*, L_2 *linearnu regresiju* kao i L_∞ *linearnu regresiju*. Detaljno ćemo proučiti ove tri vrste regresije.

Primer 5.1.1 Predpostavimo da se slučajna veličina Y sastoji iz dva dela: konstantne, ali nepoznate vrednosti x i slučajnog odstupanja od te fiksirane vrednosti ϵ. Imamo da važi:

$$Y = x + \epsilon \tag{5.1.5}$$

Ovaj model je sličan problemu regresije pri čemu ovde nemamo kontrolne faktore. Vrednost x je jednaka očekivanju slučajne promenljive Y ($E\epsilon = 0$), pa se prema tome problem svodi na određivanje nepoznatog očekivanja promenljive Y na osnovu vrednosti eksperimenata. Ukoliko posmatramo L_1, L_2 odnosno L_∞ normu, rešenja problema su data sledećim izrazom:

$$\begin{aligned} L_1: &\quad x^*_{L_1} = Y_{(m+1)/2} \\ L_2: &\quad x^*_{L_2} = \overline{Y} = \frac{Y_1 + \ldots + Y_m}{m} \\ L_\infty: &\quad x^*_{L_\infty} = \frac{Y_{(m)} + Y_{(1)}}{2}. \end{aligned} \tag{5.1.6}$$

Pri tome smo sa $Y_{(m+1)/2}$ označili *medijanu* realizovanog uzorka $(Y_1, \ldots, Y_m)$ a sa $Y_{(k)}$ odgovarajuću statistiku poredka (k-ti element po veličini realizovanog uzorka). Sve ove vrednosti na neki način predstavljaju "srednju vrednost" broja ostvarenih poena na ispitu. Koje će se od ova tri rešenja usvojiti zavisi od konkretnog slučaja.

Neka se npr. našim eksperimentom određuje broj poena studenata na nekom ispitu i neka su dobijene sledeće vrednosti:

$$28, 62, 80, 84, 86, 86, 92, 95, 98$$

Imamo da je $x^*_{L_1} = 86$, $x^*_{L_2} = 79$ a $x^*_{L_\infty} = 63$. Vidimo da prema prvoj proceni student koji je osvojio 80 poena je loše uradio test, prema drugoj je prosečno uradio, a prema trećoj dobro.

5.2 L_2 regresija

Sigurno najrasprostanjenija varijanta linearne regresije u praksi je L_2 regresija. Kod ovog tipa minimizira se euklidska L_2 norma vektora ϵ, definisana na sledeći način:

$$\|\epsilon\|_2 = \left(\epsilon_1^2 + \ldots + \epsilon_m^2\right)^{1/2}$$

Ukoliko se umesto broja 2 u predhodnom izrazu stavi broj $p > 1$ dobijamo L_p normu. Kada p teži beskonačnosti, imamo da važi $\lim_{p\to+\infty} \|y\|_p = \|y\|_{+\infty}$. Problem koji posmatramo ima sledeći oblik:

$$\min \ \|\mathbf{Y} - A\mathbf{x}\|_2. \tag{5.2.1}$$

Ukoliko kvadriramo prethodni izraz, dobijamo ekvivalentan problem:

$$\min \ F(x) = \sum_{i=1}^{m} \left(Y_i - \sum_{j=1}^{n} a_{ij} x_j \right)^2. \tag{5.2.2}$$

Ovim smo problem sveli na bezuslovni problem kvadratnog programiranja[1]. Ovaj problem se jednostavno rešava izjednačavanjem parcijalnih izvoda funkcije $F(x)$ sa

[1] Kvadratnim programiranjem se nećemo baviti u ovoj knjizi

nulom. Na taj način dobijamo da optimalno rešenje x^* zadovoljava sledeći linearni sistem jednačina:

$$A^T A\mathbf{x} = A^T\mathbf{Y}. \tag{5.2.3}$$

Ukoliko je matrica $A^T A$ regularna, rešenje sistema (5.2.3) je jedinstveno i jednako:

$$x^* = (A^T A)^{-1} A^T\mathbf{Y}. \tag{5.2.4}$$

Ukoliko matrica $A^T A$ nije regularna, može se pokazati da sistem (5.2.3) (odnosno problem (5.1.4)) ima beskonačno mnogo rešenja i da su opisana na sledeći način:

$$x^* = A^\dagger\mathbf{Y} + (I_m - AA^\dagger)z. \tag{5.2.5}$$

gde je $A^\dagger$ *uopšteni inverz (pseudoinverz)* matrice A, a $z \in R^m$ proizvoljan vektor.

Primer 5.2.1 Najjednostavniji i najčešće primenljivan u praksi regresioni model je onaj u kome je Y linearna funkcija jednog kontrolnog faktora x, tj važi:

$$Y = ax + b + \epsilon$$

Pritom su zadate vrednosti $(x_1, Y_1), \ldots (x_m, Y_m)$. Očigledno, ovaj problem je ekvivalentan provlačenju (ili *fitovanju*) najpribližnije prave kroz zadate tačke (x_i, Y_i). Dobija se:

$$A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix}, \quad A^T A = \begin{bmatrix} \sum_{i=1}^m x_i^2 & \sum_{i=1}^m x_i \\ \sum_{i=1}^m x_i & m \end{bmatrix}, \quad A^T Y = \begin{bmatrix} \sum_{i=1}^m x_i Y_i \\ \sum_{i=1}^m x_i \end{bmatrix}. \tag{5.2.6}$$

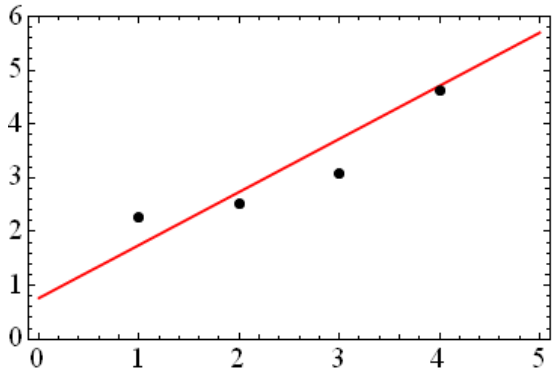
U konkretnom slučaju se primenom jednačine (5.2.4), i na osnovu prethodnih formula dobija rešenje traženog problema $x^* = [a, b]^T$. Na primer, za skup vrednosti:

x_i	1	2	3	4	5
Y_i	2.26	2.52	3.08	4.62	6.15

Imamo da važi:

$$A^T A = \begin{bmatrix} 55 & 15 \\ 15 & 5 \end{bmatrix}, \quad A^T Y = \begin{bmatrix} 65.76 \\ 18.63 \end{bmatrix},$$

pa sada lako dobijamo da je rešenje $x^* = \begin{bmatrix} a & b \end{bmatrix}^T = \begin{bmatrix} 0.98 & 0.77 \end{bmatrix}^T$. Date tačke, kao i fitovana prava prikazane su na slici 5.2.1.



Slika 5.2.1. Grafik funkcije $f(x) = ax + b$ zajedno sa tačkama.

5.3 L_1 i L_∞ regresija

Za razliku od L_2 regresije, u slučaju L_1 regresije ne postoji egzaktna formula, kao što je to (5.2.5) kojom bi se opisala rešenja ovog problema. U ovom slučaju minimizujemo L_1 normu vektora ostatka, odnosno tražimo vektor x^* koji zadovoljava:

$$x^* = \text{argmin}_{x\in\mathbb{R}^n} \|\mathbf{Y} - A\mathbf{x}\|_1 = \text{argmin}_{x\in\mathbb{R}^n} \sum_{i=1}^{m} \left| Y_i - \sum_{j=1}^{n} a_{ij}x_j \right|. \tag{5.3.1}$$

Iako ne postoji egzaktna formula za rešenje problema (5.3.1), on se relativno lako svodi na problem linearnog programiranja. Zaista, ukoliko označimo sa t_i sledeći izraz:

$$t_i = \left| Y_i - \sum_{j=1}^{n} a_{ij}x_j \right|$$

problem (5.3.1) se lako svodi na sledeći problem linearnog programiranja:

$$\begin{aligned} &\min\ t_1 + \ldots + t_m \\ &\text{p.o.}\ -t_i \le Y_i - \sum_{j=1}^{n} a_{ij}x_j \le t_i, \quad i = 1, \ldots, m. \end{aligned} \tag{5.3.2}$$

Rešavanjem problema linearnog programiranja (5.3.2) dobijamo rešenje x^* problema L_1 regresije.

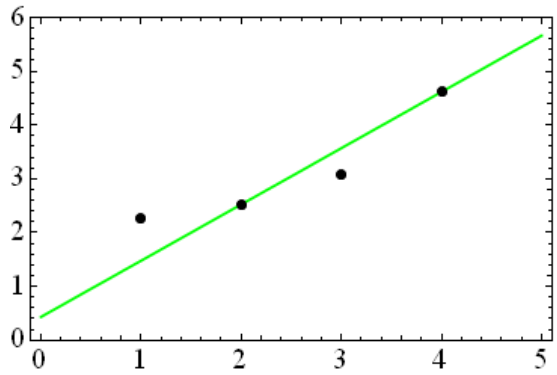
Primer 5.3.1 Posmatrajmo isti problem zadat u primeru 5.2.1. Odgovarajući ekvivalentan problem linearnog programiranja za ovaj problem je:

$$\begin{array}{llllllllll} \min & t_1 & +t_2 & +t_3 & +t_4 & +t_5 & & & & \\ \text{p.o.} & -t_1 & & & & & -a & -b & \le & -2.26 \\ & & -t_2 & & & & -2a & -b & \le & -2.52 \\ & & & -t_3 & & & -3a & -b & \le & -3.08 \\ & & & & -t_4 & & -4a & -b & \le & -4.62 \\ & & & & & -t_5 & -5a & -b & \le & -6.15 \\ & -t_1 & & & & & +a & +b & \le & 2.26 \\ & & -t_2 & & & & +2a & +b & \le & 2.52 \\ & & & -t_3 & & & +3a & +b & \le & 3.08 \\ & & & & -t_4 & & +4a & +b & \le & 4.62 \\ & & & & & -t_5 & +5a & +b & \le & 6.15. \end{array}$$

Rešenje ovog problema linearnog programiranja je:

$$[t_1, t_2, t_3, t_4, t_5, a, b]^T = \begin{bmatrix} 0.78 & 0 & 0.49 & 0 & 0.48 & 1.05 & 0.43 \end{bmatrix}^T$$

odakle dobijamo da su koeficijenti a i b redom jednaki 1.05 i 0.43. Kao i u primeru 5.2.1 i ovde prikazujemo na slici 5.3.1 grafički tačke i aproksimativnu pravu.



Slika 5.3.1. Grafik funkcije $f(x) = ax + b$ zajedno sa tačkama, L_1 regresija.

Na sličan način možemo transformisati i sledeći problem L_∞ regresije:

$$x^* = \text{argmin}_{x\in\mathbb{R}^m} \|\mathbf{Y} - A\mathbf{x}\|_\infty = \max_{1\le i\le m} \left| Y_i - \sum_{j=1}^{n} a_{ij}x_j \right| \tag{5.3.3}$$

na problem linearnog programiranja. U ovom slučaju, potrebno je da uvedemo samo još jednu dodatnu promenljivu t koja će biti upravo jednaka funkciji cilja:

$$\begin{aligned} &\min\ t \\ &\text{p.o.}\ -t \le Y_i - \sum_{j=1}^{n} a_{ij}x_j \le t, i = 1, \ldots, m. \end{aligned} \tag{5.3.4}$$

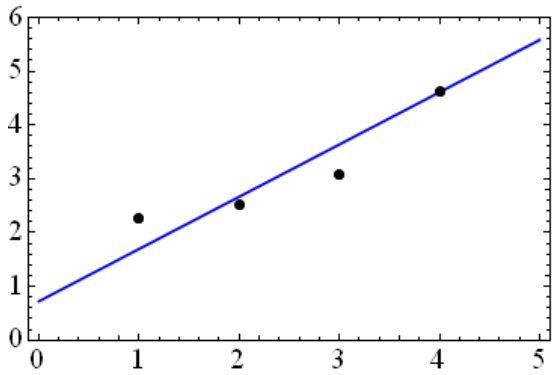
Primer 5.3.2 Sada ćemo rešiti problem iz primera 5.2.1, ali upotrebom L_∞ regresije. Problemu je ekvivalentan sledeći problem linearnog programiranja:

$$\begin{array}{lrrrcl} \min & t & & & & \\ \text{p.o.} & -t & -a & -b & \le & -2.26 \\ & -t & -2a & -b & \le & -2.52 \\ & -t & -3a & -b & \le & -3.08 \\ & -t & -4a & -b & \le & -4.62 \\ & -t & -5a & -b & \le & -6.15 \\ & -t & +a & +b & \le & 2.26 \\ & -t & +2a & +b & \le & 2.52 \\ & -t & +3a & +b & \le & 3.08 \\ & -t & +4a & +b & \le & 4.62 \\ & -t & +5a & +b & \le & 6.15 \end{array}$$

Rešenje ovog problema linearnog programiranja je:

$$x^* = \begin{bmatrix} t & a & b \end{bmatrix}^T = \begin{bmatrix} 0.56 & 0.97 & 0.72 \end{bmatrix}^T.$$

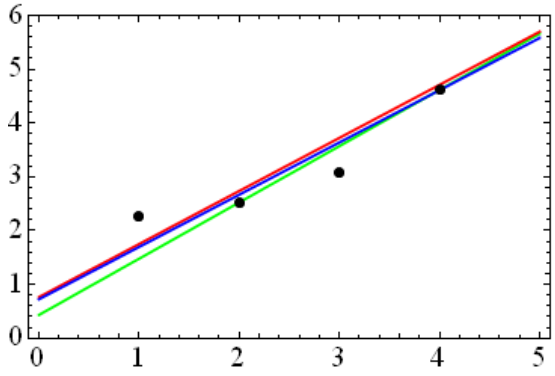
Grafik aproksimativne prave i datih tačaka prikazan je na slici 5.3.2.



Slika 5.3.2. Grafik funkcije $f(x) = ax + b$ zajedno sa tačkama, L_∞ regresija.

5.4 Poređenje L_1, L_2 i L_∞ modela regresije

Prezentovani modeli linearne regresije daju optimalna rešenja na osnovu različitih kriterijuma optimalnosti. Ova rešenja u opštem slučaju ne moraju biti jednaka. Izvršimo upoređivanje rezultata iz primera 5.2.1 dobijenih pomoću L_1, L_2 i L_∞ modela regresije. Na slici 5.4.1 su prikazane tačke zajedno sa aproksimativnim pravama za sva tri modela regresije. Uočavamo da su prave veoma bliske.



Slika 5.4.1. Sva tri modela regresije za problem iz primera 5.2.1.

Prema tome izbor modela regresije na konkretnom primeru nije od naročite važnosti. U tom slučaju je najbolje koristiti L_2 regresiju zato što se optimalno rešenje direktno dobija primenom formula (5.2.5) odnosno (5.2.6).

Međutim za neke specifične primere, L_2 model regresije često daje veoma loše rezultate koji često mogu da dovedu do potpuno pogrešnih zaključaka.

Primer 5.4.1 Posmatrajmo sledeći model, u kome je Y kvadratna funkcija [2] jednog kontrolnog faktora x:

$$Y = ax^2 + bx + c + \epsilon$$

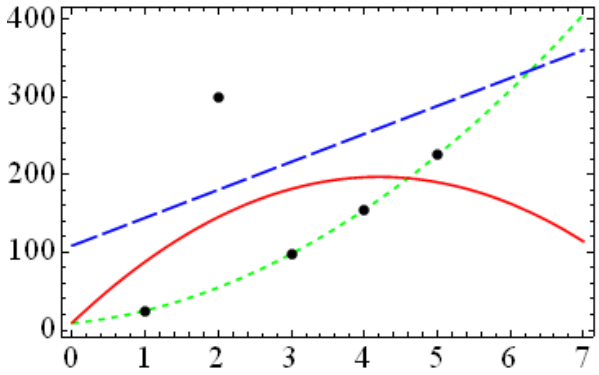
[2]Napominjemo da se ovde takođe radi o linearnom modelu regresije zato što je Y linearna funkcija parametara a, b i c modela.

Na osnovu sledećeg skupa vrednosti od $m = 5$ tačaka potrebno je odrediti parametre a, b i c tako da kvadratna funkcija $ax^2 + bx + c$ najbolje aproksimira vrednosti veličine Y:

x_i	1	2	3	4	5
Y_i	25.36	300	97.151	154.99	225.15

Primetimo da vrednosti promenljive Y rastu sa porastom vrednosti x, osim u slučaju $x = 2$ gde beležimo značajno odstupanje. Ovako velika odstupanja su često posledica sistematskih grešaka. Posmatrajmo sada kako ovo drastično odstupanje utiče na optimalne vrednosti parametara a, b i c kod svakog od modela regresije.

Postupak određivanja optimalnih vrednosti parametara prepuštamo čitaocu. Na slici 5.4.2 prikazane su date tačke i aproksimativne kvadratne funkcije za sva tri modela.



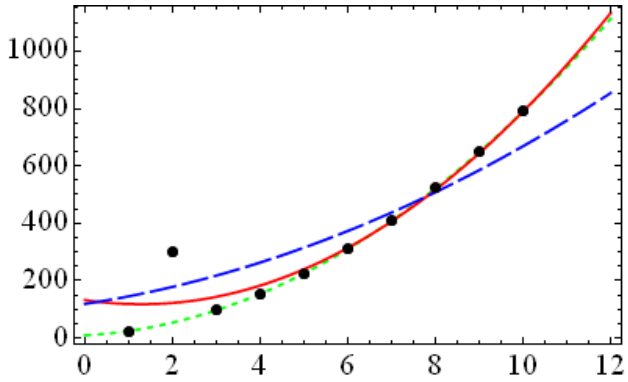
Slika 5.4.2. Sva tri modela regresije za problem kod koga jedna tačka drastično odstupa.

Primećujemo najpre da sve tačke sem druge leže praktično na jednoj paraboli koja je dobijena L_1 aproksimacijom. Prema tome, sistematska greška skoro da uopšte nije imala uticaj na optimalno rešenje kod L_1 modela pa možemo zaključiti da je na ovom primeru L_1 model dao najbolje rezultate. Nasuprot ovome, kod L_2 modela beležimo veliko odstupanje. Kao posledica sistematske greške primećujemo da je optimalna vrednost koeficijenta a u ovom slučaju negativna i da optimalna parabola ima maksimum za $x = 4.2$. Dakle može se potpuno pogrešno zaključiti da je zavisnost $Y(x)$ najpre rastuća a zatim opadajuća, pa zaključujemo da je L_2 regresija dala veoma loše rezultate na ovom primeru. Relativno loše rezultate beležimo i kod L_∞ modela. Primetimo da je optimalna parabola u ovom slučaju prava, tj da važi $a = 0$.

Primer 5.4.2 Dodajmo sada još nekoliko eksperimentalnih tačaka. Dobijamo sledeću tablicu:

x_i	1	2	3	4	5	6	7	8	9	10
Y_i	25.36	300	97.151	154.99	225.15	310.67	409.47	522.43	649.21	790.16

Ukoliko sada dodamo još nekoliko tačaka na paraboli (slika 5.4.3), dobijamo da se L_2 aproksimacija poboljšava i da se za veće vrednosti faktora x izjednačila sa L_1 aproksimacijom koja je i dalje najbolja. Primećujemo da je L_∞ aproksimacija bolja nego pre dodavanja novih vrednosti, ali je odstupanje dalje primetno.



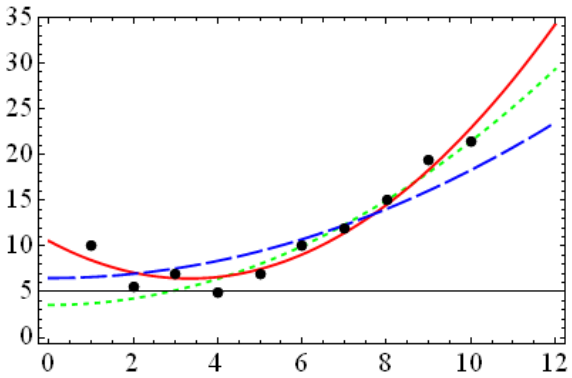
Slika 5.4.3. Poboljšana L_2 aproksimacija.

Na osnovu predhodna dva primera primećujemo da L_1 aproksimacija definitivno najbolje otklanja sistematske greške. Međutim to ponekad može biti loše, što lepo ilustruje naredni primer.

Primer 5.4.3 Posmatrajmo skup vrednosti:

x_i	1	2	3	4	5	6	7	8	9	10
Y_i	10	5.5	6.89	5.01	6.95	10.02	12.05	15	19.45	21.46

Primećujemo da Y najpre opada pa onda raste. Sa slike 5.4.4 možemo da zapazimo da parabola dobijena L_2 aproksimacijom najpre opada do vrednosti $x = 3.33$ a zatim raste, što je vrlo slično ponašanju eksperimentalnih vrednosti funkcije Y. Međutim L_1 aproksimacija ne prati početno opadanje zavisnosti, tj prvih nekoliko tačaka gde ova zavisnost opada nemaju dovoljan uticaj na optimalno rešenje L_1 modela. Slično se ponaša i L_∞ model na ovom primeru.



Slika 4.4.4. L_∞ model na istom primeru.

Na osnovu predhodna tri primera možemo zaključiti da se pri izboru modela regresije mora mnogo voditi računa o samoj prirodi eksperimentalnih rezultata na koje se model primenjuje i na osnovu toga izabrati odgovarajući model.

5.5 Implementacija u paketu MATHEMATICA

Metode L_1 i L_∞ implementirane su u programskom jeziku MATHEMATICA. Izložićemo najvažnije detalje implementacije. Funkcija `L1Reg` formira problem linearnog programiranja oblika (5.3.2), rešava ga i kao rezultat daje x^* komponentu optimalnog rešenja (t^*, x^*). Sledi implementacija ove funkcije:

```
L1Reg[A_, Y_] := Module[{i, m, n, A1, b, sol, c, a},
      {m, n} = Dimensions[A];
      c = Join[Table[1, {i, m}], Table[0, {i, n}]];
      A1 = Join[
          Table[Join[Table[KroneckerDelta[i, j], {j, m}], A[[i]]], {i, m}],
          Table[Join[Table[KroneckerDelta[i, j], {j, m}], -A[[i]]], {i, m}]];
      b = Join[Y, -Y];
      sol = LinearProgramming[c, A1, b];
      Return[Take[sol, -n]];
];
```

Funkcija `LInfReg` formira i rešava problem linearnog programiranja. Kao rezultat se takođe prosleđuje x komponenta optimalnog rešenja (t^*, x^*). Sledi implementacija ove funkcije:

```
LInfReg[A_, Y_] := Module[{i, m, n, A1, b, sol, c},
      {m, n} = Dimensions[A];
      c = Join[{1}, Table[0, {i, n}]];
      A1 = Join[
          Table[Join[{1}, A[[i]]], {i, m}],
          Table[Join[{1}, -A[[i]]], {i, m}]
      ];
      b = Join[Y, -Y];
      sol = LinearProgramming[c, A1, b];
      Return[Take[sol, -n]];
];
```

Literatura

[1] J.O. Coleman, *A Systematic Approach to the Constrained Quadratic Optimization of Embedded FIR Filters*, Conference on Information Sciences and Systems, Princeton, March 1998.

[2] Lj. Milić Z. Dobrosavljević, *Uvod u Digitalnu Obradu Signala*, Elektrotehnički fakultet, Univerzitet u Beogradu, 1999.

[3] D.M. Ivanović, V.M. Vučić, *Fizika II, Elektromagnetika i optika*, Naučna knjiga, Beograd, 1967.

[4] N.J. Kasdin, R.J. Vanderbei, D.N. Spergel, M.G. Littman. *Extrasolar Planet Finding via Optimal Apodized and Shaped Pupil Coronagraphs.* Astrophysical Journal, 582:11471161, 2003.

[5] J.J. Petrić, *Operaciona istraživanja, Knjiga prva*, Savremena administracija, Beograd, 1974.

[6] J.J. Petrić, *Operaciona istraživanja, Knjiga druga*, Savremena administracija, Beograd, 1983.

[7] O. Todorović, *Operaciona istraživanja*, Prosveta, Niš, 1999.

[8] R.J. Vanderbei, N.J. Kasdin, D.N. Spergel, *Rectangular-Mask Coronagraphs for High-Contrast Imaging*, arXiv:astro-ph/0401644 v1 30 Jan 2004.

[9] R.J. Vanderbei, *Linear Programming: Foundations and Extensions*, Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ 08544, 2001.

[10] S. Wolfram, *The Mathematica Book, 4th ed.*, Wolfram Media/Cambridge University Press, 1999.

Glava 4

Dinamičko Programiranje

1 Uvod

Opšte karakterisitke modela dinamičkog programiranja (skraćeno DP) su:

- Globalni model (problem) se rasčlanjuje na etape, i u okviru svake etape se vrši optimizacija ciljne funkcije (maksimizacija dobiti, minimizacija troškova i slično)
- Etapa (faza) se definiše kao uređen skup stanja.
- Prilikom izbora procesa upravljanja (ili dejstva), transformacija svakog stanja tekuće etape (faze) je povezana sa sledećom etapom. Ako se problem DP interpretira kao mrežni model, svaki čvor u mreži odgovara jednom stanju.
- Optimalno upravljanje počinje sa prvom ili poslednjom etapom, u zavisnosti od prirode modela.

Dinamičko programiranje je tehnika za rešavanje problema sa rekurzivnom strukturom koja poseduje sledeće karakteristike:

1. Optimalne podstrukture (Belmanov princip optimalnosti): neko optimalno rešenje na proizvoljnoj instanci sadrži optimalno rešenje svih svojih podinstanci.
2. Mali broj podproblema: ukupan broj podproblema koji će se rešavati je mali.
3. Preklapanje podproblema: u toku rekurzije pojavljuju se isti podproblemi više puta.

Princip "zavadi pa vladaj" (divide-and-conquer) je korišćen u mnogim algoritmima: da bi se rešio veliki problem, on se razbija na manje probleme koji se mogu rešiti nezavisno. U dinamičkom programiranju je ovaj princip usavršen do ekstrema:

kada se ne zna precizno koji se manji problemi rešavaju, jednostavno se reše svi, zatim se rezultati zapamte da bi se upotrebili kasnije u rešavanju većih problema [18, 19]. Postoje dve pincipijelne poteškoće u primeni ove tehnike. Prvo, nije uvek moguće kombinovanje rešenja dva problema da bi se formiralo rešenje jednog većeg. Drugo, može da postoji neprihvatljivo veliki broj malih problema koji se rešavaju.

Nije precizno definisano koji problemi mogu biti efektivno rešeni pomoću dinamičkog programiranja; postoji veliki broj "teških" (hard) problema za koje ovaj metod nije primenljiv [19], kao i mnogo "lakih" problema za koje je ovaj metod manje efikasan u odnosu na standardne algoritme [19]. Međutim, postoji tačno definisana klasa problema za koje je dinamičko programiranje sasvin efektivno [19]. Ovi problemi imaju opšte svojstvo da svaka odluka uključena u nalaženje najboljeg rešenja ostaje dobra odluka čak i kada je podproblem uključen kao deo većeg problema. Neki od takvih problema su opisani u ovoj glavi.

Dinamičko programiranje omogućava optimalno planiranje višeetapnih procesa upravljanja. Mnogi zadaci upravljanja procesima u tehnici, ekonomiji, vojsci, fizici, biologiji, itd., mogu se predstaviti u vidu višeetapnih procesa, na koje se može primeniti metod dinamičkog programiranja, a sa ciljem da se dobije optimalni plan upravljanja. Pri tome se pod optimalnim upravljanjem podrazumeva ono upravljanje koje daje najbolje rešenje.

Za praktičnu primenu dinamičkog programiranja potrebno je da svaki razmatrani proces ima svoj jasno postavljeni matematički model, sa precizno definisanim ciljem (funkcija cilja), koji treba maksimizirati (minimizirati) i ograničenjima koja se moraju uzeti u obzir u toku procesa. Za ovako postavljen zadatak, ukoliko su zadovoljeni uslovi koje zahteva primena metoda DP, potrebno je naći funkcionalne relacije primenom principa optimalnosti. U izvesnim slučajevima moguće je naći analitičko rešenje, ali rešenje se najčešće nalazi numeričkim procesima.

Dinamičko programiranje se obično primenjuje u problemima optimizacije: problem može imati mnogo rešenja, svako rešenje ima vrednost, a traži se rešenje koje ima optimalnu (najveću ili najmanju) vrednost. U slučaju da postoji više rešenja koja imaju optimalnu vrednost, obično se traži bilo koje od njih.

Sama reč "programiranje" ovde se (kao i u linearnom programiranju) odnosi na popunjavanje tabele pri rešavanju problema, a ne na upotrebu kompjutera i programskih jezika. Tehnike optimizacije koje imaju elemente dinamičkog programiranja su bile poznate i ranije, ali tvorcem metoda danas se smatra profesor Ričard E. Belman. Belman je proučavao dinamičko programiranje i dao čvrstu matematičku osnovu za ovaj način rešavanja problema. Uopšteno govoreći, problem se rešava tako što se uoči hijerarhija problema istog tipa, sadržanih u glavnom problemu, i rešavanje se počne od najjednostavnijih problema. Vrednosti i delovi rešenja svih rešenih podproblema se pamte u tabeli, pa se dalje njihovim kombinovanjem dobija rešenja većih podproblema sve do rešenja glavnog problema.

Napomenimo da se u literaturi koja obrađuje dinamičko programiranje [6, 19, 20, 25, 13] sreću dva različita pristupa ovom problemu. Prvi pristup je uglavnom vezan za knjige u kojima se proučavaju problemi operacionih istraživanja i matematičkog progamiranja, dok se drugi pristup prvenstveno javlja u knjigama koje su posvećene

teoriji algoritama. Između ova dva pristupa nema suštinskih razlika i oba pristupa tretiraju potpuno isti problem. U nastavku ove glave izložićemo veći broj problema koji se rešavaju tehnikom dinamičkog programiranja, pri čemu ćemo za neke probleme koristiti prvi a za neke drugi pristup. Konkretno, problemi 2.1, 2.2, 2.3, 5.1, 5.2 i 5.3 formulisani su korišćenjem prvog pristupa dok je za formulaciju ostalih problema korišćen drugi pristup.

Koristićemo sledeće konvencije i oznake. Elemente niza A označavaćemo sa $a_1, \ldots, a_N$, ili sa $A(1), \ldots, A(N)$. U slučaju dvostrukih indeksa, radi bolje čitljivosti nećemo pisati a_{b_c} nego $a(b_c)$, ili $a(b(c))$. Iste primedbe važe i za matrice.

2 Klasični problemi dinamičkog programiranja

U ovom odeljku izložićemo nekoliko klasičnih primena tehnike DP. Svaki pododeljak predstavlja jedan od klasičnih problema koji se ovom tehnikom uspešno rešavaju. Za svaki problem data je najpre formulacija a zatim i detaljno opisano rešenje. Svi navedeni problemi pripadaju problemima optimizacije i efikasno se svode na probleme linearnog, nelinearnog i celobrojnog programiranja a za probleme 2.1, 2.2 i 2.3 date su i ekvivalentne formulacije. Zatim je izloženo rešenje primenom tehnike DP i uradjen jedan ili više praktičnih primera gde su izložena rešenja primenjena na konkretne probleme.

2.1 Prosta raspodela jednorodnog resursa

Predpostavimo da imamo odredjenu količinu nekog resursa i da je potrebno rasporediti svu raspoloživu količinu ovog resursa na n potrošača. Potrošači mogu biti različiti proizvodni procesi. Poznata je efikasnost svakog potrošača koja predstavlja zaradu koju potrošač ostvaruje. Naravno, efikasnost nekog potrošača je funkcija količine resursa dodeljene tom potrošaču. Napomenimo da efikasnost ne mora da predstavlja zaradu, već to može biti neki drugi parametar (potrošnja materijala, vreme, i slično) koji želimo optimizovati. Za svaki potrošač je poznata maksimalna količina resursa koja mu se može dodeliti (kapacitet potrošača). Prema tome, poznate su sledeće veličine:

S - ukupna raspoloživa količina resursa.

$g_j(x)$ - funkcija dobiti koja se ostvaruje kada se količina x raspoloživog resursa dodeli j-tom procesu (mašini, liniji).

Q_j - kapacitet potrošača j.

Potrebno je pronaći količine resursa x_j, $j = 1, \ldots, n$ koje treba dodeliti svakom potrošaču tako da se maksimizira ukupna dobit koju potrošači zajedno ostvaruju.

Rešenje: Označimo sa $F(x)$ ukupnu dobit koju ostvaruju mašine ako se j-toj mašini dodeli količina resursa x_j, gde je $x = (x_1, \ldots, x_n)$. Prema tome, potrebno je naći

$$\max F(x) = \sum_{j=1}^{n} g_j(x_j). \tag{2.1.1}$$

Sledeća dva uslova se dobijaju iz činjenica da je S ukupna raspoloživa količina resursa i da je kapacitet j-tog potrošača jednak Q_j:

$$\begin{aligned} &\sum_{i=1}^{n} x_i \leq S, \\ &0 \leq x_j \leq Q_j,\ j = 1, \ldots, n. \end{aligned} \tag{2.1.2}$$

Ovim smo konstruisali jedan optimizacioni problem kod koga je funkcija cilja u opštem slučaju nelinearna. Ukoliko su g_j linearne funkcije, dobija se problem linearnog programiranja koji je moguće rešavati primenom nekog od metoda izloženih u prvoj i drugoj glavi. Pritom moramo da predpostavimo da su vrednosti x_j po prirodi realni brojevi (tj da se resursi mere količinski, npr. struja, voda, gorivo, itd...). Ukoliko su x_j celi brojevi, problem (2.1.1-2.1.2) je problem celobrojnog linearnog odnosno nelinearnog programiranja. Rešavanje problema linearnog odnosno nelinearnog, a pogotovu celobrojnog programiranja je u opštem slučaju složen problem (potrebni su složeni algoritmi koji dugo rade).

Medjutim, ovaj problem se mnogo efikasnije rešava primenom tehnike DP. Označimo sa $f_i(s)$ maksimalnu dobit koju je moguće ostvariti sa ukupnom količinom resursa s i prvih i potrošača. Potrebno je naći $f_n(S)$. Posmatraćemo složeniji problem kod koga je potrebno naći sve vrednosti $f_i(s)$ za svako $0 \leq s \leq S$ i $1 \leq i \leq n$.

Princip optimalnosti se interpretira na sledeći način: ako sa x_n označimo količinu koja je dodeljena n-tom potrošaču, tada za dalje raspoređivanje preostaje $s - x_n$ jedinica koje bi trebalo **optimalno raspodeliti** na preostale potrošače.

Prema tome, maksimalna dobit $f_n(s)$ koja se ostvaruje raspodelom s jedinica resursa na n procesa se može iskazati sledećim izrazom:

$$f_n(s) = \max_{0 \leq x_n \leq \min\{s, Q_n\}} \{g_n(x_n) + f_{n-1}(s - x_n)\}. \tag{2.1.3}$$

Predhodnu formulu možemo dalje primeniti na izračunavanje vrednosti funkcije $f_{n-1}(s)$ za $0 \leq s \leq S$. U graničnom slučaju, maksimalna dobit na osnovu dodele x_1 jedinica resursa potrošaču 1 jednaka je:

$$f_1(s) = \max_{0 \leq x_1 \leq \min\{s, Q_1\}} \{g_1(x_1)\}, \quad \text{uz uslov} \quad f_1(0) = 0. \tag{2.1.4}$$

Izloženo rešenje je po prirodi rekurzivno i jednostavno za implementaciju ali pati od eksponencijalne složenosti (vreme izvršenja je eksponencijalna funkcija argumenta n). Ovaj problem će biti detaljnije razmotren u narednom odeljku prilikom razmatranja problema ranca. Za sad, dovoljno je da konstatujemo da ukoliko vrednosti funkcija f_i izračunavamo u redosledu $i = 1, \ldots, n$ i pritom pamtimo predhodno izračunate vrednosti vremenska složenost postaje linearna po n.

Prema tome, najpre se izračunavaju vrednosti funkcije $f_1(s)$ na osnovu graničnog uslova (2.1.4), a zatim korišćenjem jednačine (2.1.3) se redom izračunavaju vrednosti funkcija $f_2(s), f_3(s), \ldots, f_n(s)$ pri čemu se predhodno izračunate vrednosti

pamte. Maksimalna dobit u polaznom problemu je jednaka $f_n(S)$. Medjutim, sada je potrebno odrediti i samo optimalno rešenje $x^* = (x_1^*, \ldots, x_n^*)$.

Optimalno rešenje rekonstruišemo u obrnutom redosledu. Na osnovu relacije (2.1.3) imamo da je x_n^* zapravo vrednost promenljive x_n za koju je izraz na desnoj strani maksimalan. Pošto smo vrednosti funkcije f_{n-1} već izračunali, možemo da odredimo x_n^*. Nastavljajući ovaj postupak dalje, dobijamo i ostale vrednosti x_j^* u sledećem redosledu $x_n^*, \ldots, x_2^*, x_1^*$:

$$\begin{aligned} f_n(S) &= \max_{0 \le x_n^* \le \min\{S, Q_n\}} \{g_i(x_n^*) + f_{n-1}(S - x_n^*)\}, \\ f_{i-1}(S - x_i^*) &= \max_{0 \le x_{i-1} \le \min\{S - x_n^*, Q_{n-1}\}} \{g_{n-1}(x_{n-1}) + f_{n-2}\left[(S - x_n^*) - x_{n-1}\right]\}. \end{aligned}$$

.....

Na kraju, za vrednost x_1^* maksimalna dobit u prvom potrošaču je:

$$\begin{aligned} f_1(S - x_n^* - x_{n-1}^* - \cdots - x_2^*) &= \max_{0 \le x_1 \le \min\{S - x_n^* - \cdots - x_2^*, Q_1\}} g_1(x_1) \\ &= g_1\left(S - x_n^* - x_{n-1}^* - \cdots - x_2^*\right). \end{aligned}$$

Napomenimo još jednom da je potrebno izračunati vrednosti funkcija $f_i(s)$ za svaku vrednost argumenta $0 \le s \le S$. Pritom, ukoliko su x_i (odnosno s) po prirodi celi brojevi, onda je i S ceo broj i potrebno je izračunati ukupno S vrednosti svake od funkcija f_i. Medjutim, ukoliko su x_i realni brojevi, tada je potrebno znati vrednosti funkcija f_i na celom intervalu $[0, S]$. Prema tome, da bi metod mogli praktično da primenimo u ovom slučaju, potrebno je naći analitički izraz za svaku od funkcija $f_i(s)$.

Razmotrimo sada dva primera problema proste raspodele jednorodnog resursa. U prvom primeru funkcija cilja je linearna dok je u drugom nelinearna.

Primer 2.1.1 U pekari je potrebno rasporediti jednorodni resurs, vreće brašna, na linije za proizvodnju peciva, kolača i hleba. Ostvarena dobit linearno zavisi od količine dodeljenog resursa i iznosi:

- za pecivo: $d_1 = 4$ n.j./jedinici resursa;
- za kolače: $d_2 = 5$ n.j./jedinici resursa;
- za hleb: $d_3 = 2$ n.j./jedinici resursa.

U razmatranom periodu, pekara će imati na raspolaganju do 8 vreća brašna, dok su kapaciteti svake linije prerada do 5 vreća brašna. Zbog higijensko-tehničkih uslova nije dozvoljeno presipanje brašna, odnosno deljenje jedinice resursa.

Rešenje: U ovom primeru, jedinica resursa jeste vreća brašna. Neka je x_1 broj vreća brašna koje su upotrebljene za pecivo, x_2 broj vreća brašna koje su upotrebljene za kolače, dok je x_3 vreća brašna koje su upotrebljene za hleb.

Matematički model je

$$\begin{aligned} \max \quad & F(x) = 4x_1 + 5x_2 + 2x_3 \\ \text{p.o.} \quad & x_1 + x_2 + x_3 \le 8 \\ & 0 \le x_i \le 5, \ i = 1, 2, 3. \end{aligned}$$

Dodatno ograničenje je zahtev da promenljive budu celobrojne (sadržaj vreća se ne moze deliti i presipati). Neposredne dobiti od raspodele resursa linijama su date u sledećoj tabeli.

S	x_1	$f_1(S)$	x_2	$f_2(S)$	x_3	$f_3(S)$
0	0	0	0	0	0	0
1	1	4	1	5	1	5
2	2	8	2	10	2	10
3	3	12	3	15	3	15
4	4	16	4	20	4	20
5	5	20	5	25	5	25
6	5	20	5	29	5	29
7	5	20	5	33	5	33
8	5	20	5	37	5	37

Tabela 2.1.1.

Vrednosti dobiti u koloni $f_1(S)$ se izračunavaju na osnovu izraza za funkciju cilja, u slučaju da se resurs dodeljuje samo prvoj liniji (pravljenje peciva):

$$f_1(S) = \max_{0 \le x_1 \le 5} 4x_1.$$

U slučaju da se resurs dodeljuje liniji 1 i 2 (za proizvodnju peciva koristi se x_1 vreća i x_2 vreća za proizvodnju kolača), optimalna dobit je prikazana u koloni $f_2(S)$, koja se računa po sledećem izrazu:

$$f_2(S) = \max_{0 \le x_2 \le 5} \{5x_2 + f_1(S - x_2)\}$$

na osnovu čega sledi:

$$f_2(0) = \max_{x_2=0} \{5x_2 + f_1(0 - x_2)\} = \max\{5 \cdot 0 + f_1(0-0)\} = 0 + 0 = 0$$

$$f_2(1) = \max_{\substack{x_2=0 \\ x_2=1}} \{5x_2 + f_1(1 - x_2)\} = \max \left\{ \begin{array}{l} 5 \cdot 0 + f_1(1-0) = 0 + 4 = 4 \\ 5 \cdot 1 + f_1(1-1) = 5 + 0 = 5 \end{array} \right\} = 5$$

$$f_2(2) = \max_{\substack{x_2=0 \\ x_2=1 \\ x_2=2}} \{5x_2 + f_1(2 - x_2)\} = \max \left\{ \begin{array}{l} 5 \cdot 0 + f_1(2-0) = 0 + 8 = 8 \\ 5 \cdot 1 + f_1(2-1) = 5 + 4 = 9 \\ 5 \cdot 2 + f_1(2-2) = 10 + 0 = 10 \end{array} \right\} = 10.$$

.....

$$f_2(8) = \max_{0 \le x_2 \le 5} \{5x_2 + f_1(8 - x_2)\} = \max \left\{ \begin{array}{l} 5 \cdot 0 + f_1(8-0) = 0 + 20 = 20 \\ 5 \cdot 1 + f_1(8-1) = 5 + 20 = 25 \\ 5 \cdot 2 + f_1(8-2) = 10 + 20 = 30 \\ 5 \cdot 3 + f_1(8-3) = 15 + 20 = 35 \\ 5 \cdot 4 + f_1(8-4) = 20 + 16 = 36 \\ 5 \cdot 5 + f_1(8-5) = 25 + 12 = 37 \end{array} \right\} = 37.$$

Ako se u razmatranje uvrsti i raspodela resursa na treću liniju, optimalna dobit se računa po sledećoj formuli:

$$f_3(S) = \max_{0 \le x_3 \le 5} \{2x_3 + f_2(S - x_3)\}$$

$$f_3(0) = \max_{x_3=0} \{2 \cdot x_3 + f_2(0 - x_3)\} = \max\{2 \cdot 0 + f_2(0-0)\} = \max\{2 \cdot 0 + f_2(0-0)\}$$

$$= 0 + 0 = 0$$

$$f_3(1) = \max_{\substack{x_3=0 \\ x_3=1}} \{2 \cdot x_3 + f_2(1 - x_3)\} = \max \left\{ \begin{array}{l} 2 \cdot 0 + f_2(1-0) = 0 + 5 = 5 \\ 2 \cdot 1 + f_2(1-1) = 2 + 0 = 2 \end{array} \right\} = 5$$

$$f_3(2) = \max_{\substack{x_3=0 \\ x_3=1 \\ x_3=2}} \{2 \cdot x_3 + f_2(2 - x_3)\} = \max \left\{ \begin{array}{l} 2 \cdot 0 + f_2(2-0) = 0 + 10 = 10 \\ 2 \cdot 1 + f_2(2-1) = 2 + 5 = 7 \\ 2 \cdot 2 + f_2(2-2) = 4 + 0 = 4 \end{array} \right\} = 10$$

.....

$$f_3(8) = \max_{0\le x_3\le 5}\{2\cdot x_3 + f_2(8-x_3)\} = \max\left\{\begin{array}{l} 2\cdot 0 + f_2(8-0) = 0+37 = 37 \\ 2\cdot 1 + f_2(8-1) = 2+33 = 35 \\ 2\cdot 2 + f_2(8-2) = 4+29 = 33 \\ 2\cdot 3 + f_2(8-3) = 6+25 = 31 \\ 2\cdot 4 + f_2(8-4) = 8+20 = 28 \\ 2\cdot 5 + f_2(8-5) = 10+15 = 25 \end{array}\right\} = 37$$

Optimalno rešenje glasi:

$$x^* = \begin{bmatrix} x_1^* \\ x_2^* \\ x_3^* \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 0 \end{bmatrix},$$

dok je vrednost funkcije cilja

$$f(x^*) = 4x_1 + 5x_2 + 2x_3 = 4\cdot 3 + 5\cdot 5 + 2\cdot 0 = 37.$$

Kako je u pitanju jednostavni slučaj distribucije jednorodnog resursa, problem se može rešiti direktnim poređenjem vrednosti funkcije dobiti, a u zavisnosti od mogućih stanja distribucije resursa, kao što je dato u tabeli 2.1.2.

x_1	0	1	2	3	4	5	5	5	5
$(S-x_1)$	8	7	6	5	4	3	3	3	3
x_2	5	5	5	5	4	3	3	3	3
$(S-x_2)$	3	3	3	3	4	5	5	5	5
x_3	3	2	1	0	0	0	0	0	0
$g_1(x_1)$	0	4	8	12	16	20	20	20	20
$g_2(x_2)$	25	25	25	25	20	15	15	15	15
$g_3(x_3)$	6	4	2	0	0	0	0	0	0
$\sum g_i(x_i)$	31	33	35	37	36	35	35	35	35

Tabela 2.1.2.

I ovim metodom su dobijeni isti rezultati, to jest, optimalno rešenje je

$$x^* = \begin{bmatrix} x_1^* \\ x_2^* \\ x_3^* \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 0 \end{bmatrix},$$

pri čemu je vrednost funkcije cilja:

$$f(x^*) = 4x_1 + 5x_2 + 2x_3 = 4\cdot 3 + 5\cdot 5 + 2\cdot 0 = 37.$$

Primer 2.1.2 Za potrebe rada proizvodnog sistema, potrebno je pamučno platno (jednorodni resurs) raspodeliti na tri moguće proizvodne linije, pri čemu je kapacitet proizvodnih linija limitiran na količinu platna (resursa) $0 \le x_i \le 4$, $i = 1, 2, 3$. Količina platna je takođe u razmatranom periodu ograničena i iznosi $R = 7$ jedinica. Ostvarena dobit na linijama zavisi od kvadrata količine prerađenog platna, i to:

za liniju 1: $d_1 = 3x_1^2$
za liniju 2: $d_2 = 4x_2^2$
za liniju 3: $d_3 = 2x_3^2$.

Ograničenu količinu platna treba tako distribuirati proizvodnim linijama tako da ostvarena dobit od njegove prerade bude maksimalna.

Rešenje: Potrebno je maksimizirati funkciju kriterijuma (cilja):

$$D(x) = 3x_1^2 + 4x_2^2 + 2x_3^2.$$

Ograničenja potiču od ograničene količine resursa i od kapaciteta linija, i glase:

$$\begin{aligned} x_1 + x_2 + x_3 &\le 7 \\ 0 \le x_i &\le 4. \end{aligned}$$

Dobit se može izraziti kao funkcija raspoloživog resursa, i data je u sledećoj tabeli.

R	x_1	$D_1(x_1)$	x_2	$D_2(x_2)$	x_3	$D_3(x_3)$
0	0	0	0	0	0	0
1	1	3	1	4	1	4
2	2	12	2	16	2	16
3	3	27	3	36	3	36
4	4	48	4	64	4	64
5	4	48	4	67	4	67
6	4	48	4	76	4	76
7	4	48	4	91	4	91

Tabela 2.1.3.

Kolona $D_1(x_1)$ u tabeli se računa pomoću izraza:

$$D_1(x_1) = \max_{0 \le x_1 \le 4} \left\{ 3x_1^2 \right\}.$$

Ako se u razmatranje uvede prvi i drugi proces, vrednosti u koloni $D_2(x_2)$ dobijaju se pomoću izraza:

$$D_2(x_2) = \max_{0 \le x_2 \le 4} \left\{ 4x_2^2 + D_1(R - x_2) \right\}.$$

Odatle slede rezultati:

$$\begin{aligned}
D_2(0) &= \max_{x_2=0} \left\{ 4x_2^2 + D_1(0 - x_2) \right\} = \max \left\{ 4 \cdot 0^2 + D_1(0-0) \right\} = 0, \\
D_2(1) &= \max_{\substack{x_2=0 \\ x_2=1}} \left\{ 4x_2^2 + D_1(1 - x_2) \right\} = \max \left\{ \begin{array}{l} 4 \cdot 0^2 + D_1(1-0) = 3 \\ 4 \cdot 1^2 + D_1(1-1) = 4 \end{array} \right\} = 4, \\
D_2(2) &= \max_{\substack{x_2=0 \\ x_2=1 \\ x_2=2}} \left\{ 4x_2^2 + D_1(2 - x_2) \right\} = \max \left\{ \begin{array}{l} 4 \cdot 0^2 + D_1(2-0) = 12 \\ 4 \cdot 1^2 + D_1(2-1) = 7 \\ 4 \cdot 2^2 + D_1(2-2) = 16 \end{array} \right\} = 16,
\end{aligned}$$

$$\begin{aligned}
D_2(3) &= \max_{\substack{x_2=0 \\ x_2=1 \\ x_2=2 \\ x_2=3}} \left\{ 4x_2^2 + D_1(3 - x_2) \right\} = \max \left\{ \begin{array}{l} 4 \cdot 0^2 + D_1(3-0) = 27 \\ 4 \cdot 1^2 + D_1(3-1) = 16 \\ 4 \cdot 2^2 + D_1(3-2) = 19 \\ 4 \cdot 3^2 + D_1(3-3) = 36 \end{array} \right\} = 36, \\
D_2(4) &= \max_{\substack{x_2=0 \\ x_2=1 \\ x_2=2 \\ x_2=3 \\ x_2=4}} \left\{ 4x_2^2 + D_1(4 - x_2) \right\} = \max \left\{ \begin{array}{l} 4 \cdot 0^2 + D_1(4-0) = 48 \\ 4 \cdot 1^2 + D_1(4-1) = 31 \\ 4 \cdot 2^2 + D_1(4-2) = 28 \\ 4 \cdot 3^2 + D_1(4-3) = 39 \\ 4 \cdot 4^2 + D_1(4-4) = 64 \end{array} \right\} = 64,
\end{aligned}$$

$$D_2(5) = \max_{\substack{x_2=0\\x_2=1\\x_2=2\\x_2=3\\x_2=4}} \left\{4x_2^2 + D_1(5-x_2)\right\} = \max \left\{ \begin{array}{l} 4\cdot 0^2 + D_1(5-0) = 48 \\ 4\cdot 1^2 + D_1(5-1) = 52 \\ 4\cdot 2^2 + D_1(5-2) = 43 \\ 4\cdot 3^2 + D_1(5-3) = 48 \\ 4\cdot 4^2 + D_1(5-4) = 67 \end{array} \right\} = 67,$$

$$D_2(6) = \max_{\substack{x_2=0\\x_2=1\\x_2=2\\x_2=3\\x_2=4}} \left\{4x_2^2 + D_1(6-x_2)\right\} = \max \left\{ \begin{array}{l} 4\cdot 0^2 + D_1(6-0) = 48 \\ 4\cdot 1^2 + D_1(6-1) = 52 \\ 4\cdot 2^2 + D_1(6-2) = 64 \\ 4\cdot 3^2 + D_1(6-3) = 63 \\ 4\cdot 4^2 + D_1(6-4) = 76 \end{array} \right\} = 76,$$

$$D_2(7) = \max_{\substack{x_2=0\\x_2=1\\x_2=2\\x_2=3\\x_2=4}} \left\{4x_2^2 + D_1(7-x_2)\right\} = \max \left\{ \begin{array}{l} 4\cdot 0^2 + D_1(7-0) = 48 \\ 4\cdot 1^2 + D_1(7-1) = 52 \\ 4\cdot 2^2 + D_1(7-2) = 64 \\ 4\cdot 3^2 + D_1(7-3) = 84 \\ 4\cdot 4^2 + D_1(7-4) = 91 \end{array} \right\} = 91.$$

Kada se uzme u obzir mogućnost raspodele resursa na sva tri procesa, izraz za maksimalnu dobit u funkciji raspoloživog resursa je:

$$D_3(x_3) = \max_{0\le x_3\le 4} \left\{2x_3^2 + D_2(R-x_3)\right\}.$$

Na osnovu ovog izraza su izračunate vrednosti u koloni $D_3(x_3)$. U ovom slučaju su dobijene maksimalne vrednosti dobiti, pri čemu je uzeta u obzir optimalna dobit drugog i trećeg procesa.

$$D_3(0) = \max_{x_3=0} \left\{2x_3^2 + D_2(0-x_3)\right\} = \max \left\{2\cdot 0^2 + D_2(0-0)\right\} = 0,$$

$$D_3(1) = \max_{\substack{x_3=0\\x_3=1}} \left\{2x_3^2 + D_2(1-x_3)\right\} = \max \left\{ \begin{array}{l} 2\cdot 0^2 + D_2(1-0) = 4 \\ 2\cdot 1^2 + D_2(1-1) = 2 \end{array} \right\} = 4,$$

$$D_3(2) = \max_{\substack{x_2=0\\x_2=1\\x_2=2}} \left\{2x_3^2 + D_2(2-x_3)\right\} = \max \left\{ \begin{array}{l} 2\cdot 0^2 + D_2(2-0) = 12 \\ 2\cdot 1^2 + D_2(2-1) = 7 \\ 2\cdot 2^2 + D_2(2-2) = 16 \end{array} \right\} = 16,$$

$$D_3(3) = \max_{\substack{x_3=0\\x_3=1\\x_3=2\\x_3=3}} \left\{2x_3^2 + D_2(3-x_3)\right\} = \max \left\{ \begin{array}{l} 2\cdot 0^2 + D_2(3-0) = 27 \\ 2\cdot 1^2 + D_2(3-1) = 16 \\ 2\cdot 2^2 + D_2(3-2) = 19 \\ 2\cdot 3^2 + D_2(3-3) = 36 \end{array} \right\} = 36,$$

$$D_3(4) = \max_{\substack{x_3=0\\x_3=1\\x_3=2\\x_3=3\\x_3=4}} \left\{2x_3^2 + D_2(4-x_3)\right\} = \max \left\{ \begin{array}{l} 2\cdot 0^2 + D_2(4-0) = 48 \\ 2\cdot 1^2 + D_2(4-1) = 31 \\ 2\cdot 2^2 + D_2(4-2) = 28 \\ 2\cdot 3^2 + D_2(4-3) = 39 \\ 2\cdot 4^2 + D_2(4-4) = 64 \end{array} \right\} = 64,$$

$$D_3(5) = \max_{\substack{x_3=0\\x_3=1\\x_3=2\\x_3=3\\x_3=4}} \left\{2x_3^2 + D_2(5-x_3)\right\} = \max \left\{ \begin{array}{l} 2\cdot 0^2 + D_2(5-0) = 48 \\ 2\cdot 1^2 + D_2(5-1) = 52 \\ 2\cdot 2^2 + D_2(5-2) = 43 \\ 2\cdot 3^2 + D_2(5-3) = 48 \\ 2\cdot 4^2 + D_2(5-4) = 67 \end{array} \right\} = 67,$$

$$D_3(6) = \max_{\substack{x_3=0\\x_3=1\\x_3=2\\x_3=3\\x_3=4}} \left\{2x_3^2 + D_2(6-x_3)\right\} = \max \left\{ \begin{array}{l} 2\cdot 0^2 + D_2(6-0) = 48 \\ 2\cdot 1^2 + D_2(6-1) = 52 \\ 2\cdot 2^2 + D_2(6-2) = 64 \\ 2\cdot 3^2 + D_2(6-3) = 63 \\ 2\cdot 4^2 + D_2(6-4) = 76 \end{array} \right\} = 76,$$

$$D_3(7) = \max_{\substack{x_3=0\\x_3=1\\x_3=2\\x_3=3\\x_3=4}} \left\{2x_3^2 + D_2(7-x_3)\right\} = \max \left\{ \begin{array}{l} 2\cdot 0^2 + D_2(7-0) = 48 \\ 2\cdot 1^2 + D_2(7-1) = 52 \\ 2\cdot 2^2 + D_2(7-2) = 64 \\ 2\cdot 3^2 + D_2(7-3) = 84 \\ 2\cdot 4^2 + D_2(7-4) = 91 \end{array} \right\} = 91.$$

Maksimalna vrednost dobijena na osnovu treće funkcije predstavlja i rešenje problema, tako da je:

$$maxD(X) = D_3(7) = 91(n.j.).$$

Optimalna vrednost za treću promenljivu je $x_3 = 0$ ($D_3(7) = 91$). Vrednosti ostale dve promenljive se određuju is tabele, tako da je $x_2 = 4$, dok iz ograničenja sledi $x_1 = 3$. Ove vrednosti su i podvučene u sledećoj tabeli.

Pošto je u pitanju jednostavni slučaj distribucije jednorodnog resursa, problem se može rešiti direktnim poređenjem vrednosti funkcije dobiti, a u zavisnosti od mogućih stanja distribucije resursa, kao što je dato u tabeli.

x_1	0	1	2	**3**	4	4	4	4
$R - x_1$	7	6	5	4	3	3	3	3
x_2	4	4	4	**4**	3	3	3	3
$R - x_2$	3	3	3	3	4	4	4	4
x_3	3	2	1	**0**	0	0	0	0
$R - x_3$	4	5	6	7	7	7	7	7
$c_1x_1^2$	0	3	12	27	48	48	48	48
$c_2x_2^2$	64	64	64	64	36	36	36	36
$c_3x_3^2$	18	8	2	0	0	0	0	0
$\sum_{i=1}^{3} c_j x_j^2$	82	75	78	**91**	84	84	84	84

Tabela 2.1.4.

Optimalno rešenje je

$$x^* = \begin{bmatrix} x_1^* \\ x_2^* \\ x_3^* \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix},$$

dok je vrednost funkcije cilja

$$\max D(x) = D(x^*) = 3x_1^{*2} + 4x_2^{*2} + 2x_3^{*2} = 91.$$

2.2 Raspodela poslova na mašine

Problem raspodele različitih poslova na određen broj mašina može se definisati na sledeći način:

U nekom pogonu postoji n jednorodnih mašina koje mogu da obavljaju operacije (ili poslove), označene sa 1 i 2. Ako x mašina u toku jednog razmatranog vremenskog perioda izvršava operaciju 1, to će kompaniji doneti dobit $g(x)$ na kraju tog perioda. Medjutim, jedan deo mašina biće amortizovan, odnosno neće biti za dalju upotrebu tako da se u narednom periodu može računati na $a(x)$ mašina od x mašina koje

su obavljale operaciju 1. Na sličan način, ako y mašina u toku prvog razmatranog perioda izvršava operaciju 2, to će kompaniji doneti dobit $h(y)$ a ukupno $b(y)$ mašina biće i dalje raspoloživo za rad. U svakom vremenskom periodu svaka raspoloživa mašina obavlja jednu od operacija, tj nijedna mašina ne pauzira.

Potrebno je odrediti broj mašina x_i koje treba da obavljaju operaciju 1 u i-tom vremenskom periodu $i = 1, \ldots, N$ (preostale ispravne mašine obavljaju operaciju 2 u tom periodu), tako da ostvarena dobit posle N perioda bude maksimalna.

Rešenje: Za prvi period važe sledeće jednačine:

- za operaciju 1 se koristi x_1 mašina, dok za obavljanje operacije 2 preostaje $y_1 = n_1 - x_1$ ($n_1 = n$= ukupan broj raspoloživih mašina na početku);
- ukupna dobit u prvom periodu iznosi: $g(x_1) + h(y_1)$;
- broj upotrebljivih mašina na kraju prvog perioda (upotrebljivih za rad u drugom periodu) je: $n_2 = a(x_1) + b(y_1)$.

Za drugi period važi:

- broj upotrebljivih mašina je: $n_2 = a(x_1) + b(y_1)$;
- za operaciju 1 se koristi x_2 mašina (od raspoloživih n_2, dok za obavljanje operacije 2 preostaje $y_2 = n_2 - x_2$;
- ukupna dobit u drugom periodu iznosi: $g(x_2) + h(y_2)$;
- broj upotrebljivih mašina na kraju drugog perioda (upotrebljivih za rad u trećem periodu) je: $n_3 = a(x_2) + b(y_2)$.

Na isti način se mogu napisati izrazi do N-tog perioda. Matematički model problema može se iskazati na sledeći način:

$$\begin{aligned} \max F(x,y) \quad &= \sum_{i=1}^{N} [g(x_i) + h(y_i)] \\ \text{p.o.} \quad &x_i + y_i = n_i \\ &n_{i+1} = a(x_i) + b(y_i), \quad i = 1, 2, \ldots, N-1 \\ &0 \le x_i \le n_i, \quad i = 1, 2, \ldots, N. \end{aligned}$$

Kao i kod predhodnog problema, i ovde se u slučaju linearne funkcije cilja $F(x, y)$ i ograničenja problem svodi na problem linearnog programiranja. Izložimo sada rešenje korišćenjem tehnike DP.

Ako se za $f_i(n)$ obeleži maksimalna dobit posle i perioda, pri čemu je n broj raspoloživih mašina na početku prvog perioda, tada se rekurentne jednačine dinamičkog programiranja mogu napisati na osnovu sledećih pravila:

- ukoliko do kraja planskog perioda prestaje samo jedan period (ili se razmatra problem koji ima samo jedan period), tada je:

$$f_1(n) = \max_{0 \le x_1 \le n} \{g(x_1) + h(n - x_1)\};$$

- ukoliko je do kraja planskog perioda je preostalo k perioda, tada važi:

$$f_k(n) = \max_{0 \le x_k \le n} \{g(x_k) + h(n - x_k) + f_{k-1}[a(x_k) + b(n - x_k)]\}, \quad k > 1.$$

Optimalne vrednosti $x_1^*, \ldots, x_n^*$ se izračunavaju na isti način kao u predhodnom problemu.

Primer 2.2.1 Preduzeće raspolaže sa 100 visokoproduktivnih presa. U naredne tri godine ($N = 3$) preduzeće bi trebalo da izrađuje otpreske tipa A i tipa B, uz zamenu alata. U zavisnosti od raspodele mašina, u i-tom periodu se može ostvariti dobit:

$$g(x_i) + h(y_i) = 0.9x_i + 0.5y_i.$$

Amortizacija zavisi od dela koji se izrađuje na mašini (zbog težine, potrebne sile i slično) i iznosi 70% za mašine koje izrađuju otpresak tipa A i 20% za mašine koje izrađuju otpresak B, tako da broj raspoloživih mašina za naredni period iznosi:

$$n_{i+1} = 0.3x_i + 0.8y_i.$$

Matematički model problema:

$$\begin{aligned} \max F(x) = & \sum_{i=1}^{3} (0.9x_i + 0.5y_i) \\ \text{p.o.} \quad & x_i + y_i = n_i \\ & n_{i+1} = 0.3x_i + 0.8y_i \\ & 0 \le x_i \le n_i, \quad n_1 = n. \end{aligned}$$

- Ako se razmatra samo treći period (period u kome je do kraja planskog perioda ostao samo jedan, treći period), maksimalna dobit će iznositi:

$$f_1(n) = \max_{0 \le x_3 \le n_3} \{0.9x_3 + 0.5(n_3 - x_3)\}.$$

Maksimalna dobit se dobija za $x_3 = n_3$, i iznosi $0.9n_3$.

- U drugom koraku ($k = 2$) posmatra se maksimalna dobit za treći i drugi period, pa rekurentna jednačina glasi:

$$f_2(n) = \max_{0 \le x_2 \le n_2} \{0.9x_2 + 0.5(n_2 - x_2) + f_1(0.3x_2 + 0.8(n_2 - x_2))\}$$

ili

$$\begin{aligned} f_2(n) &= \max_{0 \le x_2 \le n_2} \{0.4x_2 + 0.5n_2 + 0.9(0.3x_2 + 0.8(n_2 - x_2))\} \\ &= \max_{0 \le x_2 \le n_2} \{1.22n_2 - 0.05x_2\} \\ &= 1.22n_2 \quad \text{za } x_2 = 0. \end{aligned}$$

Kada se uzmu u obzir sva tri perioda, maksimalna dobit će iznositi:

$$f_3(n) = \max_{0 \le x_1 \le n_1} \{0.9x_1 + 0.5(n_1 - x_1) + f_2(0.3x_1 + 0.8(n_1 - x_1))\}.$$

Na osnovu prethodnih razmatranja, odavde se dobija

$$f_3(n) = \max_{0\le x_1\le n_1} \{0.9x_1 + 0.5(n_1 - x_1) + 1.22\,(0.3x_1 + 0.8(n_1 - x_1))\}\,,$$

ili

$$\begin{aligned} f_3(n) &= \max_{0\le x_1\le n_1} \{0.4x_1 + 0.5n_1 + 1.22(0.8n_1 - 0.5x_1)\}\,, \\ &= \max_{0\le x_1\le n_1} \{1.476n_1 - 0.21x_1\} = 1.476n_1 = 147.6. \end{aligned}$$

Maksimalna vrednost za $f_3(n)$ se dobija za $x_1 = 0$.

Rešenje problema je:

$$\begin{aligned} \max\ F(x) &= f_3(100) = 147.6 \\ x_1 &= 0,\ y_1 = 100 \\ n_2 &= 0.3x_1 + 0.8y_1 = 80, \\ x_2 &= 0,\ y_2 = 80, \\ n_3 &= 0.3x_2 + 0.8y_2 = 64, \\ x_3 &= 64,\ y_3 = 0. \end{aligned}$$

2.3 Optimalna politika zamene opreme

Principi zamene opreme moraju biti u skladu sa produktivnošću opreme, troškovima korišćenja, kao i preostalom vrednošću opreme.

Neka je:

$D(t)$ - dobit koja se ostvaruje upotrebom opreme stare t godina;
$C(t)$ - godišnji troškovi održavanja opreme;
$V(t)$ - preostala vrednost opreme stare t godina;
C - nabavna cena nove opreme.

U razmatranom periodu od N godina (gde je N razmatrani period, život projekta, život proizvoda koji se izrađuju na razmatranoj opremi) potrebno je odrediti optimalni ciklus zamene opreme.

Rešenje: Funkcija cilja $f_N(t)$ predstavlja dobit u razmatranom periodu od N godina koja se ostvaruje eksploatacijom opreme stare t godina.

Da bi se postavila funkcionalna relacija, potrebno je naći vezu između karakterističnih veličina u toku dva susedna perioda.

Periodi na koje se ukupni period N godina rasčlanjuje, računaju se od kraja. Na početku bilo kog (k-tog) perioda raspolaže se opremom starom t godina i moguća su dva alternativna rešenja:

- zadržati staru opremu (staru t godina) i ostvariti dobit:

$$D(t) - C(t) + f_{k-1}(t+1);$$

- nabaviti novu opremu i ostvariti dobit:

$$V(t) - C + D(t) - C(t) + f_{k-1}(t).$$

Optimalna dobit od eksploatacije opreme u preostalih k perioda $f_k(t)$ je:

$$f_k(t) = \max \begin{cases} D(t) - C(t) + f_{k-1}(t+1) \\ V(t) - C + D(t) - C(t) + f_{k-1}(t). \end{cases}$$

Za $k = 1$, odnosno maksimalna dobit koja se ostvaruje u slučaju kada je preostao samo jedan period, korišćenjem opreme stare t godina (pri čemu je $t = 1, \ldots, N$), iznosi:

$$f_1(t) = \max \begin{cases} D(t) - C(t) \\ V(t) - C + D(t) - C(t). \end{cases} \quad , t = 1, \ldots, N.$$

2.4 Problem maksimalnog zbira

Poreklo zadatka: [25].

Neka je dat pravougaonik A sa M vrsta i N kolona, popunjen celim brojevima. Iz svakog polja je dozvoljeno preći samo na polje ispod ili na polje desno od tog polja. Potrebno je izabrati put od gornjeg levog polja do donjeg desnog polja, tako da zbir brojeva u poljima preko kojih se prolazi, bude maksimalan. Ispisati vrednost optimalnog puta, a zatim i put kao niz koordinata polja preko kojih se prolazi.

Rešenje: Generisanje svih mogućih puteva i pamćenje onog puta koji ima najveći zbir, nije dobra ideja, jer broj mogućih puteva raste eksponencijalnom brzinom sa porastom veličine pravougaonika.

Proverimo uslove za primenu dinamičkog programiranja, tj da svaki podproblem polaznog problema ima optimalnu podstrukturu. Neka je dat put P čija polja imaju najveći zbir. Tada svaki deo optimalnog puta spaja polazno i završno polje tog dela puta na optimalan način (inače polazni put ne bi bio optimalan). Pogodno je podprobleme formalno zadati na sledeći način: neka je $P(i, j)$ optimalni put od polja $(1, 1)$ do polja (i, j), a $B(i, j)$ neka je zbir koji se na tom putu postiže. Tada je potrebno da se pronađe put $P(M, N)$ i zbir $B(M, N)$. Već smo ustanovili da problem ima optimalnu podstrukturu. Do polja (I, J) možemo doći samo preko jednog od polja $(I, J-1)$ ili $(I-1, J)$. Zbir $B(I, J)$ je zato jednak jednom (većem) od $B(I, J-1)$ ili $B(I-1, J)$, uvećanom za $A(I, J)$:

$$B(I,J) = \begin{cases} A(1,1), & I = J = 1, \\ B(1, J-1) + A(1,J), & I = 1, J \geq 2, \\ B(I-1,1) + A(I,1), & J = 1, I \geq 2, \\ \max\{B(I,J-1), B(I-1,J)\} + A(I,J), & I \geq 2, J \geq 2. \end{cases}$$

Tada se put $P(I, J)$ dobija dodavanjem polja (I, J) na jedan od puteva $P(I-1, J)$ ili $P(I, J-1)$ koji daje veći zbir.

Prema tome, rešavanje polaznog problema se svodi na rešavanje dva podproblema istog tipa, i njihovo jednostavno kombinovanje - poređenje dva zbira i sabiranje većeg od ta dva zbira sa vrednošću poslednjeg polja. Za rešavanje svih netrivijalnih podproblema važi potpuno isto. Trivijalni problemi su nalaženje elemenata prve vrste i prve kolone matrice B. Kako do polja $(1, J)$ ili $(I, 1)$ postoji samo jedan put, treba samo sabrati polja na tom putu.

Podproblemi se i u ovom problemu preklapaju. Na primer, problem za polje sa koordinatama $(i-1, j-1)$ se pojavljuje u oba potproblema $(I, J-1)$ i $(I-1, J)$. Zato rekurzija nikako nije prihvatljiva za rešavanje globalnog problema. Većina podproblema se rekurzivnim rešavanjem dalje svodi na dva manja podproblema, pa bi rekurzijom ukupno bilo potrebno rešiti eksponencijalno mnogo podproblema umesto samo $M * N$, koliko ih ukupno ima različitih. Stoga ćemo rešiti redom sve različite podprobleme, to jest, koristićemo dinamičko programiranje.

Koordinate polja koja sačinjavaju optimalan put se generišu od poslednjeg polja ka prvom. Da bismo ih ispisali od prvog ka poslednjem, možemo koristiti stek, pomoću koga obrćemo redosled podataka. Međutim, prilikom poziva funkcija i procedura, operativni sistem već koristi stek za smeštanje podataka, pa je primena rekurzije jednostavniji i prirodan način da se ispišu koordinate polja u željenom redosledu.

Napomena: Program koji sledi je dobijen dinamičkim programiranjem, a rekurzija se samo koristi za ispisivanje rešenja. Procedura $ispisi(M, N)$ koja ispisuje put do polja (M, N), poziva se rekurzivno najviše onoliko puta koliko ima polja na putu od polja $(1, 1)$ do polja (M, N), dakle ukupno manje od $M + N$ puta, tako da se ona brzo izvršava. Rekurzija se može upotrebiti zbog toga što je hijerarhija u rekurzivnom pozivanju jednoznačno određena.

Implementacija:

```
program maksimalni_zbir_u_matrici;
  var a,s:array[1..30,1..30] of integer;
      i,j,m,n:integer;
  procedure ispis(i,j:integer);
    var k:integer;
    begin
      if(i>1) and (j>1) then
        begin
          if s[i-1,j]>s[i,j-1] then ispis(i-1,j)
          else ispis(i,j-1);
          writeln(i:3,j:3);
        end
      else if i=1 then  for k:=1 to j do writeln(i:3,k:3)
      else for k:=1 to i do writeln(k:3,j:3);
    end;

  begin
    readln(m,n);
    for i:=1 to m do
      begin
        for j:=1 to n do read(a[i,j]);   readln
      end;
    s[1,1]:=a[1,1];
    for j:=2 to n do s[1,j]:=a[1,j]+s[1,j-1];
    for i:=2 to m do s[i,1]:=a[i,1]+s[i-1,1];
    for i:=2 to m do
      for j:=2 to n do
        if s[i,j-1]<s[i-1,j] then  s[i,j]:=a[i,j]+s[i-1,j]
        else s[i,j]:=a[i,j]+s[i,j-1];
    writeln('Maksimalni zbir je ',s[m,n]);
```

```
    writeln('Postize se na sledeci nacin: ');    ispis(m,n);
  end.
```

Na primer, za matricu oblika

4	3	5	7	5
1	9	4	1	3
2	3	5	1	2
1	3	1	2	0
4	6	7	2	1

dobija se maksimalni zbir 38. Ovaj optimalni put se postiže korišćenjem sledećih polja:

1 1, 1 2, 2 2, 3 2, 4 2, 5 2, 5 3, 5 4, 5 5.

2.5 Problem maksimalnog zbira: još nekoliko varijanti

Problem maksimalnog zbira može se sresti u više srodnih varijanti. U svakoj varijanti podrazumevaćemo (ako se drugačije ne napomene) da je potrebno kretanjem kroz matricu postići najveći zbir i da se na svako polje može stati najviše jednom.

- Kretanjem dole i desno stići od polja $(1,1)$ do polja (m,n). Ovo je varijanta koja je razmatrana i rešena u predhodnom problemu.
- Kretanjem dole, desno i dole-desno stici od polja $(1,1)$ do polja (m,n). U ovoj varijanti elementi matrice B se računaju neznatno drugačije:

$$B(X,Y) = \max\{B(X,Y-1), B(X-1,Y), B(X-1,Y-1)\} + A(X,Y)$$

Detalje oko konstrukcije rekurentne formule u ovoj i narednim varijantama, kao i implementaciju rešenja u narednim varijantama prepuštamo čitaocu.

Implementacija:

```
program suma;
  uses wincrt;
  type matrica=array[1..50,1..50] of integer;
  var a,s:matrica;
      i,j,n,max:integer;
  procedure ispis(i,j:integer);
    var k:integer;
    begin
     if (i>1) and (j>1) and (i>=j) then
      if j<i then
        begin
          if s[i-1,j]>s[i,j-1] then
            if s[i-1,j]>s[i-1,j-1] then ispis(i-1,j)
                                   else ispis(i-1,j-1)
          else if s[i,j-1]>s[i-1,j-1] then ispis(i,j-1)
                                      else ispis(i-1,j-1);
          write('[',i,',',j,'], ');
        end
      else
        begin
           if s[i,j-1]>s[i-1,j-1] then ispis(i,j-1)
                                  else ispis(i-1,j-1);
```

```
          write('[',i,',',j,'], ');
        end
      else if (i>=1) and (j=1) then
         for k:=1 to i do write('[',k,',',1,'], ')
    end;
begin
  writeln('Unesi stranicu trougla');   readln(n);
  writeln('Unesi vrednost svakog clana');
  for i:=1 to n do
    begin
      for j:=1 to i do read(a[i,j]);
      readln;
    end;
  s[1,1]:=a[1,1];
  for i:=2 to n do s[i,1]:=s[i-1,1]+a[i,1];
  for i:=2 to n do
    for j:=2 to i do
      begin
        if j<i then
          begin
            max:=s[i-1,j-1];
            if s[i-1,j]>max then max:=s[i-1,j];
            if s[i,j-1]>max then max:=s[i,j-1];
          end
        else if j=i then
            if s[i,j-1]>=s[i-1,j-1] then max:=s[i,j-1]
                                    else max:=s[i-1,j-1];
        s[i,j]:=max+a[i,j];
      end;
  writeln(s[n,n]);
  ispis(n,n);
end.
```

• Kretanjem gore-desno, dole-desno i desno, stići od bilo kog polja $(x, 1)$ prve kolone, do bilo kog polja (y, n) poslednje kolone. Ovde se matrica B mora popunjavati po kolonama, za elemente prve kolone važi $B(X, 1) = A(X, 1)$, a za ostale:

$$B(X,Y) = \max\{B(X-1,Y-1), B(X,Y-1), B(X+1,Y-1)\} + A(X,Y)$$

pri čemu smatramo da su $B(X-1,0)$ i $B(X-1,M+1)$ jednaki minus beskonačno, tj. za prvi i poslednji element u koloni maksimum se bira od 2, a ne od 3 člana.

• Kretanjem na dole i desno stići od gornjeg levog do donjeg desnog polja matrica A, i pri tome stati na takve susedne brojeve $x_1, x_2, ..., x_k$ u matrici A ($x_1 = a_{11}, x_k = a_{mn}$), da se maksimizira ne njihov zbir, nego zbir apsolutnih razlika uzastopnih brojeva preko kojih se ide, dakle $\sum_{i=1}^{k-1} |x_{i+1} - x_i|$.

Lako je uvideti da za $X > 1$, $Y > 1$ važi relacija

$$B(X,Y) = \max \left\{ \begin{array}{l} B(X,Y-1) + |A(X,Y-1) - A(X,Y)| \\ B(X-1,Y) + |A(X-1,Y) - A(X,Y)| \end{array} \right\}$$

dok se elementi prve vrste i prve kolone izracunavaju prema formuli

$$B(1,1) = 0$$
$$B(1,Y) + B(1,Y-1) + |A(1,Y-1) - A(1,Y)|$$
$$B(X,1) + B(X-1,1) + |A(X-1,1) - A(X,1)|$$

Na osnovu vrednosti popunjene matrice B nije teško odrediti polja preko kojih se prelazi radi maksimiziranja traženog zbira.

• Potrebno je stići od bilo kojeg elementa prve kolone do bilo kog elementa poslednje kolone, krećući se gore, dole i desno. Matricu B popunjavamo po kolonama, tako da je B(X,Y) najveća vrednost koja se može dostići kretanjem kroz prvih Y kolona i zaustavljanjem u polju A(X,Y). Tada imamo:

$$B(X,1) = \max_{1 \le k \le M} \left\{ \sum_{p=k,X} a_{p,1} \right\}, 1 \le X \le M$$

gde p od k do X po potrebi (za $k > X$) ide i unazad, i

$$B(X,Y) = \max_{1 \le k \le M} \left\{ B(k,Y-1) + \sum_{p=k,X} a_{p,Y} \right\}, 1 \le X \le M, 2 \le\ Y \le N.$$

Ukupan broj operacija je u ovom primeru veći i proporcionalan je sa M^2N, a rezultata je određen maksimumom poslednje kolone matrice B.

Osim navedenih, mogu se naći i druge slične varijante ovog problema. Sve one se rešavaju na način vrlo blizak izloženom.

Brojevni trougao

Dat je trougao popunjen brojevima: u prvom redu je jedan broj, a u drugom dva, itd. do N-tog reda sa N brojeva. Ovaj trougao možemo predstaviti donje-trouganom matricom $A(X,Y)$. U sledećoj tabeli je prikazan "brojevni trougao".

```
7
3  8
8  1  0
2  7  4  4
4  5  2  6  5
```

Napisati program koji nalazi najveću sumu brojeva, po putu koji počinje na vrhu, a završava se na nekom polju baze.

- U svakom koraku možemo ići dole ili dijagonalno dole desno.
- Broj redova u trouglu je veći od 1, a najviše 100.
- Brojevi u trouglu su celi, u intervalu od 0 do 99.

Rešenje: Neka je $B(X,Y)$ maksimalna suma koju je moguće ostvariti kretanjem po brojevnom trouglu od polja $(1,1)$ do polja (X,Y). Važi rekurentna formula

$$B(X,Y) = \max\{B(X-1,Y-1), B(X-1,Y)\} + A(X,Y).$$

Implementacija:

```
program brojnitrougao;
  const maxn=100;
  var a,mat:array[1..maxn,1..maxn] of integer;
      poz,s,i,j,k,n:integer;
      ulaz,izlaz:text;

  procedure ulazpod;
    var i,j:integer;
    begin
      assign(ulaz,'c:\tp\zadaci\input.txt'); assign(izlaz,'c:tp\zadaci\output.txt');
      reset(ulaz); rewrite(izlaz);
      readln(ulaz,n);
      for i:=1 to n do
        for j:=1 to i do
          begin
            read(ulaz,mat[i,j]); a[i,j]:=0;
          end;
      a[1,1]:=mat[1,1];
      close(ulaz);
    end;

  procedure nadjizbir;
    var i,j:integer;
    begin
      for i:=2 to n do
        for j:=1 to i do
          begin
            if (j>1) and (a[i,j]<a[i-1,j-1]+mat[i,j]) then
              a[i,j]:=a[i-1,j-1]+mat[i,j];
            if (i>j) and (a[i,j]<a[i-1,j]+mat[i,j]) then
              a[i,j]:=a[i-1,j]+mat[i,j];
          end;
      s:=0;
      for j:=1 to n do
        if a[n,j]>s then s:=a[n,j];
    end;

  procedure stampaj;
    var i:integer;
    begin
      writeln(izlaz,'Najveci zbir je ',s);   close(izlaz);
    end;

  begin
    ulazpod; nadjizbir; stampaj;
  end.
```

Primer: Za sledeći sadržaj ulazne datoteke

```
5
7
3  8
8  1  0
2  7  7  4
4  5  2  6  5
```

dobija se maksimalni zbir 30, pri čemu je optimalni put

$$\{1,1\}, \quad \{2,1\}, \quad \{3,1\}, \quad \{4,2\}, \quad \{5,2\}.$$

Razmotrimo sada još jednu varijantu problema brojevnog trougla.

• Treba stići od broja na vrhu do bilo kog broja na osnovici trougla. Trougao se može smestitiu kvadratnu matricu na više načina. Neka to bude trougao iznad sopstvene dijagonale. Tako dobijemo ekvivalentnu postavku: u kvadratnoj matrici treba kretanjem na dole i desno stići od gornjeg levog ugla do bilo kog broja na sporednoj dijagonali. Zadatak se rešava tako što na isti način kao i ranije formiramo kvadratnu matricu B, ali je popunjavamo samo do sporedne dijagonale. Rešenje je određeno najvećim elementom na sporednoj dijagonali matrice B.

2.6 Najduži zajednički podniz

Poreklo zadatka: [25].

Niz A je podniz niza B, ako se precrtavanjem nekih elemenata niza B može dobiti niz A. Na primer (1,3,3,5) je podniz od (1,2,3,3,4,5), a nije podniz ni od (1,5,2,3,3,4) ni od (1,2,3,4,5).

Data su dva niza: P od M i Q od N elemenata. Naći niz najveće moguće dužine koji je podniz i za P i za Q.

Rešenje: Neka je $NZP(X,Y)$ najduži zajednićki podniz nizova P_X i Q_Y (P_X sadrži prvix X elemenata niza P, dok Q_Y sadrži prvix Y elemenata niza Q). U zadatku se traži $NZP(M,N)$. Ako je $p_M = q_N$, tada $NZP(M,N) = NZP(M-1,N-1) \cup \{p_M\}$ (p_M se dopisuje na kraj niza $NZP(M-1,N-1)$), dok je za $p_M \neq q_N$, $NZP(M,N)$ jednak dužem od $NZP(M-1,N)$, $NZP(M,N-1)$.

Ova relacija omogućava da se izračunaju sve vrednosti $NZP(M,N)$ redom (po vrstama ili po kolonama), znajući da je $NZP(X,0) = NZP(0,Y) = \emptyset$ (prazan niz). Dovoljno je pamtiti dužine najdužih zajedničkih podnizova u matrici B, a $NZP(X,Y)$ se lako rekonstruiše na osnovu matrice B:

$$\begin{aligned} B(X,0) &= B(0,Y) = 0 \\ B(X,Y) &= \begin{cases} B(X-1,Y-1)+1 & \text{ako je } P(X) = Q(Y) \\ \max\{B(X,Y-1), B(X,Y-1)\} & \text{ako je } P(X) \neq Q(Y) \end{cases} \end{aligned}$$

```
program NZP;
  type matrica=array[0..120,0..120]of integer;
  var b:matrica;
      p,q:array[1..120]of integer;
      i,j,m,n:integer;
      used:boolean;
      f:text;

  Function max(a,b:integer):integer;
    begin
      if a>b then max:=a
```

```
      else max:=b;
    end;
  procedure ispis(var b:matrica;i,j:integer);
    begin
      while (b[i-1,j]=b[i,j])and(i>1) do i:=i-1;
      while (b[i,j-1]=b[i,j])and(j>1) do j:=j-1;
      if b[i,j-1]>0 then ispis(b,i,j-1);
      write(f,q[i]:3);
    end;

begin
  assign(f,'input.dat');reset(f);  readln(f,m,n);
  for i:=1 to m do read(f,p[i]);  for i:=1 to n do read(f,q[i]);
  close(f);
  for i:=1 to m do b[i,0]:=0;  for i:=1 to n do b[0,j]:=0;
  for i:=1 to n do
    begin
      used:=false;
      for j:=1 to m do
        if (q[i]=p[j])and(b[0,j]=0)and not used then
          begin
            b[i,j]:=max(b[i-1,j],b[i,j-1])+1;  b[0,j]:=1;
            used:=true;
          end
        else
          b[i,j]:=max(b[i-1,j],b[i,j-1]);
    end;
  assign(f,'output.dat');  rewrite(f);  ispis(b,n,m);   close(f);
end.
```

Ukoliko bi se tražila samo dužina NZP, mogli bismo da uštedimo prostor, tako što umesto matrice B koristimo samo dva niza, koji bi igrali ulogu vrste iz B koja se trenutno formira i predhodne vrste (uz redosled popunjavanja unazad, dovoljan je samo jedan niz). Da bismo rekonstruisali NZP, neophodna nam je cela matrica B, ili dodatno vreme za ponovna računanja izgubljenih informacija.

2.7 Najjeftinija ispravka reči

Poreklo zadatka: [25].

Dozvoljene operacije nad stringom su: umetanje slova, brisanje jednog slova, izmena slova i brisanje svih slova do kraja stringa. Svaka od ovih operacija ima zadatu cenu. Potrebno je odrediti najmanju ukupnu cenu operacija kojima se od datog stringa A dobija dati string B.

Rešenje: Zadatak se rešava vrlo slično kao problem maksimalnog zbira i problem najdužeg zajedničkog podniza, pa je zbog toga detaljnije objašnjenje izostavljeno. Neka je $C(i,j)$ minimalna cena potrebna da se od stringa A_i dobije string B_j. Tada važi sledeća rekurentna formula:

$$C(i,j) = \begin{cases} C(i-1,j-1), & a_i = b_j \\ \min\{C(i-1,j-1)+cZam, C(i-1,j)+cBris, C(i,j-1) \\ +cUmet\} \cup \{C(k,j)+CBrisDoKraja \mid k=1,\ldots,i-1\}, & a_i \neq b_j \end{cases}$$

Pritom su $cZam, cBris, cUmet, cBrisDoKraja$ redom cene zamene slova, umetanja jednog slova, brisanja jednog slova i brisanja svih slova do kraja počev od nekog slova. Startne vrednosti su $C(i,0) = \min\{i \cdot cBris, cBrisDoKraja\}$ i $C(0,j) = j \cdot cUmet$.

Implementacija:

```
program Najeftinija_ispravka_reci;
 var  cena:array[0..50,0..50] of real;
      a,b:string;
      cUmet,cBris1,cZam,cBrisDoKraja:real;
      i,j,k,m,n:integer;
 function min3(a,b,c:real):real;
     var x:real;
      begin
        x:=a;
        if x>b then x:=b;
        if x>c then x:=c;
        min3:=x;
      end;
   begin
     write('Cena umetanja=');readln(cUmet);
     write('Cena brisanja jednog znaka='); readln(cBris1);
     write('Cena zamene='); readln(cZam);
     write('Cena brisanja do kraja stringa='); readln(cBrisDoKraja);
     write('a='); readln(a); m:=length(a);
     write('b='); readln(b); n:=length(b);
      cena[0,0]:=0;
      for i:=1 to m do
       begin
         cena[i,0]:=cena[i-1,0]+cBris1;
         if cena[i,0]>cBrisDoKraja then   cena[i,0]:=cBrisDoKraja;
       end;
      for j:=1 to n do  cena[0,j]:=cena[0,j-1]+cUmet;
      for i:=1 to m do
       for j:=1 to n do
         begin
          if a[i]=b[j] then  cena[i,j]:=cena[i-1,j-1]
           else
             cena[i,j]:=min3(cena[i-1,j-1]+cZam,cena[i,j-1]+cUmet,
                             cena[i-1,j]+cBris1);
          for k:=1 to i-1 do
           if cena[i,j]>cena[k,j]+cBrisDoKraja then
              cena[i,j]:=cena[k,j]+cBrisDoKraja;
         end;
      writeln('Najeftinija prepravka kosta ' ,cena[m,n]:10:3);
 end.
```

2.8 Najbrže stepenovanje

Poreklo zadatka: [25].

Dat je prirodan broj N i promenljiva K. Koristeći operacije množenja i stepenovanja,male zagrade i promenljivu K, napisati izraz koji je jednak K^N, a u kome učetvuje minimalan broj operacija. Množenje se smatra jednom operacijom,

a računanje Q-tog stepena smatra se za $Q-I$ operacija. Prilikom ispisivanja izraza stepenovanje označiti sa dve zvezdice.

Primer: za $N=5$, potrebne su tri operacije: $K^5=(K\cdot K)^2\cdot K$.

Rešenje: Neka se u optimalnom izrazu jednakom K^N (N> 1) poslednja izvršava operacija množenja.Tada je izraz oblika $T_1 \cdot T_2$, gde su T_1 i T_2 optimalni izrazi jednaki K^P i K^{N-P} za neko P.Broj operacija u izrazu jednakom K^N je tada $B(N) = B(P)+B(N-P)+1$.

Ako se u optimalnom izrazu jednakom K^N $(N>1)$ poslednja izvršava operacija stepenovanja, izraz je oblika T^R, gde je N deljivo sa R, a T je optimalan izraz jednak $K^{N/R}$,pa važi $B(N)=B(N/R)+R-1$.

Videli smo u čemu se ogleda optimalnost podstrukture problema. Prilikom rešavanja svih podproblema redom (za X od 1 do N), optimalan izraz jednak K^X naći ćemo tako što svaku moguću operaciju isprobamo kao poslednju, a za podizraze koje treba uvrstiti u izraz, koristimo ranije izračunata rešenja. Za dobijanje prvog stepena K^I, potrebno je 0 operacija, pa imamo:

$$\begin{aligned} B(1) &= 0, \\ B(X) &= \min \begin{cases} \min\limits_{1\le P<X}\{B(P)+B(X-P)+1\} \\ \min\limits_{R\neq 1, R|X}\{B(X/R)+R-1\} \end{cases} , 1<X\le N \end{aligned}$$

Da bi se rekonstruisao izraz sa najmanje operacija, koji je jednak k^N, dovoljno je pri rešavanju svakog podproblema zapamtiti poslednju operaciju. Pored niza B, koji pamti najmanji broj operacija, pamtićemo i niz C iste dužine. Stavićemo $C(X)=P$, ako se K^X najjejftinije dobija kao proizvod optimalnih izraza jednakih K^P i K^{X-P} redom, ili $C(X)=-R$ ako K^X najjejftinije dobijamo kao R-ti stepen optimalnog izraza jednakog $K^{X/R}$.

```
program najstep;
  var B,C:array[1..1000] of integer;
      p,r,m,N:integer;

  procedure ispis(n:integer);
   begin
     if C[n]=0 then write('K')
     else if C[n]<0 then
       begin
         write('('); ispis(n div (-C[n]));  write(')**',-C[n]);
       end
     else begin
            ispis(C[n]); write('*'); ispis(n-C[n]);
          end;
   end;

  begin
    write('N= ? '); readln(N);
    B[1]:=0; C[1]:=0;
    for m:=2 to N do
      begin
        B[m]:=B[1]+B[m-1]+1;       C[m]:=1;
```

```
      for p:=2 to m-1 do
        if B[p]+B[m-p]+1<B[m] then
          begin
             B[m]:=B[p]+B[m-p]+1;    C[m]:=p;
          end;
      for r:= 2 to m div 2 do
        if (m mod r) = 0 then
           begin
             if B[m div r]+r-1 < B[m] then
               begin
                 B[m]:=B[m div r]+r-1; C[m]:=-r;
               end;
           end;
    end;
  writeln('Potreban broj operacija je ',B[N]);
  write('Izraz je: '); ispis(N); writeln;
end.
```

2.9 Red za karte

Poreklo zadatka: [25].

N ljudi stoji pred ulazom blagajne i čeka da kupi karte. $K-$tom čoveku u redu treba t_k vremena da kupi kartu $k = 1, \ldots, N$. Svaki čovek može se udružiti sa sledećim u redu. Vreme potrebno da $k-$ti i $k+1-$vi čovek kupe karte ako se udruže, iznosi p_k, $k = 1, \ldots, N-1$. Kupovina time može a i ne mora da se ubrza. Odrediti takav način udruživanja,da ukupno vreme potrebno da svih N ljudi kupi po kartu,bude minimalno. Ulazni podaci su broj N i nizovi T i P, trebalo bi ispisati redne brojeve onih ljudi koji se udružuju sa sledećim u redu.

Rešenje: Neka je niz A takav da $a_k = t_k + t_{k+1} - p_k$, tj. a_k je ušteda u vremenu ako se udruže k–ti i $k+1-$vi čovek u redu. Primetimo da ne mora biti $p_k < t_k + t_{k+1}$,pa elementi niza A mogu biti i negativni. Kako se k–ti čovek ne može udružiti i sa $k+1-$vim istovremeno, ne možemo istovremeno uštede a_{k-1} i a_k. Preciznije, potrebno je da se odredi podniz niza A, koji ima najveći zbir i u kome nema susednih elemenata, a to je zadatak koji je upravo rešen.

Zadatak se može rešiti i bez svođenja na prethodni: potrebno je formirati niz B, gde je $B(k)$ najmanje potrebno vreme (umesto najveće moguće uštede ,što je učinjeno u prethodnom rešenju) da prvih k ljudi kupi karte. Lako se dobija

$$B(0) = 0, B(I) = a_j, B(X) = \min\{B(X-1) + t_x, B(X-2) + p_{x-1}\}.$$

Nakon što se formira niz B, ispisati redne brojeve onih ljudi koji su se udružili sa sldećim u redu. To se, kao i u drugim sličnim zadacima, najlakše radi pomoću (brze) rekurzivne procedure. Ukoliko je prihvatljiv redosled unazad, može se koristiti i ciklus.

```
program Red_Za_Karte;
 var t,p,a,b:array[0..1000]of integer;
     f:text;
     n,i:integer;
```

```
procedure ispis(n:integer);
 begin
   if n>0 then
     if b[n]=b[n-1] then ispis(n-1)
     else
       begin
         ispis(n-2);    write(f,n:4);
       end;
 end;
function min(a,b:integer):integer;
 begin
   if a>b then min:=a else min:=b;
 end;
begin
 assign(f,'input.dat'); reset(f);  readln(f,n);
 for i:=1 to n do read(f,t[i]);  for i:=1 to n-1 do read(f,p[i]);
 close(f);
 for i:=1 to n-1 do a[i]:=t[i]+t[i+1]-p[i];
 b[0]:=0;   b[1]:=a[1];
 for i:=1 to n do b[i]:=min(b[i-1]+t[i], b[i-2]+p[i-1]);
 assign(f,'output.dat'); rewrite(f); ispis(n);  close(f);
end.
```

2.10 Raspoređivanje mašina na dva posla

Poreklo zadatka: [25].

Dato je N mašina koje mogu obabljati dva posla. Na mašinama postoji jedan jeftin deo (zanemarljive cene) koji se lako kvari, ali rezervnih delova trenutno nema. Ako p mašina obavlja prvi posao, dobit od toga je $D_1(p)$, i pri tome $R_1(p)$ mašina može da nastavi da radi. Slično, ako q mašina obavlja drugi posao, dobit je $D_2(q)$,i ostaje $R_2(q)$ ispravnih mašina. Potrebno je tokom M uzastopnih perioda raspoređivati mašine na ova dva posla, tako da se ostvari maksimalna dobit.

Rešenje: Pretpostavimo da je poznata optimalna raspodela mašina na poslove tokom prvih K perioda. Nije jasno kako se taj podproblem može upotrebiti u rešavanju problema za $K+1$ period. Preciznije, može se dogoditi da se u prethodnim periodima isplati angažovati mašine na drugačiji način, tako da se ostvari manja dobit, ali da ostane više ispravnih mašina za novi period proizvodnje. Tako bi se (možda) mogla nadmašiti dobit bazirana na maksimumu iz prvih K perioda i optimalnom korišćenju preostalih ispravnih mašina u novom periodu. Kako onda rešiti zadatak?

Pokušajmo pre svega da definišemo hijerarhiju problema sa optimalnom podstrukturom. Neka je dato optimalno rešenje problema. Jedini period za koji znamo broj mašina koje učestvuju u njemu je prvi period. Neka je u optimalnom rešenju u prvom periodu p mašina raspoređeno na prvi posao, a $N-p$ mašina na drugi posao. Tada je tokom narednih $M-1$ perioda preostalih $R_1(p)+R_2(N-p)$ mašina moralo biti raspoređeno takođe na optimalan način. Prema tome, veličina problema treba da se zada brojem ispravnih mašina i brojem perioda,a problem za K perioda odnosi se na poslednjih K perioda,a ne na prvih K.

Neka je maksimalna dobit od X mašina u (poslednjih) Y perioda označena sa

$B(X,Y)$. Tada je:

$$\begin{aligned} B(X,1) &= \max_{0\le p\le X}\{D_1(p)+D_2(X-p)\} \\ B(X,Y) &= \max_{o\le p\le X}\{D_1(p)+D_2(X-p)+B(R_1(p)+R_2(X-p),Y-1)\} \end{aligned}$$

U matrici C kao $C(X,Y)$ pamtimo ono p za koje se dostiže maksimum u B. Na osnovu matrice C lako se rekonstruiše raspodela mašina po periodima.

Implementaciju ovog rešenja prepuštamo čitaocu.

3 Problem ranca

Jedan od najpoznatijih problema dinamičkog programiranja jeste **problem ranca** (knapsack problem). Problem ima nekoliko poznatih varijanti. Problem ranca je klasični problem dinamičkog programiranja, ali je zbog njegove važnosti i značaja ovom problemu posvećen ceo jedan odeljak ove glave. U nastavku ćemo izložiti više varijatni ovog problema i svaku detaljno analizirati. Za neke od varijanti dajemo i implementaciju u programskom jeziku PASCAL.

3.1 Prva varijanta problema ranca

Poreklo formulacije problema: [25].

Provalnik sa rancem u koji može da stane N zapreminskih jedinica, upao je u prostoriju u kojoj se čuvaju vredni predmeti. U prostoriji ima ukupno M tipova predmeta, pri čemu je svaki tip predmeta raspoloživ u vrlo velikoj količini (više nego što može da stane u ranac). Za svaki tip predmeta poznata je njegova vrednost $V(k)$ i njegova zapremina $Z(k)$, $k = 1,\ldots,M$. Sve navedene veličine su celobrojne. Provalnik želi da napuni ranac najvrednijim sadržajem. Potrebno je odrediti predmete koje treba staviti u ranac, i njihovu zbirnu vrednost.

Da bi se sagledala suštinu problema, uočimo nekoliko situacija.

- Ranac koji je optimalno popunjen ne mora biti popunjen do vrha, ili preciznije, zbir zapremina uzetih predmeta ne mora biti jednak zapremini ranca. Bitno je da taj zbir nije veći od zapremine ranca, a da u isto vreme zbir vrednosti tih predmeta bude maksimalan. Na primer, ako je $N = 7, M = 3, V = (3,4,8), Z = (3,4,5)$, ranac možemo popuniti do vrha tako što u njega stavimo prvi i drugi predmet jer je $z_1 + z_2 = 3 + 4 = 7 = N$. Međutim, takvo popunjavanje nije optimalno, jer je njegova vrednost $v_1 + v_2 = 3 + 4 = 7$, dok stavljanjem samo trećeg predmeta u ranac, dobijamo ranac vrednijeg sadržaja $v_3 = 8$, pri čemu ranac nije popunjen do vrha ($z_3 = 5 < 7 = N$).

- Na osnovu predhodnog primera, može se steći pogrešan utisak da se do rešenja dolazi tako što se u svakom koraku odredi takvo k, među predmetima koji nisu upotrebljeni, za koje je količnik $V(k)/Z(k)$ najveći, pa se ranac puni predmetom čiji je indeks k. Ovaj pristup nije ispravan, što ćemo dokazati na jednostavnom primeru: neka je $N = 7$, $M = 3$, $V = (3,4,6)$, $Z = (3,4,5)$. Navedena

ideja (grabljivi izbor) nalaže da se odabere treći predmet, kao najvredniji po jedinici zapremine. Time bi u ranac bio stavljen samo jedan od predmeta trećeg tipa (ubuduće: predmet broj 3), a vrednost plena bi bila jednaka 6. Lako je videti da izbor po jednog predmeta 1 i 2 daje plen vrednosti 7, što je bolje (i optimalno) rešenje. Napomenimo samo da bi ideja grabljivog izbora vodila ka rešenju da ne moraju da se uzimaju celi predmeti i da je vrednost dela nekog predmeta srazmerna veličini tog dela.

Dokazaćemo, koristeći se indukcijom, da se svako celobrojno q, $0 \leq q \leq N$, može naći popunjavanje ranca kapaciteta q. Za $q = 0$ optimalno resenje je prazan ranac. Neka su poznata rešenja za sve rance kapaciteta $i < q$, i neka je $B(q)$ zbirna vrednost koja odgovara uzetim predmetima za ranac kapaciteta q, a $C(q)$ niz rednih brojeva predmeta stavljenih u ranac kapaciteta q pri optimalnom izboru. Svaki od M tipova predmeta koji može da stane u prazan ranac kapaciteta q, isprobamo kao poslednji izabrani za ranac kapaciteta q. Ostatak ranca u svakom od ovih slučajeva popunimo na optimalan način, što na osnovu induktivne hipoteze može uraditi. Najveća od svih dobijenih vrednosti ranca kapaciteta q biće optimalna. Ova činjenica sledi direktno na osnovu optimalnosti podstrukture, jer jedan predmet mora biti poslednji, a mi smo svaki isprobali kao poslednji. Prema rečenom, rešenje za ranac kapaciteta q je

$$B(q) = \max_{\substack{1 \leq i \leq M \\ Z(i) \leq q}} \{B(q - z(i)) + V(i)\}.$$

Ako je $B(q) = B(q - z(k)) + V(k)$, tada je

$$C(q) = C(q - Z(k)) \cup \{k\},$$

što označava da k-ti predmet ostvaruje maksimalnu vrednost $B(q)$. Primetimo da ne moramo pamtiti ceo skup $C(q)$ za svaki kapacitet ranca q, dovoljno je kao član niza $C(q)$ zapamtiti poslednji dodati element $x(q)$. Svi elementi se tada mogu naći redom (od poslednjeg ka prvom), i to su:

$$a = C(N),\ b = C(N - Z(a)),\ c = C(N - Z(a - Z(b)),\ \ldots$$

itd. dok se ne dobiju svi predmeti iz ranca.

Analizirajući problem, mozemo primetiti sledeće.

Teorema 3.1.1 *Ako je za optimalno popunjavanje ranca do kapaciteta q izabrani predmet sa rednim brojem i, onda predhodno izabrani predmet na optimalan način popunjava ranac kapaciteta $q - z(i)$.*

Dokaz. Tvrđenje se lako dokazuje svođenjem na kontradikciju. Označimo sa $B(q)$, $q = 0, \ldots, N$ optimalna popunjavanja ranca do kapaciteta q. Neka je

$$B(q) = \max\{B(q - z(i)) + V(i) | z_i \leq q, i = 1, \ldots, M\} = B(q - z(k)) + V(k).$$

Dokažimo da je popunjavanje do kapaciteta $q - z(k)$ optimalno. Pretpostavimo da ranac kapaciteta $q - z(k)$ može bolje da se popuni, za neko i. Neka je vrednost tog

popunjavanja jednaka $R(q - z(k)) > B(q - z(k))$. Popunimo na isti način prvih $q-z(k)$ jedinica zapremine ranca veličine N, i dodamo k-ti predmet. Time dobijamo popunjavanje celog ranca, koje je bolje od polaznog, jer je njegova vrednost

$$R(q - z(k)) + V(k) > B(q - z(k)) + V(k) = B(q).$$

To je kontradikcija, jer je po pretpostavci $B(q)$ maksimalno. □

Prema tome, **optimalno rešenje problema sadrži u sebi optimalna rešenja podproblema istog tipa, sadržanih u glavnom problemu.** Uobičajeno je da se u ovakvom slučaju kaže da problem ima optimalnu podstrukturu. Navedeno svojstvo se naziva i svojstvom Belmana, po autoru metoda dinamičkog programiranja. Ovo je ključna osobina dinamičkog programiranja. Zahvaljujući ovoj osobini, možemo doći do optimalnog rešenja problema, kombinujući optimalna rešenja podproblema.

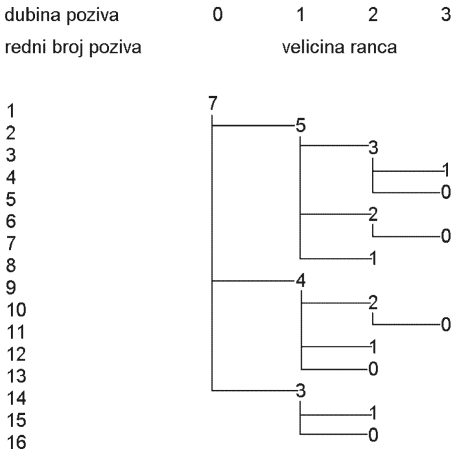
Program za prvu varijantu problema ranca u jeziku PASCAL:

```
program ranac;
  var B,C:array[0..1000] of integer;
      V,Z:array[1..10] of integer;
      N,M,q,i:integer;
  begin
   write('N = ? '); readln(N);    write('M = ? '); readln(M);
   writeln('Vrednosti i zapremine, redom? ');
   for i:=1 to M do read(V[i], Z[i]);
   B[0]:=0;   C[0]:=0;
   for q:= 1 to N do
     begin
       B[q]:=0; C[q]:=0;
       for i:=1 to M do
         if (Z[i]<=q) and (B[q-Z[i]]+V[i]>B[q]) then
           begin
             B[q]:=B[q-Z[i]]+V[i]; C[q]:=i;
           end;
     end;
   writeln('Optimalna vrednost sadrzaja u rancu je: ',B[N]);
   writeln('Redni brojevi izabranih predmeta: ');
   i:=N;
   while(C[i]>0) do
     begin
       writeln(C[i]); i:=i-Z[C[i]];
     end;
   writeln;
 end.
```

Zadatak se može resiti upotrebom rekurzivne procedure $napuni(q, B, C)$, koja za ranac kapaciteta q nalazi njegovu optimalnu vrednost B i redni broj poslednjeg dodatog predmeta C. Smatraćemo da su nizovi V i Z, kao i broj tipova predmeta M globalne veličine. Iz glavnog programa bi se pozivala procedura $napuni(N, B, C)$, a svaki poziv ove procedure može prouzrokovati M novih rekurzivnih poziva. Na primer za $N = 1000, M = 10$ i predmete zapremine od $Zmin = 5$ do $Zmax = 10$, bilo bi između $M^{N/Zmax} = 10^{100}$ i $M^{N/Zmin} = 10^{200}$ rekurzivnih poziva procedure,

i to samo na poslednjem nivou rekurzije. Kada bi svaki poziv trajao nanosekundu ($10^{-9}s$) trebalo bi vise od $10^{91}s > 10^{83}$ godina, a starost kompletne vasione procenjuje se na manje od 10^{12} godina [25]!

Umesto rekurzivno, probleme možemo rešavati i redom od $q = 1$ do $q = N$, popunjavajući pri tome nizove B i C dobijenim vrednostima. Za svako q, potrebno je M prolazaka kroz ciklus da se odredi poslednji izabrani element i najveća vrednost. To znači da je ukupan broj operacija jednak MN. U gornjem primeru to bi bilo deset hiljada ciklusa od po par računarskih koraka, što se na današnjim računarima obavlja za znatno manje od jedne sekunde. Odakle ovako drastična razlika u efikasnosti ponuđenih rešenja? Ispitajmo detaljnije na manjem primeru kako radi rekurzivni algoritam. Neka je $N = 7,\ M = 3,\ Z = (2, 3, 4)$. Vrednosti predmeta nisu od značaja za praćenje toka rekurzivnih poziva. Na sledećoj slici je prikazana hijerarhija rekurzivnih poziva procedure *napuni* u obliku drveta. Čvorovi drveta navedeni su po redosledu nastajanja, tj. po redosledu pozivanja primeraka procedure *napuni*, predstavljenih tim čvorovima. Oznake čvorova predstavljaju vrednosti argumenta q, tj. veličinu ranca koji treba popuniti.



Slika 3.1.1. hijerarhija rekurzivnih poziva procedure *napuni*.

Kao što vidimo, prilikom prolaska po drvetu rekurzivnih poziva, jedan isti podproblem (popunjavanje ranca iste velicine q susreće se više puta, i svaki put se iznova rešava. Drugim rećima, **podproblemi imaju zajedničke podprobleme, odnosno delimično se preklapaju.** Pri tome broj ponovljenih rešavanja po pravilu raste eksponencijalno sa povećanjem dimenzije polaznog problema. Preklapanje podproblema je druga osobina koja opravdava primenu dinamičkog programiranja. Ova osobina nije neophodna da bi se dinamičko programiranje primenilo, ali bez preklapanja podproblema dinamičko programiranje gubi svoju prednost u brzini u odnosu na rekurziju, jer se tada i rekurzijom svaki podproblem kreira i rešava samo jednom. Štaviše, kada nema preklapanja podproblema, u nekim problemima se događa da se rekurzivno rešenje dobije brže i/ili da se pri tome troši znatno manje memorijskog prostora.

Stroga matematička formulacija problema ranca se može naći u [7].

Problem ranca se može posmatrati kao poseban slučaj linearnog celobrojnog programiranja:

$$\begin{aligned} \max \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n, \\ \text{p.o.} \quad & a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b, \\ & x_1, x_2, \ldots, x_n \geq 0,\ x_1, x_2, ..., x_n \in Z. \end{aligned}$$

Pri tome su $a_1, a_2, \ldots, a_n$ prirodni brojevi a b ceo broj. Postoji više metoda za rešavanja problema, ranca. Ovde ćemo se detaljnije pozabaviti rešavanjem problema ranca koje je bazirano na ideji dinamičkog programiranja. U razlaganju problema na etape, ključnu ulogu igra definicija sledeće funkcije za cele brojeve $k \geq 1$ i $y \geq 0$:

$$F_k(y) = \max\{c_1x_1 + \cdots + c_kx_k | a_1x_1 + \cdots + a_kx_k \leq y, x_1, \ldots, x_k \geq 0, x_1, \ldots, x_k \in Z\}.$$

Za ovu funkciju, moguće je dokazati više rekurentnih relacija:

1.

$$F_1(y) = c_1\lfloor y/a_1 \rfloor$$

$$F_k(y) = \max\left\{c_kx_k + F_{k-1}(y - a_kx_k) | x_k \in \{0, 1, ..., \lfloor y/c_k \rfloor\} \text{ za } k \geq 2\right\}.$$

Rekurzija je posledica logičke alternative da su u optimalnom rešenju moguće vrednosti promenljive x_k upravo $0, 1, \ldots, \lfloor y/a_k \rfloor$ a posle njenog fiksiranja preostaje optimalno punjenje ranca zapremine $y - a_kx_k$ ostalim "predmetima".

2.

$$F_k(y) = \max\{F_{k-1}(y), F_k(y - a_k) + c_k\},$$

ukoliko se za $y < 0$ funkcija $F_k(y)$ dodefiniše pomoću $F_k(y) = -\infty$. Rekurzija je posledica logičke alternative: za k-tu koordinatu optimalnog rešenja važi ili $x_k = 0$ (tada je $F_k(y) = F_{k-1}(y)$) ili $x_k \geq 1$ (tada je $F_k(y) = F_k(y - a_k) + c_k$). Naravno, optimalna vrednost je uvek jednaka većoj od ovih vrednosti.

3.

$$F_n(y) = 0 \text{ za } 0 \leq y \leq \min\{a_1, \ldots, a_n\}$$

a za $y \geq \min\{a_1, \ldots, a_n\}$ je

$$F_n(y) = \max\{c_k + F_n(y - a_k) | k = 1, \ldots, n\},$$

ukoliko se dodefiniše $F_n(y) = -\infty$ za $y < 0$. Rekurzija je posledica logičke alternative da, ukoliko nula nije optimalno rešenje, bar jedna njegova koordinata, recimo x_k, je veća ili jednaka od 1 (tada je $F_n(y) = c_k + F_n(y - a_k)$).

Svaka od ovih rekurzija može da posluži za izračunavanje optimalne vrednosti $F_n(b)$ problema kao i odgovarajućeg optimalnog rešenja.

Pri korišćenju prve rekurzije, trebalo bi nepoznatu vrednost $F_n(b)$ svoditi na niže nivoe sve do poznatih vrednosti $F(y)$ a zatim se vraćati unazad. Pri tome se svi rezultati moraju pamtiti. Za određivanje optimalnog rešenja trebalo bi još zapamtiti gde se u pojedinim izračunavanjima dostiže maksimum.

Primer 3.1.1 *Primenom prve rekurzivne formule rešiti problem*

$$\begin{array}{ll} \max & 3x_1 + 4x_2 + 5x_3 + 2x_4, \\ \text{p.o.} & 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 9, \\ & x_l, x_2, x_3, x_4 \geq 0,\ x_1, x_2, x_3, x_4 \in Z. \end{array}$$

Računamo redom:

$$F_4(9) = \max\{2x_4 + F_3(9 - 5x_4) | x_4 \in 0, 1\} = \max\{F_3(9), 2 + F_3(4)\}$$

$$\begin{array}{rcl} F_3(9) & = & \max\{F_2(9), 5 + F_2(5)\} \\ F_3(4) & = & \max F_2(4), 5 + F_2(0) \\ F_2(9) & = & \max\{F_1(9), 4 + F_1(6)\} \\ F_2(5) & = & \max\{F_1(5), 4 + F_1(2)\} \\ F_2(4) & = & max\{F_1(4), 4 + F_1(1)\} \\ F_2(0) & = & 0 \\ F_1(y) & = & 3\lfloor\frac{y}{2}\rfloor. \end{array}$$

Postupkom unazad nalazimo:

$$\begin{array}{rcl} F_2(4) & = & F_1(4) == 6 \\ F_2(5) & = & 4 + F_1(2) = 7 \\ F_2(9) & = & 4 + F_1(6) = 13 \\ F_3(4) & = & F_2(4) = 6 \\ F_3(9) & = & F_2(9) = 13 \\ F_4(9) & = & F_3(9) = 13. \end{array}$$

Optimalno rešenje rekonstruišemo na sledeći način: Iz $F_4(9) = F_3(9)$ *sledi* $x_4^* = 0$. *Iz* $F_3(9) = F_2(9)$ *sledi* $x_3^* = 0$. *Iz* $F_2(9) = 4 + F_1(6)$ *sledi* $x_2^* = 1$. *Iz* $F_1(6) = 9$ *sledi* $x_1^* = 3$.

Druga rekurzija je pogodnija za kompjutersku obradu. Računanje ide unapred, redom po vrstama

$$\begin{array}{llll} F_1(1) & F_1(2) & \cdots & F_1(b) \\ F_2(1) & F2(2) & \cdots & F_2(b) \\ \vdots & & & \\ F_n(1) & F_n(2) & \cdots & F_n(b). \end{array}$$

Pri tome se pamte samo dve uzastopne vrste. Za računanje optimalnog rešenja može se uvesti prateći indeks $i_k(y)$ veličine $F_k(y)$, jednak najvećem indeksu j takvom da je j–ta promenljiva optimalnog rešenja u $F_k(y)$ pozitivna. Ukoliko je nula optimalno rešenje, ovaj indeks se definiše kao nula. Za ove indekse vredi rekurzija

$$i_k(y) = \begin{cases} i_{k-1}(y), & c_k + F_k(y) < F_{k-1}(y) \\ k, & c_k + F_k(y) \geq F_{k-1}(y). \end{cases}$$

Znajući $i_n(y)$ za $y = 0, 1, \ldots, b$, lako detektujemo optimalno rešenje iz smisla indeksa i vrednosti $i_n(b), i_n(b - a_{i_n(b)}), \ldots$.

Primer 3.1.2 *Rešiti zadatak iz Primera 3.1.1 drugom rekurzijom. Popunjavamo redom po vrstama matrice* $F_k(y)$ *i* $i_k(y)$ *za* $k = 1, 2, 3, 4$; $y = 1, 2, ..., 9$.

0	3	3	6	6	9	9	12	12
0	3	4	6	7	9	10	12	13
0	3	4	6	8	9	10	12	13
0	3	4	6	8	9	10	12	13
0	1	1	1	1	1	1	1	1
0	1	2	1	2	1	2	1	2
0	1	1	1	3	1	2	1	2
0	1	1	1	3	1	2	1	2

Primer popunjavanja: $F_2(9) = \max\{F_2(9), F_2(6) + 4\} = \max\{12, 13\} = 13$, $i_2(9) = 2$. *Rekonstrukcija optimalnog rešenja: Iz* $i_4(9) = 2$ *sledi da je u optimalnom rešenju* $x_3 = x_4 = 0$ *i* $x_2 \geq 1$. *Iz* $i_4(9-a_1) = i_4(6) = 1$ *sledi da je u optimalnom rešenju* $x_1 \geq 1$ *i* $x_2 = 1$. *Iz* $i_4(6-a_1) = i_4(4) = 1$ *sledi* $x_1 \geq 2$, *a iz* $i_4(4 - a_1) = i_4(2) = 1$, *sledi* $x_1 \geq 3$. *Zbog* $i_4(2 - a_1) = i_4(0) = 0$, *sledi* $x_1 = 3, x_2 = 1, x_3 = x_4 = 0$.

Treća rekurzija zaliteva najmanje izračunavanja. Kako u optimalnom rešenju može biti više pozitivnih koordinata, na desnoj strani formule mogu da postoje suvišni clanovi. Za dovoljno veliko b oni se mogu ispustiti.

Teorema 3.1.2 *Pretpostavimo da su promenljive u problemu ranca uređene tako da važi*

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \cdots \geq \frac{c_n}{a_n}.$$

Ako je $b > a(a_1 - 1)$, *pri čemu je* $a = \max\{a_1, \ldots, a_n\}$, *tada*

$$F_n(b) = F_n(b - a_1) + c_1.$$

Dokaz. Dovoljno je pokazati da postoji optimalno rešenje sa pozitivnom prvom koordinatom. Ako je $a_1 = 1$, optimalno rešenje linearne relaksacije, a time i problema, jednako je $x_1 = b, x_2 = \cdots = x_n = 0$, pa tvrđenje važi. Ako je $a_1 \geq 2$, uvedimo izravnavajući promenljivu x_{n+1} i posmatrajmo ekvivalentni problem

$$\begin{aligned} \min \quad & -c_1x_1 - \cdots - c_nx_n, \\ a_1x_1 + \cdots + a_nx_n + x_{n+1} \; &= \; b, \; x_1, x_2, ., x_n > 0, x_1, x_2, \ldots, x_n \in Z. \end{aligned}$$

Kako je

$$x_1 = \frac{b - a_2x_2 - \cdots - a_nx_n - x_{n+1}}{a_1},$$

njemu odgovarajući asimptotski problem sa bazičnom podmatricom $A_B = (a_1)$. izgleda

$$\min \sum_{j=2}^{n} \left(\frac{c_1}{a_1} a_j - c_j \right) x_j,$$

$$a_2x_2 + a_3x_3 + \cdots + a_nx_n + x_{n+1} = b (\text{mod } a_1),$$

$$x_2, \ldots, x_{n+1} \geq 0, x_2, ..., x_{n+1} \in Z.$$

Ovaj problem je ekvivalentan problemu najkraćeg puta na grafu koji ima a_1 čvorova. Kako najkraći put ima najviše $a_1 - 1$ grana, asimptotski problem ima optimalno rešenje $(x_2^*, \ldots, x_{n+1}^*)$ za koje važi $x_2^* + \cdots + x_{n+1}^* \leq a_1 - 1$. Za $b > a(a_1 - 1)$ važi

$$x_1^* = \frac{b - \sum_{j=2}^{n} a_j x_j^* - x_{n+1}^*}{a_1} \geq \frac{b - a\sum_{j=2}^{n+1} x_j}{a_1} \geq \frac{b - a(a_1 - 1)}{a_1} > 0,$$

pa je $(x_1^*, x_1^*, \ldots, x_n^*)$ optimalno rešenje polaznog problema sa pozitivnom (i celobrojnom) prvom koordinatom. □

Primer 3.1.3 *Primenimo treću rekurentnu formulu za rešavanje problema iz Primera 3.1.1. Primetimo da su promenljive uređene tako da je* $\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \cdots \geq \frac{c_n}{a_n}$. *i da je* $a = 5$, $a_1 = 2$. *Saglasno Teoremi 3.1.2, za* $y > 5 \cdot 1$ *vredi* $F(y) = F(y - a_1) + c_1$. *Pri tome je* $F(y) = F_4(y)$. *Otuda je:*

$$\begin{aligned}
F(9) &= F(7) + 3;\\
F(7) &= F(5) + 3;\\
F(5) &= \max\{F(3) + 3, F(1) + 4, F(1) + 5, F(0) + 2\};\\
F(3) &= \max\{F(1) + 3, F(0) + 4, -\infty, \infty, \};\\
F(1) &= 0\\
F(0) &= 0.
\end{aligned}$$

Hodom unazad nalazimo redom $F(3) = F(0) + 4 = 4$, $F(5) = F(3) + 3 = 7$, $F(7) = 10, F(9) = 13$. *Optimalno rešenje lako rekonstruišemo: Iz* $F(9) = F(7) + 3$ *sledi* $x_1 \geq 1$. *Iz* $F(7) = F(5) + 3$ *sledi da je za dostizanje optimalne vrednosti* $F(7)$ *ponovo* $x_1 \geq 1$, *dakle* $x_1 \geq 2$. *Iz* $F(5) = F(3) + 3$ *sledi da je za dostizanje* $F(3)$ *opet* $x_1 \geq 1$. *Dakle, ukupno* $x_1 \geq 3$. *Iz* $F(3) = F(0) + 4 > F(1) + 3$, *sledi da je za dostizanje* $F(3)$, $x_1 = 0$ *i* $x_2 = 1$. *Optimalno rešenje je* $x_1 = 3, x_2 = 1, x_3 = 0, x_4 = 0$.

3.2 Druge varijante problema ranca

Problem ranca se može postaviti i tako da se za svaki predmet zada broj raspoloživih primeraka, ili tako da je od svake vrste predmeta na raspolaganju tačno po jedan primerak. Sledeće razmatranje problema ranca za slučaj kada se svaki predmet može uzeti najviše jednom je uzeto iz [25].

Neka je poznato optimalno popunjavanje ranca veličine X. Ako u tom popunjavanju ne učestvuje Y-ti predmet, onda je isto popunjavanje optimalno i za ranac veličine X i prvih $Y - 1$ predmeta. Ako u optimalnom popunjavanju učestvuje i Y-ti predmet, onda ostali predmeti iz ranca čine optimalno popunjavanje za ranac veličine $X - Z(Y)$ i prvih $Y - 1$ predmeta. Zaključujemo da problem i dalje ima optimalnu podstrukturu, ali se sada veličina problema zadaje sa dva parametra. Prvi parametar je kapacitet ranca a drugi je broj predmeta. Označimo sa $B(X, Y)$ najveću vrednost ranca kapaciteta X, popunjavanog nekim od prvih Y predmeta. Tada važe sledeće rekurentne relacije:

$$\begin{aligned}
B(0,0) &= 0\\
B(X,Y) &= \begin{cases} B(X, Y-1), & Z(Y) > X,\\ \max\{B(X, Y-1), B(X - Z(Y), Y-1) + V(Y)\}, & Z(Y) \leq X. \end{cases}
\end{aligned}$$

Prema tome, podproblemi se mogu rešavati sledećim redom: najpre za sve rance i jedan predmet, pa za sve rance i dva predmeta, itd. Kada se, u poslednjoj instanci, reši problem $B(N, M)$, dobija se rešenje polaznog problema.

I ovde je, kao i u predhodnoj varijanti, radi rekonstrukcije rešenja dovoljan dodatni niz C, gde će se u $C(X)$ pamtiti indeks poslednjeg uzetog predmeta pri optimalnom popunjavanju ranca kapaciteta X.

Pri popunjavanju matrice B po kolonama koriste se samo vrednosti iz predhodne kolone (to jest iz kolone $Y - 1$). Zahvaljujući tome, možemo umesto matrice B sa M kolona, koristiti matricu sa samo dve kolone, što je značajna ušteda memorijskog prostora, koja može uticati na primenljivost postupka. Iz relacije po kojoj se računa $B(X, Y)$, može se uočiti da svi potrebni elementi predhodne kolone imaju redni broj vrste manji ili jednak X. Stoga pri popunjavanju Y-te kolone matrice B unazad (od poslednje vrste ka prvoj), možemo sve operacije izvesti u istoj koloni, pa je za čuvanje potrebnih podataka dovoljan niz B [25].

```
program ranac11;
   uses wincrt;
   var b,c:array[0..1000] of longint;
       v,z:array[1..10] of integer;
       N,M,x,y:integer;
   begin
     write('N=? '); readln(N); write('M=? '); readln(M);
     for y:=1 to M do readln(v[y],z[y]);
     for x:=0 to N do
       begin b[x]:=-1; c[x]:=0;  end;
     b[0]:=0;
     for y:=1 to M do
       for x:=N downto 1 do
         if (z[y]<=x) and (b[x-z[y]]>-1) and (b[x]<v[y]+b[x-z[y]]) then
              begin
                b[x]:=v[y]+b[x-z[y]];  c[x]:=y;
              end;

     writeln('Optimalna vrednost sadrzaja ranca je ',b[N]);
     writeln('Izabrani su predmeti: ');
     x:=N;
     while c[x]>0 do
       begin
         write(c[x],'  '); x:=x-z[c[x]];
       end;
   end.
```

Efikasniji algoritam je implementiran u sledećem kôdu:

```
program Knapsack_Problem; var  v, z: array [1..1000] of Integer;
     s: array [1..1000] of Boolean;
     d, p: array [0..12000] of Integer;
     n, m, Sol, ds: Integer;

  procedure Input;
  var f: Text;
      i: Integer;
```

```
  begin
    Assign (f, 'knapsack.in');  Reset(f);  Read(f, n, m);
    for i := 1 to n do Read(f,z[i],v[i]);
    Close (f);
  end;

  procedure Solve;
  var  i, j, x: Integer;
  begin
    FillChar(s, SizeOf(s), False);
    ds:= 0;
    repeat
       for i:= 1 to m do
         begin
             d[i]:=-1;   p[i]:=-1;
         end;
       d [0]:=0;       p[0]:=0;
       for i := 1 to n do
        if (not s [i]) then
           for j:= m downto z[i] do
              if(d[j-z[i]]<>-1) and (d[j-z[i]]+v[i]>d[j]) then
                 begin
                    d [j] := d[j-z[i]]+v[i];p [j] := i;
                 end;
        if (ds = 0) then
           begin
             x := m;
             for i := 0 to m do
                if(d[x]<d[i]) then x := i;
             m := x;
             Sol := d [m];
           end;
        x := p [m];
        while (m > 0) and (not s[p[m]]) and (p[m] <= x) do
          begin
            s[p[m]]:= True;   Inc(ds);  x := p[m];   m := m-z[p[m]];
         end;
    until (m = 0);
  end;

  procedure Output;
  var  f: Text;    i: Integer;
  begin
    Assign (f, 'knapsack.out');  ReWrite (f);  WriteLn(f, Sol);
    for i:= 1 to n do if s[i] then  WriteLn(f, i);
    Close(f);
  end;

begin
  Input; Solve;  Output;
end.
```

Pri rešavanju ove varijante problema (svaki predmet najviše jednom) treba imati na umu još jednu važnu činjenicu: rešavanje problema dinamičkog programiranja se isplati jedino ako je broj predmeta M dovoljno veliki, a kapacitet ranca N relativno mali, tako da se NM operacija (koliko je pribljižno potrebno za ovaj način rešavanja)

izvrši brže nego nalaženje svih 2^N mogućih podnizova i njihovih zbirova vrednosti, odnosno zapremina. Recimo za $N = 10000, M = 100$, dinamičkim programiranjem rešenje dobijamo bez čekanja, dok 2^{100} podnizova prakticno ne možemo formirati. Međutim, u slučaju malog M i velikog N bolje je isprobati sve podnizove [25].

U nastavku je dat program za problem ranca u kome se svaki predmet uzima određeni broj puta. Promenljive imaju sledeći smisao:

n - broj elemenata,
m - zapremina ranca,
$najbolja$ - zapremina pri kojoj se uzima najveća vrednost,
$zapremina[i]$ - zapremine,
$cena[i]$ - cene predmeta,
$broj[i]$ - broj predmeta tipa i,
$ubacen[i]$ ako je indeks predmeta koji je ubačen da bi se dobila zapremina i,
$ubacen[i] = -1$ ukoliko još nismo dobili zapreminu i,
$ostalo[i]$ - koliko je još predmeta tipa onog koji je ubačen ostalo,
$best[i]$ - najbolja cena koja može da se dobije za zapreminu i,
$uzeto[i]$ - koliko je lopov uzeo stvari i-te vrste.

```
var max,i,j,l,n,m,najbolja:integer;
    broj,zapremina,cena,best,ubacen,ostalo,uzeto:array[0..1000]of integer;
    f:text;
begin
   assign(f,'input.txt');reset(f);  read(f,n,m);
   for i:=1 to n do read(f,zapremina[i],cena[i],broj[i]);
   close(f);
   for i:=1 to m do ubacen[i]:=-1;
   fillchar(best,sizeof(best),0);   {popunjava svuda 0}
   ubacen[0]:=0;
   for i:=1 to n do                {i prolazi kroz predmete}
      for j:=0 to m do            {j prolazi kroz zapremine}
         if (ubacen[j]<>-1)and(j+zapremina[i]<=m) then
                     {ako je vec dobijena zapremina j}
            if cena[i]+best[j]>best[j+zapremina[i]] then
                     {ako je povoljno da dodajemo i}
               if (ubacen[j]<>i)or(ostalo[j]>0) then
                  begin
                     {ako moze jos jedan}
                     best[j+zapremina[i]]:=best[j]+cena[i];
                     ubacen[j+zapremina[i]]:=i;  {ubacuje i-ti predmet}
                     if ubacen[j]=i then
                        ostalo[j+zapremina[i]]:=ostalo[j]-1
                     else      {koliko je predmeta i-te vrste ostalo}
                        ostalo[j+zapremina[i]]:=broj[i]-1;
                  end;
   max:=0;      najbolja:=0;
   for i:=1 to m do
     if best[i]>max then
         begin
           max:=best[i];  najbolja:=i;  {nalazi se najbolja zapremina}
         end;
   l:=najbolja;
   fillchar(uzeto,sizeof(uzeto),0);
```

```
    while l>0 do begin
       inc(uzeto[ubacen[l]]);
       l:=l-zapremina[ubacen[l]];          {sada rekonstruise niz}
    end;
    assign(f,'output.txt');rewrite(f);
    writeln(f,'Lopov puni zapreminu ', najbolja,' pri cemu odnosi vrednost ',max);
    writeln(f,'tako sto uzima sledece elemente:');
    for i:=1 to n do
        if uzeto[i]>0 then
            writeln(f,'element broj ',i,' ',uzeto[i],' put(a)');
    close(f);
end.
```

Pomenimo poznatije primere primene varijante problema ranca u kojoj se svaki predmet može uzeti najviše jedanput.

Primer 3.2.1 Poreklo zadatka: [25].

Iz datog niza prirodnih brojeva A, izdvojiti podniz čiji je zbir elemenata dati broj N, ili ispisati poruku da takav niz ne postoji.

Uputstvo. Označimo sa M broj elemenata niza A, a sa S njihovu sumu. Možemo smatrati da su nizom A zadati takvi predmeti, kojima je $v_i = z_i = a_i, i = 1, \ldots, M$. Sada je jasno da se problem svodi na optimalno popunjavanje ranca kapaciteta N, pri čemu rešenja ima ako i samo ako je vrednost optimalnog sadržaja ranca jednaka njegovoj zapremini M, to jest ako i samo ako je ranac napunjen do vrha.

```
program ranac3;
  var b,c:array[0..1000] of longint;
      a:array[1..100] of integer;
      N,M,x,y:integer;
  begin
    write('N=? '); readln(N); write('M=? '); readln(M);
    for y:=1 to M do readln(a[y]);
    for x:=0 to N do
      begin b[x]:=0; c[x]:=0;  end;
    for y:=1 to M do
      for x:=N downto 1 do
        if (a[y]<=x) and (b[x] < a[y]+b[x-a[y]]) then
             begin
               b[x]:=a[y]+b[x-a[y]];  c[x]:=y;
             end;

    if b[N]=N then
      begin
        writeln('Trazeni podniz je ');
        x:=N;
        while c[x]>0 do
          begin
             write(a[c[x]],'  '); x:=x-a[c[x]];
          end;
      end
    else writeln('Ne postoji takav podniz');
  end.
```

Primer 3.2.2 Poreklo zadatka: [25].

Brojeve iz datog niza prirodnih brojeva A podeliti u dve grupe, tako da je razlika zbirova elemenata u pojedinim grupama minimalna.

U ovom primeru niz A ponovo ima ulogu i niza Z i niza V. Neka S i N imaju ista značenja kao i u predhodnom primeru. Rešenje problema se sastoji u sledećem: predmete koje čine optimalno popunjavanje ranca kapaciteta $S/2$ (deljenje je celobrojno) trebalo bi svrstati u jednu, a sve ostale predmete u drugu grupu. Ako je ranac popunjen do vrha, grupa će za parno S imati jednake zbirove. U protivnom prva grupa ima manji zbir, adruga veći, ali ti zbirovi su najbliži vrednosti $S/2$, a time i jedan drugom.

```
Program TegoviDin;
  var b,c:array[0..4000] of longint;
      t:array[1..50] of longint;
      q,n,y,ind,max:longint;
      s,min:real;
  begin
     write('Unesite broj tegova: ');read(n);
     s:=0;max:=0;
     for q:=1 to n do
       begin
         write('unesite ',q,'. teg: ');   read(t[q]);
         s:=s+t[q]; max:=max+t[q];
       end;
     min:=s; s:=s/2;
     for q:= 0 to (max div 2) do
       begin
         b[q]:=0; c[nq]:=0;
       end;
     for y:=1 to n do
       for q:=(max div 2) downto 1 do
         if t[y]<=q then
           if b[q]< t[y]+b[q-t[y]] then
             begin
               b[q]:=t[y]+b[q-t[y]];   c[q]:=y;
             end;
     for q:=1 to (max div 2) do
       if abs(b[q]-s)<min then
         begin
           min:=abs(b[q]-s);    ind:=q;
          end;
     writeln('jedan tas: ',b[ind]:2,', drugi tas: ',max-b[ind]);
     q:=ind;
     writeln('redni brojevi tegova na jednom od tasova:');
     while c[q]>0 do
       begin
         write(c[q]:3);      q:=q - t[c[q]];
       end;
  end.
```

Vrednost optimalno popunjenog tasa (ranca) je uvek u opsegu od 1 do (`max div 2`). To važi zato što po popunjavanju jednog tasa drugi tas ima tačno utvrđenu vrednost koja se sigurno može dobiti kombinacijom datih tegova, kao i vrednost drugog tasa. Neformalni dokaz se sastoji u sledećem. Neka je q vrednost za koju su tasovi u ravnoteži (ako je `max` neparno, tasovi su u najboljoj ravnotezi kada se njihove vrednosti razlikuju za jedan, ukoliko je `max` parno, q= `max div 2`. Vrednost `max` predstavlja zbir vrednosti svih tegova. Pretpostavimo da je optimalno popunjen tas tezine x. Tada drugi tas ima vrednost $max - x$. Pretpostavimo dalje da je

vrednost optimalno popunjenog tasa

$$x \leq max\ div\ 2.$$

Ukoliko bi postojalo bolje rešenje za koje je

$$max\ div\ 2 - x > |max\ div\ 2 - a|,$$

gde je a bolje popunjen tas i $a >$*max div 2* moralo bi da vazi i da je drugi tas popunjen vrednošću *max*$-a <$ *max div 2*, pa bi se ova vrednost sigurno našla kao optimalno rešenje. Kada su i x i a manja ili jednaka `max div 2`, dokaz je trivijalan.

Primer 3.2.3 Poreklo zadatka: [25].

Dat je izraz $a_1 - a_2 - ... - a_n$, gde su svi brojevi u nizu a celi. Postaviti zagrade u ovaj izraz, tako da vrednost izraza bude jednaka datom broju X. **Uputstvo:** Problem se ekvivalentno može postaviti ovako: brojeve iz datog niza celih brojeva A, podeliti u dve grupe, tako da a_1 obavezno pripadne prvoj, a a_2 drugoj grupi, i da razlika zbirova elemenata prve i druge grupe bude jednaka X. Ova formulacija je ekvivalentna, jer zagrade uvek mogu da se postave tako da ispred brojeva iz prve grupe ima paran broj minusa, a ispred brojeva iz druge grupe neparan broj minusa koji se na te brojeve odnose. Ovaj postupak postavljanja zagrada je verovatno lakše izvesti, nego precizno opisati, pa prepuštamo čitaocu da nakon formiranja dveju grupa brojeva zagrade postavi sam.

Neka su S_1 i S_2 sume brojeva prve i druge grupe, a S suma svih N brojeva, tako da je $X = S_1 - S_2 = 2S_1 - S$. Rešenje preformulisanog zadatka se sada svodi na izbor nekih od brojeva $a_3, a_4, ..., a_n$, (pazite: a_1 i a_2 su izostavljeni), tako da je zbir izabranih brojeva jednak $M = (X + S)/2 - a_1$, a taj zadatak je već razmatran.

U problemima koji se svode na problem ranca, varijanta "svaki predmet najviše jednom", do rešenja se može doći na potpuno isti načini u slučaju da su elementi niza celi brojevi. Uslov da su elementi niza pozitivni dat je samo zato što u protivnom interpretacija gubi fizički smisao: morale bi se uvesti negativne zapremine i negativne vrednosti. Osobina niza koja je suštinski bitna u prethodna tri primera je da su vrednosti elemenata niza, kao i njihov ukupan broj celobrojne veličine čije vrednosti nisu velike. Za niz od N celih brojeva koji su svi po apsolutnoj vrednosti manji od G, sume i/ili razlike elemenata svih podnizova (svaki element može da se uzme direktno, sa promenjenim predznakom ili da se izostavi), mogu da imaju manje od od $2NG$ različitih vrednosti, bilo pozitivnih bilo negativnih. Tipično, za $N = G = 100$, dinamičkim programiranjem se lako ispituje za svaki od mogućih 20000 brojeva, da li se može pojaviti kao suma ili razlika nekih elemenata niza, pozitivnih ili negativnih. Ovakav postupak rešavanja se ne bi mogao primeniti u slučaju da mogućih kandidata za zbir (ili razliku) nekih (ili svih) elemenata niza ima za nekoliko redova veličine više, a to se dešava ako je ograničenje elemenata niza mnogo veće, ili ako se dopuštaju realni brojevi. Tada bi morali da se isprobaju svi podnizovi, što znači da efikasnog postupka u tom slučaju i nema [25].

Sumirajmo na kraju šta je sve bilo potrebno da bismo svaku od varijanti problema ranaca (ili bilo kog drugog problema) rešili dinamičkim programiranjem [25]:

- Analiziramo strukturu optimalnog rešenja. Uočimo da su izvesni delovi jednog optimalnog rešenja problema upravo optimalna rešenja manjih problema istog tipa. Odredimo parametre koji zadaju veličinu problema i podproblema.

- Vrednost optimalnog rešenja zadamo rekurentno, tj. koristeći vrednosti optimalnih rešenja nekih manjih problema.

- Nađemo efikasan način da na osnovu vrednosti optimalnih rešenja podproblema dobijemo vrednost optimalnog rešenja problema (tipično koristeći jedan ciklus, ponekad samo par naredbi).

- Nađemo i tabeliramo vrednosti rešenja za najjednostavnije podprobleme, a zatim kombinujući vrednosti rešenja manjih podproblema nalazimo vrednost rešenja većih podproblema, sve do rešenja samog problema. Pri tome tabeliramo vrednosti rešenja i delimične informacije o načinu dobijanja rešenja za sve podprobleme.

- Na osnovu tabelarnih informacija konstruišemo traženo optimalno rešenje.

3.3 Različite varijante problema ranca

U disertaciji [18] razmatra se familija problema kombinatotne optimizacije koji su vezani za problem ranca. Svi ti problemi su NP-hard, i razmatraju se egzaktni algoritmi koji imaju razumno vreme za rešavanje problema. Slično ponašanje ima simpleks metod, koji uprkos eksponencijalnoj složenosti u najlošijem slučaju pokazuje razumno vreme za rešavanje svih realnih problema.

Problem ranca zahteva podskup datih artikala koji se odabiraju tako da se maksimizira ciljna funkcija bez prekoračenja kapaciteta ranca (ranaca). Najpoznatiji slučajevi problema ranca su sledeći.

- U $0-1$*problemu ranca* ($0-1$ knapsack problem) svaki artikal može biti odabran najviše jedanput.

- U *ograničenom problemu ranca* (bounded knapsack problem), postoji ograničena količina za svaki tip artikla.

- Problem ranca sa višestrukim izborom (multiple-choice knapsack problem) se pojavljuje kada se artikli mogu odabirati iz disjunktnih klasa.

- Kada se nekoliko ranaca može istovremeno, dobija se *višestruki problem ranca* (multiple knapsack problem).

- Najopštiji oblik problema ranca je *višestruko ograničen problem ranca* (multiconstrained knapsack problem) koji je u suštini opšti oblik celobrojnog programiranja sa pozitivnim koeficijentima.

U matematičkim modelima koji slede dati su neki artikli sa profitom p_j i težinama w_j koji se pakuju u jednom ili više ranaca kapaciteta c. Pretpostavlja se da su koeficijenti p_j, w_j, c pozitivni celi brojevi.

0− problem ranca je problem izbora podskupa od n artikala tako da je odgovarajući profit maksimalan.

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} p_j x_j \\ \text{p.o.} \quad & \sum_{j=1}^{n} w_j x_j \leq c \\ & x_j \in \{0,1\},\ j = 1,\ldots,n, \end{aligned}$$

gde je x_j binarna promenljiva koja uzima vrednost 1 ako se artikal j uključuje u ranac, a inače 0.

Ukoliko je na raspolaganju ograničen iznos m_j od svakog tipa j artikala, ograničeni problem ranca ima model

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} p_j x_j \\ \text{p.o.} \quad & \sum_{j=1}^{n} w_j x_j \leq c \\ & x_j \in \{0,1,\ldots,m_j\},\ j = 1,\ldots,n, \end{aligned}$$

Neograničeni problem ranca (unbounded knapsack problem) je generalizacija ograničenog problema ranca u kome je broj artikala svakog tpa neograničen:

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} p_j x_j \\ \text{p.o.} \quad & \sum_{j=1}^{n} w_j x_j \leq c \\ & x_j \geq 0 \text{ ceo broj},\ j = 1,\ldots,n, \end{aligned}$$

Suštinski, svaka promenljiva x_j u neograničenom problemu ranca je ograničena kapacitetom c, jer je težina svakog artikla najmanje 1.

Sledeća generalizacija $0-1$ problema ranca je mogućnost izbora tačno jednog artikla tipa j iz svake od k klasa N_i, $i = 1,\ldots,k$ tako da se profit maksimizira. Na

taj način se dobija problem ranca sa višestrukim izborom:

$$\begin{aligned}
\max \quad & \sum_{i=1}^{k} \sum_{j \in N_i} p_{ij} x_{ij} \\
\text{p.o.} \quad & \sum_{i=1}^{k} \sum_{j \in N_i} w_{ij} x_{ij} \leq c \\
& \sum_{j \in N_i} x_{ij} = 1, \ i = 1, \ldots, k, \\
& x_{ij} \in \{0, 1\}, \ i = 1, \ldots, k, \ j \in N_i.
\end{aligned}$$

U ovom modelu binarna promenljiva $x_{ij} = 1$ označava da je artikal tipa j odabran iz klase i. Ograničenje $\sum_{j \in N_i} x_{ij} = 1$, $i = 1, \ldots, k$ osigurava da se odabira tačno jedan artikal iz svake klase.

Ukoliko profit p_j postane jednak težini w_j za svaki artikal u $0 - 1$ problemu ranca, dobija se *Subset–sum Problem*:

$$\begin{aligned}
\max \quad & \sum_{j=1}^{n} w_j x_j \\
\text{p.o.} \quad & \sum_{j=1}^{n} w_j x_j \leq c \\
& x_j \in \{0, 1\}, \ j = 1, \ldots, n.
\end{aligned}$$

Ime problema označava da se problem sastoji u izboru nekog podskupa vrednosti $w_1, \ldots, w_n$ tako da je je suma što veća i da ne prevazilazi c.

Sada zamislite kasira koji želi da dobije iznos novca c koristeći najmanji mogući broj novčanica $w_1, \ldots, w_n$. Takav problem se naziva *Change–making Problem*:

$$\begin{aligned}
\min \quad & \sum_{j=1}^{n} x_j \\
\text{p.o.} \quad & \sum_{j=1}^{n} w_j x_j = c \\
& x_j \geq 0 \text{ ceo broj}, \ j = 1, \ldots, n.
\end{aligned}$$

U ovom modelu w_j predstavlja vrednost novčanice tipa j koja se koristi sa x_j jedinica.

Ako se odabira n predmeta i pakuje u m ranaca (moguće) različitih kapaciteta

c_i tako da da se dobije najveći mogući profit, dobija se višestruki problem ranca.

$$\begin{aligned}
\max \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} p_j x_{ij} \\
\text{p.o.} \quad & \sum_{j=1}^{n} w_j x_{ij} \leq c_i,\ i = 1, \ldots, m \\
& \sum_{i=1}^{m} x_{ij} \leq 1,\ j = 1, \ldots, n, \\
& x_{ij} \in \{0, 1\},\ i = 1, \ldots, m,\ j = 1, \ldots, n.
\end{aligned}$$

U ovom slučaju $x_{ij} = 1$ označava da artikal tipa j bi trebalo da bude pakovan u ranac i, dok ograničenje $\sum_{j=1}^{n} w_j x_{ij} \leq c_i$ osigurava da je kapacitativno ograničenje ispunjeno. Ograničenje $\sum_{i=1}^{m} x_{ij} \leq 1$ osigurava da je svaki artikal odabran najviše jedanput.

Vrlo koristan model je *problem binarnog pakovanja* (Bin–packing Problem) u kome u kome bi trebalo da se svaki od n artikala pakuje u kutije jednakih veličina, tako da je broj upotrebljenih kutija minimalan:

$$\begin{aligned}
\max \quad & \sum_{i=1}^{n} y_i \\
\text{p.o.} \quad & \sum_{j=1}^{n} w_j x_{ij} \leq c y_i,\ i = 1, \ldots, n \\
& \sum_{i=1}^{n} x_{ij} = 1,\ j = 1, \ldots, n, \\
& x_{ij} \in \{0, 1\},\ i = 1, \ldots, m,\ j = 1, \ldots, n \\
& y_i \in \{0, 1\},\ i = 1, \ldots, n.
\end{aligned}$$

U ovom modelu y_i da li se kutija tipa i koristi, dok x_{ij} da artikal tipa j bi trebalo da se pakuje u kutiju tipa i. Ograničenje $\sum_{i=1}^{n} x_{ij} = 1$ obezbeđuje da se svaki artikal pakuje tačno jednom, dok nejednakost $\sum_{j=1}^{n} w_j x_{ij} \leq c y_i$ osigurava da se kapacitativna ograničenja ispunjavaju za sve upotrebljene kutije.

Najopštiji oblik problema ranca jeste višestruko ograničen problem ranca. On je u suštini opšti celobrojni problem u kome su svi koeficijenti p_j, w_{ij}, c_i nenegativni

celi brojevi.

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} p_j x_j \\ \text{p.o.} \quad & \sum_{j=1}^{n} w_j x_{ij} \leq c y_i, \ i = 1, \ldots, n \\ & \sum_{j=1}^{n} w_{ij} x_j \leq c_i, \ i = 1, \ldots, m, \\ & x_j \geq 0 \text{ ceo broj }, \ j = 1, \ldots, n. \end{aligned}$$

4 Ne baš klasični problemi dinamičkog programiranja

U ovom odeljku izložićemo veći broj primera problema koji se veoma efikasno rešavaju primenom tehnike DP. Problemi su uglavnom formulisani u obliku zadataka. Za svaki problem data je formulacija, kratko uputstvo za rešavanje, kao i detaljno rešenje. Rešenja su detaljno obrazložena i data je analiza složenosti kao i memorijskih zahteva za svako rešenje. Za pojedina rešenja data je i implementacija u programskom jeziku PASCAL. Savetujemo čitaocu da pre nego što pogleda rešenje svakog zadatka, najpre prouči formulaciju i pokuša da sam konstruiše rešenje primenom tehnike DP. Neki od zadataka su vrlo slični nekom od klasičnih problema DP (odeljak 2) što će čitaoc najverovatnije bez problema uočiti. Tek nakon što iscrpe sve svoje ideje čitaoc se savetuje da najpre pročita uputstvo i ponovo razmisli o problemu a tek onda da pogleda rešenje. To je najbolji način da se ovlada ovom zaista korisnom tehnikom za rešavanje problema.

Za neke od zadataka namerno nije data implementacija već se čitaoc upućuje da sam implementira izloženo rešenje. Tako će čitaoc biti u mogućnosti da proveri da li je zaista razumeo u potpunosti izloženo rešenje. Ovo pravilo je generalno i važi za sve tehnike, metode i algoritme u programiranju.

4.1 Z-Cifre

Poreklo zadatka: Z-trening [28].

Koliko ima $2N$-tocifrenih brojeva ($1 \leq N \leq 500$) tako da je zbir prvih N cifara jednak proizvodu poslednjih N cifara.

Uputstvo: Primetimo da je najveći mogući zbir prvih N cifara posmatranih brojeva jednak $9N \leq 4500$. Prema tome, dovoljno je odrediti koliko ima N-tocifrenih brojeva čiji je zbir, odnosno proizvod jednak i (označimo ih sa $s(i)$ odnosno $p(i)$). Sada je rešenje problema $s(1)p(1) + \ldots + s(4500)p(4500)$. Brojeve $v(i)$ odnosno $p(i)$ određujemo primenom rekurentne formule za dvodimenzionalne nizove $s(i, N)$ i $p(i, N)$ koji označavaju koliko ima N-tocifrenih brojeva čiji je zbir odnosno proizvod jednak i.

Rešenje: Neka je $x = \overline{x_1, \ldots, x_N, x_{N+1}, \ldots, x_{2N}}$ proizvoljan broj koji zadovoljava uslove date u formulaciji problema i neka su $x^{(1)} = \overline{x_1 \cdots x_N}$ i $x^{(2)} = \overline{x_{N+1} \cdots x_{2N}}$ brojevi sastavljeni od prvih N odnosno poslednjih N cifara broja x. Očigledno da broj x na jedinstven način određuje brojeve $x^{(1)}$ i $x^{(2)}$. Pritom, broj $x^{(1)}$ mora imati tačno N cifara (prva cifra brojeva $x^{(1)}$ i x je x_1 i ona mora biti različita od nule) dok $x^{(2)}$ mora imati najviše N cifara, tj. može počinjati i sa nulom [1]. Primetimo da važi i obratno, da svaka dva N-tocifrena broja $x^{(1)}$ i $x^{(2)}$ takvi da je zbir cifara prvog jednak proizvodu cifara drugog broja jednoznačno određuju broj x.

Označimo sa $s(i)$, odnosno $p(i)$ broj N-tocifrenih prirodnih brojeva čiji je zbir odnosno proizvod jednak i. Imamo da je $s(i)$ odnosno $p(i)$ broj načina da se odredi broj $x^{(1)}$ odnosno $x^{(2)}$. Prema tome, rešenje problema dato je izrazom: $\sum_i s(i)p(i)$. Sada se pitanje granica u predhodnoj sumi. Donja granica je očigledno 1, dok je gornja određena maksimalnom mogućom vrednošću zbira odnosno proizvoda N-tocifrenog broja. Ove vrednosti su $9N \leq 9 \cdot 500 \leq 4500$ kao i 9^N. Očigledno, prva granica je manja, pa nju možemo uzeti kao gornju granicu sume.

Preostaje nam samo još da odredimo $s(i)$ i $p(i)$ za svako $i = 1, \ldots, 4500$. Ovaj problem je veoma sličan problemu ranca. Neka je $s(i, j)$ odnosno $p(i, j)$ broj j-tocifrenih brojeva takvih da je zbir odnosno proizvod cifara jednak i. Neka je $y = \overline{y_1 \cdots y_{j-1} y_j}$ jedan takav broj. Očigledno je zbir cifara broja y (označimo ga sa $S(y)$) jednak: $S(y) = S(y') + y_j$ gde je $y' = \overline{y_1 \cdots y_{j-1}}$. Prema tome, ukoliko odaberemo zadnju cifru y_j, broj y' možemo izabrati na $s(i - y_j, j - 1)$ načina. Naravno, mora da važi da je $y_j \leq i$, kao i $y_j \leq 9$ (y_j je cifra). Prema tome, važi sledeća rekurentna formula:

$$s(i, j) = \sum_{1 \leq y_j \leq \min\{9, i\}} s(i - y_j, j - 1) \tag{4.1.1}$$

Na sličan način se dobija i odgovarajuća rekurentna formula za niz $p(i, j)$:

$$p(i, j) = \sum_{y_j \mid i, 1 \leq y_j \leq 9} p\left(\frac{i}{y_j}, j - 1\right) \tag{4.1.2}$$

Startne vrednosti za $s(i, j)$ i $p(i, j)$ su:

$$s(i, 1) = p(i, 1) = \begin{cases} 1, & 1 \leq i \leq 9 \\ 0, & \text{u suprotnom} \end{cases}$$

Vrednosti $s(i, j)$ i $p(i, j)$ računamo za svako $i = 1, \ldots, 9N$ i $j = 2, \ldots, N$ primenom rekurentnih formula (4.1.1) i (4.1.2). Na kraju uzimamo vrednosti $s(i) = s(i, N)$ i $p(i) = p(i, N)$ za svako $i = 1, \ldots, 9N$.

Očigledno, izrazi (4.1.1) i (4.1.2) zahtevaju maksimalno po 9 aritmetičkih operacija. Pritom moramo da izračunamo ukupno $9N \cdot N = 9N^2$ elemenata nizova $s(i, j)$ i $p(i, j)$, pa je prema tome ukupan broj potrebnih aritmetičkih operacija maksimalno $2 \cdot 9 \cdot 9N^2 = 162N^2 = \mathcal{O}(N^2)$.

[1] Međutim, ukoliko je $x_{N+1} = 0$, tada je proizvod cifara broja $x^{(2)}$ jednak nuli, odnosno imamo da je zbir cifara broja $x^{(1)}$ jednak nuli, što je nemoguće

Primetimo da se u rekurentnim izrazima (4.1.1) i (4.1.2) prilikom izračunavanja vrednosti j-te kolone matrica s i p koriste samo vrednosti u $j-1$-voj koloni, kao kod problema ranca, pa je i ovde, u implementaciji dovoljno pamtiti samo poslednju kolonu ovih matrica. Prema tome, ovo rešenje zahteva samo dva niza (s i p) od po 4500 elemenata.

4.2 Z-Menadžer

Poreklo zadatka: Z-trening [28]

Mali Z je dobio zadatak da direktoru firme u kojoj radi snimi filmove. Direktor je malom Z-u dao spisak filmova, i za svaki od filmova: vreme kada počinje, vreme kada se završava, i koliko ce Z dobiti para ako snimi taj film.

Mali Z ima na raspolaganju samo jedan video rekorder, pa ukoliko se dva filma vremenski preklapaju (dva filma se preklapaju i u slučaju kada jedan počinje u trenutku kada se drugi završava), on može da snimi samo jedan od njih. Odrediti koje filmove mali Z treba da snimi tako da tako da zaradi što je moguće više novca.

Predpostaviti da važe sledeća ograničenja: $1 <= N <= 100000$ gde je N ukupan broj filmova, $1 \leq p(i) < k(i) \leq 2000000000$ i $1 \leq c(i) \leq 1000$ gde su $p(i), k(i)$ i $c(i)$ redom vremena početka i kraja i-tog filma $1 \leq i \leq N$ a $c(i)$ zarada koju mali Z dobija ako snimi i-ti film.

Uputstvo: Sortirati filmove po rastućim vrednostima vremena završetka $k(i)$. Neka je $z(i)$ maksimalna zarada koja može biti ostvarena ukoliko je poslednji snimljen i-ti film. Ako je predzadnji j-ti film, onda važi $k(j) < p(i)$ i $z(i) = z(j) + c(i)$. Sada treba naći j tako da je $z(j)$ maksimalno.

Rešenje: Sortirajmo sve filmove tako da važi $k(1) \leq k(2) \leq \cdots \leq k(n)$. Neka je $z(i)$ maksimalna zarada koja može biti ostvarena ukoliko je poslednji snimljen i-ti film. Neka je j-ti film snimljen predposlednji. Tada iz uslova zadatka dobijamo da važi $k(j) > p(i)$. Zarada ostvarena filmovima

Neka su $i_1 < i_2 < \cdots < i_k$ filmovi koje treba snimiti da bi se maksimizovala zarada, pri čemu je poslednji snimljen i-ti film (tj $i_k = i$). Iz uslova problema imamo da važi $p(i) = p(i_k) > k(i_{k-1})$, pa je ukupna zarada ostvarena filmovima $i_1, \ldots, i_{k-1}$ manja ili jednaka $z(i_{k-1})$. Prema tome, imamo da važi:

$$z(i) = c(i_1) + \ldots + c(i_{k-1}) + c(i_k) \leq z(i_{k-1}) + c(i_k) \tag{4.2.1}$$

Ukoliko u izrazu (4.2.1) važi stroga nejednakost, tada možemo odabrati filmove $j_1, \ldots, j_{l-1}, j_l = i_{k-1}$ koji se međusobno ne preklapaju, pri čemu je ukupna zarada ostvarena ovim filmovima jednaka $z(i_{k-1})$. Tada se snimanjem filmova $j_1, \ldots, j_l, i$ dobija zarada

$$z(i) = z(i_{k-1}) + c(i_k) \tag{4.2.2}$$

veća od $z(i)$ što je u suprotnosti sa definicijom broja $z(i)$. Prema tome zaključujemo da u izrazu (4.2.1) važi jednakost, odnosno $z(i) = z(i_{k-1}) + c(i_k)$. Ostaje nam još samo da za zadato i odredimo i_{k-1} (tj film koji predhodno treba snimiti). Pošto je $z(i)$ maksimalna zarada koja se može ostvariti, ako je i-ti film poslednji snimljen,

sledi da i_{k-1} treba izabrati tako da je $z(i_{k-1})$ maksimalno, pri čemu važi uslov zadatka $p(i_k) > k(i_{k-1})$. Prema tome, važi sledeća rekurentna formula za $z(i)$:

$$z(i) = c(i) + \max\{z(j) \mid 0 \leq j < i, p(i) > k(j)\} \tag{4.2.3}$$

Startna vrednost za računanje $z(i)$ je $z(0) = 0$ (ukoliko se ne snimi nijedan film, zarada je jednaka nuli). Krajnje rešenje problema dobijamo kao $\max_{1 \leq i \leq N} z(i)$.

Što se tiče memorijskih zahteva, ovo rešenje zahteva samo jedan niz z od N elemenata. Međutim situacija je nešto složenija kada je u pitanju složenost algoritma. Naime, za izračunavanje vrednosti svakog elementa $z(i)$ direktnom primenom izraza (4.2.3), potrebno je ispitati ukupno $i - 1$ različitih mogućnosti za j, pa je složenost ovakve implementacije $\mathcal{O}(N^2)$. Pokažimo sada kako se ova vrednost može poboljšati. Prva ideja je da uporedo za nizom $z(i)$ računamo i niz maksimuma: $mz(i) = \max_{1 \leq k \leq n} z(i)$. Primetimo takođe, da ako je $p(i) > k(j)$, onda je $p(i) > k(j) > k(l)$ za svako $l = 1, \ldots, j$. Prema tome, dovoljno je naći:

$$j_0 = \text{argmin}\{k(j) \mid k(j) < p(i), j = 1, \ldots, i\}$$

i onda je:

$$\begin{aligned} z(i) &= c(i) + mz(j_0) \\ mz(i) &= \max\{mz(i-1), z(i)\} \end{aligned} \tag{4.2.4}$$

Pošto je niz $k(j)$ sortiran, vrednost j_0 možemo naći primenom tehnike binarnog pretraživanja. Primenom ove modifikacije smanjujemo ukupnu složenost izračunavanja jedne vrednosti $z(i)$ na $\mathcal{O}(\log N)$ operacija, a celog algoritma na $\mathcal{O}(N \log N)$ (sortiranje možemo obaviti koristeći neki od algoritama koji rade takođe u vremenu $O(N \log N)$, na primer QuickSort, MergeSort, HeapSort, itd.). Sada se ovaj algoritam bez problema može primeniti za velike vrednosti broja N ($N \leq 100000$).

Implementacija:

```
program Z-Menadzer;

Const MaxN=100005; type tniz=array [0..MaxN] of longint;

var x,y,v:tniz;
    n:longint;

procedure sort(l,r: longint); var i,j,xp,yp,tmp: longint; begin
  i:=l; j:=r;
  xp:=x[(l+r) DIV 2]; yp:=y[(l+r) div 2];
  repeat
    while (y[i]<yp) or ((y[i]=yp) and (x[i]<xp)) do i:=i+1;
    while (yp<y[j]) or ((y[j]=yp) and (xp<x[j])) do j:=j-1;
    if i<=j then
    begin
      tmp:=y[i]; y[i]:=y[j]; y[j]:=tmp;
      tmp:=x[i]; x[i]:=x[j]; x[j]:=tmp;
      tmp:=v[i]; v[i]:=v[j]; v[j]:=tmp;
      i:=i+1; j:=j-1;
    end;
  until i>j;
```

```
  if l<j then sort(l,j);
  if i<r then sort(i,r);
end;

function binpret(el:longint):longint; var l,d,sr  :longint; begin
  l:=1;  d:=n;
  while d-l>1 do begin
    sr:=(l+d) div 2;
    if y[sr]>=el then d:=sr else l:=sr;
  end;
  if el>y[l] then binpret:=l else binpret:=l-1;
  if el>y[d] then binpret:=d;    {new}
end;

var max:tniz;
    i,res,ind:longint;

begin
  readln(n);
  for i:=1 to n do
    readln(x[i],y[i],v[i]);
  fillchar(max,sizeof(max),0);

  sort(1,n);

  max[0]:=0;  max[1]:=v[1];
  for i:=2 to n do begin
    ind:=binpret(x[i]);
    if max[ind]+v[i]>max[i-1] then max[i]:=max[ind]+v[i]
                              else max[i]:=max[i-1];
  end;

  res:=max[n];
  writeln(res);
end.
```

4.3 Igra

Mirko i Slavko igraju sledeću igru: na stolu se nalaze n brojeva poredanih u niz. Prvo Mirko uzme jedan broj, sa leve ili sa desne strane niza. Zatim Slavko uzme jedan broj sa leve ili desne strane preostalog niza, i tako naizmenično dok ne pokupe sve brojeve sa stola. Napisati program koji izracunava, pod pretpostavkom da i Mirko i Slavko igraju optimalno, koliki je maksimalni zbir brojeva koji Slavko može da skupi.

Sa standardnog ulaza se ucitava u jednom redu broj n ($n \leq 128$), a u drugom redu n brojeva, u opsegu od 0 do 100.

Na standardni izlaz treba ispisate samo jedan broj, maksimalan zbir brojeva koje Mirko može da osvoji.

Uputstvo: Označimo sa $m(i, j)$ odnosno $s(i, j)$ maksimalan zbir brojeva koje može da sakupi Mirko odnosno Slavko ukoliko obojica igraju optimalno, i ako su na tabli zapisani brojevi $a_i, a_{i+1}, \ldots, a_{j-1}, a_j$.

Rešenje: Pošto je Mirko prvi na potezu, on bira koji broj će uzeti, a_i ili a_j. Ukoliko odabere broj a_i, na tabli ostaju brojevi $a_{i+1}, \ldots, a_j$ i je Slavko na potezu. Primetimo da je situacija ista kao pre Mirkovog poteza, samo što su Mirko i Slavko zamenili uloge (sada je Slavko prvi a Mirko drugi igrač). Prema tome, ukoliko obojica igraju optimalno do kraja igre, zbir brojeva koje uzima Mirko je $a_i + s(i+1, j)$ a Slavko $m(i+1, j)$. Slično, ukoliko se Mirko odluči za a_j u prvom potezu, njegov zbir je $a_j + s(i, j-1)$ dok je Slavkov $m(i, j-1)$. Prema tome, dobijamo sledeće rekurentne izraze:

$$\begin{aligned} m(i,j) &= \max\{a_i + s(i+1,j), a_j + s(i,j-1)\} \\ s(i,j) &= \begin{cases} m(i+1,j) & , a_i + s(i+1,j) > a_j + s(i,j-1) \\ m(i,j-1) & , \text{u suprotnom} \end{cases} \end{aligned} \tag{4.3.1}$$

Kao startne vrednosti možemo uzeti $m(i,i) = a_i$ i $s(i,i) = 0$ (ukoliko je na tabli zapisan samo jedan broj, u prvom potezu ga Mirko uzima i završava igru). Konačno rešenje problema je naravno $s(1,n)$.

Za implementaciju ovog rešenja potrebno je pamtiti dva dvodimenzionalna niza $m(i,j)$ i $s(i,j)$, što zahteva $\mathcal{O}(n^2)$ memorije. Kao i kod problema ranca, i ovde je dovoljno pamtiti samo jednu vrstu matrica $s(i,j)$ i $m(i,j)$, pa su sada memorijski zahtevi $\mathcal{O}(n)$. Složenost ovog rešenja je $\mathcal{O}(n^2)$.

Implementacija:

```
program Igra;

function max(a,b:integer):integer; begin
  if a>b then max:=a else max:=b;
end;

var m,s:array [1..130,1..130] of integer;
    a:array [1..130] of integer;
    i,j,n,r,zbir:integer;

begin
  readln(n);
  for i:=1 to n do
    read(a[i]);

  fillchar(m,sizeof(m),0);
  fillchar(s,sizeof(s),0);

  for i:=1 to n do begin
    m[i,i]:=a[i]; s[i,i]:=0;
  end;

  for i:=n-1 downto 1 do begin
    for j:=i+1 to n do begin
       m[i,j]:=max((a[i]+s[i+1,j]),(a[j]+s[i,j-1]));
       zbir:=0;
       for r:=i to j do zbir:=zbir+a[r];
       s[i,j]:=zbir-m[i,j];
    end;
  end;
```

```
  writeln(m[1,n]);
end.
```

4.4 Bankomati

U novoj eri elektronskog poslovanja, bankomati sve više postaju svakodnevna pojava. Banke žele da se klijenti što krace zadržavaju kod bankomata, kako bi opslužili što više ljudi, i zbog toga zahtevaju da se traženi iznos novca klijentima isplacuje sa što manje novčanica. Medutim, bankomate su programirali lenji programeri koji su upotrebili najjednostavniji algoritam tako da se u svakom njegovom koraku bira najveća novčanica koja je manja ili jednaka od preostalog iznosa. Na primer, za sistem sa novčanicama od 1, 7 i 10 dinara, iznos od 14 dinara biće isplaćen pomoću jedne novčanice od 10 i 4 novčanice od 1 dinara, ukupno 5 novčanica. Ovo ne odgovara zahtevu banke, jer se 14 dinara može isplatiti pomoću samo dve novčanice od 7 dinara.

Vaš zadatak je da napišete program koji ce za dati sistem od $n \leq 50$ novčanica, koji uvek sadrži i apoen od jednog dinara, naći najmanji iznos koji se može isplatiti u manje novčanica od broja novčanica koji bi bankomat iskoristio. Vrednost najveće novčanice je manja ili jednaka 500000.

Uputstvo: Neka su $n_1, \ldots, n_k$ ($n_1 < n_2 < \cdots < n_k$ iznosi na novčanicama koje poseduje bankomat. Takođe neka je $o(x)$ optimalan broj novčanica potreban da se isplati suma x a $g(x)$ broj novčanica koje isplaćuje bankomat. Pronaćićemo rekurentne formule na osnovu kojih se računaju brojevi $o(x)$ i $g(x)$, a zatim ćemo naći minimalni broj x takav da je $g(x) > o(x)$. Dokazaćemo da ukoliko je $g(x) = o(x)$ za svako $x \leq M$, za neki prirodan broj M (koji zavisi od $n_1, \ldots, n_k$), da je tada $g(x) = o(x)$ za svako $x \in \mathbb{N}$.

Rešenje: Pronađimo najpre rekurentnu formulu za $o(x)$. Posmatrajmo optimalnu isplatu $x = p_1 + \ldots + p_l$ sume x pri čemu je $p_i \in \{n_1, \ldots, n_k\}$ za svako $i = 1, \ldots, l$. Neka je $p_l = n_j$. Tada je $x - n_j = p_1 + \ldots + p_{l-1}$ optimalna isplata sume $x - n_j$ sa ukupno $l - 1$ novčanica (ukoliko bi postojao način da se suma $x - n_j$ isplati pomoću manje od $l - 1$ novčanica, tada bi dodavanjem nočanice n_j dobili isplatu sume x sa manje od l novčanica, što je nemoguće). Prema tome, imamo da važi $l = o(x) = o(x - n_j) + 1$. Slično kao i u predhodnim primerima dobijamo da indeks j treba izabrati tako da je zbir na desnoj strani predhodne jednakosti minimalan. Prema tome, imamo sledeću rekurentnu jednačinu:

$$o(x) = 1 + \min_{n_j \leq x} o(x - n_j) \tag{4.4.1}$$

Startna vrednost je u ovom slučaju $o(0) = 0$. Pošto je $n_1 = 1$ prema uslovu zadatka, matematičkom indukcijom lako pokazujemo da primenom startnog uslova $o(0) = 0$ i rekurentne relacije (4.4.1) možemo izračunati vrednosti $o(x)$ za svako $x \in N$.

Konstuišimo sada rekurentnu formulu za $g(x)$. Prema uslovu zadatka, bankomat u bira najveću novčanicu koja je manja ili jednaka datom iznosu. Neka je to nočanica

n_j. Nadalje bankomat postupa na isti način, ali sa iznosom $x - n_j$. Prema tome važi sledeća rekurentna formula:

$$g(x) = g(x - n_j) + 1, \quad n_j = \max_{n_i \leq x} n_i \tag{4.4.2}$$

Startna vrednost je i u ovom slučaju $g(0) = 0$.

Prema tome, primenjujući izraze (4.4.1) i (4.4.2) možemo da računamo vrednosti $g(x)$ i $o(x)$ redom za $x = 1, 2, 3, 4, \ldots$ sve dok ne pronađemo broj x_0 takav da je $o(x_0) < g(x_0)$. Međutim, može se desiti da takav broj ne postoji, pa prema tome moramo znati do kog broja treba vršiti pretragu. Konkretno, treba naći takav broj M (M naravno zavisi od $n_1, \ldots, n_k$) takav da ukoliko je $g(x) = o(x)$ za svako $x \leq M$, da je onda i $g(x) = o(x)$ za svako $x \in \mathbb{N}$.

Dokažimo da $M = 2n_k$ zadovoljava ovaj uslov. Predpostavimo suprotno, neka je x_0 minimalan broj takav da je $g(x_0) > o(x_0)$ i neka važi $x_0 > M$. Imamo da je $g(x) = o(x)$ za svako $x = 0, 1, \ldots, x_0 - 1$. Na osnovu (4.4.1) i (4.4.2) imamo da je:

$$o(x_0) = o(x_0 - n_j) + 1, \quad g(x_0) = g(x_0 - n_k) + 1 \tag{4.4.3}$$

Iz (4.4.3) direktno dobijamo da je $o(x_0 - n_j) < g(x_0 - n_k)$ Ukoliko je $j = k$ iz $o(x_0 - n_k) = g(x_0 - n_k)$ dobijamo da je $o(x) = g(x)$, što je kontradikcija. Pretpostavimo sada da je $j \neq k$. Pošto je $o(x_0 - n_j) = g(x_0 - n_j)$ i važi

$$x_0 - n_j \geq x_0 - n_{k-1} > M - n_{k-1} = n_k,$$

imamo da je vrednost prve novčanice u optimalnoj isplati sume $x_0 - n_j$ jednaka n_k. Prema tome važi:

$$o(x_0 - n_j) = o(x_0 - n_j - n_k) + 1$$

Na osnovu (4.4.1) imamo da je:

$$o(x_0 - n_k) \leq o(x_0 - n_j - n_k) + 1 = o(x_0 - n_j) < g(x_0 - n_k)$$

što predstavlja kontradikciju sa predpostavkom o minimalnosti broja x_0. Ovim smo dokazali da je dovoljno izračunati $g(x)$ i $o(x)$ za svako $x = 1, \ldots, M$ da bi mogli da utvrdimo da li postoji broj x_0 takav da je $g(x_0) > o(x_0)$.

Za implementaciju izloženog rešenja potrebna su nam dva niza g i o od po $M = n_k + n_{k-1}$ elemenata. Međutim, primetimo da je u izrazima (4.4.1) i (4.4.2) za računanje x-tog clana nizova g i o potrebno tačno n_k prethodnih članova svakog niza. Prema tome, dovoljno je pamtiti po n_k članova svakog niza.

Kako je za primenu rekurentne formule (4.4.1) potrebno maksimalno k upoređivanja, dok za (4.4.2) je potrebno samo jedno sabiranje sledi da je ukupna složenost ovog metoda jednaka $\mathcal{O}(kM) = \mathcal{O}(k(n_k + n_{k-1}))$.

Implementacija:

```
program Bankomati;

Const MaxN=1000000;
```

```
var d1,g1,m,ind:array [0..MaxN] of longint;
    tmp,a:array [1..51] of longint;
    n:longint;

procedure sort(l,r: longint); var i,j,x,y: longint; begin
  i:=l; j:=r; x:=tmp[(l+r) DIV 2];
  repeat
    while tmp[i]<x do i:=i+1;
    while x<tmp[j] do j:=j-1;
    if i<=j then
    begin
      y:=tmp[i]; tmp[i]:=tmp[j]; tmp[j]:=y;
      i:=i+1; j:=j-1;
    end;
  until i>j;
  if l<j then sort(l,j);
  if i<r then sort(i,r);
end;

var i,k,l,v,res,maxp:longint;
    bl,nema:boolean;

begin
  readln(n);
  for i:=1 to n do read(a[i]);
  readln;

  fillchar(ind,sizeof(ind),0);

  for i:=1 to n do tmp[i]:=a[i];
  for i:=1 to n do ind[tmp[i]]:=i;

  sort(1,n);
  for i:=1 to 2*tmp[n] do d1[i]:=i;

  bl:=true;

  m[0]:=1;

  for i:=1 to 2*tmp[n] do
    if ind[i]=0 then m[i]:=m[i-1] else m[i]:=a[ind[i]];

  for i:=1 to 2*tmp[n] do
    g1[i]:=g1[i-m[i]]+1;

  d1[0]:=0;
  g1[0]:=0;

  for v:=0 to 2*tmp[n] do begin
    for l:=1 to n do
      if (a[l]<=v) then if (d1[(v-a[l])]+1<=d1[v]) then d1[v]:=d1[(v-a[l])]+1;
    bl:=g1[v]=d1[v];
    if not bl then break;
  end;
  if not bl then writeln(v) else writeln(0);
end.
```

Napomena: Ukoliko se izostavi minimalnost broja x_0, postoji rešenje ovog problema u vremenu $\mathcal{O}(k^2)$. Posmatrajmo isplatu sume n_t koju vrši bankomat pomoću novčanica $n_1, \ldots, n_{t-1}$. Neka je $b_{t,j}$ broj nočanica n_j koje učestvuju u toj isplati $(1 \leq j \leq t-1)$. Tada važi:

$$a_t = b_{t,1}n_1 + \ldots + b_{t,t-1}n_{k-1}$$

Posmatrajmo brojeve:

$$P_{t,j} = n_j + \sum_{i=j}^{t-1} b_{t,i}n_i \tag{4.4.4}$$

Može se pokazati [2] da ukoliko postoji broj x_0 takav da je $g(x_0) > o(x_0)$, onda bar jedan od brojeva $P_{k,j}$ zadovoljava da je isplata sume $P_{k,j}$ koju daje bankomat lošija od isplate definisane izrazom (4.4.4). Prema tome, ovaj broj zadovoljava uslove zadatka. Za generisanje brojeva $P_{k,j}$ kao i za proveru da li neki od njih zadovoljava uslove zadatka, potrebno je ukupno $\mathcal{O}(k^2)$ aritmetičkih operacija.

Implementaciju ovog rešenja problema prepuštamo čitaocu.

4.5 Cveće i vaze

Želite da aranžirate izlog u prodavnici cveća na najbolji mogući način. Imate F vrsta cveća i V vaza, pri čemu je $V \geq F$. Vaze su pričvršćene na stalak i numerisane redom brojevima $1, 2, \ldots, V$. Ukoliko vrstu cveća sa rednim brojem i stavite u vazu sa rednim brojem j, estetski izgled koji na ovaj način postižete jednak je a_{ij} (pritom ovaj može biti i negativan). Cveće svake vrste može biti postavljeno najviše u jednu vazu i ukoliko je cveće i-te vrste postavljeno u k-tu vazu, cveća $i+1$-ve,...,n-te vrste moraju biti postavljena u neku od vaza $k+1, \ldots, v$. Odrediti način na koji treba rasporediti cveće po vazama tako da ukupan estetski izgled bude maksimalan.

Uputstvo: Označimo sa $E(i, j)$ maksimalni estetski izgled koji je moguće dobiti sa prvih i vrsta cveća i prvih j vaza. Odredićemo rekurentnu formulu za niz $E(i, j)$.

Rešenje: Razlikovaćemo dva slučaja, ukoliko cveće i-te vrste stavljamo u j-tu vazu i ukoliko to nije slučaj. U prvom slučaju, imamo da nam ova kombinacija povećava ukupan estetski izgled za a_{ij}. Sada je potrebno cveće $1, \ldots, i-1$ rasporediti u vaze $1, \ldots, j-1$ i pritom maksimizovati ukupni estetski izgled. Ovaj maksimalni estetski izgled jednak je $E(i-1, j-1)$.

U drugom slučaju, ukoliko se cveće i-te vrste ne stavi u j-tu vazu, onda ili j-ta vaza ostaje prazna, ili cveće vrste i ne stavljamo ni u jednu od predhodnih vaza. Maksimalni estetski izgledi u ova dva slučaja jednaki su redom $E(i, j-1)$ i $E(i-1, j)$. Prema tome, imamo da važi sledeća rekurentna formula:

$$E(i, j) = \max\{E(i-1, j-1) + a_{ij}, E(i-1, j), E(i, j-1)\} \tag{4.5.1}$$

Startne vrednosti su $E(i, 0) = E(0, j) = 0$ za svako $1 \leq i \leq F$ i $1 \leq j \leq V$. Za rekonstrukciju optimalnog rasporeda, dovoljno je u niz $P(i, j)$ pamtiti koji je od tri broja sa desne strane izraza (4.5.1) najveći.

[2] *The Fifth Tournament of Towns*, Moscow, 1994

Ukoliko se traži rekonstrukcija optimalnog rasporeda, ovo rešenje zahteva pamćenje dva dvodimenzionalna niza (E i P) od po FV elemenata. Ukoliko to nije slučaj, onda pošto se u izrazu (4.5.1) prilikom izračunavanja $E(i,j)$ koriste vrednosti iz i-te i $i-1$-ve kolone matrice E, dovoljno je pamtiti samo poslednju vrstu niza E, odnosno ukupno V elemenata.

Složenost izloženog rešenja je u oba slučaja (bilo da se traži rekonstrukcija optimalnog rasporeda ili ne) jednaka $\mathcal{O}(FV)$.

Implementaciju izloženog rešenja problema prepuštamo čitaocu.

4.6 Z-Draguljčići

Poreklo zadatka: Z-trening [28].

U nekoj azbuci postoji S slova ($1 \leq S \leq 100$). Sva slova ove azbuke se sastavljaju pomoću dragulja, i pritom je broj dragulja potreban za sastavljanje i-tog slova jednak d_i.

Potrebno je pronaći ukupan broj reči (reč je bilo koji niz slova) koje se mogu sastaviti pomoću N dragulja ($N \leq 10000$).

Uputstvo: Neka je $a(i,j)$ broj reči koji može da se sastavi pomoću i dragulja, pri čemu je j-to slovo poslednje u reči. Potrebno je pronaći rekurentnu formulu za računanje elemenata niza $a(i,j)$.

Rešenje: Pošto je j-ta reč poslednja, ostatak reči je potrebno formirati pomoću $i-d_j$ dragulja (naravno, uz predpostavku da je $i \geq d_j$, u suprotnom je trivijalno $a(i,j)=0$). Ukupan broj reči koje je moguće formirati pomoću $i-d_j$ dragulja je $\sum_{k=1}^{S} a(i-d_j,k)$ (poslednje slovo ostatka reči, tj predposlednje slovo reči može biti bilo koje). Prema tome, važi sledeća rekurentna formula:

$$a(i,j) = \sum_{k=1}^{S} a(i-d_j,k) \tag{4.6.1}$$

uz startne vrednosti $a(0,0)=1$ i $a(0,j)=0$ za $1 \leq j \leq S$. Ove startne vrednosti obuhvataju slučaj prazne reči (postoji samo jedna reč koja se može sastaviti bez dragulja, a to je prazna reč). Primetimo za je izrazu (4.6.1) suma na desnoj strani funkcija od $i-d_j$. Ovaj izraz predstavlja sumu elemenata $i-d_j$-te vrste niza $a(i,j)$, i označićemo ga sa $s(i-d_j)$. Sada je:

$$\begin{aligned} a(i,j) &= \begin{cases} s(i-d_j), & i \geq d_j \\ 0, & \text{u suprotnom} \end{cases} \\ s(i) &= \sum_{j=1}^{S} a(i,j) \end{aligned} \tag{4.6.2}$$

Najpre se za konkretno i izračunaju sve vrednosti $a(i,j)$, a zatim se računa njihova suma $s(i)$. Konačno rešenje se dobija kao: $s(1)+\ldots+s(N)$.

Primetimo da ovo rešenje zahteva pamćen je nizova $a(i,j)$ i $s(i)$, za šta je potrebno ukupno $\mathcal{O}(NS)$ memorije. Međutim moguće je izbeći pamćenje niza $a(i,j)$ i direktno računati elemente niza $s(i)$ pomoću sledećeg izraza koji se dobija iz (4.6.2):

$$s(i) = \sum_{d_j \leq i} s(i - d_j) \tag{4.6.3}$$

Kod ove varijante potrebno nam je samo $O(N)$ memorije. Takođe primetimo da su $s(i)$, kao i rešenje zadatka veoma veliki, čak i za male vrednosti ulaznih parametara. Zato je neophodno implementirati neku od struktura za rad sa velikim brojevima. Npr. ukoliko je $S = 100$, $N = 10000$ i $d_i = 1$ za svako $i = 1, \ldots, S$ (slučaj u kome se dobija najveći broj reči), dobijamo da je ukupan broj reči jednak približno $1.995 \cdot 10^{3010}$.

Što se tiče složenosti ovog rešenja, bilo da se koriste izrazi (4.6.2) ili izraz (4.6.3), složenost je jednaka $\mathcal{O}(NS)$.

Da bi izbegli korišćenje strukture podataka za manipulaciju sa velikim brojevima, u implementaciji ćemo rešenje određivati po modulu 8192.

Implementacija:

```
program Z-Draguljcici Const Modul=8191; Const MaxM=105; Const
MaxN=10005;

type tniz=array [0..MaxM] of longint; type tmat=array [0..MaxN] of
tniz;

var a:tmat ;
    d,s:array [0..MaxN] of longint;
    n,m,i,j,res :longint;

begin
  readln(n,m);
  for i:=1 to m do readln(d[i]);

  fillchar(s,sizeof(s),0);
  fillchar(a,sizeof(a),0);

  s[0]:=1;  a[0][0]:=1;

  for i:=1 to n do begin
    for j:=1 to m do
      if d[j]<=i then a[i][j]:=s[i-d[j]];
    for j:=1 to m do begin
      s[i]:=s[i]+a[i][j];
      s[i]:=s[i] and Modul;
    end;
  end;

  res:=0;
  for i:=1 to n do begin
    res:=res+s[i];
    res:=res and Modul;
  end;
```

```
  res:=res and Modul;
  writeln(res);
end.
```

4.7 Pljačka

Tri lopova su ukrala n dijamanata i treba da ih podele. Pritom je svaki lopov različito procenio svaki dijamant, tako da i-ti dijamant vredi p_{li} dinara, $1 \leq p_{li} \leq P$ prema proceni l-tog od lopova ($l = 1, 2, 3$, $i = 1, 2, \ldots, n$. Izvršiti podelu dijamanata tako da je ukupna vrednost dijamanata koje dobije l-ti lopov jednaka bar s_l dinara (prema njegovoj proceni), $1 \leq s_l \leq Pn$ kao i da je ukupan zbir procenjenih zarada maksimalan. Smatrati da je $P = 40$ a $n \leq 25$.

Uputstvo: Neka je $d(z_1, z_2, k)$ maksimalna zarada koju može da ostvari treći lopov pomoću prvih k dijamanata a da pritom prvi i drugi redom zarade z_1 odnosno z_2 dinara. Pošto je $p_{li} \leq P$ sledi da je $z_1, z_2 \leq Pn$. Prema tome, niz d ima maksimalno P^2n različitih elemenata. Pronađimo rekurentnu formulu za niz $d(z_1, z_2, k)$.

Rešenje: Ukoliko k-ti dijamant uzme prvi lopov, onda on mora zaraditi tačno $z_1 - a_k$ dinara, a drugi z_2 dinara uzimajući neke od predhodnih dijamanata. U tom slučaju maksimalna zarada trećeg lopova biće $d(z_1 - p_{1k}, z_2, k - 1)$. Slično dobijamo da ukoliko drugi lopov uzme k-ti dijamant, maksimalna zarada trećeg je $d(z_1, z_2 - p_{2k}, k - 1)$. Napokon ukoliko ovaj dijamant uzme treći lopov, prva dva moraju zaraditi z_1 odnosno z_2 dinara uzimajući prvih $k - 1$ dijamanata dok je zarada trećeg $d(z_1, z_2, k-1) + p_{3k}$. Prema tome, imamo da važi sledeća rekurentna formula:

$$\begin{aligned} d(z_1, z_2, k) = \max\{&d(z_1, z_2, k-1) + p_{3k}, \\ &d(z_1 - p_{1k}, z_2, k-1), \\ &d(z_1, z_2 - p_{2k}, k-1)\} \end{aligned} \tag{4.7.1}$$

Startne vrednosti za niz d su:

$$d(z_1, z_2, 0) = \begin{cases} 0, & (z_1, z_2) = (p_{11}, 0) \\ 0, & (z_1, z_2) = (0, p_{21}) \\ p_{31}, & z_1 = z_2 = 0 \\ -\infty, & \text{u suprotnom} \end{cases} \tag{4.7.2}$$

Primetimo da je u izrazu (4.7.2) $d(z_1, z_2, 0) = -\infty$ kada ne postoji podela tako da prvi lopov zaradi z_1 a drugi z_2 dinara. Ovaj zaključak se može proširiti i na $d(z_1, z_2, k)$, i upravo zato je vrednost $-\infty$ odabrana za ovaj slučaj. Naravno, u implementaciji bi trebalo koristiti negativan broj dovoljno veliki po apsolutnoj vrednosti (npr. $-Pn$, pošto je maksimalna moguća zarada jednaka Pn). Vrednosti niza $d(z_1, z_2, k)$ potrebno je izračunati za sve $0 \leq z_1, z_2 \leq Pn$ kao i $k = 1, 2, \ldots, n$.

Za konačno rešenje problema potrebne su vrednosti $d(z_1, z_2, n)$ za $z_1 \geq s_1$ i $z_2 \geq s_2$. Maksimalan zbir procenjenih zarada lopova lako dobijamo pomoću:

$$\begin{aligned} D = \max\{&z_1 + z_2 + d(z_1, z_2, n) \mid s_1 \leq z_1 \leq Pn, \\ &s_2 \leq z_2 \leq Pn,\ d(z_1, z_2, n) \geq s_3\} \end{aligned} \tag{4.7.3}$$

Ukoliko je $D = -\infty$ (ili je skup čiji se maksimum određuje na desnoj strani izraza (4.7.3) prazan), sledi da ne postoji podela koja bi zadovoljila sva tri lopova po pitanju zarade. U suprotnom, ukoliko je $D > -\infty$, raspodela dijamanata pri kojoj se dostiže maksimum D može se rekonstruisati rekurzivno. U svakom koraku se ispituje koja je od tri vrednosti na desnoj strani izraza (4.7.1) maksimalna, odnosno koji lopov uzima k-ti dijamant.

Ako se ne zahteva rekonstrukcija raspodele dijamanata, onda je dovoljno pamtiti vrednosti $d(z_1, z_2, k-1)$ da bi se izračunale odgovarajuće vrednosti $d(z_1, z_2, k)$, dakle samo P^2 vrednosti. U suprotnom je potrebno pamtiti ceo niz d, odnosno $P^2 n$ elemenata. Alternativno, možemo pamtiti u nizu $p(z_1, z_2, k)$ koji lopov dobija k-ti dijamant. Složenost ovog rešenja je $\mathcal{O}(P^2 n)$.

Implementaciju izloženog rešenja problema prepuštamo čitaocu.

4.8 Krug

Na kružnici je dato $2k$ tačaka $A_1, \ldots, A_{2k}$ pri čemu su raspoređene bas tim redosledom u smeru kazaljke na satu. Krug je podeljen tetivama čija su temena date tačke, pri čemu je svaka tačka teme tačno jedne tetive. Ove tetive dele krug na određeni broj delova P. Naći broj načina N na koji možemo konstruisati tetive tako da je broj delova P minimalan.

Uputstvo: Minimalni broj delova P se dobija kada se nikoje dve tetive ne seku i on je jednak $P = k + 1$. Neka je $N(k)$ traženi broj kombinacija. Uočićemo tačku A_1 i utvrditi koje su sve mogućnosti za drugo teme tetive čije je prvo teme tačka A_1.

Rešenje: Dokažimo najpre da je minimalan mogući broj delova na koje se razbija krug konstruisanim tetivama jednak $P = k + 1$ i da se ovaj broj dostiže akko se nikoje dve tetive ne seku. Konstruišimo tetive redom, jednu po jednu i posmatrajmo kako se menja broj delova na koje se krug razbija. Na početku je broj delova jednak 1 (ceo krug bez tetiva). Kad konstrušemo prvu tetivu, broj delova postaje jednak $2 = 1 + 1$. Predpostavimo sada da smo konstruisali i tetiva i konstruišimo $i+1$-vu. Ukoliko ona ne seče nijednu drugu tetivu, sledi da cela pripada samo jednom delu. Ona ovaj deo deli na dva, pa smo na ovaj način dobili još jedan deo. Ukoliko tetiva seče l drugih tetiva, ona prolazi kroz ukupno $l+1$ delova, i svaki deli na dva. Prema tome dobili smo još $l + 1$ novih delova, što je više nego u prvom slučaju. Znači najmanji broj delova, $P = k + 1$ dobija se ukoliko se tetive ne seku.

Pronađimo sada rekurentnu formulu za broj $N(k)$ načina da se krug podeli na $k + 1$ delova, ili ekvivalentno da se povuku k tetiva koje se međusobno ne seku. Neka je tačka A_1 spojena sa tačkom A_p. Primetimo da tetiva A_1A_p deli krug na dva dela, pri čemu su tačke $A_2, \ldots, A_{p-1}$ u jednom a $A_{p+1}, \ldots, A_{2k}$ u drugom delu. Jasno je da ne može postojati tetiva čije je jedno teme u jednom a drugo u drugom delu. Dalje primetimo da ukoliko je p neparan broj, u prvom delu ima ukupno $p-2$ tačke što je neparan broj, pa je odgovarajuće spajanje tetivama nemoguće. Ukoliko je $p = 2i$ paran broj, tetive u svakom od delova je moguće povući redom na $N(i-1)$

odnosno $N(k-i)$ načina. Broj i, odnosno tačku A_p možemo odabrati proizvoljno. Prema tome, imamo da važi sledeća rekurentna formula:

$$N(k) = \sum_{i=1}^{k} N(k-i)N(i-1) \tag{4.8.1}$$

Startna vrednost za niz $N(k)$ je $N(0) = 1$.

Ovo rešenje zahteva pamćenje jednog niza N dužine k. Uočimo da $N(k)$ jako brzo raste kad k raste. Već za $k = 20$ dobijamo $N(20) = 6564120420$ što izlazi iz opsega 32-bitnih celih brojeva. Znači, za izračunavanje vrednosti $N(k)$ za $k \geq 20$ potrebno je korišćenje neke strukture za operacije sa velikim brojevima. Složenost ovog rešenja je $\mathcal{O}(k^2)$.

Implementacija:

```
Const MaxN=300; type tniz=array [0..MaxN] of int64;

var s:tniz;
    k,n,i,j:longint;
    sum:int64;

begin
  readln(k);
  n:=2*k;  s[0]:=1;  i:=2;

  while i<=n do begin
    sum:=0;
    for j:=1 to (i div 2) do
      sum:=sum+s[i-2*j]*s[2*(j-1)];
    s[i]:=sum;
    i:=i+2;
  end;

  writeln(s[n],' ',k+1);
end.
```

Napomena: Može se dokazati da je opšti član niza $N(k)$ jednak:

$$N(k) = \frac{1}{k+1}\binom{2k}{k}$$

Ovi brojevi su poznati kao Katalanovi brojevi.

4.9 Lift

U nekoj zgradi koja ima ukupno M spratova postoji stari i dotrajali lift. Pretpostavimo da kompanija koja je vlasnik zgrade ima n radnika i da u jednom trenutku svi oni žele da se popnu na podkrovlje (sprat iznad M-tog sprata). Pritom je i-ti radnik u posmatranom trenutku na s_i-tom spratu a lift je u prizemlju. U zgradi inače postoje i stepenice, ali se do potkrovlja može doći isključivo liftom. Ukupan broj stajanja lifta na putu do potkrovlja je S. Ukoliko lift ne stane na sprat s_i, i-ti

radnik mora stepenicama da dođe do najbližeg sprata na koji lift staje. Odrediti na kojih S spratova lift treba da stane tako da je broj spratova koje radnici trebaju preći stepenicama minimalan.

Uputstvo: Neka je $m(i,j)$ minimalan broj spratova koje bi radnici trebalo da pređu stepenicama pri čemu lift staje na ukupno i spratova a j je poslednji sprat na koji lift staje. Pronađimo rekurentnu formulu za $m(i,j)$

Rešenje: Posmatrajmo optimalno stajanje lifta tako da on staje i puta i da su predposlednji i poslednji sprat na koje lift staje redom k-ti i j-ti. Radnici koji se nalaze na spratovima ispod k-tog sigurno neće ući u lift na j-tom spratu jer im je k-ti sprat bliži. Radnici koji su iznad k-tog sprata, ićiće stepenicama ili do k-tog, ili do j-tog, zavisno koji im je sprat bliži, odnosno svi radnici p za koje važi $s(p) \leq \frac{k+j}{2}$ ićiće do k-tog a ostali do j-tog sprata. Neka je

$$st(i,j) = \sum_{i<s(p)\leq j} \min\{s(p)-i, j-s(p)\} \tag{4.9.1}$$

Izraz $st(i,j)$ predstavlja ukupan broj spratova koji pešače radnici koji su između i-tog i j-tog sprata da bi došli do bližeg od ta dva sprata. Uočimo da je ukupan broj spratova koji prelaze radnici koji su ispod k-tog sprata jednak $m(i-1,k)-st(k,+\infty)$. Radnici koji su iznad k-tog sprata prelaze ukupno $st(k,i) + st(i,+\infty)$ spratova pešaka. Broj k naravno treba odabrati tako da se ukupan broj pređenih spratova minimizuje, pa prema tome važi sledeća rekurentna jednačina:

$$m(i,j) = \min_{k=i-1,i,\ldots,j}\{m(i-1,k) - st(k,+\infty) + st(k,i) + st(i,+\infty)\} \tag{4.9.2}$$

Startne vrednosti za niz $m(i,j)$ su $m(0,k) = st(0,+\infty)$ za svako $k = 0,1,\ldots,M$ (pošto nema stajanja, svi radnici moraju najpre da siđu u prizemlje a zatim da se ukrcaju u lift). Konačno rešenje problema se dobija kao:

$$\max_{0\leq t\leq M} m(S,t) = m(S,t_0)$$

Poslednji sprat na koji lift staje je t_0. Rešenje nadalje možemo rekonstruisati obrnutom primenom rekurentne formule (4.9.2) odnosno nalaženjem indeksa k takvog da se izraz na desnoj strani formule (4.9.2) minimizuje. Neka je taj indeks k_0, i on predstavlja indeks predhodnog sprata na koji lift staje. Na isti način se rekonstruišu i ostali spratovi. Alternativno, možemo u nizu $p(i,j)$ pamtiti indeks k_0, čime uprošćavamo rekonstrukciju rešenja.

Što se tiče memorijskih zahteva, ovo rešenje zahteva pamćenje niza $m(i,j)$ i eventualno niza $p(i,j)$. Znači potrebna količina memorije je $\mathcal{O}(SM)$. Pritom, ukoliko rekonstrukcija rešenja nije potrebna, dovoljno je pamtiti poslednju izračunatu kolonu matrice $m(i,j)$, tj dovoljno je $\mathcal{O}(M)$ memorije. Pošto se pri svakoj primeni izraza (4.9.2) izračunava vrednost funkcije st na osnovu (4.9.1), ukupna složenost metoda je $\mathcal{O}(SM^2n)$. Pritom, vrednosti ove funkcije možemo prekalkulisati i smestiti u odgovarajući niz, za šta nam je potrebno $\mathcal{O}(SMn)$ operacija i još $\mathcal{O}(SM)$ memorije. Međutim, ukupna složenost rešenja u ovom slučaju redukuje se na $\mathcal{O}(SMn)$. Implementaciju izloženog rešenja problema prepuštamo čitaocu.

4.10 Sečenje štapića

Potrebno je iseći štapić dužine L na nekoliko delova. Radionica u kojoj se vrši sečenje naplaćuje svako pojedinačno sečenje srazmerno dužini štapića koji se seče. Lako se vidi da različiti načini sečenja različito koštaju.

Na primer, ako sečemo štapić dužine 10 na četiri dela čije su dužine redom $2, 3, 4$ i 1 i ako prvo presečemo štapić na dužini 5 od levog kraja a zatim svaki deo ponovo sečemo, ukupna cena koju plaćamo je $10 + 5 + 5 = 20$. Međutim, ukoliko sečemo najpre na dužini 2 od levog kraja, zatim veći štapić na dužini 3, itd... cena koju plaćamo je $10 + 8 + 5 = 23$.

Ako je data dužina štapa L, $L \leq 1000$ kao i rastojanja od levog kraja štapa na kojima treba izvršiti rez $d_1, \ldots, d_n$, $d_1 < d_2 < \cdots < d_n < L$, $n \leq 50$ naći minimalnu potrebnu cenu kao i način sečenja da bi se ta cena ostvarila.

Uputstvo: Neka je $c(i, j)$ minimalna cena potrebna da se iseče deo štapića od i-tog do j-tog mesta. Pronaćićemo rekurentnu formulu za niz $c(i, j)$.

Rešenje: Predpostavimo da se deo štapića najpre seče na k-tom mestu. Cena ovog sečenja je jednaka dužini dela štapića $d_j - d_i$. Posle sečenja imamo nova dva dela štapa, od i-tog do k-tog mesta i od k-tog do j-tog. Koje moramo nadalje da sečemo. Minimalna cena potrebna da se iseku ova dva dela štapića na odgovarajući način je $c(i, k) + c(k, j)$. Na osnovu ovoga, na sličan način kao u predhodnim problemima dobijamo da je:

$$c(i, j) = \min_{i<k<j} \{c(i, k) + c(k, j)\} + d_j - d_i \tag{4.10.1}$$

Startne vrednosti za niz $c(i, j)$ su $c(i, i+1) = 0$. Izračunavanje vrednosti elemenata matrice c pomoću jednačine (4.10.1) vrši se po dijagonalama paralelnim glavnoj dijagonali ove matrice.

Složenost ovog rešenja je $\mathcal{O}(n^3)$ a potrebna količina memorije $\mathcal{O}(n^2)$ ili $\mathcal{O}(n)$ ukoliko se ne zahteva rekonstrukcija. U drugom slučaju, u jednom nizu se pamti samo poslednja izračunata dijagonala matrice c.

Implementaciju izloženog rešenja problema prepuštamo čitaocu.

4.11 Particije prirodnih brojeva

Neka je dat prirodan broj n kao i brojevi $n_1, \ldots, n_k$ takvi da važi $n_1 + \ldots + n_k = n$ i $n_1 \geq n_2 \geq \ldots \geq n_k$. Tada uređenu k-torku $(n_1, \ldots, n_k)$ nazivamo *particijom* prirodnog broja n. Neka je $p(n)$ ukupan broj particija broja n. Na primer, za $n = 6$ imamo ukupno $p(6) = 11$ particija i one su:

$$\begin{aligned}&(6), (5, 1), (4, 2), (3, 3), (4, 1, 1), (3, 2, 1), (2, 2, 2), (2, 2, 1, 1),\\ &(2, 1, 1, 1, 1), (1, 1, 1, 1, 1, 1)\end{aligned} \tag{4.11.1}$$

Za unet prirodan broj $n \in \mathbb{N}$, $n \leq 1000$ naći broj particija $p(n)$.

Uputstvo: Označimo sa $p(n, m)$ broj particija broja n čiji je najveći član (n_1) jednak m.

Rešenje: Pokazaćemo da je broj particija broja n sa najvećim članom jednakim m jednak broju particija broja $n-m$ sa najvećim članom manjim ili jednakim m. Posmatrajmo jednu particiju $(n_1, \ldots, n_k)$ broja n pri čemu je $n_1 = m$. Tada je očigledno $(n_2, \ldots, n_k)$ particija prirodnog broja $n-m$. Takođe imamo da je $m = n_1 \geq n_2$. Prema tome, na ovaj način smo definisali jedno preslikavanje. Potrebno je dokazati da je ovo preslikavanje bijekcija. Pošto iz $(n_1, \ldots, n_k) \neq (n'_1, \ldots, n'_k)$ i $n_1 = n'_1 = m$ sledi da je $(n_2, \ldots, n_k) \neq (n'_2, \ldots, n'_k)$, posmatrano preslikavanje je $1-1$. Sa druge strane, ukoliko je $(n'_2, \ldots, n'_k)$ proizvoljna $k-1$-člana particija broja $n-m$ takva da je $n'_2 \leq m$, neposrednom proverom dobijamo da je $(m, n'_2, \ldots, n'_k)$ particija broja n sa najvećim članom jednakim m.

Na osnovu ovog razmatranja možemo zaključiti da niz $p(n,m)$ zadovoljava sledeću rekurentnu relaciju:

$$p(n,m) = \sum_{k=1}^{\min\{n-m,m\}} p(n-m,k) \tag{4.11.2}$$

Gornja granica u predhodnoj sumi je takođe ograničena brojem $n-m$ zato što očigledno nijedan član (pa ni najveći) ne može biti veći od ukupnog zbira svih članova $n-m$. Startne vrednosti u formuli (4.11.2) su:

$$p(m,m) = 1, \quad m \geq 1$$

Broj $p(n)$ na kraju dobijamo kao $p(n) = \sum_{m=1}^{n} p(n,m)$. Opisani metod očigledno radi u složenosti $\mathcal{O}(n^3)$ (pošto desnu stranu relacije (4.11.2) računamo u složenosti $\mathcal{O}(n)$) i zahteva $\mathcal{O}(n^2)$ memorije (niz $p(n,m)$).

Primetimo da desnu stranu relacije (4.11.2) računamo više puta za istu vrednost $n-m$. Da bi izbegli ova suvišna izračunavanja, označimo izraz sa desne strane (4.11.2) sa $\hat{p}(n-m,m)$. Drugim rečima, neka je:

$$\hat{p}(n,m) = \sum_{k=1}^{\min\{m,n\}} p(n,k) \tag{4.11.3}$$

Prema tome, na osnovu (4.11.2) važi $p(n,m) = \hat{p}(n-m,m)$ za svako $n, m \in \mathbb{N}$ pri čemu je $m \leq n$. Pri tome, primetimo da pod istim uslovima očigledno važi i sledeća rekurentna relacija $\hat{p}(n,m) = p(n,m) + \hat{p}(n,m-1)$:

$$\hat{p}(n,m) = \hat{p}(n-m,m) + \hat{p}(n,m-1) \tag{4.11.4}$$

Relacija (4.11.4) može se takođe za izračunavanje broja $p(n)$, pri čemu je sad $p(n) = \hat{p}(n,n)$. Startne vrednosti za primenu ove relacije su $\hat{p}(n,1) = p(n,1) = 1$ (postoji samo jedna particija broja n čiji je najveći član jednak n i ona je sastavljena od svih jedinica).

I u ovom slučaju potrebno nam je $\mathcal{O}(n^2)$ memorije, pri čemu je složenost izračunavanja sada redukovana na $\mathcal{O}(n^2)$. Uz ograničenje $n \leq 1000$, ova složenost je prihvatljiva za praktična izračunavanja.

Napomenimo samo da je $p(1000) = 2.40615 \cdot 10^{31}$ pa prema tome potrebno je koristiti neku od struktura za rad sa velikim brojevima [3].

Implementaciju izloženog rešenja problema prepuštamo čitaocu.

4.12 Različiti podnizovi

Poreklo zadatka: [20]

Neka su data dva niza a i b prirodnih brojeva redom dužina m i n. Kažemo da je niz a podniz niza b ukoliko se a dobija tako što se iz b izbrišu neki elementi, tj ukoliko postoji niz indeksa $p_1 < p_2 < \cdots < p_m$ takav da je $a(t) = b(p_t)$ za svako $t = 1, \ldots, m$. Niz indeksa $p_1, \ldots, p_m$ ne mora biti jedinstven. Naći broj pojavljivanja niza a u nizu b kao podniz. Drugim rečima, naći broj različitih nizova indeksa p takvih da važe uslovi $p_1 < p_2 < \cdots < p_m$ i $a(t) = b(p_t)$ za svako $t = 1, \ldots, m$.

Uputstvo: Ovaj problem je veoma sličan problemu nalaženja najdužeg zajedničkog podniza dva data niza. Neka je $B(i, j)$ broj pojavljivanja niza $(a(1), \ldots, a(i))$ u nizu $(b(1), \ldots, b(j))$. Pronađimo rekurentnu formulu za niz $B(i, j)$.

Rešenje: Neka je $1 \leq p_1 < \cdots < p_i \leq j$ niz indeksa takav da je $a(t) = b(p_t)$ za svako $t = 1, \ldots, i$. Razmotrićemo dva slučaja.

Najpre neka je $p_t = j$. Tada je $a(i) = b(j)$, pri čemu niz indeksa $p_1, \ldots, p_{i-1}$ predstavlja jedno pojavljivanje niza $(a(1), \ldots, a(i-1))$ u nizu $(b(1), \ldots, b(j-1))$. Obrnuto ako je $p_1, \ldots, p_{i-1}$ indeksni niz nekog pojavljivanja niza $(a(1), \ldots, a(i-1))$ u nizu $(b(1), \ldots, b(j-1))$ i ako je $a(i) = b(j)$, tada ukoliko definišemo $p_i = j$ dobijamo pojavljivanje niza $(a(1), \ldots, a(i))$ u nizu $(b(1), \ldots, b(j))$. Ovakvih pojavljivanja ima $B(i-1, j-1)$.

Ukoliko je $p_t < j$ odnosno $p_t \leq j-1$ dobijamo da je u ovom pojavljivanju ceo niz $(a(1), \ldots, a(i))$ sadržan u $(b(1), \ldots, b(j-1))$ pri čemu je p odgovarajući indeksni niz. Pritom i ovde važi obrnuti smer, proizvoljno pojavljivanje pojavljivanje (sa bilo kojim indeksnim nizom p) niza $(a(1), \ldots, a(i))$ u $(b(1), \ldots, b(j-1))$ je ujedno i pojavljivanje u nizu $(b(1), \ldots, b(j))$. Ovaj zaključak važi nezavisno od toga da li je $a(i) = b(j)$ ili je $a(i) \neq b(j)$.

Prema tome, imamo da važi sledeća rekurentna formula:

$$B(i,j) = \begin{cases} B(i-1, j-1) + B(i, j-1) & , a(i) = b(j) \\ B(i, j-1) & , a(i) \neq b(j) \end{cases} \tag{4.12.1}$$

Startne vrednosti su $B(0, 0) = 1$ kao i $B(i, 0) = 0$ za $i \geq 1$. Rešenje problema je naravno $B(m, n)$.

Ovo rešenje zahteva $\mathcal{O}(mn)$ memorije ukoliko se pamti ceo niz B ili $\mathcal{O}(n)$ ukoliko se pamti samo poslednja izračunata kolona matrice B. Složenost rešenja je naravno $\mathcal{O}(mn)$.

Napomenimo samo da vrednosti niza B za određene ulazne podatke mogu biti vrlo veliki brojevi (npr. ako je $a(i) = 1$ i $b(j) = 1$ za svako $i = 1, \ldots, m$ i $j = 1, \ldots, n$

[3] Pošto je broj cifara međurezultata maksimalno 31, ukupan broj **elementarnih operacija** na računaru je jednak 31000000 za šta je savremenim računarima potrebno manje od 1 sec.

imamo da je $B(i,j) = \binom{i}{j}$), pa je potrebno koristiti neku od struktura za rad sa velikim brojevima.

Implementaciju izloženog rešenja prepuštamo čitaocu.

4.13 Ukrcavanje trajekta

Poreklo zadatka: [20]

Trajekt se sastoji od dve trake dužine l metara ($l \leq 100$) na koje mogu da se parkiraju vozila. Pošto je širina trake jednaka širini vozila, ona moraju biti parkirana jedno iza drugog. U redu za ukrcavanje čeka n vozila ($n \leq 1000$), pri čemu je dužina i-tog vozila a_i centimetara ($a_i \in \mathbb{N}$, $a_i \leq A = 3000$). Vozila se moraju ukrcavati redom, u redosledu u kome čekaju. Radnik na trajektu može svako vozilo uputiti u jednu od dve trake, pri čemu je cilj ukrcati što više vozila. Pritom svako vozilo koje dođe na red mora biti ukrcano, ukoliko za to postoje mogućnosti (tj ima mesta na nekoj od traka). Naći maksimalan broj k takav da je moguće na opisani način ukrcati u trajekt prvih k vozila. Odrediti jedno od mogućih ukrcavanja ovih vozila.

Uputstvo: Neka je $V(l_1, i)$ logički niz koji ima vrednost $\top$ ukoliko je moguće ukrcati prvih i vozila tako da na prvoj traci ostane slobodno l_1 centimetara, a u suprotnom vrednost $\bot$. Rešenje problema se sada dobija kao

$$\max\{k \mid V(t,k) = \top, \text{ za neko } t \in \{1,2,\ldots,l\}\}.$$

Rešenje: Predpostavimo da postoji način za ukrcavanje vozila $1, 2, \ldots, i$ u trajekt. Razmotrićemo dva slučaja.

Neka se i-to vozilo ukrcava u 1. traku. Pošto je nakon ukrcanja ovog vozila slobodno l_1 centimetara, pre ukrcavanja mora biti slobodno $l_1 + a_i$ centimetara. Naravno, pre i-tog ukrcali smo sva vozila do $i-1$-vog, pa je prema tome $V(l_i + a_i, i-1) = \top$. Važi i obrat, ukoliko je $V(l_i + a_i, i-1) = \top$ sledi da i-to vozilo možemo ukrcati u 1. traku pa je i $V(l_1, i) = \top$.

Neka se sada i-to vozilo ukrcava u 2. traku. U drugoj traci je ukupno zauzeto $l_2 = \sum_{j=1}^{i} a_i - l_1$ centimetara. Ovaj broj mora biti ne veći od l. Pre ukrcavanja i-tog vozila, u 1. traci je bilo takođe l_1 slobodnih centimetara pa zaključujemo da mora da važi $V(l_1, i-1) = \top$. Važi i obrat. Ukoliko je $l_2 \leq l$ i $V(l_1, i-1) = \top$, i-to vozilo možemo ukrcati u drugu traku pa dobijamo da je i $V(l_1, i) = \top$.

Prema tome, ovim smo pokazali da važi sledeća rekurentna formula za niz $V(l_1, i)$:

$$V(l_1, i) = V(l_1 + a_i, i-1) \wedge (V(l_1, i-1) \vee l_{2i} \leq l) \tag{4.13.1}$$

gde je:

$$l_{2i} = s(i) - l_1, \quad s(i) = \sum_{j=1}^{i} a_i. \tag{4.13.2}$$

Startne vrednosti za $V(l_1, i)$ su $V(l, 0) = \top$ i $V(l_1, 0) = 0$ za $l_1 < l$ (ukoliko nemamo nijedno vozilo, prva traka je cela prazna). Sada samo treba pronaći maksimalnu vrednost indeksa i takvu da je $V(l_1, i) = \top$ bar za jednu vrednost l_1. Ova maksimalna vrednost rešenje problema.

Da bi rekonstruisali rešenje, potrebno je pamtiti u nizu $P(l_1, i)$ koji član u disjunkciji sa desne strane izraza (4.13.1) ima vrednost $\top$. Ukoliko su oba člana $\perp$ onda je $P(l_1, i) = 0$. Primetimo da je vrednost $V(l_1, i)$ potpuno određena pomoću vrednosti $P(l_1, i)$, pa je dovoljno pamtiti samo niz P. Rekonstrukcija se naravno obavlja rekurzivno kao i kod predhodnih problema.

Dakle ovo rešenje zahteva $\mathcal{O}(nl)$ memorije i ima složenost takođe $\mathcal{O}(nl)$. Ukoliko je potreban samo broj vozila a nije bitan raspored, onda je dovoljno pamtiti poslednju izračunatu kolonu niza V (niz P uopšte nije potrebno pamtiti), pa su memorijski zahtevi redukovani na $\mathcal{O}(l)$.

Implementaciju izloženog rešenja prepuštamo čitaocu.

4.14 Novopečeni bogataš

Lokalni novopečeni bogataš, kupio je poveliko kvadratno parče zemlje, na njemu želi da izgradi što je moguće veću kuću i to tako da njena osnova bude kvadratnog oblika sa stranama paralelnim stranama placa. Mapa imanja je sastavljena iz kvadratića dimenzija 1×1. Jedno drvo zauzima tačno jedan kvadratić. Pošto je imanje veliko, vrlo je teško naći najveći kvadrat čija je strana najduža na kome nema stabala. Dužina strane je broj kvadratića 1×1 duž njegove strane.

Potrebno je izračunati dužinu stranice najvećeg kvadrata koji ne sadrži ni jedno drvo. Predpostaviti da je veličina imanja $n \leq 100$ kvadrata.

Uputstvo: Neka je $K(i, j)$ maksimalna dimenzija kvadrata koji ne sadrži nijedno drvo i čiji je donji desni ugao kvadratić sa koordinatama (i, j). Potrebno je naći rekurentnu formulu za niz $K(i, j)$.

Rešenje: Ukoliko je na polju (i, j) locirano drvo, onda je trivijalno $K(i, j) = 0$. Predpostavimo sada da na polju (i, j) nema drveta. Posmatrajmo maksimalne kvadrate koji ne sadrže drvo i čiji su donji desni uglovi $(i-1, j)$, $(i, j-1)$ kao i $(i-1, j-1)$. Neka su dužine njihovih stranica redom k_1, k_2 i k_3. Tada je očigledno $K(i, j) \leq k_1, k_2, k_3 + 1$ jer bi u suprotnom odgovarajući kvadrat čija je stranica k_i, $i = 1, 2, 3$ mogao biti veći na račun kvadrata sa temenom (i, j). Neka je $k = \min\{k_1, k_2, k_3 + 1\}$. Dokažimo da je $K(i, j) \geq k$. Zaista, pošto je $k \leq k_1$ sva polja sa koordinatama $(i - p - 1, j - q)$ gde je $p, q \leq k$ ne sadrže drvo. Isto važi i za polja $(i - p, j - q - 1)$. Napokon, pošto je $k \geq k_3 + 1$ imamo da je i polje $(i - k, j - k)$ slobodno. Time smo dokazali da je $K(i, j) \geq k$. Prema tome važi sledeća rekurentna formula:

$$K(i,j) = \begin{cases} \min\{K(i-1,j), K(i,j-1), K(i-1,j-1)+1\}, & \text{na polju } (i,j) \\ & \text{nije drvo} \\ 0, & \text{u suprotnom} \end{cases}$$

Startne vrednosti za niz $K(i,j)$ su $K(1,j)$ i $K(i,1)$. Ukoliko je drvo locirano na nekoj od ovih lokacija, odgovarajuća vrednost elementa niza K je 0, u suprotnom je 1. Konačno rešenje dobijamo jednostavno kao $\max_{1\leq i,j\leq n} K(i,j)$.

Što se tiče memorijskih zahteva, ovo rešenje zahteva $\mathcal{O}(n^2)$ memorije (niz kojim se identifikuje da li je na polju (i,j) drvo i niz K). Medjutim, ukoliko se predpostavi da je broj drveća na plantaži jednak $\mathcal{O}(n)$, izloženo rešenje se može modifikovati tako da zahteva ukupno $\mathcal{O}(n)$ memorije. Modifikacija se postiže pamćenjem samo poslednje izračunate vrste niza K. U oba slučaja složenost rešenja je $\mathcal{O}(n^2)$.

Implementacija:

```
program sumadija;
 type matrica=array[1..100,1..100] of integer;
 var  a:matrica;
      n,i,p,j,x,y:integer;
function  min(a,b,c:integer):integer;
 var p:integer;
 begin
   if a>b then p:=b else p:=a;
   if c>p then min:=p else min:=c;
 end;
 begin
   write('Unesi velicinu imanja');     readln(n);
   write('Unesite broj drveca na imanju');   readln(p);
   write('unesite koordinate drveca');
   for i:=1 to n do
       for j:=1 to n do   a[i,j]:=1;
   for i:=1 to p do
         begin
           readln(x,y);      a[x,y]:=0;
         end;
   x:=0;
   for i:=2 to n do for j:=2 to n do
     begin
         a[i,j]:=a[i,j]+a[i,j]*min(a[i-1,j],a[i-1,j-1],a[i,j-1]);
         if a[i,j]>x then x:=a[i,j];
     end;
   writeln('Duzina najvece dimenzije kuce je P= ',x);
 end.
```

4.15 Maksimalna suma nesusednih u nizu

Poreklo zadatka: [25].

Dat je niz A od N celih brojeva. Odrediti podniz niza A čiji zbir elemenata je maksimalan, a u kome nema susednih elemenata niza A. Smatrati da prazan niz ima zbir elemenata nula.

Rešenje: Neka je za niz A poznat jedan takav podniz $P = P(A)$. Ako element a_N ne pripada podnizu P, onda je P optimalni podniz i za niz A_{N-1}. Ako element a_M pripada podnizu P, onda je podniz P bez poslednjeg elementa (tj. bez a_N) optimalan za niz A_{N-2}. U protivnom bi bilo moguće poboljšati optimalni podniz P niza A, tako što se na bolji podniz niza A_{N-2} dopiše broj a_N. Ustanovili smo optimalnost podstrukture problema.

Rešenje za nizove A_0 i A_1 su trivijalna. Za X od 2 do N, $P(A_X)$ je onaj od nizova $P(A_{X-1})$ i $P(A_{X-2}) \cup \{a_X\}$, koji ima veći zbir.

Formiraćemo niz B, tako da je $B(X)$ zbir elemenata optimalnog podniza $P(A_X)$. Prema predhodnom, sledi

$$B(0) = 0,\ B(1) = \max(0, a_1),$$
$$B(X) = \max\{B(X-1), B(X-2) + a_X\},\ X = 2, \ldots, N.$$

Na osnovu niza B jednostavno se pronalazi podniz P.

Implementacija:

```
Program Maksimalna_Suma_Nesusednih;
  var a,b:array[0..1000]of integer;
      n,i:integer;
      f:text;

  procedure ispis(n:integer);
    begin
      if n>0 then
      if b[n]=b[n-1] then ispis(n-1)
      else
        begin
          ispis(n-2);  write(a[n]:4); write(f,a[n]:4);
        end;
    end;

  begin
    assign(f,'input.dat');
    reset(f);
    readln(f,n);
    for i:=1 to n do readln(f,a[i]);
    close(f);
    b[0]:=0;
    if a[1]>0 then b[1]:=a[1] else b[1]:=0;
    for i:=2 to n do
      if b[i-2]+a[i]>b[i-1] then b[i]:=b[i-2]+a[i]
    else b[i]:=b[i-1];
    assign(f,'output.dat');
    rewrite(f);
    ispis(n);
    writeln(f);writeln(f,b[n]);
    close(f);
    writeln;  writeln(b[n]);
  end.
```

4.16 Svi najjeftiniji putevi

Poreklo zadatka: [25].

U jednoj zemlji ima N gradova. Svi gradovi su povezani putevima, mada ne obavezno direktnim. Nema gradova koji su spojeni sa više od jednog direktnog puta. Za svako putovanje od grada do grada poznata je (pozitivna) cena,koja ne mora biti srazmerna dužini puta. Cene su date matricom D od N vrsta i N kolona. Ako između gradova i i j ne postoji put, cena takvog puta je beskonačna. Za

praktične potrebe najpogodnije je predstaviti da je tada u matrici cena $D(i,j)$ upisan zbir cena svih postojećih puteva ili još neki veći broj. Treba odrediti najniže cene putovanja za svaki par gradova, kao i odgovarajuće maršute.

Rešenje: Ovaj problem se rešava vrlo poznatim algoritmom Flojda,koji na prvi pogled nema veze sa dinmičkim programiranjem,zbog neočekivanog načina zadavanja podproblema. Prema do sada izloženim problemima i rešenjima, moglo bi se očekivati da se veličina podproblema zada brojem gradova između kojih treba naći najkraće puteve. Podproblem veličine X bi bio nalaženje najjeftinijih puteva između prvih X gradova u nekom odabranom redosledu. Tada bi novi grad trebalo pokušati umetnuti na svaki od puteva, na svakom od mesta u tom putu, a posebne teškoće zadalo bi nalaženje najkraćih puteva od novog grada do ostalih gradova. Sve to zahteva mnogo operacija. Kako onda upotrebiti dinamičko programiranje?

Konstatujmo najpre nekoliko činjenica:

Na najjeftinijem putu od grada u do grada v gradovi mogu da se pojave najvše jednom. U protivnom bi bilo moguće pjeftiniti put izbacivanjem petlje, što je u kontradikciji sa pretpostavkom da je taj put najjeftiniji.

Ako je put P najjeftiniji put od u do v, i put P sadrži čvor W, onda je deo puta P od u do w najjejftiniji put od u do w a deo puta P od w do v je najjeftiniji put od w do v, što se jednostavno dokazuje svpđenjem na kontradikciju. U ovom tvrđenju se ogleda optimalnost podstrukture problema. Formulišimo isto zapažanje na nešto pogodniji način: neka je određen optimalan (najjeftiniji) put za svaki par gradova. Neka optimalan put P od u do v ne sasdrži grad N. Tada je taj put od u do v optimalan put koji sadrži samo (neke od) prvih $N-1$ gradova za presedanje. Ako optimalan put P od u do v sadrži grad N, onda su putevi od u do N, i od N do v optimalni putevi koji koriste samo prvih $N-1$ gradova za presedanje (jer se na putu P grad N može pojaviti samo jednom (jer je put P optimalan), pa su optimalni i njegovi delovi). Sada je jasno da veličina podproblema treba da se zada brojem gradova preko kojih se sme presedati. Podproblem veličine K je nalaženje (za svaki par gradova) optimalnih puteva, koristeći samo u prvih K gradova.

Dalje rešavanje je jednostavno. Neka je D_k matrica dužina optimalnih putevakoji kao među čvorove koriste samo prvi K čvorova. Tada je

$$\begin{aligned} D_0(i,j) &= D(i,j) \\ D_K(i,j) &= \min\{D_{K-1}(i,j), D_{K-1}(i,K) + D_{K-1}(K,j)\}, K = 1 \ldots N \end{aligned}$$

Tražena matrica najjeftinijih puteva je D_N. Da bismo mogli da rekonstruišemo i same puteve,koristićemo i matricu C. Na početku $C(i,j) = 0$ za sve i,j. Kada se pri nekom K promeni $D(i,j)$, stavimo $C(i,j) = K$, čime smo zapamtili da optimalan put (u K-tom podproblemu) od i do j ide preko K. Rekurzivna procedura ispis (i,j) ispisuje brojeve svih gradova na putu od i do j (osim samih i i j).

Na ovom primeru se vidi da iako su problemi koje rešavamo slični, ne treba ih rešavati po inerciji. U nekim problemima mogu se uočiti hijerarhije podproblema raznih vrsta. Potrebno je, međutim (najčešće kroz analizu strukture optimalnog rešenja) doći do takvih podproblema, čija rešenja su sadržana u rešenju većeg problema, i čijim se kombinovanjem rešenje većeg problema može dobiti.

Što se tiče memorijskih zahteva, ovo rešenje zahteva $\mathcal{O}(n^2)$ memorije. Složenost rešenja je $\mathcal{O}(n^3)$.

Implementacija:

```
program Svi_najjeftiniji_putevi;
 uses crt,dos;
 var d,c:array[1..50,1..50] of integer;
     n,i,j,k:integer;
 procedure ispis(a,b:integer);
    begin
     if c[a,b]>0 then
       begin
        ispis(a,c[a,b]);write(c[a,b]:5);ispis(c[a,b],b);
       end;
    end;
   begin
     textcolor(1);write('broj gradova?');textcolor(2);read(n);
     for i:=1 to n do
      for j:=1 to n do
      begin
        textcolor(3);write('od',i,'do',j,':');textcolor(6);read(d[i,j]);  c[i,j]:=0;
      end;
     readln;
     for k:=1 to n do
      for i:=1 to n do
       for j:=1 to n do
        if d[i,j]>d[i,k]+d[k,j] then
        begin
         d[i,j]:=d[i,k]+d[k,j];    c[i,j]:=k;
        end;
     textcolor(9);writeln('Optimalne cene puteva:');
     for i:=1 to n do
     begin
      for j:=1 to n do  begin
                          textcolor(14);   write(d[i,j]:5);
                        end;
        writeln;
     end;
     textcolor(4); writeln('Koji put zelite da rekonstruisete u celosti:');
     textcolor(13);write('i= ');textcolor(12);readln(i);
     textcolor(13);write('j=');textcolor(12);readln(j);
     textcolor(7);writeln(i:5);
     textcolor(8);ispis(i,j);
     textcolor(11);writeln(j:5);
 end.
```

4.17 Roman

Poreklo zadatka: [25].

U romanu ima N glava. U i-toj glavi ima a_i stranica. Treba izdati roman u K tomova tako da broj stranica najdebljeg toma bude minimalan. Glave se ne mogu deliti. Naći broj glava u svakom tomu, kao i broj stranica najdebljeg toma.

Rešenje: Označimo sa $B(X, Y)$ broj stranica najdebljeg toma u optimalnom rastavu prvih X glava na Y tomova. Tako je, na primer $B(X, 1)$ jednako zbiru prvih

X elemenata niza A, a $B(N,K)$ je vrednost koja se traži.

Predpostavimo da je dato jedno optimalno rastavljanje N glava na K tomova, i da u optimalnom rastavu romana, u poslednji tom ulazi g glava.Tada je broj strana najdebljeg toma $B(N,K)$ jednak sumi brojeva stranica poslednjih g glava ako je poslednji tom najdeblji, a najdebljem od preostalih $K-1$ tomova inače. Primetimo da tada (ako poslednji tom nije najdeblji) rastavljanje prvih $K-1$ tomova na $N-g$ glava mora takođe biti optimalno, inače bi se moglo popraviti rešenje glavnog zadatka. Zbog toga je debljina najdebljeg od prvih $K-1$ tomova jednaka $B(n-g,K-1)$. Kraće zapisano, važi:

$$B(N,K) = \max\left\{\sum_{i=N-g+1}^{N} A_i, B(N-g,K-1)\right\}$$

Primetimo da ako je poslednji tom u optimalnom rešenju za K tomova najdeblji, prethodni tomovi u optimalnom rešenju NE MORAJU biti formirani na optimalan način. Dovoljno je da ni jedan tom ne prelazi debljinu K-tog, a ne da najdeblji $K-1$ prvih bude što tanji. Međutim, ova konstatacija ne utiče na optimalnost podstrukture! Da bi problem mogao da se rešava dinamičkim programiranjem, potrebno je da se jedno optimalno rešenje problema može dobiti kombinovanjem optimalnih rešenja podproblema, a ne da je svako optimalno rešenje takvo. Jasno je da i u slučaju kada je u optimalnom rešenju poslednji tom najdeblji, prethodni tomovi MOGU biti formirani na optimalan način. Dinamičkim programiranjem pronaći ćemo ono optimalno rešenje koje u sebi sadrži optimalna rešenja podproblema.

Do rešenja dolazimo tako sto izračunamo sve $B(X,Y)$ redom: najpre za $Y=1$ imamo

$$B(X,1) = \sum_{i=1}^{x} A_i zaX = 1,N$$

za $X=1,N$ (jer sve glave idu u jedan tom), a zatim za $Y=1$ probamo za svako smisleno g (takvo da za svaki tom preostane bar jedna glava) da stavimo g glava u poslednji tom, tražeći pri tome ono g za koje se dobija najmanja debljina najdebljeg toma. Tako dobijemo:

$$\begin{aligned} B(X,Y) &= \min_{1\le g\le X+1-Y}\left\{\max\left\{B(Y-g,Y-1), \sum_{i=X-g+1}^{X} A_i\right\}\right\}, \\ & 2\le Y\le K, 1\le X\le N. \end{aligned}$$

Popunjavajući matricu B redom, dobićemo traženu debljinu kao $B(N,K)$. Da bismo znali i broj glava u pojedinim tomovima, potrebno je da zapamtimo za koje g je dobijen navedeni minimum. U tu svrhu formiraćemo matricu C, iste veličine kao B, tako da je $C(X,Y)$ jednako onom g koje ostvaruje navedeni minimum. Nakon formiranja matrice C, treba rekonstruisati brojeve glava u pojedinim tomovima. To se najlakše postiže rekurzivnom procedurom, koja se poziva ukupno K puta (onoliko puta koliko ima tomova):

Primetimo i to, da kad već vršimo rekonstrukciju na osnovu matrice C nije nam neophodna cela matrica B, već samo poslednja cela kolona i kolona koja se formira. Ako bismo kolone popunjavali s kraja (za X od N do 1), B se može zameniti jednim nizom, čime bismo zauzeli znatno manje memorije.

Implementacija:

```
program roman;
 uses crt,dos;
  var
   b,c:array[1..150,1..20] of integer;
   a:array[1..150] of integer;
   k,n,x,y,g,f:integer;
    function max(a,b:integer):integer;
     begin
      if a>b then max:=a
             else max:=b
     end;
    procedure ispis(x,y:integer);
     begin
      if y>0 then
       begin
        ispis(x-c[x,y],y-1);
        writeln('U tom  ' , y , ' ide ' ,c[x,y],  'gl.');
       end;
     end;
    begin
     textcolor(1);write('broj glava n=');textcolor(2);readln(n);
     textcolor(1);write('broj tomova k=');textcolor(2);readln(k);
     for x:=1 to n do
      begin
       textcolor(3);write('a[',x,']=');  textcolor(5);readln(a[x]);
      end;
     b[1,1]:=a[1];   c[1,1]:=1;
     for x:=2 to n do
     begin
      b[x,1]:=b[x-1,1]+a[x];    c[x,1]:=x;
     end;

     for y:=2 to k do
      for x:=1 to n do
       begin
        b[x,y]:=max(b[x-1,y-1],a[x]);   c[x,y]:=1;
        for g:=2 to x+1-y do
        begin
         f:=max(b[x-g,y-1],b[x,1]-b[x-g,1]);
         if b[x,y]>f then
         begin
          b[x,y]:=f;  c[x,y]:=g;
         end;
        end;
       end;
    textcolor(4);writeln('Broj strana u najdebljem tomu je',b[n,k],'.');
    textcolor(14);ispis(n,k);
 end.
```

5 Apsolutno neklasični problemi dinamičkog programiranja

U ovom odeljku razmotrićemo nekoliko primena tehnike DP u problemima koji na prvi pogled nemaju nikakve sličnosti sa klasičnim problemima u kojima se ova tehnika primenjuje. Ova konstatacija se naročito odnosi na problem izračunavanja generalisanih inverza matrica.

5.1 Primena dinamičkog programiranja na rešavanje problema trgovačkog putnika

Graf se predstavlja uređenim parom $G=(N,L)$, gde je $N=\{1,\ldots,n\}$ skup čvorova, i $L \subseteq \{(i,j)|\ i \in N, j \in N\}$ skup grana. Ako se elementima čvorova N i/ili elementima skupa grana L konačnog grafa $G=(N,L)$ pridruže određeni brojevi ili funkcije, takav graf se naziva *mreža*. Dužinu svake grane (i,j) ćemo označavati sa c_{ij}. Oznaka $\Gamma(i)$ predstavlja skup čvorova koji slede čvor i, tj. $\Gamma(i)=\{j \in N|\ (i,j) \in L\}$, dok oznaka $\Gamma^{-1}(i)$ predstavlja skup čvorova koji prethode čvoru i, tj. $\Gamma^{-1}(i)=\{j \in N|\ (j,i) \in L\}$. Početni i završni čvor se obeležavaju sa s i t, redom (videti, na primer [17, 26, 7]).

Zadatak nalaženja najkraće Hamiltonove konture na zadatoj mreži naziva se **problem trgovačkog putnika** (Traveling Salesman Problem, **TSP**). Hamiltonova kontura je put koji kroz sve čvorove grafa prolazi jednom i samo jednom i završava se u početnom čvoru:

$$hk=(i_1,i_2,\ldots,i_{n-1},i_n,i_1),\ i_p \neq i_k,\ \forall p,k \in \{1,\ldots,n\}, p \neq k.$$

Nalaženje najkraće Hamiltonove konture liči na realni zadatak trgovačkog putnika koji planira obilazak višše gradova, pri čemu polazi iz jednog grada i vraća se u isti.

Matematički model problema se formira na sledeći način.

Data je mreža $G=(N,L)$ sa dužinama grana c_{ij}, za svako $(i,j) \in L$, dok su $c_{ij}=\infty$ za $(i,j) \notin L$. Bez umanjenja opštosti, možemo usvojiti da je čvor 1 polazni i završni čvor. Definisaćemo skup $Q \subseteq N\backslash\{1\}$, kao bilo koji podskup čvorova grafa koji ne sadrži čvor 1. Označićemo sa $f(Q,j)$, $j \notin Q$ dužinu najkraćeg puta koji polazi od čvora 1, prolazi kroz sve čvorove iz skupa Q i završava u čvoru j. Po definiciji Hamiltonove konture, najkraća Hamiltonova kontura imaće dužinu $f(N\backslash\{1\},1)$ tj. biće jednaka dužini najkraćeg puta koji polazi od prvog čvora, prolazi kroz sve čvorove osim prvog i završava se u prvom čvoru.

Na osnovu ovoga se može definisati sledeća rekurentna formula dinamičkog programiranja [17, 26]:

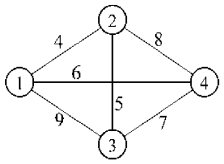
$$\begin{aligned}(\forall Q \subseteq N\backslash\{1\})(\forall j \in N\backslash Q)\quad f(Q,j) &= \min_{i \in Q}\{f(Q\backslash\{i\},i)+c_{ij}\},\\ f(\emptyset,j) &= c_{1j}.\end{aligned}$$

Na osnovu prethodnog, optimalno rešenje dobija se dobija kao rešenje sledećeg problema:

$$f(N\backslash\{1\},1) = \min_{i\in N\backslash\{1\}} \{f(N\backslash\{i,1\},i) + c_{i1}\}.$$

Sledeći primer je preuzet iz [26].

Primer 5.1.1 Odrediti najkraću Hamiltonovu konturu za zadatu mrežu.



Slika 2.4.1. Prikaz zadate mreže.

Rešenje: S obzirom da je nulta etapa trivijalna, počećemo od prve:

$$\begin{aligned}
f(\{2\},3) &= f(\emptyset,2) + c_{23} = c_{12} + c_{23} = \underline{4+5} = 9\\
f(\{3\},2) &= f(\emptyset,3) + c_{32} = c_{13} + c_{32} = 9 + 5 = 14\\
f(\{2\},4) &= f(\emptyset,2) + c_{24} = c_{12} + c_{24} = 4 + 8 = 12\\
f(\{4\},2) &= f(\emptyset,4) + c_{42} = c_{14} + c_{42} = 6 + 8 = 14\\
f(\{3\},4) &= f(\emptyset,3) + c_{34} = c_{13} + c_{34} = 9 + 7 = 16\\
f(\{4\},3) &= f(\emptyset,4) + c_{43} = c_{14} + c_{43} = \underline{6+7} = 13.
\end{aligned}$$

Druga etapa se može izraziti sledećim jednačinama:

$$\begin{aligned}
f(\{2,3\},4) &= \min_{i\in\{2,3\}} \left\{ \begin{array}{c} f(\{3\},2) + c_{24} \\ f(\{2\},3) + c_{34} \end{array} \right\} = \min \left\{ \begin{array}{c} 14+8 \\ \underline{9+7} \end{array} \right\} = 16\\
f(\{2,4\},3) &= \min_{i\in\{2,4\}} \left\{ \begin{array}{c} f(\{4\},2) + c_{23} \\ f(\{2\},4) + c_{43} \end{array} \right\} = \min \left\{ \begin{array}{c} 14+512+7 \end{array} \right\} = 19\\
f(\{3,4\},2) &= \min_{i\in\{3,4\}} \left\{ \begin{array}{c} f(\{4\},3) + c_{32} \\ f(\{3\},4) + c_{42} \end{array} \right\} = \min \left\{ \begin{array}{c} \underline{13+5} \\ 16+8 \end{array} \right\} = 18
\end{aligned}$$

I na kraju treća, završna etapa u kojoj dobijamo konačno rešenje, opisuje se sledećim jednakostima:

$$\begin{aligned}
f^* &= f(2,3,4,1) = \min_{i\in\{2,3,4\}} f(N\backslash\{1,i\},i) + c_{i1}\\
&= \min \left\{ \begin{array}{c} f(\{3,4\},2) + c_{21} \\ f(\{2,4\},3) + c_{31} \\ f(\{2,3\},4) + c_{41} \end{array} \right\} = \min \left\{ \begin{array}{c} \underline{18+4} \\ 19+9 \\ \underline{16+6} \end{array} \right\} = 22.
\end{aligned}$$

Najkraća Hamiltonova kontura zadate mreže ima dužinu 22. Redosled čvorova se dobija prateći unazad dobijene minimalne vrednosti (koje su podvučene). U zadatku imamo dva najkraća puta i to: (1, 2, 3, 4, 1) i (1, 4, 3, 2, 1).

Napomena 5.1.1 *U simetričnim mrežama se uvek dobija paran broj najkraćih Hamiltonovih kontura. Razlog je jasan: jednu konturu je moguće obići u oba smera.*

Osim problema nalaženja najkraće Hamiltonove konture, isti postupak se može primeniti i na nalaženje najkraćeg Hamiltonovog puta kroz mrežu. Hamiltonov put se definiše kao put koji polazi od početnog, prolazi kroz ostale čvorove samo jedanput, i završava se u nekom završnom čvoru:

$$hp = (s, i_1, \ldots, i_{n-1}, t),\ i_p \neq i_k,\ \forall p,k \in \{1,\ldots,n\}, p \neq k.$$

Postoje različite varijante problema najkraćeg Hamiltonovog puta kroz mrežu. Poznata je varijanta u kojoj su zadati početni i završni čvor, ili samo jedan od njih. Ove varijante se mogu rešiti opisanim postupkom dinamičkog programiranja uz male modifikacije.

5.2 Dinamički transportni problem

Određenu klasu transportnog problema (nelinearni problemi) čine problemi kod kojih se prevoz tereta obavlja transportnim sredstvom koje snabdeva robom više odredišta iz samo jednog ishodišta. Kod ovih problema kriterijum optimalnosti može biti definisan preko minimalne ukupne dužine puta, minimalnih troškova ili vremena putovanja. Primena dinamičkog programiranja u problemima transporta dala je mogućnost optimizacije troškova prevoza i u slučaju nelinearnih problema transporta. Nelinearni transportni problemi spadaju u grupu problema koji se moraju dekomponovati u niz etapa kako bi se problem mogao rešiti dinamičkim programiranjem. Korišćenje diskretnog dinamičkog programiranja omogućava i rešavanje onih transportnih problema kod kojih se može u prevozu javiti i nehomogeni teret. U ovakvim slučajevima postavlja se zahtev prevoza maksimalne vrednosti (cena, prioritet) različitog tereta, uz što veće iskorišćenje kapaciteta transportnog sredstva. U nastavku će biti razmotrena upravo ova klasa problema.

Neka je poznato transportno sredstvo (kamion, vagon, avion, brod i slično) nosivosti (kapaciteta) Q mernih jedinica. U ovo sredstvo potrebno je utovariti n različitih tipova predmeta sa različitom "vrednošću" (cenom, prioritetom). U razmatrano transportno sredstvo potrebno je utovariti predmete maksimalne "vrednosti", pri čemu P_i predstavlja težinu jednog predmeta i-tog tipa ($i = 1, 2, \ldots, n$), C_i "vrednost" predmeta i-tog tipa, a x_i je broj predmeta i-tog tipa.

Matematički zapis modela problema transporta je sledeći:

$$\max \quad F(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} C_i x_i$$

$$\text{p.o.} \quad \sum_{i=1}^{n} P_i x_i \leq Q, \; x_i = 0, 1, 2, 3, \ldots$$

Pri utovaru u transportno sredstvo predmeta samo prvog tipa maksimalna "vrednost" utovara je:

$$f_1(Q) = \max\{C_1 x_1\}$$

uz ograničenje:

$$P_1 x_1 \leq Q, \; x_1 = 0, 1, 2, 3, \ldots$$

Broj predmeta prvog tipa utovarenih u transportno sredstvo, dat je najvećim celim brojem koji ne prelazi Q/P_1:

$$x_1 = \lfloor Q/P_1 \rfloor$$

Pri određivanju maksimalne "vrednosti" problema polazi se od relacije:

$$f_1(Q) = \lfloor Q/P_1 \rfloor C_1.$$

Pri utovaru u transportno sredstvo predmeta prvog i drugog tipa maksimalna "vrednost" utovara se označava sa $f_2(Q)$. Ako je u ovom slučaju utovareno x_2 predmeta drugog tipa, tada težina predmeta prvog tipa ne sme preći težinu $Q-P_2x_2$ dok je "vrednost" jednaka

$$C_2x_2 + f_1(Q - P_2x_2).$$

Maksimalna "vrednost" utovara može se izračunati upotrebom rekurzivne relacije:

$$f_2(Q) = \max_{0\leq x_2\leq \lfloor \frac{Q}{P_2} \rfloor} \{C_2x_2 + f_1(Q - P_2x_2)\}.$$

Nastavljanjem ovog procesa dolazi se nakon n koraka do rekurzivne relacije:

$$f_n(Q) = \max_{0\leq x_n\leq \lfloor \frac{Q}{P_n} \rfloor} \{C_nx_n + f_{n-1}(Q - P_nx_n)\}$$

pri čemu je:
$f_n(Q)$ - maksimalna vrednost utovara sastavljenog upotrebom n tipova predmeta;
C_nx_n - "vrednost" koja odgovara upotrebi x_n predmeta n-tog tipa
$f_{n-1}(Q-P_nx_n)$ - maksimalna "vrednost" utovara sastavljena od $n-1$ tipova predmeta čija težina nije veća od $Q - P_nx_n$.

Ovako definisan model transporta može se koristiti u prevozu homogenog i nehomogenog tereta kada je poznato:
- kapacitet transportnog sredstva Q,
- težina P_j za jedinični prevoz j-tog tipa tereta,
- broj x_i predmeta tipa i,
- "vrednost" ("cena", "prioritet") po jedinici predmeta tipa i, te gabariti pojedinog tipa predmeta.

Poznavanje ovih parametara omogućava postizanje maksimalne "vrednosti" (cene, prioriteta) prevoza uz maksimalno korišćenje kapaciteta (nosivosti) transportnog sredstva. Primena opisanog modela biće pokazana primeru koji je preuzet iz [3].

Primer 5.2.1 Zadate su četiri vrste roba pakovane u sledećim težinskim ili drugim jedinicama: $P_1 = 11, P_2 = 9, P_3 = 7$ i $P_4 = 6$ (tona, m^3, komada i sl.). "Prioritet" ("vrednost") ovih predmeta (npr. paleta) ocenjen je od stručnih lica i iznosi: $C_1 = 40, C_2 = 38, C_3 = 26$ i $C_4 = 19$ (bodova, poena). Zadatak se sastoji u maksimiziranju "vrednosti" prevoza uz što veće korišćenje kapaciteta (nosivosti) transportnog sredstva. Transportno sredstvo ima nosivost od 25 težinskih jedinica.

Rešenje: Neka se u transportno sredstvo utovaraju predmeti prvog tipa, tj. roba koja je pakovana (paletizovana) u pakete težine $P_1 = 11$ težinskih jedinica. Ovih predmeta moguće je utovariti u količini $[Q/P_1] = [25/11] = 2$, pa je $x_1 = 0; 1; 2$ predmeta. Budući da je težina tereta $P_1 = 11$ težinskih jedinica, to kod kapaciteta transportnog sredstva $0 \leq Q \leq 10$ težinskih jedinica nije moguće utovariti ni jedan predmet prvog tipa. Jedan predmet je moguće utovariti kod kapaciteta transportnog sredstva $11 \leq Q \leq 21$ težinskih jedinica, a dva predmeta kod kapaciteta $22 \leq Q \leq 25$ težinskih jedinica. Izvedeni proračun za Q, $f_1(Q)$ i x_1 unet je u sledeću tabelu.

Q	$f_1(Q) = \lfloor Q/P_1 \rfloor C_1$	$x_1 = \lfloor Q/P_1 \rfloor$
0 - 10	0	0
11 - 21	40	1
22 - 25	80	2

Tabela 2.5.1.

Neka se u sledećoj etapi u transportno sredstvo utovaraju predmeti prvog i drugog tipa. Kako je $P_2 = 9$ težinskih jedinica, to x_2 ima moguće vrednosti 0;1;2 jer je $\lfloor Q/P_2 \rfloor = \lfloor 25/9 \rfloor = 2$. Kapacitet transportnog sredstva bi trebalo podeliti na intervale u zavisnosti od vrednosti parametara P_1 i P_2, a zatim korišćenjem relacije

$$f_2(Q) = \max_{0 \le x_2 \le \lfloor \frac{Q}{P_2} \rfloor} \{C_2 x_2 + f_1(Q - P_2 x_2)\}.$$

i činjenice da je $x_2 = 0; 1; 2$ formirati vrednosti funkcije $f_2(Q)$ za utovar predmeta prvog i drugog tipa. Pri kapacitetu $0 \le Q \le 8$ nije moguće u transportno sredstvo utovariti predmete ni prvog ni drugog tipa. Kod kapaciteta $9 \le Q \le 10$ primena formule za $f_2(Q)$ daje maksimalnu vrednost 38 (poena, bodova):

$$\begin{aligned} f_2(9) &= \max_{0 \le x_2 \le \lfloor \frac{9}{9} \rfloor} \{38x_2 + f_1(9 - 9x_2)\} \\ &= \max \left\{ \begin{array}{l} 38 \cdot 0 + f_1(9 - 9 \cdot 0) = 0 \\ 38 \cdot 1 + f_1(9 - 9 \cdot 1) = 38 \end{array} \right\} = 38, \end{aligned}$$

što znači da se pri ovom kapacitetu utovaruje samo jedan predmet drugog tipa.

Urađen je sličan proračun za svaki interval do nosivosti od 25 težinskih jedinica. Kod kapaciteta $11 \le Q \le 17$ moguće je utovariti samo jedan predmet prvog tipa, tako da primena formule za $f_n(Q)$, za $n = 1$, daje maksimalnu vrednost 40 (poena, bodova):

$$\begin{aligned} f_n(Q) &= \max_{0 \le x_n \le \lfloor \frac{Q}{P_n} \rfloor} \{C_n x_n + f_{n-1}(Q - P_n x_n)\} \\ f_1(Q) &= \max_{0 \le x_1 \le \lfloor \frac{Q}{P_1} \rfloor} \{C_1 x_1\} \\ f_1(11) &= \max_{0 \le x_1 \le \lfloor \frac{11}{11} \rfloor} \{40x_1\} = \max \left\{ \begin{array}{l} 40 \cdot 0 = 0 \\ 40 \cdot 1 = 40 \end{array} \right\} = 40. \end{aligned}$$

Kod kapaciteta $18 \le Q \le 19$ moguće je utovariti dva predmeta drugog tipa, tako da primena formule za $f_n(Q)$, za $n = 2$, daje maksimalnu vrednost 76 (poena, bodova):

$$\begin{aligned} f_n(Q) &= \max_{0 \le x_n \le \lfloor \frac{Q}{P_n} \rfloor} \{C_n x_n + f_{n-1}(Q - P_n x_n)\} \\ f_2(Q) &= \max_{0 \le x_2 \le \lfloor \frac{Q}{P_2} \rfloor} \{C_2 x_2 + f_{2-1}(Q - P_2 x_2)\} \\ f_2(18) &= \max_{0 \le x_2 \le \lfloor \frac{18}{9} \rfloor} \{38x_2 + f_1(18 - 9x_2)\} \\ &= \max \left\{ \begin{array}{l} 38 \cdot 0 + f_1(18 - 9 \cdot 0) = 0 + f_1(18) = 0 + 40 = 40 \\ 38 \cdot 1 + f_1(18 - 9 \cdot 1) = 38 + f_1(9) = 38 + 0 = 38 \\ 38 \cdot 2 + f_1(18 - 9 \cdot 2) = 76 + f_1(0) = 76 + 0 = 76 \end{array} \right\} = 76. \end{aligned}$$

Kod kapaciteta $20 \le Q \le 21$ moguće je utovariti jedan predmet prvog i jedan drugog tipa, tako da primena formule za $f_n(Q)$, za $n = 2$, daje maksimalnu vrednost 78 (poena, bodova):

$$\begin{aligned} f_2(Q) &= \max_{0 \le x_2 \le \lfloor \frac{Q}{P_2} \rfloor} \{C_2 x_2 + f_{2-1}(Q - P_2 x_2)\} \\ f_2(20) &= \max_{0 \le x_2 \le \lfloor \frac{18}{9} \rfloor} \{38x_2 + f_1(18 - 9x_2)\} \\ &= \max \left\{ \begin{array}{l} 38 \cdot 0 + f_1(20 - 9 \cdot 0) = 0 + f_1(20) = 0 + 40 = 40 \\ 38 \cdot 1 + f_1(20 - 9 \cdot 1) = 38 + f_1(11) = 38 + 40 = 78 \\ 38 \cdot 2 + f_1(20 - 9 \cdot 2) = 76 + f_1(2) = 76 + 0 = 76 \end{array} \right\} = 78. \end{aligned}$$

Kod kapaciteta $22 \le Q \le 25$ moguće je utovariti dva predmeta prvog tipa, tako da primena

formule za $f_n(Q)$, za $n = 1$, daje maksimalnu vrednost 80 (poena, bodova):

$$\begin{aligned} f_n(Q) &= \max_{0 \le x_n \le \lfloor \frac{Q}{P_n} \rfloor} \{C_n x_n + f_{n-1}(Q - P_n x_n)\} \\ f_1(Q) &= \max_{0 \le x_1 \le \lfloor \frac{22}{11} \rfloor} \{40 x_1\} \\ f_1(22) &= \max_{0 \le x_1 \le \lfloor \frac{11}{11} \rfloor} \{40 x_1\} = \max \left\{ \begin{array}{l} 40 \cdot 0 = 0 \\ 40 \cdot 1 = 40 \\ 40 \cdot 2 = 80 \end{array} \right\} = 40. \end{aligned}$$

Kako se za svaki interval izvršio proračun dolazi se do rezultata da se najveća vrednost $f_2(25)$ dobija pri utovaru u transportno sredstvo dva predmeta prvog tipa, kao što je prikazano u tabeli.

Q	$F_2(Q)$	x_2	x_1
0 - 8	0	0	0
9 - 10	38	1	0
11 - 17	40	0	1
18 - 19	76	2	0
20 - 21	78	1	1
22 - 25	80	0	2

Tabela 2.5.2.

U trećoj i četvrtoj etapi razmatrajući mogućnosti utovara predmeta prvog, drugog i trećeg tipa, odnosno prvog, drugog, trećeg i četvrtog tipa dolazi se do rezultata datih u naredne dve tabele. Vrednosti funkcija $f_3(Q)$ i $f_4(Q)$ se dobijaju korišćenjem sledećih rekurzivnih relacija:

$$\begin{aligned} f_3(Q) &= \max_{0 \le x_3 \le \lfloor \frac{Q}{P_3} \rfloor} \{C_3 x_3 + f_2(Q - P_3 x_3)\}, \ x_3 = 0, 1, 2, 3 \\ f_4(Q) &= \max_{0 \le x_4 \le \lfloor \frac{Q}{P_4} \rfloor} \{C_4 x_4 + f_2(Q - P_4 x_4)\}, \ x_4 = 0, 1, 2, 3, 4 \end{aligned}$$

U narednoj tabeli prikazane su vrednosti utovara za predmete prvog, drugog i trećeg tipa.

Q	$f_3(Q)$	x_3	x_2	x_1
0 - 6	0	0	0	0
7 - 8	26	1	0	0
9 - 10	38	0	1	0
11 - 13	40	0	0	1
14 - 15	52	2	0	0
16 - 17	64	1	1	0
18 - 19	76	0	2	0
20 - 21	78	3	1	1
22	80	0	0	2
23 - 24	90	2	1	0
25	102	1	2	0

Tabela 2.5.3.

U tabeli ispod prikazane su vrednosti utovara za predmete sva četiri tipa.

Q	$f_4(Q)$	x_4	x_3	x_2	x_1
0 - 5	0	0	0	0	0
6	19	1	0	0	0
7 - 8	26	0	1	0	0
9 - 10	38	0	0	1	0
11 - 12	40	0	0	0	1
13	45	1	1	0	0
14	52	0	2	0	0
15	57	1	0	1	0
16 - 17	64	0	1	1	0
18 - 19	76	0	0	2	0
20 - 21	78	0	0	1	1
22	83	1	1	1	0
23	90	0	2	1	0
24	95	1	0	2	0
25	102	0	1	2	0

Tabela 2.5.4.

Optimalno rešenje postavljenog problema prevoza, u vrednosti prioriteta od 102 (boda, poena), dobija se pri prevoženju jednog predmeta trećeg tipa i dva predmeta drugog tipa. U ovom slučaju, kapacitet transportnog sredstva potpuno je iskorišćen jer je 1*7+2*9 = 25 težinskih jedinica. Analiza rezultata optimizacije pokazuje da dobijeno optimalno rešenje nije dato samo za transportno sredstvo nosivosti 25 težinskih jedinica, već i za ona sredstva čiji se kapacitet kreće u intervalu $0 \leq Q \leq 25$ težinskih jedinica, što znači da je rešen jedan skup sličnih problema. Takođe je moguće uočiti i kolika je "vrednost" transporta i koliko je iskorišćenje kapaciteta transportnog sredstva ako se ne donese odluka koja nije optimalna.

5.3 Dinamičko programiranje i uopšteni inverzi

Dinamičko programiranje se može koristiti za generisanje uopštenih inverza matrice. Rezultati ovog odeljka su preuzeti iz rada [4].

Poznato je da generalisani inverzi matrica imaju važnu ulogu u najmanjim srednje-kvadratnim problemima (least squares problems) [1, 27]. Kada se minimizira veličina $q = (Ax - b)^T(Ax - b)$, rešenje je dato sa $x = A^\dagger b$, gde je $A^\dagger b$ MoorePenroseov inverz matrice A. Osim toga, ako rešenje nije jedinstveno, tada je $x = A^\dagger b$ rešenje minimalne dužine (solution of minimal length), koje je jedinstveno. Generalisani inverzi se koriste u mnogim drugim disciplinama. Oni su presudni u teoriji linearnih asocijativnih memorija (linear associative memories) [12], u regresionoj analizi (regression analysis) [2], i u teoriji ograničenog mehaničkog kretanja (theory of constrained mechanical motion) [23].

Postoji veliki broj metoda za izračunavanje generalisanih inverza. Između ostalih, poznati su metodi Decella and Grevillea [5, 24]. Istaknutu ulogu ima rastavljanje matrice pomoću singularno vrednosne dekompozicije (singular value decomposition) [23]. Osim toga, pristup dinamičkog programiranja koji koristi funkcionalne jednačine se takođe može upotrebiti, zato što se matrica može posmatrati kao konstrukcija kojoj se pridodaje po jedna kolona u nekom trenutku vremena. Dinamičko programiranje se koristi u rešavnju problema najmanjeg srednje-kvadratnog rešenja [6]. Međuim, prvi put je primena dinamičkog programiranja u izračunavanju generalisanih inverza objavljena u radu [8]. U radu [8] je to urađeno

pokazivanjem veze između dinamičkog programiranja i Greville-ovog sekvencijalnog metoda uvedenog u [9].

Izvođenje $\alpha Q \beta R$ algoritma [10, 11]

Ovaj dinamički program za izračunavanje generalisanog inverza je inspirisan $\alpha Q \beta R$ algoritmom koji traži vektor x posredstvom dinamičkog programa tako da je norma $|Ax - b|^2$ najmanja, i da je dužina vektora x minimalna. Pri tome je A matrica dimenzija $m \times n$, b je vektor-kolona dimenzije m, i x je vektor-kolona dimenzije n. Pogledajmo prvo ukratko kako $\alpha Q \beta R$ algoritam izračunava najmanje kvadratno rešenje. Dve ciljne funkcije su uvedene u ovom algoritmu. Prva je

$$f_k(b) = \text{najmanji kvadrat dužine vektora } A_k x^k, \tag{5.3.1}$$

gde je A_k matrica koja se sastoji od prvih k kolona od A, i x_k je vektor-kolona dimenzije k. Druga funkcija je data izrazom

$$g_k(b) = \text{najmanji kvadrat dužine vektora } x_k, \tag{5.3.2}$$

u kome x^k odgovara ograničenju

$$|A_k x^k - b|^2 = \min. \tag{5.3.3}$$

Kolone matrice A su označene sa $a_1, a_2, \ldots a_n$. Pretpostavimo da su vrednosti $f_{k-1}(b)$ i $g_{k-1}(b)$ poznate. Tada su vrednosti $f_k(b)$ i $g_k(b)$ dobijene korišćenjem principa optimalnosti [6]. Procedura dinamičkog programiranja za izračunavanje $f_k(b)$ i $g_k(b)$ bazirana na poznavanju funkcija $f_{k-1}(b)$ i $g_{k-1}(b)$ zavisi od toga da li je a_k linearno zavisan od $a_1, a_2, \ldots, a_{k-1}$. Ako je a_k linearno zavisan od $a_1, a_2, \ldots, a_{k-1}$, tada je

$$f_k(b) = f_{k-1}(b); \tag{5.3.4}$$

jer linearna kombinacija vektora $a_1, a_2, \ldots a_{k-1}$; a_k ne može da aproksimira vektor b bolje od linearne kombinacije vektora $a_1, a_2, \ldots a_{k-1}$. Takođe je

$$g_k(b) = \min_{x_k} \left[x_k^2 + g_{k-1}(b - x_k a_k) \right], \tag{5.3.5}$$

gde je x_k skalar, k-ta komponenta vektora x^k. To je zbog toga što kada je x_k izabrano, tada kazna iznosi x_k^2. Ostale komponente, $x_1, x_2, \ldots, x_{k-1}$, moraju biti izabrane tako da je linearna kombinacija vektora $a_1, a_2, \ldots, a_{k-1}$ što je moguće bliža novom ciljnom vektoru $b - x_k a_k$. Osim toga, suma $x_1^2 + x_2^2 + \cdots + x_{k-1}^2$ mora biti minimalna.

U slučaju kada je a_k linearno nezavisan od $a_1, a_2, \ldots, a_{k-1}$, sa bilo kojim izborom sklara x^k moramo da što približnije aproksimiramo novi ciljni vektor $b - x_k a_k$ pomoću sume $x_1 a_1 + \cdots + x_{k-1} a_{k-1}$. Prema tome,

$$f_k(b) = \min_{x_k} f_{k-1}(b - x_k a_k) \tag{5.3.6}$$

Ako je minimizirajuća vrednost skalara x_k jednaka x_k^*, tada

$$g_k(b) = (x_k^*)^2 + g_{k-1}(b - x_k^* a_k). \tag{5.3.7}$$

Razmotrimo sada funkciju $f_k(b) =$ kvadrat dužine reziduuma $(A_k x^k - b)$, pri čemu je vektor x^k izabran optimalno, gde je $\dim b = m$, i A_k se sastoji od prvih k kolona od A. Za $k = 1$ imamo

$$f_1(b) = \min(a_1 x_1 - b)^T (a_1 x_1 - b) \tag{5.3.8}$$

Uslov minimiziranja je

$$a_1^T a_1 x_1 - a_1^T b = 0, \tag{5.3.9}$$

tako da je

$$x_1^* = \frac{a_1^T b}{a_1^T a_1} = a_1^\dagger b, \tag{5.3.10}$$

gde je $a_1^\dagger$ generalisani inverz vektora a_1 (uz pretpostavku $a_1 \neq 0$). To povlači da je

$$f_1(b) = b^T \left[I - a_1 a_1^\dagger \right] b. \tag{5.3.11}$$

Funkcija $f_1(b)$ je kvadratna po b što može da se zapiše sa

$$f_1(b) = b^T Q_1 b, \tag{5.3.12}$$

gde je Q_1 simetrična pozitivno semi-definitna $m \times m$ matrica. Da bi pokazali da je $f_k(b)$ kvadratna matrica po b, to jest

$$f_k(b) = b^T Q_k b, \tag{5.3.13}$$

gde je Q_k simetrična pozitivno definitna $m \times m$ matrica, posmatramo dva zasebna slučaja, zavisno od toga da li je vektor a_k linearno zavisan ili nije linearno zavisan od prethodnih vektora.

Pretpostavimo prvo da je a_k nezavisan od $a_1, a_2, \ldots, a_{k-1}$ i da je k-ti element vektora x označen sa s. Iz jednačina (5.3.6) imamo

$$f_k(b) = \min_s f_{k-1}(b - a_k s), \ k = 2, 3, \ldots, n. \tag{5.3.14}$$

Pretpostavimo da

$$f_{k-1}(b) = b^T Q_{k-1} b. \tag{5.3.15}$$

Tada se dobija

$$f_k(b) = \min_s \left\{ f(b - sak)^T Q_{k-1} (b - sak) \right\}. \tag{5.3.16}$$

Funkcija koju treba optimizirati je

$$\begin{aligned} \{Q_{k-1}(b - sa_k)\} &= (b^T Q_{k-1} - s a_k^T Q_{k-1}(b - sa_k) \\ &= b^T Q_{k-1} b - s a_k^T Q_{k-1} b - b^T Q_{k-1} s a_k \\ &\quad + s^2 a_k^T Q_{k-1} a_k. \end{aligned} \tag{5.3.17}$$

Da bismo dobili optimalno rešenje, stavimo $(\frac{d\{\}}{ds}) = 0$, tako da je

$$\frac{d\{\}}{ds} = 2sa_k^T Q_{k-1} a_k - aTkQ_{k-1}b - b^T Q_{k-1} a_k = 0 \tag{5.3.18}$$

Primetimo da kako je $a_k^T Q_{k-1} b$ skalar, a_k^T

$$Q_{k-1}b = (a_k^T Qk - 1b)^T = b^T Q_{k-1} a_k.$$

Prema tome, optimalno rešnje je

$$s^* = \frac{b^T Q_{k-1} a_k}{a_k^T Q_{k-1} a_k} = \frac{a^T k Q_{k-1} b}{a_k^T Q_{k-1} a_k}. \tag{5.3.19}$$

Primenimo dva oblika optimalnog rešenja s, i označimo $a_k = Q_{k-1}a_k$. Jednačinu (5.3.17) možemo sada zapisati sa

$$f_k(b) = b^T Q_{k-1} b - 2b^T \alpha_k \frac{\alpha_k^T b}{\alpha^T k \alpha_k} + \frac{b^T \alpha_k}{\alpha_k^T \alpha_k} (\alpha_k^T \alpha_k) \frac{\alpha_k^T b}{\alpha_k^T \alpha_k}. \tag{5.3.20}$$

Premo tome,

$$f_k(b) = b^T \left(Q_{k-1} - \frac{\alpha_k \alpha_k^T}{a_k^T \alpha_k} \right) b = b^T Q_k b. \tag{5.3.21}$$

Rekurzivna veza za Q_{k-1} je

$$Q_k = Q_{k-1} - \frac{\alpha_k \alpha_k^T}{\alpha_k^T a_k}. \tag{5.3.22}$$

Pretpostavimo sada da je a_k linearno zavisno od $a_1, a_2, \ldots, a_{k-1}$. Tada je ispunjeno $f_k(b) = f_{k-1}(b)$, jer linearna kombinacija od $a_1, a_2, \ldots, a_{k-1}$; a_k ne može biti bliža vektoru b od linearne kombinacije vektora $a_1, a_2, \ldots, a_{k-1}$. Premo tome, u ovom slučaju važi $Q_k = Q_{k-1}$.

Razmotrimo sada drugu funkciju $g_k(b)$ = najmanji kvadrat dužine vektora x^k, za koju je $|A_k x^k - b|$ minimalno.

Za $k = 1$ sledi, iz prethodnih razmatranja, da je u jednoj fazi procesa

$$x_1^* = \frac{a_1^T b}{a_1^T a_1} = a_1^\dagger b, \tag{5.3.23}$$

gde je $a_1^\dagger$ generalisani inverz vektora a_1 (pretpostavljajući da je $a_1 \neq 0$). To povlači

$$g_1(b) = b^T (a_1^\dagger)^T a_1^\dagger b. \tag{5.3.24}$$

Funkcija $g_1(b)$ je kvadratana po b, što možemo zapisati sa

$$g_1(b) = b^T R_1 b, \tag{5.3.25}$$

gde je R_1 smetrična pozitivno semi-definitna $m \times m$ matrica. Da bi pokazali da je $g_k(b)$ takođe kvadratna po b, tj.

$$g_k(b) = b^T R_k b, \tag{5.3.26}$$

razmatramo dva zasebna slučaja, zavisno od toga da li je vektor a_k linearno zavisan od prethodnih vektora ili ne. Pretpostavimo da je a_k nezavisan od $a_1, a_2, \ldots, a_{k-1}$. Iz jednačina (5.3.19) zaključujemo

$$s^* = \frac{\alpha_k^T b}{\alpha_k^T \alpha_k}. \tag{5.3.27}$$

Primenimo s^* na jednačinu (5.3.7), koja je oblika

$$g_k(b) = (s^*)^2 + g_{k-1}(b - s^* a_k)$$

dobija se

$$\begin{aligned} g_k(b) &= (s^*)^2 + (b - s^* a_k)^T R_{k-1}(b - s^* a_k) \\ &= \frac{b^T \alpha_k}{\alpha_k^T a_k} \frac{\alpha_k^T b}{\alpha_k a_k} + b^T \left[I - \frac{a_k \alpha^T k}{\alpha_k^T a_k} \right] R_{k-1} \left[I - \frac{a_k \alpha^T k}{\alpha_k^T a_k} \right] b \\ &= b^T \left\{ \left[I - \frac{\alpha_k a_k^T}{\alpha_k^T a_k} \right] R_{k-1} \left[I - \frac{a_k \alpha_k^T}{\alpha_k^T a_k} \right] + \frac{\alpha_k \alpha_k^T}{(\alpha_k^T a_k)^2} \right\} b. \end{aligned} \tag{5.3.28}$$

Pretpostavimo sada da a_k zavisi od $a_1, a_2, \ldots, a_{k-1}$. Iz prethodnog dela znamo da rešenje jednačine (5.3.3) nije jednistveno, i da jedno od tih rešenja daje minimnalnu dužinu vektora x^k. Ako je $k$ti element rešenja x označen sa s, dobija se rekurzivna relacija

$$g_k(b) = \min_s \{s^2 + g_{k-1}(b - s a_k)\}. \tag{5.3.29}$$

Uz pretpostavku

$$g_{k-1}(b) = b^T R_{k-1} b, \tag{5.3.30}$$

dobijamo

$$g_k(b) = \min_s \{s^2 + (b - s a_k)^T R_{k-1}(b - s a_k)\}, \tag{5.3.31}$$

$$\begin{aligned} \{\ldots\} &= s^2 + (b^T - s a_k^T) R_{k-1}(b - s a_k) \\ &= s^2 + (b^T R_{k-1} - s a_k^T R_{k-1})(b - s a_k) \\ &= s^2 + b^T R_{k-1} b - 2 s a^T k R_{k-1} b + s^2 a_k^T R_{k-1} a_k. \end{aligned} \tag{5.3.32}$$

Označimo $R_{k-1} a_k$ sa β_k, i stavimo $\frac{d\{\}}{ds} = 0$. Tada je

$$2s - 2\beta_k^T b + 2 s a_k^T \beta_k = 0. \tag{5.3.33}$$

Premo tome, dobija se optimalno rešenje

$$s^* = \frac{b^T \beta_k}{1 + a_k^T \beta_k}. \tag{5.3.34}$$

Kako je s skalar, takođe imamo

$$s = \frac{\beta^T k b}{1 + a_k^T \beta k}.$$

Primenimo jednačinu (5.3.34) i poslednju jednačinu na funkciju $g_k(b)$. Vidi se da je

$$\begin{aligned} g_k(b) &= \frac{b^T \beta_k}{1 + a_k^T \beta_k} \left(1 + a_k^T \beta_k\right) \frac{\beta_k^T b}{1 + a_k^T \beta_k} + b^T R_{k-1} b - 2 \frac{b^T \beta_k}{1 + a_k^T \beta_k} \beta_k^T b \\ &= b^T \left(R_{k-1} - \frac{\beta_k \beta_k^T}{1 + a_k^T \beta_k} \right) b. \end{aligned} \tag{5.3.35}$$

Prema tome, time smo dokazali da je $g_k(b)$ takođe kvadratna po b; to jest

$$g_k(b) = b^T R_k b \tag{5.3.36}$$

gde je $R_k = R_{k-1} - (\beta_k \beta_k^T / (1 + a_k^T \beta_k))$.

U prethodnoj proceduri, potrebno je odrediti kada je a_k linearno zavisan od $a_1, a_2, \ldots, a_{k-1}$, a kada nije. Dakle, potrebno je pokazati da važi

$$a_k = \alpha_k + \text{lin. kom. od } \alpha_1, \alpha_2, \ldots, \alpha_k, \tag{5.3.37}$$

gde je $\alpha_k = Q_{k-1} a_k$. Takođe, pokazano je da je α_k ortogonalno na $a_1, a_2, \ldots, a_{k-1}$ [4]. Prema tome, umesto direktne provere da li a_k linearno zavisi od $a_1, a_2, \ldots, a_{k-1}$, možemo proveriti da li je a_k nula vektor ili nije nula vektor. Dakle, ako je

$$\alpha_k = Q_{k-1} a_k = 0,$$

tada je a_k linearno zavisan od $a_1, a_2, \ldots, a_{k-1}$; u suprotnom, ako je $a_k = Q_{k-1} a_k \neq 0$, tada je a_k linearno nezavisan od $a_1, a_2, \ldots, a_{k-1}$. U sekciji 4 ćemo pokazati da su svi podaci potrebni za dobijanja generalisanog inverza dobijeni u prethodnoj proceduri. Algoritamski koraci su:

$$\alpha_1 = a_1, \tag{5.3.38}$$

$$Q_1 = I - \frac{\alpha_1 \alpha_1^T}{\alpha_1^T \alpha_1} = I - a_1 \alpha_1^\dagger, \tag{5.3.39}$$

$$R_1 = (a_1^\dagger)^T a_1^\dagger. \tag{5.3.40}$$

Tada za svaku vrednost k, $k = 2, 3, \ldots, n$, imamo dva slučaja. Ako je

$$\alpha_k = Q_{k-1} a_k = 0, \tag{5.3.41}$$

tada je

$$Q_k = Q_{k-1}, \tag{5.3.42}$$

$$\beta_k = R_{k-1} a_k, \tag{5.3.43}$$

$$R_k = R_{k-1} - \frac{\beta_k \beta_k^T}{1 + a_k^T \beta_k}. \tag{5.3.44}$$

Ako je, sa druge strane važi

$$\alpha_k = Q_{k-1}a_k \neq 0, \tag{5.3.45}$$

tada je

$$\alpha_k^\dagger = \frac{\alpha_k^T}{\alpha_k^T\alpha_k}, \tag{5.3.46}$$

$$Q_k = Q_{k-1} - \alpha_k\alpha_k^\dagger, \tag{5.3.47}$$

$$R_k = (\alpha_k^\dagger)^T\alpha_k^\dagger + (I - a_k\alpha_k^\dagger)^T R_{k-1}(I - a_k\alpha_k^\dagger). \tag{5.3.48}$$

Izvođenje Grevilleovog algoritma

Zbog svoje velike primenljivosti, Grevilleov rezultat je široko rasprostranjen, ali bez konstruktivnog dokaza zbog kompleksnosti tehnike rešavanja. Jednostavano izvođenje su dali Udwadia i Kalaba [5]. U nastavku slede detalji dokaza.

Pretpostavimo da je poznat MP inverz matrice A, označen sa $A^\dagger$. Želimo da rešimo problem najmanjeg kvadrata $Bx \cong b$. Matrica B je podeljena sa $[A\ a]$, i vektor x je podeljen sa $\begin{bmatrix} z \\ s \end{bmatrix}$, gde je z vektor dimenzije $k = 1$, i s je skalar. Prema tome, možemo pisati $Bx \cong b$ kao

$$[Aa]\begin{bmatrix} z \\ s \end{bmatrix} s \cong b.$$

Tada

$$Az^\dagger as \cong b, \tag{5.3.49}$$

$$Az \cong b - as. \tag{5.3.50}$$

Kada znamo $A^\dagger$, rešenje linearnog sistema je

$$z = A^\dagger(b - as). \tag{5.3.51}$$

Zamenom jednačine (5.3.51) u jednačinu (5.3.50) dobija se

$$AA^\dagger(b - as) \cong b - as, \tag{5.3.52}$$

$$(I - AA^\dagger)as \cong (I - AA^\dagger)b. \tag{5.3.53}$$

Označimo $(I - AA^\dagger)a$ sa c. Ako je $c \neq 0$, imao

$$s = \frac{a^T(I - AA^\dagger)}{a^T(I - AA^\dagger)(I - AA^\dagger)a}(I - AA^\dagger)b. \tag{5.3.54}$$

Kako je $(I - AA^\dagger)$ simetrična i idempotentna matrica, što se može pokazati pomoću osobina generalisanog inverza, imamo

$$s = \frac{a^T(I - AA^\dagger)}{a^T(I - AA^\dagger)a}b, \tag{5.3.55}$$

$$s = c^\dagger b. \tag{5.3.56}$$

Sa druge strane, znamo da je rešenje $[A\ a]\begin{bmatrix} z \\ s \end{bmatrix} \cong b$ dato sa

$$\begin{bmatrix} z \\ s \end{bmatrix} = [A\ a]^\dagger b. \tag{5.3.57}$$

Zamenom rešenja za z i s, koja su ranije dobijena, dobijamo

$$[A\ a]^\dagger b = \begin{bmatrix} A^\dagger - A^\dagger a c^\dagger \\ c^\dagger \end{bmatrix} b. \tag{5.3.58}$$

Kako je vektor b proizvoljan,

$$[A\ a]^\dagger = \begin{bmatrix} A^\dagger - A^\dagger a c^\dagger \\ c^\dagger \end{bmatrix}, \text{ ako je } c \neq 0. \tag{5.3.59}$$

Razmotrimo sada slučaj kada je $c = 0$; zapravo, c je m-dimenzionalni nula vektor. Iz jednačine (5.3.53), znamo da kada je $c = 0$, skalar s može biti proizvoljan, što znači da vektor koji je rešenje nije jedinstven. Umesto da biramo proizvoljno rešenje tako da važi $[A\ a]\begin{bmatrix} z \\ s \end{bmatrix} \cong b$, tražimo rešenje minimalne dužine, to jest

$$z^T z + s^2 = \min. \tag{5.3.60}$$

Označimo vektor $(A^\dagger b)$ sa u a vektor $(A^\dagger a)$ sa v. Sada imamo

$$z = A^\dagger(b - as) = u - vs. \tag{5.3.61}$$

Prema tome

$$(u - vs)^T(u - vs) + s^2 = \min, \tag{5.3.62}$$

$$u^T u - 2sv^T u + v^T v s^2 + s^2 = \min. \tag{5.3.63}$$

Vrednost s koja minimizira dužinu vektora $z^T z + s^2$ u (5.3.60) je dobijena iz

$$(1 + v^T v)s = v^T u = v^T A^\dagger b, \tag{5.3.64}$$

što daje

$$s = \frac{v^T A^\dagger}{1 + v^T v} b, \tag{5.3.65}$$

tako da je

$$[A\ a]^\dagger = \begin{bmatrix} A^\dagger - A^\dagger a \frac{v^T A^\dagger}{1 - v^T v} \\ \frac{v^T A^\dagger}{1 - v^T v} \end{bmatrix}, \text{ if } c = 0. \tag{5.3.66}$$

Jednačine (5.3.59) i (5.3.66) određuju rekurzivnu proceduru.

Generalisani inverzi i $\alpha Q \beta R$

Ponovimo da se rešenje minimalne norme problema najmanjeg kvadrata $(Ax - b)^T(Ax - b) = \min$ takođe može dobiti iz $x = A^\dagger b$, gde je $A^\dagger$ pseudo-inverz od A. Prirodno je tražiti $A^\dagger$ primenom $\alpha Q \beta R$ algoritma.

Grevilleov algoritam pokazuje kako da znajući pseudoinverz $A_{k-1}^\dagger$ matrice A_{k-1}, koja se sastoji od prvih $k-1$ kolona matrice A, izračunamo pseudoinverz $A_k^\dagger$ matrice A_k, koja se sastoji od prvih k kolona matrice A. Vektor c igra ključnu ulogu u ovom postupku. Razmotrimo šta konkretno vektor c predstavlja. Pretpostavimo da matrica A sadrži n kolona, i da vektor kolona w sadrži n skalara $w_1, w_2, \ldots, w_n$. Oni su označeni sa

$$A = [a_1 \ a_2 \ \ldots \ a_n], \quad w = \begin{bmatrix} w_1 \\ w_2 \\ \ldots \\ w_n \end{bmatrix}.$$

Prema tome, Aw je linearna kombinacija kolona matrice A. Kako je $c = (I - AA^\dagger)a$,

$$c^T Aw = a^T (I - AA^\dagger)^T Aw. \tag{5.3.67}$$

Zato što je $I - AA^\dagger$ simetrična matrica, važi

$$\begin{aligned} c^T Aw &= a^T (I - AA^\dagger) Aw, \\ c^T Aw &= a^T (Aw - AA^\dagger Aw). \end{aligned} \tag{5.3.68}$$

Podsetimo na jednu od osobina MP generalisanog inverza, $AA^\dagger A = A$. Zaključujemo da je

$$c^T Aw = 0, \tag{5.3.69}$$

što znači da je vektor c ortogonalan na svaku linearnu kombinaciju kolona matrice A. Još više, iz $c = (I - AA)^\dagger a$ imamo

$$a = c + AA^\dagger a, \tag{5.3.70}$$

gde $A^\dagger a$ možemo posmatrati kao neki vektor w. Prema tome, a je linearna kombinacija prethodnih kolona matrice A plus vektor c koji je ortogonalan na sve te kolone matrice A. Dakle, nova kolona, a, je određena kao nezavisna od svih prethodnih kolona, kada c nije nula vektor.

Razmotrimo sada vektor α_k u $\alpha Q \beta R$ algoritmu. Može se pokazati da je vektor α_k ortogonalan na vektore $a_1, a_2, \ldots, a_{k-1}$. Umesto primene vektora c kao u Grevilleovom algoritmu, prelazak sa $A_{k-1}^\dagger$ na $A_k^\dagger$ može se izvršiti po članovima ili α_k ili β_k, zavisno od toga da li je $\alpha_k \neq 0$ ili $\alpha_k = 0$.

Razmotrimo prvo slučaj u kome je $c \neq 0$. To znači da vektor a_k ima nenula komponentu koja je ortogonalna na vektore $a_1, a_2, \ldots, a_{k-1}$. Dakle, to je slučaj u kome α_k nije linearno zavisan od $a_1, a_2, \ldots a_{k-1}$. Grevilleova rekurzija je data sa

$$A_k^\dagger = \begin{bmatrix} A_{k-1}^\dagger - A_{k-1}^\dagger a_k c^\dagger \\ c^\dagger \end{bmatrix}, \tag{5.3.71}$$

koja zahteva da znamo vektor c, pored $A_{k-1}^{\dagger}$ i a_k. Ali $\alpha Q \beta R$ algoritam daje vektor α_k, koji je komponenta od a_k koja je ortogonalna na $a_1, a_2, \ldots a_{k-1}$, kao što je pokazano ranije. Dakle, kada su $A_{k-1}^{\dagger}$, a_k i α_k poznati, možemo odrediti $A_k^{\dagger}$ sa

$$A_k^{\dagger} = \begin{bmatrix} A_{k-1}^{\dagger} - A_{k-1}^{\dagger} a_k \alpha_k^{\dagger} \\ \alpha_k^{\dagger} \end{bmatrix}. \tag{5.3.72}$$

Drugi slučaj je $c = 0$. U tom slučaju je $a_k = A_{k-1} A_{k-1}^{\dagger} a_k$, pa je vektor a_k linearno zavisan od kolona matrice A_{k-1}. Grevilleova rekurzija je data formulom

$$A_k^{\dagger} = \begin{bmatrix} A_{k-1}^{\dagger} - A_{k-1}^{\dagger} a_k v^T \\ v^T \end{bmatrix}, \tag{5.3.73}$$

gde je

$$v = \frac{(A_{k-1}^{\dagger})^T A_{k-1}^{\dagger} a_k}{1 + a_k^T (A_{k-1}^{\dagger})^T A_{k-1}^{\dagger} a_k}. \tag{5.3.74}$$

Za interpretaciju tih relacija sa stanovišta $\alpha Q \beta R$ algoritma, ponovimo jednačine (5.3.2) i (5.3.14):

$$g_k(b) = \text{najmanji kvadrat dužine vektora } x^k$$

$$\text{koji minimizira } |A_k x_k - b|^2 = b^T R_k b.$$

Kako znamo da je rešenje tog problema jednako

$$x_k = A_k^{\dagger} b, \tag{5.3.75}$$

možemo pisati

$$g_k(b) = b^T (A_k^{\dagger})^T A_k^{\dagger} b. \tag{5.3.76}$$

Prema tome, vidimo da je

$$R_k = (A_k^{\dagger})^T A_k^{\dagger}, \tag{5.3.77}$$

što bi trebalo da bude saglasno sa R_k koje se dobija u pethodnim jednačinama (5.3.21) i (5.3.25). To omogućava da pišemo

$$v = \frac{R_{k-1} a_k}{1 + a_k^T R_{k-1} a_k}, \tag{5.3.78}$$

$$v = \frac{\beta_k}{1 + a_k^T \beta_k}, \tag{5.3.79}$$

što prestavlja vektor v preko izraza β_k. Ove relacije pokazuju kako se prelazak sa A_{k-1} na A_k može izvršiti preko α_k ili β_k, zavisno od toga da li je $\alpha_k = 0$ ili $\alpha_k \neq 0$. Dodatno, znamo i rešenje problema $|A_k x^k - b|^2 = \min$ is $x^k = A_k^{\dagger} b$. Dakle

$$\begin{aligned} f_k(b) &= (A_k A_k^{\dagger} b - b)_T (A_k A_k^{\dagger} b - b), \\ f_k(b) &= b^T (A_k A_k^{\dagger} - I)(A_k A_k^{\dagger} - I) b, \\ f_k(b) &= b^T (A_k A_k^{\dagger} - A_k A_k^{\dagger} - A_k A_k^{\dagger} + I) b, \\ f_k(b) &= b^T (I - A_k A^{\dagger})_k^{\dagger} b. \end{aligned} \tag{5.3.80}$$

To pokazuje da je

$$Q_k = I - A_k A_k^{\dagger}, \tag{5.3.81}$$

gde je Q_k izračunat tokom $\alpha Q \beta R$ algoritma, A_k se sastoji od prvih k kolona matrice A, i $A_k^{\dagger}$ je generalisani inverz od A_k.

U radovima [14, 15, 16, 21, 22] upotrebljen je Grevileov metod pregrađivanja za izračunavanje uopštenih inverza racionalnih i polinomijalnih matrica.

6 Memoizacija

6.1 Transformacija stringa

Poreklo zadatka: [25].

Data su dva niza: niz X dužine N se sastoji od slova A i B, a niz Y dužine $M > N$ od cifara 0 i 1. Da li se iz prvog niza dobiti drugi, koristeći sledeće operacije:
1. zameniti bilo koje A nepraznim nizom cifara 0, ili
2. zameniti bilo koje B nepraznim nizom nula ili nepraznim nizom jedinica.

Rešenje: Tražena transformacija nije moguća ako se niz X završava slovom A, a niz Y jedinicom. Pretpostavimo zato da se niz X završava slovom A, a niz Y sekvencom od K nula,ili da se niz X završava slovom B, a niz Y sekvencom od K nila ili K jedinica. Transformacija niza X u niz Y će tada biti moguća ako is samo ako je moguće transformisati niz X_{N-1} (X bez poslednjeg elementa) u neki od nizova $Y_P, M - K \leq P < M$.

Dovoljno je rešiti sve probleme $Q(I, J)$ transformacije niz X_I, u niz $Y_J, 1 \leq I \leq N, 1 \leq J \leq M$. Odgovore možemo smestiti u logičku matricu, koju ćemo popunjavati po vrstama ili po kolonama. Odgovor na postavljeno pitanje je u polju (N, M) te logičke matrice.

Pretpostavimo da je $X = (ABA)$, $Y = (0001110000)$. Dinamičko programiranje rešava svih $N \cdot M = 3 \cdot 10 = 30$ problema, dok memoizacija mořešavati lanac $Q(3, 10) \to Q(2, 6) \to Q(1, 3) \to Q(0, 0)$, nakon čega neće stići (a neće ni morati) da postavi ostaje podprobleme (iz $Q(3, 10)$, to su $Q(2, 7), Q(2, 8)$ i $Q(2, 9)$: a iz $Q(2, 6)$ to su $Q(1, 4)$ i $Q(1, 5)$ i njihovi dalji pozivi u dubinu).

6.2 Pobednička strategija

Poreklo zadatka: [25].

Postoji mnogo igara za dva igrača koje imaju sledeće zajedničke osobine: dat je konačan skup legalnih pozicija S i konačan skup dozvoljenih poteza M, kojima se iz jedne legalne pozicije prelazi u drugu. Igrači vuku poteze naizmenično. Jedan podskup F skupa svih pozicija S čini završne pozicije. To su pozicije u kojima nije moguće odigrati ni jedan legalan potez. Kada se stigne do neke završne pozicije, igra je završena, a igrač koji je odigrao poslednji potez je pobednik. Ni jedna pozicija se ne može ponoviti tokom igre, tako da se svaka igra neminovno završava u konačnom broju poteza i nerešen ishod ne postoji. Cilj svakog igrača je da pobedi kad god je to moguće. Problem koji se u ovakvim igrama obično postavlja je da za datu

poziciju ustanovimo da li igrač koji je na potezu može da pobedi, i koji potez treba radi toga da odigra.

Navešćemo jedan postupak koji u ovakvim igrama uvek možemo da primenimo. Prema uslovima zadatka, igra se može predstaviti konačnim acikličnim grafom, tako da su čvorovi grafa pozicije, a grane grafa su potezi. Obojimo sve čvorove koji odgovaraju završnim pozicijama u crno. U tim pozicijama, igrač koji je na potezu je izgubio. Nakon toga, obojene čvorove iz kojih se nekim potezom može preći u crni čvor obojimo u belo (u takvoj poziciji igrač koji je na potezu može da pobedi tako što pređe na poziciju kojoj odgovara crni čvor), a neobojene čvorove iz kojih se svakim mogućim potezom prelazi u beli čvor obojimo u crno. Postupak bojenja je konačan i svaki čvor na opisani način dobija crnu ili belu boju. Pobednička strategija se (kao što smo već rekli) sastoji u tome da igrač koji je na potezu pređe na poziciju kojoj pdgovara crni čvor. Na taj način protivnik zatiče ili završnu poziciju pa je odmah izgubio, ili poziciju iz koje mora preći u neku kojoj odgovara beli čvor. Prema tome, prvi igrač može da nastavi da sprovodi istu strategiju, pa će na taj način povući poslednji potez i pobediti.

U igri se koristi N kuglica. Svaki igrač neizmenično uzima po izvestan broj kuglica. U prvom potezu igrač može uzeti najviše P kuglica ($P < N$), a u svakom sledećem potezu ne višeod P i ne više od za Q više od onoga koliko je uzeo protivnik u prethodnom potezu ($Q < P$). Na primer, za $N = 100, P = 10, Q = 5$, ako bi prvi igrač uzeo 3 kuglice, drugi može uzeti od 1 do 8 kuglica, a ako bi prvi uzeo 6 kuglica, legalni potezi grugog su 1 do 10 kuglica. Cilj je uzeti poslednju kuglicu. Odrediti pobedničku strategiju.

Rešenje: Pozicija se može zadati brojem preostalih kuglica na stolu i maksimalnim brojem kuglica kije igrač može uzeti. Tako je polazna pozicija $(100, 10)$, a u gornjem primeru iz nje nastaju pozicije $(97, 8)$ i $(94, 10)$. Možemo popuniti matricu B veličine $N * P$, tako da je $B(X, Y)$ broj kuglica koje prema pobedničkoj strategiji treba uzeti da bi se pobedilo, ili 0 ako pobedu nije moguće garantovati. Podproblemi se rešavaju po vrstama sleva na desno, dakle za svako X od 0 do N, a pri fiksiranom X za svako Y od $Q + 1$ do P (drugi broj u uređenom paru koji opisuje tekuću poziciju, predstavlja gornje ograničenje broja kuglica koje se mogu uzeti, a ono se u zavisnosti od prethodnog poteza kreće od $Q + 1$ do P).

Memoizacija je ovde nešto bolje rešenje, jer se ni ovde ne moraju rešavati svi podproblemi. Čim ustanovimo da je pozicija $(91, 10)$ gubitnička, sledi da je pozicija $(100, 10)$ pobednička, pa nećemo imati potrebe da rešavamo problem za pozicije 92,93,94,95,96,97,98,99 i 99 kuglica. Takvih ušteda ima mnogo tokom igre, a ne samo na prvom koraku (odozgo na dole). Pri mekim vrednostima N, p, q,(na primer 35,20,15) može se desiti da se stvarno reši manje od 30% ukupnog broja problema, što znači da u tim slučajevima memoizacija radi oko tri puta brže od klasičnog pristupa.

```
program Pobednicka_Strategija;
  uses crt,dos;
   var  b:array[0..100,0..100] of integer;
        n,p,q,x,y:integer;
```

```
function min(g,h:integer):integer;
  begin
   if g<h then min:=g    else min:=h
  end;
procedure resi(x,y:integer);
   var  k:integer;
   begin
      if b[x,y]=-1 then
       begin
        if x<=y then b[x,y]:=x;
        k:=y;
        while (k>0) and (b[x,y]=-1) do
         begin
          resi(x-k,min(k+q,p));
          if b[x-k,min(k+q,p)]=0 then b[x,y]:=k;
          k:=k+1;
         end;
        if b[x,y]=-1 then  b[x,y]:=0;
       end;
   end;
begin
    readln(n); readln(p); readln(q);
     for x:=0 to n do
        for y:=0 to p do  b[x,y]:=-1;
     resi(n,p);   writeln(b[n,p]);
end.
```

6.3 Maksimalna suma nesusednih u drvetu

Svečani prijem

U jednoj velikoj firmi vlada hijerarhija nadređenosti u obliku drveta. Direktor je koren tog drveta, a svaki drugi radnik ima jedinstvenog neposrednog šefa. Osim toga, svaki radnik ima zvanični rejting (pozitivan realan broj), koji odražava važnost tog radnika. U firmi se sprema svečani prijem i vlasnik firme je u dilemi koje radnike da pozove. Vlasnik želi da postigne dva cilja: da se niko ne bi osećao neprijatno, pozive se ili radnik ili njegov neposredni šef (ili nijedan), ali nikako obojica. Osim toga, vlasnik želi da zbir rejtinga prisutnih radnika bude maksimalan.

Za svakog od N radnika numerisanih od 1 do N, dat je redni broj P_u neposrednog šefa (za direktora je data 0), i rejting T_u tog radnika. Načiniti optimalan izbor radnika koji će biti pozvani na prijem.

Rešenje: Matematčka formulacija istog zadatka je sledeća: dato je drvo G. Svaki čvor u drveta G ima realnu pozitivnu težinu T_u. Naći podskup čvorova drveta G čiji je zbir težina najveći, a u kome nema susednih čvorova drveta G.

Pod naslednicima ili potomcima nekog čvora u drvetu podrazumevaćemo njegove neposredne naslednike. Neka je $P(G)$ optimalan podskup čvorova za drvo G neka je čvor u koren drveta i neka su $v_1, \ldots, v_K$ svi naslednici čvora u, a $w_1, \ldots, w_M$ svi naslednici čvorova $v_1, \ldots, v_K$.

Tada važi sledeće:

• Ako čvor u ne ulazi u optimalan podskup $P(G)$, onda su delovi optimalnog podskupa P koji pripadaju poddrvetima sa korenima $v_1, \ldots, v_K$ redom, optimalni podskupovi za ta poddrveta.

• Ako čvor u ulazi u optimalan podskup $P(G)$, onda su delovi optimalnog podskupa P koji pripadaju redom poddrvetima iz čvorova $w_1, \ldots, w_M$, optimalni podskupovi za ta drveta.

Ovim je ustanovljena optimalnost podstrukture problema. Za svako poddrvo potrebno je da se reši isti problem, počevši od najjednostavnijih drveta, odnosno listova polaznog drveta. Kako nije sasvim jednostavno poređati podprobleme u potreban redosled (najpre listovi, pa njihovi prethodnici, itd. do korena, koji je poslednji), možemo zadatak rešavati memoizacijom. Rešenje za drvo koje ima samo jedan čvor, je taj čvor. Za netrivijalna drveta podprobleme rešavamo na isti način kao i za glavni problem, i tako dobijamo i zbir težina $B(u)$ čvorova optimalnog podskupa za drvo koje počinje iz čvora u. Odnosno za ceo graf:

$$B(u) = \max \left\{ T(u) + \sum_{i=1}^{M} B(w_i), \sum_{i=1}^{K} B(v_i) \right\}$$

Za rekonstrukciju optimalnog podskupa dovoljan je logički niz C, gde je $C(u) =$ True ako i samo ako je čvor u uvršten podskup poddrveta čiji je koren u. Pažnja: to ne znači da u konačan izbor čvorova ulaze oni čvorovi za koje je u nizu C zapisana vrednost True. Traženi izbor ćemo rekonstruisati rekurzivnom procedurom, koristeći niz C.

Ulazni podaci su, prema rečenom zadati pomoću dva niza: niz P, gde je p_i redni broj prethodnika u drvetu za čvor i, i niz T, u kome je t_i težina čvora broj i u drvetu. Međutim, niz prethodnika nije pogodan za rešavanje zadataka. Potrebno je da za dati čvor brzo odredimo njegove sledbenike (potomke), a ne prethodnika. Zato ćemo formirati niz *potomci*, čiji k-ti član će biti lista potomaka čvora k. Kada pročitamo da je prthodnik čvora i čvor a, umesto da pamtimo $p_i = a$, dodaćemo čvor i u listu potomaka čvora a.

Kada za neki čvor v izračunamo sumu težina optimalnih podskupova svih drveta koja počinju od njegovih potomaka, možemo je zapamtiti kao $bb(v)$, jer će biti ponovo potreban pri rešavanju problema za prethodnika u čvora v. Težina optimalnog podskupa za drvo koje počinje iz v je uobičajeno označena sa $b(v)$.

```
program Svecani_Prijem;
  uses crt,dos;
  type PokClanListe=^Clanliste;
       ClanListe= record
                    vred:integer;
                    sled:PokClanListe;
                  end;
  var
    potomci:array[0..100] of PokClanListe;
    b,bb,t:array[1..100] of integer;
    c:array[1..100] of boolean;
    pom:PokClanListe;
    n,i,a,k,u:integer;
```

```
procedure resi(u:integer);
  var PokV:PokClanListe;
      sumv,sumw,v:integer;
begin
    if b[u]=-1 then
     begin
       sumv:=0;   sumw:=0;   PokV:=potomci[u];
       while PokV<>nil do
        begin
         v:=PokV^.vred;    resi(v);
         sumv:=sumv+b[v];    sumw:=sumw+bb[v];    PokV:=PokV^.sled;
        end;
       bb[u]:=sumv;
       if sumw+t[u]>sumv then
         begin
          b[u]:=sumw+t[u];    c[u]:=true;
         end
        else
          begin b[u]:=sumv;c[u]:=false; end
       end;
     end;

procedure ispis(u:integer);
   var
    PokV,PokW:PokClanListe;
    v,w:integer;
   begin
    if not c[u] then
    begin
     PokV:=potomci[u];
     while PokV<>nil do
     begin
      v:=PokV^.vred;   ispis(v);  PokV:=PokV^.sled;
     end;
    end
    else
     begin
      writeln(u);   PokV:=potomci[u];
      while PokV<>nil do
       begin
        v:=PokV^.vred;    PokW:=potomci[v];
        while PokW<>nil do
          begin
           w:=PokW^.vred;  ispis(w);  PokW:=PokW^.sled;
          end;
        PokV:=PokV^.sled;
       end;
     end;
   end;

begin
 potomci[0]:=nil;
 textcolor(1);write('broj cvorova:'); textcolor(2); readln(N);
 for k:=1 to n do
  begin
   potomci[k]:=nil;     b[k]:=-1;
  end;
```

```
 for i:=1 to n do
  begin
   textcolor(4);write('prethodnik i tezina cvora',i,':');
   textcolor(5);readln(a,t[i]);
   new(pom);   pom^.vred:=i;    pom^.sled:=potomci[a];
   potomci[a]:=pom;
  end;
   u:=potomci[0]^.vred;
   resi(u);     writeln(b[u]);     ispis(u);
end.
```

7 Veza između memoizacije, rekurzije i dinamičkog programiranja

Dinamičko programiranje se odnosi na tehniku dolaska do rešenja, a ne na samo programiranje. Pisanje programa moze da se uradi u bilo kom programskom jeziku. Konkretno u slučaju dinamičkog programiranja obično se popunjava jedna tabela rešenja (bilo niz ili matrica). Kod dinamičkog programiranja do rešenja glavnog problema dolazi se kombinacijom jednostavnijih podproblema istog tipa. Problem se dakle rešava počev od najednostavnijih problema pa dolazeći do glavnog, tako sto se prvo uoči hijearhija problema, onda se rešavaju jednostavniji problemi, njihove vrednosti se upisuju u tabelu rešenja i onda se kombinacijom dobijenih rešenja iz tabele rešavaju složeniji podroblemi sve do rešenja glavnog problema. Na kraju u tabeli se pročita traženo rešenje i često se ista tabela može iskoristiti i za traženje puta dolaska do rešenja.

Dinamičkim programiranjem rešenje se dobija znatno efikasnije u odnosu na rekurziju. Najbitniji uslov za mnogo veću efikasnost dinamičkog programiranja od rekurzije je to sto ovde podproblemi imaju zajedničke podprobleme odnosno delimično se preklapaju. Ovaj uslov nije neophodan za primenu dinamičkog programiranja, ali bez njega dinamičko programiranje gubi svoju veliku prednost u odnosu na rekurziju. Ovo se dešava zbog toga što se primenom rekurzije kao tehnike dolaska do rešenja jedan isti problem susreće više puta i svaki put se nepotrebno iznova rešava (pri tome se broj ponovljenih rešavanja po pravilu povešava eksponencijalno sa povećavanjem dimenzije glavnog problema). Kada ne bi bilo ovog preklapanja podproblema rekurzija ne bi bila manje efikasna jer bi se i tada rešavanje podproblema javlja samo jednom i čak u nekim problemima može se desiti da rekurzivno rešenje radi brže i troši znatno manje memorijskog prostora. Kao rešenje problema neefikasnosti rekurzije kod određenih tipova problema javlja se rekurzija sa memoizacijom ili kraće samo memoizacija. To se može ostvariti na sledeći način: formira se tabela rešenja kao u rešavanju problema dinamičkim programiranjem. Na početku se svi elementi tabele postave na neku besmislenu vrednost (koja se ne može dobiti rešavanjem problema, obično 0 ili -1), čime se označava da je vrednost rešenja nedefinisana, odnosno da problem nije rešavan. Rekurzivna procedura ili funkcija koja rešava zadatak trebalo bi najpre da potraži rešenje u tabeli. Ako nađe rešenje vraća to rešenje (ako se radi o funkciji) i završava sa radom, a ako ga ne nađe traži ga rekurzivno, upisuje u tabelu i na kraju vraća dobijeno rešenje (ako se radi

o funkciji) i završava sa radom. Memoizacija se može smatrati modifikacijom dinamičkog programiranja, jer se i dalje koristi tabela za pamćenje vrednosti rešenja rešenih problema. Razlika je u tome što se memoizacijom problemi (kao i uvek rekurzijom) rešavaju odozgo na dole (kod dinamičkog programiranja se rešavaju odozdo na gore). Ova razlika za efikasnost i brzinu i nije bitna, ono što je najbitnije je da se podproblemi ne rešavaju više puta. Zbog toga problemi rešeni tehnikom dinamičkog prtogramiranja mogu efikasno da se reše i memoizacijom. Po efikasnosti memoizacija je vrlo bliska dinamičkom programiranju, a najčešće je neznatno sporija jer se izvesno vreme gubi na prenos parametara i druge aktivnosi koje zahteva rekurzija. Postoji međutim problemi gde je memoizacija skoro uvek efikasnija od dinamičkog programiranja. To su problemi u kojima nije potrebno rešiti sve podprobleme (što se češće dešava) već samo neke. Dinamičko programiranje ne može unapred znati da li će neki problem biti kasnije upotrebljen u rešavanju većeg problema. Memoizacija nema taj nedostatak jer nepotrebne podprobleme neće ni postavljati. Najbolji način za proveravanje efikasnosti ovih tehnika programiranja je kroz konkretne probleme. Problemi su uzeti iz knjige [25]. Napominjem da su vremena izražena u milisekundama i da ovom prilikom sama izmerena vremena rada nisu bitna već njihov relativni odnos. Lepo se vidi da su rezultati uglavnom očekivani: da su memoizacija i dinamiško programiranje vrlo bliski po efikasnosti dok u većini problema rekurzija je znatno sporija tehnika programiranja i to konkretno u onim problemima gde, kako je navedeno, dolazi do preklapanja podproblema pa se oni nepotrebno više puta izvršavaju. Ovo znatno kod rekurzije usporava algoritam dok se kod memoizacije taj problem lako rešava prethodnim proveravanjem da li je neki problem već rešavan i ukoliko jeste on se ne rešava ponovo.

Problem ranca:

Obim problema	10	40	60	80	1000	100000
Dinamičko programiranje	0	0	10	10	110	13910
Rekurzija	0	45	41069	Dugo	Večno	Večno
Memoizacija	0	0	10	10	110	13830

Maksimalna suma nesusednih u nizu:

Obim problema	10	100	500	5000
Dinamičko programiranje	0	40	220	2354
Rekurzija	0	40	231	2284
Memoizacija	0	40	221	2323

Najduži lanac deljivosti:

Obim problema	50	100	200
Dinamičko programiranje	10	10	50
Rekurzija	20	4116	Dugo
Memoizacija	10	10	20

Roman:

Obim problema	10:5	20:10	30:10	50:5	50:10	200:10
Dinamičko programiranje	10	10	10	10	10	20
Rekurzija	10	2143	54899	170000	Dugo	Dugo
Memoizacija	10	10	10	10	10	20

Literatura

[1] A. Ben-Israel and T.N.E. Grevile, *Generalized inverses, Theory and applications, Second edition*, Canadian Mathematical Society, Springer, New York, 2003.

[2] G.W. Bohrnstedt, D. Knoke, *Statistics for Social Data Analysis, third ed.*, F.E. Peacock, Itasca, IL, 1994.

[3] S. Borović i Mili' c, Milićević, *Zbirka zadataka iz odabranih oblasti operacionih istraživanja*, Biblioteka Vojne akademije, Beograd 2001.

[4] Y. Fan, R. Kalaba, *Dynamic programming and pseudo-inverses*, Appl. Math. Comput., **139** (2003) 323-342.

[5] R. Kalaba, N. Rasakhoo, *Algorithms for generalized inverses, Journal of Optimization Theory and Applications*, **48** (1986) 427-435.

[6] R. Bellman, R. Kalaba, *Dynamic Programming and Modern Control Theory*, McGraw-Hill, New York, 1965.

[7] D. Cvetković, M. Čangalović, Đ. Dugošija, V. Kovačević – Vujčić, S. Simić, J. Vuleta, *Kombinatorna optimizacja*, DOPIS, Beograd, 1996.

[8] Y. Fan, R. Kalaba, *Dynamic programming and pseudo-inverses*, Applied Mathematics and Computations **139** (2003) 323-342.

[9] T.N.E. Grevile, *Some applications of the pseudo-inverse of matrix*, SIAM Rev., **3** (1960), 15–22.

[10] R. Kalaba, R. Xu, W. Feng, *Solving shortest length least-squares problems via dynamic programming*, Journal of Optimization Theory and Applications **85** (1995) 613-632.

[11] R. Kalaba, H. Natsuyama, *Dynamic programming and minimal norm solutions of least squares problems*, submitted for publication.

[12] T. Kohonen, *Self-Organization & Associative Memory*, Springer-Verlag, New York, Inc, 1989.

[13] S. Opricović, *Optimizacija sistema*, Nauka, Beograd, 1992.

[14] M.D. Petković, P.S. Stanimirović, P.S. *Partitioning method for two-variable rational and polynomial matrices*, Math. Balkanica, **19** (2005), 185–194.

[15] M.D. Petković, P.S. Stanimirović, *Symbolic computation of the Moore-Penrose inverse using partitioning method*, International Journal of Computer Mathematics, **82** (2005), 355–367.

[16] M.D. Petković, P.S. Stanimirović, and Tasić, M. *Effective Partitioning Method for Computing Weighted Moore-Penrose Inverse*, Computers and Mathematics with Applications, to appear.

[17] R. Petrović, *Specijalne metode u optimizaciji sistema*, Tehnička Knjiga, Beograd, 1978.

[18] D. Pisinger, *Algorithms for knapsack problem*, Ph.D. thesis, February 1995, Dept. of Computer Science, University of Copenhagen.

[19] R. Sedgewick, *Algorithms*, Addison-Wesley Publishing Company, Reading, Massachusetts, Menlo Park, California London, Amsterdam, Don Mills, Ontario, Sydney, 1984.

[20] S. Skiena, M. Revilla, *Programming Challenges*, Springer, 2002.

[21] P.S. Stanimirović, M. Tasić, *Partitioning method for rational and polynomial matrices*, Applied Mathematics and Computation **155** (2004), 137–163.

[22] Tasić,M.B., Stanimirović, P.S. and Petković, M.D. *Symbolic computation of weighted Moore-Penrose inverse using partitioning method*, Appl. Math. Comput. **189** (2007), 615–640.

[23] F. Udwadia, R. Kalaba, *Analytical Dynamics*, Cambridge University Press, London, 1996.

[24] F. Udwadia, R. Kalaba, *An alternative proof of the Greville formula*, Journal of Optimization Theory and Applications 94 (3) (1997) 23-28.

[25] M. Vugdelija, *Dinamičko programiranje*, Društvo matematičara Srbije, Beograd, 1999.

[26] M. Vujošević, *Metode optimizacije*, Društvo operacionih istraživača, Beograd, 1996.

[27] G.Wang, Y.Wei and S. Qiao, *Generalized Inverses: Theory and Computations*, Science Press, Beijing, 2004.

[28] www.z-trening.com

Glava 5

Višekriterijumska optimizacija

Modeli za nalaženje optimuma jedne kriterijumske funkcije su obično samo aproksimacija realnih problema u kojima donosilac odluke mora da vodi računa o više ciljeva. Postoje matematički modeli i metodi u kojima donosilac odluke analizira i bira rešenja na osnovu više kriterijuma koji se istovremeno razmatraju. Pritom, kao i u slučaju jednokriterijumske optimizacije, donosilac odluke implicitno zadržava slobodu da prihvati, promeni ili odbaci rešenje dobijeno na osnovu matematičkog modela optimizacije. Metode koje od samog početka formiranja matematičkog modela za određeni realni problem vode računa o više ciljeva istovremeno razvijaju se u oblasti višekriterijumske optimizacije (VKO). Ovaj deo matematičkog programiranja svoj buran razvoj ima od kraja sedamdesetih godina dvadesetog veka.

Postoji više razloga koji utiču na to da su problemi VKO suštinski drugačiji u odnosu na probleme jednokriterijumske optimizacije. Osnovni je u tome što se svi faktori koji utiču na odluku, odnosno svi ishodi koje bi imalo eventualno rešenje, posmatraju kao kriterijumi čije bi vrednosti trebalo da budu optimalne. Dakle, potrebno je naći rešenje koje je najbolje po svim razmatranim kriterijumima istovremeno a činjenica je da su neki od njih u skoro svim problemima odlučivanja međusobno konfliktni. Pored toga, razmatrani kriterijumi mogu po svojoj prirodi biti veoma raznorodni i izraženi u različitim mernim jedinicama, od novčanih jedinica, preko jedinica fizičkih veličina, do verovatnoća ili subjektivnih procena datih po nekoj skali koja se formira za konkretni problem. Sve ovo ukazuje da konačno rešenje ne može da se odredi bez učešća donosioca odluke.

Zadatke višekriterijumske optimizacije u slučajevima kada se razmatraju važne odluke kao što su odluke u vezi sa kapitalnim ulaganjima u opremu, karakteriše relativno veliki broj kriterijuma. Što je broj kriterijuma veći, zadaci višekriterijumske analize su složeniji i teži. U ovakvim situacijama, u odlučivanju učestvuje veći broj pojedinaca ili grupa i svi oni favorizuju svoje sisteme vrednosti. Radi efikasnijeg analiziranja odluke i pronalaženja pogodnog rešenja vrši se grupisanje kriterijuma.

Uobičajene su sledeće grupe kriterijuma:

- ekonomski,
- tehnički,
- tehnološki,
- socijalni i
- ekološki.

1 Uvod

Iako je linearno programiranje veoma primenljivo u praksi, mnoge probleme iz prakse je nemoguće adekvatno linearizovati a da se pritom drastično ne izgubi na tačnosti. U tom slučaju primenjuju se metodi nelinearnog programiranja. Osim nelinearnosti, u mnogim problemima je potrebno naći optimum više od jedne funkcije cilja. U tom slučaju, moramo rešavati ***problem višekriterijumske optimizacije***. Ukoliko sve funkcije cilja imaju optimum u istoj tački, problem je trivijalan i direktno se svodi na problem nelinearnog ili linearnog programiranja. U praksi je ova situacija veoma retka. Postoji više metoda za rešavanje problema višekriterijumske optimizacije [19]. Zajedničko svim tim metodima je da se polazni problem na odgovarajući način svodi na problem linearnog i nelinearnog programiranja.

Višekriterijumska optimizacija se može posmatrati kao nastavak istraživanja u klasičnoj (jednokriterijumskoj) optimizaciji, uz izvesna proširenja. Formalno, osnovno proširenje je uvođenje vektorske kriterijumske funkcije, što dovodi do problema vektorskog maksimuma. Suštinski, potrebno je da se proširi koncept optimalnosti. Razmatrajući problem vektorskog maksimuma koncept optimalnosti se zamenjuje konceptom neinferiornosti (Pareto optimalnosti). Može se uvesti pojam opšteg (jedinstvenog) kriterijuma optimizacije, koji uključuje kriterijumske funkcije i preferenciju donosioca odluke. Rešenje zadatka višekriterijumske optimizacije koje se dobija prema takvom kriterijumu je optimalno. U tom slučaju pojam optimalnog rešenja iz klasične optimizacije može se zadržati i u višekriterijumskoj. Međutim, teškoće se upravo javljaju pri pokušaju formalizacije takvog jedinstvenog kriterijuma. Zato se u višekriterijumskoj optimizaciji koriste dve faze ili etape. U prvoj fazi se određuje skup "boljih" rešenja na osnovu vektorske kriterijumske funkcije, a u drugoj se na osnovu preferencije donosioca odluke usvaja konačno rešenje, koje se može nazvati optimalnim. Skup rešenja koji se prezentira donosocu odluke trebalo bi da sadrži mali broj rešenja, koja su neinferiorna prema datim kriterijumskim funkcijama. Problem višekriterijumske optimizacije se najčešće javlja u planiranju složenih sistema; na primer, regionalni razvoj, razvoj vodoprivrednih ili elektroprivrednih sistema, urbano planiranje i očuvanje prirodne okoline [21]. Višekriterijumski problem se javlja u ekonomiji kao problem određivanja tržišne ravnoteže ili ekonomija blagostanja u decentralizovanoj ekonomiji [21]. Sličan problem se javlja i kao problem ravnoteže u teoriji igara [21]. U teoriji igara razmatraju

se igre sa više igrača, što se u teoriji odlučivanja javlja kao "grupno odlučivanje" ili odlučivanje sa više donosioca odluke.

U ovoj glavi ćemo izložićemo problem višekriterijumske optimizacije, kao i metode za njeno rešavanje. Najpre ćemo dati definiciju problema višekriterijumske optimizacije kao i neophodnih pojmova za kasnija razmatranja. Teorijski ćemo obraditi i dati implementaciju nekoliko klasičnih metoda višekriterijumske optimizacije. Svaki od opisanih metoda biće ilustrovan na jednom ili više primera. Razmatranja vezana za implementaciju metoda su originalna i preuzeta su iz radova [25], [27] kao i iz monografije [26].

Opšta formulacija višekriterijumske optimizacije (VKO) poseduje opšti oblik

$$\begin{array}{ll} \max & Q(\mathbf{x}) = Q_1(\mathbf{x}), \ldots, Q_l(\mathbf{x}), \quad \mathbf{x} \in \mathbf{R}^n \\ \text{p.o.} & f_i(\mathbf{x}) \leq 0,\ i = 1, \ldots, m \\ & h_i(\mathbf{x}) = 0,\ i = 1, \ldots, k. \end{array}$$

gde su:$Q_1(\mathbf{x}), \ldots, Q_l(\mathbf{x})$, $f_1(x), \ldots, f_m(x), g_1(x), \ldots, g_m(x)$ realne funkcije od n promenljivih koje su sadržane u vektoru $\mathbf{x} = (x_1, \ldots, x_n)$.

U ovom zadatku traži se rešenje $\mathbf{x}$ koje maksimizira svih l funkcija cilja. Zato se zadatak višekriterijumske optimizacije (VKO) naziva i zadatak vektorske optimizacije. Radi jednostavnosti ovde se razmatraju samo problemi maksimizacije. Poznato je da se zadatak minimizacije jednostavno prevodi u zadatak maksimizacije množenjem kriterijumske funkcije sa -1. Sve nadalje izložene definicije i metode moguće je prilagoditi da važe za rešavanje zadatka minimizacije.

Kažemo da je $\mathbf{X} \subseteq R^n$ *skup dopustivih rešenja* ako važi

$$\mathbf{X} = \{\mathbf{x} | f_i(\mathbf{x}) \leq 0,\ i = 1, \ldots, m;\ h_i(\mathbf{x}) = 0,\ i = 1, \ldots, k\}.$$

Svakom dopustivom rešenju $\mathbf{x} \in \mathbf{X}$ odgovara skup vrednosti kriterijumskih funkcija, tj. vektor $Q(\mathbf{x}) = (Q_1(\mathbf{x}), Q_2(x), \ldots, Q_l(\mathbf{x}))$. Na taj način se skup dopustivih rešenja preslikava u *kriterijumski skup*, tj. $S = \{Q(\mathbf{x}) | \mathbf{x} \in \mathbf{X}\}$.

U daljem tekstu biće korišćeni sledeci pojmovi:

- *Marginalna rešenja* zadatka VKO se određuju optimizacijom svake od funkcija cilja pojedinačno nad zadatim dopustivim skupom, tj. rešavanjem l jednokriterijumskih zadataka

$$\begin{array}{ll} \max & Q_j(\mathbf{x}), \quad \mathbf{x} \in \mathbf{R}^n \\ \text{p.o.} & f_i(\mathbf{x}) \leq 0,\ i = 1, \ldots, m \\ & h_i(\mathbf{x}) = 0,\ i = 1, \ldots, k. \end{array}$$

Marginalna rešenja ćemo obeležavati sa $\mathbf{x}^{(j)^*} = (x_1^{(j)^*}, x_2^{(j)^*}, \ldots, x_n^{(j)^*})$, gde je $x^{(j)^*}$ optimalno rešenje dobijeno optimizacijom j-te funkcije cilja nad zadatiom dopustivim skupom $\mathbf{X}$.

- *Idealne vrednosti funkcija cilja*, označene sa Q_j^* jesu vrednosti funkcija cilja za marginalna rešenja

$$Q_j^* = Q_j(\mathbf{x}^{(j)^*}),\ j = 1, \ldots, l.$$

- Idealne vrednosti funkcija cilja određuju *idealnu tačku* u kriterijumskom prostoru, tj. idealnu vrednost vektorske funkcije

$$Q^* = (Q_1^*, Q_2^*, \ldots, Q_l^*).$$

- Ako postoji rešenje $\mathbf{x}^*$ koje istovremeno maksimizira sve funkcije cilja, tj.

$$\mathbf{x}^* = \{\mathbf{x} | Q_j(\mathbf{x}) = Q_j^*,\ j = 1, \ldots, l\},$$

onda se takvo rešenje naziva *savršeno resenje*.

U najvećem broju slučajeva marginalna rešenja se razlikuju i savršeno rešenje ne postoji. Kada savršeno rešenje postoji, tada se u suštini ne radi o problemu VKO.

Veoma je važno imati u vidu da su u realnim problemima ciljevi gotovo uvek u koliziji, što znači da ne mogu svi biti dostignuti u potpunosti. Zbog toga najčečešće nije moguće strogo definisati optimum niti za svaka dva rešenja formalno odrediti koje je bolje od drugog. Iz tog razloga proces dobijanja rešenja zahteva učešće donosioca odluke (u daljem tekstu DO). To je najčešće neko ko ima dublji uvid u problem i po čijem se zahtevu pristupa rešavanju. Donosilaca odluke može biti i više i tada se problem može dodatno iskomplikovati zbog njihovih različitih ciljeva, udela u odlučivanju i stepena odgovornosti koji su spremni da preuzmu.

2 Osnovni pojmovi i definicije

2.1 Donosilac odluke i njegove preferencije

Na nivou odlučivanja ključnu ulogu ima **donosilac odluke** (skraćeno **DO**). Prema normativnoj teoriji odlučivanja, donosilac odluke je savršeno racionalan pojedinac koji uvek zna šta hoće i nastoji da to realizuje. Mada se ciljevi koje pred sebe postavlja razlikuju po formulaciji, sadržini, složenosti i značaju, svi oni u osnovi sadrže zajedničku komponentu. To je želja donosioca odluke da poveća dobitke, odnosno, smanji ili izbegne gubitke. Pri tome se racionalni donosilac odluke rukovodi principom maksimizacije lične dobrobiti. Nesumnjivo da svako od nas, u manjoj ili većoj meri, odstupa od ovog "ideala" koji ne razmatra izbore iz navike, brzoplete, nepromišljene ili hirovite odluke. Ipak, pažnja sa kojom se odluke donose obično raste sa njihovim značajem, tako da se pri važnim izborima pažljivo vrši procena prednosti i nedostataka pojedinih alternativa. Dobrobit koju pružaju alternative ocenjuje se na osnovu subjektivnih kriterijuma (želje, interesi, uverenja, moralni principi, ukusi i slično). Kao proizvod manje ili više saglasnih ili nesaglasnih uticaja svih ovih faktora formiraju se individualne preferencije, na osnovu kojih donosilac odluke izgrađuje stav prema alternativama i opredeljuje se za jednu od njih.

Da bi doneta odluka bila racionalna, preferencije moraju da zadovolje neke polazne pretpostavke. Za precizno definisanje ovih pretpostavki neophodno je upoznavanje sa osnovnim relacijama preferencije i indiferencije.

Relacije preferencije i indiferencije

Pretpostavimo da donosilac odluke raspolaže skupom akcija (alternativa) koje imaju samo po jedan poznat ishod, tako da se preferencije između akcija određuju na osnovu preferencija između njihovih ishoda. U opštem slučaju akcije se obeležavaju sa $A_i, i = 1, \ldots, m$. Zbog jednostavnosti izlaganja biće korišćene oznake $A_1 = x$, $A_2 = y, \ldots$.

Ako se prilikom poređenja dve alternative, x i y, smatra da je alternativa x bolja od alternative y, onda je x (strogo) preferirano u odnosu na y, tj.

$$xPy \text{ ili } x \succ y,$$

U slučaju slobodnog izbora, donosilac odluke bira alternativu x, i biće razočaran ako bude prinuđen da prihvati alternativu y. Ako su alternative x i y podjednako dobre, tada je donosilac odluke indiferentan između x i y, ili

$$xIy \text{ ili } x \sim y.$$

Prilikom izbora između x i y svejedno je koja će alternativa biti izabrana, odnosno, donosilac odluke će biti podjednako zadovoljan ili nezadovoljan da dobije x ili y. (Ovde je važno imati na umu da donosilac odluke nije indiferentan prema alternativama, već prema izboru između njih, tj. svejedno mu je koju će alternativu dobiti, a ne da li će je dobiti ili ne.)

Uslovi racionalnosti

Da bi se donela racionalna odluka neophodno je da preferencije ispune nekoliko uslova koji se nazivaju *uslovima racionalnosti* ili *uslovima logičke konzistentnosti* i formalno su izraženi u vidu sledećih osobina.

Asimetričnost - Za bilo koje dve alternative, x i y, važi

$$\text{ako } xPy, \text{ onda nije } yPx.$$

Kao neposredne posledice ove relacije, dobijaju se sledeće osobine

$$\text{ako } xPy, \text{ onda nije } xIy,$$

$$\text{ako } xIy, \text{ onda nije } xPy \text{ i nije } yPx.$$

Nezavisno od toga kakve su preferencije između dve alternative, tj. da li se x smatra boljom od y ili obrnuto, ili su jednako dobre, pretpostavlja se da su one *relativno stabilne*, tj. *da se ne menjaju u periodu između izbora akcije i njene realizacije*. Relativna stabilnost preferencija, međutim, nikako ne znači da će se pri ponovljenim izborima između x i y uvek birati ista opcija. Postoji relativno mali broj alternativa među kojima se donosilac odluke uvek i bezuslovno opredeljuje za jednu od njih. Takve su *striktne* ili *bezuslovne preferencije* i one su određene etičkim principima, religijom, zdravstvenim razlozima i slično. U ostalim slučajevima preferencije se formiraju pod uticajem brojnih faktora i njihovih različitih kombinacija, zbog

čega ih karakteriše fleksibilnost, tj. *relativna nestabilnost*, kao i tendencija promene tokom vremena.

Kompletnost - Za bilo koje dve alternative, x i y, ili se x preferira u odnosu na y, ili y preferira u odnosu na x, ili postoji indiferentnost između njih, tj.

$$xPy \text{ ili } yPx \text{ ili } xIy.$$

Kompletnost zahteva da je donosilac odluke, bez obzira na stepen sličnosti ili različitosti alternativa među kojima bira, uvek u stanju da odredi preferencije. Mada deluje kao veoma blag, ovaj uslov ne može se uvek zadovoljiti. Neodlučnost se može javiti pri izboru između dve veoma povoljne ili nepovoljne alternative, ili kada se alternative među sobom toliko razlikuju da se ocenjuju na osnovu potpuno različitih kriterijuma. Ali, to nije isto što i svesno odlaganje odluke. Ako se konačan izbor odloži sa namerom da se preispitaju preferencije, donosi se specifična odluka koja štiti od brzopletog izbora neoptimalne opcije; u tom slučaju, odlaganje odluke može se smatrati racionalnim izborom.

Tranzitivnost - Za bilo koje tri opcije x, y, z, važi da ako se x preferira u odnosu na y i y preferira u odnosu na z, onda se x preferira u odnosu na z, tj.

$$\text{ako } xPy \text{ i } yPz, \text{ onda } xPz$$

i ako postoji indiferentnost između x i y, i između y i z, onda postoji indiferentnost i između x i z, tj.

$$\text{ako } xIy \text{ i } yIz, \text{ onda } xIz.$$

Ordinalna funkcija korisnosti

Opcijama rangiranim po preferencijama moguće je pridružiti brojeve koji odražavaju njihov relativan značaj ili korisnosti, a koje se nazivaju ordinalne funkcije korisnosti. Preciznije rečeno, ordinalna funkcija korisnosti je funkcija koja strukturu preferencija preslikava u skup realnih brojeva. Naziva se ordinalnom jer brojevi otkrivaju samo poredak opcija po preferencijama, pri čemu se veći broj pridružuje bolje rangiranoj alternativi.

2.2 Pareto optimalnost

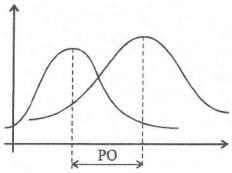
Činjenica da zadaci VKO po pravilu nemaju savršeno rešenje upućuje na preispitivanje koncepta optimalnosti i definicije optimalnog rešenja. Ključnu ulogu u tome ima koncept *Pareto optimalnosti*. To je proširenje poznatog koncepta optimalnosti koj se koristi u klasičnoj jednokriterijumskoj optimizaciji.

Pareto optimum se definiše na sledeći način

- Dopustivo rešenje $\mathbf{x}^*$ predstavlja *Pareto optimum* zadatka VKO ako ne postoji neko drugo dopustivo rešenje $\mathbf{x}$ takvo da važi

$$Q_j(\mathbf{x}) \geq Q_j(\mathbf{x}^*) \quad \forall j = 1, \ldots, l$$

pri čemu bar jedna od nejednakosti prelazi u strogu nejednakost $>$. Drugim rečima, $\mathbf{x}$ je Pareto optimum ako bi poboljšanje vrednosti bilo koje funkcije cilja prouzrokovalo pogoršanje vrednosti neke druge funkcije cilja. Za Pareto oprimum postoje sledeći sinonimi: *efikasno, dominantno i nedominirano rešenje.*



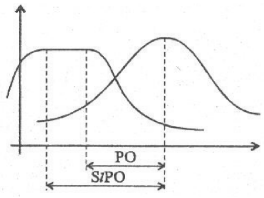
Slika 2.2.1. Geometrijska interpretacija Pareto optimalnih rešenja

Pored Pareto optimuma definišu se slabi i strogi (jaki) Pareto optimumi.

Dopustivo rešenje $\mathbf{x}^*$ je *slabi Pareto optimum* ako ne postoji neko drugo dopustivo rešenje x takvo da važi

$$Q_j(\mathbf{x}) > Q_j(\mathbf{x}^*) \quad \forall j = 1, \ldots, l.$$

Drugim rečima $\mathbf{x}^*$ je slabi Pareto optimum ako nije moguće istovremeno poboljšati sve funkcije cilja.



Slika 2.2.2. Geometrijska interpretacija Pareto optimalnih i Slabo Pareto optimalnih rešenja

Pareto optimalno rešenje $\mathbf{x}^*$ je *strogi Pareto optimum* ako postoji broj $\beta > 0$ takav da za svaki indeks $j \in \{1, \ldots, l\}$ i za svako $\mathbf{x}$ koje zadovoljava uslov

$$Q_j(\mathbf{x}) > Q_j(\mathbf{x}^*)$$

postoji bar jedno $i \in \{1, \ldots, l\} \setminus \{j\}$ takvo da je

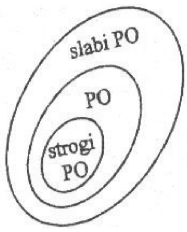
$$Q_i(\mathbf{x}) > Q_i(\mathbf{x}^*)$$

i da važi

$$\frac{Q_j(\mathbf{x}) - Q_j(\mathbf{x}^*)}{Q_i(\mathbf{x}^*) - Q_i(\mathbf{x})} \geq \beta.$$

Strogi Pareto optimum izdvaja ona Pareto rešenja čije promene ne prouzrokuju prevelike relativne promene u funkcijama cilja.

Odnos između opisanih optimuma je takav da svaki skup strožijih Pareto optimuma predstavlja podskup slabijih optimuma, tj. svaki Pareto optimum je istovremeno i slabi Pareto optimum, a svaki strogi Pareto optimum je i Pareto optimum. Odnos svih skupova dat je na slici.



Slika 2.2.3. Odnos između Pareto optimalnih rešenja

Primer 2.2.1 Za sledeći matematički model naći rešenje koristeći se metodima višeciljnog odlučivanja.

$$\begin{array}{ll} \max & f_1(\mathbf{x}) = 2x_1 + 3x_2 \\ \max & f_2(\mathbf{x}) = x_1 - 5x_2 \\ \max & f_3(\mathbf{x}) = -3x_1 + 4x_2 \\ \text{p.o.} & g1(\mathbf{x}): \quad 2x_1 + 0x_2 \leq 10 \\ & g_2(\mathbf{x}): \quad 0x_1 + 3x_2 \leq 9 \\ & g_3(\mathbf{x}): \quad 4x_1 + 0x_2 \leq 32 \\ & g_4(\mathbf{x}): \quad 5x_1 + 0x_2 \geq 5. \end{array}$$

Rešenje. Najpre se određuju marginalna rešenja, koja predstavljaju optimalna rešenja za pojedine kriterijume, a potom i idealne vrednosti funkcija svakog od kriterijuma.

$$\begin{array}{rcl} 2x_1 + 0x_2 \leq 10 & \Rightarrow & x1 \leq 5 \\ 0x_1 + 3x_2 \leq 9 & \Rightarrow & x2 \leq 3 \\ 4x_1 + 0x_2 \leq 32 & \Rightarrow & x1 \leq 8 \\ 5x_1 + 0x_2 \geq 5 & \Rightarrow & x1 \geq 1 \\ x_1, x_2 \geq 0 & \Rightarrow & x_1, x_2 \geq 0 \end{array}$$

U koordinatnom sistemu $(\mathbf{x}) = (x_1, x_2)$ određene su tačke (1,0), (5,0), (5,3), (1,3), koje predstavljaju marginalna rešenja za funkcije kriterijuma.

Idealna vrednost funkcije kriterijuma $f_1(\mathbf{x})$

$$\begin{array}{ll} \max & f_1(\mathbf{x}) = 2x_1 + 3x_2 \\ \text{p.o} & g_1(\mathbf{x}): \quad 2x_1 + 0x_2 \leq 10 \\ & g_2(\mathbf{x}): \quad 0x_1 + 3x_2 \leq 9 \\ & g_3(\mathbf{x}): \quad 4x_1 + 0x_2 \leq 32 \\ & g_4(\mathbf{x}): \quad 5x_1 + 0x_2 \geq 5 \\ & x_1, x_2 \geq 0 \end{array}$$

$$\begin{aligned} f_1(1,0) &= 2 \cdot 1 + 3 \cdot 0 = 2 \\ f_1(5,0) &= 2 \cdot 5 + 3 \cdot 0 = 10 \\ f_1(5,3) &= 2 \cdot 5 + 3 \cdot 3 = 19 \\ f_1(1,3) &= 2 \cdot 1 + 3 \cdot 3 = 11 \end{aligned}$$

Idealna vrednost funkcije kriterijuma $f_1(\mathbf{x})$ jednaka je maksimalnoj vrednosti funkcije $f_1^*(\mathbf{x}) = 19$ za marginalno rešenje $\mathbf{x}^{(1)*} = (5, 3)$.

Idealna vrednost funkcije kriterijuma $f_2(\mathbf{x})$

$$\begin{aligned} \max \quad & f_2(\mathbf{x}) = x_1 - 5x_2 \\ \text{p.o} \quad & g_1(\mathbf{x}): \quad 2x_1 + 0x_2 \leq 10 \\ & g_2(\mathbf{x}): \quad 0x_1 + 3x_2 \leq 9 \\ & g_3(\mathbf{x}): \quad 4x_1 + 0x_2 \leq 32 \\ & g_4(\mathbf{x}): \quad 5x_1 + 0x_2 \geq 5 \\ & x_1, x_2 \geq 0 \end{aligned}$$

$$\begin{aligned} f_2(1,0) &= 1 \cdot 1 - 5 \cdot 0 = 1 \\ f_2(5,0) &= 1 \cdot 5 - 5 \cdot 0 = 5 \\ f_2(5,3) &= 1 \cdot 5 - 5 \cdot 3 = -10 \\ f_2(1,3) &= 1 \cdot 1 - 5 \cdot 3 = -14 \end{aligned}$$

Idealna vrednost funkcije kriterijuma $f_2(\mathbf{x})$ jednaka je maksimalnoj vrednosti funkcije $f_2^*(x) = 5$ za marginalno rešenje $\mathbf{x}^{(2)*} = (5, 0)$.

Idealna vrednost funkcije kriterijuma $f_3(\mathbf{x})$.

$$\begin{aligned} \max \quad & f_3(\mathbf{x}) = -3x_1 + 4x_2 \\ \text{p.o} \quad & g_1(\mathbf{x}): \quad 2x_1 + 0x_2 \leq 10 \\ & g_2(\mathbf{x}): \quad 0x_1 + 3x_2 \leq 9 \\ & g_3(\mathbf{x}): \quad 4x_1 + 0x_2 \leq 32 \\ & g_4(\mathbf{x}): \quad 5x_1 + 0x_2 \geq 5 \\ & x_1, x_2 \geq 0 \end{aligned}$$

$$\begin{aligned} f_3(1,0) &= -3 \cdot 1 + 4 \cdot 0 = -3 \\ f_3(5,0) &= -3 \cdot 5 + 4 \cdot 0 = -15 \\ f_3(5,3) &= -3 \cdot 5 + 4 \cdot 3 = -3 \\ f_3(1,3) &= -3 \cdot 1 + 4 \cdot 3 = 9 \end{aligned}$$

Idealna vrednost funkcije kriterijuma $f_3(\mathbf{x})$ jednaka je maksimalnoj vrednosti funkcije $f_3^*(x) = 9$ za marginalno rešenje $\mathbf{x}^{(3)*} = (1, 3)$.

Tabelarni prikaz posledice marginalnih rešenja na funkcije kriterijuma i skup ograničenja. Prve dve kolone tabele sadrže vrednosti promenljivih, zatim sledeće tri kolone sadrže ostvarene vrednosti kriterijumskih funkcija, dok poslednje četiri kolone sadrže ostvarene vrednosti ograničenja.

x_1	x_2	f_1	f_2	f_3	$g_1 \leq 10$	$g_2 \leq 9$	$g_3 \leq 32$	$g_4 \geq 5$
5	3	**19**	-10	-3	10	9	20	25
5	0	10	**5**	-15	10	0	20	25
1	3	11	-14	**9**	2	9	4	5

3 Metodi za rešavanje zadataka VKO

Pre izlaganja različitih metoda za rešavanje problema VKO napomenimo da se one uglavnom zasnivaju na svođenju problema na problem jednokriterijumske optimizacije (JKO).

Prilikom razmatranja metoda trebalo bi obratiti pažnju na sledeća pitanja [21]:

1. Da li je metoda kvantitativna ili kvalitativna?

2. Kako se izbor donosioca odluke uključuje u algoritam?
3. Da li donosilac odluke kontroliše izbor konačnog rešenja?
4. Kakve informacije i odgovori se očekuju od donosioca odluke?
5. Koje rezultate metoda pruža kao pomoć odlučivanju?
6. Na koju vrstu problema može biti primenjena?
7. Računski aspekti (programiranje, memorija i dr.)

Prema pristupu rešavanju zadatka metodi VKO dele se u sledeće tri grupe:

1) *A posteriori metodi* u kojima se donosilac odluke (DO) informiše o dominantnim (Pareto optimalnim) rešenjima matematičkog modela, a on na osnovu njih donosi konačnu odluku.

2) *A priori metodi* u kojima se informacije o odnosu DO prema kriterijumima ugrađuju u matematički model ili metodu *a priori*, tj. pre bilo kakvog rešavanja modela, a konačna odluka se odnosi na osnovu tako dobijenog rešenja.

3) *Interaktivni metodi* u kojima DO aktivno učestvuje tokom rešavanja modela. U njima se iterativno kombinuju metodi iz prethodne dve grupe, tj. DO prvo daje preliminarne informacije o svojim preferencijama, a zatim kada dobije rešenje, može promeniti informacije ili rešenje. Ovaj postupak se iterativno ponavlja sve dok DO ne bude konačno zadovoljan dobijenim rešenjem.

Od a priori metoda obradićemo relaksirani leksikografski metod i metod ε ograničenja i objasniti princip grupe metoda nazvanih metodi rastojanja. Kao predstavnika interaktivnih metoda opisaćemo metod interaktivnog kompromisnog programiranja.

Svi do sada razvijeni metodi višeciljnog odlučivanja (VCO) imaju sledeće karakteristike:

- skup ciljeva koji mogu biti kvantifikovani,
- skup dobro definisanih ograničenja, i
- proces dobijanja informacija (eksplicitnih ili implicitnih) o identifikovanim ciljevima.

Poslednja osobina je posebno značajna. Naime, većinu realnih ciljeva je vrlo teško kvantifikovati, pa je za korišćenje metoda iz ove grupe potrebno raspolagati procesom koji bi bio u stanju da obezbedi određeni nivo kvantifikacije svih ciljeva.

Poznati svetski autori saglasni su da se većina realnih problema može rešavati primenom interaktivnih metoda, kada donosilac odluke aktivno učestvuje u kreiranju i analizi efikasnih rešenja, a na kraju procesa bira konačno, najprihvatljivije rešnje.

U nastavku sledi opis najvažnijih metoda višekriterijumske optimizacije kao i opis njihove simboličke implementacije u programskom paketu MATHEMATICA. Opis implementacije je baziran na radu [25].

3.1 Metod globalnog kriterijuma

Metod globalnog kriterijuma izuzetno je jednostavan i ne zahteva preferencije o kriterijumima. Nakon određivanja idealnih vrednosti kriterijuma formira se pomoćni jednokriterijumski model sa ograničenjima kao u modelu i funkcijom kriterijuma

$$\min f_r(\mathbf{x}) = \sum_k \left(\frac{f_k^*(\mathbf{x}) - f_k(\mathbf{x})}{f_k^*(\mathbf{x})} \right)^r, \ r \geq 1.$$

Rešenje problema jednako je minimalnoj vrednosti jednokriterijumskog modela, a predstavlja odabranu varijantu zbira normalizovanih odstojanja ostvarenih vrednosti od idealnih vrednosti kriterijuma.

Primer 3.1.1 Razmatra se napred prikazani primer i najjednostavnija varijanta formiranja funkcije kriterijuma u pomoćnom modelu metoda globalnog kriterijuma kada je $r = 1$.

$$\begin{array}{lll} \max & f_1(\mathbf{x}) = 2x_1 + 3x_2 \\ \max & f_2(\mathbf{x}) = x_1 - 5x_2 \\ \max & f_3(\mathbf{x}) = -3x_1 + 4x_2 \\ \text{p.o.} & g1(\mathbf{x}): & 2x_1 + 0x_2 \leq 10 \\ & g_2(\mathbf{x}): & 0x_1 + 3x_2 \leq 9 \\ & g_3(\mathbf{x}): & 4x_1 + 0x_2 \leq 32 \\ & g_4(\mathbf{x}): & 5x_1 + 0x_2 \geq 5. \end{array}$$

Rešenje. Idealne vrednosti kriterijuma

$$f_1^*(\mathbf{x}) = 19, \ f_2^*(\mathbf{x}) = 5, \ f_3^*(\mathbf{x}) = 9.$$

Globalna funkcija (za $r = 1$) jednaka je

$$\begin{aligned} \min f_{r=1}(x) &= \frac{f_1^*(\mathbf{x}) - f_1(\mathbf{x})}{f_1^*(\mathbf{x})} + \frac{f_2^*(\mathbf{x}) - f_2(\mathbf{x})}{f_2^*(\mathbf{x})} + \frac{f_3^*(\mathbf{x}) - f_3(\mathbf{x})}{f_3^*(\mathbf{x})} \\ &= \frac{19 - (2x_1 + 3x_2)}{19} + \frac{5 - (x_1 - 5x_2)}{5} + \frac{9 - (-3x_1 + 4x_2)}{9} \end{aligned}$$

$$\min f_{r=1}(x) = 3 + 0.03x_1 0.40x_2.$$

x_1	x_2	$\min f_{r=1}(\mathbf{x}) = 3 + 0.03x_1 + 0.40x_2$
5	3	4.3
5	0	**3.1**
1	3	4.2

Odatle se dobija $\min f_{r=1}(\mathbf{x}) = 3 + 0.03x_1 0.40x_2 = 3.1$ za $\mathbf{x}^* = (5, 0)$.

3.2 Metod težinskih koeficijenata

Metod težinskih koeficijenata je najstariji metod za VKO-u koja je korišćena. Po ovom metodu uvode se težinski koeficijenti w_i za sve kriterijumske funkcije $Q_i(\mathbf{x})$, $i = 1, \ldots, l$, pa se problem optimizacije svodi na sledeću skalarnu optimizaciju

$$\begin{array}{ll} \max & Q(\mathbf{x}) = \sum_{i=1}^{l} w_i Q_i(\mathbf{x}) \\ \text{p.o.} & \mathbf{x} \in \mathbf{X}, \end{array} \tag{3.2.1}$$

gde težine w_i, $i=1,\ldots,l$ ispunjavaju sledeće uslove

$$\sum_{i=1}^{l} w_i = 1, \quad w_i \geq 0,\ i = 1,\ldots,l.$$

Često se koristi metod težinskih koeficijenata tako što se zadaju vrednosti ovih koeficijenata. Međutim, to uvek izaziva određene teškoće i primedbe na ovakav postupak, jer se unosi subjektivan uticaj na konačno rešenje preko zadatih vrednosti težinskih koeficijenata.

Glavna ideja u metodu težinskih koeficijenata je da se odaberu težinski koeficijenti w_i koji odgovaraju ciljnim funkcijama $Q_i(\mathbf{x})$, $i = 1,\ldots,l$. Mnogi autori su razvili sistematske pristupe u selektovanju težina, čiji se pregled može naći u [10], [11] i [28]. Jedna od poteškoća ovog metoda je da variranje težina konzistentno i neprekidno ne mora uvek da rezultuje u tačnoj i kompletnoj reprezentaciji Pareto optimalnog skupa. Ovaj nedostatak je diskutovan u [8].

Teorema 3.2.1 *Ako su svi težinski koeficijenti w_i pozitivni, onda je rešenje problema* (3.2.1) *Pareto optimalno rešenje polaznog problema VKO.*

Dokaz. Neka je $\mathbf{x}^*$ rešenje problema (3.2.1), i neka su svi težinski koeficijenti strogo pozitivni. Pretpostavimo da ono nije Pareto optimalno, tj. da postoji $\mathbf{x} \in S$ tako da za $i = 1,\ldots,l$ važi $Q_i(\mathbf{x}) \geq Q_i(\mathbf{x}^*)$, pri čemu važi bar jedna stroga nejednakost (recimo za indeks j). Kako je $w_i > 0$ za svako i, važi

$$\sum_{i=1}^{l} w_i Q_i(\mathbf{x}) > \sum_{i=1}^{l} w_i Q_i(\mathbf{x}^*)$$

pa dobijamo kontradikciju sa pretpostavkom da je $\mathbf{x}^*$ rešenje težinskog problema (3.2.1). Sledi da je $\mathbf{x}^*$ Pareto optimalno. □

Teorema 3.2.2 *Ako je za svako $i \in \{1,\ldots,l\}$ ispunjen uslov $w_i \geq 0$, onda je rešenje problema* (3.2.1) *slabi Pareto optimum polaznog problema VKO.*

Dokaz. Neka je $\mathbf{x}^*$ rešenje problema (3.2.1) i da je ispunjen uslov $w_i \geq 0$. Pretpostavimo da ono nije slabo Pareto optimalno, tj. da postoji $\mathbf{x} \in S$ tako da za $i = 1,\ldots,l$ važi $Q_i(\mathbf{x}) > Q_i(\mathbf{x}^*)$. Svi koeficijenti w_i su nenegativni i bar jedan je strogo veći od nule (zbog $\sum_{i=1}^{l} w_i = 1$), pa važi

$$\sum_{i=1}^{l} w_i Q_i(\mathbf{x}) > \sum_{i=1}^{l} w_i Q_i(\mathbf{x}^*)$$

pa dobijamo kontradikciju sa pretpostavkom da je $\mathbf{x}^*$ rešenje težinskog problema. Dakle, $\mathbf{x}^*$ je slabi Pareto optimum. □

Teorema 3.2.3 *Ako je rešenje problema* (3.2.1) *jedinstveno, ono je onda i Pareto optimalno.*

Dokaz. Neka je $\mathbf{x}^*$ jedinstveno rešenje problema (3.2.1). Pretpostavimo da ono nije Pareto optimalno rešenje problema VKO, tj. da postoji $\mathbf{x} \in S$ tako da za $i = 1, \ldots, l$ važi $Q_i(x) \geq Q_i(x^*)$, pri čemu važi bar jedna stroga nejednakost (recimo za indeks j). Primetimo da to znači $\mathbf{x} \neq \mathbf{x}^*$.

Kako je $w_i \geq 0$ za svako i, važi

$$\sum_{i=1}^{l} w_i Q_i(\mathbf{x}) \geq \sum_{i=1}^{l} w_i Q_i(\mathbf{x}^*)$$

Ako bi važila stroga nejednakost, onda $\mathbf{x}^*$ ne bi bilo rešenje problema (3.2.1). Dakle, važi jednakost. To znači da postoje dva različita rešenja $\mathbf{x}$ i $\mathbf{x}^*$ problema (3.2.1), što je kontradikcija. □

Pokazaćemo sada jedno jače tvrđenje.

Teorema 3.2.4 *Ako su svi $w_i > 0$, $i \in \{1, \ldots, l\}$, tada je rešenje problema* (3.2.1) *strogi Pareto optimum problema VKO.*

Dokaz. Neka je $\mathbf{x}^*$ rešenje težinskog problema. Pokazali smo da je ono Pareto optimalno. Dokažimo da je to rešenje takođe i strogi Pareto optimum sa konstantom

$$M = (k-1) \max_{i,j} \frac{w_j}{w_i}.$$

Pretpostavimo suprotno, da postoje $\mathbf{x} \in S$ i indeks i takvi da je $Q_i(\mathbf{x}) > Q_i(\mathbf{x}^*)$ pri čemu za svako j za koje je $Q_j(\mathbf{x}^*) > Q_j(\mathbf{x})$ važi $Q_i(\mathbf{x}^*) - Q_i(\mathbf{x}) < M(Q_j(\mathbf{x}) - Q_j(\mathbf{x}^*))$. Zamenom

$$M = \frac{(k-1)w_j}{w_i}$$

dobijamo

$$w_i \frac{Q_i(\mathbf{x}^*) - Q_i(\mathbf{x})}{k-1} < w_j \left(Q_j(\mathbf{x}) - Q_j(\mathbf{x}^*)\right) > 0 \ .$$

Dakle, za svako $j \neq i$ za koje je $Q_j(\mathbf{x}^*) > Q_j(\mathbf{x})$ važi prethodna nejednakost. Za one indekse $j \neq i$ za koje je $Q_j(\mathbf{x}^*) \leq Q_j(\mathbf{x})$ gornja nejednakost svakako važi. Dakle, za svako $j \neq i$ važi

$$w_i \frac{Q_i(\mathbf{x}^*) - Q_i(\mathbf{x})}{k-1} < w_j \left(Q_j(\mathbf{x}) - Q_j(\mathbf{x}^*)\right),$$

pa sabiranjem ovih nejednakosti za $j = 1, \ldots, i-1, i+1, \ldots, l$ dobijamo

$$w_i \left(Q_i(\mathbf{x}^*) - Q_i(\mathbf{x})\right) < \sum_{j=1, j\neq i}^{l} w_j \left(Q_j(\mathbf{x}) - Q_j(\mathbf{x}^*)\right)$$

tj.

$$\sum_{j=1}^{l} w_j Q_j(\mathbf{x}^*) < \sum_{j=1}^{l} w_j Q_j(\mathbf{x}) \ .$$

Dobijamo da $\mathbf{x}^*$ nije rešenje težinskog problema tj. dolazimo do kontradikcije. Zaključujemo da je $\mathbf{x}^*$ zaista strogi Pareto optimum polaznog problema. □

Težinski koeficijent na neki način treba da predstavi značaj kriterijumske funkcije kojoj je dodeljen. Da bismo to postigli najpre moramo da normalizujemo kriterijumske funkcije tj. da ih promenimo tako da imaju približno jednake vrednosti, a pri tome da zadrže sve bitne osobine. Na primer, ako je kriterijumska funkcija linearna $Q_j(\mathbf{x}) = \sum_{i=1}^{n} a_i x_i$, tada će normalizovan oblik ove funkcije biti

$$\frac{Q_j(\mathbf{x})}{\sum_{i=1}^{n} a_i}\,.$$

Ako je kriterijumska funkcija ograničena tada se za njen normalizovan oblik može uzeti sledeća funkcija

$$\overline{Q_i}(\mathbf{x}) = \frac{Q_i(\mathbf{x}) - Q_i^*}{\max_{\mathbf{x}\in S} Q_i(\mathbf{x}) - Q_i^*}\,,$$

čiji je kodomen $[0, 1]$.

Ukoliko donosilac odluke sam definiše težinske koeficijente, onda ovaj metod spada u grupu a priori metoda. Međutim, najčešće se malo zna o tome kako koeficijente treba izabrati. Zato se obično težinski problem rešava za razne vrednosti vektora $(w_1, \ldots, w_l)$ i na taj način dobijaju različita rešenja među kojima DO bira ono koje mu najviše odgovara. Ako se koristi ovakav pristup, onda ovaj metod postaje a posteriori metod.

Glavna mana ove metode je teškoća određivanja težinskih koeficijenata kada nemamo dovoljno informacija o problemu.

Sledi opis implementacije metoda težinskih koeficijenata.

Funkcijom `Compositions[n,l]` možemo odrediti "l-dimenzionalne tačke", čiji zbir koordinata daje n. Kada tu listu podelimo sa brojem n, možemo dobiti tačke u intervalu $[0, 1]$ čije koordinate mogu predstavljati koeficijente w_i. Na taj način smo obezbedili automatsko generisanje koeficijenata w_i potrebnih za realizaciju metoda. Ostavljena je i mogućnost da korisnik sam izabere koeficijente w_i. To se postiže tako što pri pozivu funkcije `MultiW`, kojom se implementira metod težinskih koeficijenata, poslednji parametar zadamo kao nepraznu listu, tj. zadamo koeficijente w_i u obliku liste.

Sledi program kojim je implementiran metod težinskih koeficijenata [25].

Ulazne veličine:

q_,var_List - ciljna funkcija i lista njenih parametara;

constr_List - lista ograničenja;

*var_* - korak podele intervala [0,1];

w1_List - lista težinskih koeficijenata u intervalu [0,1]. Prazna lista kao vrednost parametra $w1$ označava da će težinski koeficijentu biti generisani pomoću funkcije *Compositions*. Inače, pretpostavlja se da je svaki element $w1[[i]], 1 \leq 1 \leq Length[w1]$ liste $w1$ jedan mogući skup koeficijenata w_j, $j = 1, \ldots, l$: $w[[j]] = w1[[i, j]], j = 1, \ldots, l$.

Lokalne promenljive:
fun - formirana funkcija za jednodimenzionalnu optimizaciju.

```
MultiW[q_, constr_List, var_List, w1_List] :=
 Module[{i=0,k,l=Length[q],res={},W={},fun,sk={},qres={},mxs={},m,ls={}},
   If[w1=={},
    k=Input["Initial sum of weighting coefficients?"]; W=Compositions[k,l]/k,
    W=w1;
   ];
   Print["Weighting Coefficients: "]; Print[W];
   Print["Single-objective problems: "];
   k = Length[W];
   For[i=1, i<=k, i++,
       fun = Simplify[Sum[W[[i,j]]*q[[j]], {j,l}]];  (* 3V *)
       AppendTo[res, Maximize[fun, constr, var]];    (* 1V *)
   ];
   Print["Solutions of single-objective problems: "];  Print[res];
   For[i=1, i<=k, i++,
      AppendTo[qres, q/.res[[i, 2]] ];     AppendTo[mxs, res[[i, 1]] ];
   ];
   Print["Choose the best solution: "];
   m=Max[mxs];
   For[i=1, i<=Length[mxs], i++,
      If[m==mxs[[i]],  AppendTo[ls, {qres[[i]], res[[i,2]]}]; ]
   ];
   Return[ls];
 ]
```

Za pozitivne težine i konveksni problem, optimalna rešenja jednokriterijumskog problem jesu Pareto optimalna [33], tj. minimiziranje odgovarajućeg jednokriterijumskog problema je dovoljno za Pareto optimalnost. Međutim, formulacija ne obezbeđuje neophodan uslov za Pareto optimalnost [34]. Kada je višekriterijumski problem konveksan, primena funkcije `W=Compositions[k,l]` produkuje b Pareto optimalnih rešenja, gde integer b ispunjava $1 \leq b \leq \binom{k+l-1}{l-1}$.

Primer 3.2.1 Maksimizirati funkcije koje se nalaze u listi:

$$Q = \{Q_1(x,y) = x + y, \;\; Q_2(x,y) = 2x - y\}$$

prema ograničenjima

$$-3x + 5y \leq 9, \; 3x + 2y \quad \leq \quad 12, \; 5x - 4y \leq 9, \; x \geq 0, \; y \geq 0.$$

Prvo se koristi prazna lista za parametar $w1$, i na taj način se koeficijenti w_i generišu pomoću funkcije *Compositions*:

```
In[1]:=MultiW[{x+y,2x-y},{-3x+5y<=9,3x+2y<=12,5x-4y<=9,x>=0,y>=0},{x,y},{}]
```

U slučaju $k = 5$, izraz `w=Compositions[k,l]/k` produkuje

w={{0, 1},{$\frac{1}{5}$, $\frac{4}{5}$}, {$\frac{2}{5}$, $\frac{3}{5}$},{$\frac{3}{5}$, $\frac{2}{5}$}, {$\frac{4}{5}$, $\frac{1}{5}$}, {1, 0}}

Takođe, jednokriterijumski problemi optimizacije dati su sledećim unutrašnjim reprezentacijama.

Za $\{w_1, w_2\}$={0, 1}:
{2x-y,{$-3x + 5y <= 9, 3x + 2y <= 12, 5x - 4y <= 9, x >= 0, y >= 0$},{$x, y$}}

Za $\{w_1, w_2\}$={$\frac{1}{5}$, $\frac{4}{5}$}:
{$\frac{3}{5}(3x - y)$, {$-3x + 5y <= 9, 3x + 2y <= 12, 5x - 4y <= 9, x >= 0, y >= 0$},{$x, y$}}

Za $\{w_1, w_2\}$=$\{\frac{2}{5}, \frac{3}{5}\}$:
$\{\frac{1}{5}(8x - y)$, $\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x, y\}\}$

Za $\{w_1, w_2\}$=$\{\frac{3}{5}, \frac{2}{5}\}$:
$\{\frac{1}{5}(7x + y)$, $\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x, y\}\}$

Za $\{w_1, w_2\}$=$\{\frac{4}{5}, \frac{1}{5}\}$:
$\{\frac{3}{5}(2x + y)$, $\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x, y\}\}$

Za $\{w_1, w_2\}$=$\{1, 0\}$:
$\{x + y$, $\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x, y\}\}$

Elementi liste *res*, koji odgovaraju rešenjima jednokriterijumskih problema, jednaki su

$\{\{\frac{9}{2}$,$\{x->3$,$y->\frac{3}{2}\}\}$,$\{\{\frac{9}{2}$,$\{x->3$,$y->\frac{3}{2}\}\}$,$\{\{\frac{9}{2}$,$\{x->3$,$y->\frac{3}{2}\}\}$,
$\{\{\frac{9}{2}$,$\{x->3$,$y->\frac{3}{2}\}\}$,$\{\{\frac{9}{2}$,$\{x->3$,$y->\frac{3}{2}\}\}$,$\{\{5$,$\{x->2$,$y->3\}\}\}$

a rezultat je lista koja sadrži vrednosti ciljnih funkcija u tački $\{x-> 2, y-> 3\}$, koja odgovara maksimalnoj vrednosti težinske funkcije (koja je jednaka 5):

Out[1]=$\{\{\{5$,$1\}$, $\{x-> 2, y-> 3\}\}$

Sada se vrednosti koeficijenata w_i definišu od strane korisnika. Vrednosti težinskih koeficijenata sadržane su u svakom elementu liste
$\{\{1, 0\}, \{0.9, 0.1\}, \{0.875, 0.125\}, \{0.8, 0.2\}, \{0, 1\}\}$.

```
In[2]:=MultiW[{x+y,2x-y},{-3x+5y<=12,5x-4y<=9,x>=0,y>=0},{x,y},
                {{1,0},{0.9,0.1},{0.875,0.125},{0.8,0.2},{0,1}}]
```

Sledeći problemi se rešavaju u ciklusu:

Za $\{w_1, w_2\}$=$\{1, 0\}$:
$\{x + y$,$\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x$,$y\}\}$
Za $\{w_1, w_2\}$=$\{0.9, 0.1\}$:
$\{1.1x + 0.8y$,$\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x$,$y\}\}$
Za $\{w_1, w_2\}$=$\{0.875, 0.125\}$:
$\{1.125x + 0.75y$,$\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x$,$y\}\}$
Za $\{w_1, w_2\}$=$\{0.8, 0.2\}$:
$\{1.2x + 0.6y$,$\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x$,$y\}\}$
Za $\{w_1, w_2\}$=$\{0, 1\}$:
$\{2x - y$,$\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x$,$y\}\}$

Dobija se sledeća vrednost za listu *res* koja odgovara rešenjima ovih problema:

$\{\{5$,$\{x->2$,$y->3\}\}$,$\{4.6$,$\{x->2.$,$y->3.\}\}$,$\{4.5$,$\{x->3.$,$y->1.5\}\}$,
$\{4.5$,$\{x->3.$,$y->1.5\}\}$,$\{\frac{9}{2}$,$\{x->3$,$y->\frac{3}{2}\}\}\}$

a rezultat je

Out[2]= $\{\{\{5,1\},\{x-> 2,y-> 3\}\}\}$

3.3 Metod sa funkcijom korisnosti

Za primenu ovog metoda neophodno je znati preferencije za kriterijume koje se unapred uključuju u model. Preferencije se određuju na osnovu analize značajnosti kriterijuma za svaki konkretan slučaj, i to je najvažnija, kardinalna informacija o

problemu. Najpre se formira pomoćni jednokriterijumski model u vidu funkcije korisnosti $U(f)$

$$\max U(f) = U\{f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_l(\mathbf{x})\}$$

koja se rešava uz primenu postojećih ograničenja (p.o.)

$$g_i(\mathbf{x}) \leq 0,\ i \in I$$
$$x_j \geq 0,\ \text{za svako } j \in J.$$

Potom se nađeno optimalno rešenje uvodi u svaki od kriterijuma i određuju njihove konkretne vrednosti. Funkcija korisnosti zavisi od prirode rešavanog problema. Najčešće se koriste sledeće separabilne funkcije od zadatih funkcija modela višeciljnog odlučivanja:

$$\begin{aligned} U(f) &= \sum_k f_k(\mathbf{x}) \\ U(f) &= \prod_k f_k(\mathbf{x}) \\ U(f) &= \sum_k t_k f_k(\mathbf{x}). \end{aligned}$$

Definicija 3.3.1 Funkcija $U : \mathbf{R}^k \to \mathbf{R}$ kojoj je domen kriterijumski prostor i koja odražava težnje donosioca odluke zove se vrednosna funkcija ili funkcija korisnosti.

Boljoj odluci odgovara veća vrednost vrednosne funkcije. Iz tog razloga, vrednosna funkcija bi trebalo da bude strogo opadajuća po svakoj promenljivoj. Dakle, problem se svodi na problem jednokriterijumske optimizacije

$$\max\ U(f(\mathbf{x}))\ p.o.\ \mathbf{x} \in S. \tag{3.3.1}$$

Definicija 3.3.2 *Funkcija $f : \mathbf{R}^k \to \mathbf{R}$ je strogo rastuća (opadajuća) ako za proizvoljne $\mathbf{x} = (x_1, \ldots, x_k)$ i $\mathbf{y} = (y_1, \ldots, y_k)$ iz $\mathbf{R}^k$ važi, redom*

$$(x_i < y_i \text{ za svako } i = 1, \ldots, k) \Rightarrow f(\mathbf{x}) < f(\mathbf{y}).$$

$$(x_i \leq y_i \text{ za svako } i = 1, \ldots, k) \Rightarrow f(\mathbf{x}) > f(\mathbf{y})).$$

Definicija 3.3.3 *Funkcija $f : \mathbf{R}^k \to \mathbf{R}$ je jako rastuća (opadajuća) ako za proizvoljne $\mathbf{x} = (x_1, \ldots, x_k)$ i $\mathbf{y} = (y_1, \ldots, y_k)$ iz $\mathbf{R}^k$ važi, redom*

$$(x_i \leq y_i \text{ za svako } i = 1, \ldots, k \text{ i } x_j < y_j \text{ za neko } j) \Rightarrow f(\mathbf{x}) < f(\mathbf{y}).$$

$$(x_i \leq y_i \text{ za svako } i = 1, \ldots, k \text{ i } x_j < y_j \text{ za neko } j) \Rightarrow f(\mathbf{x}) > f(\mathbf{y}).$$

Teorema 3.3.1 *Za jako opadajuću funkciju U rešenje problema* (3.3.1) *je Pareto optimalno rešenje početnog višekriterijumskog problema.*

Dokaz. Pretpostavimo suprotno, tj. da neko rešenje problema (3.3.1) (označimo ga sa $\mathbf{x}^*$) nije Pareto optimalno. To znači da postoji $x \in S$ za koje važi da je $f_i(\mathbf{x}) \leq f_i(\mathbf{x}^*)$, pri čemu postoji indeks j za koji važi stroga nejednakost. Kako je funkcija U jako opadajuća, sledi $U(f(\mathbf{x})) > U(f(\mathbf{x}^*))$. Dobijamo kontradikciju sa pretpostavkom da se maksimum funkcije $U \circ f$ dostiže u tački $\mathbf{x}^*$. □

Metod funkcije korisnosti je jednostavan i jako dobar u slučajevima kada ga je moguće primeniti. Međutim, u praksi su česti slučajevi kada je neprekidnom stabilnom funkcijom jako teško ili nemoguće izraziti želje donosioca odluke. Osim toga može se desiti da donosilac odluke nije u stanju da strogo matematički izrazi svoje težnje ili da ih delimično promeni nakon što bude upoznat sa rezultatima.

Primer 3.3.1 Rešava se problem

$$\begin{aligned}
\max \quad & f_1(\mathbf{x}) = 2x_1 + 3x_2 \\
\max \quad & f_2(\mathbf{x}) = x_1 - 5x_2 \\
\max \quad & f_3(\mathbf{x}) = -3x_1 + 4x_2 \\
\text{p.o.} \quad & g1(\mathbf{x}): \quad 2x_1 + 0x_2 \leq 10 \\
& g_2(\mathbf{x}): \quad 0x_1 + 3x_2 \leq 9 \\
& g_3(\mathbf{x}): \quad 4x_1 + 0x_2 \leq 32 \\
& g_4(\mathbf{x}): \quad 5x_1 + 0x_2 \geq 5.
\end{aligned}$$

sa sledećim težinama za kriterijume $t_1 = 0.6, t_2 = 0.3$ i $t_3 = 0.1$.

Uvođenjem vrednosti težinskih koeficijenata u funkciju korisnosti dobija se

$$\max U(f) = t_1 \cdot f_1(\mathbf{x}) + t_2 \cdot f_2(x) + t_3 \cdot f_3(x)$$
$$\max U(f) = 0.6(2x_1 + 3x_2) + 0.3(x_1 - 5x_2) + 0.1(-3x_1 + 4x_2),$$

ili posle sređivanja

$$\max U(f) = 0.9x_1 0.7x_2.$$

Maksimum se određuje iz sledeće tabele

x_1	x_2	$\max U(f) = 0.9x_1 0.7x_2$	f_1	f_2	f_3
5	3	2.4			
5	0	**4.5**	5	10	-15
1	3	-1.2			

Rešenje. $\max f(\mathbf{x}) = 0.9x_1 0.7x_2 = 4.5$ za $\mathbf{x}* = (5, 0)$.

3.4 Metod ograničavanja kriterijuma

Metod ograničavanja kriterijuma jedan je od najstarijih metoda rešavanja modela višeciljnog odlučivanja, čiji se elementi direktno ili indirektno uključuju u procedure drugih metoda. Po ovom metodu vrši se optimizacija najznačajnijeg kriterijuma, neka je to s-ti kriterijum, dok se ostali prevode u ograničenja sa zahtevima da se ostvare željene vrednosti tih kriterijuma. Kao i kod metoda sa funkcijom korisnosti, i u ovom metodu se preferencije o kriterijumima zadaju unapred (a priori) i to je

najvažnija, kardinalna informacija. Prvo se definiše jednokriterijumski pomoćni model koji odgovara polaznom modelu višeciljnog odlučivanja

$$\begin{aligned} \max \quad & f_s(\mathbf{x}) \\ \text{p.o.} \quad & g_i(\mathbf{x}) \leq 0,\ i = 1, 2, ..., l \\ & f_k(\mathbf{x}) \begin{cases} \geq L_k \\ \leq H_k \end{cases}, k \neq s, k = 1, 2, ..., l \\ & x_j \geq 0,\ j = 1, 2, ..., n. \end{aligned}$$

gde veličine L_k i H_k predstavljaju donju i gornju granicu, respektivno, za k-te kriterijume prevedene u ograničenja, $k \neq s$, $k = 1, 2, ..., l$. Određivanjem optimalnog rešenja modela dobija se uslovljena optimalna vrednost s-tog kriterijuma, koja se uvodi u ostale kriterijume radi proračuna njihovih vrednosti. U tu svrhu mogu se koristiti odgovarajuće granice i izravnavajuće promenljive pripradajućih ograničenja. Za razliku od prethodnih metoda višeciljnog odlučivanja koji zahtevaju određivanje maksimalnih vrednosti, ovaj metod se sprovodi uvođenjem samo donjih granica za k-te kriterijume u ograničenja. Pri tome se mora imati na umu koje su moguće vrednosti tih kriterijuma, što se određuje na osnovu analize tabele sa idealnim vrednostima kriterijuma i posledicama marginalnih rešenja na kriterijume.

Primer 3.4.1 Neka se u napred razmatranom primeru smatra da je prvi kriterijum ($s = 1$) od najvećeg značaja. Potrebno je odrediti maksimalnu vrednost tog kriterijuma sa zahtevom da se ostalim kriterijumima ostvare najmanje 60% idealnih vrednosti.

$$0.60 f_2^*(\mathbf{x}) = 0.60 \cdot 5 = 3.0$$
$$0.60 f_3^*(\mathbf{x}) = 0.60 \cdot 9 = 5.4$$

x_1	x_2 f_1	$g_1 \leq 10$	$g_2 \leq 9$	$g_3 \leq 32$	$g_4 \geq 5$	$g_5 \geq 3$	$g_6 \geq 5.4$	
5	3	**19**	10	9	20	25	-10	-3
5	0	10	10	0	20	25	5	-15
1	3	11	2	9	4	5	-14	9

Optimalno Rešenje je $f_1(\mathbf{x}) = 19$ za $\mathbf{x}^{(1)*} = (5, 3)$.

3.5 Leksikografska višekriterijumska optimizacija

Često se do optimalnog rešenja dolazi posle uzastopnog donošenja odluka. Prvo se nađe optimalno rešenje za najvažniju funkciju cilja. Ako je optimalno rešenje jedinstveno, tada je problem rešen. Međutim, ako optimalno rešenje nije jedinstveno, tada se na skupu svih optimalnih rešenja optimizuje funkcija cilja koja je druga po važnosti. Ako je optimalno rešenje sada jedinstveno, problem je rešen; ako nije, optimizuje se funkcija cilja treća po važnosti na skupu optimalnih rešenja prve i druge funkcije cilja itd.

Dakle posmatrajmo problem minimizacije date uređene sekvence ciljnih funkcija

$$Q_1(x), \ldots, Q_l(x)$$

i skup ograničenja

$$\mathbf{x} \in \mathbf{X}.$$

Trebalo bi da se reši sledeći skup konveksnih uslovnih nelinearnih programa

$$\begin{array}{ll} \max & Q_i(\mathbf{x}) \\ \text{p.o.} & \mathbf{x} \in \mathbf{X} \\ & Q_j(\mathbf{x}) \geq a_j, \quad j = 1, \ldots, i-1,\ i \geq 2, \end{array}$$

gde su $a_j = Q_j(\mathbf{x_j}^*)$, $j = 1, \ldots, i-1$ optimalne vrednosti prethdono postavljenih jednokriterijumskih problema na nivoima prioriteta $j = 1, \ldots, i-1$, $i \geq 2$.

Nejednakosti u poslednjem problemu mogu biti zamenjene jednakostima [24]. leksikografski metod spada u grupu apriornih metoda.

Pretpostavimo da su kriterijumske funkcije $Q_1, \ldots, Q_l$ poređane od najvažnije do najmanje važne. Polazni problem višekriterijumske optimizacije se rešava sledećim algoritmom

1. $S_0 := S,\ i := 1$
2. Rešava se problem:

   $$\text{Minimizirati } Q_i(\mathbf{x}) \text{ pod uslovom } \mathbf{x} \in S_{i-1}$$

   Neka je rešenje ovog problema $\mathbf{x}^{(i)^*}$.
3. Ako je $\mathbf{x}^{(i)^*}$, dobijeno u koraku 2, jedinstveno rešenje, ono se proglašava za rešenje višekriterijumskog problema i algoritam se završava.
4. Formira se skup $S_i := \{\mathbf{x} \in S_{i-1} \mid Q_i(\mathbf{x}) \geq Q_i(\mathbf{x}^{(i)^*})\}$
5. Ako je $i = l$ skup S_l se proglašava za skup rešenje problema višekriterijumske optimizacije, i algoritam se završava.
6. Stavlja se $i := i + 1$ i vraćamo se na korak 2.

Jedan od načina za implementaciju višekriterijumske optimizacije dat je funkcijom *MultiLex* [25].

```
MultiLex[q_List,constr_List,var_List]:=
  Module[{res={},s,f=constr,l=Length[q] ,i},
     For[i=1,i<=l,i++,
         s=Maximize[q[[i]],f,var];       (* 2V *)
         If[i<l, AppendTo[f,q[[i]]>=First[s]]  ];   (* 3V,2V *)
         AppendTo[res,{q/.Last[s],Last[s]}];
     ];
     Print[res]; Return[{q/.Last[s],Last[s]}];
  ]
```

Primer 3.5.1 Problem maksimizacije funkcija koje su sadržane u listi
$\{8x1 + 12x2, 14x1 + 10x2, x1 + x2\}$
prema ograničenjima
$\{8x1 + 4x2 <= 600, 2x1 + 3x2 <= 300, 4x1 + 3x2 <= 360, 5x1 + 10x2 >= 600,$

$x1 >= 0, x2 >= 0\}$
može biti rešen koristeći izraz

```
In[3]:=MultiLex[{8x1+12x2,14x1+10x2,x1+x2},
   {8x1+4x2<=600,2x1+3x2<=300,4x1+3x2<=360,5x1+10x2>=600,x1>=0,x2>=0},
   {x1,x2}]
```

Unutrašnje reprezentacije jednokriterijumskih problema su:

za $i = 1$:

$8x1 + 12x2,$
$\{8x1 + 4x2 <= 600,\ 2x1 + 3x2 <= 300,\ 4x1 + 3x2 <= 360,\ 5x1 + 10x2 >= 600,$
$x1 >= 0, x2 >= 0\},$
$\{x1, x2\}$

za $i = 2$:

$14x1 + 10x2,$
$\{8x1 + 4x2 <= 600,\ 2x1 + 3x2 <= 300,\ 4x1 + 3x2 <= 360,\ 5x1 + 10x2 >= 600,$
$x1 >= 0,\ x2 >= 0,\ 8x1 + 12x2 >= 1200\},$
$\{x1, x2\}$

za $i = 3$:

$x1 + x2,$
$\{8x1 + 4x2 <= 600,\ 2x1 + 3x2 <= 300,\ 4x1 + 3x2 <= 360,\ 5x1 + 10x2 >= 600,$
$x1 >= 0,\ x2 >= 0,\ 8x1 + 12x2 >= 1200,\ 14x1 + 10x2 >= 1220\},$
$\{x1, x2\}$

Rešenja uzastopnih optimizacionih problema su zapamćena u listi res, koja je jednaka

```
{{1200,{x1->0,x2->100}},{1220,{x1->30,x2->80}},{110,{x1->30,x2->80}}}
```

Teorema 3.5.1 *Rešenje dobijeno leksikografskom metodom je Pareto optimalno rešenje problema višekriterijumske optimizacije.*

Dokaz. Označimo sa $\mathbf{x}^*$ rešenje dobijeno leksikografskim metodom. Pretpostavimo da ono nije Pareto optimalno tj. da postoji neko $x \in S$ tako da je $Q_i(x) \geq Q_i(x^*)$ za svako $i = 1, \ldots, l$ pri čemu za neko k važi $Q_k(\mathbf{x}) > Q_k(\mathbf{x}^*)$. Mogu nastati dva slučaja. Prva mogućnost je da je $\mathbf{x}^*$ jedinstveno rešenje i da su iskorišćeni svi kriterijumi do j-tog (to znači da je na kraju prethodno opisanog algoritma $i = j$). Kako je $\mathbf{x} \in S = S_0$ i $Q_1(\mathbf{x}) \geq Q_1(x^*)$ sledi da je $Q_1(\mathbf{x}) = Q_1(\mathbf{x}^*)$, kao i $x \in S_1$. Kako je sada $x \in S_1$ i $Q_2(x) \geq Q_2(x^*)$ sledi $Q_2(x) = Q_2(x^*)$ i $\mathbf{x} \in S_2$. Nastavljajući ovakvo rezonovanje zaključujemo $Q_{j-1}(\mathbf{x}) = Q_{j-1}(\mathbf{x}^*)$ i $\mathbf{x} \in S_{j-1}$. Kako je $\mathbf{x}^*$ jedinstveno rešenje problema

$$\text{Minimizirati } Q_j(\mathbf{x}) \text{ pod uslovom } \mathbf{x} \in S_{j-1}$$

a pri tome važi $Q_j(\mathbf{x}) \geq Q_j(\mathbf{x}^*)$ i $\mathbf{x} \in S_{j-1}$ zaključujemo da mora biti $\mathbf{x} = \mathbf{x}^*$ pa je $Q_k(\mathbf{x}) = Q_k(\mathbf{x}^*)$ odakle dobijamo kontradikciju sa pretpostavkom $Q_k(\mathbf{x}) > Q_k(\mathbf{x}^*)$.

Druga mogućnost je da su iskorišćeni svi kriterijumi (tj. u prethodnom algoritmu je $i = l$). Potpuno analogno kao u prvom slučaju se pokazuje da $\mathbf{x} \in S_k$ i da za svako $i = 1, \ldots, k$ važi $Q_i(\mathbf{x}) = Q_i(\mathbf{x}^*)$, što je kontradikcija sa $Q_k(\mathbf{x}) > Q_k(\mathbf{x}^*)$. □

Iz ovog dokaza se može zaključiti da se ograničenje oblika $Q_j(x) \leq Q_j(\mathbf{x}^{(i)*})$ može zameniti ograničenjem $Q_j(\mathbf{x}) = Q_j(\mathbf{x}^{(i)*})$.

Mane leksikografske metode su očigledne:

- U praksi je često teško odrediti koji je kriterijum važniji od drugog.
- Najčešće se dešava da se jedinstveno rešenje dobije pre nego što se iskoriste svi kriterijumi, ili čak nakon korišćenja samo jednog kriterijuma. Na taj način, neki kriterijumi uopšte ne učestvuju u donošenju odluke, što je krajnje nepoželjno.

Zbog toga klasična leksikografski medod ima jako ograničenu primenu.

3.6 Relaksirani leksikografski metod

Hijerarhijski metod je modifikacija leksikografskog metoda, u kome se koriste ograničenja oblika [22]

$$Q_j(\mathbf{x}) \geq \left(1 + \frac{\delta_j}{100}\right) Q_j(\mathbf{x_j}^*).$$

Relaksacija se sastoji u povećanju desne strane ograničenja za procenat $Q_j(\mathbf{x_j}^*) \cdot \delta_j$. Variranjem parametra δ_j mogu se generisati različite Pareto optimalne tačke [18]. Još jedna varijacija leksikografskog metoda je uvedena u [30], gde su ograničenja jednaka

$$Q_j(\mathbf{x}) \geq Q_j(\mathbf{x_j}^*) - \delta_j.$$

U ovom slučaju, $\delta_j \leq 100$ jesu pozitivne tolerancije definisane od strane DO.

Relaksirani leksikografski metod je iterativni postupak u kome se rešavaju jednokriterijumski zadaci optimizacije. Pretpostavlja se da su od strane DO dati prioriteti kriterijuma i da su u skladu sa njima dodeljeni indeksi kriterijumima. U relaksiranoj leksikografskoj metodi se po svakom od p kriterijuma rešava jednokriterijumski zadatak optimizacije. Pri tome se u narednoj iteracijine postavlja kao ograničenje zahtev da rešenje bude optimalno po kriterijumu višeg prioriteta, već se ono relaksira tako da se zahteva da rešenje bude u okolini optimalnog rešenja dobijenog u prethodnoj iteraciji. Nataj način, svaki kriterijum utiče na konačno rešenje.

DO zadaje redosled kriterijuma po značajnosti. Pored toga, svakom kriterijumu, uzimajući poslednji, dodeljuje se vrednost δ_k , $k = 1, \ldots, l-1$, za koju kriterijum višeg prioriteta sme da odstupi od svoje optimalne vrednosti. Metoda obuhvata izvršavanje sledećih l koraka

Korak 1. Rešiti problem

$$\begin{array}{ll} \max & Q_1(\mathbf{x}) \\ \text{p.o.} & \mathbf{x} \in \mathbf{X} \end{array}$$

Rešenje ovog problema označimo sa Q_1^*.

Korak 2. Rešiti problem

$$\begin{array}{ll} \max & Q_2(\mathbf{x}) \\ \text{p.o.} & \mathbf{x} \in \mathbf{X} \\ & Q_1(\mathbf{x}) \geq Q_1^* - \delta_1 \end{array}$$

Korak p**.** Za svako $3 \leq p \leq l$ rešiti jednokriterijumski problem

$$\begin{array}{ll} \max & Q_p(\mathbf{x}) \\ \text{p.o.} & \mathbf{x} \in \mathbf{X} \\ & Q_j(\mathbf{x}) \geq Q_j^* - \delta_j,\ j = 1, \ldots, l-1 \end{array}$$

Za rešenje polaznog modela se usvaja rezultat dobijen u poslednjem koraku, a vrednosti funkcija cilja za dobijeno rešenje se moraju posebno računati.

Rešenje dobijeno ovim metodom obezbeđuje slabi Pareto optimum, a ako je rešenje jedinstveno, ono je i Pareto optimalno. Ako je dopustiva oblast konveksna, podešavanjem parametara $\delta_k, k = 1, \ldots, l-1$, može se dobiti bilo koje Pareto optimalno rešenje.

Teorema 3.6.1 *Rešenje dobijeno relaksiranim leksikografskim metodom je slabi Pareto optimum problema višekriterijumske optimizacije.*

Dokaz. Pretpostavimo suprotno, da rešenje $\mathbf{x}^*$ dobijeno relaksiranim leksikografskim metodom nije slabo Pareto optimalno. To znači da postoji neko $\mathbf{x} \in S$ tako da za svako $i = 1, \ldots, l$ važi $Q_i(\mathbf{x}) \geq Q_i(x^*)$. Kako je $Q_i(\mathbf{x}^*) = Q_i^*$ tim pre važi $Q_i(\mathbf{x}) \geq Q_i^* - \delta_i,\ i = 1, \ldots, k-1$, pa kako je $Q_k(\mathbf{x}) > Q_k(\mathbf{x}^*)$ dobijamo kontradikciju sa pretpostavkom da je $\mathbf{x}^*$ rešenje dobijeno relaksiranim leksikografskom metodom. □

Teorema 3.6.2 *Ako se relaksiranim leksikografskim metodom dobija jedinstveno rešenje, ono je Pareto optimalno.*

Dokaz. Pretpostavimo suprotno, da jedinstveno rešenje $\mathbf{x}^*$ dobijeno relaksiranim leksikografskim metodom nije Pareto optimalno. To znači da postoji neko $\mathbf{x} \in S$ tako da za svako $i = 1, \ldots, l$ važi $Q_i(\mathbf{x}) \geq Q_i(\mathbf{x}^*)$ pri čemu za neko $j \in \{1, \ldots, l\}$ važi stroga nejednakost. Razlikujemo dva slučaja:

1) $Q_k(\mathbf{x}) > Q_k(\mathbf{x}^*)$. Kako iz pretpostavljenog sledi da je $Q_i(\mathbf{x}) \geq Q_i^* - \delta_i$, $i = 1, \ldots, k-1$, zaključujemo da $\mathbf{x}^*$ nije rešenje dobijeno u koraku k gore opisanog algoritma tj. da $\mathbf{x}^*$ nije rešenje dobijeno relaksiranim leksikografskim metodom, što je kontradikcija.

2) $Q_k(\mathbf{x}) = Q_k(\mathbf{x}^*)$. Kao i pod 1), važi $Q_i(\mathbf{x}) \geq Q_i^* - \delta_i,\ i = 1, \ldots, k-1$, pa dobijamo da je $\mathbf{x}$ rešenje dobijemo relaksiranim leksikografskim metodom. Međutim, kako je $\mathbf{x}^*$ jedinstveno rešenje zaključujemo da mora biti $\mathbf{x} = \mathbf{x}^*$. Tada je i $Q_j(\mathbf{x}) = Q_j(\mathbf{x}^*)$, što je kontradikcija.

Dakle, $\mathbf{x}^*$ je zaista Pareto optimalno rešenje. □

U praktičnoj primeni leksikografskog metoda vrednosti δ_i moraju biti pažljivo odabrane. Ukoliko se ovim metodom dobije više rešenja bilo bi dobro smanjiti te vrednosti.

Primer 3.6.1 Primenom relaksiranog leksikografskog metoda rešiti sledeći zadatak VKO (funkcije cilja su relaksirane po prioritetu)

$$\begin{array}{ll} \max & \{Q_1(\mathbf{x}), Q_2(\mathbf{x}), Q_3(\mathbf{x})\} \\ \text{p.o.} & 2x_1 + x_2 \leq 6,\ x_1 + 3x_2 \leq 6,\ x_1 \geq 0,\ x_2 \geq 0. \end{array}$$

gde je

$$\begin{array}{lcll} Q_1(\mathbf{x}) & = & x_1 + 4x_2, & \delta_1 = 1 \\ Q_2(\mathbf{x}) & = & x_1, & \delta_2 = 1 \\ Q_3(\mathbf{x}) & = & x_1 + x_2. & \end{array}$$

Rešenje.
Korak 1. Rešiti problem

$$\begin{array}{ll} \max & Q_1(x) = x_1 + 4x_2 \\ \text{p.o.} & 2x_1 + x_2 \leq 6,\ x_1 + 3x_2 \leq 6,\ x_1, x_2 \geq 0. \end{array}$$

Ovaj zadatak ima jedinstveno optimalno rešenje u tački $x^{1^*} = (0, 2)$, pri čemu je $Q_1^* = 8$.

Korak 2. Rešiti problem

$$\begin{array}{ll} \max & Q_2(\mathbf{x}) = x_1 \\ \text{p.o.} & 2x_1 + x_2 \leq 6,\ x_1 + 3x_2 \leq 6,\ x_1, x_2 \geq 0,\ x_1 + 4x_2 \geq 7. \end{array}$$

Dobija se rešenje $x^{2^*} = (2.43, 1.14), Q_2^* = 2.43$
Korak 3. Rešiti problem

$$\begin{array}{ll} \max & Q_3(\mathbf{x}) = x_1 + 4x_2 \\ \text{p.o.} & 2x_1 + x_2 \leq 6,\ x_1 + 3x_2 \leq 6,\ x_1, x_2 \geq 0,\ x_1 + 4x_2 \geq 7, x_1 \geq 1.43. \end{array}$$

Rešenje ovog zadatka se usvaja kao konačno. To je rešenje $x_1^* = 3.6,\ x_2^* = 1.2$. Vrednosti funkcija cilja u ovoj tački su $Q^* = (7.2, 2.4, 3.6)$.

Relaksirani leksikografski metod je veoma osetljiv na izbor koeficijenata δ_j, tako da se "pogrešnim" izborom njegove vrednosti mogu dobiti neprihvatljiva rešenja. Tako u prethodnom primeru imamo slučaj da se konačno resenje poklapa sa marginalnim rešenjem funkcije cilja koja ima najniži prioritet, dok je vrednost najznačajnijeg kriterijuma smanjena. Kod primene ove metode se preporučuje da DO kritički preispita dobijena rešenja, uporedi ih sa marginalnim i da po potrebi koriguje zadate koeficijente.

Primer 3.6.2 Vlasnik hotela pre početka sezone odlučuje o opremanju soba nameštajem. Hotel raspolaže sa ukupno 58 soba, od kojih su 16 male, tako da mogu da budu samo jednokrevetne, dok ostale mogu biti jednokrevetne, dvokrevetne ili trokrevetne. Podaci o ceni opremanja soba i sezonskoj zaradi po sobi su dati u sledećoj tabeli

	Jednokrevetna	Dvokrevetna	Trokrevetna
Cena opremanja	20000	40000	60000
Sezonska zarada	15000	25000	30000

Tabela 4.6.1.

Za opremanje soba, vlasnik može da izdvoji 2.5 miliona dinara. On želi da postigne dva cilja

1. da ostvari što veću sezonsku zaradu hotela i
2. da omogući primanje što većeg broja gostiju kako ne bi došlo do toga da gostima bude uskraćeno gostoprimstvo.

a) Formulisati matematički model za određivanje optimalnog opremanja soba potrebnim nameštajem (broj kreveta) ako se žele ostvariti oba postavljena kriterijuma.

b) Rešiti zadatak ako je prvi kriterijum prioritetan i ako je vlasnik spreman da "žrtvuje" 50.000 din. svoje sezonske zarade da bi poboljšao drugi kriterijum.

Rešenje.

a) Promenljive x_1, x_2 i x_3 će predstavljati broj jednokrevetnih, dvokrevetnih i trokrevetnih soba u hotelu, respektivno. Tada matematički model ima sledeći oblik

$$\begin{array}{ll} \max & [Q_1(\mathbf{x}), Q_2(\mathbf{x})] \\ \text{p.o.} & x_1 + x_2 + x_3 \leq 58, \\ & 20.000x_1 + 40.000x_2 + 60.000x_3 \leq 2.500.000, \\ & x_1 \geq 16,\ x_2 \geq 0,\ x_3 \geq 0. \end{array}$$

gde su

$$\begin{aligned} Q_1(\mathbf{x}) &= 15.000x_1 + 25.000x_2 + 30.000x_3 \\ Q_2(\mathbf{x}) &= x_1 + x_2 + 3x_3 \end{aligned}$$

b) Iako nije eksplicitno naglašeno, na osnovu zahteva zadatka je očigledno da se traži rešenje dobijeno relaksiranim leksikografskim metodom, s tim da je najvažniji prvi kriterijum, a zadati koeficijent $\alpha_1 = 50.000$. U prvom koraku rešavamo model

$$\begin{array}{ll} \max & Q_1(\mathbf{x}) = 15.000x_1 + 25.000x_2 + 30.000x_3 \\ \text{p.o.} & \mathbf{x} \in \mathbf{X}, \end{array}$$

čije je rešenje $Q_1^* = 1.415.000$ din, $x_1^{1^*} = 16$, $x_2^{1^*} = 17$, $x_3^{1^*} = 25$. U drugoj iteraciji rešavamo model u koji smo dodali ograničenje koji obezbeđujemo da se prvi kriterijum ne "pokvari" za više od 50.000 ($1.415.000 - 50.000 = 1.365.000$)

$$\begin{array}{ll} \max & Q_2(\mathbf{x}) = x_1 + 2x_2 + 3x_3 \\ \text{p.o.} & \mathbf{x} \in \mathbf{X},\ 15.000x_1 + 25.000x_2 + 30.000x_3 \geq 1.365.000. \end{array}$$

Rešenje ovog zadatka je konačno rešenje problema. Ono, međutim, nije jedinstveno. Štaviše, simpleks metodom se mogu generisati četiri rešenja:

$$x^{*I} = (23, 0, 34),\ x^{*II} = (16, 7, 31.67),\ x^{*III} = (24.5, 0, 33.5),\ x^{*IV} = (16, 17, 25).$$

Sva optimalna rešenja se mogu dobiti konveksnom kombinacijom ova četiri. Grafički posmatrano, u trodimenzionalnom prostoru se dobija da su optimalna rešenja deo ravni oivičen trapezoidom čija su temena date četiri tačke. U svim tačkama koje se nalaze unutar tog trapezoida, vrednost druge funkcije cilja iznosi 125, dok vrednost prve varira između 1.365.000 i 1.415.000. Dakle, dobili smo skup tačaka koje predstavljaju slaba Pareto rešenja. Jedino tačka x^{*IV}, koja je optimalna i po prvom kriterijumu, je Pareto optimalno rešenje.

Zaključak. Da bi vlasnik hotela na što bolji način zadovoljio oba kriterijuma, najbolje je da opremi 16 jednokrevetnih, 17 dvokrevetnih i 25 trokrevetnih soba i tako ostvari zaradu od 1.415.000 din. po sezoni. Pri tome će hotel moći da prima 125 gostiju.

Jedan od načina za implementaciju relaksiranog leksikografskog metoda dat je funkcijom **MultiRelax**[] [25]:

Ulazne veličine:

q - lista ciljnih funkcija;

parcf - parametri ciljnih funkcija;

constr - lista ograničenja.

```
MultiRelax[q_List, constr_List, var_List, delta_List] :=
   Module[{res={}, s, f=constr, l=Length[q], i},
      For[i=1, i<=l, i++,
         s = Maximize[q[[i]], f, var];      (* 2V *)
         If[i<l, AppendTo[f, q[[i]]>=First[s]-delta[[i]] ]  ]; (* 3V,2V *)
         AppendTo[res, s];
      ];
      Print[res]; Return[{q/.Last[s],Last[s]}];
   ]
```

Primer 3.6.3 Problem vlasnika hotela rešava se izrazom

```
MultiRelax[{15000x1+25000x2+30000x3,x1+2x2+3x3},
  {x1+x2+x3<=58, 20000x1+40000x2+60000x3<=2500000,x1>=16,x2>=0,x3>=0},
  {x1,x2,x3},{50000}]
```

Dobija se sledeće rešenje:

```
{{1365000,125},{x1->23,x2->0,x3->34}}
```

3.7 Metod e-ograničenja

U ovom metodu DO izdvaja kriterijum $Q_p(\mathbf{x})$, $1 \leq p \leq l$ koji ima najviši prioritet i njega maksimizira, dok ostale funkcije cilja ne smeju imati vrednosti manje od unapred zadatih ε_k, $k = 1, \ldots, l$, $k \neq p$. Sve druge ciljne funkcije se koriste da se kreiraju dodatna ograničenja oblika $l_i \leq Q_i(\mathbf{x}) \leq \varepsilon_i$, $i = 1, \ldots, l$, $i \neq p$ [12]. U ovoj relaciji, l_i i ε_i jesu donja i gornja granica ciljne funkcije $Q_i(\mathbf{x})$, respektivno.

Haimes u [9] je uveo *e-constraint* (ili trade-off) metod (metod e-ograničenja), u kome su sve vrednostoi l_i isključene. Sistematsko variranje vrednosti ε_i produkuje skup Pareto optimalnih rešenja [12]. Međutim, nepodesna selekcija vrednosti $\varepsilon \in \mathbf{R}^k$ može da proizvede rešenje koje ne ispunjava ograničenje. Opšte pravilo za selektovanje ε_i je sledeće ([2])

$$Q_p(\mathbf{x_i}^*) \leq \varepsilon_i \leq Q_i(\mathbf{x_i}^*),$$

gde je $\mathbf{x_i}^*$ tačka koja maksimizira ciljnu funkciju $Q_i(\mathbf{x})$.

Neophodno je da se reši sledeći problem

$$\begin{array}{ll} \max & Q_p(\mathbf{x}) \\ \text{p.o.} & x \in \mathbf{X}, \\ & Q_i(\mathbf{x}) \geq \varepsilon_i,\ i = 1, \ldots, l,\ i \neq p, \end{array}$$

gde je $Q_p(\mathbf{x})$ odabrani kriterijum sa najvećim prioritetom i ϵ_i su odabrani realni brojevi. Ako optimalno rešenje ne postoji, neophodno je da se smanji vrednost za ε_i.

Teorema 3.7.1 *Rešenje problema ε ograničenja je slabo Pareto optimalno rešenje početnog problema VKO.*

Dokaz. Pretpostavimo da je $\mathbf{x}^*$ rešenje problema ε ograničenja, i da ono nije slabi Pareto optimum. Dakle, postoji $\mathbf{x} \in S$ tako da je $Q_j(\mathbf{x}) > Q_j(x^*) \wedge Q_i(\mathbf{x}) > Q_i(\mathbf{x}^*) \geq \varepsilon_i$ za svako $i = 1, \ldots, l$, $i \neq j$, odakle sledi da $\mathbf{x}^*$ nije rešenje problema ε ograničenja, što je kontradikcija. Zaključujemo da je rešenje problema ε ograničenja slabi Pareto optimum. □

Teorema 3.7.2 *Tačka $\mathbf{x}^* \in S$ je Pareto optimalno rešenje polaznog problema VKO akko za svako $j = 1, \ldots, l$ tačka $\mathbf{x}^*$ jeste rešenje problema ε ograničenja pri čemu je $\varepsilon_i = Q_i(\mathbf{x}^*)$, $i = 1, \ldots, l$, $i \neq j$.*

Dokaz. Uslov je dovoljan jer iz njega sledi da ni za jedno $j = 1, \ldots, l$ ne postoji $\mathbf{x} \in S$ tako da je $Q_j(\mathbf{x}) > Q_j(\mathbf{x}^*) \wedge Q_i(\mathbf{x}) \geq \varepsilon_i = Q_i(\mathbf{x}^*)$, $i \neq j$, tj. sledi da je $\mathbf{x}^*$ Pareto optimum.

Pretpostavimo, sa druge strane, da je $\mathbf{x}^*$ Pareto optimum, a da za neko j ono nije rešenje problema ε ograničenja sa $\varepsilon_i = Q_i(\mathbf{x}^*)$, $i \neq j$. Dakle, postoji $\mathbf{x} \in S$ tako da važi $Q_j(\mathbf{x}) > Q_j(\mathbf{x}^*)$ i $Q_i(\mathbf{x}) \geq Q_i(\mathbf{x}^*) = \varepsilon_i$, $i \neq j$. Dolazimo u kontradikciju sa pretpostavkom da je $\mathbf{x}^*$ Pareto optimalno rešenje. Dakle, uslov je i potreban. □

Primetimo da iz ove teoreme sledi da se metodom ε ograničenja može dobiti proizvoljno Pareto optimalno rešenje.

Teorema 3.7.3 *Neka je vektor ε proizvoljan. Tada, ako problem ε ograničenja ima jedinstveno rešenje onda je ono Pareto optimalno rešenje polaznog problema VKO.*

Dokaz. Neka je $\mathbf{x}^*$ jedinstveno rešenje problema ε ograničenja. Pretpostavimo da $\mathbf{x}^*$ nije Pareto optimalno tj. da postoji $\mathbf{x}'$ iz S tako da važi $Q_i(\mathbf{x}') \geq Q_i(\mathbf{x}^*)$ $i = 1, \ldots, l$, pri čemu važi stroga jednakost za $i = l$. Ako je $l = j$, tada dobijamo $Q_j(\mathbf{x}') > Q_j(\mathbf{x}^*) \wedge Q_i(\mathbf{x}') \geq Q_i(\mathbf{x}^*) \geq \varepsilon_i$, $i = 1, \ldots, l$, $i \neq j$, tj. dobijamo kontradikciju sa pretpostavkom da je $\mathbf{x}^*$ rešenje problema ε ograničenja. Ako je $l \neq j$, tada je $Q_i(\mathbf{x}') \geq Q_i(\mathbf{x}^*) \geq \varepsilon_i$, $i = 1, \ldots, l$, $i \neq j$ i $Q_j(x') \leq Q_j(x^*)$. Međutim kako je x^* jedinstveno rešenje te iz $Q_i(\mathbf{x}') \geq \varepsilon_i$, $i = 1, \ldots, l$, $i \neq j$ sledi da mora biti $Q_j(\mathbf{x}') < Q_j(\mathbf{x}^*)$. Ponovo dolazimo do kontradikcije pa zaključujemo da je $\mathbf{x}^*$ zaista Pareto optimalno rešenje polaznog problema VKO. □

U slučaju da problem ε ograničenja nema rešenja trebalo bi povećati vrednosti granica ε_i. Naravno, nijedno ε_i ne sme biti manje od idealne vrednosti funkcije cilja Q_i. U slučaju da DO sam izabere prioritetnu funkciju Q_j i konstante ε_i na ovaj metod se može gledati kao na a priori metod. Međutim, on u tom slučaju ne može biti siguran da će biti zadovoljan dobijenim rešenjem. Zbog toga je pristup obično drugačiji. Postupak rešavanja problema ε ograničenja se ponavlja za razne vrednosti granica ε_i i za razne funcije Q_j kao prioritetne, sve dok DO ne bude zadovoljan dobijenim rešenjem.

Da bismo dokazali da je rešenje dobijeno metodom ograničenja Pareto optimalno moramo dokazati da je ono jedinstveno ili da je rešenje l različitih problema za svaku od funkcija Q_j, što nije nimalo jednostavno. Ovo je glavna mana ove metode.

Sledi opis implementacije prema [25].

Ulazni parametri:

$q\_$, $constr\_$, $var\_$: unutrašnja forma problema;
$p\_$: redni broj ciljne funkcije sa najvećim prioritetom.

```
MultiEps[q_List, constr_List, var_List, p_Integer] :=
 Module[{s, f = constr, i, j, k, l = Length[q], eps, lb = {}, ub = {}},
   For[i = 1, i <= l, i++, (* Find bounds in (3.4)  *)
     xi = Maximize[q[[i]], constr, var];
     AppendTo[lb,q[[p]]/.First[Rest[xi]]];
     AppendTo[ub,q[[i]]/.First[Rest[xi]]];
   ];
   For[i = 1, i <= l, i++,
      For[eps = lb[[i]], eps <= ub[[i]], eps = eps + 0.1,
         f = constr;
         For[k = 1, k <= l, k++,
            If[k != p, AppendTo[f, q[[k]]>=eps ]  ]  (* 3V *)
         ];
         s = Maximize[q[[p]], f, var];   (* 2V, 1V *)
      ]
   ]
 ]
```

Primer 3.7.1 Rešiti problem:

$$\begin{array}{ll} \max: & \{Q_1(\mathbf{x}) = x_1,\ Q_2(\mathbf{x}) = x_2\} \\ \text{p.o.:} & 0 \le x_1 \le 1,\ 0 \le x_2 \le 1. \end{array}$$

Pretpostavimo da prvi kriterijum ima najveći prioritet. Ovaj problem se rešava sledećim pozivom funkcije *MultiEps*:

In[5]:= `MultiEps[{x1,x2},{x1>=0,x2>=0,x1<=1,x2<=1},{x1,x2},{0.5},1]`

U slučaju kada drugi kriterijum uzima vrednosti veće ili jednake od 0.5 ($eps = 0.5$) neophodno je da se reši sledeći jednokriterijumski problem:

$$\begin{array}{ll} \max & Q_1(\mathbf{x}) = x_1 \\ \text{p.o.} & 0 \le x_1 \le 1,\ 0 \le x_2 \le 1,\ x_2 \ge 0.5. \end{array}$$

Njegova unutrašnja forma, korišćena u MATHEMATICA funkciji `Maximize` je

$x1,\ \{x1>=0, x2>=0, x1<=1, x2<=1, x2>=0.5\},\ \{x1, x2\}$

Rešenje problema je

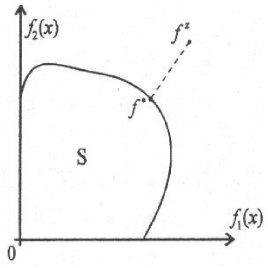
`{{1.,0.5},{x1->1., x2->0.5}}`

i nije Pareto optimalno. Pareto optimalno rešenje se generiše u slučaju $eps = 1$, i jednako je

`{{1.,1},{x1->1, x2->1}}`

3.8 Metodi rastojanja

Metodi rastojanja čine grupu za rešavanje zadataka VKO čija je osnovna ideja da se u kriterijumskom prostoru traži tačka koja je najbliža nekoj unapred određenoj tački koja se želi dostići ili ka kojoj treba težiti ako ona nije dopustiva. Drugim rečima, minimizira se rastojanje između željene tačke i dopustive oblasti. Razlike između metoda ove grupe potiču od toga kako se željena tačka određuje, na koji način se rastojanje od nje "meri", da li se uvode težinski koeficijenti, itd.



Slika 3.8.1. Tačka najbliža željenoj vrednosti f^z.

U opsťem slučaju, DO za svaki od l kriterijuma zadaje željene vrednosti ili određuje način kako će se one izračunati. Na taj način se u l -dimenzionalnom kriterijumskom prostoru definiše tačka

$$f^z = (f_1^z, ..., f_l^z)$$

koja po pravilu ne pripada dopustivoj oblasti S. U slučaju

$$f^z \in S$$

zadatak se rešava rešavanjem sistema jednačina koje su definisane skupom ograničenja.

U metodu rastojanja rešava se sledeći opšti zadatak

$$\begin{array}{ll} \min & d(f^z - f(x)) \\ \text{p.o.} & \mathbf{x} \in \mathbf{x} \end{array}$$

gde $d(*.*)$ označava rastojanje definisano pogodnom metrikom.

U kontekstu merenja rastojanja u kriterijumskom prostoru ovde ćemo ponoviti definicije metrika koje se najčešće koriste u metodima rastojanja

$$l_1 \ (pravougaona) \ metrika : d_{l_1}(f^z, f(\mathbf{x})) = \sum_{k=1}^{l} |f_k^z - f_k(\mathbf{x})| \,,$$

$$l_2 \ (Euklidova) \ metrika : d_{l_2}(f^z, f(\mathbf{x})) = \sqrt{\sum_{k=1}^{l} (f_k^z - f_k(\mathbf{x}))^2},$$

$$l_\infty \ (Čebiševljeva) \ metrika : d_{l_\infty}(f^z, f(\mathbf{x})) = \max_{1 \le k \le l} \{|f_k^z - f_k(\mathbf{x})|\} \,.$$

Kriterijumima je moguće dodeliti težinske koeficijente, tako da prethodne formule za rastojanje između željenog i traženog rešenja dobijaju oblik

$$d_{l_1}(f^z, f(x)) = \sum_{k=1}^{l} w_k \, |f_k^z - f_k(\mathbf{x})|$$

za pravougaonu metriku,

$$d_{l_2}(f^z, f(x)) = \sqrt{\sum_{k=1}^{l} w_k (f_k^z - f_k(x))^2}$$

za Euklidovu metriku,

$$d_{l_\infty}(f^z, f(\mathbf{x})) = \max_{1 \le k \le l} \{w_k \, |f_k^z - f_k(\mathbf{x})|\} .$$

za Čebiševljevu metriku.

Bez obzira na oblik polaznih funcija cilja, zadaci minimizacije rastojanja su problemi nelinearnog programiranja, za koje, u opštem slučaju, nije jednostavno pronaći optimalno rešenje. Izuzetno se u slučaju l_1 metrike i linearnih funkcija cilja i ograničenja zadatak minimizacije rastojanja može formulisati kao problem LP što ćemo objasniti na sledećem primeru.

Napomena 3.8.1 *Objašnjene transformacije su osnova za formulaciju zadatka višekriterijumskog linearnog programiranja (VLP) i oblik koji se naziva ciljno programiranje. Prema tome, zadatak ciljnog programiranja je poseban oblik zadatka metoda rastojanja u VKO. Dodatno je moguće kriterijumima dodeliti prioritete dostizanja željenih vrednosti i težinske koeficijente.*

Rešenje zadatka VKO dobijeno metodom rastojanja, kada skupu vrednosti kriterijuma ne pripada željna vrednost, predstavlja slabi Pareto optimum, a ako je jedinstveno, onda je i Pareto optimalno.

Metod rastojanja (ciljno programiranje ili goal programming method) je razvijen u [4], [5] [13]. Za svaku od l kriterijumskih funkcija odabiraju se ciljne vrednosti, ili se izračunavaju pomoću funkcije *Maximize*. Na taj način, generiše se l-dimenzionalna tačka $b(\mathbf{x}) = (b_1, \ldots, b_l)$, koja sadrži ciljne vrednosti za ciljne funkcije. Potom se minimizira rastojanje između tačke $b(\mathbf{x})$ i $Q(\mathbf{x}) = (q_1(\mathbf{x}), \ldots, q_l(\mathbf{x}))$. Drugim rečima, rešava se sledeći opšti zadatak

$$\begin{array}{ll} \min & d(b - Q(\mathbf{x})) \\ \text{p.o.} & \mathbf{x} \in \mathbf{X}, \end{array}$$

gde je $d(*.*)$ metrika koja definiše rastojanje između tačaka b i $Q(\mathbf{x})$. Uzmimo, na primer metriku definisanu L_1-normom

$$d(b - Q(\mathbf{x})) = \sum_{i=1}^{l} |b_i - q_i(\mathbf{x})|.$$

U ovom slučaju, prirodno je da se koriste zamene $y_i = b_i - q_i(\mathbf{x})$, $i = 1, \ldots, l$. Koristeći dodatne smene $y_i = \$[i]^- - \$[i]^+$, $i = 1, \ldots, l$, u kojima je $\$[i]^-, \$[i]^+ \ge 0$,

(i tada je $|y_i| = \$[i]^+ + \$[i]^-,\ i = 1, \ldots, l$), dobija se sledeći jednokriterijumski optimizationi problem

$$\begin{array}{ll} \min & \displaystyle\sum_{i=1}^{l}(\$[i]^+ + \$[i]^-) \\ \text{p.o.} & \mathbf{x} \in \mathbf{X} \\ & Q_i(\mathbf{x}) - \$[i]^+ + \$[i]^- = b_i,\ i = 1, \ldots, l \\ & \$[1]^+ \geq 0, \ldots, \$[l]^+ \geq 0, \$[1]^- \geq 0, \ldots, \$[l]^- \geq 0. \end{array}$$

Lista ograničenja se transformiše ciklusom:

```
For[i=1,i<=l,i++,
   AppendTo[constr,q[[i]]-$[2i-1]+$[2i]==b[[i]]];
];
For[i=1,i<=2l,i++, AppendTo[constr,$[i]>=0]  ];
```

Odgovarajuća jednokriterijumska ciljna funkcija se generiše i minimizira izrazom

```
s=Minimize[Array[$,2l,1,Plus],constr,Union[Variables[Array[$,2l,1]],var]];
```

Standardna funkcija `Variables[expr]` daje listu nezavsinih promenljivih u izrazu `expr`. Izraz `Union[l1,l2,...]` daje sortiranu listu različitih elemenata koji se pojavljuju u bilo kom od izraza l_i [32].

Primer 3.8.1 Dat je zadatak VLP

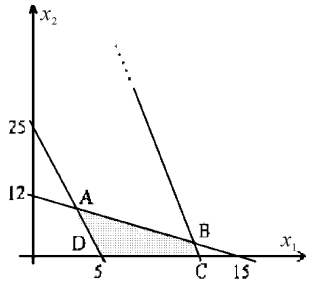
$$\begin{array}{ll} \max & [Q_1(\mathbf{x}), Q_2(\mathbf{x}), Q_3(\mathbf{x})] \\ \text{p.o.} & 35x_1 + 7x_2 \geq 175 \\ & 136x_1 + 170x_2 \leq 2400 \\ & 20x_1 + 3x_2 \leq 240 \\ & x_1, x_2 \geq 0 \end{array}$$

gde su

$$Q_1(\mathbf{x}) = 4x_1 + 5x_2,\ Q_2(\mathbf{x}) = x_1 + x_2,\ Q_3(\mathbf{x}) = 40x_1 + 6x_2.$$

a) Rešiti zadatak metodom rastojanja sa pravougaonom metrikom, ako se želi dostići idealna tačka.

b) Koja rešenja predstavljaju Pareto, a koja slabi Pareto optimum polaznog zadatka.



Slika 3.8.2. Skup dopustivih rešenja u primeru f^z.

Rešenje. a) U ovom zadatku DO je odredio da su željene vrednosti kriterijumskih funkcija idealne vrednosti tih funkcija. Zato je najpre potrebno odrediti marginalna rešenja i odgovarajuće idealne vrednosti funkcije

$f_1^* = 60$ za višestruko rešenje koje pripada duži **AB** sa krajnjim tačkama

$$x_1^{(1)^*} = 3.09,\ x_2^{(1)^*} = 9.52$$

i

$$x_1^{(1)^{**}} = 11.59,\ x_2^{(1)^{**}} = 2.73.$$

$f_2^* = 14.32$ za : $x_1^{(2)^*} = 11.59$, $x_2^{(2)^*} = 2.73$ (tačka **B**),
$f_3^* = 480$ za višestruko rešenje koje pripada duži **BC** sa krajnjim tačkama

$$x_1^{(3)^{**}} = 11.59,\ x_2^{(3)^{**}} = 2.73$$

i

$$x_1^{(3)^{**}} = 12,\ x_2^{(3)^{**}} = 0.$$

Zadatak VKO prilagođen za rešavanje metodom rastojanja sada ima oblik

$$\min\ \Phi(\mathbf{x}) = |f_1^* - f_1(\mathbf{x})| + |f_2^* - f_2(\mathbf{x})| + |f_3^* - f_3(\mathbf{x})|$$

pri istim ograničenjima.

Posto sve kriterijume trebamo maksimizirati, a po definiciji marginalnih rešenja je $f_k^* \geq f_k(x)$, može se staviti da je $|f_k^* - f_k(\mathbf{x})| = f_k^* - f_k(\mathbf{x})$, $k = 1, 2, 3$. Tada se umesto polaznog zadatka može rešavati zadatak minimizacije funkcije $\Phi(\mathbf{x}) = 554,32 - 45x_1 - 12x_2$, pri istim ograničenjima. Pokazaćemo kako se ovaj zadatak rešava opštim postupkom koji koristi definiciju prebačaja i podbačaja. U tom slučaju matematički model ima oblik

$$\begin{aligned}
\min\quad & F(x,y) = y_1^+ + y_1^- + y_2^+ y_2^- + y_3^+ + y_3^- \\
\text{p.o.}\quad & 4x_1 + 5x_2 - y_1^+ + y_1^- = 60 \\
& x_1 + x_2 - y_2^+ + y_2^- = 14,32 \\
& 40x_1 + 6x_2 - y_3^+ + y_3^- = 480 \\
& 35x_1 + 7x_2 \geq 175 \\
& 136x_1 + 170x_2 \leq 2400 \\
& 20x_1 + 3x_2 \leq 240 \\
& x_1, x_2, y_1^+, y_1^-, y_2^+, y_2^-, y_3^+, y_3^- \geq 0.
\end{aligned}$$

Rešavanjem ovog zadatka dobija se jedinstveno rešenje $F^* = 0$, $x_1^* = 11.59$ i $x_2^* = 2.73$.

To što je vrednost funkcije cilj jednako nuli ukazuje da je dobijeno rešenje savršeno jer nema ni prebačaja ni podbačaja vrednosti funkcija cilja od idealne tačke. Ovo je očigledno i na grafiku. Vidimo da u tački **B** sve funkcije cilja dostižu svoju najveću vrednost.

b) Jedino je tačka **B**(11.59, 2.73) Pareto optimalna. Sve tačke na duži **AB** na jednu stranu i **BC** na drugu stranu su slabi Pareto optimumi.

Sledi funkcija **MultiDist** kojom se daje implementacija metode rastojanja. Kao pomoćna funkcija koristi se funkcija **ideal** [25].

```
ideal[q_,constr_,var_]:=
  Module[{res={},i,l=Length[q]},
     For[i=1,i<=l,i++,
       AppendTo[res,First[Maximize[q[[i]],constr,var]]];
     ];
     Return[res];
  ]
```

```
MultiDist[q_,constr_,w_List,var_]:=
 Module[{con=constr,point={},l=Length[q],s},
   If[w=={}, point=ideal[q,con,var], point=w];
   For[i=1,i<=l,i++,
      AppendTo[con,q[[i]]-$[2i-1]+$[2i]==point[[i]]]; (* 3V,2V *)
   ];
   For[i=1,i<=2l,i++, AppendTo[con,$[i]>=0]  ];
   s=Minimize[Array[$,2l,1,Plus],con,Union[Variables[Array[$,2l,1]],var]];
   (* 3V,2V *)
   Return[{q/.Last[s],Last[s]}]
  ];
```

Primer 3.8.2 Rešiti problem

$$\begin{aligned} \max \quad & \{Q_1(\mathbf{x}) = x_1,\ Q_2(\mathbf{x}) = x_2\} \\ \text{p.o.} \quad & 0 \le x_1 \le 1,\ 0 \le x_2 \le 1. \end{aligned}$$

Ovaj problem se rešava izrazom

In[5]:= `MultiDist[{x1,x2},{x1>=0,x2>=0,x1<=1,x2<=1},{},{x1,x2}]`

Pozivom funkcije *ideal*, dobija se idealna tačka $\{2, 3\}$. Posle toga, dobija se sledeći

$$\begin{aligned} \min \quad & \Phi(\mathbf{x}) = |2 - x_1| + |3 - x_2| \\ \text{p.o.} \quad & x_1 \le 1,\ x_2 \le 1\ x_1 \ge 0,\ x_2 \ge 0. \end{aligned}$$

Posle zamena $y_1 = 2 - x1 = \$[1]^- - \$[1]^+$, $y_2 = 3 - x2 = \$[2]^- - \$[2]^+$, koristeći $|y_1| = \$[1]^+ + \$[1]^-$, $|y_2| = \$[2]^+ + \$[2]^-$, s obzirom na (2.7), polazni problem se transformiše u sledeći

$$\begin{aligned} \min \quad & \$[1]^+ + \$[1]^- + \$[2]^+ + \$[2]^- \\ \text{p.o.} \quad & x_1 - \$[1]^+ + \$[1]^- = 2,\ x_2 - \$[2]^+ + \$[2]^- = 3, \\ & x_1 \le 1,\ x_2 \le 1, \\ & x_1 \ge 0,\ x_2 \ge 0,\ \$[1]^+ \ge 0,\ \$[1]^- \ge 0,\ \$[2]^+ \ge 0,\ \$[2]^- \ge 0. \end{aligned}$$

Unutrašnja reprezentacija ovog problema je

$\$[1] + \$[2] + \$[3] + \$[4]$,
$\{x1 >= 0,\ x2 >= 0,\ x1 <= 1,\ x2 <= 1,$
$x1 - \$[1] + \$[2] == 2,\ x2 - \$[3] + \$[4] == 3,$
$\$[1] >= 0, \$[2] >= 0, \$[3] >= 0, \$[4] >= 0\}$,
$\{x1, x2, \$[1], \$[2], \$[3], \$[4]\}$

dok je njegovo optimalno rešenje

$\{3,\ \{x1 -> 1, x2 -> 1, \$[1] -> 0, \$[2] -> 1, \$[3] -> 0, \$[4] -> 2\}\}$

Prema tome, optimalno rešenje polaznog problema je

Out[5]= `{{1,1},{x1->1, x2->1, $[1]->0,$[2]->1,$[3]->0,$[4]->2}}`

3.9 Metod PROMETHEE

Metod **PROMETHEE** (Preference Ranking Organization METHods for Enrichment Evaluation) je metod višekriterijumske analize i služi za rangiranje konačnog broja alternativa. Postoje četiri varijante ovog metoda: PROMETHEE I, II, III i IV (poslednja predstavlja proširenje za neprekidne skupove). Opisaćemo metod II, koja daje potpuni poredak alternativa [21].

Cilj nam je da, od N tačaka $x^{(1)}$,..., $x^{(N)}$ u skupu dopustivih rešenja S izaberemo onu koja daje najbolju vrednost rešenja.

Uvedimo funkciju preferencije $P_i(x^{(1)}, x^{(2)})$ za alternative $x^{(1)}$ i $x^{(2)}$ u odnosu na kriterijum Q_i

$$P_i(x^{(1)}, x^{(2)}) = \begin{cases} 0, & \text{ako je } Q_i(x^{(1)}) \geq Q_i(x^{(2)}) \\ P_i(f_i(x^{(2)}) - f_i(x^{(1)})), & \text{ako je } Q_i(x^{(1)}) < Q_i(x^{(2)}) \end{cases}$$

Uvodimo oznaku $d = Q_i(x^{(2)}) - Q_i(x^{(1)})$.
Predloženo je šest tipova funkcije preferencije:

1. Jednostavan kriterijum

$$P_i(x^{(1)}, x^{(2)}) = \begin{cases} 0, & \text{ako je } d \leq 0 \\ 1, & \text{ako je } d > 0 \end{cases}$$

2. Kvazi-kriterijum

$$P_i(x^{(1)}, x^{(2)}) = \begin{cases} 0, & \text{ako je } d \leq q \\ 1, & \text{ako je } d > q \end{cases}$$

3. Kriterijum sa linearnom preferencijom

$$P_i(x^{(1)}, x^{(2)}) = \begin{cases} 0, & \text{ako je } d \leq 0 \\ \frac{d}{p,} & \text{ako je } 0 < d \leq p \\ 1, & \text{ako je } d > p \end{cases}$$

4. Nivoski kriterijum

$$P_i(x^{(1)}, x^{(2)}) = \begin{cases} 0, & \text{ako je } d \leq q \\ \frac{1}{2,} & \text{ako je } q < d \leq p \\ 1, & \text{ako je } d > p \end{cases}$$

5. Kriterijum sa linearnom preferencijom i oblašću indiferentnosti

$$P_i(x^{(1)}, x^{(2)}) = \begin{cases} 0, & \text{ako je } d \leq q \\ \frac{d-q}{p-q,} & \text{ako je } q < d \leq p \\ 1, & \text{ako je } d > p \end{cases}$$

6. Gausov kriterijum

$$P_i(x^{(1)}, x^{(2)}) = \begin{cases} 0, & \text{ako je } d \leq 0 \\ 1 - e^{\frac{-d^2}{2\sigma^2}}, & \text{ako je } d > 0 \end{cases}$$

Pri tome paremetre p, q, σ treba zadati za svaku kriterijumsku funkciju.

Definišemo višekriterijumski indeks preferencije alternative $x^{(1)}$ nad $x^{(2)}$

$$\Pi(x^{(1)}, x^{(2)}) = \sum_{i=1}^{k} w_i P_i(x^{(1)}, x^{(2)})$$

gde je w_i težina i-tog kriterijuma. Podrazumeva se $\sum_{i=1}^{k} w_i = 1$.

Uvodimo zatim tokove preferencije:

$$\begin{aligned}
\phi_i^+(x^{(i)}) &= \sum_{m=1}^{N} \Pi(x^{(i)}, x^{(m)}) \quad \text{(pozitivni tok)} \\
\phi_i^-(x^{(i)}) &= \sum_{m=1}^{N} \Pi(x^{(m)}, x^{(i)}) \quad \text{(negativni tok)} \\
\phi_i(x^{(i)}) &= \phi_i^+(x^{(i)}) - \phi_i^-(x^{(i)}) \quad \text{(neto tok).}
\end{aligned}$$

Neto tok je funkcija pomoću koje rangiramo alternative. Alternativa $x^{(i)}$ je višekriterijumski bolja od $x^{(j)}$ ako je $\phi_i(x^{(i)}) > \phi_j(x^{(j)})$.

Time smo dobili potpuno uređenje skupa alternativa na osnovu koga možemo izabrati najbolju.

3.10 Metod ELECTRE

Metod ELECTRE I (ELimination and ET Choice Translating REality) prvi put je objavio Roy sa svojim saradnicima (1971.). Za određivanje delimičnih poredaka alternativa najčešće se koristi metod ELECTRE I, a za potpuno uređenje skupa alternativa metod ELECTRE II. Ovi metodi omogućavaju parcijalno uređenje skupa rešenja na osnovu preferencija donosioca odluke, a pogodne su za diskretne probleme i raznorodne kriterijumske funkcije. Modeli dozvoljavaju uključivanje subjektivnih procena, bilo kroz vrednosti kriterijumskih funkcija, bilo kroz relativne važnosti pojedinih kriterijuma. Metodi ELECTRE III i IV su metodi "višeg" ranga.

Metod **ELECTRE** je metod višekriterijumske analize pomoću koga se može dobiti delimično uređenje skupa alternativa. Na osnovu dobijene relacije delimičnog uređenja ρ konstruiše se graf G u kome čvorovi predstavljaju alternative. Grana grafa se usmerava od čvora $x^{(i)}$ prema čvoru $x^{(j)}$ ako je $x^{(i)} \rho x^{(j)}$. Na osnovu tako konstruisanog grafa dobija se parcijalna rang lista (nepotpuno uređen skup) alternativa, a može se desiti da neke alternative ostanu izolovane.

Dakle, metod ELECTRE je razvijen za analizu odnosa među alternativama, a ne za potpuno uređenje skupa alternativa.

Opisaćemo ukratko noviju varijantu ELECTRE II [21].

Označimo sa $I = \{1, \ldots, l\}$ skup indeksa kriterijumskih funkcija. Za svaki par rešenja $(x^{(1)}, x^{(2)})$ definišemo

$$I^+(x^{(1)}, x^{(2)}) := \{i \in I \mid Q_i(x^{(1)}) < Q_i(x^{(2)})\}$$

$$I^{=}(x^{(1)}, x^{(2)}) := \{i \in I \mid Q_i(x^{(1)}) = Q_i(x^{(2)})\}$$
$$I^{-}(x^{(1)}, x^{(2)}) := \{i \in I \mid Q_i(x^{(1)}) > Q_i(x^{(2)})\}$$

Donosilac odluke zadaje težine kriterijuma w_i, $i = 1 \ldots, k$. Zatim se određuje

$$\begin{aligned} W^{+}(x^{(1)}, x^{(2)}) &= \sum_{i \in I^{+}} w_i \\ W^{=}(x^{(1)}, x^{(2)}) &= \sum_{i \in I^{=}} w_i \\ W^{-}(x^{(1)}, x^{(2)}) &= \sum_{i \in I^{-}} w_i \\ W &= W^{+} + W^{=} + W^{-}. \end{aligned}$$

Definišemo indeks saglasnosti

$$C(x^{(1)}, x^{(2)}) = \frac{W^{+} + W^{=}}{W}$$

Uslov saglasnosti za par $(x^{(1)}, x^{(2)})$ je ispunjen ako važi

$$C(x^{(1)}, x^{(2)}) \geq q \;\text{ i }\; \frac{W^{+}}{W^{-}} > 1$$

gde je q parametar čijim variranjem ćemo dobijati različite rezultate.

Da bismo definisali indeks nesaglasnosti moramo uvesti intervalnu skalu S koja će omogućiti poređenje funkcija raznorodnih vrednosti. Tako se dobijaju funkcije s_i koje predstavljaju surogat kriterijumskih funkcija Q_i.

Indeks nesaglasnosti je

$$d(x^{(1)}, x^{(2)}) = \frac{1}{S} \max_{i \in I^{-}} |s_i(x^{(1)}) - s_i(x^{(2)})|$$

Uslov nesaglasnosti za par $(x^{(1)}, x^{(2)})$ je ispunjen ako važi

$$d(x^{(1)}, x^{(2)}) \leq r$$

gde variranjem parametra r dobijamo različite rezultate.

Alternativa $x^{(j)}$ je superiorna nad $x^{(k)}$ ako su ispunjeni odgovarajući uslovi saglasnosti i nesaglasnosti. U tom slučaju grana $(x^{(j)}, x^{(k)})$ ulazi u rezultantni graf.

Povećanjem vrednosti parametra q i smanjivanjem vrednosti parametra r smanjuje se broj grana rezultantnog grafa. U praksi je dobro ispitati vrednosti $0.5 \leq q \leq 1$ i $0 \leq r \leq 0.5$.

Iz rezultantnog grafa možemo identifikovati alternative nad kojima nema superiornih. Naravno, neke alternative mogu ostati izolovane. Treba imati na umu da se metodom ELECTRE pre mogu dobiti korisne informacije o alternativama nego što se može potpuno rešiti problem multikriterijumske analize.

4 Interaktivni metodi

U interaktivnim metodima DO aktivno učestvuje tokom rešavanja problema. Najpre DO daje preliminarne informacije o svojim preferencijama na osnovu kojih analitičar generiše neki skup rešenja ili neki skup novih korisnih informacija. Kada dobije ove podatke DO obezbeđuje nove informacije o svojim zahtevima i postupak se iterativno ponavlja sve dok DO ne bude konačno zadovoljan dobijenim rešenjem. Prednost ovakvog pristupa reăvanju VKO je taj što se generiše samo deo Pareto optimalnog skupa i što DO iskazuje i menja svoje odluke i preferencije tokom samog procesa rešavanja problema.

Postoji mnogo interaktivnih metoda a mi ćemo navesti ukratko jednu od njih.

4.1 Metod referentne tačke

U raznim metodima VKO se koristi takozvana *funkcija ostvarenja* (achievement function). Uočimo referentnu tačku $\overline{z} \in \mathbf{R}^k$ čije su koordinate željene vrednosti kriterijumskih finkcja. Tačka $\overline{z}$ može biti ili ne biti dostižna. Funkcija ostvarenja je funkcija $s_{\overline{z}}$ koja zavisi od $\overline{z}$,

$$s_{\overline{z}} : Z \to \mathbf{R}$$

Posmatrajmo problem

$$\text{Minimizirati } s_{\overline{z}}(Q(\mathbf{x})) \text{ pod uslovom } \mathbf{x} \in S \tag{4.1.1}$$

Teorema 4.1.1 *Ako je funkcija ostvarenja $s_{\overline{z}} : Z \to \mathbf{R}$ strogo rastuća, tada je rešenje problema* (4.1.1) *slabo Pareto optimalno rešenje početnog problema VKO. Ako je izvršna funkcija $s_{\overline{z}} : Z \to \mathbf{R}$ jako rastuća, tada je rešenje problema* (4.1.1) *Pareto optimalno rešenje početnog problema VKO.*

Dokaz. Pretpostavimo suprotno, da je $\mathbf{x}^* \in S$ rešenje Problema (4.1.1), a da ono nije slabo Pareto optimalno. Sledi da postoji neko $\mathbf{x} \in S$ za koje važi da je $Q_i(\mathbf{x}) < Q_i(\mathbf{x}^*)$ za svako $i = 1, \ldots, k$. Kako je $s_{\overline{z}}$ strogo rastuća funkcija, dobijamo $s_{\overline{z}}(Q(\mathbf{x})) < s_{\overline{z}}(Q(\mathbf{x}^*))$, odakle dobijamo kontradikciju sa pretpostavkom da je $\mathbf{x}^*$ rešenje problema (4.1.1).

Drugi deo teoreme se dokazuje slično. Pretpostavimo da je $\mathbf{x}^*$ rešenje problema (4.1.1), a da ono nije Pareto optimalno. Dakle, postoji $\mathbf{x} \in S$ za koje važi $Q_i(\mathbf{x}) \leq Q_i(\mathbf{x}^*)$ za svako $i = 1, \ldots, k$, pri čemu za neko $j \leq k$ važi $Q_j(\mathbf{x}) < Q_j(\mathbf{x}^*)$. Kako je $s_{\overline{z}}$ jako rastuća funkcija dobijamo $s_{\overline{z}}(f(\mathbf{x})) < s_{\overline{z}}(Q(\mathbf{x}^*))$, odakle sledi da $\mathbf{x}^*$ nije rešenje problema (4.1.1), što je kontradikcija. □

U metodu referentne tačke generisanje Pareto optimalnog skupa je bazirano na referentnoj tački, a ne na vrednosnoj funkciji ili težinskim koeficijentima. Najpre se donosiocu odluke, ako je moguće, dostave neke informacije o problemu (idealna tačka, maksimalne vrednosti kriterijumskih funkcija u odnosu na dopustivi skup i slično). Takođe, potrebno je odrediti odgovarajuću funkciju ostvarenja. Na osnovu prethodne teoreme se može zaključiti da je poželjno da izvršna funkcija bude jako rastuća. Metod se sastoji iz sledećih koraka [15]:

1. Donosiocu odluka se predstave informacije o problemu. Stavi se $h = 1$.

2. DO navodi referentnu tačku $\overline{z}^h$ (koordinate ove tačke su željeni nivoi kriterijumskih funkcija).

3. Minimizira se funkcija ostvarenja pod uslovom $\mathbf{x} \in S$ i dobije se Pareto optimalno rešenje $\mathbf{x}^h$ i odgovarajuće z^h. Rešenje z^h se predstavi DO.

4. Minimizira se k funkcija ostvarenja $s_{\overline{z}(i)}$ sa referentnim tačkama $\overline{z}(i) = \overline{z}^h + ||\overline{z}^h - z^h||e^i$, gde je e^i i-ti jedinični vektor, $i = 1, \ldots, k$. Na ovaj način se dobije k novih Pareto optimalnih rešenja. Dakle, ukupno imamo $k + 1$ dominantnih rešenja, tj. $k + 1$ odgovarajućih tačaka u kriterijumskom skupu Z.

5. Ako među ovih $k + 1$ tačaka DO izabere neku kao zadovoljavajuću, onda je odgovarajuće x^h konačno rešenje. U suprotnom, DO bira (među dobijenih $k + 1$ tačaka skupa Z) novu referentnu tačku $\overline{z}^{h+1}$. Postavljamo $h = h + 1$ i prelazimo na korak 3.

Prednost ovog metoda je što DO neposredno upravlja procesom rešavanja problema i što je u mogućnosti da menja svoje mišljenje tokom procesa rešavanja. Nedostatak je što to može dugo da traje i što DO ne može biti siguran da je u koraku 5 koraku dobro izabrao novu referentnu tačku.

Literatura

[1] H.P. Benson, *Existence of efficient solution for vector maximization problems*, Journal of Optimization Theory and Applications, **28** (1978) 569-580.

[2] D.G. Carmichael, *Computation of Pareto optima in structural design*, Int. J. Numer. Methods Eng. 15, (1980) 925-952.

[3] V. Chankong, Y. Haimes, *Multiobjective decision making: Theory and methodology Series, Volume 8*, North-Holland, New York, Amsterdam, Oxford, 1983.

[4] A. Charnes, W.W. Cooper, *Management Models and Industrial Applications of Linear Programming*, New York: John Wiley and Sons, 1961.

[5] A. Charnes, W.W. Cooper, *Goal programming and multiple objective optimization; part 1*, Eur. J. Oper. Res. 1, (1977) 39-54.

[6] C.A. Coello, *A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques*,

Internet lokacija http://www.lania.mx/∼ccoello/EMOO/informationfinal.ps.gz

[7] M. Ćupić, R.V.M. Tummala, M. Suknović M., *Odlučivanje: Formalni pristup*, FON, Beograd, 2001.

[8] I. Das, J.E Dennis, *A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems*, Struct. Optim. **14** (1997) 63-69.

[9] Y.Y. Haimes, L.S. Lasdon, D.A. Wismer, *On a bicriterion formulation of the problems of integrated system identification and system optimization*, IEEE Trans. Syst. Man Cybern. SMC-1, (1971) 296-297.

[10] B.F. Hobbs, *A comparison of weighting methods in power plant siting.*, Decis. Sci., **11** (1980) 725-737.

[11] C.L. Hwang, K. Yoon, *Multiple attribute decision making methods and applications: a state-of-the-art survey.*, In: Beckmann, M.; Kunzi, H.P. (eds.) Lecture Notes in Economics and Mathematical Systems, No. 186. Berlin: Springer-Verlag, 1981.

[12] C.L. Hwang, Md. Masud, A.S., in collaboration with Paidy, S.R. and Yoon, K. *Multiple objective decision making, methods and applications: a state-of-the-art survey* In: Beckmann, M.; Kunzi, H.P. (eds.) Lecture Notes in Economics and Mathematical Systems, No. 164. Berlin: Springer-Verlag, 1979.

[13] Y. Ijiri, *Management Goals and Accounting for Control*, Amsterdam: North-Holland, 1965.

[14] J.P. Ignizio, *Linear programming in single-multiple-objective systems*, Englewood Cliffs: Prentice Hall, 1982.

[15] K. Miettien *Nonlinear Multiobjective Optimization*, Kluver Academic Publishers, Boston, London, Dordrecht, 1999.

[16] K.R. Mac-Crimmon, *An overview of multiple objective decision making, in J.L. Cochrane and M. Zeleny (Eds), Multiple criteria decision making*, University of South Carolina Press, Columbia, 1973.

[17] R. Maeder, *Programming in Mathematica, Third Edition* Redwood City, California: Adisson-Wesley, 1996.

[18] R.T. Marler, *Survey of multi-objective optimization methods for engineering* Struct. Multidisc. Optim. **26** (2004) 369–395.

[19] K. Miettinen, L. Kirilov, *Interactive reference direction approach using implicti parametrization for nonlinear multiobjective optimization*, MCDM 2004, Whistler, B. C. Canada August 6-11, 2004.

[20] I. Nikolić, S. Borović., *Višekriterijumska optimizacija: metode, primena u logistici*, Centar vojnih škola Vojske Jugoslavije, Beograd, 1996.

[21] S. Opricović, *Optimizacija sistema*, Građevinski fakultet Univerziteta u Beogradu, 1992.

[22] A. Osyczka, *Multicriterion Optimization in Engineering with Fortran Programs*, New York: John Wiley and Sons, 1984.

[23] K. Schittkowski, *Multicriteria Optimization -User's Guide-*, http://www.klaus-schittkowski.de, November 2004.

[24] W. Stadler, *Fundamentals of multicriteria optimization*, In: Stadler,W. (ed.) Multicriteria Optimization in Engineering and in the Sciences, pp. 125. New York: Plenum Press, 1988.

[25] P.S. Stanimirović and I.P. Stanimirović, *Implementation of polynomial multi-objective optimization in MATHEMATICA*, Structural Multidisciplinary Optimization, DOI 10.1007/s00158-007-0180-9.

[26] P.S. Stanimirović, G.V. Milovanović, *Programski paket MATHEMATICA i primene*, Elektronski fakultet u Nišu, Edicija monografije, Niš, 2002.

[27] P.S. Stanimirović, N.V. Stojković, M.D. Petković, *Run-time transformations in implementation of linear multi-objective optimization*, XVI Conference on Applied Mathematics, N. Krejic, Z. Luzanin, eds. Department of Mathematics and Informatics, Novi Sad, (2004),157–165.

[28] H. Voogd, *Multicriteria Evaluation for Urban and Regional Planning*, London: Pion, 1983.

[29] M. Vujošević , M. Stanojević , N. Mladenović , *Metode optimizacije*, Društvo operacionih istraživača Jugoslavije, Beograd, 1996.

Internet lokacija http://www.laboi.fon.bg.ac.yu/download/MetOpt/uvoduvko.pdf

[30] F.M. Waltz *An engineering approach: hierarchical optimization criteria*, IEEE Trans. Autom. Control AC-12, (1967) 179-180.

[31] R.E. Wendell, D.N. Lee, *Efficiency in multiple objective optimization problems*, Mathematical Programming, **12** (1977) 406-414.

[32] S. Wolfram, *The Mathematica Book, 4th ed.*, Wolfram Media/Cambridge University Press, 1999.

[33] L.A. Zadeh, *Optimality and non-scalar-valued performance criteria*, IEEE Trans. Autom. Control AC-8, (1963) 5960.

[34] S. Zionts, *Multiple criteria mathematical programming: an updated overview and several approaches.*, In: Mitra, G. (ed.) Mathematical Models for Decision Support, 1988, pp. 135167, Berlin: Springer-Verlag

[35] K. Zotos, *Performance comparison of Maple and Mathematica*, Appl. Math. Comput. (2006), doi:10.1016/j.amc.2006.11.008.

Index

Predgovor

Zadatak matematičkog programiranja je da odredi maksimum (minimum) funkcije koja zavisi od više promenljivih pod uslovom da su ove promenljive nenegativne i da zadovoljavaju ograničenja u obliku jednačina i/ili nejednačina. Ciljna funkcija koja se optimizira naziva se *ciljna funkcija* ili funkcija cilja.

Matematičko programiranje je disciplina koja se izučava od tridesetih godina prošlog veka. Do danas je matematičko programiranje zadržalo status jedne od najdinamičnijih i najprimenljivijih matematičkih disciplina. Kao takvo, ono je našlo veliku primenu u mnogim naučnim granama, na primer u ekonomiji, menadžmentu, bankarstvu, na svim tehničkim fakultetima kao i u privredi.

Popularnost obrađene teme će pomoći da ova knjiga bude od koristi mnogima koji žele da se naučno bave ovom problematikom, kao i onima koji rešavaju probleme optimizacije realnih sistema.

Motivacija za izradu ove knjige je odsustvo slične literature, kako na srpskom jeziku, tako i u svetskim razmerama. Uz priložene teorijske rezultate data je implementacija velikog broja metoda. Implementacija je izvršena u proceduralnim programskim jezicima PASCAL, C i VISUAL BASIC kao i u funkcionalnom i simboličkom programskom jeziku MATHEMATICA. U tom smislu, ova knjiga je nastavak nastavak prethodne knjige *Programski paket Mathematica i primene*, Niš, 2002.

Prva glava je posvećena simpleks metodu. Prvo su detaljno date osnove ovog metoda, a zatim algoritam za određivanje početnog rešenja bez uvođenja veštačkih promenljivih koji se ne javlja tako često u literaturi. Nakon toga se razmatra dualni i revidirani simpleks metod. Za rešavanje problema cikliranja navedeni su leksikografski metod kao i manje poznata ali jednostavna Blandova pravila. Sledi složenost simpleks algoritma kao i detaljan opis Minty-Klee polieadara. U nastavku su dati naši rezultati o poboljšanju algoritma za odredjivanje početnog bazično dopustivog rešenja, implementacija i uporedni testovi sa postojećim algoritmima. Posle postoptimalne analize slede rezultati o direktnim metodima za rešavanje problema linearnog programiranja.

Uveden je takozvani *metod minimalnih uglova*, široko prihvaćen u literaturi pod tim nazivom. Ovaj metod se koristi za ubrzanje simpleks metoda kod nekih klasa problema linearnog programiranja. Metod u najgorem slučaju daje pogodnu tačku za start standardnog simpleks algoritma. Za neke probleme linearnog programiranja simpleks metod samo potvrđuje da je izabrana tačka optimalno rešenje i algoritam se završava u samo jednom koraku. Glavna ideja na kojoj je metod zasnovan jeste da se poboljša izbor inicijalne baza. Poznato je da se optimalno teme formira kao presek n ograničenja, gde je n broj promenljivih u linearnom problemu. Ovakvih n ograničenja koja formiraju optimalno teme trebalo bi da zahvataju najmanje uglove sa ciljnom funkcijom. U najgorem slučaju, metod minimalnih uglova daje teme

konveksnog skupa koje je u susedstvu sa ekstremnom tačkom. Na taj način metod minimalnih uglova obezbeđuje u izvesnim slučajevima značajnu redukciju broja iterativnih koraka u odnosu na klasičan simpleks metod. Osim toga, pored ubrzanja konvergencija, za određene tipove problema se smanjuje i dimenzija problema.

Na kraju ove glave sledi opis softvera MarPlex u kome su implementirani naši rezultati.

Druga glava je posvećena primal-dual metodima unutrašnje tačke.

Dugi niz godina je Dantzigov simpleks metod bio osnova za rešavanje problema linearnog programiranja i mnogobrojnih srodnih problema. U novije vreme, sa pojavom računara, razvijaju se i metode unutrašnje tačke koje se zasnivaju na iterativnim procesima. Zbog toga se osim klasične linearne algebre traži i poznavanje osnova numeričke analize i programiranja. Knjiga je namenjena prvenstveno studentima osnovnih i postdiplomskih studija ali u njoj ima materijala koji može korisno da posluži ekonomistima, menadžerima, onima koji se bave tehničkim naukama kao i onima koji se bave praktičnom primenom metoda optimizacije.

Primal–dual metodi su izabrani iz dva razloga. Istorijski važan Karmarkarov metod je već detaljno opisivan u postojećoj literaturi na našem jeziku tako da se njime nismo posebno bavili. Sa druge strane, prima-dual metodi su se pokazali kao dominantni u već postojećim softverskim paketima a koliko je nama poznato, i pored velike važnosti, vrlo malo se pominju u našoj literaturi. Napomenimo da smo u ovoj glavi uglavnom davali rezultate bez detaljnih dokaza zato što je obimnost materije takva da zahteva posebnu monografiju samo na tu temu. Prvo je data osnovna šema primal-dual algoritma a zatim se razmatraju potencijano-redukcioni metodi i algoritmi koji slede centralnu putanju. Nakon toga sledi deo o algoritmima koji startuju iz nedopustive tačke koji su od velikog praktiňog značaja. Posle opisa Mehrotraovog algoritma koji je primenjen u svim modernim softverskim paketima, sledi opis naše simboličke implementacije i dve jednostavne heuristike za pronalaženje početnog rešenja kao i test primeri. Nakon toga slede naši rezultati o smanjenju dimenzije problema i implementacija. U nastavku smo razmatrali prošireni i normalni sistem jednačina u algoritmu i dali naša numerička iskustva sa primenom oba pristupa. Na kraju se razmatra važan problem stabilizacije algoritma jer u završnim koracima algoritma matrice problema postaju po pravilu loše uslovljene što može dovesti do numeričkih problema. Saradnja sa dr Verom Kovačević-Vujčić je bila inspiracija za ta istraživanja.

U trećoj glavi opisane su neke primene linearnog programiranja jer je namera autora da u ovoj monografiji osim teorijskih rezultata da i praktičnu primenu opisanih algoritama. Na početku se daje primena u transportnim problemima, a zatim klasični poznati primeri iz ekonomije. Dajemo i dva ne tako često pominjana primera primene u astronomiji i elektronici za projektovanje optimalne maske teleskopa i digitalnih FIR filtara.

Dinamičko programiranje je predmet četvrte glave. Dinamičko programiranje se obično primenjuje u problemima optimizacije: problem može imati mnogo rešenja, svako rešenje ima vrednost, a traži se rešenje koje ima optimalnu (najveću ili najmanju) vrednost. U slučaju da postoji više rešenja koja imaju optimalnu vrednost,

obično se traži bilo koje od njih. Sama reč "programiranje" ovde se (kao i u linearnom programiranju) odnosi na popunjavanje tabele pri rešavanju problema, a ne na upotrebu kompjutera i programskih jezika. Uopšteno govoreći, primenom dinamičkog programiranja problem se rešava tako što se uoči hijerarhija problema istog tipa, sadržanih u glavnom problemu, i rešavanje se počne od najjednostavnijih problema. Vrednosti i delovi rešenja svih rešenih podproblema se pamte u tabeli, pa se dalje njihovim kombinovanjem dobija rešenja većih podproblema sve do rešenja glavnog problema.

Posle uvodnih pojmova u ovoj glavi sledi definicija funkcije procesa i vrste procesa. U nastavku prvo dajemo jednostavnije primene ali nismo zaobišli ni dobro poznate probleme trgovačkog putnika, dinamičkog transportnog problema i problem ranca. Data je i primena za izračunavanje generalisanog inverza matrice. Zatim sledi veliki broj raznovsnih zadataka rešenih primenom dinamičkog programiranja koji pokazuju veliku praktičnu primenljivost ovog metoda. Za većinu rešenih zadataka dat je i odgovarajući softver. Na kraju se razmatra veza između memoizacije, rekurzije i dinamičkog programiranja.

U petoj glavi se razmatra višekriterijumska optimizacija i pojam Pareto optimalnosti. U nastavku je dat pregled različitih metoda za rešvanje problema višekriterijumske optimizacije.

Autori su svesni činjenice da su mnogobrojne oblasti koje su u tesnoj vezi sa linearnim programiranjem (transpotni problem, teorija igara, celobrojno programiranje, kvadratno programiranje) ostale neobrađene. Razlog leži u činjenici da su autori uglavnom pratili svoj naučni afinitet kao i da te oblasti mogu biti materijal za posebnu knjigu koja bi bila prirodni nastavak ove. Sve glave u knjizi su uglavnom delo zajedničkog rada tako da za eventualne propuste svi autori snose podjednaku odgovornost.

S obzirom na raznovrsnost oblasti koje su obrađene u knjizi, odlučili smo da reference budu na kraju svake glave. Time smo doprineli boljoj čitljivosti priloženog materijala, uz minimalno preklapanje referenci navedenih u glavama.

Ova knjiga je nastala kao plod višegodišnje međusobne saradnje sva tri autora. Priloženi materijal je praćen obimnom literaturom kao i originalnim rezultatima autora. O tome svedoči veći broj radova i dve monografije čiji su rezultati uklopljeni u poznate rezultate. Neki od citiranih radova autora su citirani u vodećim međunarodnim i domaćim časopisima. Metod minimalnih uglova, koji je opisan u prvoj glavi je višestruko citiran. On je bio predmet nekih doktorskih disertacija odbranjenih na poznatim univerzitetima. Osim toga, citiran je rezultat koji uvodi eliminaciju suvišnih ograničenja pomoću teorije igara. Iz druge glave citirani su rezultati koji se odnose na početnu tačku primal-dual algoritma kao i na njegovu stabilizaciju.

Kolege Ivan Stanković, Milan Bašić, Zoran Jovanović, Branimir Momčilović, Ivan Stanimirović, dr Dragan Đorđević, dr Milan Tasić su svojom saradnjom sa autorima doprineli objavljivanju ove monografije na čemu im se iskreno zahvaljujemo. Posebnu zahvalnost na saradnji dugujemo dr Veri Kovačević-Vujčić čiji su radovi bili inspiracija za neka naša istraživanja.

Recenzenti dr Gradimir V. Milovanović, redovni profesor Elektronskog Fakulteta u Beogradu i dr Stojan Bogdanović, redovni profesor Ekonomskog fakulteta u Nišu, pomogli su svojim savetima i sugestijama u poboljšanju kvaliteta teksta. Koristimo ovu priliku da im se zahvalimo za trud koji su uložili.

Autori su svesni činjenice da u knjizi postoje greške koje nisu primećene. Bićemo zahvalni svima koji nam pomognu u otklanjanju tih grešaka, kako bi buduća izdanja ove knjige bila poboljšana.

Niš, avgust 2007. Autori

Sadržaj

Glava 1

Linearno programiranje

U prvom poglavlju ove glave dajemo matematički model problema linearnog programiranja i opisujemo uobičajene pretpostavke pod kojima se problem rešava. Zatim ćemo proučiti definiciju i rešavanje problema linearnog programiranja. To najpre podrazumeva matematički model i osnovne definicije, kao i osnovne osobine skupa dopustivih rešenja. Zatim ćemo navesti najjednostavniji geometrijski metod za rešavanje problema linearnog programiranja i pokazati kako se on može implementirati.

U drugom poglavlju ćemo proučiti simpleks metod za rešavanje problema linearnog programiranja u opštem obliku i detaljno ćemo opisati sve njegove faze kao i dualni simpleks metod. Takođe ćemo razmotriti još jednu verziju simpleks metoda, revidirani simpleks metod, kojom se rešava problem numeričke stabilnosti klasičnog simpleks metoda. Na kraju, ukazaćemo na problem cikliranja kao i na dva načina da se taj problem prevaziđe. Takođe ćemo pokazati da je simpleks algoritam, i pored svojih odličnih osobina na praktičnim problemima, eksponencijalne složenosti.

Treće poglavlje predstavlja naše originalne rezultate preuzete iz radova [58, 78, 70], i bavi se modifikacijama i poboljšanjima pojedinih faza simpleks metoda. U radovima [70, 78] korišćen je taj algoritam, jer ne zahteva uvođenje veštačkih promenljivih. U tim radovima su uvedena dva algoritma za dobijanje početnog bazično dopustivog rešenja u fazi I dvofaznog simpleks algoritma opisanog u [51] i [79]. Opisano je novo pravilo za izbor bazičnih i nebazičnih promenljivih, tj. za izbor promenljive koja ulazi u bazu kao i promenljive koja napušta bazu. Na kraju ove glave detaljno ćemo opisati implementaciju simpleks i revidiranog simpleks metoda, kao i rezultate testiranja programima MarPlex i RevMarPlex koje su nastale kao rezultat implementacije modifikacija uvedenih u radovima [58, 78, 70]. Pokazaćemo da su uvedene modifikacije u praksi pokazale bolje performanse u odnosu na klasične algoritme.

U čevrtom poglavlju se izučava postoptimalna analiza simpleks metoda.

U petom poglavlju razmatramo direktne metode za rešavanje problema linearnog programiranja. Dajemo rezultate iz našeg rada [76] i rada [67] gde ćemo ukazati na

moguću primenu naših rezultata iz teorije generalisanih inverza [59, 60, 71, 61].

Na kraju ćemo dati kôd najvažnijih procedura programa MarPlex, RevMarPlex i GEOM uz objašnjenje implementacionih detalja.

1 Opšti zadatak linearnog programiranja

Zadatak linearnog programiranja je da odredi maksimum (minimum) linearne funkcije koja zavisi od više promenljivih pod uslovom da su ove promenljive nenegativne i da zadovoljavaju linearna ograničenja u obliku jednačina i/ili nejednačina. Linearna ciljna funkcija koja se optimizira naziva se *ciljna funkcija* ili funkcija cilja . Linearno programiranje je jedan od najefikasnijih pristupa formulisanju i rešavanju složenih problema donošenja odluka i kao takvo predstavlja osnovnu disciplinu operacionih istraživanja.

Linearno programiranje se javilo u oblasti rešavanja praktičnih zadataka kao što su planiranje ekonomskog razvoja kako na nivou radne organizacije tako i na širem regionalnom ili najširem društvenom planu.

Opšte prihvaćeno mišljenje je da je prvi rad koji pripada linearnom programiranju objavio L.V. Kantorovič 1939. godine [35]. U njemu se prvi put definiše transportni zadatak. Među prve radove iz ove oblasti spada i rad pukovnika Vlastimira Ivanovića iz 1940. godine, pod nazivom "Pravila za proračun potrebnog broja transportnih sredstava" [34]. U ovom radu, pukovnik Ivanović je pokazao kako se izračunava minimalan broj vozila za prevoz date količine materijala korišćenjem "Principa najveće ekonomije". Po nekim autorima se nastanak linearnog programiranja vezuje za članak [24] iz 1931. godine. Linearni modeli se koriste i u rešavanju određenih transportnih zadataka, koji su poznati u teoriji i praksi kao transportni problemi, kao i u drugim oblastima planiranja. Prvi algoritam za rešavanje transportnog problem razradio je F.F Hitchock u radu [31] 1941. godine. Na formulisanju transportnog problema i njegovom rešavanju radio je i T.C. Koopmans, koji je rezultate svojih istraživanja objavio 1947. godine. Najveći doprinos razvoju linearnog programiranja dao je G.B. Dantzig, koji je 1947. godine formulisao opšti problem linearnog programiranja i postavio simpleks metod. U radu [20] su izložene osnove simpleks metoda. Radovi John von Neumanna iz tog perioda su omogućili teorijsko formulisanje dualnog problema kao i pronalaženje veze između linearnog programiranja i teorije igara. Metod za otklanjanje degeneracije predložen je u radu [12]. Značajni su i radovi S. Gass-a i T. Saaty-a iz 1955. o parametarskom programiranju kao i radovi E. Beale-a i R. Gomory-a iz 1958. o celobrojnom programiranju.

Problemi matematičkog programiranja javljaju se u različitim disciplinama. Na primer, menadžer na berzi mora da odabere ulaganja koja će generisati najveći mogući profit a da pri tome rizik od velikih gubitaka bude na unapred zadatom nivou. Menadžer proizvodnje organizuje proizvodnju u fabrici tako da količina proizvoda i kvalitet budu maksimalni a utrošak materijala i vremena i škart minimalni, pri čemu ima na raspolaganju ograničene resurse (broj radnika, kapacitet mašina, radno vreme). Naučnik pravi matematički model fizičkog procesa koji najbolje opisuje određenu fizičku pojavu, a na raspolaganju ima konačni broj mernih

rezultata. Takođe, model ne sme biti suviše komplikovan.

U svim ovim situacijama možemo da indetifikujemo tri zajednička pojma [7]:

1. Postoji globalna veličina (cilj) koja se želi optimizovati (profit, razlika između predviđanja modela i eksperimentalnih podataka).

2. Uz globalni cilj obično su prisutni dodatni zahtevi ili ograničenja, koja moraju biti zadovoljena (ograničen rizik, resursi, kompleksnost modela).

3. Postoje određene veličine tako da ako se njihove vrednosti izaberu "dobro", zadovoljeni su i cilj i ograničenja. Te veličine se nazivaju optimizacione promenljive ili parametri.

Znači, da bi zadali problem matematičkog programiranja moramo:

1. odabrati jednu ili više optimizacionih promenljivih,

2. odabrati funkciju cilja i

3. formirati skup ograničenja.

Nakon toga, identifikuje se klasa problema kojoj dobijeni matematički model pripada i bira se metod za njegovo rešavanje.

Postoji više metoda za rešavanje problema linearnog programiranja [7, 10, 33, 73, 84]. Geometrijski metod je primenljiv na probleme kod kojih je broj promenljivih $n = 2$ ili kada je $n - m = 2$, gde je m broj ograničenja. U principu, zadatak linearnog programiranja se može rešiti geometrijski i u slučaju $n = 3$. Nedostatak geometrijskog metoda je u tome što ne rešava opšti zadatak linearnog programiranja već samo neke specijalne slučajeve. Do prvog opšteg metoda za rešavanje problema linearnog programiranja došao je američki matematičar Danzing 1947. godine [20]. Ovaj metod je poznat kao simpleks metod linearnog programiranja. On je takođe formulisao opšti oblik problema linearnog programiranja i dao algoritam za njegovo rešavanje, poznat kao ***simpleks metod***. Dantzig-ov rad je više godina kružio među stručnjacima i poslužio kao osnova svim narednim razmatranjima problema linearnog programiranja. Iako su u međuvremenu pronađeni i drugi metodi, ovaj metod se i danas koristi kroz brojne modifikacije (vidi radove [8, 25]). I geometrijski i simpleks metod traže maksimum (minimum) funkcije cilja na rubovima oblasti ograničenja.

U kasnijem periodu se pojavljuju i alternativni pristupi rešavanju problema linearnog programiranja. Pomenimo radove u kojima se koriste generilisani inverzi [63, 64, 65, 11], kao i radove Conna [14] i Daxa [22] koji ne koriste simpleks metod. Iako je praksa pokazala da je simpleks metod veoma efikasan, 1972. godine su V. Klee (Kli) i G.L. Minty (Minti) dokazali da on nije polinomijalan [38]. Oni su konstruisali jednostavan primer zadatka linearnog programiranja kod koga je dopustivi skup deformisana n-dimenzionalna kocka sa 2^n temena, za čije je rešavanje simpleks metodu sa standardnim izborom vodećeg elementa potrebno $2^n - 1$ iterativnih koraka. Prvi polinomijalni algoritam za rešavanje problema linearnog programiranja

je dao Hačijan u radu [37] 1979. godine. Time je napravljen veliki zaokret u razvoju linearnog programiranja. Hačijan je pokazao da njegov metod elipsoida rešava problem linearnog programiranja za $O(n^4L)$ elementarnih aritmetičkih operacija, gde je L broj bitova potrebnih za zapis svih parametara problema (matrice ograničenja, funkcije cilja i desne strane ograničenja. Hačijan je rešio vrlo važno teorijsko pitanje. Međutim, ispostavilo se da metod elipsoida nije primenljiv u praksi. Testiranja su pokazala da je simpleks metod daleko efikasniji, jer on "retko" dostiže gornju granicu složenosti, za razliku od metoda elipsoida, koji to često čini. Ovakav zaključak je inicirao da se pronađu novi metodi za rešavanje problema linearnog programiranja koji će osim polinomijalne složenosti imati i praktičnu primenljivost. Prvi takav metod predložio je 1984. godine N. Karmarkar u radu [36]. Karmarkarov algoritam je neke test primere rešavao i do 50 puta brže od simpleks metoda. Karmarkarov rezultat je izazvao pravu revoluciju u razvoju ove oblasti. Posle Karmarkarovog metoda pojavila se čitava familija metoda koji su poznati pod nazivom *unutrašnji metodi*. Danas su ***primal-dual*** metodi unutrašnje tačke dominantni za rešavanje problema linearnog programiranja [93]. Iako su otkriveni polinomijalni metodi, simpleks metod se i danas koristi i kroz brojne modifikacije još uvek živi. Simpleks metod je mnogo bolji od metoda unutrašnje tačke na tzv. loše uslovljenim problemima, zbog svoje spore, ali pouzdane konvergencije. Isto tako, u praksi se često javlja potreba za rešavanjem klase srodnih problema gde se optimalno rešenje jednog od njih može efikasno iskoristiti kao početna tačka za rešavanje ostalih problema. I u ovom slučaju je simpleks metod bolji izbor od metoda unutrašnje tačke.

Osim teorijskih rezultata pojavili su se i programski paketi za rešavanje problema linearnog programirana, i koji su bazirani na teorijskim rezultatima. Ovi programski paketi su našli široku primenu u praksi.

Unutrašnji metodi dele se na *primalne, dualne* i *primalno-dualne*. Primalni metodi rešavaju problem linearnog programiranja u standardnom obliku

$$\begin{aligned} \min \quad & c^T x, \\ \text{p.o.} \quad & Ax = b, \quad x \geq 0, \end{aligned}$$

gde je A matrica tipa $m \times n$, dok su a, b i c vektori odgovarajućih dimenzija. Dualni problemi rešavaju dualni problem oblika

$$\begin{aligned} \max \quad & b^T y, \\ \text{p.o.} \quad & A^T y + s = c, \quad s \geq 0. \end{aligned}$$

Primalno-dualni metodi rade simultano i sa primalnim i sa dualnim problemom (primal-dual metodi). Danas su primal-dual metodi dominantni. Ovi metodi su opisani u [93]. Izučavanje metoda unutrašnje tačke za rešavanje problema linearnog programiranja pripada savremenom i modernom trendu u svetu što potvrđuju već objavljene monografije o polinomijalnim metodima, kao što su [7, 40, 49, 81, 93]. Poslednjih desetak godina se objavljuje veliki broj radova iz ove oblasti tako da bibliografija broji više hiljada naslova.

Postoje tri alata u rešavanju problema linearnog programiranja.

1. Modeli: formulacija problema u detaljnim matematičkim terminima.
2. Algoritmi: tehnike za rešavanje modela.
3. Kompjuteri i softver: mašine za izvršavanje algoritamskih koraka.

Postoji veći broj programskih paketa koji su namenjeni rešavanju problema linearnog programiranja.

HOPDM je napisan u programskom jeziku FORTRAN [29].

LIPSOL je napisan u programskim jezicima MATLAB i FORTRAN. Deo koda koji se odnosi na **Sparse Cholesky** faktorizaciju i rešavanje linearnih sistema je napisan u FORTRAN-u, a ostatak koda u MATLAB-u. Algoritam je opisan u [94].

LOQO je napisan u programskom jeziku C, a opisan je u [83].

PCx je napisan u programskim jezicima C i FORTRAN. **Sparse Cholesky** kod je napisan u FORTRAN-u, a ostatak koda u C-u. Detaljan opis se može naći u [19].

MOSEK je napisan u programskom jeziku C++ i predstavlja jedan od najjačih solvera ne samo za probleme linearnog programiranja već uopšte i za probleme kvadratnog i nelinearnog programiranja. Za razliku od predhodno pomenutih programa, ovaj program je komercijalan.

LINDO - Linear Interactive and Discrete Optimizer - je interaktivni softverski paket koji se može koristiti za rešavanje problema linearnog programiranja. Razvijen je 1980. godine i od tada je prilagođen Windows okruženju i grafički orijentisanim programima. Softverski paket LINDO se koristi za rešavanje problema zadatih direktno sa tastature.

MarPlex je naš program. Napisan je u programskom jeziku Visual Basic 6.0 i koristi simpleks metod, odnosno modifikacije prezentovane u odeljku 3 ove glave. Izdvaja se od ostalih programa zbog svog jednostavnog interface-a kao i ne toliko obimnog i citljivog koda. Pošto je implementiran u Visual Basic-u, i koristi simpleks metod, zaostaje po pitanju brzine.

RevMarPlex je takođe naš program i predstavlja revidiranu verziju programa MarPlex napisanu u programskom jeziku MATHEMATICA.

Implementacija nekih metoda linearnog programiranja u programskom jeziku MATHEMATICA se može naći u [7].

1.1 Matematički model

Opšti oblik problema linearnog programiranja možemo izraziti na sledeći način. Odrediti vrednosti promenljivih $x_1, \ldots, x_k$ koje odgovaraju linearnim jednačinama

i nejednačinama

$$\begin{aligned} N_i^{(1)}: \quad & \sum_{j=1}^{k} a_{ij}x_j \;\leq\; b_i, \quad i \in I_1 \\ J_i: \quad & \sum_{j=1}^{k} a_{ij}x_j \;=\; b_i, \quad i \in I_2 \\ N_i^{(2)}: \quad & \sum_{j=1}^{k} a_{ij}x_j \;\geq\; b_i, \quad i \in I_3 \\ & x_j \;\geq\; 0, \quad j \in J \subseteq \{1,\ldots,k\}, \end{aligned} \tag{1.1.1}$$

gde je $I_1 \cup I_2 \cup I_3 = \{1,\ldots,m\}$, $I_1 \cap I_2 = \emptyset$, $I_1 \cap I_3 = \emptyset$, $I_2 \cap I_3 = \emptyset$, tako da linearna ciljna funkcija

$$z(x) = z(x_1,\ldots,x_k) = c_1x_1 + \cdots + c_kx_k \tag{1.1.2}$$

ima ekstremum, tj. minimum ili maksimum. Pri tome su a_{ij}, b_i, c_j poznati realni brojevi.

Po konvenciji, u opštem slučaju vektor $y \in \mathbb{R}^k$ predstavlja $k-$torku realnih brojeva $y = (y_1,\ldots,y_k)$, dok se u matričnim formulama podrazumeva da je y matrica tipa $k \times 1$ (vektor), tj.

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}, \quad y^T = [y_1 \ldots y_k].$$

Dalje, $y \geq 0$ označava $y_1 \geq 0, \ldots, y_k \geq 0$.

Rešenje $x = (x_1, x_2, \ldots, x_k)$ u primenama najčešće ima značenje plana ili programa (proizvodnje, prevoza), pa je otuda ovaj zadatak dobio naziv "programiranje", a naziv "linearno programiranje" označava da su ograničenja promenljivih (1.1.1), kao i funkcija cilja (1.1.2) linearni.

Proizvoljno rešenje sistema nejednačina (1.1.1) može da se zamisli i kao vektor $x = (x_1, x_2, \ldots, x_k)$, koji u geometrijskoj interpretaciji predstavlja tačku k-dimenzionalnog prostora $\mathbb{R}^k$. Svako nenegativno rešenje sistema nejednačina (1.1.1) nazivamo *dopustivim rešenjem* . Ako sa Ω_P označimo skup dopustivih rešenja, onda je $\Omega_P \subset R^k$. Za skup Ω_P pretpostavljamo da nije prazan i da sadrži najmanje jedan element (slučaj kada zadatak (1.1.1)-(1.1.2) ima rešenje). Optimalno rešenje $x^* = (x_1^*, \ldots, x_k^*) \in \Omega_P$ zadatka linearnog programiranja je ono dopustivo rešenje za koje funkcija cilja $z(x) = z(x_1,\ldots,x_k)$ dostiže maksimum (minimum). U slučaju maksimizacije funkcije cilja ispunjen je uslov

$$z(x^*) = \max_{x \in \Omega_P} z(x)$$

ili

$$z(x^*) \geq z(x) \quad \forall x \in \Omega_P.$$

Često se u praksi traži da optimalna vrednost funkcije cilja bude najmanja na skupu dopustivih rešenja Ω_P. U ovom slučaju optimalno rešenje $x^* \in \Omega_P$ je ono dopustivo rešenje za koje je ispunjeno

$$z(x^*) = \min_{x \in \Omega_P} z(x).$$

ili

$$z(x^*) \leq z(x) \quad \forall x \in \Omega_P.$$

U jednom konkretnom zadatku rešava se samo problem maksimizacije, odnosno problem minimizacije. Problem minimuma se može transformisati u problem maksimuma (i obratno) jednostavnim množenjem ciljne funkcije sa -1, koristeći sledeći rezultat.

Propozicija 1.1.1 *Dati kriterijum optimizacije može da se zameni suprotnim, pri čemu ta zamena ne utiče na optimalno rešenje, to jest ako je za $x^* \in \Omega_P$ ispunjeno*

$$z(x^*) = \max_{x \in \Omega_P} z(x)$$

onda je

$$-z(x^*) = \min_{x \in \Omega_P} [-z(x)]$$

i obrnuto.

Dokaz. Prema pretpostavci teoreme ispunjeno je

$$z(x^*) \geq z(x) \qquad \forall x \in \Omega_P.$$

Ako ovu nejednakost pomnožimo sa -1, dobija se

$$-z(x^*) \leq -z(x) \qquad \forall x \in \Omega_P,$$

čime je teorema dokazana. □

Zadatak linearnog programiranja ima rešenje ako veličina z_{max} (z_{min}) ima konačnu vrednost na skupu Ω_P dopustivih rešenja. Zadatak linearnog programiranja nema rešenja ako sistem nejednačina (1.1.1) nema nenegativnih rešenja ili ako veličina z_{max} (z_{min}) nema konačnu vrednost.

Ograničenja (1.1.1) određuju u k-dimenzionalnom prostoru konveksnu oblast Ω_P ograničenu skupom hiperravni

$$\sum_{j=1}^{k} a_{ij} x_j = b_i, \ i = 1, \ldots, m.$$

Nazovimo ovu oblast Ω_P-poliedrom, mada u izvesnim slučajevima ona može biti i beskonačna. Ta oblast se naziva *oblast dopustivih rešenja*. Kako su funkcije koje treba maksimizirati ili minimizirati linearne, klasični matematički metodi pokazuju da se maksimumi ili minimumi funkcije cilja $z(x) = z(x_1, \ldots, x_k)$ dostižu na

granicama oblasti Ω_P koja je određena datim ograničenjima. Ako hiperravan $z(x) = \text{const}$ nije paralelna ni sa jednom od pomenutih hiperravni, koje predstavljaju strane poliedra, onda će ciljna funkcija $z(x)$ dostići maksimum (minimum) u jednom od temena poliedra.

Neka je $A = [a_{ij}]_{m\times k}$ data matrica sa vrstama $V_1, \ldots, V_m$, neka su $b \in \mathbb{R}^m$ i $c \in \mathbb{R}^k$ dati vektori i neka je $x \in \mathbb{R}^k$ nepoznat vektor. U *matričnom obliku* problem (1.1.1)-(1.1.2) se može zapisati na sledeći način:

$$\begin{array}{lll} \min & c^T x, & \\ \text{p.o.} & V_i^T x \le b_i, & i \in I_1, \\ & V_i^T x = b_i, & i \in I_2, \\ & V_i^T x \ge b_i, & i \in I_3, \\ & x_j \ge 0, & j \in J \subseteq \{1, \ldots, k\}, \end{array} \tag{1.1.3}$$

gde je $I_1 \cup I_2 \cup I_3 = \{1, \ldots, m\}$, $I_1 \cap I_2 = \emptyset$, $I_1 \cap I_3 = \emptyset$, $I_2 \cap I_3 = \emptyset$.

Ukoliko je $I_2 = \{1, \ldots, m\}$ i $J = \{1, \ldots, k\}$, $A = [a_{ij}]_{m\times n}$, $c, x \in \mathbb{R}^n$, problem (1.1.3) se svodi na tzv. *standardni oblik* problema linearnog programiranja

$$\begin{array}{lll} \min & c^T x, & \\ \text{p.o.} & Ax = b, & \\ & x \ge 0. & \end{array} \tag{1.1.4}$$

U slučaju $I_3 = \{1, \ldots, m\}$ i $J = \{1, \ldots, k\}$ problem (1.1.3) se svodi na tzv. *simetrični oblik*

$$\begin{array}{lll} \min & c^T x, & \\ \text{p.o.} & Ax \ge b, & \\ & x \ge 0. & \end{array} \tag{1.1.5}$$

Bez narušavanja opštosti se može pretpostaviti da je $b_i \ge 0$ za svako $i = 1, \ldots, m$ (u suprotnom, odgovarajuća nejednačina se može pomnožiti sa -1).

Za vektor x koji zadovoljava uslove (1.1.1) kažemo da je *dopustivo rešenje* problema. Dopustivo rešenje x^* je *minimalno* ako je

$$c^T x^* \le c^T x$$

za svako dopustivo rešenje x.

Problem zadat u opštem obliku (1.1.1)-(1.1.2) je moguće transformisati u standardni ili simetrični oblik. Ako se uvedu nove nepoznate

$$\begin{aligned} x_{k+i} &= b_i - \sum_{j=1}^{k} a_{ij} x_j, \quad i \in I_1 \\ x_{k+i} &= \sum_{j=1}^{k} a_{ij} x_j - b_i, \quad i \in I_3. \end{aligned}$$

Tada uslovi (1.1.1) prelaze u sistem jednačina sa $k+q=n$ nepoznatih, gde je $q=|I_1|+|I_3|$.

$$\begin{aligned}\sum_{j=1}^{k} a_{ij}x_j + x_{k+i} &= b_i, \quad i \in I_1\\ \sum_{j=1}^{k} a_{ij}x_j &= b_i, \quad i \in I_2\\ \sum_{j=1}^{k} a_{ij}x_j - x_{k+i} &= b_i, \quad i \in I_3.\end{aligned} \tag{1.1.6}$$

Pri tome posmatramo funkciju

$$z = c_1x_1 + \cdots + c_kx_k + \cdots + c_nx_n \tag{1.1.7}$$

gde je $c_{k+1} = \ldots = c_n = 0$.

U sledećoj tabeli je ilustrovana veza između osnovnih oblika linearnog problema.

Ograničenje	Kanonska forma	Standardna forma
$a_i^T x \le b_i$	$-a_i^T x \ge -b_i$	$a_i^T x + s_i = b_i,\ s_i \ge 0$
$a_i^T x \ge b_i$		$a_i^T x - s_i = b_i,\ s_i \ge 0$
$a_i^T x = b_i$	$a_i^T x \ge b_i,\ -a_i^T x \ge -b_i$	

Promenljive s_i se nazivaju *slack promenljive* . Uvođenjem svake slack promenljive povećava se dimenzija problema n za 1, i matrica A se proširuje kolonom jedinične matrice I, dok se vektor ciljne funkcije c proširuje nultim elementom.

Promenljive iz skupa $\{1,\ldots,n\}\backslash J$, na koje nije nametnut uslov nenegativnosti, nazivaju se *slobodne promenljive* . Svaku slobodnu promenljivu $x_j,\ j \in \{1,\ldots,n\}\backslash J$ možemo zameniti u ciljnoj funkciji i svim ograničenjima izrazom $x_j^+ - x_j^-$, pri čemu je $x_j^+ \ge 0, x_j^- \ge 0$, i tako dolazimo do modela kod koga je na sve promenljive nametnut uslov nenegativnosti. Ovako postavljen problem je ekvivalentan problemu (1.1.1)-(1.1.2).

Primer 1.1.1 Razmotrimo problem linearnog programiranja

$$\begin{aligned}\min\quad & -5x_1 - 4x_2,\\ \text{p.o.}\quad & x_1 + x_2 \le 80,\\ & 3x_1 + x_2 \le 180,\\ & x_1 + 3x_2 \le 180,\\ & x_1 \ge 0,\ x_2 \ge 0.\end{aligned}$$

Standardni oblik ovog problema se dobija uvođenjem izravnavajućih promenljivih x_3, x_4 i x_5 :

$$\begin{aligned}\min\quad & -5x_1 - 4x_2,\\ \text{p.o.}\quad & x_1 + x_2 + x_3 = 80,\\ & 3x_1 + x_2 + x_4 = 180,\\ & x_1 + 3x_2 + x_5 = 180,\\ & x_i \ge 0,\ i = 1,\ldots,5.\end{aligned}$$

Da bi se problem u opštem obliku preveo na simetričan oblik potrebno je eliminisati ograničenja tipa jednačina. Jednačine $V_i^T x = b_i$, $i \in I_2$ se mogu ekvivalentno zameniti nejednačinama $V_i^T x \geq b_i$, $(-V_i)^T x \geq -b_i$, $i \in I_2$. Ograničenje oblika $(-V_i)^T x \geq -b_i$ dobijeno je množenjem ograničenja oblika $V_i^T x \leq b_i$ sa -1. Dalje, svaku slobodnu promenljivu $x_j, j \in \{1, \ldots, n\} \backslash J$ zamenimo u ciljnoj funkciji i svim ograničenjima kao ranije sa po dve nenegativne promenljive i time dobijamo problem tipa (1.1.5).

Napomena 1.1.1 *Razmotreno svođenje na simetričan oblik važi uz pretpostavku da je sistem jednačina $\{J_i\}_{i \in I_2}$ konzistentan i potpunog ranga. U suprotnom prvo bi trebalo eliminisati suvišne jednačine (na primer Gauss-ovim metodom eliminacije), odnosno formirati ekvivalentan sistem potpunog ranga.*

Iz ovih razmatranja sledi da se ne gubi na opštosti ako se posmatra linearni problem oblika (1.1.4) ili (1.1.5). U nastavku se uglavnom bavimo problemom linearnog programiranja u standardnom obliku jer je najpogodniji za teorijska razmatranja.

U vezi sa postavkom zadatka linearnog programiranja u standardnom obliku napomenimo sledeće. Sistem jednačina $Ax = b$ ima rešenje ako je rang matrice sistema jednačina

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}$$

jednak rangu proširene matrice $[A|b]$, koja se dobija dodavanjem kolone slobodnih članova $b_1, \ldots, b_m$ matrici A (teorema Kronecker-Capellija). Rang r matrice A naziva se rangom sistema i predstavlja broj linearno nezavisnih jednačina među datim ograničenjima. Ako taj uslov nije ispunjen, skup dopustivih rešenja Ω_P je prazan i problem nema rešenja.

Očigledno, rang sistema jednačina $Ax = b$ ne može biti veći od broja jednačina m, to jest uvek je ispunjeno $r \leq m$, ali broj r ne može biti veći ni od broja promenljivih n, pa je takođe ispunjeno $r \leq n$. Ako je $r < m$, tada primenom Gaussovog algoritma dolazimo do zaključka o nesaglasnosti sistema ili eliminišemo $m - r$ suvišnih jednačina u sistemu $Ax = b$. Zato pretpostavimo da je $r = m$, tj. da među ograničenjima (1.1.4) nema linearno zavisnih i da je sistem $Ax = b$ saglasan. U slučaju da je $r = m = n$, sistem ima jedinstveno rešenje $x = A^{-1}b$ pa ostaje samo da se proveri uslov $x \geq 0$. Nas će interesovati slučaj $r = m < n$, kada je broj linearno nezavisnih jednačina manji od broja promenljivih. Tada, ako je sistem jednačina $Ax = b$ saglasan, postoji beskonačno mnogo rešenja. Svako od tih rešenja dobija se tako što se za $n - r = s$ promenljivih izaberu proizvoljne vrednosti, a zatim se vrednosti preostalih r promenljivih izračunavaju iz sistema jednačina $Ax = b$. Promenljive veličine koje izračunavamo nazivamo *zavisnim* ili *bazičnim* (ima ih r), a promenljive veličine čije se vrednosti biraju proizvoljno nazivamo *nezavisnim* ili *slobodnim* promenljivim (njih ima $n - r = s$). Praktično se iz sistema jednačina r zavisnih promenljivih izražava pomoću $n - r = s$ nezavisnih promenljivih, pa se za nezavisne promenljive biraju proizvoljne vrednosti. Kako

se za nezavisne promenljive može birati beskonačno mnogo različitih vrednosti, to sistem jednačina $Ax = b$ ima beskonačno mnogo rešenja.

Definicija 1.1.1 *Nenegativno rešenje, koje se dobija tako što se za nezavisne promenljive izaberu vrednosti jednake nuli, naziva se bazično rešenje zadatka linearnog programiranja.*

Primer 1.1.2 Dat je zadatak linearnog programiranja u opštem obliku:

$$\begin{aligned} \max \quad & z(x) = 2x_1 + 5x_2 \\ \text{p.o.} \quad & x_1 + 4x_2 \leq 24 \\ & 2x_1 + x_2 \leq 21 \\ & x_1 + x_2 \leq 9 \\ & x_1 \geq 0,\ x_2 \geq 0. \end{aligned}$$

Transformisati ovaj zadatak na standardni oblik, naći broj zavisnih i nezavisnih promenljivih, izraziti zavisne promenljive pomoću nezavisnih i naći jedno bazično rešenje.

Levim stranama nejednačina dodajemo nenegativne promenljive x_3, x_4 i x_5 tako da one postanu jednačine:

$$\begin{aligned} x_1 + 4x_2 + x_3 &= 24 \\ 2x_1 + x_2 + x_4 &= 21 \\ x_1 + x_2 + x_5 &= 9. \end{aligned}$$

Sada se zadatak linearnog programiranja (u kanoničnom obliku sa maksimizacijom funkcije cilja) može formulisati na sledeći način: naći nenegativno rešenje $x = (x_1, x_2, x_3, x_4, x_5)$, $x_i \geq 0$, $i = 1, 2, 3, 4, 5$ koje zadovoljava poslednji sistem ograničenja i za koje ciljna funkcija $z(x) = 2x_1 + 5x_2 + 0 \cdot x_3 + 0 \cdot x_4 + 0 \cdot x_5$ dostiže maksimum.

Matrica sistema jedačina i proširena matrica jednake su (redom):

$$A = \begin{bmatrix} 1 & 4 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix} \qquad Ap = [A|b] = \begin{bmatrix} 1 & 4 & 1 & 0 & 0 & 24 \\ 2 & 1 & 0 & 1 & 0 & 21 \\ 1 & 1 & 0 & 0 & 1 & 9 \end{bmatrix}.$$

S obzirom da je rang matrice A jednak rangu matrice Ap ($r = r_A = r_{A_p} = 3$), imamo $r = 3$ zavisne promenljive i $k = n - r = 5 - 3 = 2$ nezavisne promenljive. Za nezavisne promenljive izaberimo x_1 i x_2, i pomoću njih izrazimo zavisne promenljive x_3, x_4, x_5:

$$\begin{aligned} x_3 &= 24 - x_1 - 4x_2 \\ x_4 &= 21 - 2x_1 - x_2 \\ x_5 &= 9 - x_1 - x_2 \end{aligned}$$

Bazično rešenje se sada dobija ako se stavi $x_1 = x_2 = 0$. Sledi, $x_3 = 24$, $x_4 = 21$ i $x_5 = 9$, to jest, dobili smo bazično rešenje $(0, 0, 24, 21, 9)$. Za ovo bazično rešenje ciljna funkcija ima vrednost $z = 0$. Naravno, možemo dobiti i druga bazična rešenja ako drugi par promenljivih odaberemo za nezavisne promenljive. Na primer, ako bismo za nezavisne promenljive odabrali x_2 i x_5, onda bi zavisne promenljive bile definisane na sledeći način:

$$\begin{aligned} x_1 &= 9 - x_2 - x_5 \\ x_3 &= 24 - (9 - x_2 - x_5) - 4x_2 = 15 - 3x_2 + x_5 \\ x_4 &= 21 - 2(9 - x_2 - x_5) - x_2 = 3 + x_2 + 2x_5. \end{aligned}$$

Za $x_2 = x_5 = 0$ dobijamo drugo bazično rešenje $(9, 0, 15, 3, 0)$. Ako funkciju cilja izrazimo pomoću x_2 i x_5:

$$z(x) = 2(9 - x_2 - x_5) + 5x_2 = 18 + 3x_2 - 2x_5$$

vidimo da ona dobija vrednost 18 za $x_2 = x_5 = 0$. Ova vrednost je veća od vrednosti funkcije cilja u prvom bazičnom rešenju, tj. bazično rešenje $(9, 0, 15, 3, 0)$ je bliže optimalnom rešenju od $(0, 0, 24, 21, 9)$. Maksimalna vrednost funkcije cilja iznosi $z_{max} = 33$. Ovu vrednost ciljna funkcija je dostigla u bazičnom rešenju $(4, 5, 0, 13, 0)$, tj. bazično rešenje $(4, 5, 0, 13, 0)$ je optimalno rešenje.

Primer 1.1.3 Fabrika proizvodi dve vrste artikala A_1 i A_2, i to na mašinama M_1 i M_2. Za artikal A_1 mašina M_1 radi 2^h, a mašina M_2 radi 4^h, i za vrstu A_2 mašina M_1 radi 4^h, a mašina M_2 radi 2^h. Fabrika dobija 3500 dinara po jedinici proizvoda A_1, a 4800 dinara po jedinici proizvoda A_2. Koliko treba proizvoditi artikala A_1 i A_2 i kako iskoristiti rad mašina M_1 i M_2 da dnevna dobit fabrike bude maksimalna?

Rešenje: Neka je x_1 broj proizvedenih artikala A_1, a x_2 broj proizvedenih artikala A_2 u toku dana. Tada je dnevna dobit fabrike:

$$z(x) = 3500x_1 + 4800x_2$$

uz uslove:

$$\begin{aligned} 2x_1 + 4x_2 &\leq 24 \\ 4x_1 + 2x_2 &\leq 24 \\ x_1 \geq 0, x_2 &\geq 0. \end{aligned}$$

Ovim smo dobili simetrični oblik. Ako sada uvedemo slack promenljive x_3 i x_4 dobijamo ekvivalentan problem u standardnom obliku:

$$\begin{aligned} \max \quad & 3500x_1 + 4800x_2 \\ \text{p.o.} \quad & 2x_1 + 4x_2 - 24 = -x_3 \\ & 4x_1 + 2x_2 - 24 = -x_4 \\ & x_1 \geq 0,\ x_2 \geq 0. \end{aligned}$$

odnosno u matričnoj formi (1.1.4):

$$A = \begin{bmatrix} 2 & 4 & 1 & 0 \\ 4 & 2 & 0 & 1 \end{bmatrix} \quad b = \begin{bmatrix} 24 \\ 24 \end{bmatrix} \quad c = \begin{bmatrix} 3500 \\ 4800 \end{bmatrix}$$

1.2 Osobine skupa ograničenja

Neka je problem linearnog programiranja zadat u standardnom obliku (1.1.4). Skup $\Omega_P = \{x | Ax = b, x \geq 0\}$ na kome je definisana funkcija z bitno utiče na ekstremne vrednosti i ima interesantne geometrijske osobine. U nastavku pretpostavljamo da je $r = m < n$, gde su m i n dimenzije matrice A, a r je njen rang. Tada sistem ima beskonačno mnogo rešenja pa ima smisla tražiti ekstremnu vrednost funkcije $z(x)$ definisane na skupu Ω_P. Označimo kolone matrice A sa $K_1, \ldots, K_n$.

Primetimo da su vektori $K_1, \ldots, K_n$ i b dimenzije m. Podsetimo da su vektori $x_1, \ldots, x_n$ linearno nezavisni ako iz jednačine $\alpha_1 x_1 + \cdots + \alpha_n x_n = 0$ sledi $\alpha_1 = \cdots = \alpha_n = 0$, a u suprotnom su linearno zavisni. Izraz oblika $a = \lambda a^1 + (1-\lambda)a^2$, $0 \leq \lambda \leq 1$ naziva se konveksna kombinacija vektora a^1 i a^2. Uopštena konveksna kombinacija vektora $a^1, \ldots, a^k$ je svaki vektor a oblika

$$a = \sum_{i=1}^{k} \lambda_i a^i, \quad \sum_{i=1}^{k} \lambda_i = 1, \quad (\lambda_1, \ldots, \lambda_k \geq 0.)$$

Skup vektora K je konveksan ako

$$(\forall a^1, a^2 \in K)(\lambda a^1 + (1-\lambda)a^2 \in K, \quad 0 \leq \lambda \leq 1).$$

Geometrijski, skup K je konveksan ako za svake dve tačke $a^1, a^2 \in K$ važi da je segment određen tim tačkama sadržan u K. Najmanji konveksan skup koji u

prostoru $\mathbb{R}^n$ sadrži $n+1$ različitih tačaka naziva se *simpleks* u $\mathbb{R}^n$. U prostoru $\mathbb{R}^2$ svaki trougao je simpleks, a u $\mathbb{R}^3$ svaki tetraedar je simpleks. Navodimo neka poznata tvrđenja i osnovne definicije koje su neophodne za izučavanje simpleks metoda kao i metoda unutrašnjih tačaka.

Teorema 1.2.1 *Skup* $\Omega_P = \{x | Ax = b, x \geq 0\}$ *je konveksan.*

Dokaz. Neka je $x^1, x^2 \in \Omega_P$. Tada za njihovu konveksnu kombinaciju

$$x = \lambda x^1 + (1-\lambda)x^2, \quad 0 \leq \lambda \leq 1$$

važi

$$Ax = \lambda Ax^1 + (1-\lambda)Ax^2 = \lambda b + b - \lambda b = b.$$

Kako je očigledno $x \geq 0$, to je $x \in \Omega_P$. □

Definicija 1.2.1 *Moguće rešenje* x *je osnovno (bazično) ako su u jednačini*

$$b = x_1 K_1 + \cdots + x_n K_n$$

vektori K_i *za koje je* $x_i > 0$, *linearno nezavisni.*

Primetimo da bazično rešenje može da ima najviše m koordinata većih od nule.

Definicija 1.2.2 *Bazično rešenje za koje je tačno m koordinata veće od nule naziva se nedegenerisano. Bazično rešenje za koje je manje od m koordinata veće od nule naziva se degenerisano.*

Primetimo da od n kolona matrice A tj. od vektora $K_1, \ldots, K_n$ možemo formirati $\binom{n}{m}$ podmatrica sa m kolona. Ako su odabrani vektori $K_{i_1}, \ldots, K_{i_m}$ linearno nezavisni, tada je matrica $B = [K_{i_1} \cdots K_{i_m}]$ regularna i postoji B^{-1}. Tada je vektor $x = B^{-1}b$ jedinstveno određen i ako je $x \geq 0$, onda je x osnovno rešenje. Napomenimo da je $x \in \mathbb{R}^n$ a $B^{-1}b \in \mathbb{R}^m$, i da $x = B^{-1}b$ po definiciji znači da su koordinate x_i za $i \notin \{i_1, \ldots, i_m\}$ jednake nuli.

Definicija 1.2.3 *Matrica* $A_B = [K_{i_1} \cdots K_{i_m}]$ *je osnovna (bazična) ako je regularna. Preostale kolone matrice A formiraju nebazičnu matricu* A_N. *Promenljive* $x_{i_1}, \ldots x_{i_m}$ *nazivamo bazičnim dok preostale promenljive nazivamo nebazičnim . Dve bazične matrice su susedne ako se razlikuju u jednoj koloni.*

Neka je A_B bazična matrica. Sada sistem iz (1.1.4) možemo napisati kao:

$$A_B x_B + A_N x_N = b \Longrightarrow x_B = A_B^{-1} b - A_B^{-1} A_N x_N$$

Ako sada stavimo $x_N = 0$ dobijamo jedno bazično rešenje ($x = (x_B, x_N) = (B^{-1}b, 0)$, posle odgovarajuće prenumeracije promenljivih). Obrnuto, svako bazično rešenje određuje odgovarajuću bazičnu matricu (ukoliko je nedegenerisano, onda je ta matrica jedinstvena a u suprotnom nije). Ovim smo opravdali naziv "bazična matrica" u Definiciji 1.2.3.

Iz prethodne definicije sledi da je maksimalan broj osnovnih rešenja jednak $\binom{n}{m}$.

Definicija 1.2.4 *Tačka x je ekstremna tačka konveksnog skupa Ω_P ako ispunjava uslove*

$$x = \lambda x^{(1)} + (1-\lambda)x^{(2)} \wedge \lambda \in [0,1] \Leftrightarrow x = x^{(1)} = x^{(2)}.$$

Definicija 1.2.5 *Bazično rešenje x sistema $Ax = b$ za koje važi i uslov $x \geq 0$ je bazično dopustivo rešenje.*

Primer 1.2.1 U primeru (1.1.1) je

$$K_1 = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}, \quad K_2 = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix}, \quad K_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad K_4 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad K_5 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Kako je $m = 3$ i $n = 5$, bazičnih matrica i bazičnih rešenja ima najviše $\binom{5}{3} = 10$. Odredimo bazične matrice, bazična rešenja i vrednosti ciljne funkcije u njima:

$$\begin{array}{lll} B_1 = [K_3\, K_4\, K_5], & x^1 = (0,0,80,180,180), & z^1 = 0, \\ B_2 = [K_1\, K_3\, K_5], & x^2 = (60,0,20,0,120), & z^2 = -300, \\ B_3 = [K_1\, K_1\, K_5], & x^3 = (50,30,0,0,40), & z^3 = -370, \\ B_4 = [K_1\, K_2\, K_4], & x^4 = (30,50,0,40,0), & z^4 = -350, \\ B_5 = [K_2\, K_3\, K_4], & x^5 = (0,60,20,120,0), & z^1 = -240. \end{array}$$

Dakle, postoji pet osnovnih rešenja. Za ostale baze rešenja nisu dopustiva:

$$\begin{array}{lll} B_6 = [K_1\, K_2\, K_3], & x^6 = (45,45,-10,0,0), & z^6 = -405, \\ B_7 = [K_1\, K_4\, K_5], & x^7 = (80,0,0,-60,100), & z^7 = -400, \\ B_8 = [K_1\, K_3\, K_4], & x^8 = (180,0,-100,-360,0), & z^8 = -900, \\ B_9 = [K_2\, K_3\, K_5], & x^9 = (0,180,-100,0,-360), & z^9 = -720, \\ B_{10} = [K_2\, K_4\, K_5], & x^{10} = (0,80,0,100,-60), & z^{10} = -320. \end{array}$$

Vrlo važna činjenica je da egzistencija osnovnog rešenja sledi iz postojanja dopustivog rešenja što sledi iz sledeće teoreme.

Teorema 1.2.2 *Ako je $\Omega_P = \{x | Ax = b, x \geq 0\} \neq \emptyset$, tada on sadrži bar jedno osnovno rešenje.*

Dokaz. Neka je $x = (x_1, \ldots, x_n) \in \Omega_P$. Ako je potrebno prenumerišimo kolone K_i i koordinate vektora x tako da je $x_i > 0$ za $i \leq p$ i $x_i = 0$ za $i > p$. Sada je

$$b = \sum_{i=1}^{n} x_i K_i = \sum_{i=1}^{p} x_i K_i.$$

Ako su vektori $K_1, \ldots, K_p$ linearno nezavisni, onda je $p \leq m$ i x je osnovno rešenje. Ako su vektori $K_1, \ldots, K_p$ linearno zavisni, tada postoje $\lambda_1, \ldots, \lambda_p \in \mathbb{R}$ tako da je

$$\sum_{i=1}^{p} \lambda_i K_i = 0$$

i postoji $\lambda_i \neq 0$, $1 \leq i \leq p$. Neka je $\lambda_k \neq 0$ i $\lambda_k > 0$. Tada je

$$K_k = -\sum_{j \neq k} \frac{\lambda_j}{\lambda_k} K_j \quad \text{i} \quad b = \sum_{j \neq k} \left(x_j - x_k \frac{\lambda_j}{\lambda_k} \right) K_j.$$

Dakle, vektor b je prikazan kao zbir $p-1$ kolona matrice A. Ako je pri tome još i $x_j - x_k\frac{\lambda_j}{\lambda_k} \geq 0$, $j \neq k$, tada je vektor

$$x^1 = (x_1 - x_k\frac{\lambda_1}{\lambda_k}, \ldots, x_{k-1} - x_k\frac{\lambda_{k-1}}{\lambda_k}, 0, x_{k+1} - x_k\frac{\lambda_{k+1}}{\lambda_k}, 0, \ldots, 0, x_p - x_k\frac{\lambda_p}{\lambda_k}, 0, \ldots, 0)$$

element skupa Ω_P (tj. rešenje sistema $Ax = b$). Ako je $\lambda_j \leq 0$, tada je očigledno i $x_j^1 \geq 0$. U suprotnom je $x_j^1 = x_j - x_k\frac{\lambda_j}{\lambda_k} \geq 0$, što je ekvivalentno sa $\frac{x_j}{\lambda_j} \geq \frac{x_k}{\lambda_k}$. Prema tome i u ovom slučaju važi $x_j^1 \geq 0$, pa zaključujemo da je $x_1 \in \Omega_P$. Međutim, x^1 ima najviše $p-1$ strogo pozitivnih koordinata, što je kontradikcija. Prema tome, vektori $K_1, \ldots, K_p$ su linearno nezavisni, i x je bazično dopustivo rešenje. □

Vezu između bazičnih rešenja i temena skupa Ω_P daje sledeća teorema.

Teorema 1.2.3 *Ako je $x \in \Omega_P$ bazično rešenje sistema $Ax = b, x \geq 0$, tada je x ekstremna tačka skupa Ω_P. Obratno, ako je x ekstremna tačka skupa Ω_P, tada je x bazično rešenje.*

Dokaz. Neka je x bazično rešenje i neka je novom numeracijom (ako je potrebno)

$$b = \sum_{j=1}^{m} x_j K_j, \quad x = (x_1, \ldots, x_m, 0, \ldots, 0).$$

Pretpostavimo da x nije ekstremna tačka skupa Ω_P, tj. postoje $x^1, x^2 \in \Omega_P$, $x^1 = (x_1^1, \ldots, x_n^1), x^2 = (x_1^2, \ldots, x_n^2)$, tako da je

$$x = \lambda x^1 + (1-\lambda)x^2, \quad 0 < \lambda < 1.$$

Kako je $x_i = \lambda x_i^1 + (1-\lambda)x_i^2$, $i = 1, \ldots, n$, to je $x_i^1 = x_i^2 = 0$ za $i > m$. Dakle, x^1 i x^2 su bazična rešenja za bazu $K_1, \ldots, K_m$, tj.

$$b = \sum_{j=1}^{m} x_j^1 K_j = \sum_{j=1}^{m} x_j^2 K_j = \sum_{j=1}^{m} x_j K_j.$$

Kako su $K_1, \ldots, K_m$ linearno nezavisni, to je $x = x^1 = x^2$ što znači da je x ekstremna tačka skupa Ω_P.

Dokažimo obratno tvrđenje. Neka je $x \in \Omega_P$ ekstremna tačka skupa Ω_P. Novom numeracijom (ako je potrebno) postižemo da je $x_i > 0$ za $i \leq p$ i $x_i = 0$ za $i \geq p$. Pretpostavimo da su odgovarajući vektori $K_1, \ldots, K_m$ iz $b = \sum_{j=1}^{p} x_j K_j$, linearno zavisni, tj. da postoji bar jedno $\lambda_j > 0$, $1 \leq j \leq p$, tako da važi

$$\sum_{j=1}^{p} \lambda_j K_j = 0.$$

Sada je

$$b = \sum_{j=1}^{p} x_j K_j \pm \sum_{j=1}^{p} \mu\lambda_j K_j = \sum_{j=1}^{p} (x_j \pm \mu\lambda_j) K_j$$

za svako $\mu \in \mathbb{R}$. Ako je $\mu = \frac{1}{2}\min\left\{\frac{x_j}{\lambda_j} | \, 1 \leq j \leq p\right\}$, tada je $x_j \pm \mu\lambda_j > 0$, $1 \leq j \leq p$, pa je

$$\begin{aligned} x^1 &= (x_1 + \mu\lambda_1, \ldots, x_p + \mu\lambda_p, 0, \ldots, 0) \in \Omega_P, \\ x^2 &= (x_1 - \mu\lambda_1, \ldots, x_p - \mu\lambda_p, 0, \ldots, 0) \in \Omega_P, \end{aligned}$$

i važi

$$x = \frac{1}{2}x^1 + \frac{1}{2}x^2,$$

što je nemoguće jer je x ekstremna tačka skupa Ω_P. □

Iz prethodnog sledi da je broj ekstremnih tačaka manji od $\binom{n}{m}$.

Posledica 1.2.1 *Skup Ω_P ima konačno mnogo ekstremnih tačaka.*

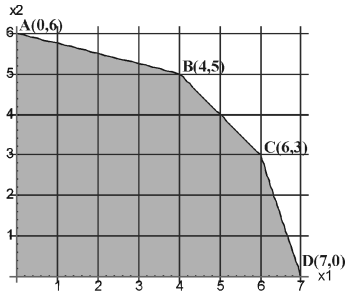
Dokaz. Svaka bazična matrica B određuje tačno jedno bazično rešenje (na osnovu Teoreme 1.2.3), i za svako bazično rešenje može naći odgovarajuća matrica B tako da važi Teorema 1.2.3. Zaključujemo da bazičnih matrica ima ne manje nego bazičnih rešenja, a na osnovu prethodne teoreme ekstemnih tačaka skupa Ω_P. Očigledno, bazičnih matrica ima ne više od $\binom{n}{m}$. □

Značaj prethodne teoreme i posledice je u tome što se broj potencijalnih ekstremuma funkcije $z(x)$ redukuje sa (u opštem slučaju) beskonačnog skupa Ω_P na konačan skup temena koji ima maksimalno $\binom{n}{m}$ elemenata. Ciljna funkcija $z(x)$ dostiže maksimum ili minimum u temenima konveksnog skupa Ω_p. Formalno, dovoljno je izračunati vrednosti ciljne funkcije u svim ekstremnim tačkama skupa Ω_P i odrediti onu tačku, ili tačke, u kojima je vrednost funkcije $z(x)$ ekstremna. Ovaj broj potencijalnih ekstremuma veoma brzo raste sa povećanjem vrednosti m i n.

Primer 1.2.2 Posmatramo sledeći problem

$$\begin{aligned} \max \quad & z(x) = 2x_1 + 5x_2 \\ \text{p.o.} \quad & x_1 + 4x_2 \leq 24, \; 3x_1 + x_2 \leq 21, \; x_1 + x_2 \leq 9. \end{aligned}$$

Oblast Ω_p sa ekstremnim tačkama A, B, C, D prikazana je na sledećoj slici.



Slika 1.2.1. Oblast dopustivih rešenja zadatog problema nacrtana u programu MATHEMATICA.

Posle izračunavanja vrednosti $z(A), z(B), z(C), z(D)$ možese uočiti da najveću vrednost ima $z(B) = z(4,5) = 33$, što i predstavlja rešenje problema.

Dokazaćemo kasnije u Teoremi 2.1.1 da se ekstemum funkcije cilja $z(x)$ (ako postoji) nalazi upravo u ekstremnim tačkama skupa Ω_P, a na osnovu Teoreme 1.2.3 u bazično dopustivim rešenjima sistema $Ax = b$.

Sledeća teorema daje kriterijum za detekciju neograničenosti skupa Ω_P.

Teorema 1.2.4 *Ako postoji bazična matrica $A_B = [K_{i_1} \cdots K_{i_m}]$ i kolona K_p matrice A tako da je $A_B^{-1}K_p \leq 0$, tada je Ω_P neograničen skup.*

Dokaz. Neka je $A_B = [K_1 \dots K_m]$ i $B^{-1}K_p \leq 0$ za neko p. Tada je

$$K_p = \lambda_1 K_1 + \cdots + \lambda_m K_m$$

i važi $\lambda_j \leq 0$, $1 \leq j \leq m$. Sada iz

$$b = \sum_{j=1}^{m} x_j K_j - \mu K_p + \mu K_p, \quad \mu > 0,$$

sledi

$$b = \sum_{j=1}^{m} (x_j - \mu\lambda_j) K_j + \mu K_p, \quad \mu > 0,$$

odakle je

$$x^\mu = (x_1 - \mu\lambda_1, \dots, x_m - \mu\lambda_m, 0, \dots, 0, \overset{(p)}{\rightarrow} \mu, 0, \dots, 0) \in \Omega_P$$

za svako $\mu > 0$, tj. Ω_P je neograničen skup. □

Ako je za svaku osnovnu matricu B i svaki vektor K_p vektor $B^{-1}K_p$ nenegativan, tada je skup Ω_P ograničen i svako $x \in \Omega_P$ je konveksna kombinacija osnovnih (bazičnih) rešenja. Znači, dovoljno je znati samo bazična rešenja. Ako je jedno bazično rešenje poznato, možemo odrediti drugo bazično rešenje. Neka je $B = [K_1 \cdots K_m]$ osnovna matrica i neka je $x = B^{-1}b = (x_1, \dots, x_m, 0, \dots, 0)$ poznato osnovno rešenje. Ako za neko K_r važi $B^{-1}K_r \leq 0$, skup Ω_P je neograničen pa zato pretpostavimo da je Ω_P ograničen, tj. $B^{-1}K_r$ nenegativno. Neka je

$$B^{-1}K_r = (\lambda_1, \dots, \lambda_m, 0, \dots, 0)$$

nenegativno i neka je $\lambda_k > 0$ za neko $k, 1 \leq k \leq m$. Tada iz

$$K_r = \lambda_1 K_1 + \cdots + \lambda_m K_m$$

sledi

$$K_k = \frac{1}{\lambda_k} K_r - \sum_{j \neq k} \frac{\lambda_j}{\lambda_k} K_j. \tag{1.2.1}$$

Zamenom (1.2.1) u

$$b = x_1 K_1 + \cdots + x_k K_k + \cdots + x_m K_m$$

dobijamo

$$b = x_1K_1 + \cdots + \frac{x_k}{\lambda_k}K_r - \sum_{j\neq k} x_k\frac{\lambda_j}{\lambda_k}K_j + \cdots + x_mK_m$$

to jest

$$b = \sum_{j\neq k}(x_j - x_k\frac{\lambda_j}{\lambda_k})K_j + \frac{x_k}{\lambda_k}K_r.$$

Odgovarajuće rešenje

$$x^1 = (x_1 - x_k\frac{\lambda_1}{\lambda_k}, \ldots, \overset{(k)}{\rightarrow} 0, \ldots, x_m - x_k\frac{\lambda_m}{\lambda_k}, 0, \ldots, 0, \overset{(r)}{\rightarrow} \frac{x_k}{\lambda_k}, 0, \ldots, 0)$$

je u skupu Ω_P ako je $x_j - x_k\frac{\lambda_j}{\lambda_k} \geq 0, j \neq k$, što je ispunjeno ako k izaberemo tako da je

$$\frac{x_k}{\lambda_k} = \min_j \left\{ \frac{x_j}{\lambda_j} |\ \lambda_j > 0 \right\}.$$

Bazično rešenje x^1 je različito od x ako je $x_k \neq 0$ što je ispunjeno ako je x nedegenerisano rešenje. Odgovarajuća osnovna matrica dobija se tako što kolonu K_k zamenimo kolonom K_r.

1.3 Geometrijski metod

Svakom uslovu nenegativnosti odgovara u vektorskom prostoru $\mathbb{R}^n$ poluprostor u kome je odgovarajuća promenljiva nenegativna. Svakoj uslovnoj jednačini u $\mathbb{R}^n$ odgovara jedna hiperravan. Svakoj uslovnoj nejednačini odgovara poluprostor omeđen hiperravni pridruženoj odgovarajućoj jednačini. Skup svih dopustivih vektora x je presek svih datih poluprostora i datih hiperravni, dakle čini jedan konveksni poliedar Ω_P. Jednačina $c^Tx = k$ za neko k predstavlja hiperravan paralelnu sa prostorom $\mathbb{R}^{n-1}$ koji je normalan na c. Projekcija poliedra Ω_P na pravac određen vektorom c je zatvoren skup $[l, L]$ realnih brojeva, gde je l minimum a L maksimum ciljne funkcije (1.1.2). Odgovarajuće hiperravni normalne na c su dodirne hiperravni poliedra Ω_P. Zajedničke tačke tih dodirnih hiperravni sa poliedrom Ω_P daju vrednosti u kojima funkcija (1.1.2) dostiže ekstremnu vrednost.

Geometrijski metod se može iskoristiti kod problema koji sadrže $n = 2$ promenljive, a najviše $n = 3$ promenljive. Zadatak linearnog programiranja dat u osnovnom obliku koji ispunjava uslov $n - m = 2$ (a najviše $n - m = 3$) takođe se može rešavati geometrijskim metodom. Geometrijski metod, iako ne baš pristupačan, koristi se zato što olakšva pristup opštoj algebarskoj metodi.

Neka je dat linearni problem u obliku

$$\begin{aligned} \max\ & z(x) = c_1x_1 + \ldots + c_nx_n \\ \text{p.o.}\ & a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leq b_1 \\ & \cdots\cdots\cdots \\ & a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leq b_m. \end{aligned}$$

Za dati sistem znamo da je svako rešenje sistema nejednačina jedna tačka prostora $\mathbb{R}^n$, a skup nenegativnih dopustivih rešenja Ω_p je podskup prostora $\mathbb{R}^n$. Svaka od nejednačina

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i,\ i = 1, 2, ..., m$$

određuje podskup $D_i \subset \mathbb{R}^n$, $i = 1, \ldots, m$ koji predstavlja skup tačaka s jedne strane hiperravni

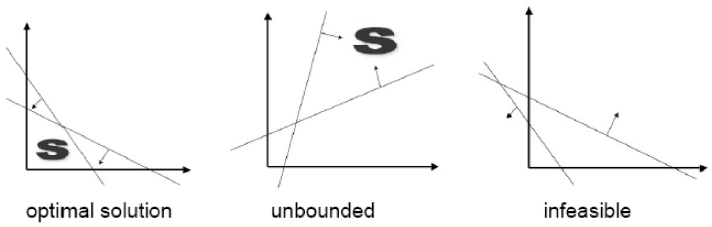
$$\sum_{j=1}^{n} a_{ij}x_j = b_i,$$

pa je oblast dopustivih rešenja (poliedar u $\mathbb{R}^n$) određena presekom skupova

$$\Omega_p = D_1 \cap D_2 \cap \cdots \cap D_m \cap D_{m+1} \cap \cdots \cap D_{m+n},$$

gde se podskupovi $D_{m+1}, ..., D_{m+n}$ dobijaju iz uslova nenegativnosti promenjivih $x_1 \geq 0, \ldots, x_n \geq 0$. Skup dopustivih rešenja geometrijski predstavlja poliedar (simplicijalni kompleks).

Različite mogućnosti pri rešavanju problema linearnog programiranja prikazane su na slici 1.3.1.



Slika 1.3.1. Različite mogućnosti pri rešavanju problema linearnog programiranja.

Skup tačaka u kojima funkcija cilja $z(x)$ ima vrednost d predstavlja takođe jednu hiperravan $\mathcal{H}_{z,d} = \{x \in \mathbb{R}^n \mid z(x) = d\}$, koju u zavisnosti od vrednosti d možemo translirati u pravcu vektora c. Sada se problem linearnog programiranja svodi na nalaženje maksimalne (minimalne) vrednosti za d tako da je $\mathcal{H}_{z,d} \cap \Omega_P \neq \emptyset$. Upravo na ovoj činjenici se zasniva geometrijski metod. Sada je jasno zašto je ovaj metod primenljiv samo u slučajevima $n = 2$ i $n = 3$. Primetimo još da se, na osnovu Teoreme 2.1.1, ekstremum funkcije cilja dostiže u ekstremnoj tački skupa Ω_P. Skup optimalnih tačaka Ω_P^* iz iste teoreme je konveksan i predstavlja k-dimenzionalni poliedar. Ciljna funkcija, koja se za $z(x) = d$ interpretira kao hiperravan, dostiže maksimum (minimum) u jednom temenu poliedra (slučaj jedinstvenog optimalnog rešenja) ili po jednoj strani poliedra ako je hiperravan $z(x) = d$ njoj paralelana (slučaj beskonačno mnogo optimalnih rešenja).

Geometrijski metod ilustrujemo na sledećem primeru, a ujedno izvlačimo zaključke koji će biti od značaja za opšti algebarski metod.

Primer 1.3.1 Fabrika proizvodi dve vrste artikala A_1 i A_2, i to na mašinama M_1 i M_2. Za artikal A_1 mašina M_1 radi 2^h, a mašina M_2 radi 4^h, dok za vrstu A_2 mašina M_1 radi 4^h, a mašina M_2

radi 2^h. Fabrika dobija 3500 dinara po jedinici proizvoda A_1, a 4800 dinara po jedinici proizvoda A_2. Koliko treba proizvoditi artikala A_1 i A_2 i kako iskoristiti rad mašina M_1 i M_2 da dnevna dobit fabrike bude maksimalna?

Rešenje: Neka je x_1 broj proizvedenih artikala A_1, a x_2 broj proizvedenih artikala A_2 u toku dana. Tada je dnevna dobit fabrike:

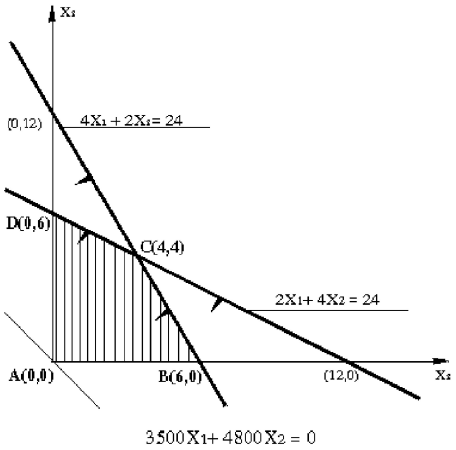
$$z(x) = 3500x_1 + 4800x_2$$

uz uslove:

$$\begin{aligned} 2x_1 + 4x_2 &\leq 24 \\ 4x_1 + 2x_2 &\leq 24 \\ x_1 \geq 0, x_2 &\geq 0. \end{aligned}$$

Ovaj model se sastoji od funkcije $z(x)$ i ograničenja (sistema nejednačina ili jednačina), i naziva se matematički model. Dnevna dobit fabrike je funkcija $z(x)$ koja zavisi od dve promenljive x_1 i x_2 i taj oblik zavisnosti je linearan. Zaista, vrednost jedinice A_1 na tržištu, tj. 3500 se množi sa brojem proizvedenih jedinica x_1 te vrste proizvoda, i tome se dodaje vrednost jedinice proizvoda A_2 na tržištu, tj. 4800 pomnožene sa brojem x_2 proizvedenih jediničnih artikala A_2. Ograničenja proizilaze iz činjenice da M_1 proizvodi jedinicu proizvoda A_1 za 2^h, a M_2 tu istu jedinicu za 4^h, dok dok se proizvod A_2 proizvodi na mašini M_1 za 4^h, a na M_2 za 2^h, kao i da mašine u idealnom stepenu iskorišćenja rade najviše 24^h. Otuda slede linearne nejednačine u navedenom matematičkom modelu, gde se prva odnosi na mogućnosti mašine M_1, a druga na mogućnosti mašine M_2. Ovim nejednakostima dodaje se priroda traženih promenjivih x_1 i x_2, koja se sastoji u tome da budu nenegativne veličine. Skup rešenja Ω_p je konveksan skup, jer je presek konačnog broja konveksnih skupova. Funkcija $z(x)$ dostiže maksimu ili minimum u temenima konveksnog skupa Ω_p.

Ako se ograničenja grafički predstave u koordinatnom sistemu x_1x_2 dobija se jedan četvorougao $ABCD$ koji je konveksan, u čijoj oblasti se nalaze moguća rešenja datog linearnog modela. Unutar četvorougla, kao i na tačkama ruba, (duž AB, BC, CD i DA) nalaze se moguća rešenja datog modela. Međutim, uočava se da je teme $C(4,4)$ sa koordinatama $x_1 = 4$ i $x_2 = 4$ takvo da odgovara optimalnom rešenju datog modela, jer je najudaljenije od prave $3500x_1 + 4800x_2 = 0$ dobijene iz funkcije cilja $z(x)$ za $z(x) = 0$. Zaista, odgovarajuća vrednost funkcije kriterijuma je max $z(x) = 3500 * 4 + 4800 * 4 = 14000 + 19200 = 33200$ dinara dnevne dobiti, koja je najveća moguća u datim uslovima.



Slika 1.3.2. Grafički prikaz ograničenja.

Primer 1.3.2 Dnevna količina stočne hrane treba da sadrži najmanje $0.4kg$ komponente A, $0.5kg$ komponente B, $2kg$ komponente C i $1.8kg$ komponente D. Na tržištu se mogu naći dve vrste

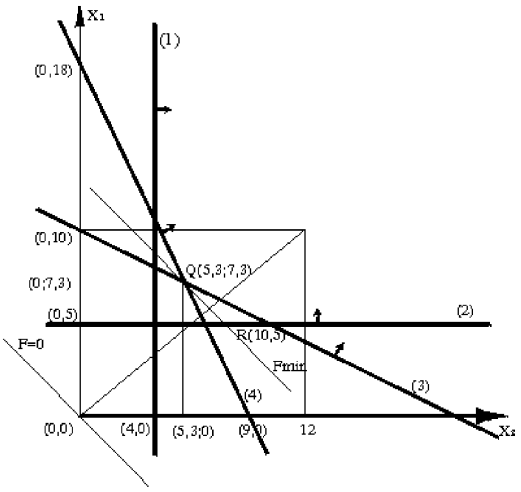
mešavina stočne hrane M_1 i M_2, pri čemu ove mešavine sadrže komponente A, B, C i D u sledećim količinama:

komponente	M_1	M_2	propisane količine za komponente
A	0.1	0	0.4
B	0	0.1	0.5
C	0.1	0.2	2.0
D	0.2	0.1	1.8
cena mešavina po kg	60	50	-

Koliku količinu svake od mešavina M_1 i M_2 treba kupiti da dnevni troškovi budu minimalni?

Rešenje: Potrebno je da se reši sledeći linearni program:

$$\begin{aligned} \min \quad & z(x) = 60x_1 + 50x_2 \\ \text{p.o.} \quad & 0.1x_1 \geq 0.4 \\ & 0.1x_2 \geq 0.5 \\ & 0.1x_1 + 0.2x_2 \geq 2.0 \\ & 0.2x_1 + 0.1x_2 \geq 1.8 \\ & x_1 \geq 0,\ x_2 \geq 0. \end{aligned}$$



Slika 1.3.3. Grafički prikaz ograničenja.

Optimalno rešenje problema je tačka $Q(5.3, 7.3)$.

Primer 1.3.3 Dva rudnika R_1 i R_2 snabdevaju ugljem tri grada A, B i C. Rudnik R_1 može dnevno da isporučuje 500 tona uglja, a rudnik R_2 800 tona. Troškovi prevoza jedne tone uglja u novčanim jedinicama od rudnika do gradova prikazani su u tabeli. Kako organizovati prevoz da ukupni troškovi prevoza budu najmanji?

Rudnici	A	B	C
R_1	8	5	5
R_2	4	6	8

Rešenje: Označimo sa x_1 broj tona uglja koji treba prevesti iz rudnika R_1 u grad A, a sa x_2 broj tona uglja iz R_1 u B. Uzimajući u obzir da se u gradovima potražuje dnevno isto onoliko

uglja koliko se dnevno može dobiti iz rudnika, možemo ostale količine uglja da izrazimo pomoću x_1 i x_2:

$$\begin{aligned} R_1 &\rightarrow C : 500 - x_1 - x_2 \\ R_2 &\rightarrow A : 500 - x_1 \\ R_2 &\rightarrow B : 400 - x_2 \\ R_2 &\rightarrow C : 800 - 500 + x_1 - 400 + x_2 = x_1 + x_2 - 100. \end{aligned}$$

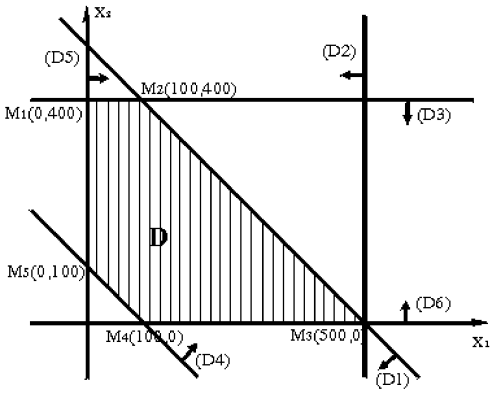
Ograničenja i uslovi nenegativnosti glase:

$$\begin{aligned} &(D_1) \quad 500 - x_1 - x_2 \geq 0 \ \Rightarrow \ x_1 + x_2 \leq 500 \\ &(D_2) \quad 500 - x_1 \geq 0 \ \Rightarrow \ x_1 \leq 500 \\ &(D_3) \quad 400 - x_2 \geq 0 \ \Rightarrow \ x_2 \leq 400 \\ &(D_4) \quad x_1 + x_2 - 100 \geq 0 \ \Rightarrow \ x_1 + x_2 \geq 100 \\ &(D_5) \quad x_1 \geq 0 \\ &(D_6) \quad x_2 \geq 0. \end{aligned}$$

Oblasti $D_1, D_2, \ldots, D_6$ su označene strelicama na datoj slici, a oblast dopustivih rešenja

$$D = \bigcap_{i=1}^{6} D_i$$

šrafirana je na slici.



Slika 1.3.4. Grafički prikaz ograničenja.

Ukupni troškovi prevoza jednaki su:

$$\begin{aligned} z =& 8x_1 + 5x_2 + 5(500 - x_1 - x_2) + 4(500 - x_1) + 6(400 - x_2) + 8(x_1 + x_2 - 100) \\ =& 7x_1 + 2x_2 + 6500. \end{aligned}$$

Veličine x_1 i x_2 treba sada odrediti tako da ukupni troškovi budu minimalni. Prava $7x_1 + 2x_2 + 6500 = \text{const}$ uzima najmanju vrednost u tačkama oblasti dopustivih vrednosti (D), koje su najbliže koordinatnom početku. Najjednostavnije je zbog toga izračunati vrednost funkcije cilja u tačkama oblasti dopustivih vrednosti (D), koje su najbliže koordinatnom početku. Najjednostavnije je zbog toga izračunati vrednost funkcije cilja u tačkama $M_4(100, 0)$ i $M_5(0, 100)$:

$$\begin{aligned} z(M_4) &= 7 * 100 + 2 * 0 + 6500 = 7200 \\ z(M_5) &= 7 * 0 + 2 * 100 + 6500 = 6700 = z_{min}. \end{aligned}$$

Na taj način minimalni troškovi prevoza od rudnika R_1 i R_2 do gradova A, B i C obezbeđeni su ako je plan prevoza sledeći $(x_1 = 0, x_2 = 100)$:

Rudnici	A	B	C
R_1 (500)	0	100	400
R_2 (800)	500	300	0

U ovoj tabeli je, dakle prikazan optimalan plan prevoza uglja. Ovakav zadatak je poznat pod nazivom transportni zadatak. Kao što vidimo, u slučaju kada imamo dve otpremne stanice (rudnici) i tri prijemne stanice (gradovi), transportni zadatak može da se reši geometrijski, jer se broj nepoznatih svodi na dve zahvaljujući uslovu da je količina ponuđenog uglja od strane rudnika R_1 i R_2 jednaka količini traženog uglja u gradovima A, B i C.

Primer 1.3.4 Rešiti problem linearnog programiranja grafičkim metodom

$$\begin{aligned} \max \quad & z(x) = 2x_1 + 3x_2 \\ \text{p.o.} \quad & x_1 + x_2 \leq 5 \\ & x_1 - 3x_2 \geq 0 \\ & 2x_1 + 3x_2 \geq 6 \\ & x_i \geq 0, \quad i = 1, 2. \end{aligned}$$

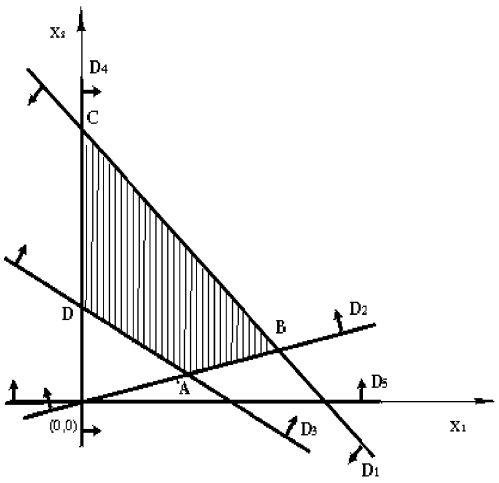
Označimo ograničenja i uslove negativnosti respektivno sa D_1, D_2, D_3, D_4, D_5:

$$\begin{aligned} (D_1) \quad & x_1 + x_2 \leq 5 \\ (D_2) \quad & x_1 - 3x_2 \geq 0 \\ (D_3) \quad & 2x_1 + 3x_2 \geq 6 \\ (D_4) \quad & x_1 \geq 0 \\ (D_5) \quad & x_2 \geq 0. \end{aligned}$$

Svako od ograničenja i uslova nenegativnosti predstavlja poluravan u prostoru $\mathbb{R}^2$. Jasno sva dopustiva rešenja će se naći u podskupu

$$D = D_1 \cap D_2 \cap D_3 \cap D_4 \cap D_5.$$

Prikažimo geometrijski oblast dopustivih rešenja određenu ravnima $D_i (i = 1, 2, ..., 6)$:



Slika 1.3.5. Grafički prikaz ograničenja.

Šrafirana površ predstavlja oblast dopustivih rešenja. Optimalno rešenje će biti u nekoj od tačaka A, B, C, D. Da bismo utvrdili koja tačka predstavlja optimalno rešenje, treba najpre nacrtati pravu

koja predstavlja funkciju cilja i to za $z = 0$. Ovako nacrtanu pravu treba paralelno pomerati u pravcu rasta nepoznatih X_1 i X_2, sve dok ne stignemo do poslednje tačke šrafirane površi, odnosno do poslednje tačke poligona dopustivih rešenja. U toj tački će ciljna funkcija imati maksimalnu vrednost tj. ta tačka će predstavljati optimalno rešenje. U našem primeru to će očigledno biti tačka B čije koordinate treba izračunati i uvrstiti u funkciju cilja. Iz sistema

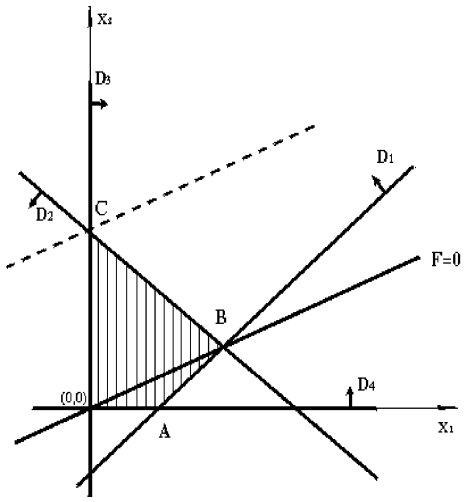
$$x_1 + x_2 = 5$$
$$x_1 - 3x_2 = 0$$

dobijamo $B(15/4, 5/4)$ i

$$z_{max} = z(B) = z(15/4, 5/4) = 2 * 15/4 + 3 * 5/4 = 45/4.$$

Primer 1.3.5 Naći minimum funkcije $z(x) = x_1 - 2x_2$ uz ograničenja

$$(D_1) \quad x_1 - x_2 \leq 1$$
$$(D_2) \quad x_1 + x_2 \leq 3$$
$$(D_3) \quad x_1 \geq 0$$
$$(D_4) \quad x_2 \geq 0$$



Slika 1.3.5. $z_{min} = z(C) = z(0, 3) = -6$.

Pravac pomeranja je u pravcu rašćenja x_2 jer se njenim povećanjem smanjuje vrednost funkcije cilja.

Sada možemo izvesti sledeće zaključke u vezi geometrijskog metoda:

I Ako je oblast dopustivih rešenja D konačna onda na njenoj granici postoji najmanje jedna tačka koja predstavlja optimalno rešenje.

Dokaz sledi iz teoreme Weierstrassa koja govori da neprekidna funkcija definisana na zatvorenom i ograničenom skupu dostiže bar u jednoj tački tog skupa najveću vrednost, i bar u jednoj tački najmanju vrednost. Kako je ciljna funkcija linearna i neprekidna to ova teorema važi i kad je u pitanju konveksna i ograničena oblast.

II U zadatku linearnog programiranja ne postoji optimalno rešenje kada je skup dopustivih rešenja D prazan ili kada je D neograničen i istovremeno na njemu ciljna funkcija, koja se minimizira (maksimizira), beskonačno opada (raste).

III Ako su dobijena dva optimalna rešenja onda su sve tačke duži između tih tačaka, optimalna rešenja. Dakle, ako su $x^{(1)} \in D$ i $x^{(2)} \in D$ optimalna rešenja, onda je svaki vektor oblika

$$x_\lambda^{(0)} = \lambda x^{(2)} + (1-\lambda)x^{(1)}, \quad 0 \leq \lambda \leq 1$$

optimalno rešenje.

U slučaju $n = 2$, ako se ograničenja grafički predstave u koordinatnom sistemu x_1x_2 dobija se konveksan poligon, na čijim se temenima (ekstremnim tačkama) nalaze moguća rešenja datog problema linearnog programiranja. Teme najudaljenije od prave određene funkcijom cilja predstavlja optimalno rešenje datog problema.

Geometrijski metod, za slučaj $n = 2$ smo implementirali u programskom jeziku MATHEMATICA [90]. Tako je nastao program GEOM [80], koji za unet problem linearnog programiranja u simetričnom obliku od dve promenljive pronalazi optimalno rešenje geometrijskim metodom i pri tome grafički prikazuje sve međukorake. Za grafičko prikazivanje skupa Ω_P dopustivih rešenja, koristili smo sledeće standardne funkcije programskog jezika MATHEMATICA: `InequalityPlot`, `InequalitySolve`, `FindInstance`, itd. Ove funkcije se nalaze u standardim paketima `Graphics‘InequalityGraphics` i `Algebra’InequalitySolve`. Kompletan kôd programa GEOM kao i detalji implementacije prikazani su u dodatku, a mogu se naći i u našem radu [80]. Razmotrimo sada rad programa GEOM na sledećem primeru:

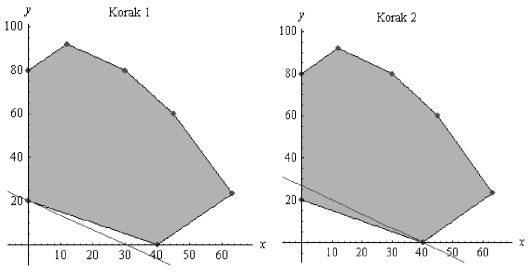
Primer 1.3.6 Rešiti problem linearnog programiranja:

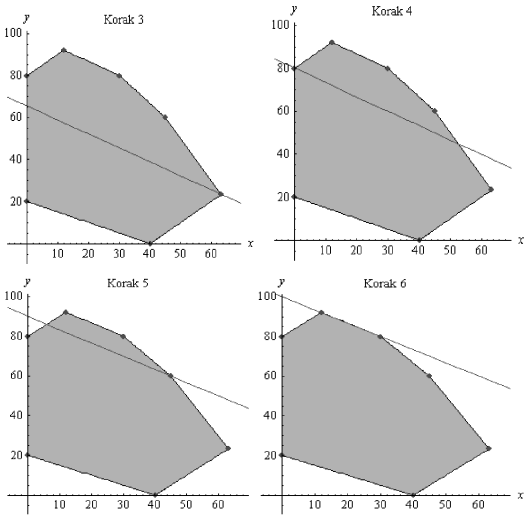
$$\begin{aligned} \max \quad & z(x,y) = 8x + 12y \\ \text{p.o.} \quad & 8x + 4y \leq 600 \\ & 2x + 3y \leq 300 \\ & 4x + 3y \leq 360 \\ & 5x + 10y \geq 600 \\ & x - y \geq -80 \\ & x - y \leq 40 \\ & x, y \geq 0 \end{aligned}$$

Problem rešavamo sledećom naredbom:

```
Geom[8x+12y, {8x+4y<=600, 2x+3y<=300, 4x+3y<=360, 5x+10y>=600,x>=0, y>=0}]
```

Program daje grafički prikaz skupa dopustivih rešenja Ω_P, kao i prave koja odgovara ciljnoj funkciji ($\mathcal{H}_{z,d}$). Pravu pomeramo "naviše", u pravcu vektora $c = \begin{bmatrix} 8 \\ 12 \end{bmatrix}$ sve dok postoji presek sa poligonom dopustivih rešenja. Na slici su prikazani položaji prave $\mathcal{H}_{z,d}$ kada prolazi kroz temena poligona (ekstremne tačke):





Slika 1.3.6. Vizuelizacija geometrijskog metoda.

Program takođe daje optimalno rešenje (ili izraz koji opisuje sva optimalna rešenja). U ovom slučaju to je:

$$x^* = \lambda \begin{bmatrix} \frac{190}{3} \\ \frac{70}{3} \end{bmatrix} + (1-\lambda) \begin{bmatrix} 45 \\ 60 \end{bmatrix},\ 0 \leq \lambda \leq 1$$

2 Simpleks metod

Simpleks metod se zasniva na tri bitna principa:

1. Postoji mogućnost određivanja bar jednog dopustivog rešenja (plana), koji se često naziva bazičnim planom ili dopustivim bazičnim planom.

2. Postoji mogućnost da se proveri da li je bazični dopustivi plan optimalan ili ne.

3. Postoji mogućnost, da se u slučaju da dopustivi plan nije optimalan, izabere novi, koji je bliži optimalnom.

Prema gore navedenom, simpleks metod se zasniva na sukcesivnom poboljšanju početnog dopustivog plana, sve dok se ne dobije optimalan plan. Algoritam simpleks metoda takođe omogućava da se ustanovi da li je zadatak rešiv ili ne, odnosno, da li postoji protivurečnost u ograničenjima.

2.1 Osobine simpleks metoda

Primetimo prvo da vektor x^* u kome ciljna funkcija $z(x)$ dostiže ekstremnu vrednost ne mora biti jedinstven. Sledeća teorema pokazuje da ciljna funkcija dostiže ekstremnu vrednost u nekoj od ekstremnih tačaka skupa Ω_P.

Teorema 2.1.1 *Ako je* $\Omega_P = \{x : Ax = b, x \geq 0\}$ *ograničen skup i* $z(x) = c_1x_1 + \cdots + c_nx_n$ *zadata linearna funkcija, tada postoji bar jedna ekstremna tačka* $x^* \in \Omega_p$

takva da je

$$\inf_{x\in\Omega_P} z(x) = z(x^*).$$

Skup $\{x|\, x \in \Omega_P,\, z(x) = z(x^*)\}$ *je konveksan.*

Dokaz. Neka su $x^1, \ldots, x^p$ ekstremne tačke skupa Ω_P i neka je x^* ekstremna tačka za koju je $z(x^i) \geq z(x^*)$, $i = 1, \ldots, p$. Kako je svako $x \in \Omega_P$ konveksna kombinacija ekstremnih tačaka, to postoje pozitivni skalari $\lambda_1, \ldots, \lambda_p$ takvi da je

$$x = \sum_{k=1}^{p} \lambda_k x^k, \quad \sum_{k=1}^{p} \lambda_k = 1.$$

Sada je

$$z(x) = z\left(\sum_{k=1}^{p} \lambda_k x^k\right) = \sum_{k=1}^{p} \lambda_k z(x^k) \geq \sum_{k=1}^{p} \lambda_k z(x^*) = z(x^*),$$

što dokazuje da z dostiže minimum u x^*.

Neka je $z(x^1) = z(x^2) = z(x^*)$. Tada je

$$z(\lambda x^1 + (1-\lambda)x^2) = \lambda z(x^1) + (1-\lambda) z(x^2) = z(x^*)$$

za svako $0 \leq \lambda \leq 1$. □

Značaj prethodne teoreme je u tome što se broj potencijalnih ekstremuma funkcije $z(x)$ redukuje sa (u opštem slučaju) beskonačnog skupa Ω_P na konačan skup temena koji ima maksimalno $\binom{n}{m}$ elemenata. Formalno, dovoljno je izračunati vrednosti ciljne funkcije u svim ekstremnim tačkama skupa Ω_P i odrediti onu tačku, ili tačke, u kojima je vrednost funkcije $z(x)$ ekstremna.

Neka su $j_1 < \cdots < j_m$ indeksi takvi da je $\mathcal{A}_B = \{K_{j_1}, \ldots, K_{j_m}\}$ jedna baza matrice A i neka je $B = \{j_1, \ldots, j_m\}$. Ako se drugačije ne naglasi, podrazumevamo da se bazi $\mathcal{A}_B$ pridružuje bazična podmatrica $A_B = [K_{j_1} \ldots K_{j_m}]$. Ukoliko je bazično rešenje dopustivo, bazu $\mathcal{A}_B$ nazivamo *dopustivom bazom.* Neka je

$$N = \{i_1, \ldots, i_{n-m}\} = \{1, \ldots, n\} \backslash B$$

rastuće uređen skup i neka je $A_N = \{K_{i_1}, \ldots, K_{i_{n-m}}\}$. Sada se sistem ograničenja $Ax = b$ može zapisati u obliku

$$A_B x_B + A_N x_N = b, \tag{2.1.1}$$

gde je x_B vektor bazičnih, a x_N vektor nebazičnih promenljivih. Sada iz (2.1.1) sledi $x_B = A_B^{-1} b - A_B^{-1} A_N x_N$, tako da je moguće eliminisati bazične promenljive iz ciljne funkcije. Analogno sa x_B i x_N definišemo c_B i c_N tako da je

$$c^T x = c_B^T x_B + c_N^T x_N = c_B^T A_B^{-1} b + (c_N^T - c_B^T A_B^{-1} A_N) x_N,$$

odakle je $c^T x^* = c_B^T A_B^{-1} b$ za bazično rešenje x^*. Uvedimo oznake

$$d' = c_B^T A_B^{-1} b,\ (c'_N)^T = c_N^T - c_B^T A_B^{-1} A_N,\ A'_N = A_B^{-1} A_N,\ b' = A_B^{-1} b.$$

Sada se standardni oblik problema (1.1.4) može zapisati u ekvivalentnom obliku

$$\begin{array}{lll} \min & d' & +(c'_N)^T x_N, \\ \text{p.o.} & x_B & +A'_N x_N = b', \\ & x_B & \geq 0,\ x_N \geq 0. \end{array} \tag{2.1.2}$$

Problem (2.1.1) je *kanonski oblik problema* (1.1.4) *u odnosu na bazu* $\mathcal{A}_B$. U kanonskom obliku matrica odgovarajućeg sistema jednačina uz bazične promenljive sadrži različite jedinične kolone, a u ciljnoj funkciji su koeficijenti uz bazične promenljive jednaki nuli.

Lema 2.1.1 *Vektor koeficijenata ciljne funkcije u kanonskom obliku problema linearnog programiranja je jednoznačno određen bazom* $\mathcal{A}_B$*, tj. ne zavisi od poretka kolona u matrici* $\mathcal{A}_B$*.*

Dokaz. Neka je bazi $\mathcal{A}_B$ pridružena matrica $\bar{A}_B$ sa poretkom kolona različitim od $j_1, \ldots, j_m$ tada je $\bar{A}_B = A_B P$, gde je P permutaciona matrica. Kako je $P^{-1} = P^T$, polazni problem se može zapisati u obliku

$$\begin{gathered} \min \bar{c}_B^T \bar{x}_B + c_N^T x_N, \\ \bar{A}_B \bar{x}_B + A_N x_N = b, \\ \bar{x}_B \geq 0, \quad x_N \geq 0, \end{gathered}$$

gde je $\bar{x}_B = P^{-1} x_B$, $\bar{c}_B = P^{-1} c_B$, odakle se dobija kanonski oblik

$$\begin{array}{lll} \min & \bar{c}_B^T & (\bar{A}_B)^{-1} b + (c_N^T - \bar{c}_B^T (\bar{A}_B)^{-1} A_N) x_N, \\ \text{p.o.} & \bar{x}_B & +(\bar{A}_B)^{-1} A_N x_N = (\bar{A}_B)^{-1} b, \\ & \bar{x}_B & \geq 0, \quad x_N \geq 0. \end{array} \tag{2.1.3}$$

S obzirom da je $\bar{c}_B^T (\bar{A}_B)^{-1} = c_B^T (P^{-1})^T (P^{-1}) A_B^{-1} = c_B^T A_B^{-1}$, to su slobodni članovi i vektori koeficijenata u ciljnim funkcijama problema (2.1.1) i (2.1.3) jednaki, dok se sistem jednačina u (2.1.3) može zapisati u obliku $P^{-1}(x_B + A_B^{-1} A_N x_N) = P^{-1} b$, tj. radi se o permutaciji jednačina sistema u (2.1.1). □

Koristićemo problem linearnog programiranja u simetričnom obliku:

$$\begin{array}{ll} \min & z(x) = \displaystyle\sum_{i=1}^{n} c_i x_i + d \\ \text{p.o} & \displaystyle\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, m, \\ & x_j \geq 0, \quad j = 1, \ldots, n. \end{array} \tag{2.1.4}$$

Definicija 2.1.1 *Problem* (2.1.4) *je bazično dopustiv ako je* $b_i \geq 0$ *za svako* $i = 1, \ldots, m$

Prethodna definicija je u bliskoj vezi sa pojmom bazično dopustivog rešenja. Naime, ako uvedemo dodatne slack promenljive, dobijamo:

$$\begin{aligned} \min \quad & z(x) = \sum_{i=1}^{n} c_i x_i + d \\ \text{p.o} \quad & \sum_{j=1}^{n} a_{ij} x_j - b_i = -x_{n+i}, \quad i = 1, \ldots, m. \end{aligned} \tag{2.1.5}$$

Oblik (2.1.5) se često naziva *kanonski* oblik problema linearnog programiranja. Promenljive na desnoj strani (u ovom slučaju $x_{n+1}, \ldots, x_{n+m}$) nazivamo *bazičnim* dok one na levoj strani jednačina (u ovom slučaju $x_1, \ldots, x_n$) *nebazičnim*. Bazične i nebazične promenljive nadalje ćemo označavati sa $x_{B,1}, \ldots, x_{B,m}$ i $x_{N,1}, \ldots, x_{N,n}$, redom. Ako sada stavimo $x_{N,1} = \ldots = x_{N,n} = 0$, tada je $x_{B,i} = b_i$ za $i = 1, \ldots, m$. Ovako dobijeno rešenje je bazično dopustivo, akko je $b_i \geq 0$ za svako $i = 1, \ldots, m$. Primetimo takođe da je matrica sistema u standardnom obliku (2.1.5) $A' = [A|I_m]$ gde je I_m jedinična matrica. I_m je bazična matrica koja odgovara upravo konstruisanom bazičnom rešenju.

Posledica 2.1.1 *Neka u bazično dopustivom kanonskom obliku* (2.1.5) *važi* $c_j \geq 0$ *za svako* $j = 1, \ldots, n$. *Tada je bazično rešenje* $x^* = (0, \ldots, 0, b_1, \ldots, b_m)$ *optimalno.*

Dokaz. S obzirom da je $x_B^* = b' \geq 0$ a $x_N^* = 0$, može se zaključiti da je x^* dopustivo rešenje. Ako je x proizvoljno dopustivo rešenje, tada iz $x \geq 0$ i $x_{N,j}^* = 0$ za $j = 1, \ldots, n$ sledi

$$z(x) = c_1 x_{N,1} + \ldots + c_n x_{N,n} + d \leq d = f(x^*)$$

pa je x^* optimalno rešenje. □

Fundamentalna osobina kanonskog oblika je da se na osnovu koeficijenata može odrediti da li je odgovarajuće bazično rešenje optimalno, kao i u kom slučaju je ciljna funkcija neograničena odozdo.

Teorema 2.1.2 *Neka je u kanonskom obliku* (2.1.3)

$$c_j' \geq 0, \ j \in N, \ b_i' \geq 0, \ i = 1, \ldots, m.$$

Tada je bazično rešenje koje odgovara bazi $\mathcal{A}_B$ *optimalno.*

Dokaz. Iz $x_N^* = 0$ sledi $x_B^* = b' \geq 0$, dakle x^* je dopustivo rešenje. Ako je x proizvoljno dopustivo rešenje, tada iz $x \geq 0$ i $x_j^* = 0$, $j \in N$ sledi

$$c^T x = d' + \sum_{j \in N} c_j' x_j \geq d' = d' + \sum_{j \in N} c_j' x_j^* = c^T x^*,$$

pa je x^* optimalno rešenje. □

Teorema 2.1.3 *Neka je u kanonskom obliku* (3.1.2) $b'_i \geq 0$, $i = 1, \ldots, m$ *i za neko* $k \in N$ *je ispunjeno* $c'_k < 0$ *i* $a'_{ik} \leq 0$, $i = 1, \ldots, m$. *Tada je ciljna funkcija na dopustivom skupu neograničena odozdo.*

Dokaz. Neka su za $t \geq 0$ koordinate tačke $x(t)$ definisane sa

$$x_{j_i}(t) = b'_i - a'_{ik}t,\ i = 1, \ldots, m, \quad x_j(t) = 0,\ j \in N\backslash\{k\}, \quad x_k(t) = t.$$

Direktno se proverava da je $x(t)$ dopustiva tačka problema (2.1.1) i važi

$$c^T x(t) = d' + \sum_{j \in N} c'_j x_j(t) = d' + c'_k t \to -\infty, \quad t \to \infty.$$

□

Ukoliko u kanonskom obliku nisu ispunjeni uslovi iz Teoreme 3.1.2 ili Teoreme 3.1.3 tada možemo preći na kanonski oblik koji odgovara nekoj novoj susednoj bazi ali tako da se vrednost ciljne funkcije u novom bazično dopustivom rešenju smanji ili bar ne poveća. Ta ideja je u osnovi simpleks metoda. Pri tome se koristi *tablični zapis* problema linearnog programiranja.

Elementarne transformacije LP-tablice su:
1) množenje $i-$te vrste sa $\lambda \neq 0$, za $i \in \{1, \ldots, m\}$,
2) dodavanje $i-$te vrste, za $i = 1, \ldots, m$, $j-$toj vrsti za $j \in \{0, 1, \ldots, m\}$.

Lema 2.1.2 *LP-tablica se primenom konačno mnogo elementarnih transformacija transformiše u LP-tablicu koja odgovara ekvivalentnom problemu linearnog programiranja.*

Dokaz. Elementarne transformacije sa $i-$tom vrstom za $i \in \{1, \ldots, m\}$ očigledno ne menjaju dopustivi skup. Dodavanje $i-$te vrste za $i \in \{1, \ldots, m\}$ nultoj vrsti transformiše je u vrstu

$$[-d + b_i \,|\, c_1 + a_{i1} \cdots c_n + a_{in}],$$

koja odgovara novoj ciljnoj funkciji

$$\begin{aligned} z &= d - b_i + (c_1 + a_{i1})x_1 + \cdots + (c_n + a_{in})x_n \\ &= d + c_1x_1 + \cdots + c_nx_n + (-b_i + a_{i1}x_1 + \cdots + a_{in}x_n) \\ &= d + c_1x_1 + \cdots + c_nx_n, \end{aligned}$$

tj. na dopustivom skupu vrednosti transformisane ciljne funkcije i polazne ciljne funkcije su jednake, pa su problemi ekvivalentni. □

Iz Leme 3.1.2 sledi da se postupkom koji je analogan Gauss-Jordan-om metodu dobijaju različite, međusobno ekvivalentne, formulacije problema.

U ovoj glavi prikazaćemo simpleks metod za rešavanje zadataka linearnog programiranja. Ovaj metod je razvio 1940. godine George B. Danzig. Najpre ćemo pokazati algebarski pristup, koji je pogodan kada se radi o zadatku manjih dimenzija, a zatim ćemo dati uprošćenje simpleks algoritma pomoću tzv. simpleks-tabela.

2.2 Algebarska suština simpleks metoda

Neka je zadat sledeći problem linearnog programiranja sa $r = m$ linearno nezavisnih ograničenja:

$$\begin{aligned} \min \quad & z(x) = c^T x = \sum_{i=1}^{n} c_i x_i \\ \text{p.o} \quad & \sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1, \ldots, m, \\ & x_j \geq 0, \quad j = 1, \ldots, n. \end{aligned}$$

Optimalno rešenje se nalazi u jednom od temena konveksnog poliedra, gde je najmanje $k = n-m$ promenljivih jednako nuli. Izaberimo proizvoljno k promenljivih za nezavisne (slobodne) promenljive i pomoću njih izrazimo zavisne promenljive. Neka su nezavisne promenljive $x_1, \ldots, x_k$ takve da kada pomoću njih izrazimo $m = n - k$ zavisnih promenljivih:

$$\begin{aligned} -x_{k+1} &= \alpha_{k+1,1}x_1 + \alpha_{k+1,2}x_2 + \cdots + \alpha_{k+1,k}x_k - \beta_{k+1} \\ \cdots &\quad \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots \\ -x_n &= \alpha_{n,1}x_1 + \alpha_{n,2}x_2 + \cdots + \alpha_{n,k}x_k - \beta_n \end{aligned} \tag{2.2.1}$$

koeficijenti $\beta_{k+1}, \ldots, \beta_n$ jesu pozitivni. Funkcija cilja u tački $x = (x_1, \ldots, x_k)$ dobija vrednost

$$z(x_1, \ldots, x_n) = z(x_1, \ldots, x_k) = \gamma_0 + \gamma_1 x_1 + \cdots + \gamma_k x_k. \tag{2.2.2}$$

Ako stavimo da je $x_1 = x_2 = \cdots = x_k = 0$ dobijamo bazično rešenje

$$(\underbrace{0, 0, \ldots, 0}_{k}, \beta_{k+1}, \beta_{k+2}, \ldots, \beta_n), \quad \beta_{k+1}, \ldots, \beta_n \geq 0$$

i minimalnu vrednost

$$z = \gamma_0.$$

Razmotrimo sada da li možemo da umanjimo vrednost ciljne funkcije z uvećavajući neku od promenljivih $x_1, x_2, \ldots, x_k$. Ako su svi koeficijenti $\gamma_1, \gamma_2, \ldots, \gamma_k$ uz promenljive $x_1, x_2, \ldots, x_k$ u funkciji cilja pozitivni, onda uvećavajući neku od ovih promenljivih (na vrednosti koje su veće od nule) ne možemo da umanjimo vrednost funkcije cilja. To znači da je dobijeno bazično rešenje i optimalno. Međutim, ako između koeficijenata $\gamma_1, \gamma_2, \ldots, \gamma_k$ ima i negativnih, onda uvećavajući vrednost promenljive ispred koje je koeficijent u funkciji cilja negativan, možemo da umanjimo vrednost funkcije cilja z, tj. možemo da dođemo do boljeg bazičnog rešenja. Neka je, na primer negativan koeficijent γ_j u izrazu (2.2.2). Sada ima smisla uvećati promenljivu x_j i na taj način preći od dobijenog bazičnog rešenja, gde je x_j bilo jednako nuli, do novog bazičnog rešenja gde će umesto x_j biti jednaka nuli neka od zavisnih promenljivih. Uvećavajući x_j smanjujemo vrednost funkcije cilja z ali moramo

da pazimo da nam pri tom neka od zavisnih promenljivih $x_{k+1}, x_{k+2}, \ldots, x_n$ ne postane negativna. One, očigledno, neće postati negativne, ako su u jednačinama (2.2.1) koeficijenti $\alpha_{k+1,j}, \ldots, \alpha_{nj}$ uz x_j negativni, pa se tada x_j može uvećati do beskonačnosti, za neko $j, 1 \leq j \leq k$. To znači da i ciljna funkcija nije ograničena sa donje strane (tj. $z_{\min} = -\infty$). U tom slučaju zadatak nema optimalno rešenje. Pretpostavimo sada da se između koeficijenata uz x_j u jednačinama (2.2.1) nalaze i pozitivni koeficijenti. Neka je, na primer, u jednačini kojom se izražava zavisna promenljiva x_i, $i \in \{k+1, \ldots, n\}$

$$-x_i = \alpha_{i1}x_1 + \cdots + \alpha_{ij}x_j + \cdots + \alpha_{ik}x_k - \beta_i$$

pozitivan koeficijent uz x_j, tj. $\alpha_{ij} > 0$. Ako stavimo $x_1 = \cdots = x_{j-1} = x_{j+1} = \cdots = x_k = 0$, dobijamo sistem

$$\begin{aligned} -x_{k+1} &= \alpha_{k+1,j}x_j - \beta_{k+1} \\ &\cdots \\ -x_n &= \alpha_{n,j}x_j - \beta_n. \end{aligned}$$

To znači da x_j možemo uvećavati do vrednosti

$$\frac{\beta_i}{\alpha_{ij}}, \quad \beta_i > 0, \alpha_{ij} > 0, \ i = k+1, \ldots, n$$

jer za tu vrednost promenljive x_j promenljiva x_i postaje jednaka nuli: $x_i = 0$. Pri daljem uvećavanju x_j promenljiva x_i bi postala negativna. Izaberimo sada između promenljivih $x_{k+1}, x_{k+2}, \ldots, x_n$ promenljivu x_p prema uslovu

$$\frac{\beta_p}{\alpha_{pj}} = \min\left\{\frac{\beta_i}{\alpha_{ij}}, \quad \alpha_{ij} > 0, \ k+1 \leq i \leq n\right\}.$$

Tada se veličina x_j izražava iz jednačine

$$-x_p = \alpha_{p1}x_1 + \cdots + \alpha_{p,j}x_j + \cdots + \alpha_{pk}x_k - \beta_p$$

i zamenjuje u ostalim jednačinama (2.2.1) i u funkciji cilja (2.2.2). Na taj način, umesto promenljive x_j među nezavisne promenljive je ušla promenljiva x_p, a promenljiva x_j, koja je u prethodnom bazičnom rešenju bila nezavisna, sada postaje zavisna promenljiva. Simbolično, tu promenu označavamo oznakom $x_p \leftrightarrow x_j$. Znači, sada su zavisne promenljive

$$x_j, x_{k+1}, \ldots, x_{p-1}, x_{p+1}, \ldots, x_n.$$

Ove zavisne promenljive se izražavaju pomoću nezavisnih promenljivih

$$x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k, x_p.$$

Takođe, i ciljna funkcija se izražava pomoću nezavisnih promenljivih.

Izvedeni postupak se sada ponavlja. Ako su svi koeficijenti uz nezavisne promenljive $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k, x_p$ u funkciji cilja pozitivni, onda je za

$$x_j = x_{k+1} = \cdots = x_{p-1} = x_{p+1} = \cdots = x_n = 0$$

dobijeno bazično rešenje i optimalno rešenje. Ukoliko ima i negativnih koeficijenata uz nezavisne promenljive postupak se ponavlja dok se ne dođe do optimalnog rešenja.

Neka je, u opštem slučaju zadat bazično dopustiv problem (2.1.5) kod koga je $c_j < 0$ za neko $j \in \{1, \ldots, n\}$ i $a_{ij} > 0$ za neko $i \in \{1, \ldots, m\}$. S obzirom na $c_j < 0$, ima smisla uvećati promenljivu $x_{N,j}$ i na taj način preći od dobijenog bazičnog rešenja, gde je $x_{N,j}$ bilo jednako nuli, do novog bazičnog rešenja gde će umesto $x_{N,j}$ biti jednaka nuli neka od zavisnih promenljivih. Uvećavajući $x_{N,j}$ smanjujemo vrednost funkcije cilja $z(x)$ ali moramo da pazimo da nam pri tom neka od bazičnih promenljivih $x_{B,1}, x_{B,2}, \ldots, x_{B,m}$ ne postane negativna. Ukoliko je $a_{sj} < 0$, promenljiva $x_{B,s}$ očigledno neće postati negativna. Posmatrajmo sada jednačinu:

$$-x_{B,i} = a_{i1}x_{N,1} + \ldots + a_{ij}x_{N,j} + \ldots + a_{in}x_{N,n} - b_i$$

Ako stavimo $x_{N,1} = \cdots = x_{N,j-1} = x_{N,j+1} = \cdots = x_{N,n} = 0$, dobijamo:

$$\begin{aligned} -x_{B,1} &= a_{1j}x_{N,j} - b_i \\ &\cdots \\ -x_{B,m} &= a_{nj}x_{N,j} - b_n. \end{aligned}$$

To znači da $x_{N,j}$ možemo uvećavati do vrednosti:

$$\frac{b_i}{a_{ij}}, \quad b_i > 0, a_{ij} > 0,$$

jer za tu vrednost promenljive $x_{N,j}$ promenljiva $x_{B,i}$ postaje jednaka nuli. Pri daljem uvećavanju $x_{N,j}$ promenljiva $x_{B,i}$ bi postala negativna. Izaberimo sada bazičnu promenljivu $x_{B,p}$ prema uslovu

$$\frac{b_p}{a_{pj}} = \min\left\{\frac{b_i}{a_{ij}}, \quad a_{ij} > 0,\ k+1 \leq i \leq n\right\}. \tag{2.2.3}$$

Izvršimo sada zamenu promenljivih $x_{B,p}$ i $x_{N,j}$, tj izrazimo sada promenljivu $x_{N,j}$ iz jednačine

$$-x_{B,p} = a_{p1}x_{N,1} + \cdots + a_{pj}x_{N,j} + \cdots + a_{pn}x_{N,n} - b_p$$

i zamenimo je u ostalim jednačinama i u funkciji cilja. Na taj način, umesto promenljive $x_{N,j}$ među nebazične promenljive je ušla promenljiva $x_{B,p}$, a promenljiva $x_{N,j}$ koja je u prethodnom bazičnom rešenju bila nezavisna, sada postaje zavisna promenljiva. Ovim smo dobili novo bazično dopustivo rešenje:

$$x^1 = (x_N^1, x_B^1) = \left((0, \ldots, 0, b_p^1, 0, \ldots, 0), (b_1^1, \ldots, b_{p-1}^1, 0, b_{p+1}^1, \ldots, b_n^1)\right)$$

gde je:

$$b_p^1 = \frac{b_p}{a_{pj}}, \quad b_l^1 = b_l - a_{lj}\frac{b_p}{a_{pj}}, \quad l \neq p$$

kao i ekvivalentan problem u kanonskom obliku. U tački x^1, funkcija cilja ima veću vrednost nego u polaznom bazično dopustivom rešenju. Ako sada nastavimo da primenjujemo isti postupak sa novim kanonskim oblikom, funkcija cilja će se povećavati i u jednom trenutku ćemo sigurno doći u situaciju da možemo da primenimo lemu (2.1.1) ili dolazimo do zaključka da je ciljna funkcija neograničena. Ovo intuitivno razmatranje je ključno za simpleks metod, i biće u sledećem odeljku formalnije sprovedeno. Takođe, uvešćemo pojam Tuckerove tabele koji u mnogome pojednostavljuje upravo opisan postupak konstruisanja novog bazičnog rešenja. Ovakav pristup se najčešće koristi prilikom implementacije simpleks metoda.

Primer 2.2.1 Rešiti sledeći problem

$$\begin{array}{lll} \min & z(x) & = 5x_1 - 2x_3 \\ \text{p.o.} & & -5x_1 - x_2 + 2x_3 \leq 2 \\ & & -x_1 \quad\quad + x_3 + \ x_4 \leq 5 \\ & & -3x_1 \quad\quad\quad + 5x_4 \leq 7 \\ & & x_i \geq 0, \quad i = 1, 2, 3, 4. \end{array}$$

Rešenje: Ograničenjima dodajemo dopunske promenljive tako da se zadatak transformiše na sledeći standardni oblik:

$$\begin{array}{lll} \multicolumn{3}{l}{\min z(x) = 5x_1 - 2x_3} \\ \text{p.o.} & -5x_1 - x_2 + 2x_3 \quad\quad + x_5 \quad\quad\quad = 2 \\ & -x_1 \quad\quad + x_3 + x_4 \quad\quad + x_6 \quad\quad = 5 \\ & -3x_1 \quad\quad\quad + 5x_4 \quad\quad\quad + x_7 = 7 \\ & x_i \geq 0, \quad i = 1, \ldots, 7. \end{array}$$

Kako je $r = m = 3$ i $k = n - m = 7 - 3 = 4$, znači da je broj nezavisnih promenljivih 4, a broj zavisnih 3. Neka su nezavisne promenljive x_1, x_2, x_3 i x_4. Iz sistema jednačina dobijamo:

$$\begin{array}{lll} -x_5 & = & -5x_1 - x_2 + 2x_3 - 2 \\ -x_6 & = & -x_1 \quad\quad + x_3 + x_4 - 5 \\ -x_7 & = & -3x_1 \quad\quad\quad + 5x_4 - 7 \\ z(x) & = & 5x_1 - 2x_3 \end{array}$$

Početno bazično rešenje za $x_1 = x_2 = x_3 = x_4 = 0$ jednako je

$$I : (0, 0, 0, 0, 2, 5, 7) \text{ i } z_1(x) = 0.$$

Kako je koeficijent uz x_3 u funkciji cilja negativan, vrednost funkcije cilja se može umanjiti ako x_3 uzme pozitivne vrednosti. Iz sistema jednačina se dobija za $x_1 = x_2 = x_4 = 0$:

$$x_5 = 0 \text{ za } x_3 = 1 = \frac{\beta_1}{\alpha_{13}}$$

$$x_6 = 0 \text{ za } x_3 = 5 = \frac{\beta_2}{\alpha_{23}}.$$

Takođe, rašćenje x_3 ne utiče na promenu vrednosti x_7, tj., u ovom slučaju može da poraste do $+\infty$. Kako je

$$\min\{1, 5, +\infty\} = 1.$$

To znači da promenljive x_3 i x_5 menjaju uloge u drugom bazičnom rešenju ($x_3 \leftrightarrow x_5$). Eliminacijom x_3 iz prvog ograničenja dobija se

$$\begin{array}{lll} -x_3 & = & -\frac{5}{2}x_1 - \frac{1}{2}x_2 + \frac{1}{2}x_5 - 1 \\ -x_6 & = & -x_1 + \left(1 + \frac{5}{2}x_1 + \frac{1}{2}x_2 - \frac{1}{2}x_5\right) + x_4 - 5 = \frac{3}{2}x_1 + \frac{1}{2}x_2 + x_4 - \frac{1}{2}x_5 - 4 \\ -x_7 & = & -3x_1 + 5x_4 - 7 \\ z(x) & = & -2 - x_2 + x_5. \end{array}$$

Drugo bazično rešenje je jednako

$$II : (0, 0, 1, 0, 0, 4, 7) \text{ i } z_2(x) = -2.$$

Ni ovo rešenje nije optimalno jer se, zbog negativnog koeficijenta uz x_2 u funkciji cilja, njena vrednost može umanjiti povećanjem vrednosti promenljive x_2. Povećanjem vrednosti x_2 nenegativnost promenljive x_3 se neće izgubiti, niti promenljive x_7. Međutim kod promenljive x_6 vidimo da x_2 može da uzme najveću vrednost

$$\frac{\beta_2}{\alpha_{25}} = \frac{4}{\frac{1}{2}} = 8,$$

kada x_6 postaje jednako nuli, to jest, imamo $x_2 \leftrightarrow x_6$. Sledi

$$\begin{aligned}
-x_2 &= 3x_1 + 2x_4 - x_5 + 2x_6 - 8 \\
-x_3 &= -\frac{5}{2}x_1 - \frac{1}{2}(8 - 3x_1 - 2x_4 + x_5 - 2x_6) + \frac{1}{2}x_5 - 1 = -x_1 + x_4 + \frac{1}{2}x_5 + x_6 - 5 \\
-x_7 &= -3x_1 + 5x_4 - 7 \\
z(x) &= -10 + 3x_1 + 2x_4 + 2x_6.
\end{aligned}$$

Treće bazično rešenje je jednako:

$$III : (0, 8, 5, 0, 0, 0, 7) \text{ i } z_3(x) = -10$$

Kako su sada svi koeficijenti uz promenljive u funkciji cilja pozitivni, to je ovo treće bazično rešenje i optimalno rešenje, a $z_{min} = -10$.

2.3 Pojam Tuckerove tabele i Simpleks metod za bazično dopustive kanonske oblike

Neka je zadat jedan kanonski oblik problema linearnog programiranja:

$$\begin{aligned}
&\max f(x) = c_1 x_{N,1} + \ldots + c_n x_{N,n} + d \\
&\text{p.o. } a_{11}x_{N,1} + a_{12}x_{N,2} + ... + a_{1n}x_{N,n} - b_1 = -x_{B,1} \\
&\quad a_{21}x_{N,1} + a_{22}x_{N,2} + ... + a_{2n}x_{N,n} - b_2 = -x_{B,2} \\
&\quad \ldots\ldots\ldots \\
&\quad a_{m1}x_{N,1} + a_{m2}x_{N,2} + ... + a_{mn}x_{N,n} - b_m = -x_{B,m}.
\end{aligned} \tag{2.3.1}$$

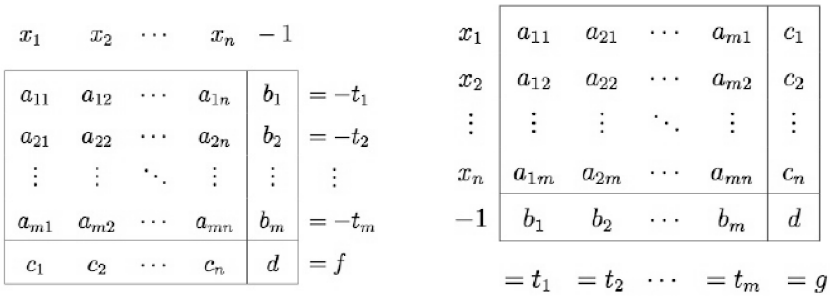
Problem možemo tabelarno prikazati na sledeći način:

$x_{N,1}$	$x_{N,2}$	$\cdots$	$x_{N,n}$	-1		
a_{11}	a_{12}	$\cdots$	a_{1n}	b_1	$=$	$-x_{B,1}$
a_{21}	a_{22}	$\cdots$	a_{2n}	b_2	$=$	$-x_{B,2}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$		$\vdots$
a_{m1}	a_{m2}	$\cdots$	a_{mn}	b_m	$=$	$-x_{B,m}$
c_1	c_2	$\cdots$	c_n	d	$=$	f

(2.3.2)

Ovu tabelu nazivamo ***Tuckerovom tabelom*** za problem (2.3.1). Uvedimo oznake $a_{m+1,j} = c_j$ za $j = 1, \ldots, n$ kao i $a_{i,n+1} = b_i$ za $i = 1, \ldots, n$. Takođe, neka je $a_{m+1,n+1} = d$. Sada nam je za opisivanje kanonskog oblika linearnog programiranja dovoljna *proširena Tuckerova tabela*: $\overline{A} = \{a_{ij}\}_{i=\overline{1,m+1}, j=\overline{1,n+1}}$. U nastavku ćemo često poistovećivati matrice $\overline{A}$ i A kad god nema opasnosti od zabune.

U literaturi se sreću dva oblika Tuckerovih tabela:



gde su $x_1, \ldots, x_n$ nebazične, a $t_1, \ldots, t_m$ bazične promenljive.

Na kraju prethodnog odeljka bilo je potrebno izraziti nebazičnu promenljivu $x_{N,j}$ iz p-te jednačine sistema (2.3.1) i zameniti je u ostalim jednačinama i u funkciji cilja. Na taj način, promenljiva $x_{B,p}$ postaje nebazična, dok $x_{N,j}$ postaje bazična.

Posmatrajmo sada šta se dešava sa matricom sistema A' odgovarajućeg standardnog oblika (2.1.2). U matrici A' kolone koje odgovaraju bazičnim promenljivima formiraju jediničnu matricu dok ostale formiraju Tuckerovu tabelu A. Posle zamene promenljivih kolona $K_{v_{N,j}}$ matrice A' postaje jednaka p-toj koloni jedinične matrice. Da bi to uradili, dovoljno je da za svako $i = 1, \ldots, m$, $i \neq p$ od i-te vrste matrice A' oduzmemo p-tu vrstu pomnoženu sa:

$$\frac{a'_{i,v_{n,j}}}{a'_{p,v_{n,j}}} = \frac{a_{ij}}{a_{pj}}, \quad A' = [a'_{ij}]_{i=\overline{1,m},j=\overline{1,m+n}}$$

dok p-tu vrstu samo podelimo sa $a_{pj} = a'_{p,v_{n,j}}$. Na isti način transformišemo vektor b' i funkciju cilja smatrajući ih redom $n+m+1$-om kolonom i $m+1$-om vrstom matrice A'. Ovu transformaciju opisujemo u vektorskom i skalarnom obliku na sledeći način:

$$V_q^1 = V_q - \frac{a_{qj}}{a_{pj}} V_p, \qquad (a')_{ql}^1 = a'_{ql} - \frac{a'_{pl} a'_{q,v_{B,j}}}{a'_{p,v_{B,j}}} \tag{2.3.3}$$

Gde je sa V_i označena i-ta vrsta $A'_{i\bullet}$ matrice A'. U tom slučaju nova Tuckerova tabela jednaka je:

$$A^1 = \begin{array}{ccccccccc}
x_{N,1} & \cdots & x_{N,j-1} & x_{B,p} & x_{N,j+1} & \cdots & x_{N,n} & -1 & \\
a^1_{11} & \cdots & a^1_{1,j-1} & a^1_{1j} & a^1_{1,j+1} & \cdots & a^1_{1n} & b^1_1 & = -x_{B,1} \\
a^1_{21} & \cdots & a^1_{2,j-1} & a^1_{2j} & a^1_{2,j+1} & \cdots & a^1_{2n} & b^1_2 & = -x_{B,2} \\
\vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\
a^1_{p-1,1} & \cdots & a^1_{p-1,j-1} & a^1_{p-1,j} & a^1_{p-1,j+1} & \cdots & a^1_{p-1,n} & b^1_{p-1} & = -x_{B,p-1} \\
a^1_{p1} & \cdots & a^1_{p,j-1} & a^1_{pj} & a^1_{p,j+1} & \cdots & a^1_{pn} & b^1_p & = -x_{B,j} \\
a^1_{p+1,1} & \cdots & a^1_{p+1,j-1} & a^1_{p+1,j} & a^1_{p+1,j+1} & \cdots & a^1_{p+1,n} & b^1_{p+1} & = -x_{B,p+1} \\
\vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\
a^1_{m1} & \cdots & a^1_{m,j-1} & a^1_{mj} & a^1_{m,j+1} & \cdots & a^1_{mn} & b^1_m & = -x_{B,m} \\
c^1_1 & \cdots & c^1_{j-1} & c^1_j & c^1_{j+1} & \cdots & c^1_n & d^1 & = f
\end{array} \tag{2.3.4}$$

gde su elementi a^1_{ql} dati pomoću izraza:

$$\begin{aligned} a^1_{pj} =& \frac{1}{a_{pj}}; \\ a^1_{pl} =& \frac{a_{pl}}{a_{pj}}, && l \neq j; \\ a^1_{qj} =& -\frac{a_{qj}}{a_{pj}}, && q \neq p; \\ a^1_{ql} =& a_{ql} - \frac{a_{pl}a_{qj}}{a_{pj}}, && q \neq p,\ l \neq j; \end{aligned} \tag{2.3.5}$$

Naravno, ovde podrazumevamo da je $q = 1, \ldots, m+1$ i $l = 1, \ldots n+1$. Izrazi (2.3.5) dobijaju se direktno iz izraza (2.3.3) ako se uzme u obzir struktura matrice A'.

2.4 Algoritam simpleks metoda

Posmatramo linearni program

$$\begin{aligned} \max \quad & z(x) = z(x_{N,1}, \ldots, x_{N,n_1}) = \sum_{i=1}^{n_1} c_i x_{N,i} - d \\ \text{p.o.} \quad & N_i^{(1)}: \quad \sum_{j=1}^{n_1} a_{ij} x_{N,j} \leq b_i, \quad i = 1, \ldots, r \\ & N_i^{(2)}: \quad \sum_{j=1}^{n_1} a_{ij} x_{N,j} \geq b_i, \quad i = r+1, \ldots, s \\ & J_i: \quad \sum_{j=1}^{n_1} a_{ij} x_{N,j} = b_i, \quad i = s+1, \ldots, m \\ & x_{N,j} \geq 0, \quad j = 1, \ldots, n_1. \end{aligned} \tag{2.4.1}$$

Pri tome su a_{ij}, b_i i c_j poznati realni brojevi. Svaka nejednakost oblika $N_i^{(1)}$ (LE ograničenja) se transformiše u odgovarajuću jednačinu dodajući *slack* (*izračunavajuće*) promenljive $x_{B,i}$:

$$N_i^{(1)}: \quad \sum_{j=1}^{n_1} a_{ij} x_{N,j} + x_{B,i} = b_i, \quad i = 1, \ldots, r.$$

Takođe, svaka nejednakost oblika $N_i^{(2)}$ (GE ograničenje) se transformiše u jednakost oduzimanjem *surplus* (*izravnjavajućih*) promenljivih $x_{B,i}$:

$$N_i^{(2)}: \quad \sum_{j=1}^{n_1} a_{ij} x_{N,j} - x_{B,i} = b_i, \quad i = r+1, \ldots, s.$$

Na taj način, dobijamo linearni program u standardnoj formi

$$\begin{array}{ll} \max & c_1x_1+\cdots+c_{n_1}x_{n_1}-d \\ \text{p.o.} & Ax=b, \\ & b=(b_1,\ldots,b_m),\ x=(x_{N,1},\ldots,x_{N,n_1},x_{B,1},\ldots,x_{B,m}) \\ & x_{N,j}\geq 0,\ j=1,\ldots,n_1, \\ & x_{B,i}\geq 0,\ i=1,\ldots,s,\ x_{B,i}=0,\ i=s+1,\ldots,m \end{array} \tag{2.4.2}$$

gde matrica A pripada skupu $\mathbb{R}^{m\times(n_1+s)}$.

U svakoj jednačini odaberemo jednu od promenljivih $x_{N,j}$ za koju važi $a_{p,j}\neq 0$ za bazičnu, i izvršimo odgovarajuće zamene u ostalim jednačinama. Može se koristiti algoritam za zamenu bazične promenljive $x_{B,p}=0$ i nebazične $x_{N,j}$. Posle zamene $n=n_1+s$, kanonička forma problema (2.4.2) može biti zapisana u sledećem tabličnom obliku T_0:

Početna tabela T_0

$x_{N,1}$	$x_{N,2}$	$\ldots$	$x_{N,n}$	-1	
a_{11}^0	a_{12}^0	$\ldots$	a_{1n}^0	b_1^0	=-$x_{B,1}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
a_{m1}^0	a_{m2}^0	$\ldots$	a_{mn}^0	b_m^0	=-$x_{B,m}$
c_1^0	c_2^0	$\ldots$	c_n^0	d^0	$=z^0$

(2.4.3)

gde su $x_{N,1},\ldots,x_{N,n}$ nebazične promenljive i $x_{B,1},\ldots,x_{B,m}$ bazične promenljive. U tabeli (2.4.3) su transformisani koeficijenti matrice A i vektora c označeni sa a_{ij}^0 i c_j^0, respektivno. Tabela (2.4.3) naziva se Tuckerova koja odgovara maksimizaciji funkcije cilja.

Definicija 2.4.1 *Rešenje zadatka linearnog programiranja kod koga su nezavisne promenljive jednake nuli naziva se bazično rešenje.*

Propozicija 2.4.1 *Neka je Tuckerova tabela osnovnog maksimizacionog zadatka linearnog programiranja predstavljena tabelom T_k oblika*

Tabela T_k u k-toj iteraciji.

$x_{N,1}$	$x_{N,2}$	$\ldots$	$x_{N,n}$	-1	
a_{11}^k	a_{12}^k	$\ldots$	a_{1n}^k	b_1^k	=-$x_{B,1}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
a_{m1}^k	a_{m2}^k	$\ldots$	a_{mn}^k	b_m^k	=-$x_{B,m}$
c_1^k	c_2^k	$\ldots$	c_n^k	d^k	$=z^k$

(2.4.4)

Ako je $b_1^k,\ldots,b_m^k\geq 0$, tada u bazično dopustivoj maksimizacionoj tabeli bazično rešenje jeste dopustivo rešenje.

Dokaz. Zaista, stavljajući da su sve nezavisne promenljive jednake nuli, sva glavna ograničenja svode se na jednačine $-b_i^k=-x_{B,i}$ ili $b_i^k=x_{B,i}$. Na taj način, svako rešenje zadovoljava sva ograničenja postavljenog zadatka, pa znači da je dopustivo. □

Sledi algoritam simpleks metoda za maksimizacionu bazično dopustivu tabelu T_k datu u (2.4.4).

Algoritam SimpleksBasicMax

(Simpleks metod za maksimizacionu bazično dopustivu tabelu).

k-ta iteracija simpleks metoda sastoji se od sledećih koraka:

Korak 1. Ako je $c_1^k, \ldots, c_n^k \leq 0$, algoritam staje. Bazično dopustivo rešenje koje odgovara simpleks tablici T_k je optimalno.

Korak 2. Izaberi proizvoljno $c_j^k > 0$. (Može se uzeti maksimalno $c_j^k > 0$. Selekcija prvog $c_j^k > 0$ rešava problem cikliranja.)

Korak 3. Za svako l za koje je $c_l^k > 0$, ispitati da li je $a_{il}^k \geq 0$ za svako $i = 1, \ldots, m$. Ako takvo l postoji algoritam staje. Ciljna funkcija je na dopustivom skupu neograničena odozgo, tj. maksimum je $+\infty$.

Ovaj korak je modifikacija odgovarajućeg koraka iz [84]. Modifikacija se sastoji u tome što se uslov iz Koraka 3 proverava za svako l za koje važi $c_l^k > 0$, a ne samo za $l = j$.

Korak 4. Izračunati

$$\min_{1 \leq i \leq m} \left\{ \frac{b_i^k}{a_{ij}^k}, \quad a_{ij}^k > 0 \right\} = \frac{b_p^k}{a_{pj}^k}$$

i zameniti nebazičnu promenljivu $x_{N,j}$ i bazičnu promenljivu $x_{B,p}$. Simbolički se ova transformacija zapisuje na sledeći način: $x_{N,j} \leftrightarrow x_{B,p}$.

Detaljnije je ovakva transformacija opisana u algoritmu **Replace**.

Algoritam Replace.

(Zamena bazične promenljive $x_{B,p}$ i nebazične promenljive $x_{N,j}$.)
Transformacija vodećeg elementa:

$$a_{pj}^{k+1} = \frac{1}{a_{pj}^k}. \tag{2.4.5}$$

Transformacija elementa u vodećoj vrsti, izuzimajući vodeći element:

$$a_{pl}^{k+1} = \frac{a_{pl}^k}{a_{pj}^k}, \quad l \neq j, \tag{2.4.6}$$

Transformacija elementa vektora b u vodećoj vrsti:

$$b_p^{k+1} = \frac{b_p^k}{a_{pj}^k}, \tag{2.4.7}$$

Transformacija elementa u vodećoj vkoloni, izuzev vodećeg elementa:

$$a_{qj}^{k+1} = -\frac{a_{qj}^k}{a_{pj}^k}, \quad q \neq p, \tag{2.4.8}$$

Transformacija elementa izvan vodeće vrste i vodeće kolone:

$$a_{ql}^{k+1} = a_{ql}^{k} - \frac{a_{pl}^{k} a_{qj}^{k}}{a_{pj}^{k}}, \quad q \neq p,\, l \neq j; \tag{2.4.9}$$

Transformacija elementa vektora b izvan vodeće vrste:

$$b_{q}^{k+1} = b_{q}^{k} - \frac{b_{p}^{k} a_{qj}^{k}}{a_{pj}^{k}}, \quad q \neq p \tag{2.4.10}$$

Transformacija elementa vektora c u vodećoj vrsti:

$$c_{j}^{k+1} = -\frac{c_{j}^{k}}{a_{pj}^{k}}, \tag{2.4.11}$$

Transformacija elementa vektora c izvan vodeće vrste:

$$c_{l}^{k+1} = c_{l}^{k} - \frac{c_{j}^{k} a_{pl}^{k}}{a_{pj}^{k}}, \quad l \neq j; \tag{2.4.12}$$

Transformacija slobodnog člana d:

$$d^{k+1} = d^{k} - \frac{b_{p}^{k} c_{j}^{k}}{a_{pj}^{k}}. \tag{2.4.13}$$

Korak 5. Zameniti k sa $k+1$ i ići na Korak 1.

Razmotrimo *Algoritam Replace.*

Jednačina kojom je izražena bazična promenljiva $x_{B,p}$ je oblika

$$\sum_{l=1}^{n} a_{pl}^{k} x_{N,l} - b_{p}^{k} = -x_{B,p}$$

odakle se dobija posle zamene

$$\sum_{l=1, l\neq j}^{n} \frac{a_{pl}^{k}}{a_{pj}^{k}} x_{N,l} + \frac{1}{a_{pj}^{k}} x_{B,p} - \frac{b_{p}^{k}}{a_{pj}^{k}} = -x_{N,j}. \tag{2.4.14}$$

Odatle se dobijaju jednakosti (2.4.5), (2.4.6) i (2.4.7). Za $q \neq p$ iz

$$\sum_{l=1}^{n} a_{ql}^{k} x_{N,l} - b_{q}^{k} = -x_{B,q}$$

i (2.4.14) dobijamo

$$\sum_{l=1, l\neq j}^{n} a_{ql}^{k} x_{N,l} - a_{qj}^{k} \left(\sum_{l=1, l\neq j}^{n} \frac{a_{pl}^{k}}{a_{pj}^{k}} x_{N,l} + \frac{1}{a_{pj}^{k}} x_{B,p} - \frac{b_{p}^{k}}{a_{pj}^{k}} \right) - b_{p}^{k} = -x_{B,q},$$

odnosno

$$\left(a_{q1}^k - \frac{a_{p1}^k a_{qj}^k}{a_{pj}^k}\right) x_{N,1} + \cdots + \left(a_{q,j-1}^k - \frac{a_{p,j-1}^k a_{qj}^k}{a_{pj}^k}\right) x_{N,j-1}$$
$$-\frac{a_{qj}^k}{a_{pj}^k} x_{B,p}$$
$$+\left(a_{q,j+1}^k - \frac{a_{p,j+1}^k a_{qj}^k}{a_{pj}^k}\right) x_{N,j+1} + \left(a_{qn}^k - \frac{a_{pn}^k a_{qj}^k}{a_{pj}^k}\right) x_{N,n}$$
$$-\left(b_q^k - \frac{b_p^k a_{qj}^k}{a_{pj}^k}\right) = -x_{B,q}.$$

Odatle proizilaze (2.4.8), (2.4.9) i (2.4.10). Konačno, iz

$$z^k = c_1^k x_{N,1} + \cdots + c_n^k x_{N,n} - d^k$$

i (2.4.14) dobijamo

$$z^k = \sum_{l=1, l\neq j}^{n} c_l^k x_{N,l} - c_j^k \left(\sum_{l=1, l\neq j} \frac{a_{pl}^k}{a_{pj}^k} x_{N,l} + \frac{1}{a_{pj}^k} x_{B,p} - \frac{b_p^k}{a_{pj}^k}\right) - d^k,$$

odnosno

$$\begin{aligned} z^k \;=\; & \left(c_1^k - \frac{c_j^k a_{p1}^k}{a_{pj}^k}\right) x_{N,1} + \cdots + \left(c_{j-1}^k - \frac{c_j^k a_{p,j-1}^k}{a_{pj}^k}\right) x_{N,j-1} \\ & -\frac{c_j^k}{a_{pj}^k} x_{B,p} \\ & + \left(c_{j+1}^k - \frac{c_j^k a_{p,j+1}^k}{a_{pj}^k}\right) x_{N,j+1} + \left(c_n^k - \frac{c_j^k a_{pn}^k}{a_{pj}^k}\right) x_{N,n} \\ & - \left(d^k - \frac{b_p^k c_j^k}{a_{pj}^k}\right). \end{aligned}$$

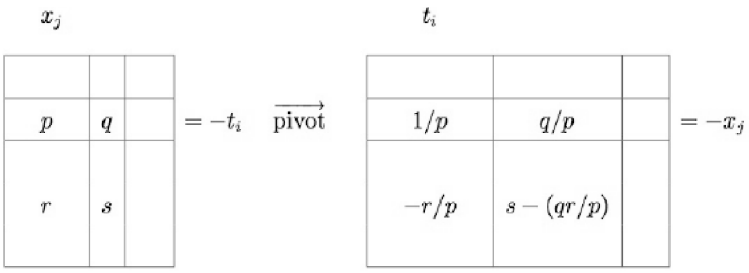
Odatle proizilaze (2.4.11), (2.4.12) i (2.4.13).

Ovakva transformacija je slična Gaussovom postupku eliminacije:

- odabrati $p = a_{ij} \neq 0$;
- razmeniti $x_{N,j}$ i $x_{B,i}$;
- zameniti p sa $1/p$;
- zameniti svaku vrednost q u $i$toj vrsti sa q/p;
- zameniti svaku vrednost r u koloni j sa $-r/p$;

- zameniti druge vrednosti s sa $s-(q\cdot r/p)$.

Ovaj proces se može prikazati sledećom slikom:



Slika 2.4.1. Grafička ilustracija transformacije u simpleks algoritmu

Napomena 2.4.1 *U opštem slučaju indeksi p i j u Koraku 3 nisu jednoznačno određeni, tj. može da postoji više kandidata koji ispunjavaju navedene uslove. Kada su p i j izabrani, oni određuju tzv. stožerni ili pivot element a_{pj} pomoću kojeg se obavljaju transformacije u Koraku 4. Prelazak sa tablice T_k na tablicu T_{k+1} naziva se pivotiranje. Niz elementarnih transformacija u Koraku 4 čini jednu složenu, tzv. stožernu transformaciju. U sledećoj teoremi se dokazuje da pivotiranje ne povećava ciljnu funkciju i da je T_{k+1} takođe simpleks tablica.*

Algoritam simpleks metoda za minimizacionu bazično dopustivu tabelu (2.4.4) sličan je algoritmu za maksimizacionu tabelu:

Algoritam SimpleksBasicMin

(Simpleks metod za minimizacionu bazično dopustivu tabelu, kao u [79, 84]).

k-ta iteracija simpleks metoda sastoji se od sledećih koraka:

Korak 1. Ako je $c_1^k,\dots,c_n^k\geq 0$, algoritam staje. Bazično dopustivo rešenje koje odgovara simpleks tablici T_k je optimalno.

Korak 2. Izaberi proizvoljno $c_j^k<0$. (Može se uzeti minimalno $c_j^k<0$ ili prvo $c_j^k<0$.)

Korak 3. Ispitati da li je $a_{ij}^k\leq 0$ za svako $i=1,\dots,m$. Ako je taj uslov ispunjen, algoritam staje. Ciljna funkcija je na dopustivom skupu neograničena odozdo, tj. minimum je $-\infty$.

Korak 4. Izračunati

$$\min_{1\leq i\leq m}\left\{\frac{b_i^k}{a_{ij}^k},\quad a_{ij}^k>0\right\}=\frac{b_p^k}{a_{pj}^k}$$

i zameniti nebazičnu promenljivu $x_{N,j}$ i bazičnu promenljivu $x_{B,p}$, koristeći algoritam *Replace*.

Korak 5. Zameniti k sa $k+1$ i ići na Korak 1.

Praktično to znači da se na tabeli T_k primenjuju sledeće elementarne transformacije:

- pomnožiti $p-$tu vrstu sa $-\frac{a_{qj}^k}{a_{pj}^k}$ i dodati vrstama $q = 0, \ldots, m$, $q \neq p$;
- podeliti p-tu vrstu sa a_{pj}^k.

Teorema 2.4.1 *Primenom elementarnih transformacija opisanih u Koraku 4 Algoritma SimpleksBasicMin se od simpleks tablice T_k dobija simpleks tablica T_{k+1} koja odgovara ekvivalentnom problemu linearnog programiranja. Pri tom je $d^{k+1} \geq d^k$.*

Dokaz. Na osnovu Leme 2.1.2 tablica T_{k+1} odgovara ekvivalentnom problemu linearnog programiranja. Pri tome iz bazične matrice izlazi kolona koja je imala jedinicu u $p-$toj vrsti a ulazi $j-$ta kolona. Na osnovu pretodnih razmatranja o prelasku na susednu bazu, iz uslova

$$\frac{b_p^k}{a_{pj}^k} = \min\left\{ \frac{b_i^k}{a_{ij}^k} | \, a_{ij}^k > 0 \right\}$$

sledi da je x^{k+1} bazično dopustivo rešenje, tj. $b_i^{k+1} \geq 0$. Dokaz sledi iz

$$b_i^{k+1} = b_i^k - \frac{b_p^k a_{ij}^k}{a_{pj}^k}$$

i

$$\frac{b_p^k}{a_{pj}^k} \leq \frac{b_i^k}{a_{ij}^k}.$$

Iz $c_j^k < 0$ sledi

$$-d^{k+1} = -d^k + \frac{c_j^k}{a_{pj}^k} b_p^k \leq -d^k,$$

tj. $d^{k+1} \geq d^k$. $\square$

Primenimo sada Algoritam *SimpleksBasicMin* na već razmatrani Primer 1.1.1:

Primer 2.4.1 Problemu

$$\begin{array}{ll} \min & -5x_1 - 4x_2, \\ \text{p.o.} & x_1 + x_2 \leq 80, \\ & 3x_1 + x_2 \leq 180, \\ & x_1 + 3x_2 \leq 180, \\ & x_1 \geq 0, \ x_2 \geq 0 \end{array}$$

odgovara simpleks tablice T_0 : (pivot elementi su zbog preglednosti uokvireni)

$T_0 =$

x_1	x_2	-1	
1	1	80	$=$-x_3
$\boxed{3}$	1	180	$=$-x_4
1	3	180	$=$-x_5
-5	-4	0	$=z$

Odgovarajuće bazično dopustivo rešenje je $x^0 = (0, 0, 80, 180, 180)$, a vrednost ciljne funkcije $c^T x^0 = 0$. Očigledno da testovi u koracima 1 i 2 nisu zadovoljeni. Izaberimo $j = 1$. Sada je

$$b_1^0/a_{11}^0 = 80,\ b_2^0/a_{21}^0 = 60,\ b_3^0/a_{31}^0 = 180,$$

odakle određujemo $p = 2$. Znači, potrebno je da se izvrši zamena $x_1 \leftrightarrow x_4$. Primenom Algoritma *Replace* dobija se sledeći niz transformacija kao i odgovarajuća tabela T_1:

$$a_{21}^1 = a_{pj}^1 = \frac{1}{a_{pj}^0} = 1,$$

$$a_{22}^1 = a_{pl}^1 = \frac{a_{pl}^0}{a_{pj}^0} = \frac{a_{22}^0}{a_{21}^0} = \frac{1}{3},$$

$$a_{11}^1 = a_{qj}^1 = -\frac{a_{qj}^0}{a_{pj}^0} = -\frac{a_{11}^0}{a_{21}^0} = -\frac{1}{3},$$

$$a_{31}^1 = a_{qj}^1 = -\frac{a_{qj}^0}{a_{pj}^0} = -\frac{a_{31}^0}{a_{21}^0} = -\frac{1}{3}a_{31}^0 - \frac{a_{21}^0 a_{31}^0}{a_{21}^0} = 1 - \frac{3\cdot 1}{3} = \frac{2}{3},$$

$$a_{12}^1 = a_{ql}^1 = a_{ql}^0 - \frac{a_{pl}^0 a_{qj}^0}{a_{pj}^0} = a_{12}^0 - \frac{a_{22}^0 a_{11}^0}{a_{21}^0} = 1 - \frac{1\cdot 1}{3} = \frac{2}{3},$$

$$a_{32}^1 = a_{ql}^1 = a_{ql}^0 - \frac{a_{pl}^0 a_{qj}^0}{a_{pj}^0} = a_{32}^0 - \frac{a_{22}^0 a_{31}^0}{a_{21}^0} = 3 - \frac{1\cdot 1}{3} = \frac{8}{3}.$$

$$b_1^1 = b_q^1 = b_q^0 - \frac{b_p^0 a_{qj}^0}{a_{pj}^0} = b_1^0 - \frac{b_2^0 a_{11}^0}{a_{21}^0} = 80 - \frac{80\cdot 1}{3} = 20,$$

$$b_2^1 = b_p^1 = \frac{b_p^0}{a_{pj}^0} = \frac{b_2^0}{a_{21}^0} = \frac{180}{3} = 60,$$

$$b_3^1 = b_q^1 = b_q^0 - \frac{b_p^0 a_{qj}^0}{a_{pj}^0} = b_3^0 - \frac{b_2^0 a_{31}^0}{a_{21}^0} = 180 - \frac{180\cdot 1}{3} = 120.$$

$$c_1^1 = c_j^1 = -\frac{c_j^0}{a_{pj}^0} = -\frac{c_1^0}{a_{21}^0} = -\frac{-5}{3} = \frac{5}{3},$$

$$c_2^1 = c_l^1 = c_l^0 - \frac{c_j^0 a_{pl}^0}{a_{pj}^0} = c_2^0 - \frac{c_1^0 a_{22}^0}{a_{21}^0} = -4 - \frac{-5\cdot 1}{3} = -\frac{7}{3},$$

$$d^1 = d^0 - \frac{b_2^0 c_1^0}{a_{21}^0} = 0 - \frac{180\cdot(-5)}{3} = -300.$$

$T_1 =$

x_4	x_2	-1	
$-\frac{1}{3}$	$\frac{2}{3}$	20	$=-x_3$
$\frac{1}{3}$	$\frac{1}{3}$	60	$=-x_1$
$-\frac{1}{3}$	$\frac{8}{3}$	120	$=-x_5$
$\frac{5}{3}$	$-\frac{7}{3}$	300	$=z$

Sada je bazično dopustivo rešenje $x^1 = (60, 0, 20, 0, 120)$, kome odgovara vrednost ciljne funkcije $c^T x^1 = -300$. Testovi u Koracima 1 i 2 nisu zadovoljeni. Sada $j = 2$ i

$$b_1^1/a_{12}^1 = 30,\ b_2^1/a_{22}^1 = 180,\ b_3^1/a_{32}^1 = 45,$$

odakle proizilazi $p = 1$. Znači, potrebno je da se izvrši zamena $x_2 \leftrightarrow x_3$. Primenom Algoritma

Replace dobija se sledeći niz transformacija kao i odgovarajuća tabela T_2:

$$a_{12}^2 = a_{pj}^2 = 1,$$

$$a_{11}^2 = a_{pl}^2 = \frac{a_{pl}^1}{a_{pj}^1} = \frac{a_{11}^1}{a_{12}^1} = -\frac{1}{2},$$

$$a_{22}^2 = a_{qj}^2 = -\frac{a_{qj}^1}{a_{pj}^1} = -\frac{a_{22}^1}{a_{12}^1} = -\frac{1}{2},$$

$$a_{32}^2 = a_{qj}^2 = -\frac{a_{qj}^1}{a_{pj}^1} = -\frac{a_{32}^1}{a_{12}^1} = -4,$$

$$a_{21}^2 = a_{ql}^2 = a_{ql}^1 - \frac{a_{pl}^1 a_{qj}^1}{a_{pj}^1} = a_{21}^1 - \frac{a_{11}^1 a_{22}^1}{a_{12}^1} = \frac{7}{6},$$

$$a_{31}^2 = a_{ql}^2 = a_{ql}^1 - \frac{a_{pl}^1 a_{qj}^1}{a_{pj}^1} = a_{31}^1 - \frac{a_{11}^1 a_{32}^1}{a_{12}^1} = 1.$$

$$b_1^2 = b_p^2 = \frac{b_p^1}{a_{pj}^1} = \frac{b_1^1}{a_{12}^1} = \frac{20}{2/3} = 30,$$

$$b_2^2 = b_q^2 = b_q^1 - \frac{b_p^1 a_{qj}^1}{a_{pj}^1} = b_2^1 - \frac{b_1^1 a_{22}^1}{a_{12}^1} = 60 - \frac{20 \cdot 1/3}{2/3} = 50,$$

$$b_3^2 = b_q^2 = b_q^1 - \frac{b_p^1 a_{qj}^1}{a_{pj}^1} = b_3^1 - \frac{b_1^1 a_{32}^1}{a_{12}^1} = 120 - \frac{20 \cdot 8/3}{2/3} = 40.$$

$$c_2^2 = c_j^2 = -\frac{c_j^1}{a_{pj}^1} = -\frac{c_2^1}{a_{11}^1} = \frac{7}{2},$$

$$c_1^2 = c_l^2 = c_l^1 - \frac{c_j^1 a_{pl}^1}{a_{pj}^1} = c_1^1 - \frac{c_2^1 a_{11}^1}{a_{12}^1} = \frac{5}{3} - \frac{-7/3 \cdot (-1/3)}{2/3} = \frac{1}{2},$$

$$d^2 = d^1 - \frac{b_1^1 c_2^1}{a_{12}^1} = 300 - \frac{20 \cdot (-7/3)}{2/3} = 370.$$

$T_2 =$

x_4	x_3	-1	
$-\frac{1}{2}$	$\frac{3}{2}$	30	$=-x_2$
$\frac{1}{2}$	$-\frac{1}{2}$	50	$=-x_1$
1	-4	40	$=-x_5$
$\frac{1}{2}$	$\frac{7}{2}$	370	$=z$

Bazično dopustivo rešenje je $x^2 = (50, 30, 0, 0, 40)$, a ciljna funkcija uzima vrednost $c^T x^2 = -370$. Kako je test u Koraku 1 zadovoljen, to je x^2 optimalno rešenje, a vrednost ciljne funkcije je $z = -370$.

Primetimo da svakoj simpleks tablici odgovara jedna ekstremna tačka skupa Ω_P, tako da simpleks metod predstavlja "šetnju" po temenima skupa Ω_P. Kako je $d^{k+1} < d^k$, to se nijedan vrh ne može ponoviti. Kako je broj vrhova konačan, sledi da posle konačno mnogo iteracija dolazimo ili do optimalnog rešenja ili do zaključka da ciljna funkcija nije ograničena odozdo.

Napomena 2.4.2 *Može se dogoditi više od jedne vrednosti p za koju je razlomak $\frac{b_p^k}{a_{pj}^k}$ najmanji. Tada se može izabrati proizvoljno p. U sledećim poglavljima reći ćemo nešto više o ovome.*

Teorema 2.4.2 *Ako je* $c_1^k, \ldots, c_n^k \leq 0$ *ciljna funkcija ima ekstremum u tački* $x_{N,i} = 0$, $i = 1, \ldots, n$.

Dokaz. Neka su $x_{N,1}, \ldots, x_{N,n}$ proizvoljni nenegativni brojevi. Tada važi

$$z(x) = \sum_{i=1}^{n} c_i^k x_{N,i} - d^k \leq -d^k$$

jer je $c_i^k x_{N,i} \leq 0$ za svako $i = 1, \ldots, n$. Jednakost važi ako i samo ako je $x_{N,i} = 0$, $i = 1, \ldots, N$. □

Teorema 2.4.3 *Neka je za neko* $j \in \{1, \ldots, n\}$ *ispunjeno* $c_j^k > 0$ *i* $a_{ij}^k \leq 0$ *za svako* $i = 1, \ldots, m$, *tada je ciljna funkcija neograničena.*

Dokaz. Ako je $a_{ij}^k \leq 0$ za svako $i = 1, \ldots, m$, s obzirom na ograničenja

$$a_{i1}^k x_{N,1} + \cdots + a_{ij}^k x_{N,j} + \cdots + a_{in}^k x_{N,N} - b_i^k = -x_{B,i},\ i = 1, \ldots, m,$$

promenljiva $x_{N,j}$ se može uvećati do beskonačnosti, a da nijedna zavisna promenljiva $x_{B,i}$ ne bude negativna. Kako je $c_j^k > 0$, to se sa povećanjem promenljive x_j povećava i vrednost ciljne funkcije. To znači da je ciljna funkcija neograničena. □

U opštem slučaju je $c_j^k > 0$ za neko j i $a_{ij}^k > 0$ za neko i. Pokazaćemo da tabela (2.4.4) može da se transformiše u ekvivalentnu, pri čemu su jedna nezavisna i zavisna promenljiva zamenile mesta. Izaberimo p tako da bude:

$$\frac{b_p^k}{a_{pj}^k} = \min_{1 \leq i \leq m} \left\{ \frac{b_i^k}{a_{ij}^k} |\ a_{ij}^k > 0 \right\}. \tag{2.4.15}$$

Promenljivu $x_{N,j}$ možemo zameniti iz jednačine

$$x_{B,p} = b_p^k - (a_{p1}^k x_{N,1} + \cdots + a_{pj}^k x_{N,j} + \cdots + a_{pn}^k x_{N,n})$$

i zameniti je u svim ostalim jednačinama i u funkciji cilja. Na ovaj način dobijamo sistem ekvivalentan sistemu (2.4.2) pri čemu je sada $x_{B,p}$ nezavisna a $x_{N,j}$ zavisna promenljiva. I dalje su slobodni članovi b_i u svim jednačinama pozitivni dok c_j^k postaje nepozitivno. Dokaz ovog tvrđenja (sledeća teorema) kao i algoritam za transformaciju sistema jednačina (2.4.2) (za zamenu promenljivih $x_{B,p}$ i $x_{N,j}$) biće pokazani u nastavku. Sada primenjujemo isti postupak sve dok je i jedan od brojeva c_j pozitivan. Kada su svi c_j^k nepozitivni, dobijeno dopustivo rešenje $x_{N,1} = \cdots = x_{N,n} = 0$ je optimalno, tj ciljna funkcija dostiže ekstremum.

Teorema 2.4.4 *Ako je* $b_1^k, \ldots, b_m^k \geq 0$ *i ako je indeks* p *odabran prema kriterijumu* (3.3.15) *i* $c_j^k > 0$, *tada i posle zamene mesta promenljivih* $x_{B,p}$ *i* $x_{N,j}$ *važi* $b_1^{k+1}, \ldots, b_n^{k+1} \geq 0$. *Takođe je i* $c_j^{k+1} < 0$.

Dokaz. Prema algoritmu za zamenu promenljivih $x_{B,p}$ i $x_{N,j}$ posle transformacije je

$$b_q^{k+1} = \frac{a_{pj}^k b_q^k - a_{qj}^k b_p^k}{a_{pj}^k}.$$

Ako je $a_{qj}^k \geq 0$, s obzirom na $\frac{b_p^k}{a_{pj}^k} \leq \frac{b_q^k}{a_{qj}^k}$ dobijamo

$$b_q^{k+1} = \frac{a_{pj}^k b_q^k - a_{qj}^k b_p^k}{a_{pj}^k} \geq \frac{a_{pj}^k b_q^k - a_{pj}^k b_q^k}{a_{pj}^k} = 0.$$

Pretpostavimo sada da je $a_{qj}^k < 0$. S obzirom na $-\frac{a_{qj}^k b_p^k}{a_{pj}^k} > 0$, dobijamo

$$b_q^{k+1} = \frac{a_{pj}^k b_q^k - a_{qj}^k b_p^k}{a_{pj}^k} > b_q^k > 0.$$

Posle smene dobijamo

$$c_j^{k+1} = -\frac{c_j^k}{a_{pj}^k} < 0,$$

čime je teorema dokazana. □

Element a_{pj} nazvaćemo ključnim (pivot) elementom a zamenu promenljivih $x_{B,p}$ i $x_{N,j}$ nazvaćemo zamenom pomoću ključnog elementa a_{pj}^k.

Primer 2.4.2 Rešiti sledeći linearni problem:

$$\begin{array}{ll} \max & L = 7x_1 + 5x_2 \\ \text{p.o.} & 2x_1 + 3x_2 + x_3 = 19 \\ & 2x_1 + x_2 + x_4 = 13 \\ & 3x_2 + x_5 = 15 \\ & 3x_1 + x_6 = 18. \end{array}$$

Rešenje. Rang matrice sistema i proširene matrice

$$\left[\begin{array}{ccccccc} 2 & 3 & 1 & 0 & 0 & 0 & 19 \\ 2 & 1 & 0 & 1 & 0 & 0 & 13 \\ 0 & 3 & 0 & 0 & 1 & 0 & 15 \\ 3 & 0 & 0 & 0 & 0 & 1 & 18 \end{array}\right]$$

jednak je 4. To znači da možemo uzeti 4 promenljive za bazne (na primer x_3, x_4, x_5, x_6), a dve promenljive (x_1, x_2) za nezavisne (slobodne).

$$\begin{aligned} x_3 &= 19 - 2x_1 - 3x_2 \\ x_4 &= 13 - 2x_1 - x_2 \\ x_5 &= 15 - 3x_2 \\ x_6 &= 18 - 3x_1 \end{aligned}$$

Osim toga, na osnovu ciljne funkcije zaključujemo

$$L - 7x_1 - 5x_2 = 0.$$

Početna simpleks tabela ima oblik

$x_{N,1}$	$x_{N,2}$	$x_{B,1}$	$x_{B,2}$	$x_{B,3}$	$x_{B,4}$	b_i	$x_{B,i}$
x_1	x_2	x_3	x_4	x_5	x_6		
2	3	1	0	0	0	19	x_3
2	1	0	1	0	0	13	x_4
0	3	0	0	1	0	15	x_5
3	0	0	0	0	1	18	x_6
-7	-5	0	0	0	0	0	L

Tabela 2.4.1.

U poslednjoj vrsti su negativni koeficijenti –7 i –5. Uzmimo najmanji negativni koeficijent, –7. Zatim u koloni za x_1 uočimo tri pozitivna elementa: 2,2,3. Podelimo tim brojevima odgovarajuće slobodne članove. Minimum tih količnika je

$$\min\left\{\frac{19}{2}, \frac{13}{2}, \frac{18}{3}\right\} = \frac{18}{3}.$$

U preseku vrste za x_6 i kolone za x_1 nalazi se broj 3. Znači, promenljive x_1 i x_6 zamenjuju uloge. Novu bazu čine x_3, x_4, x_5, x_1, dok su nezavisne promenljive x_2, x_6. Da bismo na mestu preseka dobili 1, podelićemo posmatranu vrstu brojem 3. Ostalim vrstama dodajemo podeljenu vrstu, prethodno umnoženu brojem tako da u koloni koja odgovara x_2, ispod i iznad vodećeg elementa dobijemo nule.

$x_{B,4}$	$x_{N,2}$	$x_{B,1}$	$x_{B,2}$	$x_{B,3}$	$x_{N,1}$	b_i	$x_{B,i}$
x_1	x_2	x_3	x_4	x_5	x_6		
0	3	1	0	0	-2/3	7	x_3
0	1	0	1	0	-2/3	1	x_4
0	3	0	0	1	0	15	x_5
1	0	0	0	0	1/3	6	x_1
0	-5	0	0	0	7/3	42	L

Tabela 2.4.2.

Isti postupak se nastavlja na novoj tabeli. U poslednjoj vrsti negativan je koeficijent –5. U koloni za x_2 uočimo dva pozitivna elementa: 3,1,3. Podelimo tim brojevima odgovarajuće slobodne članove. Minimum tih količnika je

$$\min\left\{\frac{7}{3}, \frac{1}{1}, \frac{15}{3}\right\} = \frac{1}{1}.$$

U preseku vrste za x_4 i kolone za x_2 nalazi se broj 1. Sada promenljive x_2 i x_4 zamenjuju uloge. Novu bazu čine x_3, x_2, x_5, x_1, dok su nezavisne promenljive x_4, x_6. Ostalim vrstama dodajemo drugu vrstu, prethodno umnoženu brojem tako da u koloni x_2 ispod i iznad vodećeg elementa dobijemo nule.

$x_{B,4}$	$x_{B,2}$	$x_{B,1}$	$x_{N,2}$	$x_{B,3}$	$x_{N,1}$	b_i	$x_{B,i}$
x_1	x_2	x_3	x_4	x_5	x_6		
0	0	1	-3	0	4/3	4	x_3
0	1	0	1	0	-2/3	1	x_2
0	0	0	-3	1	2	12	x_5
1	0	0	0	0	1/3	6	x_1
0	0	0	5	0	-1	47	L

Tabela 2.4.3.

Postupak se ponavlja na poslednjoj tabeli.

$$\min\left\{\frac{4}{4/3}, \frac{12}{2}, \frac{6}{1/3}\right\} = \frac{4}{4/3}$$

Sada promenljive x_6 i x_3 zamenjuju uloge. Novu bazu čine x_6, x_2, x_5, x_1, dok su nezavisne promenljive x_3, x_4.

$x_{B,4}$	$x_{B,2}$	$x_{N,1}$	$x_{N,2}$	$x_{B,2}$	$x_{B,4}$	b_i	$x_{B,i}$
x_1	x_2	x_3	x_4	x_5	x_6		
0	0	3/4	-9/4	0	1	3	x_6
0	1	1/2	-1/2	0	0	3	x_2
0	0	-2/3	3/2	1	0	6	x_5
1	0	-1/4	3/4	0	0	5	x_1
0	0	3/4	11/4	0	0	50	L

Tabela 2.4.4.

Nova vrsta za L nema negativnih koeficijenata. Maksimalna vrednost ciljne funkcije je $L_{max}=50$, a dobija se za bazu $\{5, 3, 0, 0, 6, 3\}$.

Napomena 2.4.3 *Ukoliko u u nekoj iteraciji postoji više jednakih minimalnih količnika b_i/a_{ip}, potrebno je izabrati onu vrstu za koju je odnos elemenata sledeće kolone u odnosu na elemente one kolone koju koristimo za rešavanje najmanji. Ovaj postupak se ponavlja dok ne dođemo do jednoznačnog minimalnog elementa b_j/a_{jp}.*

Još jedan oblik simpleks tabela

Simpleks model problema linearnog programiranja tipa maksimuma je sistem nejednačina oblika:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m &\leq b_1 \\ a_{21}x_2 + a_{22}x_2 + \cdots + a_{2m}x_m &\leq b_2 \\ &\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m &\leq b_n \end{aligned}$$

Ovaj sistem nejednačina prevodimo u sistem jednačina, uvođenjem dopunskih promenljivih, na sledeći način:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m + x_{m+1} &= b_1 \\ a_{21}x_2 + a_{22}x_2 + \cdots + a_{2m}x_m + x_{m+2} &= b_2 \\ &\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m + x_{m+n} &= b_n \end{aligned}$$

Funkcija kriterijuma ima oblik:

$$F(x) = \sum_{j=1}^{n} c_j x_j + 0 \cdot (c_{n+1}x_{n+1} + \cdots + c_{n+m}x_{n+m}) .$$

Na osnovu ovakvog modela, postavljamo nultu simpleks tabelu oblika:

max ST - 0

Bazne promenljive			Slobodne Promenljive								
C			c_1	c_2	...	c_n	0	0	0	0	b_j/a_{ij}
C_b	X_b	B	X_1	X_2	...	X_n	X_{n+1}	X_{n+2}	...	X_{n+m}	
0	x_{n+1}	b_1	a_{11}	a_{12}	...	a_{1n}	1	0	...	0	
0	x_{n+2}	b_2	a_{21}	a_{22}	...	a_{2n}	0	1	...	0	
...	...	...	...	...	...	...	0	0	...	0	
0	x_{n+m}	b_m	a_{m1}	a_{m2}	...	a_{mn}	0	0	...	1	
$F_j - c_j$		0	F_1-c_1	F_2-c_2		F_n-c_n	0	0	0	0	$\theta(c_j - F_j)$

gde su:

C - vektor koeficijenata c_i uz promenljive x_i funkcije kriterijuma, $i = 1, \ldots, n$.

C_b - vektor koeficijenata u funkciji kriterijuma uz promenljive koje sačinjavaju bazično dopustivo rešenje. Kod max ST - 0 vrednost ovih koeficijenata je 0,

X_b - vektor promenljivih bazično dopustivog rešenja.

B - vektor vrednosti promenljivih bazno dopustivog rešenja za posmatranu iteraciju,

X_j - množitelji vektora baze.

$F_j - c_j$ - kriterijum optimalnosti. Na osnovu ovog kriterijuma se odlučuje da li je dobijeno rešenje maksimalno. Rešenje je maksimalno ako je ispunjen uslov

$$(\forall j)\ F_j - c_j \geq 0.$$

U slučaju da rešenje nije optimalno, ovim kriterijumom se određuje vektor koji ulazi u bazu, a to je vektor x_j koji ispunjava uslov:

$$\min_j \{F_j - c_j |\ F_j - c_j < 0\}.$$

S druge stane, na osnovu "teta" kriterijuma se određuje koji vektor izlazi iz baze, tj:

$$\theta = \min_i \left\{ \frac{b_i}{a_{ij}}, \quad a_{ij}^k > 0 \right\} = \frac{b_p}{a_{pj}}.$$

Procedura za nalaženje optimalnog rešenja simpleks metodom

Posmatra se opšti problem linearnog programiranja. Potrebno je da se pronađu nenegativne vrednosti promenljivih $x_1, \ldots, x_k$ koje će ispunjavati sistem ograničenja dat linearnim nejednačinama i jednačinama 1.1.1, i obezbediti maksimalnu vrednost kriterijumske funkcije

$$z_0 = \sum_{j=1}^{k} c_j x_j.$$

Ovako formulisani problem linearnog programiranja rešavamo simpleks metodom prema sledećoj proceduri:

1) Sve nejednačine sistema prevodimo u jednačine. Ako je nejednačina tipa "$\leq$", onda dodajemo odgovarajuću promenljivu; ako je nejednačina tipa "$\geq$", onda odgovarajuću izravnavajuću promenljivu oduzimamo od leve strane nejednačine. Uz izravnavajuće promenljive u funkciji kriterijuma uvek stoje nule.

Veštačke promenljive se dodaju svim jednačinama i nejednačinama tipa "$\geq$". Uz ove promenljive u funkciji kriterijuma stavljamo koeficijente M.

2) Od koeficijenata prilagođenog modela sastavljamo početnu simpleks tabelu. Bazu vektorskog prostora za početno rešenje čine jedinični vektori. To su vektori koji odgovaraju "dodatim" promenljivim (dodate izravnavajuće i sve veštačke).

3) Za dodatni red simpleks tabele računamo koeficijente $F_j - c_j$ za svako j. Simpleks tabela sadrži početno rešenje.

4) Proverimo koeficijente reda $(F_j - c_j)$:
– Ako je $F_j - c_j \geq 0$ za svako j, onda tvrdimo da je pronađeno optimalno rešenje.
– Ako postoji $F_j - c_j < 0$, onda biramo

$$F_s - c_s = \left[\min_j (F_j - c_j)\right] < 0 \ .$$

Koeficijent $(F_s - c_s)$ odgovara vektoru A_s, pa određujemo da vektor A_s uđe u novu bazu vektora.

5) Koordinate vektora B delimo odgovarajućim pozitivnim koordinatama vektora A_s i određujemo

$$\theta = \frac{b_r}{a_{rs}} = \min_i \frac{b_i}{a_{is}} \quad (a_{is} > 0) \ .$$

Najmanja vrednost od ovih količnika odgovara vektoru A_r, pa određujemo da vektor A_r treba da izađe iz početne baze.

6) Koeficijente za novu simpleks tabelu izračunavamo na osnovu koeficijenata iz početne simpleks tabele prema opisanim pravilima transformacije.

7) Vraćamo se na tačku 4) iz ove procedure. Svaka naredna iteracija počinje tačkom 4) i ponavlja postupak do tačke 7), sve dok se u tački 4) ne utvrdi da su sve razlike $F_j - c_j \geq 0$. Tada se postupak završava, pronađeno je optimalno rešenje linearnog problema.

Primer 2.4.3 Rešiti sistem nejednačina primenom simpleks metode, ako su date sledeća funkcija kriterijuma i ograničenja:

$$\begin{array}{ll} \max & F(X) = 10x_1 + 12x_2 + 10x_3 \\ \text{p.o.} & 4x_1 + 5x_2 + 4x_3 \leq 4200 \\ & 2x_1 = 2x_2 + x_3 \leq 1500 \\ & 2x_1 + 3x_2 + 4x_3 \leq 2400 \end{array}$$

Na osnovu ovog modela formiraćemo polaznu simpleks tabelu, max ST-0.

$$\begin{aligned} \max \quad & F(X) = 10x_1 + 12x_2 + 10x_3 \\ \text{p.o.} \quad & 4x_1 + 5x_2 + 4x_3 + x_4 = 4200 \\ & 2x_1 = 2x_2 + x_3 + x_5 = 1500 \\ & 2x_1 + 3x_2 + 4x_3 + x_6 = 2400 \end{aligned}$$

max ST - 0

C			10	12	10	0	0	0	b_j / a_{i2}
C_b	X_b	B	X_1	X_2	X_3	X_4	X_5	X_6	
0	x_4	4200	4	5	4	1	0	0	840
0	x_5	1500	2	2	1	0	1	0	750
0	x_6	2400	2	3	4	0	0	1	800
$F_j - c_j$		0	-10	-12	-10	0	0	0	9000

max ST - 1

C			10	12	10	0	0	0	b_j / a_{i2}
C_b	X_b	B	X_1	X_2	X_3	X_4	X_5	X_6	
0	x_4	450	-1	0	1.5	1	-2.5	0	300
12	x_2	750	1	1	0.5	0	0.5	0	1500
0	x_6	150	-1	0	2.5	0	-1.5	1	60
$F_j - c_j$		9000	2	0	-4	0	6	0	240

max ST - 2

C			10	12	10	0	0	0
C_b	X_b	B	X_1	X_2	X_3	X_4	X_5	X_6
0	x_4	360	-0.4	0	0	1	-1.6	-0.6
12	x_2	720	1.2	1	0	0	0.8	-0.2
10	x_3	60	-0.4	0	1	0	-0.6	0.4
$F_j - c_j$		9240	0.4	0	0	0	3.6	1.6

Ova tabela sadrži optimalno rešenje, koje iznosi:

$$X^* = [0, 720, 60, 360, 0, 0]^T, \quad F(X^*) = F_{\max} = 9240.$$

2.5 Određivanje početnog bazično dopustivog rešenja

Primena Algoritma *SimpleksbasicMax* podrazumeva da je poznato jedno bazično dopustivo rešenje. To je ispunjeno ako su sva ograničenja tipa $\leq$ i ako je $b_i \geq 0$, $i = 1, \ldots, m$, odnosno ako su sva ograničenja tipa $\geq$ i ako je $b_i \geq 0$, $i = 1, \ldots, m$. Dodavanjem izravnavajućih promenljivih odmah formiramo i bazično dopustivo rešenje. Međutim, u opštem slučaju bazično dopustivo rešenje nije poznato. Dokazaćemo teoremu na osnovu koje je moguće odrediti bazično dopustivo rešenje ukoliko je skup dopustivih rešenja problema linearog programiranja neprazan.

Do sada je opisan algoritam (Algoritam *SimpleksBasicMax*/*SimpleksBasicMin*) za nalaženje optimalnog rešenja, ako postoji, u slučajevima kada se u zadacima

osnovnog oblika traži maksimum/minimum funkcije cilja i kada su početne maksimizacione/minimizacione tabele bazično dopustive. Šta učiniti ako početna tabela osnovnog zadatka linearnog programiranja nije bazično dopustiva? Pre nego što možemo da primenimo simpleks algoritam moramo najpre tabelu maksimizacije da transformišemo na maksimizacionu tabelu sa bazično dopustivim rešenjem. Tako je u opisanom algoritmu potrebno prethodno uvesti korake ovakve transformacije. Razmotrimo na kraju simpleks algoritam za maksimizacionu tabelu u opštem slučaju.

Definicija 2.5.1 *Osnovni zadatak linearnog programiranja sa maksimizacijom je nedopustiv ako nema dopustiva rešenja.*

Teorema 2.5.1 *Ako je* $a_{i1}^k, \ldots, a_{in}^k \geq 0$ *i* $b_i^k < 0$ *tada i-ta jednačina u sistemu* (2.3.2) *nema rešenja, tj. sistem je nedopustiv.*

Dokaz. Uz navedene pretpostavke dobijamo

$$-x_{B,i} = a_{i1}^k x_{N,1} + \cdots a_{in}^k x_{B,N} - b_i^k \geq -b_i^k > 0,$$

odnosno $x_{B,i} < 0$, što predstavlja kontradikciju sa polaznom predpostavkom da su sve promenljive nenegativne. □

U ovom odeljku razmotrićemo problem pronalaženja prvog bazično dopustivog rešenja (kanonskog oblika). Najčešće su u upotrebi dve vrste metoda kojima se rešava ovaj problem. Jedna grupa metoda se nazivaju *dvofazni simpleks metodi* (two-phase simplex method), a drugu vrstu čine *BigM metodi* (BigM methods). Ovde su opisana tri pristupa. Dva pristupa su tzv. dvofazni simpleks metodi. Jedan metod zahteva uvođenje veštačkih promenljivih i samim tim povećava dimenzije problema, dok drugi ne zahteva primenu veštačkih promenljivih. Treći metod kombinuje ove dve faze u jednu, i naziva se *BigM* metod.

2.6 Dvofazni simpleks metodi

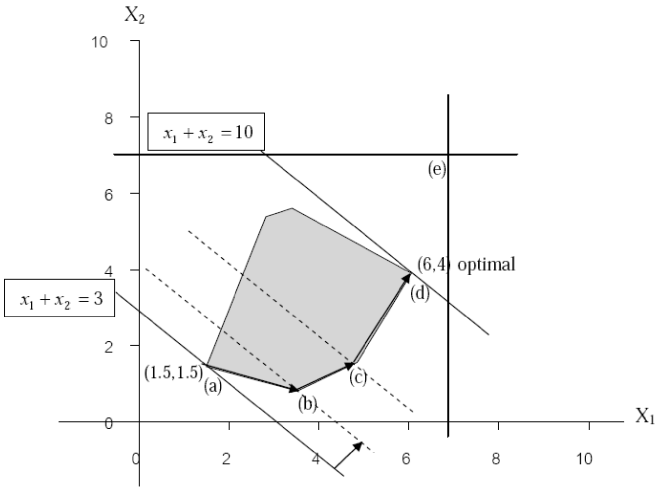
Dvofazni simpleks metod se sastoji iz dve faze, faze I i faze II. Faza I pokušava da pronađe neko početno bazično dopustivo rešenje (initial basic feasible solution). Kada je početno bazično dopustivo rešenje pronađeno, tada se primenjuje faza II da bi se pronašlo optimalno rešenje. Simpleks metod je iterativna procedura čija se svaka iteracija karakteriše određivanjem m bazičnih promenljivih $x_{B,1}, \ldots, x_{B,m}$ i n nebazičnih promenljivih $x_{N,1}, \ldots, x_{N,n}$.

Geometrijski, simpleks metod se kreće iz jedne ekstremne tačke (ugla) skupa dopustivih rešenja u drugu i pri tome poboljšava vrednosti ciljne funkcije u svakoj iteraciji. Dvofazni simpleks metod prolazi kroz dve faze, faza I i faza II. Faza I pokušava sa nađe inicijalnu ekstremnu tačku. Kada se inicijalna ekstremna tačka jedanput pronađe, primenjuje se faza II da bi se rešio originalni LP.

Primer 2.6.1 Za sledeći problem

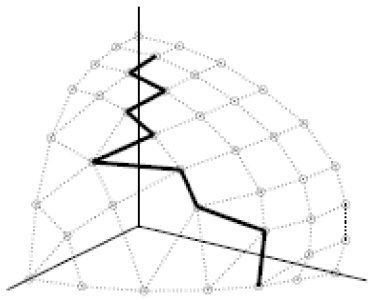
$$\begin{aligned} \min \quad & -x_1 - x_2 \\ \text{p.o.} \quad & 2x_1 + 3x_2 \leq 24,\ 2x_1 - x_2 \leq 8,\ x_1 - 2x_2 \leq 2, \\ & -x_1 + 2x_2 \leq 8,\ x_1 + 3x_2 \geq 6, 3x_1 - x_2 \geq 3, \\ & 0 \leq x_1 \leq 7,\ 0 \leq x_2 \leq 7 \end{aligned}$$

faza I simpleks algoritma se završava u ekstremnoj tački koja je označena sa (a) na sledećoj slici. Tada faza II sledi sekvencu ekstremnih tačaka koje su označene strelicama duž ivica skupa dopustivih rešenja. Optimalna ekstremna tačka je označena sa (d).



Slika 2.6.1. Geometrija simpleks metoda

Na sledećoj slici je prikazan trodimenzionalni slučaj.



Slika 2.6.2. Temena i simpleks metod u R^3

Ako je $\mathbf{b} \geq 0$ i ako su sve nebazične promenljive $x_{N,1}, \ldots, x_{N,n}$ jednake nuli, tada je $x_{B,1} = b_1, \ldots, x_{B,m} = b_m$ bazično dopustivo rešenje. Ako uslov $\mathbf{b} \geq 0$ nije ispunjen, neophodno je da se nađe početno bazično dopustivo rešenje ili da se odredi da ono ne postoji. Postoji više strategija za fazu I.

Dvofazni simpleks metod koji koristi veštačke promenljive

Klasični pristup se sastoji u tome da se linearnom programu u standardnom obliku pridruži tzv. *prošireni problem* [5, 17, 33].

Neka je dat problem linearog programiranja u standardnom obliku

$$\begin{array}{ll} \min & c^T x, \\ \text{p.o.} & Ax = b, \\ & x \geq 0. \end{array} \tag{2.6.1}$$

Jasno je da bez umanjenja opštosti možemo pretpostaviti da je u standardnom obliku $b \geq 0$ (u suprotnom pomnožimo odgovarajuće jednačine sa -1). Problemu (2.6.1) pridružimo pomoćni problem linearnog programiranja

$$\begin{array}{ll} \min & e^T w, \\ \text{p.o.} & Ax + w = b, \\ & x \geq 0,\ w \geq 0, \end{array} \tag{2.6.2}$$

gde je $e = (1, \ldots, 1) \in \mathbb{R}^m$ i $w \in \mathbb{R}^m$ je vektor tzv. *veštačkih* promenljivih. Važna činjenica je da je skup dopustivih rešenja problema (2.6.2) neprazan jer mu sigurno pripada tačka ($x = 0$, $w = b$). Jasno je takođe da je ciljna funkcija na tom skupu odozdo ograničena sa nulom. Dopustivu bazu čine kolone koje odgovaraju promenljivim $w_1, \ldots, w_m$, a kanonski oblik problema (2.6.2) se dobija eliminacijom w iz ciljne funkcije pomoću jednačine $w = b - Ax$. Problemi (2.6.1) i (2.6.2) su povezani sledećom teoremom:

Teorema 2.6.1 *Skup dopustivih rešenja problema* (2.6.1) *je neprazan ako i samo ako je optimalna vrednost ciljne funkcije problema* (2.6.2) *jednaka nuli.*

Dokaz. Neka je $\bar{x}$ dopustivo rešenje problema (2.6.1). Tada je $(\bar{x}, 0)$ dopustivo rešenje problema (2.6.2), pri čemu je vrednost ciljne funkcije jednaka nuli. S obzirom da je nula donja granica za ciljnu funkciju problema (2.6.2), sledi da je $(\bar{x}, 0)$ optimalno rešenje i da je nula optimalna vrednost ciljne funkcije tog problema.

Pretpostavimo sada da je $(\bar{x}, \bar{w})$ optimalno rešenje problema (2.6.2) i neka je $e^T \bar{w} = 0$. Iz $e > 0$, $\bar{w} \geq 0$ sledi da je $\bar{w} = 0$, pa je $A\bar{x} = b$, tj. $\bar{x}$ je dopustivo rešenje problema (2.6.2). □

Na prethodnoj teoremi se zasniva tzv. *dvofazna modifikacija simpleks metoda.*

Algoritam 2. (Dvofazna modifikacija simpleks metoda).

I Faza:

Za problem (2.6.1) se formira problem (2.6.2) kome se pridružuje LP-tablica. Eliminacijom jedinica iz nulte vrste, koje odgovaraju veštačkim promenljivim, LP-tablica se svodi na simpleks tablicu, a zatim se primenjuje Algoritam *SimpleksBasic*. Kako je ciljna funkcija ograničena odozdo, moguća su dva slučaja:

1) Optimalna vrednost je veća od nule. Po Teoremi 2.6.1 problem (2.6.1) nema dopustivih rešenja i postupak se završava.

2) Optimalna vrednost je nula. Tada su u optimalnom bazično dopustivom rešenju sve veštačke promenljive jednake nuli. Prelazi se na drugu fazu.

II Faza:

Korak 1. Iz poslednje simpleks tablice uklanjaju se sve nebazične kolone koje odgovaraju veštačkim promenljivim, nulta vrsta se zamenjuje vrstom $[0|\, c_1 \ldots c_n\, 0 \ldots 0]$ koja ima $n + k + 1$ element, pri čemu je k broj bazičnih veštačkih promenljivih. Dobijena LP-tablica se svodi na simpleks tablicu eliminacijom onih $c_j \neq 0$ koji odgovaraju bazičnim promenljivim.

Korak 2. Ukoliko je $k = 0$ ići na Korak 3. Ako je $k > 0$ u simpleks tablici postoje bazične kolone koje odgovaraju veštačkim promenljivim. Uočimo bazičnu kolonu koja odgovara veštačkoj promenljivoj w_s. Neka ta kolona sadrži jedinicu u vrsti v. Moguća su dva slučaja:

1) Svi elementi vrste v osim bazične jedinice su jednaki nuli. Tada se vrsta v i kolona koja odgovara promenljivoj w_s izostavljaju iz simpleks tablice.

2) Neka je osim bazične jedinice, recimo, $r-$ti element vrsta v različit od nule. Očigledno je $r > 0$ jer se u nultoj koloni nalazi veštačka promenljiva w_s koja je jednaka nuli, i da $r-$ta kolona ne odgovara veštačkoj promenljivoj, jer su izostavljene sve kolone koje odgovaraju nebazičnim promenljivim. Stožernom transformacijom sa stožernim elementom a_{vs}, učiniti $r-$tu kolonu bazičnom i zatim izostaviti kolonu koja odgovara promenljivoj w_s jer je sada to nebazična kolona koja odgovara veštačkoj promenljivoj. Zameniti k sa $k - 1$ i ponoviti korak 2.

Korak 3. Dobijena simpleks tablica sadrži samo kolone koje odgovaraju promenljivim iz problema (2.6.1). Primeniti Algoritam 1.

Primetimo da se Algoritam 2 očigledno završava u konačnom broju koraka.

Primer za ilustraciju Algoritma 2 je preuzet iz [17].

Primer 2.6.2 Odrediti rešenje sledećeg problema:

$$\begin{aligned} \min \quad & x_1 - x_2 + x_3, \\ \text{p.o.} \quad & x_1 + x_2 + 2x_3 + x_4 = 3, \\ & -x_2 - x_3 + x_4 = 3, \\ & x_1 - x_2 + 3x_4 = 9, \\ & x \geq 0. \end{aligned}$$

Kako nije poznato bazično dopustivo rešenje, primenićemo dvofaznu modifikaciju simpleks algoritma.

I Faza.

Pridruženi problem je:

$$\begin{array}{lrrrrrrrrr} \min & w_1 + w_2 + w_3, & & & & & & & & \\ & x_1 & +x_2 & +2x_3 & +x_4 & +w_1 & & & = & 3, \\ \text{p.o.} & & -x_2 & -x_3 & +x_4 & & +w_2 & & = & 3, \\ & x_1 & -x_2 & & +3x_4 & & & +w_3 & = & 9, \\ & x \geq 0, & w \geq 0, & & & & & & & \end{array}$$

i odgovara mu sledeća LP-tablica:

$$\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 3 & 1 & 1 & 2 & 1 & 1 & 0 & 0 \\ 3 & 0 & -1 & -1 & 1 & 0 & 1 & 0 \\ 9 & 1 & -1 & 0 & 3 & 0 & 0 & 1 \end{array}$$

odakle se eliminacijom jedinica iz nulte vrste (oduzimanjem svih ostalih vrsta od nulte) dobija simpleks tablica (u nastavku su pivot elementi uokvireni):

$$\begin{array}{cccccccc} -15 & -2 & 1 & -1 & -5 & 0 & 0 & 0 \\ 3 & \boxed{1} & 1 & 2 & 1 & 1 & 0 & 0 \\ 3 & 0 & -1 & -1 & 1 & 0 & 1 & 0 \\ 9 & 1 & -1 & 0 & 3 & 0 & 0 & 1 \end{array}$$

Primenom simpleks algoritma dobijamo novu simpleks tablicu

$$\begin{array}{cccccccc} -9 & 0 & 3 & 3 & -3 & -2 & 0 & 0 \\ 3 & 1 & 1 & 2 & 1 & 1 & 0 & 0 \\ 3 & 0 & -1 & -1 & 1 & 0 & 1 & 0 \\ 6 & 0 & -2 & -2 & \boxed{2} & -1 & 0 & 1 \end{array}$$

Kako u nultoj koloni postoje negativni elementi, ponovo primenjujemo simpleks algoritam:

$$\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 3/2 \\ 0 & 1 & 2 & 3 & 0 & 2 & 0 & -1/2 \\ 0 & 0 & 0 & 0 & 0 & -1/2 & 1 & -1/2 \\ 3 & 0 & -1 & -1 & 1 & -1/2 & 0 & 1/2 \end{array}$$

Kako je optimalna vrednost pomoćnog problema $\min e^T w = 0$, prelazimo na drugu fazu.

II Faza. Uklanjanjem nebazičnih kolona koje odgovaraju veštačkim promenljivim (peta i sedma) i zamenom koeficijenata nulte vrste odgovarajućim koeficijentima polaznog problema dobija se LP-tablica

$$\begin{array}{cccccc} 0 & 1 & -1 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 3 & 0 & -1 & -1 & 1 & 0 \end{array}$$

odakle se, posle eliminacije jedinice u nultoj vrsti prve kolone, dobija simpleks tablica

$$\begin{array}{cccccc} 0 & 0 & -3 & -2 & 0 & 0 \\ 0 & 1 & \boxed{2} & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 3 & 0 & -1 & -1 & 1 & 0 \end{array}$$

Sada eliminišemo drugu vrstu jer su u njoj svi elementi (osim koeficijenta koji odgovara veštačkoj promenljivoj) jednaki nuli i eliminišemo petu kolonu jer su svi elementi jednaki nuli. Dobijamo simpleks tablicu iz koje su eliminisane sve veštačke promenljive:

$$\begin{array}{ccccc} 0 & 0 & -3 & -2 & 0 \\ 0 & 1 & \boxed{2} & 3 & 0 \\ 3 & 0 & -1 & -1 & 1 \end{array}$$

Sada su ispunjeni svi uslovi za primenu Algorima 1. U sledećem koraku se dobija optimalno rešenje

$$\begin{array}{ccccc} 0 & 3/2 & 0 & 5/2 & 0 \\ 0 & 1/2 & 1 & 3/2 & 0 \\ 3 & 1/2 & 0 & 1/2 & 1 \end{array}$$

Druga varijanta dvofaznog simpleks metoda je opisana u [51] i [79]. Ovaj algoritam ima prednost u odnosu na prošireni problem zato što ne koristi veštačke promenljive.

Dvofazni simpleks metod bez veštačkih promenljivih

Izložićemo sada jednu verziju algoritma za određivanje početnog bazično dopustivog rešenja koja ne zahteva uvođenje veštačkih promenljivih. Osnovni nedostatak prethodnog metoda jeste povećanje dimenzije linearnog problema. Razmatramo standardni oblik problema linearnog programiranja bez ograničenja o znaku koeficijenata b_i u kome nije poznato bazično dopustivo rešenje. Egzistencija dopustivog rešenja sledi iz sledeće leme:

Lema 2.6.1 *Neka je za neko i koeficijent $b_i = a_{i0}$ negativan. Ako je $a_{ij} \geq 0$, $j = 1, \ldots, n$, tada je skup dopustivih rešenja prazan.*

Dokaz. S obzirom na uslove iz leme, u skupu ograničenja postoji jednačina u kojoj je zbir proizvoda nenegativnih bojeva negativan broj, što je nemoguće. □

Neka je skup rešenja problema linearnog programiranja dopustiv i neka je x bazično nedopustivo rešenje sa q negativnih koordinata. Neka je (novom numeracijom ako je potrebno) postignuto $x = (x_1, \ldots, x_m, 0, \ldots, 0)$ bazično nedopustivo rešenje takvo da je $x_1, \ldots, x_q < 0$, $q \leq m$, i $x_p \geq 0$ za $p > q$. Odgovarajuća baza je $K_1, \ldots, K_m$.

Ako je $q = m$, izaberemo za stožerni element $a_{ms} < 0$ (takvo postoji po Lemi (2.6.1) i primenom simpleks transformacije odmah dobijamo novo rešenje u kome je bar jedna koordinata pozitivna.

Ako je $q < m$, posmatramo $a_{rs} < 0$ za $r = 1, \ldots, q$ (postoje po Lemi (2.6.1). Ako postoje $r \in \{1, \ldots, q\}$ i $s \in \{1, \ldots, n\}$ takvi da je

$$\min_{h>q} \left\{ \frac{x_h}{a_{hs}} | \, a_{hs} > 0 \right\} \geq \frac{x_r}{a_{rs}}, \tag{2.6.1}$$

izaberemo a_{rs} za stožerni element. Tada je

$$x^1 = \sum_{j \neq r} \left(x_j - \frac{x_r}{a_{rs}} a_{js} \right) K_j + \frac{x_r}{a_{rs}} K_s.$$

Za $j > q$ i $a_{js} < 0$ tivijalno važi $x_j - \dfrac{x_r}{a_{rs}} a_{js} \geq 0$. Za $a_{js} \geq 0$ i $j > q$ je

$$\frac{x_j}{a_{js}} \geq \frac{x_r}{a_{rs}} a_{js} \Leftrightarrow x_j - \frac{x_r}{a_{rs}} a_{js} \geq 0$$

i kako je $\dfrac{x_r}{a_{rs}} > 0$, očigledno je da x^1 ima najviše $q - 1$ negativnih koordinata.

Ako ne postoji r takvo da je uslov (2.6.1) ispunjen, neka su sada $r \notin \{1, \ldots, q\}$ i $s \in \{1, \ldots, n\}$ takvi da je

$$\min_{h>q} \left\{ \frac{x_h}{a_{hs}} | \, a_{hs} > 0 \right\} = \frac{x_r}{a_{rs}}. \tag{2.6.2}$$

Za tako određeno r i s postavimo a_{rs} za stožerni element. Primenom simpleks transformacije dobijamo novo rešenje x^1 u kome se dokazuje analogno prethodnom da su koordinate $x^1_{q+1}, \ldots, x^1_m$ ostale nenegativne. Kako je skup bazičnih rešenja konačan, (primenjujući pravila protiv cikliranja ukoliko je potrebno), posle konačno mnogo transformacija će biti ispunjen uslov (2.6.1). Na osnovu ovih razmatranja sledi konačnost sledećeg algoritma.

Algoritam 3.

(Rešavanje problema linearnog programiranja bez uvođenja veštačkih promenljivih).

Korak 1. Formiramo početnu LP-tablicu oblika

$$\begin{array}{rrcr} -d & c_1 & \cdots & c_n \\ b_1 & a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & & \vdots \\ b_m & a_{m1} & \cdots & a_{mn} \end{array}$$

Ukoliko je $b_i \geq 0$, $i = 1, \ldots, m$, primenimo Algoritam 1.

Korak 2. Neka je $b_{i_1}, , \ldots, b_{i_q} < 0$. Ukoliko je $a_{ij} \geq 0$ za neko $i \in \{i_1, \ldots, i_q\}$ i svako $j = 1, \ldots, n$, algoritam staje jer je skup ograničenja nedopustiv. U suprotnom nastavljamo.

Korak 3. Ukoliko je $q = m$ biramo $a_{ms} < 0$ za stožerni element.

Ako je $q < m$ i ako postoje $r \in \{i_1, \ldots, i_q\} = I$ i s takvi da je ispunjen uslov

$$\min_{h \notin I} \left\{ \frac{b_h}{a_{hs}} | \, a_{hs} > 0 \right\} \geq \frac{b_r}{a_{rs}}, \quad a_{rs} < 0,$$

izaberemo a_{rs} za stožerni element.

Ukoliko ne postoji r takvo da je prethodni uslov ispunjen, neka je sada r takvo da je

$$\min_{h \notin I} \left\{ \frac{b_h}{a_{hs}} | \, a_{hs} > 0 \right\} = \frac{b_r}{a_{rs}}.$$

Za tako određeno r i s postavimo a_{rs} za stožerni element.

Korak 4. Primenimo simpleks transformaciju i prelazimo na Korak 1.

Primer 2.6.3 Neka je početna LP-tablica

$$\begin{array}{rrcccc} 720 & 7 & 0 & 0 & 5 & 0 \\ -100 & -2 & 0 & 1 & -1 & 0 \\ -360 & \boxed{-8} & 0 & 0 & -3 & 1 \\ 180 & 3 & 1 & 0 & 1 & 0 \end{array}$$

Kako u nultoj koloni postoje negativni koeficijenti, primenimo Algoritam 3. Uslov (2.6.1) je ispunjen u prvoj koloni (stožerni elementi su uokvireni), i primenom simpleks transformacije dobijamo

$$\begin{array}{rccccr} 405 & 0 & 0 & 0 & 11/8 & 7/8 \\ -10 & 0 & 0 & 1 & -1/4 & \boxed{-1/4} \\ 45 & 1 & 0 & 0 & 3/8 & -1/8 \\ 45 & 0 & 1 & 0 & -1/8 & 3/8 \end{array}$$

U sledećem koraku ponovo primenimo Algoritam 3. jer je $a_{01} < 0$, i odmah dobijamo optimalno rešenje

$$\begin{array}{rrrrrr} 370 & 0 & 0 & 3/2 & 1/2 & 0 \\ 40 & 0 & 0 & -4 & 1 & 1 \\ 50 & 1 & 0 & -1/2 & 1/2 & 0 \\ 30 & 0 & 1 & 3/2 & -1/2 & 0 \end{array}$$

Dokazali smo da ako je $\Omega_P \neq \emptyset$ tada postoji bazično dopustivo rešenje. Uslov za egzistenciju optimalnog bazično dopustivog rešenja da je sledeća lema:

Lema 2.6.2 *Ako je dopustivi skup problema* (1.1.4) *neprazan i ako je ciljna funkcija na njemu ograničena odozdo, onda problem* (1.1.4) *ima bazično dopustivo optimalno rešenje.*

Dokaz. Kako je ciljna funkcija problema (1.1.4) ograničena odozdo, primena simpleks algoritma na početno bazično dopustivo rešenje se završava posle konačno mnogo koraka i to bazično dopustivim optimalnim rešenjem. □

Direktna posledica Leme 2.6.2 je da egzistencija optimalnog rešenja povalači egzistenciju bazično dopustivog optimalnog rešenja.

Algoritam SimpleksMax/SimpleksMin.

Algoritam simpleks metoda za maksimizacione/minimizacione tabele koje nisu bazično dopustive. Tekuća tabela je oblika (2.3.4)

Korak 1. Ako su $b_1^k, b_2^k, \ldots, b_m^k \geq 0$, prelazimo na Korak 5.
U suprotnom nastavljamo.

Korak 2. Biramo $b_i^k < 0$ (Na primer poslednje).

Korak 3. Ako $a_{i1}^k, a_{i2}^k, \ldots, a_{in}^k \geq 0$, STOP:
maksimizacioni zadatak je nedopustiv. (Ovaj slučaj ćemo detaljnije razmotriti malo kasnije).
U suprotnom nastavljamo.

Korak 4. Ako je $i = m$, biramo $a_{mj}^k < 0$, uzimamo za ključni element a_{mj}^k i prelazimo na Korak 1. Ako je $i < m$, biramo $a_{ij}^k < 0$ i izračunavamo

$$\min_{l>i} \left(\left\{ \frac{b_i^k}{a_{ij}^k} \right\} \bigcup \left\{ \frac{b_l^k}{a_{lj}^k}; a_{lj}^k > 0 \right\} \right) = \frac{b_p^k}{a_{pj}^k}$$

pa za ključni element biramo a_{pj}^k (što odgovara zameni bazične promenljive $x_{B,p}$ i nebazične promenljive $x_{N,j}$). Preći na Korak 1.

Korak 5. Primeniti simpleks algoritam za bazično dopustivu maksimizacionu (minimizacionu) tabelu (Algoritam *SimpleksBasicMax*/*SimpleksBasicMin*).

Napomena 2.6.1 *Kao i u Algoritmu SimpleksBasicMax može postojati više od jedne vrednosti za p za koju je* $\frac{b_p^k}{a_{pj}^k}$ *najmanje. Može se odabrati proizvoljno p.*

Implementacija simpleks algoritma koji ne koristi veštačke promenljive pomoću funkcionalnih mogućnosti u jeziku MATHEMATICA opisana je u monografiji [72].

Primer 2.6.4 Ovim primerom se ilustruje opšti simpleks algoritam (Algoritam *SimpleksMax*) na tabelu maksimizacije

x_1	x_2	-1	
-1	-2	-3	$=-t_1$
1	1	3	$=-t_2$
1	1	2	$=-t_3$
-2	4	0	$=z$

Rešenje:
(1) Početna tabela je očigledno tabela maksimizacije.
(2) Prelazimo na korak (2) jer je $b_1 = -3$ negativno.
(3) Biramo $b_1 = -3$.
(4) Prelazimo na korak (4), jer su oba koeficijenta $a_{11} = -1$ i $a_{12} = -2$ negativna.
(5) Možemo da izaberemo $a_{11} = -1$ ili $a_{12} = -2$. Radi određenosti biramo $a_{11} = -1$. Kako je $1 = i < m = 3$ izračunavamo

$$\min\left(\left\{\frac{b_1}{a_{11}} = \frac{-3}{-1}\right\} \bigcup \left\{\frac{b_2}{a_{21}} = \frac{3}{1}, \frac{b_3}{a_{31}} = \frac{2}{1}\right\}\right) = \frac{b_3}{a_{31}}$$

Za ključni element uzimamo a_{31}:

x_1	x_2	-1	
-1	-2	-3	$=-t_1$
1*	1	3	$=-t_2$
1	1	2	$=-t_3$
-2	4	0	$=z$

$\Rightarrow$

t_3	x_2	-1	
-1	-1	-1	$=-t_1$
-1	0	1	$=-t_2$
1	1	2	$=-x_1$
2	6	4	$=z$

Prelazimo na korak (1).
(1) Očigledno!
(2) Prelazimo na korak (2), jer je $b_2 = -1$ negativno.
(3) Moramo da izaberemo $b_2 = -1$.
(4) Prelazimo na korak (4), jer je $a_{12} = -1$ negativno.
(5) Moramo da izaberemo $a_{12} = -1$. Kako je $1 = i < m = 3$, izračunavamo

$$\min\left(\left\{\frac{b_1}{a_{12}} = \frac{-1}{-1}\right\} \bigcup \left\{\frac{b_3}{a_{32}} = \frac{2}{1}\right\}\right) = \frac{b_1}{a_{12}}$$

Ključni element je a_{12}:

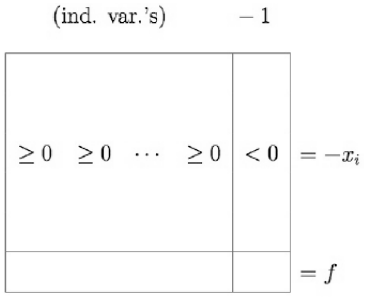
t_3	x_2	-1	
-1	-1	-1	$=-t_1$
-1	0	1	$=-t_2$
1	1	2	$=-x_1$
2	6	4	$=z$

$\Rightarrow$

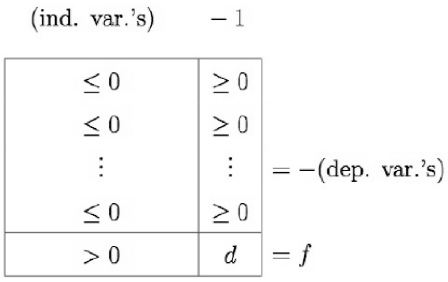
t_3	t_1	-1	
-1	-1	1	$=-x_2$
-1	0	1	$=-t_2$
2	1	1	$=-x_1$
8	6	-2	$=z$

Prelazimo na korak (1).
(1) Očigledno!
(2) $b_1, b_2, b_3 \geq 0$, to jest, tabela je maksimizaciona bazično dopustiva. Prelazimo na korak(5).
(5) Primeniti simpleks algoritam za maksimizacione tabele sa bazično dopustivim rešenjima (prepustimo čitaocu da sam završi zadatak).

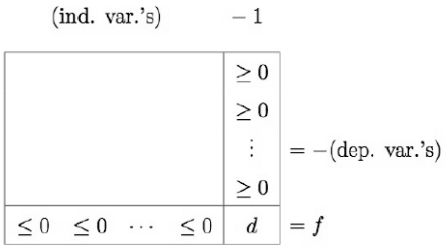
Mogući završeci u simpleks algoritmu prikazani su na sledećim slikama:



Slika 2.6.1. Tuckerova tabela kada optimalno rešenje ne postoji.



Slika 2.6.2. Tuckerova tabela kada je optimalno rešenje neograničeno.



Slika 2.6.3. Tuckerova tabela kada optimalno rešenje postoji.

2.7 BigM metod

Kombinovanjem I i II faze dobijen je algoritam kojim je moguće rešiti proizvoljan problem linearnog programiranja. Napomenimo na kraju da je suština ideje dvofazne modifikacije simpleks metoda u sledećem: ograničenja polaznog problema (2.5.1) uvođenjem veštačkih promenljivih $w_1, \ldots, w_m$ dopunimo do bazično dopustivog rešenja, posmatrajući pri tome proširenu ciljnu funkciju

$$z^w = c^T x + Mw_1 + \cdots + Mw_m,$$

gde je M proizvoljno veliki koeficijent. Sve dok se veštačke promenljive javljaju u bazično dopustivom rešenju, optimalno rešenje nije pronađeno zbog proizvoljnosti koeficjenta M. Dakle, cilj je da se primenom simpleks metoda sve veštačke

promenljive eliminišu iz bazično dopustivog rešenja i time izjednače sa nulom. Kada se formira bazično dopustivo rešenje bez veštačkih promenljivih, one nam nisu više potrebne jer su u redukovanom problemu ispunjeni uslovi za primenu Algoritma 1. Smisao uvođenja veštačkih promenljivih je samo u tome da se kanališe redosled izvođenja elementarnih transformacija.

U jednoj varijanti ovog metoda [17, 73], pomoćni problem se formira na sledeći način:

$$\begin{aligned} \min \quad & c^T x + Me^T w, \\ \text{p.o.} \quad & Ax + w = b, \\ & x \geq 0,\ w \geq 0, \end{aligned} \tag{2.7.1}$$

gde je M dovoljno velika konstanta. Početno bazično rešenje ovog problema je $(0, b)$. Ukoliko se, primenom simpleks metoda dobije optimalno rešenje kod koga je neka od promenljivih w_i bazična, polazni problem je nedopustiv (zbog proizvoljnosti konstante M). U suprotnom, dobijeno rešenje je optimalno i za polazni problem. Zbog konstante M ovaj metod se često naziva i ***BigM*** metod. Glavni nedostatak ovog metoda je povećanje dimenzije problema i neodređenost oko izbora konstante M.

Primer 2.7.1 Razmotrimo sada problem u kome se traži minimalna vrednost funkcije kriterijuma primenom ***BigM*** metoda:

$$\begin{aligned} \min \quad & 160x_1 + 100x_2 + 120x_3, \\ \text{p.o.} \quad & 2x_1 + x_2 + 2x_3 \geq 350, \\ & x_1 + x_2 + x_3 \geq 300, \\ & 4x_1 + x_2 + 2x_3 \geq 400, \\ & x_1 \geq 0\ ,\ x_2 \geq 0\ ,\ x_3 \geq 0. \end{aligned} \tag{2.7.2}$$

Uvođenjem izravnavajućih promenljivih, sistem (2.7.2) ćemo izraziti u obliku jednačina. Pošto je leva strana nejednačina ovog sistema veća od desne, oduzimamo izravnavajuće promenljive i dobijamo sledeći sistem jednačina:

$$\begin{aligned} & 2x_1 + x_2 + 2x_3 - x_4 = 350\ , \\ & x_1 + x_2 + x_3 - x_5 = 300\ , \\ & 4x_1 + x_2 + 2x_3 - x_6 = 400\ . \end{aligned} \tag{2.7.3}$$

Koeficijenti uz izravnavajuće promenljive su negativni. To znači da ne možemo odrediti početno nenegativno bazično rešenje u prvoj simpleks tabeli. Zbog toga uvodimo nove promenljive u model, tzv. veštačke promenljive. Ove promenljive ne pripadaju sistemu ograničenja i nemaju konkretno ekonomsko značenje, osim što služe kao kalkulativno sredstvo. U postupku rešavanja problema one se eliminišu iz rešenja, tako što se u optimalnom rešenju ne može pojaviti ni jedna veštačka promenljiva. Ako nenegativno bazično rešenje problema sadrži bar jednu pozitivnu veštačku promenljivu (koje se ne možemo osloboditi), onda ne postoji moguće rešenje problema.

Da se veštačke promenljive ne bi pojavile u optimalnom rešenju, u funkciji kriterijuma uz ove promenljive uvodimo koeficijente obeležene sa M, gde je M relativno veliki pozitivan broj.[1] Tako smo, posle proširenja problema sa veštačkim promenljivim, dobili sledeći model:

$$(\min);\ z_0 = 160x_1 + 100x_2 + 120x_3 + 0x_4 + 0x_5 + +0x_6 + Mx_7 + Mx_8 + Mx_9\ ,$$

[1] Kada tražimo maksimalnu vrednost funkcije kriterijuma, uz veštačke promenljive uvodimo koeficijent $-M$.

$$\begin{aligned} 2x_1 + x_2 + 2x_3 - x_4 \qquad\quad + x_7 \qquad\quad &= 350\ , \\ x_1 + x_2 + \ x_3 \qquad - x_5 \qquad + x_8 \qquad &= 300\ , \\ 4x_1 + x_2 + 2x_3 \qquad - x_6 \qquad + x_9 \quad &= 400\ , \end{aligned}$$

u kome sve promenljive moraju biti nenegativne.

Sada se može popuniti početna simpleks tabela i odrediti početno bazično rešenje. Njega će sačinjavati veštačke promenljive.

			160	100	120	0	0	0	M	M	M	
C	B	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	
M	X_7	350	2	1	2	-1	0	0	1	0	0	
M	X_8	300	1	1	1	0	-1	0	0	1	0	
M	X_9	400	$\boxed{4}$	1	2	0	0	-1	0	0	1	$\rightarrow$
$z_j - c_j$		0	-160	-100	-120	0	0	0	0	0	0	
		1050	7	3	5	-1	-1	-1	0	0	0	
			$\uparrow$									

Tabela 2.7.1.

Simpleks tabela je popunjena na uobičajeni način. Jedino su koeficijenti u redu $(z_j - c_j)$ podeljeni na dva dela. U prvi deo reda unose se koeficijenti uz koje se ne javlja M, a u drugi deo koeficijenti uz koje se javlja M. Tako je, na primer, koeficijent $(z_1 - c_1) = 7M - 160$. Dobijen je množenjem kolone C i kolone X_1, proizvodi su sabrani i od njih je oduzet koeficijent iz zaglavlja. Na isti način utvrđeni su i koeficijenti za ostale kolone. Ova podela koeficijenata iz reda $(z_j - c_j)$ na dva dela izvršena je samo iz praktičnih razloga. Kako je M veoma veliki broj, to će, sve dok su i veštačke promenljive u rešenju, drugi deo reda određivati koja promenljiva ulazi u naredno rešenje.

Da li je u prvoj simpleks tabeli pronađeno optimalno rešenje i, ako nije, koju promenljivu treba izabrati da uđe u naredno rešenje? Ovde je odgovor sasvim očigledan: početno rešenje čine veštačke promenljive, pa je sigurno da ono ne može biti optimalno. Sve dok u redu $(z_j - c_j)$ postoje pozitivni koeficijenti (a tražimo minimalnu vrednost funkcije), nije pronađeno optimalno rešenje. Dodajmo ovome i drugi kriterijum: u naredno rešenje treba da uđe ona promenljiva kojoj u simpleks tabeli odgovara najveća (pozitivna) vrednost koeficijenta iz reda $(z_j - c_j)$.

Vratimo se prvoj tabeli i konstatujmo da nije pronađeno optimalno rešenje, te da u naredno rešenje treba da uđe promenljiva x_1. Na osnovu količnika između kolone X_0 i kolone X_1, iz rešenja izlazi promenljiva x_9. U prvoj simpleks tabeli označena je kolona X_1, treći red u kome je promenljiva x_9 i zaokružen je karakterističan koeficijent. Na osnovu tako pripremljene prve simpleks tabele, dobijamo drugu simpleks tabelu.

Bazično rešenje u drugoj tabeli sačinjavaju promenljive

$$x_1 = 100\ , \quad x_7 = 150\ , \quad x_8 = 200\ ,$$

a vrednost funkcije kriterijuma iznosi $z_0 = 16000 + 350M$. Optimalno rešenje nije pronađeno. U naredno rešenje treba da uđe promenljiva x_3, a iz rešenja će izaći promenljiva x_7. Zbog toga smo u drugoj tabeli označili kolonu X_3, prvi red u kome je promenljiva x_7 i zaokružili karakterističan koeficijent.

			160	100	120	0	0	0	M	M	M	
C	B	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	
M	X_7	150	0	1/2	$\boxed{1}$	-1	0	1/2	1	0	$-1/2$	$\rightarrow$
M	X_8	200	0	3/4	1/2	0	-1	1/4	0	1	$-1/4$	
160	X_1	100	1	1/4	1/2	0	0	$-1/4$	0	0	1/4	
$z_j - c_j$		16000	0	$-$ 60	$-$ 40	0	0	$-$ 40	0	0	40	
		350	0	5/4	3/2	-1	-1	3/4	0	0	$-7/4$	
					$\uparrow$							

Tabela 2.7.2.

Sastavljamo treću simpleks tabelu.

			160	100	120	0	0	0	M	M	M	
C	B	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	
120	X_3	150	0	1/2	1	− 1	0	1/2	1	0	−1/2	
M	X_8	125	0	1/2	0	1/2	−1	0	−1/2	1	0	→
160	X_1	25	1	0	0	1/2	0	−1/2	−1/2	0	1/2	
$z_j - c_j$		22000	0	− 40	0	− 40	0	− 20	40	0	20	
		125	0	1/2	0	1/2	−1	0	−3/2	0	− 1	
				↑								

Tabela 2.7.3.

I u trećoj simpleks tabeli nije pronađeno optimalno rešenje. U rešenju se nalazi još jedna veštačka promenljiva ($x_8 = 125$), a u drugom delu reda $(z_j - c_j)$ postoje pozitivni koeficijenti. Treba odrediti koja promenljiva ulazi u naredno rešenje. Vidimo da u trećoj simpleks tabeli dve promenljive (x_2 i x_4) ravnopravno konkurišu za ulazak u naredno rešenje jer imaju potpuno iste koeficijente. Prema dopunskom kriterijumu prednost će dobiti ona promenljiva koja (ukoliko bude izabrana da uđe u rešenje) dobija veću vrednost. Zato najpre delimo kolonu X_0 kolonom X_2 i određujemo da će promenljiva x_2, ukoliko uđe u naredno rešenje, dobiti vrednost 250; zatim delimo kolonu X_0 kolonom X_4 i određujemo da će promenljiva x_4, ukoliko uđe u rešenje, dobiti vrednost 50.

Prema tome, u naredno rešenje ulazi promenljiva koja dobija veću vrednost, tj. promenljiva x_2. U trećoj tabeli označili smo kolonu X_2, zatim drugi red (jer promenljiva x_8 izlazi iz rešenja) i zaokružili karakterističan koeficijent.

Sastavljamo četvrtu simpleks tabelu.

			160	100	120	0	0	0	M	M	M
C	B	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
120	X_3	25	0	0	1	−3/2	1	1/2	3/2	− 1	−1/2
100	X_2	250	0	1	0	1	− 2	0	− 1	2	0
160	X_1	25	1	0	0	1/2	0	−1/2	−1/2	0	1/2
$z_j - c_j$		32000	0	0	0	0	−80	− 20	0	80	20
		0	0	0	0	0	0	0	− 1	− 1	− 1

Tabela 2.7.4.

U četvrtoj simpleks tabeli nema veštačkih promenljivih u rešenju. To se može zaključiti prema koloni B, ali i prema drugom delu reda $(z_j - c_j)$. Pošto su veštačke promenljive izašle iz rešenja, u ovom delu reda $(z_j - c_j)$ ispod svih kolona su nule, osim ispod kolona veštačkih promenljivih gde su jedinice sa znakom minus.

Problem se dalje može rešavati bez kolona koje odgovaraju veštačkim promenljivim (ove promenljive kad jednom izađu iz rešenja, ne mogu se više vratiti u njega) i bez drugog dela reda $(z_j - c_j)$. Ove kolone se mogu zanemarivati i odmah pošto veštačka promenljiva izađe iz rešenja. Postoji, međutim, razlog da se zadrže i ove kolone do kraja postupka, jer se u tom delu simpleks tabele nalaze podaci koji se mogu korisno upotrebiti za analizu optimalnog rešenja.

Rešavanje problema simpleks tabelom treba nastaviti tako što se za ocenu optimalnosti rešenja i za izbor promenljive koja će ući u rešenje uzima u obzir prvi deo reda $(z_j - c_j)$. Posmatranjem ovog reda u četvrtoj simpleks tabeli, vidimo da ne postoje pozitivni koeficijenti, pa možemo zaključiti da je rešenje iz ove tabele optimalno. Njega sačinjavaju promenljive:

$$x_1 = 25\ , \quad x_2 = 250\ , \quad x_3 = 25\ ,$$

a minimalna vrednost funkcije kriterijuma iznosi $z_0 = 32000$.

Rekli smo da je u četvrtoj simpleks tabeli pronađeno optimalno rešenje. Međutim, u redu $(z_j - c_j)$ kolone X_4 (kolona nebazične promenljive x_4) nalazi se koeficijent nula. To znači da

možemo odrediti da promenljiva x_4 uđe u naredno rešenje i da odredimo još jedno rešenje sa istom vrednošću funkcije kriterijuma. Drugim rečima, možemo odrediti još jedno optimalno rešenje, pa je potrebno da sastavimo još jednu simpleks tabelu. Pošto promenljiva x_4 ulazi u naredno rešenje, u četvrtoj tabeli označili smo kolonu X_4. Iz rešenja izlazi promenljiva x_1, pa smo označili i treći red i zaokružili karakterističan koeficijent. Sada možemo odrediti petu simpleks tabelu.

			160	100	120	0	0	0	M	M	M
C	B	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
120	X_3	100	3	0	1	0	1	− 1	0	− 1	1
100	X_2	200	−2	1	0	0	− 2	1	0	2	− 1
0	X_4	50	2	0	0	1	0	− 1	−1	0	1
$z_j - c_j$		32000	0	0	0	0	−80	−20	0	80	20
		0	0	0	0	0	0	0	−1	− 1	− 1

Tabela 2.7.5.

U petoj simpleks tabeli dobili smo još jedno optimalno rešenje. Njega sačinjavaju promenljive

$$x_2 = 200\ , \quad x_3 = 100\ , \quad x_4 = 50\ ,$$

a minimalna vrednost funkcije kriterijuma, $z_0 = 32000$, ista je kao i kod rešenja iz četvrte simpleks tabele.

Napomenimo samo da se konveksnom kombinacijom ova dva optimalna rešenja mogu dobiti i druga, takođe, optimalna rešenja. Rešenja koja su pronađena pomoću simpleks tabela bazična su, a sva rešenja koja se mogu dobiti kao konveksna kombinacija bazičnih rešenja nisu bazična.

Obeležimo optimalno rešenje iz četvrte simpleks tabele sa X_0^1, a rešenje iz pete tabele sa X_0^2, pa ćemo dobiti sledeći izraz za konveksnu kombinaciju:

$$X_0^3 = qX_0^1 + (1-q)X_0^2\ , \quad 0 < q < 1\ ,$$

gde X_0^3 predstavlja novo optimalno rešenje.

Za problem minimuma koji smo rešavali bilo je potrebno da se u svakoj jednačini doda veštačka promenljiva. To je problem u kome je sistem ograničenja imao sve nejednačine smera "$\geq$", pa je u svim nejednačinama oduzeta izravnavajuća promenljiva i dodata veštačka. Za ovakve probleme, kada je broj veštačkih promenljivih jednak broju ograničenja, kažemo da imaju potpunu veštačku bazu. Razume se da ne mora i neće uvek tako biti. Ukoliko problem ima manje veštačkih promenljivih nego što je broj ograničenja, onda imamo nepotpunu veštačku bazu. U postupku rešavanja ovih problema nema nikakvih razlika u odnosu na već razmatrani postupak.

Razmotrimo sada metod iz [84] kod koga nema povećanja dimenzija problema. Posmatraćemo jedan kanonski oblik (2.3.1) problema linearnog programiranja:

$$\begin{aligned} \max\ & z(x) = c_1x_{N,1} + \ldots + c_nx_{N,n} + d \\ \text{p.o.}\ & a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n - b_1 = -x_{B,1} \\ & a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n - b_2 = -x_{B,2} \\ & \quad \ldots\ldots\ldots \\ & a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n - b_m = -x_{B,m}. \end{aligned} \tag{2.7.4}$$

pri čemu nisu svi $b_i \geq 0$, tj postoji neko $i \in \{1, \ldots, m\}$ takvo da je $b_i < 0$ ali koeficijenti c_i ispunjavaju uslov optimalnosti. Sada odgovarajuće bazično rešenje $x = (0, \ldots, 0, b_1, \ldots, b_n)$ nije dopustivo, tj ne pripada skupu Ω_P. Zadatak koji rešavamo je nalaženje ekvivalentnog bazično dopustivog problema (2.7.4), odnosno nalaženje jednog bazično dopustivog rešenja. Za to nam je potreban dualni simpleks metod.

2.8 Dualnost u linearnom programiranju

U određenim slučajevima matematički model osnovnog zadatka LP ne može da posluži za nalaženje optimalnog plana primenom opisanog simpleks metoda. U ovakvim slučajevima se pribegava preformulaciji primalnog zadatka LP u dualni zadatak, pomoću koga je moguće naći traženo rešenje. Između primalnog i dualnog zadatka postoje sledeće korespodencije:

1. Dualni model ima onoliko promenljivih koliko primalni zadatak ograničenja, i ograničenja koliko primalni zadatak promenljivih;

2. Slobodni članovi u ograničenjima primalnog zadatka postaju koeficijenti uz promenljive funkcije kriterijuma (cilja) dualnog modela, a koeficijenti uz promenljive funkcije kriterijuma primalnog modela postaju slobodni članovi u ograničenjima dualnog modela;

3. Smer nejednačina dualnog modela je suprotan smeru nejednačina primalnog modela;

4. Ako je u primernom modelu tražen maksimum funkcije kriterijuma, u dualnom se traži minimum, i obrnuto;

5. Dopunskoj promenljivoj x_{n+1} primala koja je u optimalnom baznom rešenju odgovara promenljiva y_j u dualnom modelu, pri čemu je $x_{n+1}y_j = 0,\ j = 1, \ldots, m$.

6. Realnoj promenljivoj x_j primala iz bazičnog dopustivog rešenja odgovara dopunska promenljiva y_{m+j} dualnog modela sa nultom vrednošću $x_j y_{m+j} = 0,\ j = 1, \ldots, n$.

7. Matrica ograničenja u primalnom modelu jednaka je transponovanoj matrici ograničenja u dualnom modelu.

Razmatramo problem linearnog programiranja u standardnom obliku

$$\begin{aligned} \min \quad & c^T x, \\ \text{p.o.} \quad & Ax = b, \\ & x \geq 0, \end{aligned} \tag{2.8.1}$$

gde je A matrica tipa $m \times n$ sa uobičajenom pretpostavkom $\text{rang}(A) = m$. Problemu (2.8.1) pridružimo tzv. *dualni problem* oblika

$$\begin{aligned} \max \quad & b^T y, \\ \text{p.o.} \quad & A^T y \leq c \end{aligned} \tag{2.8.2}$$

u kome su slobodni koeficijenti problema (2.8.1) postali koeficijenti u ciljnoj funkciji, a koeficijenti iz ciljne funkcije problema (2.8.1) postali slobodni koeficijenti u sistemu ograničenja. Primetimo da za vektor y nije nametnut uslov nenegativnosti. Problem (2.8.1) se u kontekstu teorije dualnosti naziva *primalni problem.* Dodavanjem izravnavajućih promenljivih problem (2.8.2) se svodi na

$$\begin{aligned} \max \quad & b^T y, \\ \text{p.o.} \quad & A^T y + s = c, \\ & xs \geq 0. \end{aligned} \tag{2.8.3}$$

Primer 2.8.1 Neka je dat primalni problem linearnog programiranja

$$\begin{array}{lrrrrrl} \min & -x_1 - x_2, & & & & & \\ \text{p.o.} & x_1 & -x_2 & -x_3 & & = 1, \\ & -x_1 & +x_2 & & -x_4 & = 1, \\ & x \geq 0. & & & & & \end{array}$$

Ovom problemu, dualni problem je

$$\begin{array}{lrrrrrrrr} \max & y_1 + y_2, & & & & & & & \\ \text{p.o.} & y_1 & -y_2 & +s_1 & & & & = & -1, \\ & -y_1 & +y_2 & & +s_2 & & & = & -1, \\ & -y_1 & & & & +s_3 & & = & 0, \\ & & -y_2 & & & & +s_4 & = & 0, \\ & s \geq 0. & & & & & & & \end{array}$$

Neka su Ω_P i Ω_D dopustivi skupovi problema (2.8.1) i (2.8.3), respektivno. Između ciljnih funkcija primala i duala postoji sledeća veza:

Teorema 2.8.1 (Slaba dualnost). *Ako su Ω_P i Ω_D neprazni skupovi i ako je $x \in \Omega_P$ i $(y, s) \in \Omega_D$, tada je $c^T x \geq b^T y$.*

Dokaz. S obzirom da važi $x \geq 0$ i $s \geq 0$ dobija se

$$0 \leq s^T x = (c - A^T y)^T x = c^T x - y^T (Ax) = c^T x - b^T y.$$

Dokaz je završen. □

Sledeća teorema je fundamentalna u teoriji dualnosti.

Teorema 2.8.2 (Jaka dualnost) *Ako problem* (2.8.1) *ima optimalno rešenje onda i problem* (2.8.3) *ima optimalno rešenje i pri tome* $\min c^T x = \max b^T y$.

Dokaz. Kako problem (2.8.1) po pretposavci teoreme ima optimalno rešenje, to se primena Algoritma 2 na (2.8.1) završava u konačnom broju iteracija sa optimalnim bazično dopustivim rešenjem x^*. Neka su $j_1 < \cdots < j_m$ indeksi bazičnih, a $i_1 < \cdots < i_{n-m}$ indeksi nebazičnih kolona u odgovarajućoj simpleks tablici i neka je $A_B = [K_{j_1} \dots K_{j_m}]$, $A_N = [K_{i_1} \dots K_{j_{n-m}}]$, $c_B = (c_{j_1}, \dots, c_{j_m})$, $c_N = (c_{i_1}, \dots, c_{j_{n-m}})$. Na osnovu prethodnih razmatranja kanonskog oblika linearnog problema, nulta vrsta na nultoj poziciji sadrži $-cTx^* = -c_B^T A_B^{-1} b$, na pozicijama $j_1, \dots, j_m$ sadrži nule i na pozicijama $i_1, \dots, j_{n-m}$ sadrži vektor $c_N - A_N^T (A_B^{-1})^T c_B$ koji je nenegativan zbog uslova optimalnosti.

Neka je $y^* = (A_B^{-1})^T c_B$ i neka je s^* definisano sa $s_B^* = 0$, $s_N^* = c_N - A_N^T (A_B^{-1})^T c_B$. Tada je $s^* \geq 0$ i

$$A_B^T y^* + s_B^* = c_B, \quad A_N^T y^* + s_N^* = c_N,$$

tj. rešenje (y^*, s^*) je dopustivo. Pored toga je $c^T x^* = c_B^T A_B^{-1} b = (y^*)^T b = b^T y^*$. Kako po Teoremi 2.8.1 važi $b^T y \leq c^T x^* = b^T y^*$, sledi da je (y^*, s^*) optimalno rešenje problema (2.8.3). □

Iz sledeće teoreme slede Karash-Kuhn-Tuckerovi uslovi optimalnosti.

Teorema 2.8.3 *(i)* $x^* \in \Omega_P$ *je optimalno rešenje problema* (2.8.1) *ako i samo ako postoji* $(y^*, s^*) \in \Omega_D$ *tako da je* $c^T x^* = b^T y^*$.

(ii) $x^* \in \Omega_P$ *je optimalno rešenje problema* (2.8.1) *ako i samo ako postoji* $(y^*, s^*) \in \Omega_D$ *tako da je* $(x^*)^T s^* = 0$.

Dokaz. (i) Neka je $x^* \in \Omega_P$ optimalno rešenje problema (2.8.1). Tada po Teoremi 2.8.2 problem (2.8.3) ima optimalno rešenje $(y^*, s^*) \in \Omega_D$ i važi $c^T x^* = b^T y^*$.

Ako postoji $(y^*, s^*) \in \Omega_D$ tako da je $c^T x^* = b^T y^*$, na osnovu Teoreme 2.8.1 sledi da za svako $x \in \Omega_P$ važi $c^T x \geq c^T x^*$. Kako je $x^* \in \Omega_P$, to je x^* optimalno rešenje problema (2.8.1).

(ii) Neka je $x^* \in \Omega_P$ optimalno rešenje problema (2.8.1). Tada prema (i) postoji $(y^*, s^*) \in \Omega_D$ i važi $c^T x^* = b^T y^*$. Kada ovu jednakost transponujemo i od nje oduzmemo jednakost $(x^*)^T A^T y^* = (y^*)^T A x^*$ dobijamo

$$(x^*)^T (c - A^T y^*) = (y^*)^T (b - A x^*).$$

Sada iz $s^* = c - A^T y^*$ i $b - A x^* = 0$ sledi $(x^*)^T s^* = 0$.

Ako je $(y^*, s^*) \in \Omega_D$ takvo da je $(x^*)^T s^* = 0$, zbog dopustivosti x^* i (y^*, s^*) sledi

$$(x^*)^T s^* = (x^*)^T (c - A^T y^*) = (x^*)^T c - (x^*)^T A^T y^* = (x^*)^T c - b^T y^* = 0,$$

pa na osnovu (i) sledi da je x^* optimalno rešenje problema (2.8.1). □

Značaj Teoreme 2.8.3 je u tome što se uspostavlja sledeća ekvivalencija: vektor $x^* \in \mathbb{R}^n$ je optimalno rešenje problema (2.8.1) ako i samo ako postoje vektori $s^* \in \mathbb{R}^n$ i $y^* \in \mathbb{R}^m$ tako da važe sledeći uslovi

$$\begin{aligned} A^T y^* + s^* &= c, \\ A x^* &= b, \\ x_i^* s_i^* &= 0, \quad i = 1, \ldots, n, \\ (x^*, s^*) &\geq 0. \end{aligned} \tag{2.8.4}$$

Očigledno je da je (2.8.4) takođe potreban i dovoljan uslov da (y^*, s^*) bude optimalno rešenje problema (2.8.3). Ovi uslovi igraju važnu ulogu u takozvanim primal-dual metodima unutrašnje tačke.

Pitanje egzistencije dopustivih rešenja primalnog i dualnog problema rešava sledeća teorema:

Teorema 2.8.4 (i) $\Omega_P \neq \emptyset$ *i* $\Omega_D \neq \emptyset$ *ako i samo ako problemi* (2.8.1) *i* (2.8.3) *imaju optimalna rešenja.*

(ii) *Neka je* $\Omega_P \neq \emptyset$. *Tada je* $\Omega_D = \emptyset$ *ako i samo ako je* $\inf\limits_{x \in \Omega_P} c^T x = -\infty$.

(iii) *Neka je* $\Omega_D \neq \emptyset$. *Tada je* $\Omega_P = \emptyset$ *ako i samo ako je* $\sup\limits_{(y,s) x \in \Omega_D} b^T y = \infty$.

Dokaz. (i) Ako problemi (2.8.1) i (2.8.3) imaju optimalna rešenja, trivijalno je $\Omega_P \neq \emptyset$ i $\Omega_D \neq \emptyset$.

Neka je $\Omega_P \neq \emptyset$ i $\Omega_D \neq \emptyset$ i uočimo proizvoljno $(y, s) \in \Omega_D$. Tada je po Teoremi 2.8.1 ciljna funkcija problema (2.8.1) ograničena odozdo i stoga problem (2.8.1) ima optimalno rešenje. Na osnovu Teoreme 2.8.2 sledi da i problem (2.8.3) ima optimalno rešenje.

(ii) Neka je $\Omega_D = \emptyset$ i pretpostavimo suprotno, tj. da je $c^T y \geq M$ za $x \in \Omega_P$. Tada problem (2.8.1) ima optimalno rešenje i na osnovu Teoreme 2.8.2 sledi da i problem (2.8.3) ima optimalno, stoga i dopustivo rešenje, što je kontradikcija.

Neka je $\inf_{x \in \Omega_P} c^T x = -\infty$ i pretpostavimo da postoji $(y, s) \in \Omega_D$. Tada je prema Teoremi 2.8.1 $c^T x \geq c^T y$ za svako $x \in \Omega_P$, što je nemoguće.

(iii) Uvođenjem smene $y = y^+ - y^-$, $y^+ \geq 0$, $y^- \geq 0$, problem (2.8.3) je ekvivalentan minimizacionom problemu

$$\begin{array}{ll} \min & (-b)^T (y^+ - y^-), \\ \text{p.o.} & A^T (y^+ - y^-) + s = c, \\ & y^+ \geq 0,\ y^- \geq 0,\ s \geq 0, \end{array} \tag{2.8.5}$$

čiji je dual

$$\begin{array}{ll} \max & c^T u, \\ \text{p.o.} & Au \leq -b, \quad -Au \leq b, \quad u \leq 0. \end{array} \tag{2.8.6}$$

Smenom $x = -u$ dual (2.8.6) postaje

$$\begin{array}{ll} \max & (-c)^T x, \\ \text{p.o.} & Ax = b, \\ & x \geq 0. \end{array} \tag{2.8.7}$$

Primenom (ii) na dualni par (2.8.5) i (2.8.7) direktno sledi da je $\Omega_P = \emptyset$ ako i samo ako je $\inf\limits_{(y,s) x \in \Omega_D} (-b)^T y = -\infty$, tj. $\sup\limits_{(y,s) x \in \Omega_D} b^T y = \infty$. $\square$

Primetimo da iz dokaza Teoreme 2.8.4 (iii) sledi zaključak da je dual dualnog problema jednak primalnom problemu.

Napomena 2.8.1 *U Primeru 2.6.1 je $\Omega_P = \emptyset$ i $\Omega_D = \emptyset$, što pokazuje da je moguć i taj slučaj.*

Neka je $I_1 \cup I_2 = \{1, \ldots, m\}$, $I_1 \cap I_2 = \emptyset$, $J_1 \cup J_2 = \{1, \ldots, n\}$, $J_1 \cap J_2 = \emptyset$, i neka su $V_1, \ldots, V_m$ vrste $K_1, \ldots, K_n$ kolone matrice A. Tada problemu

$$\begin{array}{ll} \min & c^T x, \\ \text{p.o.} & V_i^T x = b_i, \quad i \in I_1, \\ & V_i^T x \geq b_i, \quad i \in I_2, \\ & x_j \geq 0, \qquad j \in J_1, \\ & x_j \text{ neograničeno po znaku}, \ j \in J_2, \end{array} \tag{2.8.8}$$

odgovara dualni problem

$$\begin{array}{ll} \max & b^T y, \\ \text{p.o.} & y_i \text{ neograničeno po znaku, } i \in I_1, \\ & y_i \geq 0, \quad i \in I_2, \\ & K_j^T y \geq c_j, \quad j \in J_1, \\ & K_j^T y = c_j, \quad j \in J_2, \end{array} \tag{2.8.9}$$

i važe analogoni Teorema 2.8.1 i 2.8.2. Dual simetričnog problema linearnog programiranja

$$\begin{array}{ll} \min & c^T x, \\ \text{p.o.} & Ax \geq b, \\ & x \geq 0, \end{array}$$

dat je sa

$$\begin{array}{ll} \max & b^T y, \\ \text{p.o.} & A^T y \leq c, \\ & y \geq 0. \end{array}$$

Iz (2.8.8) i (2.8.9) sledi da je problem dualan sa (1.1.3) dat sa

$$\begin{array}{ll} \max & b^T y, \\ \text{p.o.} & y_i \text{ neograničeno po znaku, } i \in I_1, \\ & y_i \geq 0, \quad i \in I_2, \\ & y_i \leq 0, \quad i \in I_3, \\ & K_j^T y \leq c_j, \quad j \in J, \\ & K_j^T y = c_j, \quad j \in \{1, \ldots, n\} \backslash J. \end{array}$$

2.9 Dualni simpleks metod

Razmotrimo problem linearnog programiranja kome odgovara LP-tablica

$$\begin{array}{cccc} -d & c_1 & \cdots & c_n \\ b_1 & a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & & \vdots \\ b_m & a_{m1} & \cdots & a_{mn} \end{array}$$

koju nazivamo *dualnom simpleks tablicom* ako ona sadrži m različitih bazičnih kolona i važi $c_1 \geq 0, \ldots, c_n \geq 0$. Ako je $b_1 \geq 0, \ldots, b_m \geq 0$, tada je dualna simpleks tablica ujedno i simpleks tablica kojoj odgovara optimalno bazično rešenje. Ako postoji $k \in \{1, \ldots, m\}$ tako da je $b_k < 0$ i $a_{k1} \geq 0, \ldots, a_{kn} \geq 0$, pokazali smo da

je skup dopustivih rešenja prazan. Ako postoji $a_{ki} < 0$ moguće je primeniti dualni simpleks metod.

Neka je dat problem linearnog programiranja

$$\begin{array}{ll} \min & d + c^T x, \\ \text{p.o.} & Ax = b, \\ & x \geq 0, \end{array}$$

kome odgovara početna dualna simpleks tablica DT_0 :

$$\begin{array}{cccc} -d^0 & c_1^0 & \cdots & c_n^0 \\ b_1^0 & a_{11}^0 & \cdots & a_{1n}^0 \\ \vdots & \vdots & & \vdots \\ b_m^0 & a_{m1}^0 & \cdots & a_{mn}^0 \end{array}$$

Sada su ispunjeni uslovi za primenu dualnog simpleks metoda po sledećem algoritmu:

Algoritam 4. (Dualni simpleks metod).
Staviti $k = 0$; $k-$ta iteracija se satoji opd sledećih koraka

Korak 1. Ako je $b_i^k \geq 0$ za sve $i = 1, \ldots, m$, algoritam staje, jer je bazično dopustivo rešenje optimalno.

Korak 2. Za svako i za koje je $b_i^k < 0$ ispitati da li je $a_{ij}^k \geq 0$ za sve $j = 1, \ldots, n$. Ako takvo j postoji, algoritam staje jer je dopustivi skup prazan.

Korak 3. Odrediti $s \in \{1, \ldots, m\}$ za koje je $b_s^k < 0$.
Odrediti $r \in \{1, \ldots, m\}$ takvo da je $\dfrac{c_r^k}{a_{sr}^k} = \max\left\{ \dfrac{c_j^k}{a_{sj}^k} | a_{sj}^k < 0 \right\}$.

Korak 4. Primeniti na DT_k sledeće elementarne transformacije:
- pomnožiti $s-$tu vrstu sa $-a_{ir}^k/a_{sr}^k$ i dodati vrstama $i = 0, \ldots, m$, $i \neq s$;
- podeliti $s-$tu vrstu sa a_{sr}^k.

Korak 5. Zameniti k sa $k + 1$ i ići na Korak 1.

Dualni simpleks algoritam je maksimizacioni za ciljnu funkciju, što sledi iz sledeće teoreme:

Teorema 2.9.1 *Primenom elementarnih transformacija iz Koraka 4 Algoritma 4 se od dualne simpleks tablice DT_k dobija dualna simpleks tablica DT_{k+1} koja odgovara ekvivalentnom problemu linearnog programiranja. Pri tome je $d^{k+1} \geq d^k$.*

Dokaz. Očigledno je da se primenom elementarnih transformacija dobija LP-tablica DT_{k+1} koja odgovara ekvivalentnom problemu linearnog programiranja. Pri tome sve bazične kolone koje su u $s-$toj vrsti imale nule ostaju bazične, a $r-$ta

kolona postaje bazična umesto one koja je u $s-$toj vrsti imala jedinicu. Za $j = 1, \ldots, n$ je

$$c_j^{k+1} = c_j^k - \frac{a_{sj}^k}{a_{sr}^k} c_r^k.$$

Ako je $a_{sj}^k \geq 0$, iz $c_r^k \geq 0$ i $a_{sr}^k < 0$ sledi $c_j^{k+1} \geq c_j^k \geq 0$. Ako je $a_{sj}^k < 0$, na osnovu izbora r u Koraku 3 sledi

$$c_j^{k+1} = a_{sj}^k \left(\frac{c_j^k}{a_{sj}^k} - \frac{c_r^k}{a_{sr}^k} \right) \geq 0.$$

Stoga je DT_{k+1} dualna simpleks tablica. Iz $c_r^k b_s^k / a_{sr}^k \geq 0$, sledi

$$-d^{k+1} = -d^k - \frac{b_s^k}{a_{sr}^k} c_r^k \leq -d^k \quad \Leftrightarrow \quad d^{k+1} \geq d^k$$

čime je dokaz kompletiran. □

Primer 2.9.1 Neka je dat problem

$$\begin{array}{lrrrrrrrr}
\min & 3x_1 + 2x_2, & & & & & & & \\
 & -x_1 & +3x_2 & +x_3 & & & & = & -1, \\
 & -2x_1 & -10x_2 & & +x_4 & & & = & -10, \\
\text{p.o.} & 2x_1 & +4x_2 & & & +x_5 & & = & 8, \\
 & 3x_1 & -5x_2 & & & & +x_6 & = & 6, \\
 & x \geq 0. & & & & & & &
\end{array}$$

kome odgovara dualna simpleks tablica

$$\begin{array}{rrrrrrr}
0 & 3 & 2 & 0 & 0 & 0 & 0 \\
-1 & \boxed{-1} & 3 & 1 & 0 & 0 & 0 \\
-10 & -2 & -10 & 0 & 1 & 0 & 0 \\
8 & 2 & 4 & 0 & 0 & 1 & 0 \\
6 & 3 & -5 & 0 & 0 & 0 & 1
\end{array}$$

Primenom Algoritma 4 sa uokvirenim pivot elementima, dobijamo

$$\begin{array}{rrrrrrr}
-3 & 0 & 11 & 3 & 0 & 0 & 0 \\
1 & 1 & -3 & -1 & 0 & 0 & 0 \\
-8 & 0 & \boxed{-16} & -2 & 1 & 0 & 0 \\
6 & 0 & 10 & 2 & 0 & 1 & 0 \\
3 & 0 & 4 & 3 & 0 & 0 & 1
\end{array}$$

odakle odmah sledi dualna simpleks tablica

$$\begin{array}{rrrrrrr}
-17/2 & 0 & 0 & 13/8 & 11/16 & 0 & 0 \\
5/2 & 1 & 0 & -5/8 & -3/16 & 0 & 0 \\
1/2 & 0 & 1 & 1/8 & -1/16 & 0 & 0 \\
1 & 0 & 0 & 3/4 & 10/16 & 1 & 0 \\
1 & 0 & 0 & 5/2 & 1/4 & 0 & 1
\end{array}$$

Odgovarajuće bazično dopustivo rešenje je $(5/2, 1/2, 0, 0, 1, 1)$ je optimalno i optimalna vrednost ciljne funkcije je $17/2$. Primetimo da su u ovom slučaju koraci dualnog algoritma identični sa Algoritmom 3.

Napomenimo da je Algoritam 4 moguće primeniti ako je poznata početna dualna simpleks tablica. Ukoliko to nije slučaj, dualnu simpleks tablicu možemo konstruisati primenom sledećeg algoritma:

Algoritam 5. (Dualni algoritam za proizvoljnu simpleks tablicu).

Korak 1. Formiramo početnu LP-tablicu oblika

$$\begin{array}{cccc} -d & c_1 & \cdots & c_n \\ b_1 & a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & & \vdots \\ b_m & a_{m1} & \cdots & a_{mn} \end{array}$$

Ukoliko je $c_1, \ldots, c_n \geq 0$, primenimo Algoritam 4.

Korak 2. Ako je $c_j < 0$ za neko j i $a_{1j}, \ldots, a_{mj} \leq 0$, algoritam staje jer je skup ograničenja nedopustiv.

Korak 3. Neka je $c_{j_1}, \ldots, c_{j_q} < 0$.

Ukoliko je $q = n$ biramo $a_{in} < 0$ za stožerni elemet, primenimo simpleks transformaciju i prelazimo na Korak 1.

Ako je $q < n$ i ako postoji $p \in \{j_1, \ldots, j_q\}$ takvo da je ispunjen sledeći uslov

$$\min_k \left\{ \frac{c_k}{a_{ik}} | \, a_{ik} > 0, \, c_k \geq 0 \right\} \geq \frac{c_p}{a_{ip}} \geq 0, \tag{2.9.1}$$

biramo $a_{ip} < 0$ za stožerni element, primenimo simpleks transformaciju i prelazimo na Korak 1. Ukoliko postoji više elemenata $a_{ip} < 0$ koji zadovoljvaju uslov (2.9.1) biramo onaj za koji eventualno važi

$$\min \left\{ \frac{b_h}{a_{hp}} | \, b_h \geq 0, \, a_{hp} > 0 \right\} \geq \frac{b_i}{a_{ip}}, \quad b_i < 0.$$

Ukoliko uslov 2.9.1 nije ispunjen neka je

$$\min_k \left\{ \frac{c_k}{a_{ik}} | \, c_k \geq 0, \, a_{ik} > 0 \right\} = \frac{c_r}{a_{ir}}.$$

Za tako određeno r postavimo a_{ir} za stožerni element, primenimo simpleks transformaciju i prelazimo na Korak 1.

Napomenimo da nijednoj simpleks tablici generisanoj dualnim alogoritmom ne odgovara dopustivo rešenje primalnog problema osim poslednjoj koja je istovremeno simpleks tablica. Pokazaćemo da svakoj dualnoj simpleks tablici odgovara dopustivo rešenje dualnog problema. Neka je u dualnoj simpleks tablici DT_k skup indeksa bazičnih kolona označen sa B i neka je odgovarajuća baza $\mathcal{A}_B$. Prenumeracijom kolona u pridruženoj matrici A_B možemo postići da DT_k odgovara sledećem problemu linearnog programiranja

$$\begin{array}{ll} \min & c_B^T A_B^{-1} b + (c_N^T - c_B^T A_B^{-1} A_N) x_N, \\ \text{p.o.} & x_B + A_B^{-1} A_N x_N = A_B^{-1} b, \\ & x_B \geq 0, \quad x_N \geq 0. \end{array} \tag{2.9.2}$$

Problem (2.9.2) je kanonski oblik standardnog problema linearnog programiranja u odnosu na bazu $\mathcal{A}_B$ za koju važi $c_N^T - c_B^T A_B^{-1} A_N \geq 0$. Definišemo (y^*, s^*) sa $y^* = (A_B^{-1})^T c_B$, $s_B^* = 0$, $s_N^* = c_N - A_N^T y^*$. Tada je

$$\begin{aligned} A_B^T y^* + s^* &= c_B, \qquad & A_N^T y^* + s^* &= c_N \\ s_B^* &\geq 0, & s_N^* &\geq 0, \end{aligned}$$

tj. (y^*, s^*) je dopustivo rešenje dualnog problema. Dakle, svakoj dualnoj simpleks tablici odgovara dopustivo rešenje duala. Koordinate vektora s^* su u tablici date eksplicitno a vektor y^* se računa iz sistema $A_B^T y^* = c_B$.

2.10 Eliminacija jednačina i slobodnih promenljivih

Još na početku, kada smo definisali oblike problema linearnog programiranja, pokazali smo kako se opšti oblik problema linearnog programiranja svodi na standardni, odnosno simetrični oblik. Ovde ćemo pokazati kako se mogu iskoristiti Tuckerove tabele i zamena promenljivih (algoritam **Replace**) za svođenje opšteg problema linearnog programiranja na kanonski oblik. Znači, posmatrajmo sada opšti oblik problema linearnog programiranja

$$\begin{aligned} \max\ & z(x) = c_1 x_1 + \ldots + c_n x_n + d \\ \text{p.o.}\ & N_i^{(1)}: \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, p \\ & N_i^{(2)}: \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = p+1, \ldots, q \\ & J_i: \quad \sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = q+1, \ldots, m \\ & x_j c \geq 0,\ j \in \mathcal{J} = \{1, \ldots, s\}, \quad s \leq n. \end{aligned}$$

Pomnožimo sve nejednačine tipa $N^{(2)}$ sa -1 i uvedimo dodatne promenljive $x_{n+1}, \ldots, x_{n+q}$ kao kod svođenja na standardni oblik. Formalno, označimo promenljive sa leve strane tabele sa $x_{B,i}$ a one sa desne $x_{N,j}$. Ovim smo dobili problem u obliku koji podseća na kanonski.

$$\begin{aligned} \max\ & z(x) = c_1 x_{N,1} + \ldots + c_n x_{N,n} + d \\ \text{p.o.}\ & a_{11} x_{N,1} + a_{12} x_{N,2} + \ldots + a_{1n} x_{N,n} - b_1 = -x_{B,1} \\ & a_{21} x_{N,1} + a_{22} x_{N,2} + \ldots + a_{2n} x_{N,n} - b_2 = -x_{B,2} \\ & \ldots\ldots\ldots \\ & a_{q1} x_{N,1} + a_{q2} x_{N,2} + \ldots + a_{qn} x_{N,n} - b_m = -x_{B,q} \\ & a_{q+1,1} x_{N,1} + a_{q+1,2} x_{N,2} + \ldots + a_{q+1,n} x_{N,n} - b_{q+1} = -0 \\ & \ldots\ldots\ldots \\ & a_{m1} x_{N,1} + a_{m2} x_{N,2} + \ldots + a_{mn} x_{N,n} - b_m = -0 \end{aligned} \tag{2.10.1}$$

Razlika je samo u jednačinama $q+1,\ldots,m$. Ako pretpostavimo da su njima dodeljene dodatne promenljive koje su identički jednake nuli, dobijamo problem u kanonskom obliku. Ekvivalentna Tuckerova tabela ima sledeći izgled:

$$\begin{array}{ccccccc} x_{N,1} & x_{N,2} & \cdots & x_{N,n} & -1 & & \\ a_{11} & a_{12} & \cdots & a_{1n} & b_1 & = & -x_{B,1} \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 & = & -x_{B,2} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{q1} & a_{q2} & \cdots & a_{qn} & b_q & = & -x_{B,q} \\ a_{q+1,1} & a_{q+1,2} & \cdots & a_{q+1,n} & b_{q+1} & = & -x_{B,q+1} \equiv 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m & = & -x_{B,m} \equiv 0 \\ c_1 & c_2 & \cdots & c_n & d & = & z \end{array} \tag{2.10.2}$$

Posmatrajmo sada jednu takvu jednačinu (npr. poslednju).

$$a_{m1}x_{N,1} + a_{m2}x_{N,2} + ... + a_{mn}x_{N,n} - b_m = -x_{B,m} = -0$$

Ukoliko su svi $a_{q+1,j} = 0$ onda je ova jednačina ili nemoguća, pa je problem nedopustiv, ili identitet, pa se može jednostavno izbaciti iz problema. Neka je sada $a_{mj} \neq 0$ za neko j. Ukoliko zamenimo promenljive $x_{B,m}$ i $x_{N,j}$ dobijamo:

$$\begin{array}{ccccccc} x_{N,1} & \cdots & x_{B,m} \equiv 0 & \cdots & x_{N,n} & -1 & & \\ a_{11} & \cdots & a_{1j} & \cdots & a_{1n} & b_1 & = & -x_{B,1} \\ \vdots & & \vdots & & \vdots & \vdots & & \vdots \\ a_{m1} & \cdots & a_{mj} & \cdots & a_{mn} & b_m & = & -x_{N,j} \\ c_1 & \cdots & c_j & \cdots & c_n & d & = & z \end{array} \tag{2.10.3}$$

Očigledno, sada nezavisna promenljiva $x_{B,m} \equiv 0$ ne utiče ni na funkciju cilja ni na uslove. Sledi da j-tu kolonu tabele (2.10.3) možemo izbaciti. Time smo broj nebazičnih promenljivih smanjili za 1. Ovaj postupak ponavljamo sve dok bazične promenljive koje su identične nuli ne eliminišemo u potpunosti. Može se opisati sledeći algoritam.

Algoritam ElJed (Eliminacija jednačina)

Korak 1. Ako ne postoji nijedna bazična promenljiva identički jednaka nuli, primeniti algoritam **NoBasicMax**.

Korak 2. Neka je $x_{B,i} \equiv 0$. Nađimo $a_{ij} \neq 0$. Ako takvo ne postoji, i ako je $b_i = 0$, izbaciti i-tu jednačinu i preći na korak 1, u suprotnom STOP. Problem je nedopustiv.

Korak 3. Zameniti promenljive $x_{B,i}$ i $x_{N,j}$, izbaciti j-tu kolonu, smanjiti n za 1, i preći na korak 1.

Na sličan način vršimo eliminaciju slobodnih promenljivih. Ako je neka od bazičnih promenljivih slobodna, npr. $x_{B,1}$, njenu vrednost uvek možemo izračunati iz jednačine:

$$a_{11}x_{N,1} + a_{12}x_{N,2} + ... + a_{1n}x_{N,n} - b_1 = -x_{B,1}$$

za proizvoljne vrednosti nebazičnih promenljivih. Znači, ova jednačina je uvek zadovoljena, te može biti izbačena iz problema.

Pretpostavimo da smo na ovaj način eliminisali sve bazične slobodne promenljive. Takođe, pretpostavimo da je neka nebazična promenljiva (npr. $x_{N,j}$) slobodna. Neka je najpre $a_{ij} = 0$ za svako $i = 1, \ldots, m$. Ako je $c_j \neq 0$ onda je funckija cilja neograničena ako je problem dopustiv. U suprotnom, promenljiva $x_{N,j}$ nema uticaja ni na funkciju cilja, pa može biti izbačena.

Neka je sada $a_{ij} \neq 0$ za neko $i \in \{1, \ldots, m\}$. Izvršimo zamenu promenljivih $x_{B,i}$ i $x_{N,j}$. Posle transformacije, promenljiva $x_{N,j}$ postaje bazična, pa se eliminiše kao u prethodnom slučaju. Prema tome imamo:

Algoritam ElSl (Eliminacija slobodnih promenljivih)

Korak 1. Ukoliko nema slobodnih promenljivih, primeniti algoritam **ElJed**.

Korak 2. Neka je promenljiva $x_{B,i}$ slobodna. Izbaciti i-tu jednačinu (vrstu) i preći na korak 1.

Korak 3. Neka je promenljiva $x_{N,j}$ slobodna. Ako je $a_{ij} = 0$ za svako $i = 1, \ldots, m$, preći na korak 2'. U suprotnom izabrati $a_{ij} \neq 0$ i preći na korak 4.

Korak 3'. Ako je $c_j = 0$ izbaciti j-tu kolonu, smanjiti n za 1 i preći na korak 1. U suprotnom, izbaciti j-tu kolonu, preći na korak 1 i nastaviti sa algoritmom sve dok se ne utvrdi da li je problem dopustiv ili nije. Ako je dopustiv, STOP. Funkcija cilja je neograničena.

Korak 4. Zameniti promenljive $x_{B,i}$ i $x_{N,j}$. Sada $x_{N,j}$ postaje bazična slobodna promenljiva, eliminisati je kao u koraku 2.

Sada je uveden kompletan algoritam koji rešava polazni problem linearnog programiranja u opštem obliku. Na sledećem primeru ćemo pokazati kako se istovremeno eliminišu jednačine i slobodne promenljive.

Primer 2.10.1 Neka je data sledeća tabela:

$$T_0 = \begin{array}{cccccc} x_1 & \boxed{x_2} & x_3 & -1 & & \\ 1 & 1 & 1 & 6 & = & -0 \\ 1 & 1 & 0 & 1 & = & -x_4 \\ 1 & 2 & 1 & 0 & = & z \end{array}$$

Uokvirena promenljiva je slobodna. Odaberimo a_{12} za pivot element. Posle zamene promenljivih dobija se tabela T_1. Izbacivanjem vrste koja odgovara slobodnoj promenljivoj i kolone koja odgovara nuli, dobija se ekvivalentna tabela T_1'.

$$T_1 = \begin{array}{cccccc} x_1 & 0 & x_3 & -1 & & \\ 1 & 1 & 1 & 6 & = & -\boxed{x_2} \\ 0 & -1 & -1 & -5 & = & -x_4 \\ -1 & -2 & -1 & -12 & = & z \end{array} \qquad T_1' = \begin{array}{cccccc} x_1 & x_3 & -1 & & \\ 0 & -1 & -5 & = & -x_4 \\ -1 & -1 & -12 & = & z \end{array}$$

Primetimo da ako sada izaberemo (prema algoritmu **NoBasicMax**) a_{12} za pivot, dobijamo optimalno rešenje.

2.11 Revidirani Simpleks metod

Simpleks metod radi tako što u svakom koraku vrši zamenu promenljivih sa ciljem povećanja (smanjenja) funkcije cilja ili dobijanja dopustivog rešenja. U svakoj iteraciji vrši se zamena promenljivih, pri čemu se računaju nove vrednosti elemenata Tuckerove tabele. Prilikom te popravke, vrši se niz deljenja, što dovodi do nagomilavanja numeričke greške. Zbog ovoga simpleks metod greši u primerima u kojima je potreban veliki broj iteracija. Da bi se to izbeglo, potrebno je da se elementi Tuckerove tabele računaju pomoću polazne matrice problema. To se postiže revidiranim simpleks metodom.

Posmatramo problem linearnog programiranja u standardnom obliku (1.1.4). Neka je dato jedno bazično rešenje (dobijeno rešavanjem sistema $Ax = b$ ili metodom eliminacije jednačina) x^*. Posmatrajmo sada polazni problem u kanonskom obliku:

$$\begin{array}{ll} \max & (c^*)^T x_N - d \\ p.o. & Tx_N - b^* = -x_B \\ & x = (x_B, x_N) \geq 0 \end{array} \tag{2.11.1}$$

Označimo sada sa n i m redom broj nebazičnih i bazičnih promenljivih a sa T matricu tipa $m \times n$ koja predstavlja Tuckerovu tabelu. Vektori b^* i c^* su odgovarajući slobodni vektor i vektor funkcije cilja u kanonskom obliku. Sistem jednačina u (2.11.1) može se drugačije napisati kao $[T|I_m] \begin{bmatrix} x_N \\ x_B \end{bmatrix} = b^*$. Napišimo sada sistem $Ax = b$ u obliku $A_B x_B + A_N x_N = b$. Iz ovog uslova i (2.11.1) dobijamo direkno da važi:

$$T = A_B^{-1} A_N \qquad b^* = A_B^{-1} b. \tag{2.11.2}$$

Znači, ako znamo indekse bazičnih promenljivih $v_{B,1}, \ldots v_{B,m}$ ($x_{B,i} = x_{v_{B,i}}$) možemo da rekonstruišemo Tuckerovu tabelu pomoću formula (2.11.2). Pri tome je ispunjeno $A_B = [K_{v_{B,1}} \cdots K_{v_{B,m}}]$.

Funkciju cilja $f(x) = c^T x$ ovde možemo tretirati kao jednačinu, pri čemu moramo uvesti dodatnu promenljivu x_{n+1} tako da važi $c^T x + x_{n+1} = 0$. Sada matrica sistema ima oblik $A_c = \begin{bmatrix} A & 0 \\ c^T & 1 \end{bmatrix}$ pri čemu je $x_c = (x_1, \ldots, x_{n+1})$. Takođe je

$$(A_c)_B = \begin{bmatrix} A_B & 0 \\ c_B & 1 \end{bmatrix}.$$

Znači, ako sada primenimo formule (2.11.2) za prošireni sistem

$$A_c x_c = \begin{bmatrix} b^* \\ d \end{bmatrix},$$

u potpunosti možemo da rekonstruišemo Tuckerovu tabelu.

Kao što možemo videti, kod upravo izloženog metoda rekonstrukcije Tuckerove tabele nema nagomilavanja numeričkih grešaka, pa je ovaj metod stabilniji od

klasičnog simpleks metoda. Međutim, primetimo da u svakom koraku moramo da računamo inverznu matricu, što čini da jedna iteracija revidiranog simpleks metoda bude znatno algoritamski kompleksnija od odgovarajuće iteracije klasičnog simpleksa.

Kod većine test primera (a i problema u praksi koji se svode na linearno programiranje) matrice sistema su veoma retke (*sparse*), tj. mali je broj nenula elemenata. Na žalost, inverzne matrice *sparse* matrica u opštem slučaju ne moraju biti *sparse*. Moguće je čak konstruisati primer *sparse* matrice relativno malih dimenzija čiji inverz nema ni jednu nulu, a u većini slučajeva broj nula je veoma mali. Metodi za inverziju matrica rade veoma dugo, a inverzna matrica zahteva mnogo prostora za pamćenje.

Takođe, u algoritmima simpleks metode nije potrebno da znamo celu Tuckerovu tabelu. Dovoljno je da znamo (da rekonstruišemo) pojedine redove ili kolone Tuckerove tabele. U nastavku ćemo opisati metod kod koga smo izračunavanje potrebnog dela Tuckerove tabele sveli samo na rešavanje sistema linearnih jednačina. Označimo j-tu kolonu matrice A sa $A_{\bullet j}$ a i-ti red sa $A_{i\bullet}$. Da bi rekonstruisali j-tu kolonu dovoljno je da odgovarajuću kolonu polazne matrice pomnožimo sa A_B^{-1} odnosno treba rešiti sistem $A_B T_{\bullet j} = K_{v_{N,j}}$. Ovde moramo rešiti onoliko sistema koliko je potrebno kolona rekonstruisati. U algoritmu **NoBasicMax** potrebna nam je samo jedna kolona i vektor b. Za rekonstrukciju i-tog reda potrebno nam je da znamo i-ti red matrice A_B^{-1} odnosno [7, 58, 73]:

$$T_{i\bullet} = (A_B^{-1})_{i\bullet} A_N \tag{2.11.3}$$

Vektor $(A_B^{-1})_{i\bullet}$ određujemo iz sledeće jednačine:

$$(A_B^{-1})_{i\bullet} A_B = (I_m)_{i\bullet} \Longrightarrow A_B^T (A_B^{-1})_{i\bullet}^T = (I_m)_{i\bullet}^T \tag{2.11.4}$$

što takođe predstavlja sistem linearnih jednačina koji treba rešiti.

Napomenimo da ako za rešavanje sistema linearnih jednačina (2.11.3) i (2.11.4) koristimo metode kao što su Gausova eliminacija, LR faktorizacija, itd. [48] dovoljno je jednom da odradimo potrebne transformacije matrice A_B a zatim samo da rekonstruišemo rešenja za sve potrebne RHS vektore. U algoritmu **BasicMax** potrebno nam je da rekonstruišemo samo zadnji red (funkciju cilja), što znači da je u svakoj iteraciji algortitma **BasicMax** potrebno da rešimo 3 sistema linearnih jednačina. U algoritmu **NoBasicMax** u svakoj iteraciji potrebno je rekonstruisati jedan red i dve kolone. Kao što možemo videti, osim velike uštede u memoriji (rekonstrukciju pomoću inverzne matrice zbog pomenutog problema nema smisla implementirati pomoću *sparse* reprezentacije matrica) dobija se i na vremenu.

Formulišimo sada algoritme **BasicMax** i **NoBasicMax** "jezikom" revidiranog simpleks metoda.

Algoritam RevBasicMax (Revidirani simpleks metod za bazično dopustive kanonske oblike [73]) Formirati matricu $(A_c)_B = \begin{bmatrix} A_B & 0 \\ c_B & 1 \end{bmatrix}$ gde je c_B vektor koji se sastoji od koeficijenata funkcije cilja uz bazične promenljive.

Korak 1. Rešiti sisteme $(A_B)_c b^* = b$ i $(A_B)_c^T((A_B)_c^{-1})_{m+1\bullet}^T = (I_{m+1})_{m+1\bullet}^T$, rekonstruisati $n+1$-vi red Tuckerove tabele $c^* = ((A_c)_B^{-1})_{n+1\bullet} N_c$.

Korak 2. Ako je $c^* \leq 0$ funkcija cilja ima ekstremum. U suprotnom neka je $c_j^* > 0$

Korak 3. Rekonstruisati j-tu kolonu matrice T_c , tj rešiti sistem $A_B T_{\bullet j} = K_{v_{N,j}}$. Ako je $T_{\bullet j} \leq 0$, STOP. Funkcija cilja je neograničena.

Korak 4. Izračunati:

$$\min_{1\leq i\leq m}\left\{\frac{b_i^*}{T_{ij}} \mid T_{ij} > 0\right\} = \frac{b_p^*}{T_{pj}}$$

Korak 5. Izbaciti iz baze promenljivu $x_{B,j}$ (odnosno vektor $K_{v_{B,j}}$) a ubaciti promenljivu $x_{N,j}$ (tj vektor $K_{v_{B,j}}$). Drugim rečima, izvršiti zamenu vrednosti $v_{B,p}$ i $v_{N,j}$. Preći na korak 2.

Sledeći algoritam nije pronađen u literaturi, tako da je delimično i originalan. Međutim, korišćenje predhodnog alogritma bez sledećeg je besmisleno, jer se već u algoritmu **NoBasicMax** numeričke greške nagomilavaju.

Algoritam RevNoBasicMax (Revidirani simpleks metod za kanonske oblike koji nisu bazično dopustivi)

Korak 1. Neka je početna bazična matrica B.

Korak 2. Rešiti sistem $(A_B)_c b^* = b$. Ako je $b^* \geq 0$, primeniti algoritam **RevBasicMax**. U suprotnom izabrati $b_i^* < 0$ tako da je i maksimalno.

Korak 3. Rekonstruisati i-tu vrstu $T_{i\bullet}$ matrice T (Tuckerove tabele). Ako je $T_{i\bullet} \leq 0$, STOP. Problem linearnog programiranja je nedopustiv. U suprotnom izabrati $T_{ij} < 0$.

Korak 4. Ako je $i = m$ zameniti promenljive $x_{B,i}$ i $x_{N,j}$ i preći na korak 2.

Korak 5. U rekonstruisati j-tu kolonu matrice T, tj rešiti sistem $A_B T_{\bullet j} = K_{v_{N,j}}$

Korak 6. Izračunati

$$\min_{l>i}\left(\left\{\frac{b_i^*}{T_{ij}}\right\}\bigcup\left\{\frac{b_l^*}{T_{lj}} \mid a_{lj} > 0\right\}\right) = \frac{b_p^*}{T_{pj}}$$

Zameniti promenljive $x_{B,p}$ i $x_{N,j}$ i preći na Korak 2.

Analogno se mogu opisati i algoritmi za eliminaciju jednačina i slobodnih promenljivih. Na kraju, još jednom pomenimo prednosti i nedostatke revidiranog simpleks metoda.

Prednosti:

- Elementi Tuckerove tabele T računaju se direkno na osnovu matrice A čime se izbegava nagomilavanje greške računskih operacija.
- Pošto nam u algoritmima simpleks metode nije potrebna cela matrica T već samo pojedini redovi i kolone, revidiranom simpleks metodom mi rekonstruišemo samo te redove i kolone a ne celu matricu T.

Nedostaci:

- S obzirom da u svakom koraku moramo ili da tražimo inverznu matricu ili da rešavamo nekoliko (maksimalno 3) sistema linearnih jednačina, iteracija revidiranog simpleksa je znatno sporija od iteracije običnog.
- Kod običnog simpleksa smo koristili jednostavan algoritam za zamenu promenljivih. Ovde je potrebno implementirati kompleksne algoritme za rešavanje sistema linearnih jednačina ili inverziju matrica.

2.12 Pojam cikliranja i anticiklična pravila

U Teoremi 2.4.1 pokazali smo da posle svake iteracije algoritma **BasicMax** vrednost funkcije cilja se ili povećava ili ostaje ista. Na taj način smo "dokazali" da se isti algoritam završava u konačno mnogo iteracija (pošto ima konačno mnogo bazično dopustivih rešenja). Pri tome, nismo razmatrali mogućnost da se vrednost funkcije cilja ne menja tokom rada algoritma **BasicMax**. U tom slučaju, nemamo garancije da će se algoritam **BasicMax** završiti u konačnom vremenu. Da je to stvarno moguće pokazuje sledeći primer [84]:

Primer 2.12.1 Posmatrajmo sledeći problem linearnog programiranja u simetričnom obliku:

$$\begin{aligned} \max\ & z = \frac{3}{4}x_1 - 20x_2 + \frac{1}{2}x_3 - 6x_4 \\ \text{p.o.}\ & \frac{1}{4}x_1 - 8x_2 - x_3 + 9x_4 \leq 0 \\ & \frac{1}{2}x_1 - 12x_2 - \frac{1}{2}x_3 + 3x_4 \leq 0 \\ & x_3 \leq 1 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned} \tag{2.12.1}$$

Formirajmo odgovarajući kanonski oblik, Tuckerovu tabelu i primenimo algoritam **BasicMax** 7 puta:

$$\begin{array}{cccccccc} x_1 & x_2 & x_3 & x_4 & -1 & & \\ \frac{1}{4} & -8 & -1 & 9 & 0 & = & -x_5 \\ \frac{1}{2} & -12 & -\frac{1}{2} & 3 & 0 & = & -x_6 \\ 0 & 0 & 1 & 0 & 1 & = & -x_7 \\ \frac{3}{4} & -20 & \frac{1}{2} & -6 & 0 & = & z \end{array} \tag{2.12.2}$$

Ukoliko bi nastavili sa primenom algoritma **BasicMax**, dobijenih 6 tabela bi se stalno ponavljale i nikad ne bi došli do optimalnog rešenja. Primetimo da u koracima 2,3 i 4 izbor elementa ne mora biti jedinstven.

Upravo ovo je glavni razlog pojave cikliranja. Cikliranje se najčešće tretira kao retka pojava koja se javlja samo u veštački konstruisanim primerima [84, 79]. Međutim još 1977. godine su Kotiah i Steinberg [41] otkrili čitavu klasu "nevештačkih" problema koji cikliraju. Takođe, kako ćemo kasnije videti, prilikom testiranja naših programa uočili smo pojavu cikliranja na više primera.

U ovom odeljku ćemo proučiti pravila kojima se sprečava cikliranje. Ta pravila se nazivaju ***anticiklična pravila***. Obradićemo dve vrste anticikličnih pravila: ***Leksikografski metod*** i ***Blandova pravila***.

Definišimo ***leksikografsko pravilo*** protiv cikliranja.

Za vektor $x \in \mathbb{R}^n$, $x \neq 0$ kažemo da je *leksikografski pozitivan* ($x \overset{lex}{>} 0$) ako je njegova prva koordinata različita od nule pozitivna. Ako je $x = 0$ kažemo da je x *leksikografska nula* ($x \overset{lex}{=} 0$), a za x kažemo da je *leksikografski negativan* ($x \overset{lex}{<} 0$) ako je $-x \overset{lex}{>} 0$. Za vektor $y \in \mathbb{R}^n$ kažemo da je *leksikografski veći* od x ako je $y - x \overset{lex}{>} 0$. Analogno se uvode pojmovi *leksikografski manji od* i *leksikografski jednak*. Oznaka $\hat{x} = \text{lex-min}\, X$ znači $\hat{x} \in Z$ i $\hat{x} \overset{lex}{\leq} x$, $x \in X$. Analogno se uvodi i oznaka lex-max.

Kako je u Tuckerovoj tabeli $b_i = a_{i,n+1}$ i $c_j = a_{m+1,j}$, Korak 3 u simpleks metodu (algoritmu **BasicMax**) može se zapisati i na ovakav način:

- *Korak 3. Odrediti* $j \in \{1, \ldots, n\}$ *za koje je* $c_j < 0$. *Odrediti* $p \in \{1, \ldots, m\}$ *takvo da je*
$$\frac{a_{p,n+1}}{a_{pj}} = \min\{\frac{a_{i,n+1}}{a_{ij}} \mid a_{ij} > 0\}.$$

Posmatrajmo sada ponovo odgovarajući standardni oblik 2.1.2, pri čemu ćemo matrici A' dodati vektor b' kao nultu kolonu. Cikliranje se izbegava ako se u Koraku 3 odabere p tako da se umesto minimuma postiže leksikografski minimum. Modifikovani korak je:

- *Korak 3'. Odrediti* $j \in \{1, \ldots, n\}$ *za koje je* $c_j < 0$. *Odrediti* $p \in \{1, \ldots, m\}$ *takvo da je*
$$\frac{V_p}{a_{pj}} = \textit{lex-min}\left\{\frac{V_i}{a_{ij}} \mid a_{ij} > 0\right\},$$
gde je, podsetimo se, sa V_i *označena* i*-ta vrsta* $A'_{i\bullet}$ *matrice* A'.

Primer 2.12.2 *U slučaju Tuckerove tabele iz primera sa početka ovog odeljka, uz isto pravilo za izbor indeksa* j *sledi da je* $j = 1$, $a_{ij} > 0$ *za* $i = 1, 2$, i

$$\frac{V_1}{a_{1j}} = [0, 1, -32, -4, 36, 4, 0, 0], \quad \frac{V_2}{a_{2j}} = [0, 1, -24, -1, 6, 0, 2, 0],$$

pa se lex-min postiže za $p = 1$, *pa je prvi pivot element* a_{11}. *Primenom leksikografskog pravila se dalje dobijaju pivot elementi* $a^1_{22}, a^2_{23}, a^3_{34}$, *koji vode do Tuckerove tabele* T_4 *sa novom vrednošću ciljne funkcije i cikliranje se izbegava.*

Napomena 2.12.1 *Važno je primetiti da je lex-min uvek jedinstven jer iz pretpostavke da je* $\text{rang}A' = m$ *sledi da ne postoje dve proporcionalne vrste.*

Dokažimo sada korektnost leksikografskog pravila.

Teorema 2.12.1 *Neka su u početnoj matrici* A' *sve vrste* $V_1, \ldots, V_m$ *leksikografski pozitivne i neka je u algoritmu* **BasicMax** *korak 3 zamenjen korakom 3'. Tada je cikliranje eliminisano, tj. simpleks metod u konačnom broju iteracija dolazi ili do optimalnog rešenja ili do zaključka da ciljna funkcija nije ograničena odozdo.*

Dokaz. Pokazaćemo prvo da vrste V_i^k, $i = 1, \ldots, m$, matrice A' ostaju leksikografski pozitivne. Za $i = p$ je $V_p^1 = \frac{V_p}{a_{pj}}$ pa $a_{pj} > 0$ i $V_p \overset{lex}{>} 0$ povlači $V_p \overset{lex}{>} 0$. Za $i \neq p$ je

$$V_i^1 = V_i - \frac{a_{ij}}{a_{pj}} V_p = a_{ij} \left(\frac{V_i}{a_{ij}} - \frac{V_p}{a_{pj}} \right) \overset{lex}{>} 0,$$

gde stroga nejednakost važi na osnovu prethodne napomene. Za $i \neq p$ i $a_{ij}^k \leq 0$ je $V_i^1 = V_i + \frac{|a_{ij}|}{a_{pj}} V_p \overset{lex}{\geq} V_i \overset{lex}{>} 0$. Za $m+1$-vu vrstu važi:

$$V_{m+1}^1 = V_{m+1} - \frac{a_{m+1,j}}{a_{pj}} V_p = V_{m+1} + \frac{|a_{m+1,j}|}{a_{pj}} V_p \overset{lex}{>} V_{m+1}$$

zbog

$$a_{m+1,j} < 0,\ a_{pj} > 0 \quad \text{i} \quad V_p \overset{lex}{>} 0.$$

Dakle, $m+1$-va vrsta strogo leksikografski raste i ne može se ponoviti. □

Napomena 2.12.2 *Pretpostavka o leksikografskoj pozitivnosti vrsta matrice A' u prethodnoj teoremi nije ograničavajuća jer se to može postići u svakom kanonskom obliku linearnog programiranja. Dovoljno je, na početku, izvršiti prenumeraciju promenljivih i dovesti kolone jedinične matrice na pozicije $1, \ldots, m$.*

Ako u Algoritmu 4 umesto Koraka 3 zapisanog u obliku:

Korak 3. Odrediti $s \in \{1, \ldots, m\}$ za koje je $a_{s0}^k < 0$. Odrediti $r \in \{1, \ldots, m\}$ takvo da je $\frac{a_{0r}^k}{a_{sr}^k} = \max \left\{ \frac{a_{0j}^k}{a_{sj}^k} | a_{sj}^k < 0 \right\}$;

primenimo sledeći korak:

Korak 3'. Odrediti $s \in \{1, \ldots, m\}$ za koje je $b_s^k < 0$. Odrediti $r \in \{1, \ldots, m\}$ takvo da je $\frac{K_r^k}{a_{sr}^k} = \text{lex-max} \left\{ \frac{K_j^k}{a_{sj}^k} | a_{sj}^k < 0 \right\}$, gde je K_j^k $j-$ta kolona $k-$te dualne simpleks tablice.

Možemo dokazati konačnost dualnog simpleks metoda.

Teorema 2.12.2 *Neka su u dualnoj simpleks tablici DT_0 kolone $K_1^0, \ldots, K_n^0$ leksikografski pozitivne i neka je u Algoritmu 4 Korak 3 zamenjen Korakom 3'. Tada niz nultih kolona (K_0^k) strogo leksikografski opada i dualni simpleks metod u konačnom broju iteracija dolazi ili do optimalnog rešenja ili do zaključka da je dopustivi skup prazan.*

Razmotrimo sada ***Blandova pravila***. Po Blandu [9], treba primenjivati sledeće dve modifikacije koraka 2 i 4:

- *Korak 2'. Izabrati $c_j > 0$ tako da je indeks odgovarajuće nebazične promenljive $x_{N,j}$ najmanji.*

- *Korak 4'. Izračunati*

$$\min_{1\le i\le m}\left\{\frac{b_i}{a_{ij}},\quad a_{ij}>0\right\}=\frac{b_p}{a_{pj}}$$

U slučaju jednakih vrednosti izabrati ono p takvo da je indeks odgovarajuće bazične promenljive $x_{B,p}$ najmanji. Zameniti nebazičnu promenljivu $x_{N,j}$ i bazičnu promenljivu $x_{B,p}$ i preći na Korak 1.

Dokažimo sada ispravnost ovih pravila. U dokazu će matrica A predstavljati matricu sistema, vektori b i c RHS vektor i vektor funkcije cilja u odgovarajućem standardnom obliku problema, a matrica T proširenu Tuckerovu tabelu. Ove oznake su iste kao u odeljku 2.11.

Teorema 2.12.3 *Blandova pravila eliminišu cikliranje kod simpleks metoda.*

Dokaz. Pretpostavimo suprotno. Neka je τ skup indeksa j takvih da promenljiva x_j tokom ciklusa **postaje** bazična. Neka je $q=\max\tau$. Neka je T' tabela u kojoj je promenljiva x_q **postaje** bazična a T'' tabela gde x_q **postaje** nebazična (u samim tabelama T' i T'' promenljiva x_q je redom nebazična i bazična). Označimo sa t pivot kolonu transformacije posle koje dobijamo tabelu T''.

Definišimo vektore $y=(y_1,\dots,y_n,y_{n+1})$ i $v=(v_1,\dots,v_n,v_{n+1})$ na sledeći način:

$$y_i=\begin{cases}1 & i=n+1\\ T'_{n+1,j} & i=v'_{N,j}\\ 0 & \text{u suprotnom}\end{cases}\qquad v_i=\begin{cases}-1 & i=v''_{N,t}\\ T''_{it} & i=v''_{B,i}\\ 0 & \text{u suprotnom}\end{cases}$$

Na osnovu **prvog Blandovog pravila** važi $y_1,\dots,y_{q-1}\ge 0$ i $y_q<0$. Pošto je $T=(A_c)_B^{-1}(A_c)_N$. Ako preuredimo kolone matrica A_c i vektora y i v možemo pisati $A_c=\begin{bmatrix}(A_c)_B & (A_c)_N\end{bmatrix}$ kao i $v=\begin{bmatrix}T''_{\bullet t} & -(I_m)_{\bullet t}\end{bmatrix}^T$ i $y=\begin{bmatrix}T'_{n+1\bullet}0\end{bmatrix}^T$. Sada imamo da je:

$$A_cv=[(A_c)_B\ (A_c)_N]\begin{bmatrix}T''_{\bullet t}\\ -(I_m)_{\bullet t}\end{bmatrix}=((A_c)_N)_{\bullet t}-((A_c)_N)_{\bullet t}=0$$

Znači v pripada jezgru $\mathcal{N}(A_c)$ matrice A_c. Primetimo takođe da je vektor y baš $n+1$-va vrsta matrice $(A_c)_B^{-1}A_c$. Odavde sledi da važi $y^Tv=0$. Primetimo da je $v_{n+1}=T''_{m+1,t}<0$, pa je $y_{n+1}v_{n+1}<0$. Znači, postoji j, tako da je $y_jv_j>0$.

Pošto je $y_j\neq 0$, promenljiva x_j je bazična u T' a pošto je $v_j\neq 0$, ili je x_j nebazična u T'' ili je $j=t$. U svakom slučaju je $j\in\tau$, pa je $j\le q$. Takođe, pošto je $v_q=T''_{p''t}>0$ ($q=v''_{B,p''}$, prema koraku 4 algoritma **BasicMax**) a $y_q=T'_{m+1,p'}<0$ ($q=v'_{N,p'}$, prema koraku 2 algoritma **BasicMax**), imamo da je $y_qv_q<0$ pa je $1\ge j<q$. Pošto je $y_j\ge 0$ sledi da je i $v_j\ge 0$.

Zaključujemo da je promenljiva x_j bazična u tabeli T''. Zato neka je $j=v_{B,k}$. Sada je $T''_{kt}=v_j>0$. Primetimo takođe da se tokom cikliranja poslednja kolona

Tuckerove tabele T (vektor b^*) ne menja. Naime, pošto se vrednost funkcija cilja $T_{m+1,n+1}$ ne menja, na osnovu formula (2.3.5) zaključujemo da je $b^*_p = 0$ (p je indeks pivot vrste), pa se na osnovu istih formula ne menja ni b^*.

Vrednosti svih promenljivih iz τ u odgovarajućim bazičnim rešenjima su 0. Zaista, ako je $z \in \tau$ i x_z bazična promenljiva, uočimo trenutak kada je ona postala bazična. Tada, ako je pivot element T''_{ql}, pri čemu je $v_{N,l} = z$ i $b^*_q = 0$, posle transformacije imamo da je $v_{B,q} = z$ a b^*_q ostaje 0.

Prema tome $(b^*)''_k = T''_{k,n+1} = 0$. Znači, sada imamo da je $(b^*)''_k = 0$, $T''_{kt} > 0$, $v''_{B,k} = j$ a $v''_{B,p''} = q$ gde je p'' odgovarajuća pivot vrsta (u sledećoj iteraciji x_q postaje nebazična). Na osnovu **drugog Blandovog pravila** zaključujemo da je $q = v''_{B,p''} \leq v''_{B,k} = j$ što je kontradikcija. Ovim je teorema dokazana. □

Primer kako se pomoću ovih pravila izbegava cikliranje kod prethodnog primera može se naći u [84].

Pomenimo na kraju dva nedostatka Blandovih pravila. Primena ovih pravila može da rezultuje takvim izborom pivot elemenata da su promene vrednosti funkcije cilja male, što često uzrokuje veći broj iteracija simpleks metoda. Takođe postoji opasnost od izbora pivot elemenata koji su bliski nuli i koji prouzrokuju velike numeričke greške.

2.13 Složenost simpleks metoda i Minty-Klee poliedri

U uvodu smo napomenuli da i pored dobrih osobina koje je pokazao u praksi, simpleks algoritam nije polinomijalan. To tvrđenje su prvi dokazali Minty i Klee u radu [38], još 1970. godine uz pretpostavku da se za pivot kolonu uzima prva kolona kod koje je $c_j < 0$. Kasnije je dokazano [39] da za skoro svako determinističko pravilo izbora pivot kolone postoji klasa primera problema linearnog programiranja tako da broj iteracija simpleks metoda zavisi eksponencijalno od dimenzije problema.

Definicija 2.13.1 *Posmatrajmo sledeći problem linearnog programiranja zadat u kanonskom obliku i preko Tuckerove tabele:*

$$\begin{aligned}
\min \epsilon^{n-1}x_1 + \epsilon^{n-2}x_2 + \ldots + \epsilon x_{n-1} + x_n & \\
x_1 & \leq t \\
2\epsilon x_1 + x_2 & \leq t^2 \\
2\epsilon^2 x_1 + 2\epsilon x_2 & \leq t^3 \\
\vdots & \vdots \\
2\epsilon^{n-1}x_1 + 2\epsilon^{n-2}x_2 + \ldots + 2\epsilon x_{n-1} + x_n & \leq t^n \\
x \geq 0 &
\end{aligned}$$

$$\begin{array}{ccccccc} x_1 & x_2 & \cdots & x_n & -1 & \\ 1 & 0 & \cdots & 0 & t & = -x_{n+1} \\ 2\epsilon & 1 & \cdots & 0 & t^2 & = -x_{n+2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 2\epsilon^{n-1} & 2\epsilon^{n-2} & \cdots & 1 & t^m & = -x_{n+m} \\ \epsilon^n & \epsilon^{n-1} & \cdots & 1 & 0 & = f \end{array}$$

Označimo ovaj problem sa $\mathcal{P}_n(\epsilon, t)$ i nazovimo ga uopštenim problemom Minty-Kleea dimenzije n.

Minty i Klee su u svom radu [38] posmatrali problem $\mathcal{P}_n(2, 5)$. Očigledno, za $t > 0$, ovaj problem je bazično dopustiv pa možemo odmah primeniti algoritam **BasicMax**. Dokažimo sada da algoritam **BasicMax** posle $2^n - 1$ iteracija dolazi do optimalnog rešenja $x^* = (0, \ldots, 0, t^n)$ $(x^* \in \mathbb{R}^n)$. Upravo to tvrdi glavna teorema ovog odeljka:

Teorema 2.13.1 *Neka je $\epsilon, t > 0$ i $\frac{\epsilon}{t} > \frac{1}{2}$. Algoritam* **BasicMax**, *primenjen na problem $\mathcal{P}(\epsilon, t)$, prolazi $2^n - 1$ iteracija do optimalnog rešenja $x^* = (0, \ldots, 0, t^n)$.*

Pre nego što damo dokaz, razmotrimo neke osobine uopštenog Minty-Klee problema.

Lema 2.13.1 *Neka važe uslovi teoreme 2.13.1. Ako primenimo algoritam za zamenu promenljivih k puta, pri čemu su pivot elementi $a^i_{p_i p_i} = 1$ gde su brojevi $p_i \in \{1, \ldots, n\}$, $i = 0, \ldots, k-1$ a sa a^l_{ij} je označen element (i, j) Tuckerove tabele posle l transformacija, dobijamo Tuckerovu tabelu T_k čiji su elementi jednaki $t^k_{ij} = (-1)^{c^k_{ij}} t^0_{ij}$, za $j < n + 1$. Sa c^l_{ij} smo označili broj pivot elemenata p_s za koje važi $j \leq p_s < i$ i $1 \leq s \leq l$.*

Dokaz. Dokaz ćemo izvesti matematičkom indukcijom. Za $l = 0$ tvrđenje trivijalno važi, sobzirom da je $c^0_{ij} = 0$ za svako $(i, j) \in \{1, \ldots, m + 1\} \times \{1, \ldots, n\}$. Pretpostavimo da tvrđenje važi za sve brojeve manje ili jednake k i dokažimo ga za $k+1$. Neka je $p = p_{k+1}$. Prema indukcijskoj hipotezi, u pivot vrsti i pivot koloni p, redom su različiti od nule elementi t^k_{ip} i t^k_{pj} tako da je $i \geq p$ a $p \geq j$. Prema tome, ako bar jedan od ova dva uslova ne važi, imamo da je $t^{k+1}_{ij} = t^k_{ij}$ i $c^{k+1}_{ij} = c^k_{ij}$, jer $p \notin [j, i)$ pa tvrđenje leme važi.

Za $i = p$ važi

$$t^{k+1}_{pj} = \frac{t^k_{pj}}{t^k_{pp}} = t^k_{pj} = (-1)^{c^k_{pj}} t^0_{pj} = (-1)^{c^{k+1}_{pj}} t^0_{pj}$$

jer je $c^{k+1}_{pj} = c^k_{pj}$, za $j \geq p$. Za $j = p$ imamo da je

$$t^{k+1}_{ip} = -\frac{t^k_{ip}}{t^k_{pp}} = -t^k_{ip} = -(-1)^{c^k_{ip}} t^0_{ip} = (-1)^{c^{k+1}_{ip}} t^0_{ip}$$

jer je $c_{ip}^{k+1} = c_{ip}^k + 1$ za $i \leq p$.

Za $j < p$ i $p < i$ dobija se

$$t_{ij}^{k+1} = t_{ij}^k - t_{pj}^k t_{ip}^k = (-1)^{c_{ij}^k} t_{ij}^0 - (-1)^{c_{ip}^k + c_{pj}^k} t_{ip}^0 t_{pj}^0$$

Kako je $t_{ij}^0 = 2\epsilon^{i-j}$ i $c_{ip}^k + c_{pj}^k = c_{ij}^k$ imamo

$$\begin{aligned} t_{ij}^{k+1} &= (-1)^{c_{ij}^k} \left(t_{ij}^0 - t_{ip}^0 t_{pj}^0\right) = (-1)^{c_{ij}^k} \left(2\epsilon^{i-j} - 4\epsilon^{i-p+p-j}\right) = -(-1)^{c_{ij}^k} 2\epsilon^{i-j} \\ &= (-1)^{c_{ij}^{k+1}} t_{ij}^0. \end{aligned}$$

U poslednjoj jednakosti smo iskoristili da je $c_{ij}^{k+1} = c_{ij}^k + 1$ jer je $i > p$ i $p > j$. Ovim je lema dokazana. □

Iz dokaza prethodne leme sledi da ako za pivot vrstu uvek biramo $p = j$ (pivot kolonu biramo kao u algoritmu **BasicMax**) posle $2^n - 1$ koraka dolazimo do optimalnog rešenja. Da bi to dokazali, pretpostavimo da smo u k-tom koraku odabrali j-tu kolonu za ključnu. Tada je $T_{m+1,1}^k, \ldots, T_{m+1,j-1}^k < 0$ a $T_{m+1,j}^k > 0$. Sada izborom elementa T_{jj}^k za ključni, na osnovu dokaza prethodne leme zaključujemo da će posle transformacije važiti $T_{m+1,1}^{k+1}, \ldots, T_{m+1,j-1}^{k+1} > 0$ a $T_{m+1,j}^{k+1} < 0$. Pridružimo sada svakoj Tuckerovoj tabeli T^k jedan prirodan broj $\tau(T^k)$ na sledeći način. Cifra na l-toj poziciji u binarnom zapisu broja $\tau(T^k)$ jednaka je 0 ako je $T_{m+1,l}^k$ pozitivan, u suprotnom je jednaka 1. Upravo smo dokazali da važi $\tau(T^{k+1}) = \tau(T^k) + 1$, odnosno $\tau(T^k) = k$. Algoritam staje kada su sve cifre broja $\tau(T^k)$ jednake jedinici, odnosno kada je $k = 2^n - 1$.

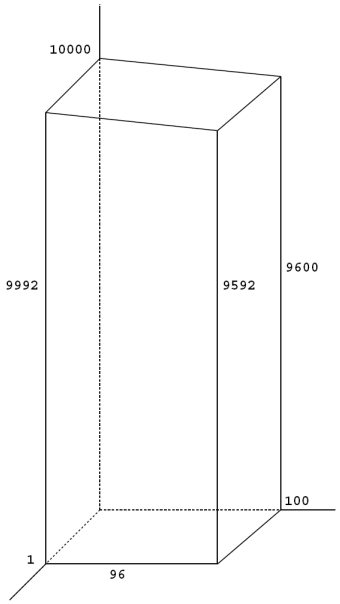
Sledeća lema, koju nećemo dokazivati završava dokaz teoreme 2.13.1:

Lema 2.13.2 *U svakoj iteraciji algoritma* **BasicMax**, *primenjenog na problem* $\mathcal{P}_n(\epsilon, t)$ *važiće* $p = j$, *tj. pivot element biće na glavnoj dijagonali Tuckerove tabele.*

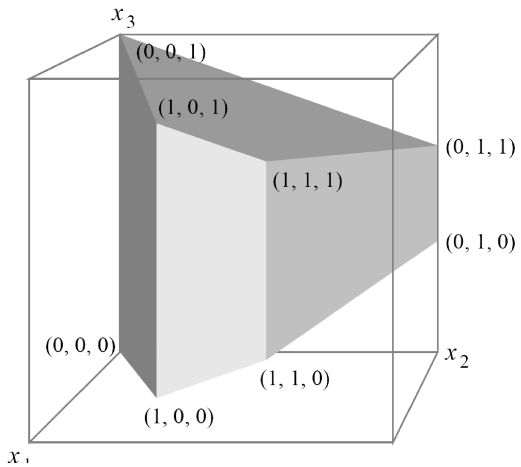
Poliedar dopustivih rešenja Ω_P za problem $\mathcal{P}(\epsilon, t)$ nazivamo *Minty-Klee poliedar*. Napomenimo, da ako izvršimo odgovarajuće smene promenljivih, Minty-Klee poliedar možemo opisati i sledećim sistemom nejednačina

$$\begin{array}{lrl} \min & x_n & \\ \text{p.o.} & & x_1 \leq 1 \\ & \epsilon x_1 \leq & x_2 \leq 1 - \epsilon x_1 \\ & \epsilon x_2 \leq & x_3 \leq 1 - \epsilon x_2 \\ & \vdots & \\ & \epsilon x_{n-1} \leq & x_n \leq 1 - \epsilon x_{n-1} \\ & x \geq 0 & \end{array}$$

Geometrijski, poslednji sistem predstavlja deformisanu n-dimenzionalnu jediničnu hiperkocku. Na slikama 2.13.1 i 2.13.2 su prikazani Minty-Klee poliedar za problem $\mathcal{P}_3(10, 100)$ kao i poliedar koji odgovara sistemu (2.13.1). Primetimo da putanja koju prolazi simpleks metod odgovara Hamiltonovom putu u odgovarajućem grafu poliedra.



Slika 2.13.1. Skup dopustivih rešenja za primer $\mathcal{P}_3(10, 100)$.



Slika 2.13.2. Minty-Klee poliedar za $\epsilon = \frac{1}{3}$.

Minty-Klee poliedri i njihove osobine proučavani od strane velikog broja autora. Uz višestruko ponavljanje odgovarajućih nejednakosti, u radu [23] je pokazano da određena klasa interior-point metoda takođe ima eksponencijalnu složenost u najgorem slučaju. U radu [28] razmatrana je varijanta simpleks metoda u kojoj se i pivot kolona i pivot vrsta biraju slučajno. U tom slučaju, izračunat je očekivani broj iteracija za primer $\mathcal{P}_n(\epsilon, t)$ koji iznosi:

$$F_n(\bar{x}) = n + 2\sum_{k=1}^{n} \frac{(-1)^{k+1}}{k+2} \binom{n-k}{2} \approx \left(\frac{\pi}{4} - \frac{1}{2}\right) n^2.$$

3 Modifikacije simpleks metoda i implementacija

U ovoj glavi pokazaćemo nekoliko originalnih modifikacija i poboljšanja pojedinih faza simpleks metoda [2] (algoritama **Replace**, **NoBasicMax**, **ElJed**, **ElSl**). Ukazaćemo na nekoliko nedostataka odgovarajućih algoritama i predložićemo modifikacije kod kojih nema tih nedostataka.

3.1 Dva poboljšanja simpleks metoda

U Algoritmu **Replace** opisan je postupak kojim se Tuckerova tabela transformiše u ekvivalentnu tabelu zamenom promenljivih. U praksi, Tuckerove tabele mnogih problema linearnog programiranja imaju dosta nula (oko 80%) u sebi. Broj operacija koje izvrši algoritam je $(m+1)(n+1)$ i ne zavisi od strukture same tabele. Primetimo sada da, pri transformaciji, mali broj elemenata promeni vrednost, pošto su mnogi elementi u Tuckerovoj tabeli jednaki nuli. Pošto se za ključni element a_{pj} uvek bira broj različit od nule, to ovaj elemenat uvek menja vrednost (osim kad je jednak 1). Elementi koji se nalaze u istoj vrsti ili istoj koloni sa ključnim elementom posle transformacije redom postaju jednaki:

$$\begin{aligned} a^1_{pl} &= \frac{a_{pl}}{a_{pj}}, \quad l \neq j; \\ a^1_{qj} &= -\frac{a_{qj}}{a_{pj}}, \quad q \neq p; \end{aligned} \tag{3.1.1}$$

Odavde se vidi da će oni menjati vrednosti ako i samo ako su različiti od nule. Takođe, svaki elemenat koji nije ni u istoj vrsti ni u istoj koloni sa ključnim postaje jednak:

$$a^1_{ql} = a_{ql} - \frac{a_{pl}a_{qj}}{a_{pj}}, \quad q \neq p,\ l \neq j$$

On će menjati vrednost ako i samo ako su projekcije na ključnu vrstu a_{pl} i a_{qj} različite od 0.

Znači, bilo koji element iz Tuckerove tabele se menja akko su obe njegove projekcije različite od 0. Konstruišimo sada skupove V i K na sledeći način:

$$\begin{aligned} V &= \{l \mid a_{pl} \neq 0,\ l = 1, \ldots, n+1\}, \\ K &= \{q \mid a_{qj} \neq 0,\ q = 1, \ldots, m+1\}. \end{aligned} \tag{3.1.2}$$

[2] Rezultati izloženi u ovoj glavi su preuzeti iz naših radova [78, 58, 70]

Znači, bilo koji element iz Tuckerove tabele se menja akko su mu koordinate redom u skupovima V i K.

Algoritam ModReplace (Poboljšanje algoritma Replace)

Korak 1. Formirati skupove V i K.

Korak 2. Sve elemente a_{ql} takve da $q \in K$ i $l \in V$ transformisati prema algoritmu **Replace**.

Ovim smo postigli da se broj operacija smanji od $(m+1)(n+1)$ na $|V||K|$ operacija. Ako u matrici ima mnogo nula, tada i skupovi V i K imaju mali broj elemenata, pa se u tom slučaju znatno redukovao broj operacija.

Razmotrimo sada jedno poboljšanje algoritma **ElJed** za eliminaciju jednačina.

U Algoritmu **ElJed** u svakoj iteraciji mi izbacujemo po jednu kolonu matrice sistema. Za izbacivanje elementa iz matrice potrebno je izvršiti približno mn operacija. Pošto se izbacivanje vrši u svakoj iteraciji to sledi da je broj operacija koji se izvrši za izbacivanje kolona iz matrice približno jednak mnJ gde je J broj jednačina. Taj broj se drastično redukuje predloženom modifikacijom. Naime, umesto izbacivanja, kolonu možemo samo markirati. Za to koristimo logički niz *outc*. Analogno, formiramo niz *outr* kojim markiramo izbačene vrste matrice sistema. Na kraju, izbacujemo sve markirane vrste i kolone i za to nam je ukupno potrebno mn operacija što predstavlja poboljšanje za ceo red veličine. Ovaj metod se može upotrebiti i kod algoritma za eliminaciju slobodnih promenljivih **ElSl**.

Pošto su modifikovani algoritmi veoma slični originalnim, nećemo ih eksplicitno formulisati, već cemo samo napomenuti da je u koracima 2 i 3 algoritma **ElJed**, odnosno 2 i 3' algoritma **ElSl** potrebno zameniti izbacivanje vrste, odnosno kolone sa $outc(j) = \top$, odnosno $outr(j) = \top$. Na kraju algoritma treba dodati još jedan korak kojim se izbacuju sve označene vrste.

3.2 Modifikacija Simpleks metoda za probleme koji nisu bazično dopustivi

Posmatramo problem linearnog programiranja (the linear programming (LP) problem) u standardnoj formi:

$$\begin{aligned} \max \quad & \mathbf{c}^T\mathbf{x} - d, \\ \text{p.o.} \quad & A\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \geq 0, \end{aligned} \tag{3.2.1}$$

gde je $A \in \mathbb{R}^{m\times(m+n)}$ matrica potpunog ranga vrsta ($\text{rank}(A) = m$), $\mathbf{c} \in \mathbb{R}^{n+m}$ i sistem $A\mathbf{x} = \mathbf{b}$ je definisan sa $\mathbf{x} \in \mathbb{R}^{m+n}$, $\mathbf{b} \in \mathbb{R}^m$. Pretpostavlja se da je (i,j)-ti element u A označen sa a_{ij}, $\mathbf{b} = (b_1, \ldots, b_m)^T$, $d \in \mathbb{R}$, $\mathbf{x}^T = (x_1, \ldots, x_{n+m})$ i $\mathbf{c}^T = (c_1, \ldots, c_{n+m})$.

Dvofazni simpleks metod se odvija u dve faze, koje se nazivaju faza I i faza II (phase I and phase II). Faza I pokušava da pronađe jedno inicijalno bazično dopustivo rešenje (initial basic feasible solution). Kada se jedno inicijalno bazično

dopustivo rešenje pronađe, primenjuje sa faza II da bi se pronašlo optimalno rešenje. Simpleks metod čini iteracije kroz skup bazičnih rešenja (dopustiva u fazi II) problema linearnog programiranja. Svako bazično rešenje se karakteriše skupom od m basičnih promenljivih (basic variables) $x_{B,1}, \ldots, x_{B,m}$. Preostale n promenljive se nazivaju nebazične promenljive (nonbasic variables), i označavaju se $x_{N,1}, \ldots, x_{N,n}$.

Ako je $\mathbf{b} \geq 0$ i ako su sve nebazične promenljive $x_{N,1}, \ldots, x_{N,n}$ jednake nuli, tada $x_{B,1} = b_1, \ldots, x_{B,m} = b_m$ jeste bazično dopustivo rešenje. Ako uslov $\mathbf{b} \geq 0$ nije ispunjen, neophodno je da se pronađe jedno bazično dopustivo rešenje ili da se odredi da ono ne postoji. Postoji veći broj strategija za fazu I. Klasični pristup se sastoji u tome da se sa standardnim linearnim problemom asocira sledeći prošireni problem:

$$\begin{array}{ll} \min & \mathbf{ew}, \\ \text{p.o.} & A\mathbf{x} + \mathbf{w} = \mathbf{b}, \\ & \mathbf{x} \geq 0,\ \mathbf{w} \geq 0, \end{array} \tag{3.2.2}$$

gde je $\mathbf{e} = (1, \ldots, 1) \in \mathbb{R}^m$ i $\mathbf{w} \in \mathbb{R}^m$ je vektor veštačkih promenljivih (artificial variables). Poznato je da ako je $(\mathbf{x}^*, \mathbf{w}^*)$ optimalno rešenje problema (3.2.2), tada neophodan i dovoljan uslov da (3.2.1) ima dopustivo rešenje jeste $w_i^* = 0, \quad i = 1, \ldots, m$ [5, 33]. Tada nema veštačkih promenlivih u finalnoj bazi, veštačke promenljive i odgovarajuće kolone se eliminišu, i dopustivo rešenje početnog LP problema je generisano. Nedostatak ovog pristupa jeste upotreba veštačkih promenljivih, jer se u fazi I dodaje m novih veštačkih promenljivih $w = (w_1, \ldots, w_m)$, po jedna za svako ograničenje.

Druga varijanta dvofaznog simpleks metoda (two-phase simplex method) je definisana u [51] i [79], i opisana u odeljcima 2.3, 2.4 i 2.5. U radovima [70], [78] korišćen je taj algoritam, jer ne zahteva uvođenje veštačkih promenljivih. U tim radovima su uvedena dva algoritma za dobijanje početnog bazično dopustivog rešenja u fazi I dvofaznog simpleks algoritma opisanog u [51] i [79]. Opisano je novo pravilo za izbor bazičnih i nebazičnih promenljivih, tj. za izbor promenljive koja ulazi u bazu kao i promenljive koja napušta bazu. Naš konačni cilj je da se minimizira ukupan broj iteracija u simpleks algoritmu. Međutim, taj cilj je nepristupačan. Prema tome, naš cilj je da se optimizira tekući simpleks korak. Na taj način je poboljšana efikasnost simpleks algoritma, što je potvrđeno numeričkim primerima.

Posmatraćemo kanonički oblik problema linearnog programiranja (kao i u odeljku 2.3) zadatog pomoću Tuckerove tabele.

$x_{N,1}$	$x_{N,2}$	$\ldots$	$x_{N,n}$	-1	
a_{11}	a_{12}	$\ldots$	a_{1n}	b_1	$= -x_{B,1}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
a_{m1}	a_{m2}	$\ldots$	a_{mn}	b_m	$= -x_{B,m}$
c_1	c_2	$\ldots$	c_n	d	$= z$

(3.2.3)

U ovom odeljku razmotrićemo modifikaciju metoda za nalaženje početnog bazično

dopustivog rešenja (algoritma **NoBasicMax**). Ovi rezultati su preuzeti iz naših radova [70, 78]. Možemo uočiti dva nedostatka algoritma **NoBasicMax**:

1. Ako je $p = i$ i ako postoji indeks $t < i = p$ tako da je

$$\frac{b_t}{a_{tj}} < \frac{b_p}{a_{pj}}, \quad b_t > 0,\ -a_{tj} < 0$$

u sledećoj iteraciji promenljiva $x_{B,t}$ postaje negativna:

$$x_{B,t} = b_t^1 = b_t - \frac{b_i}{a_{ij}} a_{tj} < 0.$$

2. Ako je $p > i$, i u sledećoj iteraciji slobodni koeficijent b_i^1 je negativan, ali može da postoji $b_t < 0$, $t < i$ tako da je

$$\min_{k>t} \left(\left\{ \frac{b_t}{a_{tj}},\ -a_{tj} > 0 \right\} \cup \left\{ \frac{b_k}{a_{kj}} \mid -a_{kj} < 0,\ b_k > 0 \right\} \right) = \frac{b_t}{a_{tj}}.$$

U tom slučaju je moguće izabrati a_{tj} za pivot element i dobijamo

$$x_{B,t} = b_t^1 = \frac{b_t}{a_{tj}} \geq 0.$$

Takođe, kako je

$$\frac{b_t}{a_{tj}} \leq \frac{b_k}{a_{kj}},$$

svaki $b_k > 0$ ostaje pogodan za bazično dopustivo rešenje:

$$x_{B,k} = b_k^1 = b_k - \frac{b_t}{a_{tj}} a_{kj} \geq 0.$$

Iz tih razloga predlažemo modifikaciju koraka 4. Glavna ideja je sadržana u sledećoj lemi:

Lema 3.2.1 *Neka je problem 3.2.3 dopustiv, neka je* $b_{i_1}, \ldots, b_{i_q} < 0$ *i neka je* $I = \{i_1, \ldots, i_q\}$. *U sledeća dva slučaja:*

a) $q = m$,

b) $q < m$ *i postoji* $r \in I$ *i* $s \in \{1, \ldots, n\}$ *tako da važi:*

$$\min_{h \notin I} \left\{ \frac{b_h}{a_{hs}} \mid -a_{hs} < 0 \right\} \geq \frac{b_r}{a_{rs}},\ -a_{rs} > 0, \tag{3.2.4}$$

moguće je dobiti novo bazično rešenje $x^1 = \{x_{B,1}^1, \ldots, x_{B,m}^1\}$ *sa najviše* $q - 1$ *negativnih koordinata u samo jednom iterativnom koraku simpleks metoda, ako se* a_{rs} *izabere za pivot element, tj. ako zamenimo nebazičnu promenljivu* $x_{N,s}$ *bazičnom promenljivom* $x_{B,r}$.

Dokaz. a) Ako je $q = m$, izaberemo proizvoljan koeficijent $-a_{ms} > 0$ za pivot element. Primenom simpleks metoda dobijamo novo rešenje sa najmanje jednom koordinatom pozitivnom. Na primer, imamo

$$x^1_{B,m} = b^1_m = \frac{b_m}{a_{ms}} \geq 0.$$

b) Preptostavimo sada da su uslovi $q < m$ i (3.2.4) zadovoljeni. Izaberemo a_{rs} za pivot element. U tom slučaju koordinate novog bazičnog rešenja su

$$x^1_{B,i} = b^1_i = b_i - \frac{b_r}{a_{rs}} a_{is}, \quad i \neq r,$$
$$x^1_{B,r} = b^1_r = \frac{b_r}{a_{rs}}.$$

Za $k \neq r$, $k \notin I$ i $a_{ks} < 0$ očigledno je da je

$$x^1_{B,k} = b_k - \frac{b_r}{a_{rs}} a_{ks} \geq 0.$$

Za $a_{ks} > 0$ i $k \notin I$, iz

$$\frac{b_k}{a_{ks}} \geq \frac{b_r}{a_{rs}}$$

odmah sledi

$$x^1_{B,k} = b^1_k = b_k - \frac{b_r}{a_{rs}} a_{ks} \geq 0$$

što je i trebalo pokazati. □

U skladu sa rezultatom iz Leme 3.2.1 predlažemo sledeću modifikaciju koraka 4 koja smanjuje broj negativnih koordinata u novom bazičnom rešenju za najmanje jedan u svakoj iteraciji, ukoliko izaberemo pivot element a_{rs} tako da b_r i a_{rs} zadovoljavaju uslov (3.2.4).

- *Korak 4'. Neka je $b_{i_1}, \ldots, b_{i_q} < 0$. Ako je $q = m$, izabrati proizvoljan koeficijent $a_{ms} < 0$ za pivot element.*

  Ako je $q < m$, razmotriti $a_{rs} < 0$ za $r \in \{i_1, \ldots, i_q\} = I$. Ako postoje $r \in I$ i $s \in \{1, \ldots, n\}$ tako da je uslov (3.2.4) *zadovoljen, tada izabrati a_{rs} za pivot element.*

Napomena 3.2.1 Ako ne postoji r ako da važi uslov (3.2.4), neka su $r \notin I$ i $s \in \{1, \ldots, n\}$ takvi da je

$$\min_{h \notin I} \left\{ \frac{b_h}{a_{hs}} \mid -a_{hs} < 0 \right\} = \frac{b_r}{a_{rs}}.$$

Za takvo r i s izabrati a_{rs} za pivot element. Primenom simpleks transformacije dobija se novo rešenje x^1 sa nenegativnim koordinatama

$$x^1_{B,i} = b^1_i = b_i - \frac{b_r}{a_{rs}} a_{is}, \; i \notin I.$$

Prema tome, broj negativnih koordinata u novom bazičnom rešenju (broj negativnih vrednosti b_i) u opštem slučaju ostaje isti.

Na osnovu predhodne napomene možemo zaključiti da i ovde postoji opasnost od cikliranja. Sada ćemo pokazati jedno anticiklično pravilo koje u ovom slučaju možemo primeniti [70].

S obzirom da je i_s fiksirano u koracima 4, 5 i 6, algoritam **ModNoBasicMax** može da ciklira samo ako je $b_{i_s}^1 = b_{i_s}$. Ako sada primenimo Blandova pravila pri čemu i_s-ti red smatramo funkcijom cilja, dobićemo da će posle konačno mnogo iteracija ili biti ispunjen uslov Leme 3.2.1 ili da će svi $a_{i_s,j}$ biti pozitivni, odnosno problem će biti nedopustiv.

U skladu sa ovim razmatranjima, predlažemo sledeće poboljšanje algoritma **NoBasicMax**.

Algoritam ModNoBasicMax (Modifikacija algoritma **NoBasicMax**)

Korak 1. Ako je $b_1, \ldots, b_m \geq 0$ preći na korak 7.

Korak 2. Konstruisati skup

$$B = \{b_{i_1}, \ldots, b_{i_e}\} = \{b_{i_k} \mid b_{i_k} < 0,\ k = 1, \ldots, e\}.$$

Korak 3. Izabrati proizvoljno $b_{i_s} < 0$.

Korak 4. Ako je $a_{i_s,1}, \ldots, a_{i_s,n} \geq 0$ tada STOP. Problem linearnog programiranja nema rešenja. U suprotnom, konstruisati skup

$$Q = \{a_{i_s,j_p} < 0,\ k = 1, \ldots, t\},$$

i postaviti $p = 1$.

Korak 5. Naći minimum

$$\min_{1 \leq k \leq m} \left\{ \frac{b_k}{a_{k,j_p}} \mid a_{k,j_p} > 0,\ b_k > 0 \right\} = \frac{b_u}{a_{u,j_p}}.$$

Ako je

$$\frac{b_{i_s}}{a_{i_s,j_p}} \leq \frac{b_u}{a_{u,j_p}}$$

zameniti nebazičnu i bazičnu promenljivu x_{N,j_p} i x_{B,i_s} i preći na korak 2. Vrednost b_{i_s} postaje pozitivna.

Korak 6. Ako je $p \leq t$ staviti $p = p+1$ i preći na korak 5. U suprotnom zameniti promenljive x_{N,j_p} i $x_{B,u}$ i preći na korak 4. Vrednost b_{i_s} je i dalje negativna.

Korak 7. Primeniti simpleks algoritam za bazično dopustive probleme, algoritam **BasicMax**.

Prednost algoritma **ModNoBasicMax** u odnosu na algoritam **NoBasicMax** je u tome što se kod algoritma **ModNoBasicMax** broj negativnih vrednosti b_i ne povećava prilikom iteracija, dok kod **NoBasicMax** to ne mora da bude slučaj. Mada, kako se pokazalo u praksi (videti odeljak 3.5) ova modifikacija ponekad može da ima kontraefekat: da veoma brzo dođe do bazično dopustivog problema, ali da posle algoritam **BasicMax** napravi mnogo više iteracija nego kada se primeni algoritam **NoBasicMax**.

3.3 Poboljšana verzija modifikacije za probleme koji nisu bazično dopustivi

U našem radu [70], uočili smo da algoritam **ModNoBasicMax** možemo još malo poboljšati. Naime, u algoritmu **ModNoBasicMax** vrednost i_s je fiksirana. Zato može da se dogodi da uslov Leme 3.2.1 nije zadovoljen za $i = i_s$ ali da postoji neko drugo $b_i < 0$ tako da je $a_{ij_p} < 0$ i da je uslov leme zadovoljen. Tada je bolje izabrati a_{ij_p} za pivot jer onda b_i postaje pozitivno.

Posmatrajmo šta se dešava sa vrednostima b_l posle zamene promenljivih $x_{B,i}$ i $x_{N,j}$:

$$b_l^1 = b_l - \frac{a_{lj}b_i}{a_{ij}} = a_{lj}\left(\frac{b_l}{a_{lj}} - \frac{b_i}{a_{ij}}\right).$$

Neka je sad $\frac{b_i}{a_{ij}} \geq 0$ (b_i i a_{ij} su istog znaka). Ako je $b_l > 0$, da bi ono ostalo pozitivno posle iteracije mora biti ili $a_{lj} < 0$ ili $a_{lj} > 0$ i $\frac{b_l}{a_{lj}} \geq \frac{b_i}{a_{ij}}$. Međutim, ako je $b_l < 0$, ono postaje pozitivno ako su a_{lj} i $\frac{b_l}{a_{lj}} - \frac{b_i}{a_{ij}}$ istog znaka. Ovo razmatranje nas navodi na sledeću strategiju izbora pivot elementa:

- Najpre moramo obezbediti da elementi b_l koji su pozitivni takvi i ostanu. Zato mora da bude

$$\frac{b_i}{a_{ij}} \leq \min\left\{\frac{b_k}{a_{kj}} \mid b_k > 0, a_{kj} > 0\right\}.$$

- Da bi negativan b_l postao pozitivan mora da je ili $a_{lj} > 0$ ili

$$\frac{b_l}{a_{lj}} \leq \frac{b_i}{a_{ij}}.$$

Na osnovu ovoga konstruišemo sledeći algoritam [70]:

Algoritam AdvModNoBasicMax

(Poboljšana verzija algoritma **ModNoBasicMax**)

Korak 1. Ako je $b_1, \ldots, b_m \geq 0$ preći na korak 5.

Korak 2. Konstruisati skup

$$B = \{b_{i_1}, \ldots, b_{i_e}\} = \{b_{i_k} \mid b_{i_k} < 0,\ k = 1, \ldots, e\}.$$

Korak 3. Neka je $s = 1$.

Korak 3.1 Ako je $a_{i_s,1}, \ldots, a_{i_s,n} \geq 0$ onda STOP. Problem je nedopustiv. U suprotnom konstruisati skup

$$Q = \{a_{i_s,j_p} < 0,\ p = 1, \ldots, t\},$$

i postaviti $p = 1$.

Korak 3.2 Naći minimume

$$M(j_p) = \min\left\{\frac{b_k}{a_{k,j_p}} \,|\, b_k > 0, a_{k,j_p} > 0\right\}.$$
$$p' = \operatorname{argmin}\left\{\frac{b_k}{a_{k,j_p}} \,|\, b_k < 0, a_{k,j_p} < 0\right\}.$$

Ako je $\frac{b_k}{a_{k,j_p}} \leq M(j_p)$ odabrati a_{p',j_p} za pivot element, izvršiti zamenu promenljivih x_{B,j_p} i $x_{N,p'}$ i preći na korak 1. (U sledećoj iteraciji b_k postaje pozitivno).

Korak 3.3 Ako je $p < t$ onda staviti $p = p + 1$ i preći na korak 3.2.

Korak 3.4 Ako je $s < e$ onda staviti $s = s + 1$ i preći na korak 3.1.

Korak 4. (Uslov leme 3.2.1 nije zadovoljen ni za jedno i_s i j_p) Označimo sa $v_{N,j}$ indeks bazične promenljive $x_{N,j}$ (odnosno takvu vrednost da važi $x_{v_{N,j}} = x_{N,j}$). Određujemo pivot prema Blandovim pravilima.

Korak 4.1 Naći $j_0 = \operatorname{argmin}\left\{v_{N,l} \,|\, a_{i_q,l} < 0\right\}$.

Korak 4.2 Naći

$$p'' = \operatorname{argmin}\left\{v_{B,p} \,|\, \frac{b_p}{a_{p,j_0}} = M(j_0)\right\}.$$

Korak 4.3 Izvršiti zamenu promenljivih $x_{B,j}$ i $x_{N,p''}$ i preći na korak 3.

Korak 5. Primeniti simpleks algoritam za bazično dopustive probleme, algoritam **BasicMax**.

Primetimo da u koraku 3.2 minimumi koje računamo zavise samo od j_p (ne zavise od i_s). Takođe, može se desiti da za nekoliko različitih vrednosti i_s nepotrebno računamo iste vrednosti za isto j_p. U tom slučaju, treba jednostavno preskočiti ponovljenu vrednost j_p i preći na sledeću (odnosno na korak 3.3).

3.4 Modifikacija revidiranog simpleks metoda

Kao i u odeljku 2.11 posmatramo problem linearnog programiranja u standardnom obliku:

$$\begin{aligned} &\min c^T x + d, \\ &p.o. Ax = b, \\ &x \geq 0, \end{aligned} \tag{3.4.1}$$

U odeljcima 3.2 i 3.3 razmotrili smo modifikaciju algoritma za nalaženje bazično dopustivog rešenja. Nedostaci koje smo uočili kod algoritma **NoBasicMax** važe i ovde i odnose se na algoritam **RevNoBasicMax**. Pri čemu, ovde moramo uzeti u obzir da nemamo celu Tuckerovu tabelu na raspolaganju već samo jedan njen deo. U ovom odeljku ćemo opisati modifikaciju algoritma **RevNoBasicMax** baziranu na već obrađenim modifikacijama kod klasičnog simpleks metoda.

Ovde je glavna ideja da nađemo minimum:

$$\frac{b_p^*}{T_{pj}} = \min\left\{\frac{b_k^*}{T_{kj}} \mid b_k^* < 0, T_{kj} < 0, k = 1, \ldots, m\right\}$$

i ako je relacija (3.4.2) zadovoljena, onda zamenom promenljivih $x_{B,p}$ i $x_{N,j}$ dobijamo novo bazično rešenje sa manjim brojem negativnih koordinata $(b^*)_i^1$.

Zato predlažemo modifikaciju koraka 6. algoritma **RevNoBasicMax**.

Lema 3.4.1 *Neka je problem* (3.4.1) *dopustiv i neka je x bazično nedopustivo rešenje sa q negativnih koordinata. Tada postoji $T_{ij} < 0$. Takođe, u sledeća dva slučaja:*

a) $q = m$,

b) $q < m$ *and*

$$\begin{aligned} Neg = &\left\{\frac{b_k^*}{T_{kj}} \mid b_k^* < 0, T_{kj} < 0, k = 1, \ldots, m\right\}, \\ Pos = &\left\{\frac{b_k^*}{T_{kj}} \mid b_k^* > 0, T_{kj} > 0, k = 1, \ldots, m\right\}, \\ &\min\left(Neg \cup Pos\right) = \frac{b_r^*}{T_{rj}} \in Neg \end{aligned} \tag{3.4.2}$$

moguće je naći novo bazično rešenje sa najviše $q-1$ negativnih koordinata u samo jednoj iteraciji revidiranog simpleks metoda, ako odaberemo T_{rj} za pivot element, tj. zamenimo promenljive $x_{N,j}$ i $x_{B,r}$.

Dokaz. Prvi deo teoreme koji je vezan za postojanje elementa $T_{ij} < 0$ je ekvivalentan lemi 3.2.1.

Dokažimo drugi deo teoreme.

a) Ako je $q = m$, za proizvoljni pivot elemenat $T_{ij} < 0$ dobijamo novo bazično rešenje sa bar jednom pozitivnom koordinatom:

$$(b^*)_i^1 = \frac{b_i^*}{T_{ij}} > 0.$$

b) Neka sada važi $q < m$ i (3.4.2). Odaberimo T_{rj} za pivot elemenat.

Za $b_k^* > 0$ i $T_{kj} < 0$ je trivijalno

$$(b^*)_k^1 = b_k^* - \frac{b_r^*}{T_{rj}} T_{kj} > b_k^* \geq 0.$$

Za $b_k^* > 0$ i $T_{kj} > 0$, korišćenjem $\frac{b_k^*}{T_{kj}} \geq \frac{b_r^*}{T_{rj}}$, odmah dobijamo

$$(b^*)_k^1 = b_k^* - \frac{b_r^*}{T_{rj}} T_{kj} \geq 0.$$

Znači svi pozitivni b_k^* ostaju pozitivni. Za $b_r^* < 0$ dobijamo

$$(b^*)_r^1 = \frac{b_r^*}{T_{rj}} \geq 0.$$

Ovim je dokaz završen. □

Napomena 3.4.1 Neka uslov (3.4.2) ne važi, tj. važi $\min(Neg \cup Pos) = \frac{b_r^*}{T_{rj}} \in Pos$. Ako odaberemo T_{rj} za pivot elemenat imamo

$$(b^*)_k^1 = b_k^* - \frac{b_r^*}{T_{kj}} T_{rj} \geq 0,$$

za $(b^*)_k > 0$ i T_{rj} bilo pozitivno ili negativno. Ali za negativno $b_k^* > 0$ možemo na sličan način dokazati

$$(b^*)_k^1 = b_k^* - \frac{b_r^*}{T_{kj}} T_{rj} < 0.$$

za T_{rj} bilo pozitivno ili negativno. Znači, naše novo rešenje ima isti broj negativnih koordinata q kao prethodno. I u ovom slučaju primenom odgovarajućih anticikličnih pravila kao u odeljku 3.3 omogućavamo da algoritam završi rad u konačnom vremenu.

Na osnovu razmatrane teorije konstruišemo sledeći algoritam.

Algoritam ModRevNoBasicMax (Modifikacija algoritma **RevNoBasicMax**)

Korak 1. Neka su A_B i A_N bazična i nebazična matrica. Rekonstruisati vektor $b^* = A_B^{-1} b$.

Korak 2. Konstruisati skup

$$B = \{b_{i_1}^*, \ldots, b_{i_q}^*\} = \{b_{i_k}^* \mid b_{i_k}^* < 0,\ k = 1, \ldots, q\}.$$

Korak 3. Izabrati proizvoljno $b_{i_s}^* < 0$.

Korak 4. Rekonstruisati i_s-tu vrstu $T_{i_s \bullet}$. Ako je $T_{i_s \bullet} \geq 0$ onda STOP. Problem je nedopustiv. U suprotnom, odabrati $T_{i_s, j} \leq 0$.

Korak 5. Rekonstruisati j-tu kolonu $T_{\bullet j}$. Izračunati

$$\min_{1 \leq i \leq n} \left\{ \frac{b_i^*}{T_{ij}} \mid b_i^* T_{ij} > 0, i = 1, \ldots, m \right\} = \frac{b_p^*}{T_{pj}},$$

zameniti promenljive $x_{N,j}$ i $x_{B,p}$ i preći na korak 2.

3.5 Implementacija Simpleks metoda i rezultati testiranja programa

Simpleks algoritam (algoritmi **Replace**, **BasicMax**, **NoBasicMax**, **ElJed**, **ElSl**) kao i modifikacije (algoritmi **ModNoBasicMax**, **AdvModNoBasicMax**) implementirani su u programskom jeziku Visual Basic 6.0 [47]. Tako je nastao program

MarPlex koji, kako ćemo videti, vrlo uspešno rešava široku klasu problema linearnog programiranja.

Ulazni podaci u MarPlex-u se zadataju u MPS fajlu. Ovaj tip fajla predstavljaju svetski standard za zadavanje problema linearnog programiranja u vidu simpleks matrica (tablica). Format je dat sledećom "tabelom":

```
------------------------------------------------------------------
Polje:    1          2            3        4        5       6
Kolone :  2-3       5-12         15-22    25-36    40-47   50-61
          NAME   naziv problema
          ROWS
           tip     ime
          COLUMNS
                  ime           ime    vrednost   ime    vrednost
                  kolone        vrste             vrste
          RHS
                  ime           ime    vrednost   ime    vrednost
                  rhs           vrste             vrste
          RANGES
                  ime           ime    vrednost   ime    vrednost
                  oblasti       vrste             vrste
          BOUNDS
            tip   ime           ime
                  ogranicenja   kolone vrednost
          ENDATA
------------------------------------------------------------------
```

U sekciji `ROWS` (vrste), svaka vrsta mora da ima tip i neko simboličko ime. U sekciji `COLUMNS` (kolone) data su simbolička imena kolona sa simboličkim imenima vrsta i vrednostima (`VALUES`). RHS je deo MPS formata koji opisuje slobodne članove nejednačina posmatranog problema. Postoje još i delovi `RANGES` i `BOUNDS` kojima se zadaje opseg vrednosti koji može da uzme svaka promenljiva (ograničenja u kojima učestvuje samo jedna promenljiva. Više o samom MPS formatu može se naći na internetu, npr [52]. MarPlex je besplatan program i dostupan je na internetu na adresi

`http://tesla.pmf.ni.ac.yu/people/dexter/Software/MarPlex.zip`

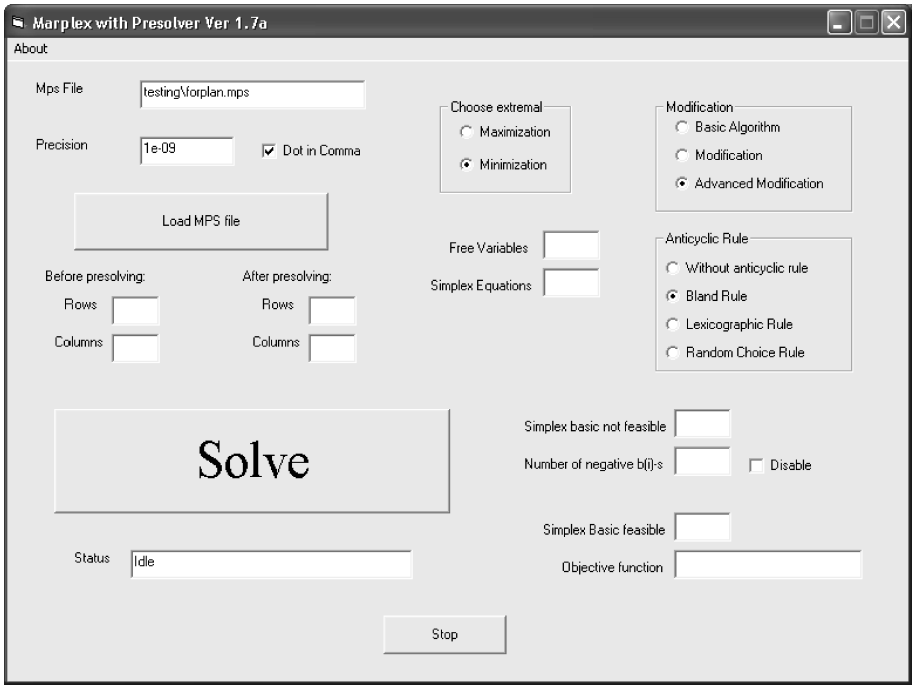
Interface MarPlex-a je prvenstveno prilagođen korisniku i omogućava udoban rad i za razliku od drugih sličnih programa ne opterećuje korisnika brojnim opcijama koje se mahom ne koriste.

Od opcija MarPlex poseduje:

- **Preciznost.** Ovo je jedna od najvažnijih opcija. Ona omogućava korisniku da zada preciznost sa kojom će program raditi. Sve vrednosti koje su po apsolutnoj vrednosti manje od broja ϵ koji se zadaje, MarPlex će tretirati kao da su jednake nuli. Time se sprečava opasan izbor malog pivot elementa u

simpleks metodu. Međutim, ukoliko je ova vrednost isuviše velika, može doći do generisanja pogrešnih rezultata.

- **Maksimizacija/Minimizacija.** Omogućava korisniku da izabere da li se rešava problem maksimizacije ili minimizacije funkcije cilja. Napominjemo da MPS fajl NE SADRŽI ovakvu specifikaciju. Prilikom testiranja uvek se radilo o minimizaciji funkcije cilja.



Slika 3.5.1. Interface programa MarPlex

- **Modifikacija.** Omogućava izbor algoritma za nalaženje početnog bazičnog rešenja. Postoje 3 opcije: Osnovni algoritam, Modifikacija ili Poboljšana modifikacija.
- **Anticiklična pravila.** Omogućava izbor anticikličnog pravila koje će se koristiti. Ponuđene su opcije: Bez pravila, Blandova pravila, Leksikografska metoda i Metoda slučajnog izbora. Kod ove poslednje, pivot se bira na slučajan način pri čemu su sve mogućnosti jednako verovatne.
- **Praćenje toka algoritma.** MarPlex ima mogućnost da se prilikom rada prate nekoliko parametara samog algoritma. To su:

  Trenutni (ukupan) broj iteracija svake faze algoritma.

  Trenutna vrednost funkcije cilja.

  Preostali broj negativnih vrednosti RHS vektora.

  Trenutni status programa (koji se algoritam trenutno izvršava).

Prilikom učitavanja podataka potrebno je naznačiti putanju do MPS fajla kao i opciju preciznost. Nakon toga, pritiskom na taster `Load MPS` vrši se učitavanje MPS fajla i presolving. Presolving je postupak u kome se problem što je moguće više uprošćava, eliminisanjem suvišnih podataka. Detaljnije o presolvingu u linearnom programiranju moše se naći u [93] gde je opisan presolver koji koristi program PCx. Presolver koji smo implementirali u MarPlex-u koristi modifikovanu verziju onog iz [93]. MarPlex ima opciju prikazivanja broja vrsta i kolona problema pre i posle presolvera. Pritiskom na taster `Solve` pokreće se solver i pristupa se rešavanju problema linearnog programiranja.

Program MarPlex smo testirali na referentnim svetskim *Netlib* test problemima. U sledećoj tabeli su prikazani rezultati testiranja. Za svaki problem smo rezervisali tri reda u tabeli: u prvom redu su rezultati postignuti primenom poboljšane modifikacije (algoritam **AdvModNoBasicMax**), zatim klasičnog algoritma **NoBasicMax** i na kraju modifikacije (algoritam **ModNoBasicMax**). Crtica u tabeli ukazuje da je progam dao pogrešan rezultat kao posledicu nagomilavanja računskih grešaka.

Brojevi iteracija za nalaženje bazično dopustivog rešenja, za nalaženje optimalnog rešenja (algoritam **BasicMax**) i ukupan broj iteracija dati su u kolonama označenim redom sa Bf., Sim. and Sum. U zadnjoj koloni su rezultati dobijeni programom PCx [93]. Napomenimo još jednom da je PCx baziran na primal-dual interior point metodu i predstavlja jedan od najjačih i najrobustnijih solvera za problem linearnog programiranja.

Name	Bf.	Sim.	Bf.+Sim.	Objective value	PCx
adlittle	57	44	101	225494.963162364	2.25494963e+005
	77	38	115	225494.96316238	
	21	54	76	225494.963162379	
afiro	4	8	12	−464.753142857143	−4.64753143e+002
	17	5	22	−464.753142857143	
	2	9	11	−464.753142857143	
agg	67	22	89	−35991767.2865765	−3.59917673e+007
	84	31	115	−35991767.2865765	
	38	25	63	−35991767.2865765	
agg2	40	69	109	−20239252.3559771	−2.02392521e+007
	52	64	118	−20239252.3559771	
	31	123	154	−20239252.3559771	
agg3	71	77	148	10312115.9293083	1.03121159e+007
	141	81	222	10312115.7307162	
	51	143	194	10312115.9372015	
bandm	273	159	432	−158.628018177046	−1.58628018e+002
	3128	171	3299	–	
	1495	127	1622	–	
beaconfd	1	33	34	33591.8961121999	3.35924858e+004
	1	33	34	33591.8961121999	
	1	33	34	33591.8961121999	
	1	732	733	−30.769485006264	

Name	Bf.	Sim.	Bf.+Sim.	Objective value	PCx
blend	1	732	733	−30.769485006264	−3.08121498e+001
	1	732	733	−30.769485006264	
brandy	1276	72	1348	1518.50982913344	1.51851054e+003
	2248	81	2329	–	
	624	90	714	1518.50992977114	
capri	251	120	371	2690.01291380796	2.69001291e+003
	214	138	352	2690.01291380796	
	1316	163	1479	2691.57274856721	
czprob	6933	591	7524	2185196.69885648	2.18519682e+006
	11261	635	11886	2185196.69882955	
	6824	648	7472	2185196.69885615	
e226	215	318	533	−18.7519290765415	−1.87519291e+001
	5663	567	6230	–	
	395	364	759	–	
etamacro	191	257	448	−755.715233369051	−7.55715223e+002
	185	176	361	−755.715233352295	
	162	215	377	−755.715233346024	
finnis	141	375	516	172791.065595611	1.72791066e+005
	276	308	584	172791.065595611	
	809	225	1034	172791.03306592	
fit1d	1	834	835	−9146.3780989634	−9.14637809e+003
	1	834	835	−9146.3780989634	
	1	834	835	−9146.3780989634	
ganges	1	420	421	−109585.736129308	−1.09585736e+005
	1	420	421	−109585.736129308	
	1	420	421	−109585.736129308	
gfrd-pnc	229	305	534	6902235.99954881	6.90223600e+006
	240	337	577	6902235.99954882	
	126	311	437	6902235.99954882	
grow15	1	879	880	−106870942.285325	−1.06870941e+008
	1	879	880	−106870942.285325	
	1	879	880	−106870942.285325	
grow22	1	3569	3570	−160871482.230788	−1.60834336e+008
	1	3569	3570	−160871482.230788	
	1	3569	3570	−160871482.230788	
grow7	1	240	241	−47787811.8605706	−4.77878118e+007
	1	240	241	−47787811.8605706	
	1	240	241	−47787811.8605706	
israel	2	157	159	−896644.821863043	−8.96644817e+005
	2	157	159	−896644.821863043	
	2	157	159	−896644.821863043	
kb2	1	50	51	−1749.9001299062	−1.74990013e+003
	1	50	51	−1749.9001299062	
	1	50	51	−1749.9001299062	
lotfi	76	128	204	−25.2647060618762	−2.52647061e+001
	339	158	397	−25.2647060618632	
	111	137	248	−25.2647060618773	

Name	Bf.	Sim.	Bf.+Sim.	Objective value	PCx
recipe	9	30	39	−266.616	−2.66616000e+002
	8	29	37	−266.616	
	9	28	37	−266.616	
sc105	1	56	57	−52.2020612117073	−5.22020612e+001
	1	56	57	−52.2020612117073	
	1	56	57	−52.2020612117073	
sc205	1	135	136	−52.2020612117073	−5.22020612e+001
	1	135	136	−52.2020612117073	
	1	135	136	−52.2020612117073	
sc50a	1	26	27	−64.5750770585645	−6.45750771e+001
	1	26	27	−64.5750770585645	
	1	26	27	−64.5750770585645	
sc50b	1	29	28	−70	−7.00000000e+001
	1	29	28	−70	
	1	29	28	−70	
scagr7	81	41	122	−2331389.82433099	−2.33138982e+006
	90	35	125	−2331389.82433097	
	69	26	95	−2331389.82433098	
scfxm1	1133	97	1220	18417.3255500362	1.84167590e+004
	2478	200	2878	–	
	311	123	434	18416.7590283489	
scorpion	90	38	128	1878.12482273811	1.87812482e+003
	114	37	151	1878.12482273811	
	70	70	140	1878.12482273811	
sctap1	320	8	328	1412.25	1.41225000e+003
	496	57	553	1412.24999999998	
	131	137	268	1412.25	
sctap2	739	195	934	1724.80714285713	1.72480714e+003
	739	195	934	1724.80714285713	
	739	195	934	1724.80714285713	
sctap3	469	252	721	1424	1.42400000e+003
	618	247	865	1424	
	369	909	1278	1424	
seba	79	32	111	15711.6000000006	1.57116000e+004
	90	40	130	15711.5999999923	
	–	–	–	–	
share1b	89	65	154	−76589.3185791853	−7.65893186e+004
	366	69	435	−76589.3224159041	
	368	63	431	−76589.3185791526	
share2b	135	38	173	−415.73224074142	−4.15732241e+002
	123	46	169	−415.732240741419	
	92	25	117	−415.732240741416	
shell	41	276	317	1208825346	1.20882535e+009
	55	279	334	1208825346	
	78	278	356	1208825346	
ship04l	8	251	259	1793324.53797036	1.79332454e+006
	450	124	574	1793324.53797036	

Name	Bf.	Sim.	Bf.+Sim.	Objective value	PCx
	100	374	474	1793324.53797035	
	14	172	186	1798714.70044539	
ship04s	116	185	301	1798714.70044539	1.79871471e+006
	57	194	251	1798714.70044539	
	320	530	850	1909055.21138913	
ship08l	461	364	825	1909055.21138913	1.90905521e+006
	144	631	775	1909055.21138913	
	54	258	312	1920098.21053462	
ship08s	169	239	408	1920098.21053462	1.92009821e+006
	67	272	339	1920098.21053462	
	49	1019	1068	1470187.91932926	
ship12l	938	1908	2846	1470187.91932926	1.47018797e+006
	232	1711	1943	1470187.91932926	
	55	439	494	1489236.13440613	
ship12s	429	556	985	1489236.13440613	1.48923613e+006
	166	486	632	1489236.13440613	
	75	326	401	15394362.1836319	
sierra	65	325	490	15381546.3836319	1.53943622e+007
	82	310	392	15394362.1836319	
	2450	44	2494	−251.26695098074	
stair	686	33	719	−251.266951192317	−2.51266951e+002
	11066	168	11234	−	
	21	138	159	1257.6995	
standata	76	98	174	1257.6995	1.25769951e+003
	146	116	262	1257.69949999999	
	131	109	240	1406.0175	
standmps	260	155	415	1406.01749999996	1.40601750e+003
	752	72	824	1406.0175	
	1	17	18	−41131.9762194364	
stocfor1	1	17	18	−41131.9762194364	−4.11319762e+004
	1	17	18	−41131.9762194364	
	69	55	124	129831.462637412	
vtp.base	179	71	250	129831.462461362	1.29831463e+005
	430	47	477	129831.464051472	

Tabela 3.5.1.

Analizirajući tabelu, možemo primetiti da su na skoro svim primerima modifikacije dale bolje rezultate od klasičnog simpleksa. Čak i ako se posmatra ukupan broj iteracija (kolona Sum.), ponovo su modifikacije bolje. Primetimo takođe da su modifikacija i poboljšana modifikacija skoro izjednačene. Upravo ova činjenica opravdava razmatranje originalne verzije modifikacije iz rada [78] u ovom radu, a i takođe dodatno zakomplikuje situaciju oko izbora algoritma za određivanje početnog bazično dopustivog rešenja. Međutim, ova originalna verzija je na 5 primera dala pogrešno rešenje. Klasični algoritam je na 3 primera dao pogrešan rezultat, dok je poboljšana modifikacija uspešno rešila sve primere iz ove klase. Na osnovu svega rečenog, možemo zaključiti da su se obe modifikacije dobro pokazale

u praksi, kao i da odgovor na pitanje koja je od njih bolja u mnogome zavisi od same strukture i prirode ulaznih podataka.

Revidirani simpleks metod (algoritmi **RevBasicMax** i **RevNoBasicMax**) kao i modifikacija (algoritam **ModRevNoBasicMax**) implementirani su u programskom jeziku MATHEMATICA. Tako je nastao program RevMarPlex, varijanta MarPlex-a koja koristi revidirani simpleks metod. Za razliku od MarPlex-a koji je implementiran u proceduralnom programskom jeziku, kod RevMarPlex-a smo se odlučili za paket MATHEMATICA prvenstveno zbog integrisanog solvera za sisteme linearnih jednačina (funkcija `LinearSolve`, [90]). Već smo napomenuli da je za implementaciju revidiranog simpleksa najpre neophodan brz i robustan (otporan na numeričke greške) solver za sisteme linearnih jednačina. Pored toga, kod RevMarPlex-a koristili smo prednosti MATHEMATICA-e po pitanju ulaznih i izlaznih podataka (ovde se funkcija cilja i ograničenja zadaju u svom prirodnom obliku).

Glavni nedostatak programa RevMarPlex je brzina rada. Dok MarPlex skoro sve probleme rešava u trenutku, RevMarPlex-u je često trebalo dosta vremena da reši problem. Da bi ovo objasnili, prvo moramo uzeti u obzir da se programi napisani u MATHEMATICA-i izvršavaju znatno sporije od odgovarajućih programa u proceduralnim jezicima. Međutim, to nije glavni razlog zašto je RevMarPlex znatno sporiji od MarPlex-a. Naime, prethodno smo već spomenuli da je iteracija revidiranog simpleks metoda algoritamski znatno kompleksnija od odgovarajuće iteracije simpleksa zato što je kod revidiranog simpleksa potrebno rešiti nekoliko sistema jednačina. Takođe, MarPlex koristi poboljšanje algoritma **Replace**, tj koristi sparse strukturu Tuckerove tabele dok kod RevMarPlex-a nemamo poboljšanje tog tipa.

U sledećoj tabeli su prikazani rezultati testiranja programa RevMarPlex na primerima manjih dimenzija

Problem	PCx	RevMarPlex	Mod.	Klas. alg.
Adlittle	2.25494963×10^{5}	225494.963162	32	39
Afiro	-4.64753143×10^{2}	-464.753142	2	17
Agg	3.59917673×10^{7}	-35991767.286576	151	151
Agg2	-2.0239251×10^{7}	-20239252.3559776	75	129
Blend	-3.08121498×10^{1}	-30.812150	-	-
Sc105	$-5.2202061212\times10^{1}$	-52.202061	-	-
Sc205	-5.22020612×10^{1}	-52.202061	-	-
Sc50a	$-6.4575077059\times10^{1}$	-64.575077	-	-
Sc50b	-7.000000000×10^{1}	-70	-	-
Scagr25	-1.47534331×10^{7}	-14753433.060769	520	> 1500
Scagr7	-2.33138982×10^{6}	-2331389.824330	55	74
Stocfor1	$-4.1131976219\times10^{4}$	-41131.976219	14	17
LitVera	1.999992×10^{-2}	0	-	-
Kb2	-1.7499×10^{3}	-1749.9001299062	-	-
Recipe	-2.66616×10^{2}	-266.6160	13	15
Share1B	-7.65893186×10^{2}	76589.3185791859	498	> 1500

Tabela 3.5.2.

Kao što možemo videti i u ovom slučaju se modifikacija pokazala bolja od klasičnog algoritma.

Oba programa smo testirali na još jednoj klasi ekstemno loše uslovljenih primera *KBAPAH* preuzetih iz [4], [42]. Ovi primeri se mogu naći i na internet stranici `www.psmtmath.s5.com`. Program PCx nije uspeo da reši ove primere. U sledećoj tabeli su prikazane optimalne vrednosti funkcije cilja izračunate pomoću MarPlex-a i RevMarPlex-a. Primeri su konstruisani tako da su optimalne vrednosti funkcija cilja jednaki 0.

Problem	RevMarPlex	MarPlex
07-20-02	0	0
15-30-03	7.2345×10^{-9}	2.16968×10^{-5}
15-30-04	2.16968×10^{-5}	1.3984×10^{-3}
15-30-06	4.90359×10^{-6}	1.671×10^{-3}
15-30-07	3.2807×10^{-4}	1.749×10^{-3}
15-60-07	0	-6.29305×10^{-7}
15-60-09	-6.29305×10^{-7}	-1.8353×10^{-6}
20-40-05	0	1.63948×10^{-6}
30-60-05	0	1.64567×10^{-11}
30-60-08	0	5.22527×10^{-5}
LitVera	0	0

Tabela 3.5.3.

Primećujemo da je u svim primerima RevMarPlex postigao bolje rezultate od MarPlex-a. Napominjemo da se kondicioni brojevi ($k = \|A\|\|A^{-1}\|$) kod ovih primera kreću u intervalu $(10^{15}, 10^{20})$.

3.6 Postoptimalna analiza

Često se javlja potreba da treba rešiti više problema linearnog programiranja koji su blisko povezani. Postoji više razloga zbog kojih se dolazi do takvih problema. Na primer, podaci koji definišu problem mogu biti nesigurni ili dati sa približnim vrednostima. Tada je poželjno razmotriti različite mogućnosti za polazne podatke. I u slučaju kada su polazni podaci dati sa tačnim vrednostima, može se desiti da se ti podaci menjaju svakodnevno, tako da se problem mora nanovo rešavati. U oba slučaja situacija je slična. Dakle, pitanje je da li prethodno dobijeno optimalno rešenje možemo iskoristiti da bi smo brže dobili rešenje novog sličnog problema. Naravno, odgovor je potvrdan u većini slučajeva.

Razmotrimo problem zadat u matričnom obliku

$$\begin{aligned} \max \quad & c^T x, \\ \text{p.o.} \quad & Ax = b, \\ & x \geq 0. \end{aligned}$$

Odgovarajuću simpleks tabelu koristeći ranije uvedene oznake možemo zapisati u obliku

$$d = c_B^T A_B^{-1} b - (c_B^T A_B^{-1} A_N - c_N^T) x_N,$$
$$x_B = A_B^{-1} b - A_B^{-1} A_N x_N,$$

ili korišćenjem oznaka

$$d' = c_B^T A_B^{-1} b, \quad (c'_N)^T = c_B^T A_B^{-1} A_N - c_N^T, \quad b' = A_B^{-1} b,$$

u obliku

$$d = d' - (c'_N)^T x_N, \tag{3.6.1}$$
$$x_B = b' - A_B^{-1} A_N x_N.$$

Neka je rešenje dato simpleks tabelom 3.6.1 optimalno. Pretpostavimo da se koeficijenti ciljne funkcije menjaju sa c na c'. Primetimo da se pri tome b' ne menja, tj. novo rešenje je takođe bazično dopustivo, dakle može se upotrebiti kao početno rešenje. Ako je razlika između c i c' mala, možemo očekivati da se novo optimalno rešenje dobije u relativno malom broju koraka.

Pretpostavimo sada da se c ne menja ali da se menja vektor b. U tom slučaju se $(c'_N)^T$ ne menja. Dakle, nova tabela je dualno dopustiva i primenom dualnog simpleks metoda dobijamo novo optimalno rešenje u malom broju koraka.

Dakle, promena vektora c ili b ne utiče na dopustivost optimalnog rešenja (3.6.1). Razmotrimo sada slučaj kada se menjaju vrednosti svih polaznih podataka. U tom slučaju se menjaju d', $(c'_N)^T$, b' ali se mogu promeniti i vrednosti matrica A_B i A_N. Ako je nova bazična matrica A_B nesingularna, možemo i dalje zadržati staru podelu na bazične i nebazične promenljive. Nova simpleks tabela najčešće neće biti ni primalno ni dualno dopustiva, ali ako je perturbacija polaznih podataka relativno mala, možemo očekivati da ta polazna tabela dovede do optimalnog rešenja u manjem broju iteracija u odnosu na klasično rešavanje. Iako ne postoji dokaz da bilo koji od tih takozvanih vrućih startova smanjuje broj potrebnih iteracija, iz empirijskih podataka je očigledno da ova procedura često dovodi do značajnog poboljšanja. Ponekad se vrućim startom problem reši i sto puta brže u odnosu na originalni simpleks metod.

Jedan od čestih problema postoptimalne analize se sastoji u sledećem: u kojim granicama se mogu menjati koeficijenti ciljne funkcije a da se pri tome ne naruši podela na bazične i nebazične promenljive. Pretpostavimo da se vektor c menja za $t\Delta c$, gde je t realan broj i Δc dat vektor (pri čemu je najčešće samo jedna koordinata različita od nule, ali diskusija važi i opštem slučaju). Tada se c'_N povećava za $t\Delta c'_N$, gde je

$$\Delta c'_N = \Delta c_B^T A_B^{-1} A_N - \Delta c_N^T.$$

Dakle, bazične promenljive se ne menjaju sve dok je

$$c'_N + t\Delta c'_N \geq 0.$$

Za $t > 0$ nejednakost važi ako je

$$t \leq \left(\max_{j \in N} -\frac{\Delta c'_j}{c'_j} \right)^{-1}.$$

Analogno, za $t < 0$, donja granica je

$$t \geq \left(\min_{j \in N} -\frac{\Delta c'_j}{c'_j} \right)^{-1}.$$

Iz ove dve nejednakosti dobijamo interval kome mora da pripada t

$$\left(\min_{j \in N} -\frac{\Delta c'_j}{c'_j} \right)^{-1} \leq t \leq \left(\max_{j \in N} -\frac{\Delta c'_j}{c'_j} \right)^{-1}.$$

Primer 3.6.1 [82] Razmotrimo problem linearnog programiranja

$$\begin{array}{rrcl} \max & 5x_1 + 4x_2 + x_3 & & \\ \text{p.o.} & 2x_1 + 3x_2 + x_3 & \leq & 5 \\ & 4x_1 + x_2 + 2x_3 & \leq & 11 \\ & 3x_1 + 4x_2 + 2x_3 & \leq & 8 \\ & x_1, x_2, x_3 & \geq & 0. \end{array}$$

Optimalna tabela za ovaj problem je

$$\begin{aligned} d &= 13 - 3x_2 - x_4 - x_6 \\ x_3 &= 1 + x_2 + 3x_4 - 2x_6 \\ x_1 &= 2 - 2x_2 - 2x_4 + x_6 \\ x_5 &= 1 + 5x_2 + 2x_4. \end{aligned}$$

Optimalna baza je $\{3, 1, 5\}$. Ispitajmo u kojim granicama se može promeniti koeficijent ciljne funkcije uz x_1 a da se optimalna baza ne promeni. Kako je $c = (5, 4, 3, 0, 0, 0)$, stavićemo $\Delta c = (1, 0, 0, 0, 0, 0)$. Sada je

$$\Delta c_B = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{i} \quad \Delta c_N = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Kako je

$$\Delta c'_N = \Delta c_B^T A_B^{-1} A_N - \Delta c_N^T,$$

i iz optimalne tabele je

$$A_B^{-1} A_N = \begin{bmatrix} -1 & -3 & 2 \\ 2 & 2 & -1 \\ -5 & -2 & 0 \end{bmatrix},$$

to je

$$\Delta c'_N = (2, 2, -1).$$

Dakle, granice za t određujemo iz nejednačina

$$3 + 2t \geq 0, \quad 1 + 2t \geq 0, \quad \text{i} \quad 1 - t \geq 0.$$

Odatle je $-1/2 \leq t \leq 1$, pa se koeficijent ciljne funkcije koji stoji uz x_1 može kretati u granicama od $4,5$ do 6.

Pretpostavimo sada da se menjaju vrednosti koordinata vektora b, tj. da se b menja za $t\Delta b$. Sada se c'_N ne menja ali se b' promeni za $t\Delta x_B$ gde je

$$\Delta x_B = A_B^{-1}\Delta b.$$

Dakle, baza ostaje optimalna sve dok t zadovoljava nejednačinu

$$\left(\min_{i\in B} -\frac{\Delta x_i}{b'_i}\right)^{-1} \leq t \leq \left(\max_{i\in B} -\frac{\Delta x_i}{b'_i}\right)^{-1}.$$

4 Tri direktna metoda u linearnom programiranju

Osim klasičnog simpleks metoda postoje i alternativni metodi za rešavanje problema linearnog programiranja koji se zasnivaju ili na geometrijskim osobinama skupa ograničenja [14, 20] ili na uopštenim inverzima [67, 11, 64, 63, 65]. Pomenimo i radove koji se odnose na Banachove prostore [43, 44]. Rezultate sličnog tipa opisujemo u ovom poglavlju i uvodimo tri direktna metoda za rešavanje problema linearnog programiranja koji zadovoljavaju izvesne uslove. Prvi metod pronalazi početnu tačku koja predstavlja ili ekstremnu tačku ili tačku koja je susedna ekstremnoj tački. Drugi metod se zasniva na teoriji igara a treći na uopštenim inverzima. U slučaju da uslovi za direktno nalaženje ekstremne tačke nisu zadovoljeni, ovi metodi se mogu primeniti za konstrukciju početne tačke za klasičan simpleks metod. Efikasnost poboljšanog simpleks metoda, sa početnom tačkom dobijenom primenom novih metoda, upoređena je sa originalnim simpleks metodom kao i sa metodima unutrašnje tačke, što je ilustrovano sa nekoliko karakterističnih primera. Osim toga, razmatra se i eliminacija suvišnih ograničenja u problemu linearnog programiranja.

Do kraja ovog odeljka sledimo radove [75, 76, 77] i [67].

4.1 Osnovni pojmovi

Razmotrimo problem linearnog programiranja u kome su ograničenja nejednakosti. Odrediti maksimum linearne ciljne funkcije

$$z(x) = \sum_{j=1}^{n} c_j x_j = cx \tag{4.1.1}$$

u odnosu na linearna ograničenja

$$\begin{aligned} \sum_{j=1}^{n} a_{ij}x_j &= r_i x \leq b_i, \quad i = 1,\ldots,m \\ x_j &\geq 0,\ j = 1,\ldots,n \end{aligned} \tag{4.1.2}$$

gde je $r_i x$, $i=1,\ldots,m$ skalarni proizvod vektora r_i i x, i

$$c=(c_1,\ldots,c_n),\ x=(x_1,\ldots,x_n),\ r_i=(a_{i1},\ldots,a_{in}),\quad i=1,\ldots,m.$$

Razmatramo sledeći problem: generisati pogodnu početnu tačku za simpleks metod, sa ciljem da se redukuje potreban broj iterativnih koraka za pronalaženje optimalnog rešenja. Dobijen je metod koji je primenljiv na probleme linearnog programiranja koji su definisani bez suvišnih uslova. Pokazuje se da je isti metod ponekad primenljiv i na probleme linearnog programiranja kod kojih postoje suvišni uslovi.

U nastavku uvodimo metod (takozvani *metod minimalnih uglova*), za ubrzanje simpleks metoda kod nekih klasa problema linearnog programiranja. Metod u najgorem slučaju daje pogodnu tačku za start standardnog simpleks algoritma. Ta tačka je rešenje linearnog sistema koji se dobija posle zamene $l \leq n$ izabrnih ograničenja sa odgovarajućim jednačinama pri čemu su $n - l$ promenljivih izjednačene sa nulom. Za neke probleme linearnog programiranja simpleks metod samo potvrđuje da je izabrana tačka optimalno rešenje i algoritam se završava u samo jednom koraku. U najgorem slučaju, metod minimalnih uglova daje teme konveksnog skupa koje je u susedstvu sa ekstremnom tačkom. Na taj način metod minimalnih uglova obezbeđuje u izvesnim slučajevima značajnu redukciju broja iterativnih koraka u odnosu na klasičan simpleks metod. Osim toga, pored ubrzanja konvergencija, za određene tipove problema se smanjuje i dimenzija problema.

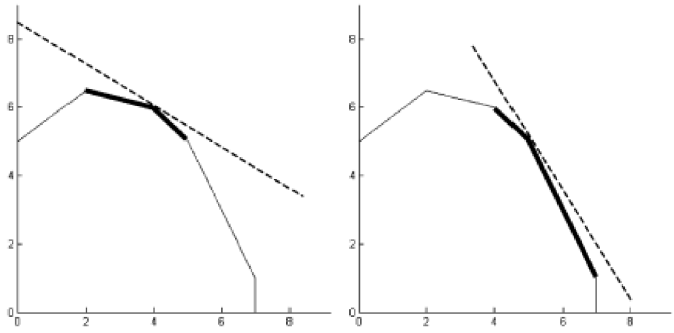
Osnovni preduslov za uspešnu primenu metoda minimalnih uglova je da je dati problem bez suvišnih ograničenja. Iz tih razloga, razmatramo i problem eliminacije suvišnih ograničenja. Takođe uvodi se jedan direktan metod koji se zasniva na teoriji igara. Kod nekih problema linearnog programiranja rešenje se može dobiti neposredno, dok se kod nekih problema njegovom primenom eliminišu suvišna ograničenja.

U četvrtom odeljku su razvijeni odgovarajući algoritmi za uvedene metode. Takođe su opisani najvažniji detalji implementacije metoda minimalnih uglova u programskom paketu MATHEMATICA.

Efikasnost predstavljenih metoda je pokazana na nekoliko ilustrativnih primera. Takođe je na tim primerima izvršeno upoređivanje sa standardnom simpleks metodom kao i sa metodima unutrašnje tačke.

4.2 Metod minimalnih uglova

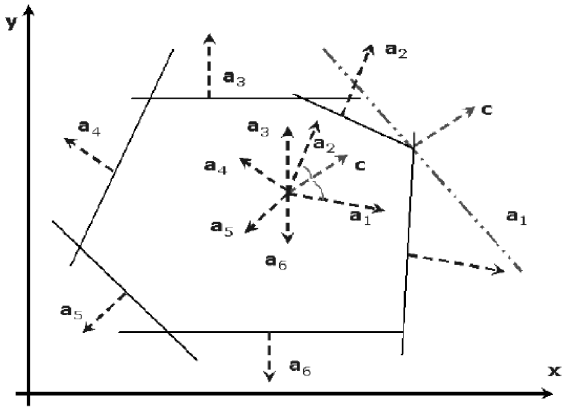
Glavna ideja u radu [76] jeste da se poboljša izbor inicijalne baza. Poznato je da se optimalno teme formira kao presek n ograničenja, gde je n broj promenljivih u LP. Ovakvih n ograničenja koja formiraju optimalno teme trebalo bi da zahvataju najmanje uglove sa ciljnom funkcijom. Ovakvo tvrđenje je ilustrovano na Slici 4.2.1, na kojoj isprekidana linija predstavlja ciljnu funkciju a boldovane linije predstavljaju ograničenja koja su najbliža prema zahvaćenom uglu sa ciljnom funkcijom.



Slika 4.2.1. Ilustracija ideje minimalnih uglova.

Primer 4.2.1 Posmatra se linearni problem

$$\begin{array}{ll} \max & z = x + y \\ \text{p.o.} & 3x - y \leq 70 \\ & x + 4y \leq 100 \\ & y \leq 20 \\ & -10x + 3y \leq -20 \\ & -5x - 3y \leq -55 \\ & -y \leq 5 \end{array}$$



Slika 4.2.2. Ilustarcija metoda minimalnih uglova.

Jedinstveno rešenje ovog problema je $x^* = (29.23, 17.69)$. Prvo i drugo ograničenje u ovom problemu jesu jedina aktivna ograničenja u x^*. Na slici 4.2.2 se vidi da su uglovi između gradijenata za funkcije date prvim i drugim ograničenjem i gradijenta ciljne funkcije c minimalni između uglova svih ograničenja i c.

Za opis metoda minimalnih uglova potrebna je sledeća definicija.

Definicija 4.2.1 *Neka je $P \subseteq \mathbb{R}^n$ poliedar (konveksan skup ili konus) definisan bez suvišnih nejednakosti sa:*

$$P: \quad \sum_{j=1}^{n} a_{ij} x_j = r_i x \leq b_i, \quad i = 1, \ldots, m.$$

Tangencijalni poliedar P^0 poliedra P je definisan sledećim skupom nejednakosti

$$P^0: \quad \sum_{j=1}^{n} a_{ij}x_j = r_i x \le |r_i|, \quad |r_i| = \sqrt{a_{i1}^2 + \cdots + a_{in}^2},\ i = 1, \ldots, m.$$

Metod predstavljen u sledećoj teoremi je direktan i za pogodno definisane probleme linearnog programiranja (4.1.1)-(4.1.2) daje optimalnu tačku u samo jednom koraku. Opisaćemo sada glavnu ideju tog metoda. U slučaju da je problem linearnog programiranja sa samo dve promenljive, možemo primeniti grafički postupak za rešavanje [66, 84]. Ako su ograničavajući uslovi dati nejednakostima, svaka od odgovarajućih pravih deli ravan na skup dopustivih i skup nedopustivih tačaka za date uslove. Dopustive tačke se nalaze u oblasti P koja zadovoljava sve date uslove. Optimalno rešenje je sada moguće odrediti crtanjem grafika modifikovane ciljne funkcije $z(x_1, x_2) = 0$ i njenim paralenim pomeranjem u smeru gradijentnog vektora (c_1, c_2). Optimalno rešenje je jedinstveno ako prava $z(x_1, x_2) = z_{max}$ prolazi kroz samo jedno teme (ekstremnu tačku) dopustive oblasti. U slučaju minimizacionog problema, datu pravu pomeramo paralelno u suprotnom smeru.

Poznato je da u n-dimenzionalnom slučaju teme dopustivog poliedra određujemo rešavanjem odgovarajućeg sistema sa n jednačina datih ograničavajućim uslovima. U sledećoj teoremi uvodimo kriterijum za pogodan izbor tih jednačina, koji se bazira na generalizaciji i formalizaciji grafičkog metoda. Time dobijamo glavni rezultat ovog odeljka.

Teorema 4.2.1 [*76*] *Neka je dat maksimizacioni problem linearnog programiranja* (4.1.1)-(4.1.2) *bez suvišnih nejednakosti. Neka je $P \subseteq \mathbb{R}^n$ poliedar definisan sa* (4.1.2). *Sa $c = (c_1, \ldots, c_n)$ označavamo gradijent vektor ciljne funkcije. Vektori r_i su definisani sa $r_i = (a_{i1}, \ldots, a_{in})$, $i = 1, \ldots, m$. Razmotrimo skup*

$$V = \left\{ v_i = \frac{cr_i}{|r_i|},\ |r_i| = \sqrt{a_{i1}^2 + \cdots + a_{in}^2}, \quad i = 1, \ldots, m \right\}. \tag{4.2.1}$$

Pretpostavimo da skup V sadrži l pozitivnih elemenata, označenih sa $v_{i_1}, \ldots v_{i_l}$.

Razlikujemo sledeće slučajeve:

(a) *Ako je P prazan skup, ne postoji rešenje datog problema.*

(b) *U slučaju da je $l = 0$, tada je $z_{\max} = +\infty$, gde $z_{\max}$ označava maksimalnu vrednost ciljne funkcije $z(x)$.*

(c) *U slučaju da je $l \ge n$, neka je x_0 rešenje sledećeg sistema jednačina:*

$$\begin{aligned} &a_{i_1,1}x_1 + \cdots + a_{i_1,n}x_n = r_{i_1}x = b_{i_1} \\ &\cdots \qquad\qquad \cdots \\ &a_{i_n,1}x_1 + \cdots + a_{i_n,n}x_n = r_{i_n}x = b_{i_n}, \end{aligned} \tag{4.2.2}$$

pri čemu indeksi $i_1, \ldots, i_n$ odgovaraju skupu od n maksimalnih i pozitivnih vrednosti izabranih iz skupa V. Ako optimalnu tačku u P označimo sa x_P, mogu da nastupe sledeći slučajevi:

(i) $x_0 = x_P$, *ili*

(ii) x_0 *i* x_P *pripadaju istoj graničnoj hiperravni poliedra* P.

(d) *U slučaju da je* $0 < l < n$, *razmotrimo sledeći sistem*

$$\begin{aligned} &a_{i_1,1}x_1 + \cdots + a_{i_1,n}x_n = r_{i_1}x = b_{i_1} \\ &\cdots \qquad\qquad \cdots \\ &a_{i_l,1}x_1 + \cdots + a_{i_l,n}x_n = r_{i_l}x = b_{i_l}, \end{aligned} \tag{4.2.3}$$

gde indeksi $i_1, \ldots, i_l$ *odgovaraju pozitivnim vrednostima* $v_{i_1}, \ldots, v_{i_l}$ *iz skupa* V. *Bazično rešenje* x_0 *problema* (4.1.1)-(4.1.2) *dobija se izjednačavanjem* $n - l$ *promenljivih sa nulom i rešavanjem* l *jednačina iz* (4.2.3) *Tada* x_0 *i* x_P *pripadaju istoj hiperravni poliedra* P.

Primer 4.2.2 Da bismo ilustrovali geometrijski smisao Teoreme 4.2.1, razmotrimo sledeći problem:

$$\begin{aligned} \max\ & z(x_1, x_2) = x_1 + x_2 \\ \text{p.o.}\ & (C_1) \qquad x_1/3 + x_2 \le 1 \\ & (C_2) \qquad x_1 + x_2 \le 2 \\ & (C_3) \qquad x_1 + x_2/3 \le 1. \end{aligned}$$

Minimalan ugao je zahvaćen između gradijentnog vektora $c = (1, 1)$ ciljne funkcie i prave $H_2 \equiv x_1 + x_2 = 2$. Uglovi između vektora c i pravih $H_1 \equiv x_1/3 + x_2 = 1$ i $H_3 \equiv x_1 + x_2/3 = 1$ su identični. Ako uzmemo dva ugla između vektora c i hiperravni H_2 i H_3, tada rešenje $x_0 = H_2 \cap H_3$ zadovoljava $x_0 = (1/2, 3/2) \notin P$. Slično, ako uzmemo hiperravni H_2 i H_1, dobijamo $x_0 = H_2 \cap H_1 = (3/2, 1/2) \notin P$. Optimalno rešenje x_P je u temenu $x_P = (3/4, 3/4)$. Međutim, x_0 i x_P su zajednički elementi na pravoj H_2. Uslov (C_2) je suvišan. Ako se uslov (C_2) eliminiše, dobijamo $x_0 = x_P = (3/4, 3/4)$.

Napomena 4.2.1 Ako je poliedar P definisan bez suvišnih ograničenja, tada Teorema 4.2.1 može biti uspešno primenjena. Ako sistem nejednačina (4.1.2) sadrži nekoliko suvišnih ograničenja, moguće je da su neki od minimalnih uglova u (4.2.1) određeni suvišnim ograničenjima. U tom slučaju može da se desi da $x_0 \notin P$, tako da tačke x_0 i x_P nisu elementi iste hiperravni od P. Naravno, i u tom slučaju tačka x_0 može biti iskorišćena kao početna tačka za simpleks metod, ali ubrzanje konvergencije nije garantovano u tom slučaju. Ustvari, za uspešnu primenu Teoreme 4.2.1, dovoljno je da nijedan od l minimalnih uglova nije zahvaćen sa nekim suvišnim ograničenjem. To daje motivaciju za eliminaciju suvišnih ograničenja, što se razmatra u narednom odeljku.

Napomena 4.2.2 Metod koji koristimo u slučaju $0 < l < n$ je takođe moguće primeniti i u slučaju $m < n$. U tom slučaju, predloženi metod je modifikacija pozantog metoda za konstrukciju bazičnog rešenja iz [10]. Primetimo da je u [10] bazično rešenje problema (4.1.1)-(4.1.2) za koji važi $m < n$, dobijeno izjednačavanjem $n - m$ promenljivih sa nulom i rešavanjem m jednačina.

Napomena 4.2.3 Metod minimalnih uglova daje bazično rešenje koje pripada istoj hiperravni kao i ekstremna tačka. Osim ove značajne osobine, naglasimo sledeće

značajno svojstvo ovog metoda. U simpleks metodu sva ograničenja se koriste u svakom koraku kao i dodate izravnavajuće promenljive. Kod metoda minimalnih uglova broj aktivnih ograničenja je manji u odnosu na standardni simpleks algoritam. Još više, kod metoda minimalnih uglova, izravnavajuće promenljive se ne koriste. Prema tome, dimenzija problema, razmatrana kod metoda minimalnih uglova, je značajno manja u odnosu na dimenziju odgovarajućeg problema kod primene simpleks metoda. Dakle, zamena nekoliko iteracija simpleks metoda sa samo jednom primenom metoda minimalnih uglova, obično značajno smanjuje broj potrebnih operacija i procesorsko vreme.

Napomena 4.2.4 U radu [88] dat je kontraprimer gde Teorema 4.2.1 ne važi. Kasnije je u svojoj magistarkoj tezi Wang [85] pokazao da primer iz [88] nije korektan jer sadrži suvišna ograničenja i zatim dao koretan kontraprimer. I pored toga, na osnovu radova [53, 54, 55, 56], Teorema 4.2.1 može da se koristi kao uspešna heuristika za pronalaženje početne tačke simpleks algoritma ili, u pogodnim uslovima, optimalne tačke linearnog problema. Pomenimo još i to da je rad [75] osim u [88, 85] citiran i u radovima [2, 3, 13, 15, 16, 26, 27, 30, 32, 45, 57, 68, 86].

4.3 Suvišna ograničenja i primena teorije igara

U početku ovog odeljka uvodimo nekoliko pravila za eliminaciju suvišnih ograničenja. Ta pravila su korisna i u slučaju transformacije problema (4.1.1)-(4.1.2), na oblik pogodan za primenu simpleks metoda. Ovaj odeljak je baziran na radovima [75] i [76].

Poznato je da Gausova eliminacija ili QR faktorizacija može biti primenjena za eliminaciju suvičnih ograničenja [18, 84]. Međutim, pri tome se javljaju sledeće poteškoće.

A. Pre primene Gausove eleiminacije, nejednakosti moraju biti transformisane u odgovarajuće jednakosti. Na taj način se dimenzija sistema na koji se primenjuje Gausova eliminacija obično značajno povećava.

B. Još više, greške zaokruživanja i potreban broj aritmetičkih operacija su značajni u mnogim slučajevima.

C. Postupak Gausove eliminacije prepoznaje samo linearno zavisna ograničenja tipa jednakosti. Primena simpleks algoritma obično zahteva uvođenje dopunskih promenljivih. U tom ekvivalentnom obliku, matrica ograničenja A je obično potpunog ranga, tako da problem nepotpunog ranga nije od velikog značaja u praksi [93].

Implementacija metoda unutrašnje tačke za probleme linearnog programiranja velikih dimenzija obično sadrži pripremnu fazu za eliminaciju suvišnih promenljivih i suvišnih ograničenja [1]. Na primer, pripremna faza programa PCx izvršava nekoliko tipova eliminacija suvišnih promenljivih i suvišnih ograničenja [19].

Pripremna faza u programu PCx radi sa problemima linearnog programiranja oblika

$$\min_{x\in\mathbb{R}^n} c^T x \quad \text{u odnosu na} \quad Ax = b,$$
$$\begin{cases} 0 \le x_i, & i \in \mathcal{N} \\ 0 \le x_i \le u_i, & i \in \mathcal{U} \\ x_i \text{ slobodno}, & i \in \mathcal{F}, \end{cases}$$

gde je $\mathcal{N}\cup\mathcal{U}\cup\mathcal{F}$ particija indeksnog skupa $\{1,\ldots,n\}$ [19]. Pripremna faza proverava ulazne podatke u odnosu na sledeća eliminaciona pravila [19].

Prazne vrste. Ako matrica A ima nula vrstu i odgovarajuću nula koordinatu u vektoru b, ta vrsta se može izostaviti iz razmatranja.

Duplirane vrste. Kada je vrsta matrice A (i odgovarajući element iz vektora ograničenja b) proporcionalna nekoj drugoj vrsti, može biti izostavljena.

Duplirane kolone. Kada je kolona matrice A proporcinalna nekoj drugoj koloni, te dve kolone je moguće objediniti. Ulazne promenljive objedinjenih kolona su ili normalne ili slobodne, što zavisi od toga da li je faktor proporcionalnosti kolona pozitivan ili negativan.

Fiksirane promenljive. Ako promenljiva ima nulu za gornje i donje ograničenje, očigledno je da je moguće tu promenljivu izjednačiti sa nulom i izostaviti iz problema.

Jednoelementne vrste. Ako i-ta vrsta matrice A sadrži samo jedan element A_{ij} različit od nule, jasno je da je $x_j = b_i/A_{ij}$, tako da ta promenljiva može biti eliminisana iz problema. Takođe se i-ta vrsta matrice A može izostaviti.

Jednoelementne kolone. Kada je A_{ij} jedini element različit od nule u koloni j matrice A, i x_j je slobodna promenljiva, tada je promenljivu x_j moguće izraziti preko drugih promenljivih koje se javljaju u i-toj vrsti matrice A i eliminisati je iz problema. Čak i ako promenljiva nije slobodna, x_j može biti eliminisana ako su njena ograničenja slabija od ograničenja koja slede iz domena ostalih elemenata koji figurišu u vrsti A_i. Slična tehnika je upotrebljena u [1]. Međutim, neka od suvišnih ograničenja nisu obuhvaćenja razmatranjima opisanim u [1] i [19]. Kako je napomenuto u [1], u slučaju problema linearnog programiranja velikih dimenzija nije moguće otkloniti sve suvišne nejednakosti manuelno. Dakle, analiza pripremne faze ima za cilj poboljšanja formulacije problema.

U tom cilju, u ovom odeljku predlažemo nekoliko dodatnih pravila za eliminaciju suvišnih ograničenja. Prvo pravilo se bazira na skoro očiglenim geometrijskim osobinama nekih suvišnih ograničenja.

Razmotrimo sledeća ograničenja u odnosu na (4.1.2):

$$\begin{aligned} \frac{1}{b_i} r_i x = \sum e_{ij} x_j &\le 1, \quad i = 1,\ldots,m \\ x_j &\ge 0, \quad j = 1,\ldots,n. \end{aligned} \tag{4.3.1}$$

gde je $e_{ij} = \dfrac{a_{ij}}{b_i},\ b_i \neq 0,\ i = 1, \ldots, m,\ j = 1, \ldots n.$ Ako je uslov

$$(\exists p, q)(1 \leq p, q \leq m)\ e_{ps} \geq e_{qs}, \quad s = 1, \ldots, n, \tag{4.3.2}$$

zadovoljen, tada q-to ograničenje iz (4.3.1) (respektivno (4.1.2)) može biti izostavljeno.

Međutim, ovo pravilo eliminacije nije univerzalno. Na primer, suvišno ograničenje (C_2) iz problema problema (4.2.2) ne zadovoljava uslov (4.3.2).

U sledećoj teoremi je primenjen poznat rezultat iz teorije igara, i definisano je nekoliko dodatnih pravila za eliminaciju, kao i direktan metod za rešavanje nekih tipova problema linearnog programiranja.

Teorema 4.3.1 [76] *Razmotrimo sledeći problem linearnog programiranja: odrediti maksimum ciljne funkcije*

$$z(x) = \sum_{j=1}^{n} c_j x_j = cx, \quad c_j > 0,\ j = 1, \ldots, n$$

u odnosu na ograničenja

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_j = r_i x \quad &\leq \quad b_i,\ b_i > 0, \quad i = 1, \ldots, m, \\ x_j \quad &\geq \quad 0, \quad j = 1, \ldots, n. \end{aligned} \tag{4.3.3}$$

Neka je $d_{ij} = \dfrac{a_{ij}}{b_i c_j}, \quad i = 1, \ldots, m,\ j = 1, \ldots, n.$ *Tada važe sledeća tvrđenja:*

(a) *U slučaju*

$$\max_{1 \leq i \leq m} \min_{1 \leq j \leq n} d_{ij} = \min_{1 \leq j \leq n} \max_{1 \leq i \leq m} d_{ij} = d_{kl}, \tag{4.3.4}$$

optimalno rešešenje problema (4.1.1)-(4.1.2) *je*

$$x_j = \begin{cases} \dfrac{b_k}{a_{kl}}, & j = l, \\ 0, & j \neq l. \end{cases} \tag{4.3.5}$$

(b) *Ako je*

$$\max_{1 \leq i \leq m} \min_{1 \leq j \leq n} d_{ij} \neq \min_{1 \leq j \leq n} \max_{1 \leq i \leq m} d_{ij}$$

i ukoliko postoje k, l *takvi da važi* $d_{ik} \geq d_{il},\ i = 1, \ldots, m,$ *tada to povlači da je* $x_k = 0$*, tako da možemo izostaviti* k*-tu kolonu.*

Tkođe, ako postoje k, l *takvi da važi* $d_{kj} \leq d_{lj},\ j = 1, \ldots, n,$ *tada možemo izostaviti* k*-tu vrstu.*

Dokaz. Neka je $y_j = c_j x_j,\ j = 1, \ldots, n$. Tada je problem (4.3.3) ekvivalentan problemu $z(y) = \sum_{j=1}^{n} y_j \to max$ sa ograničenjima

$$\sum_{j=1}^{n} d_{ij} y_j \leq 1, \quad i = 1, \ldots, m, \quad y_j \geq 0,\ \ j = 1, \ldots, n.$$

Primetimo da je ovaj problem linearnog programiranja ekvivalentan sa igrom dva igrača sa matricom plaćanja $D = ||d_{ij}||$ [50, 76]. Označimo optimalnu strategiju za drugog igrača vektorom $q = (q_1, \ldots, q_n)$.

(a) Ako je uslov (4.3.4) ispunjen, tada je optimalna strategija za drugog igrača čista strategija $q_l = 1$, $q_j = 0$, $j \neq l$. Kako je $c_j x_j = y_j = \dfrac{q_j}{v}$, gde je v vrednost igre, dobijamo [50] $x_j = 0$, $j \neq l$ i

$$x_l = \max_{1 \leq i \leq m} \left\{ \frac{a_{il}}{b_i} \right\} = \min_{1 \leq i \leq m} \left\{ \frac{b_i}{a_{il}} \right\} = \frac{b_k}{a_{kl}}.$$

(b) Ako postoje k, l tako da je $d_{ik} \geq d_{il}, i = 1, \ldots, m$, tada iz teorije igara sledi $q_k = 0$, što povlači $x_k = 0$ [50]. Dakle, možemo izostaviti k-tu kolonu.

Ako postoje k, l tako da je $d_{kj} \leq d_{lj}, j = 1, \ldots, n$, tada

$$\sum_{j=1}^{n} d_{lj} \xi_j \leq 1 \Rightarrow \sum_{j=1}^{n} d_{kj} \xi_j \leq 1, \ \text{ za proizvoljne } \xi_j > 0,\ \ j = 1, \ldots, n$$

tako da je k-to ograničenje suvišno. □

Uopštenja Teoreme 4.3.1 se mogu naći u radu [85].

Napomena 4.3.1 Ako su sva suvišna ograničenja eliminisana, tada je uslov $x_0 \in P$ ispunjen. Nažalost, predloženi algoritmi za eliminaciju suvišnih ograničenja ne garantuju kompletnu eliminaciju. Na primer, suvišno ograničenje (C_2) u problemu (4.2.2) ostaje posle svih eliminacija. U tom slučaju predlažemo da se bazično rešenje konstruisano primenom metode minimalnih uglova iskoristi kao početna tačka za primenu simpleks algoritma.

4.4 Algoritmi i implementcioni detalji

U skladu sa Teoremom 4.2.1 uvodimo dva algoritma, označena sa *Algoritam An* i *Algoritam Al*, za implementaciju metoda minimalnih uglova. Ovi algoritmi se mogu upotrebiti za maksimizaciju ciljne funkcije (4.1.1) pod ograničenjima (4.1.2), i moguće ih je primeniti u slučaju $l \geq n$.

Algoritam An

Korak 1. Eliminisati suvišna ograničenja, koristeći rezultate iz prethodnog odeljka i Gausovu eliminaciju.

Korak 2. Izračunati vrednosti $v_i = |c|\cos(c, r_i) = \dfrac{cr_i}{|r_i|}, \quad i = 1, \ldots, m.$

Korak 3. Odrediti n maksimalnih i pozitivnih vrednosti

$$v_{i_1} \geq \cdots \geq v_{i_n} > 0$$

iz skupa $\{v_1, \cdots v_m\}$.

Korak 4. Izračunati x_0 kao rešenje sistema jednačina (4.2.2).

Korak 5. Proveriti da li je x_0 bazično dopustivo rešenje ($x_1 \geq 0, \ldots, x_n \geq 0$), zbog primene simpleks metoda u sledećem koraku.

Korak 6. Ako je uslov iz prethodnog koraka ispunjen, primeniti algoritam simpleks metoda *SimMax* iz odeljka 1.2 za bazično dopustivo rešenje [84]). U suprotnom, primeniti pripremni algoritam simpleks metoda *PreSim Max* iz odeljka 1.2, koji generiše prvo bazično rešenje i završiti algoritmom za bazično dopustivo rešenje [84].

Neki softverski detalji implementacije *Algoritma An* su opisani u nastavku. Unutrašnji oblik problema postavljenog u (4.1.1) sadrži dva različita dela. Prvi deo je proizvoljna ciljna funkcija, određena sa odgovarajućim izrazom iz paketa MATHEMATICA, označena parametrom *objective*. Drugi deo je lista izabranih ograničenja koju nazivamo *constraints*. Prema unutrašnjoj formi linearnih programa, koji se zahtevaju u funkcijama *LinearProgramming*, *ConstrainedMin* i *ConstrainedMax* iz paketa MATHEMATICA [90], [92], izostavljamo listu promenljivih koje se koriste u ciljnoj funkciji i datim ograničenjima. Listu promenljivih izostavljamo zbog jednostavnije upotrebe programa. Tu listu je moguće rekonstruisati koristeći standardne funkcije *Variables* i *Union* (o tim funkcijama videti [91] i [92]). U tu svrhu, definišemo sledeću funkciju koja izdvaja listu promenljivih upotrebljenih u listi *lis_*.

```
SelectVariables[lis_] :=
  Module[{duz, l={}},
      duz = Length[lis];
      Do[l=Union[l, Variables[lis[[i]]]],{i,duz}];
      Return[l]
  ]
```

U glavnom programu možemo da pišemo

```
var=SelectVariables[Append[objective, constraints]].
```

Implementacija Koraka 2 i Koraka 3.

A. Matrica datih ograničenja, čiji su elementi jednaki a_{ij}, $i = 1, \ldots, m$, $j = 1, \ldots, n$, može biti formirana na sledeći način.

```
Formmatrix[constr_, var_]:=
  Module[{a={}, i,j,m,n, p, c=constr},
     m=Length[c];  n=Length[var];
     For[i=1, i<= m, i++,
         p={};
         For[j=1, j<= n, j++,
             AppendTo[p,Coefficient[c[[i,1]], var[[j]]]];
```

```
        ];
        AppendTo[a,p];
    ];
    Return[a];
]
```

B. Izbor l maksimalnih vrednosti $v_{i_q} > 0$, $q = 1, \ldots, l$ iz skupa V, definisanmog sa (4.2.1), može biti izvršen primenom funkcije `maximaln[mat_, ncf_, constr_]`. Formalni parametar $mat\_$ ove funkcije označava matricu datih ograničenja, i može biti konstruisan primenom funkcije $Formmatrix$. Parametar $ncf\_$ označava vektor $\{c_1, \ldots, c_n\}$, i može biti konstruisan na sledeći način:

```
For[i=1, i<= n, i++,
    AppendTo[nc, Coefficient[objective,var[[i]]]];
];
```

Konačno, parametar $constr\_$ označava listu datih ograničenja.

```
maximaln[mat_, ncf_, constr_]:=
  Module[{i,j,m,n, mt=mat,c=ncf, sp={},csp=constr, p},
     {m,n}=Dimensions[mt];
     For[i=1, i<= m, i++,
       AppendTo[sp, mt[[i]].c/Sqrt[mt[[i]].mt[[i]]] ]
     ];
     For[i=1, i<=m-1, i++,
         For[j=i+1, j<=m, j++,
            If[sp[[i]]<sp[[j]],
               p=sp[[i]]; sp[[i]]=sp[[j]]; sp[[j]]=p;
               p=csp[[i]]; csp[[i]]=csp[[j]]; csp[[j]]=p;
         ]  ]
     ];
     For[i=1; l=0, i<= m, i++,
        If[sp[[i]]>0, l++]
     ];
     Return[{Take[csp,l], Drop[csp,l]} ];
  ]
```

Implementacija Koraka 4 i Koraka 5.

A. Za svako $i = 1, \ldots, m$ transformišemo dato ograničenje oblika

$$\sum_{k=1}^{n} a_{ik}x_k + r_i \quad \begin{matrix} \leq \\ \geq \\ = \end{matrix} \quad \sum_{k=1}^{n} b_{ik}x_k + s_i$$

u oblik

$$\left\{ \sum_{k=1}^{n} a_{ik}x_k + r_i - \left(\sum_{k=1}^{n} b_{ik}x_k + s_i \right), \quad \begin{matrix} \leq \\ \geq \\ = \end{matrix} \right\}. \tag{4.4.1}$$

Nule na desnoj strani formule (4.4.1) se podrazumevaju. U tu svrhu, definišemo sledeću funkciju:

```
Transform[lis_] :=
  Module[{l={}, h, p},
     Do[h=Head[lis[[i]]];   p=lis[[i]]/.h->Subtract;  AppendTo[l, List[p,h]],
        {i,Length[lis]}
     ];
     Return[l]
  ]
```

Primenom izraza `lista =Transform[constraints]` lista datih ograničenja je transformisana u listu čiji elementi su oblika (4.4.1).

B. *Implementacija Koraka 4.*

B1. U sledećem kodu koristimo parametre $goal\_$ i $constr\_$, da bi predstavilii unutrašnji oblik postavljenog problema.

```
c=Transform[c]; var=SelectVariables[Append[c,g]];
n=Length[var];  m=Length[c];
For[i=1, i<= n, i++,
  AppendTo[nc, Coefficient[g,var[[i]]]];
];
mat=Formmatrix[c, var];  {l1,l2}=maximaln[mat,nc,c];
```

B2. Kostruisati i rešiti sistem (4.2.2). Sistem (4.2.2) se formira na sledeći način:

```
For[i=1, i<= n, i++,
   l11[[i]]=Equal[l1[[i,1]],0];
];
```

Rešiti sistem (4.2.2) i izračunati vrednost ciljne funkcije.

```
x=First[Solve[l11,var]];   x=x/.Rule->List;
For[i=1, i<= n, i++,
    g=g/.x[[i,1]]->x[[i,2]]
];
```

C. *Implementacija Koraka 5.* U ovom delu koda se proverava da li je konstruisano rešenje bazično dopustivo. Ovaj korak je takođe i priprema za simpleks metod.

```
test=True; i=1;
While[test && i<=n+m,
    If[x[[i,2]]<0, test=False, i++];
];
If[test,
   Print["Solution ",{x,g}, " is basic"],
   Print["Solution ", {x,g}, " is nonbasic"]
];
Return[{l1,l2,test}]
```

Implementacija Koraka 6.

Proizvoljno ograničenje zadato linearnom nejednakošću se može lako transformisati u odgovarajuće linearno ograničenje zadato jednakošću dodavanjem ili oduzimanjem pozitivne izravnavajuće promenljive. Taj cilj se postiže u sledećim koracima.

Korak S1. Transformišemo data ograničenja u oblik (4.4.1).

Korak S2. U sledećoj funkciji formalni parametar *lis* predstavlja ograničenje oblika (4.2.2) i *prom* je pozitivna izravnavajuća promenljiva. Rezultat je leva strana ekvivalentnih ograničenja zadatih jednakostima:

$$\sum_{k=1}^{n} a_{ik}x_k + r_i - \left(\sum_{k=1}^{n} b_{ik}x_k + s_i\right) \pm prom = 0.$$

```
AddVariable[lis_, prom_] :=
   Module[{},
      Switch[lis[[2]],
         LessEqual,     Return[Plus[lis[[1]],prom]],
         GreaterEqual,  Return[Substract[lis[[1]],prom]],
         Equal,         Return[lis[[1]]]
   ]  ]
```

Korak S3. Za svako $i = 1, \ldots, m$ u glavnoj funkciji sada transformišemo i-to dato ograničenje dodavanjem izravnavajuće promenljive $\$[i]$.

```
  ct=Transform[constraints];
  Do[ith=AddVariable[ct[[i]], $[i] ];
  AppendTo[system,Equal[ith,0]], {i,Length[constraints]} ];
```

Rešenje $x[[i,2]]$, $i = 1, \ldots, n$, dobijeno kao izlaz iz *Koraka 5*, može biti pripremljeno kao početna tačka $ksi[[i]]$, $i = 1, \ldots, m+n$ za simpleks metod na sledeći način:

```
  ksi=Table[i,{n+m}];
  For[i=1, i<= n, i++, ksi[[i]]=x[[i,2]] ];
  For[i=n+1,i<=2*n, i++, ksi[[i]]=0];
  For[i=1, i<= m-n, i++,
      ksi[[2*n+i]]=l2[[i,1]]
     ];
  For[i=1, i<= m-n, i++,
      For[j=1, j<= n, j++,
          ksi[[2*n+i]]=ksi[[2*n+i]]/.x[[j,1]]->x[[j,2]]
      ]
  ];
  For[i=2*n+1, i<= n+m, i++, ksi[[i]]=-ksi[[i]]];
```

Kada broj pozitivnih vrednosti $v_{i_1}, \ldots, v_{i_l}$, definisanih sa (4.2.1), zadovoljava $0 < l < n$, predlažemo sledeći algoritam:

Algoritam Al

Korak 1, *Korak 2.* Identični sa *Korakom 1* i *Korakom 2* of *Algoritma An*, respektivno.

Korak 3. Odrediti pozitivne vrednosti $v_{i_1}, \ldots v_{i_l}$ iz skupa $\{v_1, \ldots, v_m\}$.

Korak 4. Eliminisati proizvoljnih l nepoznatih, koristeći sistem (4.2.3). Pretpostavimo da je sistem (4.2.3) rešen u odnosu na promenljive $x_{i_1}, \ldots, x_{i_l}$. Neka je

bazično rešenje x_0 generisano izjednačavanjem ostali $n-l$ promenljivih sa nulom, i izračunavanjem promenljvih $x_{i_1}, \ldots, x_{i_l}$.

Korak 5. Primeniti opšti simpleks metod, koristeći bazično rešenje x_0.

Implementacija *Koraka 1*, *Koraka 2* i *Koraka 3* je opisana ranije. Sada opisujemo implementaciju *Koraka 4* i *Koraka 5*. Implementacija *Koraka 4* je opisana u nastavku. Sa *lvar* označavamo skup bazičnih promenljivih $x_{i_1}, \ldots, x_{i_l}$, dok *rvar* označava skup komplementarnih promenljivih.

```
x=First[Solve[l11,var]];
lvar={};
For[i=1, i<= l, i++,
    AppendTo[lvar, x[[i,1]] ]
];
rvar=Complement[var,lvar];
For[i=1, i<= n-l, i++,
    AppendTo[l11, Equal[rvar[[i]],0]]
];
x=First[Solve[l11,var]];
```

Rešenje $x[[i,2]]$, $i = 1, \ldots, n$, se može transformisati u početnu tačku $ksi[[i]]$, $i = 1, \ldots, m+n$ za simpleks metod primenom sledećeg kôda:

```
For[i=1, i<=n, i++,
    ksi[[i]]=x[[i,2]]
];
For[i=n+1, i<= n+l, i++, ksi[[i]]=0 ];
For[i=n+l+1, i<=n+m, i++, ksi[[i]]=l2[[i-n-l,1]] ];
For[i=n+l+1, i<=n+m, i++,
    For[j=1, j<=n, j++,
        ksi[[i]]=ksi[[i]]/.x[[j,1]]->x[[j,2]]
    ]
];
For[i=n+l+1, i<= n+m, i++, ksi[[i]]=-ksi[[i]]    ]
```

Konačno, prema Teoremi 4.3.1 predlažemo sledeći algoritam. Uslovi za primenu Teoreme 4.3.1 su pretpostavljeni.

Algoritam Ag

Korak 1. Izračunati

$$d_{ij} = \frac{a_{ij}}{b_i c_j}, \quad i = 1, \ldots, m, \ j = 1, \ldots, n.$$

Korak 2. Ako je uslov (4.3.4) ispunjen, iskoristiti vektor $(x_1, \ldots, x_n)$, definisan sa (4.3.5), za optimalnu tačku. Izlaz je vektor $(x_1, \ldots, x_n)$ i algoritam staje.

Korak 3. Ako uslov (4.3.4) nije zadovoljen, razmotriti sledeće slučajeve:

Slučaj 1. Ako postoje k, l tako da je $d_{ik} \geq d_{il}$, $i = 1, \ldots, m$, izbaciti k-tu kolonu iz početnog problema.

Slučaj 2. Ako postoje k, l tako da je $d_{kj} \leq d_{lj}$, $j = 1, \ldots, n$, izbaciti k-tu vrstu.

Slučaj 3. U suprotnom, eliminisati suvišna ograničenja primenom (4.3.1).

Posle svakog od ovih slučajeva, primeniti opšti algoritam simpleks metoda.

4.5 Numerički primeri

Geometrijske osobine Teoreme 4.2.1 *Algoritma An* su ilustrovane u sledeća četiri primera.

Primer 4.5.1 U ovom primeru maksimiziramo ciljnu funkciju $321x1+287x2$ pod sledećim skupom ograničenja:

$1.02x1+5x2 \leq 5103$, $1.032x1+4x2 \leq 4130.2$, $0.314x1+x2 \leq 1048.2$, $1.122x1+3x2 \leq 3202.8$,
$0.872x1+2x2 \leq 2182.2$, $0.504x1+x2 \leq 1119.8$, $0.577x1+x2 \leq 1154.7$,
$1.974x1+3x2 \leq 3592.2$, $3x1+4x2 \leq 5000$, $0.855x1+x2 \leq 1315.9$,
$0.98x1+x2 \leq 1400.2$, $4x1+3x2 \leq 5000$, $3.036x1+2x2 \leq 3636.6$,
$3.464x1+2x2 \leq 4000$, $1.134x1+x2 \leq 1511.8$, $1.984x1+x2 \leq 2222.2$,
$2.291x1+x2 \leq 2500$, $2.676x1+x2 \leq 2856.9$, $3.873x1+x2 \leq 4000$

Vrednosti v_i su, respektivno, sledeće:

345.3707765, 358.0919724, 369.9832633, 381.2617368, 391.3741373,
400.7613457, 409.0139395, 416.1997108, 422.2, 426.739932,
429.6561598, 429., 425.9470426, 421.495273, 430.5825196,
415.82367, 409.0075608, 401.1548539, 382.5567078

Transformišući nejednakosti koje odgovaraju maksimalnim v-ovima u odgovarajuće jednakosti, dobijamo sledeći linearni sistem:

$$\{-1511.8 + 1.134x1 + x2 = 0,\ \ -1400.2 + 0.98x1 + x2 = 0\}$$

Rešenje $x_0 = (724.675, 690.018)$ tog sistema je bazično dopustivo. Simpleks metod samo potvrđuje da je ta tačka ekstremna. Prema tome, u ovom slučaju je $x_0 = x_P$, i minimalni uglovi su zahvaćeni u ekstremnoj tački. Maksimalna vrednost ciljne funkcije je $z_{\max} = 430655.997402597545$.

Sa druge strane, originalni simpleks metod daje isto rešenje u 9 iteracija. Takođe, kanoničan oblik datog problema sadrži 21 promenljivu:

$$\begin{aligned}
-5103 + 1.02x1 + 5x2 + \$[1] &= 0, & -4130.2 + 1.032x1 + 4x2 + \$[2] &= 0,\\
-1048.2 + 0.314x1 + x2 + \$[3] &= 0, & -3202.8 + 1.122x1 + 3x2 + \$[4] &= 0,\\
-2182.2 + 0.872x1 + 2x2 + \$[5] &= 0, & -1119.8 + 0.504x1 + x2 + \$[6] &= 0,\\
-1154.7 + 0.577x1 + x2 + \$[7] &= 0, & -3592.2 + 1.974x1 + 3x2 + \$[8] &= 0,\\
4 - 5000 + 3x1 + 4x2 + \$[9] &= 0, & -1315.9 + 0.855x1 + x2 + \$[10] &= 0,\\
-1400.2 + 0.98x1 + x2 + \$[11] &= 0, & -5000 + 4x1 + 3x2 + \$[12] &= 0,\\
-3636.6 + 3.036x1 + 2x2 + \$[13] &= 0, & -4000 + 3.464x1 + 2x2 + \$[14] &= 0,\\
-1511.8 + 1.134x1 + x2 + \$[15] &= 0, & -2222.2 + 1.984x1 + x2 + \$[16] &= 0,\\
-2500 + 2.291x1 + x2 + \$[17] &= 0, & -2856.9 + 2.676x1 + x2 + \$[18] &= 0,\\
-4000 + 3.873x1 + x2 + \$[19] &= 0.
\end{aligned}$$

Napomenimo da se primenom programa PCx dobija isto rešenje u 6 iteracija.

Primer 4.5.2 U ovom primeru, *Algoritam An* daje bazično dopustivo rešenje koje pripada istoj hiperravni kao i ekstremna tačka. Posle toga se ekstremna tačka određuje u samo jednom koraku simpleks metoda. Razmotrimo sledeći maksimizacioni problem: maksimizirati $z(x) = x1 + x2$, u odnosu na ograničenja

$$\{x1/1000 + x2/9 \leq 1,\ \ x1/5 + x2/70 \leq 1,\ \ x1/4.9 + x2/111 \leq 1\}.$$

Primenom *Algoritma An*, dobijamo sledeće bazično dopustivo rešenje:

$$x_0 = (4.73821, 3.66509).$$

Međutim, tačka x_0 nije ekstremna. Primenom simpleks algoritma za bazično dopustivo rešenje, uzimajući za početnu tačku x_0, dobijamo u jednom dodatnom koraku simpleks metoda sledeću ekstremnu tačku

$$x_0 = (4.35995, 8.96076)$$

i maksimalnu vrednost 13.32071 ciljne funkcije.

Primetimo da tačke x_0 i x_P pripadaju hiperravni $x1/5 + x2/70 = 1$.

Primenom simpleks metoda, dobijamo identičan rezultat u tri koraka.

Primer 4.5.3 U ovom primeru, *Algoritam An* takođe daje bazično dopustivo rešenje koje pripada istoj hiperravni kao i ekstremna tačka. Međutim, ekstremna tačka se ne dobija u jednom koraku simpleks metoda. Maksimizirati $11x1 + 18x2 + 29x3 - 0.8x4$ u odnosu na uslove

$$x1 + 4x2 + 6x3 + 2x4 \leq 60,\ \ 4x1 + x2 + 5x3 + 9x4 \leq 91$$
$$6x1 + 5x2 + 8x3 + 5x4 \leq 102,\ \ 2x1 + 5x2 + 7x3 + 7x4 \leq 110,$$
$$-x1/2 + x2/5 - x3/3 - x4/2 \leq 1$$

Primenom *Algoritma An*, dobijamo bazično dopustivo rešenje

$$x_0 = \left(\frac{233}{157},\ \frac{2705}{314},\ \frac{227}{157},\ \frac{2415}{314}\right) \tag{4.5.1}$$

i odgovarajuću vrednost $\frac{67301}{157}$ ciljne funkcije. Kasnije, primenom simpleks metoda, startujući iz tačke (4.5.1), dobijamo maksimalnu vrednost $\frac{4467}{14}$ u 2 dodatne iteracije. Ukupno procesorsko vreme je 1.26 sekundi. Ekstremna tačka je

$$x_P = \left(\frac{33}{7},\ 0,\ \frac{129}{14},\ 0\right). \tag{4.5.2}$$

Primetimo da tačke x_0 i x_P, izražene sa (4.5.1) i (4.5.2), pripadaju istoj hiperravni $x1 + 4x2 + 6x3 + 2x4 = 60$.

Original simpleks metod daje rešenje u 7 iteracija i 1.59 sekundi.

Primer 4.5.4 U ovom primeru pokazujemo da je *Algoritam An* moguće primeniti i na linearne programe koji sadrže negativne koeficijente u ograničenjima. Maksimizirati $11x1 + 18x2 + 29x3 + 28x4$ pod uslovima

$$x1 + 4x2 + 6x3 + 2x4 \leq 60,\ \ 4x1 + x2 + 5x3 + 9x4 \leq 91,$$
$$6x1 + 5x2 + 8x3 + 5x4 \leq 102,\ \ 2x1 + 5x2 + 7x3 + 7x4 \leq 110,$$
$$-x1/2 + x2/5 - x3/3 - x4/2 \leq 1,\ \ x1/7 - x2/3 - x3/2.3 + x4/3.4 \leq 1$$

Primenom *Algoritma An*, dobijamo maksimalnu vrednost 428.668789808917161 i ekstremnu tačku

$$x_0 = x_P = (1.48408,\ 8.61465,\ 1.44586,\ 7.69108).$$

Napomenimo da simpleks metod daje ovo rešenje u 5 iteracija.

U sledećem primeru ilustrujemo primenu *Algoritma Al*.

Primer 4.5.5 Maksimizirati $z = x4$ pod ograničenjima

$$1/4 \leq x1,\ \ x1 \leq 1,\ \ 1/4 * x1 \leq x2,\ \ x2 \leq 1 - 1/4 * x1,$$
$$1/4 * x2 \leq x3,\ \ x3 \leq 1 - 1/4 * x2,\ \ 1/4 * x3 \leq x4,\ \ x4 \leq 1 - 1/4 * x3$$

Ovaj problem je poseban slučaj ($n = 4$) poznatog primera iz [38], koji demonstrira da simpleks algoritam nije polinomijalan. Simpleks metod konvergira u 7 iteracija i daje maksimalnu vrednost 255/256 i ekstremnu tačku

$$(1/4,\ 1/16,\ 1/64,\ 255/256). \tag{4.5.3}$$

Primenom *Algoritma Al*, dobijamo sledeće skalarne proizvode, respektivno:

$$\{0,\ 0,\ 0,\ 0,\ 0,\ 0,\ -0.9701425001,\ 0.9701425001\}.$$

Postoji samo jedan pozitivan skalarni proizvod, i *Algoritam An* nije u mogućnosti da da rešenje. Tada je primenjen *Algoritam Al*. Poslednji skalarni proizvod je pozitivan i pouzdano možemo upotrebiti samo jednačinu $x4 = 1 - 1/4 * x3$, koja nastaje iz poslednje nejednakosti. Koristeći tu jednačinu zajedno sa $x1 = x2 = x4 = 0$, dobijamo bazično rešenje

$$x_0 = (0,\ 0,\ 4,\ 0).$$

Sada, primenom simpleks metoda, startujući iz tačke x_0, dobijamo maksimalnu vrednost 255/256 i ekstremnu tačku (4.5.3) u 5 iteracija.

Sada ilustrujemo eliminaciju suvišnih ograničenja pod uslovima (4.3.2).

Primer 4.5.6 Razmotrimo maksimizaciju ciljne funkcije $2x1 + 3x2 + x3 + 5x4$, uz ograničenja

$$x1/10 + x2/15 + x3/12 + x4/15 <= 1,\ x1/12 + x2/16 + x3/14 + x4/20 <= 1,$$
$$x1/12 + x2/10 + x3/9 + x4/11 <= 1,\ x1/22 + x2/15 + x3/12 + x4/30 <= 1,$$
$$x1/8 + x2/6 + x3/5 + x4/9 <= 1,\ x1/3 - x2/4 + x3/2 - x4/5 >= 1.$$

Koristeći klasičan simpleks metod, dobijamo maksimalnu vrednost $\frac{1797}{67}$ i ekstremnu tačku

$$\left(\frac{336}{67},\ 0,\ 0,\ \frac{225}{67}\right)$$

u 9 iteracija. Kako je uslov (4.3.2) ispunjen za $p = 1$, $q = 2$, druga nejednačina je suvišna. Izostavljajući drugo ograničenje, simpleks metod daje identičan rezultat u 8 iteracija. Dakle, eliminacija suvišnih ograničenja ne samo da smanjuje dimenziju problema, već i u mnogim slučajevima redukuje broj iterativnih koraka.

U ovom primeru zbog postojanja suvišnog ograničenja ne važi uslov $x_0 \in P$, i rezultati Teoreme 4.2.1 nisu pouzdani u tom slučaju.

Primer 4.5.7 Razmotrimo problem (4.2.2). Kako je u Primeru 4.2.2 pokazano, metod minimalnih uglova daje tačku $x_0 = (1/2, 3/2) \notin P$. Međutim, tačka x_0 nije bazično dopustiva, i primenjujemo simpleks metod za nedopustivu startnu tačku. Tada simpleks metod konvergira u 3 iteracije. Originalni simpleks metod takođe daje rešenje u 3 iteracije. Dakle, računanje minimalnih uglova ne poboljšava konvergenciju simpleks metoda što znači da je uslov $x_0 \in P$ potreban.

U sledeća dva primera ilustrujemo primenu Teoreme 4.3.1.

Primer 4.5.8 Odrediti maksimum ciljne funkcije $x1 + x2 + 2x3 + 5x4 + 2x5$ pod uslovima:

$$30.4x1 + 12x2 + 16.5x3 + 30x4 + 16.3x5 \leq 2,$$
$$60x1 - 15.1x2 - 20x3 - 25.6x4 + 40x5 \leq 5,$$
$$48.7x3 + 24x2 + 80.2x3 + 40x4 + 96.5x5 \leq 8,$$
$$18.7x1 + 15x2 + 18.7x3 + 30x4 + 18.7x5 \leq 3.$$

Simpleks metod daje optimalnu vrednost 0.333333333333333303 i optimalnu tačku

$$(0,\ 0,\ 0,\ 0.0666667,\ 0)$$

u 9 iteracija. Primenom dela (a) Teoreme 4.3.1 dobijamo direktno to rešenje.

Primer 4.5.9 Maksimizirati $2x1 + 0.5x2 + 4x3 + x4 + 3x5 + 5x6$ pod uslovima:

$$20x1 + 6x2 + 32.3x3 - 6x4 + 24x5 + 60.5x6 \leq 2,$$
$$32x1 - 6x2 - 32x3 - 4x4 + 48.3x5 + 160.6x6 \leq 4,$$
$$20x1 + 7.5x2 + 100x3 + 5x4 + 90.7x5 + 50x6 \leq 5,$$
$$24x1 + 15x2 + 72.7x3 + 12x4 + 54x5 + 120.6x6 \leq 6.$$

Za 2.47 sekundi i 4 iteracije simpleks metoda dobijamo maksimalnu vrednost $\frac{1}{2}$ i sledeću ekstremnu tačku:

$$\left(\frac{3}{20},\ 0,\ 0,\ \frac{1}{5},\ 0,\ 0\right).$$

Posle eliminacije kolona 5 i 6 (na osnovu Teoreme 4.3.1), dobijamo isto rešenje u 1.32 sekundi i takođe u 4 iteracije.

Napomena 4.5.1 *Za mali broj ograničenja broj iteracija se smanjuje u odnosu na simpleks algoritam, ali može da se desi da je procesorsko vreme povećano zbog dodatnih izračunavanja pre primene simpleks metoda.*

U sledećem primeru dajemo jedan problem većih dimenzija sa 63 ograničenja i 32 promenljive.

Primer 4.5.10 Maksimizirati ciljnu funkciju

$$\begin{aligned}&5x11 + 4x12 + 2x13 + 3x14 + 3x15 + 9x16 + 2x17 + x18 + 0.5x19 + x20 + x21 + x22\\&+ 5x23 + x24 + 3x25 + x26 + x27 + x28 + x29 + 8x30 + x31 + 3x32 + x33\\&+ 5x34 + x35 + 7x36 + x37 + 0.5x38 + x39 + x40 + x41 + x42\end{aligned}$$

u odnosu na ograničenja

$10x13+8x17+0.1x24+0.5x37 \leq 220$, $5x11 \leq 9$, $6x17+0.1x21-0.2x29+0.1x31+0.7x37 \leq 90$, $4x12 \leq 6$ $-0.4x13 + 2x21 + 0.2x24 + 0.1x27 \leq 20$, $3x14 \leq 20$, $2x24 + 0.3x29 + 0.2x35 \leq 40$, $3x15 \leq 40$, $9x16 \leq 16$, $0.4x13 + 4x27 - 0.2x31 + 0.5x42 \leq 160$, $x18 \leq 120$, $-0.4x17 + 3x29 + 0.6x35 - 0.1x37 \leq 120$, $0.5x19 \leq 320$, $0.6x17 + 4x31 \leq 320$, $x20 \leq 200$, $-x21 - 2x22 \leq -1$, $-0.4x21 - 0.1x27 + 0.6x29 + 5x35 \leq 200$, $5x23 \leq 7$, $-0.1x24-0.3x31+6x37+0.1x42 \leq 120$, $x26 \leq 2$, $-0.2x13-0.2x21+0.6x27+4x42 \leq 40$, $3x25 \leq 8$, $x40 \leq 42$, $-2x22 - 15x23 \leq -2$, $-15x23 - 4x24 \leq -0.5$, $x39 \leq 14$, $-4x24 - 15x25 \leq -3$, $-15x25 - 6x26 \leq -2$, $0.5x38 \leq 70$, $-6x26 - 7x27 \leq -2$, $-7x27 - 8x28 \leq -5$, $7x36 \leq 11$, $-8x28 - 9x29 \leq -3$, $-9x29 - 8x30 \leq -4$, $5x34 \leq 6$, $-8x30 - x31 \leq -0.2$, $-x31 - 6x32 \leq -1$, $x33 \leq 30$, $-6x32 - 3x33 \leq -2$, $-3x33 - 20x34 \leq -5$, $3x32 \leq 20$, $-20x34 - 5x35 \leq -3$, $-5x35 - 42x36 \leq -4$, $8x30 \leq 16$, $-42x36 - 7x37 \leq -9$, $-7x37 - 4x38 \leq -7$, $x28 \leq 49$, $-4x38 - 9x39 \leq -5$, $-9x39 - x40 \leq -1$, $x22 \leq 40$, $-x40 - x41 \leq -0.1$, $-x41 - 2x42 \leq 2$, $-10x11 - 12x12 \leq -0.2$, $-16x12 - 10x13 \leq -2$, $x41 \leq 120$, $-6x13 - 3x14 \leq -0.3$, $-6x14 - 12x15 \leq -2$, $-24x15 - 54x16 \leq -3$, $-54x16 - 14x17 \leq -2$, $14x17 - 8x18 \leq -5$, $-8x18 - 4.5x19 \leq -3$, $-4.5x19 - x20 \leq -1$, $-x20 - x21 \leq -0.5$.

PCx rešava ovaj problem u 7 iteracija. Koristeći *Algoritam An* dobijamo maksimalnu vrednost 1657.32225559107115 i ekstremnu tačku

$$\begin{aligned}&(1.8,\ 1.5,\ 20.6827,\ 6.66667,\ 13.3333,\ 17.7778,\ 0.,\ 120.,\ 640.,\ 200.,\ 10.9497,\\&40.,\ 1.4,\ 11.0254,\ 2.66667,\ 2.,\ 41.2576,\ 49.,\ 33.2613,\ 2.,\ 80.,\ 6.66667,\\&30.,\ 1.2,\ 37.7098,\ 1.57143,\ 24.0974,\ 140.,\ 14.,\ 42.,\ 120.,\ 5.392980.)\end{aligned}$$

u samo dve iteracije.

Sledeći primer je iz [4]. Programi PCx i HOPDM nisu u mogućnosti da reše dati problem usled numeričke nestabilnosti. Sa druge strane, *Algoritam Al*, kao i originalni simpleks metod rešavaju ovaj problem posle samo 3 iteracije!

Primer 4.5.11 Maksimizirati

$$\begin{aligned}&- 8919x1 - 5981x2 - 9892x6 - 3x10 - 9800x11 - 9989x14\\&- 993x15 + 9978x17 - 9687x18 - 9993x19 + 9800x20\end{aligned}$$

pod uslovima

$\{8919x1 - 4788x2 - 2x3 - 9733x4 - 3993x5 - x7 - x8 - 9002x9 - 9789x10 + x11 - 3x12 - 9971x13 + x14 - x15 - x16 - 9978x17 + 9687x18 + 9993x19 - x20 = -9791$,

$-8919x1 - 4790x2 + 2x3 - 9733x4 - 3993x5 - 2x6 + x7 + x8 - 9002x9 + 9789x10 + x11 - 3x12 - 9971x13 + 4902x14 - x15 + x16 + 9978x17 - 9687x18 - 9993x19 - x20 = 9789$,

$8919x1 - x2 - 2x3 - 2x6 - x7 - x8 + 9789x10 - x11 + 4901x14 - x16 - 9978x17 + 9687x18 + 9993x19 + x20 = -9790$, $-4788x2 - 2x3 - x7 - x8 - 9789x10 + x14 - x15 - x16 = -9790$,

$8919x1 - 4789x2 + 2x3 + 9733x4 + 3993x5 + x7 + x8 + 9002x9 + 3x12 + 9971x13 + x14 - x15 + x16 - 9978x17 + 9687x18 + 9993x19 = 9791,$

$-8919x1 + 4789x2 - 2x3 + 9733x4 + 3993x5 + 2x6 - x7 - x8 + 9002x9 + x11 + 3x12 + 9971x13 - 4902x14 + x15 - x16 + 9978x17 - 9687x18 - 9993x19 - x20 = -9789,$

$4788x2 + 2x3 - 9733x4 - 3993x5 - 2x6 + x7 + x8 - 9002x9 + 9789x10 + x11 - 3x12 - 9971x13 + 4900x14 + x15 + x16 - x20 = 9789,$

$-x1 \leq -9872, \quad -x16 \leq -8790\}.$

U 3 iteracije dobijamo maksimalnu vrednost 0, i sledeću ekstremnu tačku:

$$(9872,\ 0,\ 500,\ 0,\ 0,\ 0\,0,\ 0,\ 0,\ 0,\ 0,\ 0,1/3,\ 0,\ 0,\ 0,8790,\ \frac{14674728}{1663},\ 0,\ 0,\ 0).$$

Dalje, PCx nije u mogućnosti da reši ovaj problem i staje posle 9 iteracija sa neodređenim statusom:

```
Problem 'KBAPAH ' terminated with UNKNOWN status
9 iterations
Terminated with status UNKNOWN (code 3)
Solution at iteration 8:
Primal Objective = -3.23998779e+003
Dual Objective = -4.40527272e+002
```

Slično, HOPDM staje posle 6 iteracija sa podoptimalnim statusom, nudeći maksimalnu vrednost $-7.1e + 1$.

U radu [4] je data familija degenerisanih problema sa ekstremnom vrednošću ciljne funkcije koja je jednaka nuli. Malom modifikacijom dobija se sledeći primer kod koga je ekstremna vrednost različita od nule. Tako dobijeni problem takođe nije rešiv primenom programa PCx.

Primer 4.5.12 U ovom primeru maksimiziramo ciljnu funkciju kao u Primeru 4.5.11. Ograničenja su kao u Primeru 4.5.11, osim prvog i sedmog ograničenja, koja su zamnenjena sa sledećim ograničenjima, respektivno:

$8919x1 - 4788x2 - 2x3 - 9733x4 - 3993x5 - x7 - x8 - 9002x9 - 9789x10 + x11 - 3x12 - 9971x13 + x14 - x15 - x16 - 9978x17 + 9687x18 + 9993x19 - x20 \leq -9791,$

$4788x2 + 2x3 - 9733x4 - 3993x5 - 2x6 + x7 + x8 - 9002x9 + 9789x10 + x11 - 3x12 - 9971x13 + 4900x14 + x15 + x16 - x20 = 9782.$

U 7 iteracija dobijamo maksimalnu vrednost -24196384, i ekstremnu tačku

$$(9872,\ 0,\ \frac{1001}{2},\ 0,\ 0,\ 2450,\ 0,\ 0,\ 0,\ 0,\ 0,\ 4/3,\ 0,\ 1,\ 0,\ 8790,\ \frac{88048373}{9978},\ 0,\ 0,\ 5).$$

Originalni simpleks metod rešava ovaj problem u 8 iteracija. Kao što smo napomenuli, PCx nije u mogućnosti da reši ovaj problem:

```
Problem 'NIS ' terminated with UNKNOWN status
10 iterations
Terminated with status UNKNOWN (code 3)
Solution at iteration 10:
Primal Objective = -7.80713398e+004
Dual Objective = -5.46606497e+004
```

U ovom odeljku smo razmotrili nekoliko važnih problema u linearnom programiranju i primeni simpleks metoda: konstrukciju pogodne početne tačke, izbor pivot elementa i eliminaciju suvišnih ograničenja. Dalje, uveli smo takozvani metod minimalnih uglova. Tom heuristikom biramo $l \leq n$ oganičenja, tako da hiperravan

generisana tim ograničenjima zahvata l minimalnih uglova iz intervala $\left[0, \frac{\pi}{2}\right)$ sa gradijent vektorom ciljne funkcije. Izabrana ograničenja, koja su data nejednakostima, transformišemo u odgovarajuće jednakosti. U slučaju kada je $l \geq n$, biramo n minimalnih uglova i dobijamo sistem od n linearnih jednačina sa n promenljivih. Rešenje tog linearnog sistema se koristi ako bazično rešenje za simpleks metod. Ako je broj l uglova iz intervala $\left[0, \frac{\pi}{2}\right)$ manji od n, eliminišemo $l < n$ nepoznatih i koristimo hiperravni koje odgovaraju pozitivnim vrednostima iz skupa V. Tada početno bazično rešenje za simpleks metod može biti generisano izjednačavanjem ostalih $n - l$ promenljivih sa nulom.

Za uspešnu primenu metoda minimalnih uglova, potrebno je da problem (4.1.1)-(4.1.2) bude postavljen bez suvišnih ograničenja. U nekim slučajevima, metod je primenljiv i na probleme sa suvišnim ograičenjima.

Međutim, problem otkrivanja suvišnih ograničenja je veoma kompleksan. Uveli smo nekoliko pravila eliminacije koja se zasnivaju na geometrijskim osobinama suvišnih ograničenja. Takođe su data eliminaciona pravila koja se zasnivaju na teoriji igara. U nekim slučajevima primena teorije igara generiše direktno rešenje problema.

Nažalost, svi predloženi algoritmi za eliminaciju suvišnih ograničenja ne garantuju kompletnu eliminaciju. U tom slučaju, metod minimalnih uglova se može primeniti za gnerisanje startne tačke za simpleks metod. Ako je problem postavljen bez suvišnih ograničenja, u mnogim slučajevima dobijamo značajno ubrzanje simpleks procedure.

Implementacija metoda minimalnih uglova je razvijena u programskom paketu MATHEMATICA. Efikasnost primenjenih metoda je upoređena sa standardnim simpleks algoritmom i metodima unutrašnje tačke. Generalno, PCx i HOPDM su brži od metoda minimalnih uglova. Na primer, metod minimalnih uglova rešava test problem $Afiro$ iz $Netlib$ biblioteke problema u 15 iteracija i PCx rešava isti problem u 8 iteracija. Međutim, postoji familija degenerisanih problema linearnog programiranja kod koje se javljaju numerički problemi kod primene metoda unutrašnje tačke, kao što su PCx i HOPDM. Ta familija je konstruisana u [4] i [42]. Ti problemi su rešivi metodom minimalnih uglova. Sa druge strane, postoje problemi koji ne mogu biti rešeni metodom minimalnih uglova. Na primer, test problem *Blend* sadrži loše uslovljenu matricu što prouzrokuje značajne numeričke greške, tako da metod minimalnih uglova ne može da ga reši. Iz tih razloga se u narednoj glavi razmatra implementacija i modifikacija primal-dual metoda sa ciljem da se poveća tačnost i numerička stabilnost iterativnog procesa.

4.6 Direktni heuristični algoritam sa uopštenim inverzima

U ovom delu dajemo algoritam iz rada [67]. Ovaj heuristički metod vrlo često daje ili optimalno, ili rešenje iz koga se, primenom simpleks metoda dobija optimalno rešenje primenom malog broja iteracija. Osnovna ideja ovog metoda je prikazivanje opšteg rešenja linearnog sistema $Ax = b$ pomoću *pseudoinverza* $A^{\dagger}$

(Moore-Penroseovog uopštenog inverza) matrice A. Više o Moore-Penroseovom inverzu kao i ostalim uopštenim (generalisanim) inverzima operatora i matrica može se naći u monografijama [6, 87]. Napomenimo samo da je matrica $A^\dagger$ jedinstveno rešenje sledećeg sistema matričnih jednačina:

$$AXA = A, \quad XAX = X, \quad (AX)^* = AX, \quad (XA)^* = XA$$

Na sličan način se definišu i ostali uopšteni inverzi matrice A. Postoji više metoda za izračunavanje Moore-Penroseovog i ostalih generalisanih inverza matrice. Metod koji je u literaturi poznat kao Leverrier-Faddev ili Souriau-Frame metod koristi karakteristični polinom matrice A. Ovaj metod je prevashodno namenjen za simboličko izračunavanje uopštenih inverza. U našim radovima [71, 62, 61] pokazana je modifikacija metoda Leverrier-Faddev ukoliko je matrica A polinomijalna matrica. Pomenimo još i metod pregradjivanja u kome se Moore-Penroseov inverz računa primenom odgovarajućih rekurentnih formula. Modifikacije ovog metoda za polinomijalne matrice jedne ili više promenljivih prikazane su u našim radovima [60, 59].

Posmatramo problem linearnog programiranja u standardnom obliku:

$$\begin{array}{ll} \min & c^T x, \\ \text{p.o.} & Ax = b, \\ & x \geq 0. \end{array} \tag{4.6.1}$$

Ulazni podaci algoritma su A, b, c dok su izlazni podaci vektor x, vrednost ciljne funkcije z i komentari na osnovu testa optimalnosti.

Algoritam DHALP.

Korak 1. Učitati m, n, $A = [a_{ij}]$, b, c.

Korak 2. Izračunati $d = A^\dagger b$, gde je $d = [d_i]$ vektor dimezije $n \times 1$ a $A^\dagger$ je Moore-Penrose inverz (ili p-inverz) i neka je $e = Ad$.

Ako je $e \neq b$, izlaz je "Problem je nedopustiv" i algoritam staje.

Korak 3. Izračunati

$$\begin{gathered} H = A^\dagger A, \\ c' = (I - H)c, \\ s_k = \min\left\{ \tfrac{d_i}{c'_i} \,|c'_i > 0 \right\}, \\ x = d - c' s_k, \end{gathered}$$

gde je $c' = [c'_i]$ $n \times 1$ vektor, I je $n \times n$ jedinična matrica, H je $n \times n$ matrica. Vektor pravca $c' s_k$ pokušava da vektor x iz skupa rešenja jednačine $Ax = b$ pomeri u skup dopustivih rešenja definisan sa $Ax = b$, $x \geq 0$ ukoliko on već nije u njemu. Rešenje će biti ekstremna tačka skupa dopustivih rešenja.

Korak 4. Ukloniti onu nepoznatu x_i koja se izjednačava sa nulom, ukloniti i-tu kolonu matrice A i odgovarajuću koordinatu c_i iz vektora c ciljne funkcije. Izračunati $d = A^\dagger b$.

Napomena 4.6.1 *Sledeća matrica $A^\dagger$ u Koraku 4 je izračunata iz prethodne $A^\dagger$. To izračunavanje zahteva $O(mn)$ operacija. Moguće je izračunati A^+ iz A, ali to zahteva $O(mn^2)$ operacija. Ako se dve ili više koordinata x_i izjednačavaju sa nulom treba primeniti algoritam izbacujući jedno x_i i izračunati odgovarajući vektor x. Onaj vektor x koji daje minimum ciljne funkcije će biti traženo rešenje (izlaz) algoritma. Ne postoji odgovarajući kriterijum za odluku koje od dve (ili više) koordinata x_i koje se izjednačavaju sa nulom izbaciti iz baze. Međutim, takva situacija nije tako česta.*

Korak 5. Ponoviti Korake 3 i 4 sve dok je $s_k \neq 0$ ili ga nije moguće izračunati (tj. $c_i' \leq 0$ za svako i).

Korak 6. Izračunati vrednost ciljne funkcije $z = c^T x$ gde su x i c dobijeni u poslednjem koraku algoritma i izbaciti rezultat.

Test optimalnosti. Neka je x vektor koji smo dobili primenom algoritma, i neka su A, b i c iz (4.6.1) poznati. Neka je B bazična matrica koja se sastoji od kolona matrice A koje odgovaraju bazičnim koordinatama x_i vektora x, neka je c_B^T vektor koji se sastoji od koordinata vaktora c koje odgovaraju bazičnim koordinatama x_i vektora x, i neka je p_j $j$ta kolona nebazične kolone od A.

Korak 7. Izračunati $y^T = c_B^T B^{-1}$ (vektor vrsta).

Korak 8. Izračunati $z_j - c_j = y^T p_j - c_j$ za sve nebazične vektore p_j.

Korak 9. Ako je $z_j - c_j \leq 0$ za svako j, rešenje je optimalno; u suprotnom rešenje je neograničeno ili algoritam ne može da odredi optimalno rešenje. U tom slučaju se optimalno rešenje dobija primenom simpleks algoritma koristeći vektor x kao dopustivu početnu tačku.

Komentar algoritma DHALP.

U Koraku 1 učitavamo podatke. Korak 2 proverava saglasnost jednačina $Ax = b$. U slučaju $e = AA^\dagger b \neq b$, problem nema dopustivo rešenje. Opšte rešenje sistema $Ax = b$ je $x = A^\dagger b \pm Pz$, gde je $P = (I - AA^\dagger)$ operator ortogonalne projekcije koji proizvoljan vektor z projektuje ortogonalno na nul prostor matrice A. Izračunavamo tačku c' u nul prostoru matrice A u Koraku 3. Vektor $x = d - c's_k = A^\dagger b - (I - A^\dagger A)cs_k$ je oblika $A^\dagger b - Pz$, gde cs_k odgovara proizvoljnoj koloni vektora z. Skalar s_k se u Koraku 3 računa na taj način iz dva razloga:

(i) jedna (ili više) koordinata x_i vektora x se izjednačavaju sa nulom,

(ii) smanjuje se vrednost ciljne funkcije.

Osim toga vektor x se pomera u skup dopustivih rešenja, ukoliko već nije u njemu. Ponekad (prilično retko) se desi da se bazična koordinata izjednači sa nulom i označi kao nebazična. Otvoren je problem pod kojim se potrebnim i dovoljnim uslovima to ne može desiti.

U Koraku 4 se eliminišu promenljive x_i koje se izjednačavaju sa nulom. U numeričkim eksperimentima, u više od 95% problema su takve koordinate nebazične.

U Koraku 5 proveravamo kriterijum za kraj algoritma dok u Koraku 6 dajemo izlazne podatke. U Koracima 7, 8 i 9 proveravamo da li je rešenje optimalno.

Najčešće je matrica B kvadratna i nesigularna i u tom slučaju optimalnost rešenja proveravamo direktno.

Ako je matrica B pravougaona ($m < n$), tada dodajemo jednu (ili više) vrsta matrici B i jedan (ili više) odgovarajućih elemenata vektoru b tako da novodobijen matrica B i vektor b ne menjaju vektor x i da je pri tome B kvadratna nesingularna matrica. Nakon toga primenimo test optimalnosti kao u Koracima 7, 8 i 9. Ukoliko nije $z_j - c_j \leq 0$ za svako j, primenimo simpleks algoritam koristeći x kao početnu tačku.

5 Opis korišćenih programa

U ovoj sekciji su opisani neki od programa koji su korišćeni u prethodnim sekcijama. Program GEOM se može koristiti kao ilustracija za geometrijski metod rešavanja dvodimenzionog linearnog programiranja. Pomoću programa Marplex i RevMarplex implementirane su opisane modifikacije simpleks algoritma.

5.1 Program GEOM

U prvom delu funkcije GEOM traži se skup dopustivih rešenja, odnosno njegove ekstremne tačke [80].

Najpre se prikazuje skup ograničenja koristeći funkciju `InequaltyPlot` koja se nalazi u paketu `Graphics‘InequalityGraphics`. Rešavajući sistem nejednačina određujemo oblast dopustivih rešenja `h`. Koristimo funkciju `InequalitySolve` iz paketa `Algebra‘InequalitySolve`. Ekstremne tačke pamtimo u `res` i dobijamo ih u preseku svake dve prave dobijene prevođenjem nejednačina iz liste ograničenja u jednačine. Sada sortiramo skup ekstremnh tačaka `res` prema udaljenosti od prave određene funkcijom cilja.

```
GEOM[f_, g_List] :=
    Module[{res = {}, res2 = {},var,p2,h,g1,i,j,res,res2,p1,mx,my,r,cf,n},

    (* Nalazenje skupa dopustivih resenja *)
      var = Variables[f];
      p2 = InequalityPlot[g, {var[[1]]}, {var[[2]]},
                          AspectRatio->1, DisplayFunction->Identity
                         ];
      h = g/.{List->And};
      h = InequalitySolve[h, var];
      If [h == False, Print["Problem is infeasible!!!!"]; Break[]; ];
      g1 = g/.{LessEqual->Equal, GreaterEqual->Equal, Less->Equal, Greater->Equal};
      For[i = 1, i=Length[g1]-1, i++,
        For[j= i+1, j = Length[g1], j++,
            t = FindInstance[g1[[i]] && g1[[j]] && h, var];
            If[t != {},
               AppendTo[res, {t[[1, 1, 2]], t[[1, 2, 2]]}];
               AppendTo[res2, {ReplaceAll[f, t[[1]]], t[[1]]}];
            ];
        ];
      ];
```

```
res2 = Union[res2];     res = Union[res];

(* Graficki prikaz *)
p1 = ListPlot[res,
 PlotStyle -> {PointSize[0.025], Hue[1]}, DisplayFunction -> Identity];
r = Solve[f == 0, var[[2]]][[1, 1, 2]];

mx = Max[Table[res[[i, 1]], {i, 1, Length[res]}]];
my = Max[Table[res[[i, 2]], {i, 1, Length[res]}]];
Print[res]; Print[res2];
For[i = 1, i = Length[res], i++,
  n = ReplaceAll[y - r, res2[[i, 2]]];
  cf[i] = Plot[r + n, {x, -mx/10, mx*1.1},
    PlotStyle -> {Thickness[0.007], Hue[1]}, DisplayFunction->Identity];
];

 ShowAnimation[Table[Show[p2, p1, cf[i], AspectRatio->1,
  PlotRange -> {{-mx/10, mx*1.1}, {-my/10, my*1.1}},
  TextStyle -> {FontFamily -> \"Times", FontSize->14},
  AxesLabel -> TraditionalForm /@ var,
  PlotLabel -> "Korak " <> ToString[i]],
  {i, 1, Length[res], 1}]
 ];
 If[res2[[Length[res2], 1]] == res2[[Length[res2] - 1, 1]],
      Print["Optimal solution is given by l*", res[[Length[res]]],
          "+(1-l)*", res[[Length[res] - 1]], ",    0=l=1"],
      Print["Optimal solution is: ", res[[Length[res]]]];
 ];
];
```

5.2 Program MarPlex

Razmotrimo najvažnije detalje implementacije programa MarPlex.

```
AdvModification FreeVariables MainModule, GlobalVariables, Output,
Modification, MPS, SimplexBazFeasible, SimplexBazNotFeasible,
SimplexMaximMinim, SimplexEquation, Input, Presolver Univerzal.
```

Ovde navodimo samo neke bitne procedure, jer je celokupan kôd suviše veliki. Napomenimo samo da modul `MainModule` služi za povezivanje drugih procedura u jedinstvenu celinu. U modulu `Output` se štampaju izlazni podaci a to je vrednost funkcije cilja (ili poruka ako je funkcija cilja neograničena ili ako je problem nedopustiv) kao i vrednosti promenljivih u optimalnoj tački. U modulu `GlobalVariables` deklarisane su globalne promenljive koje se koriste u drugim procedurama. To su:

1. Brojevi `m` i `n` koji predstavljaju dimenzije problema.
2. Matrica `a()()` koja predstavlja Tuckerovu tabelu.
3. Nizovi `nbasicv` i `basicv` koji predstavljaju prvu vrstu i zadnju kolonu matrice Tuckerove tabele.
4. Četiri konstante (`Infinity, Impossibile, Solution, Error`) jedna logička promenljiva `maksim` koja označava da li se radi o problemu maksimizacije ili minimizacije, kao i tačnost `eps`.

Najvažnije konstante i promenljive su definisane (deklarisane) na sledeći način:

```
Option Explicit

Public Const Infinity = 1 Public Const Impossible = 2 Public Const
Solution = 0 Public Const Error = -1

Public a() As Double Public eps As Double Public m As Integer
Public n As Integer Public basicv() As Integer Public nbasicv() As
Integer Public modif As Boolean

Public outk() As Boolean Public outv() As Boolean

Public maxim As Boolean
```

Svi nizovi i matrice su deklarisani bez dimenzija. Dimenzije svakog niza se posle učitavanja podataka postavljaju korišćenjem naredbe `ReDim`. Modul `MPS` služi za čitanje podataka iz MPS fajla. Ostali moduli implementiraju odgovarajuće faze simpleks algoritma. Od njih navodimo samo najbitnije.

Sledi kod procedure `SolveSystem` koja predstavlja implementaciju algoritma **Replace**.

```
Public Sub SolveSystem(pivrow As Integer,pivcol As Integer)
    Dim p As Double, nv As Integer, nk As Integer, h As Integer
        j As Integer, pomint As Integer

  p = a(pivrow, pivcol)
  nv = 0
  nk = 0
  For h=1 To n+1
    If (a(pivrow,h)<>0) And (h<>pivcol) And (outk(h)=True) Then
      nv=nv+1
      v(nv)=h
    End If
  Next h
  For h=1 To m+1
    If (a(h,pivcol)<>0) And (h<>pivrow) And (outv(h)=True) Then
      nk = nk+1
      k(nk) = h
    End If
  Next h
  For h=1 To nk
    For j=1 To nv
      a(k(h),v(j))=a(k(h),v(j))-a(pivrow,v(j))*a(k(h),pivcol)/p
      If Abs(a(k(h),v(j)))<epsil Then a(k(h),v(j))=0
    Next j
  Next h
  For h=1 To nk
    a(k(h),pivcol)=a(k(h),pivcol)/p
    If abs(a(k(h),pivcol)) <epsil Then a(k(h),pivcol)=0
  Next h
  For h=1 To nv
    a(pivrow,v(h))=a(pivrow,v(h))/p
    If Abs(a(pivrow,v(h))) <epsil Then a(pivrow,v(h))=0
  Next h
```

```
  a(pivrow,pivcol)=1/p
  If Abs(a(pivrow,pivcol))<epsil Then a(pivrow,pivcol)=0
  pomint = basicv(pivcol)
  basicv(pivcol) = nbasicv(pivrow)
  nbasicv(pivrow) = pomint
End Sub
```

Ovde su skupovi V i K predstavljeni redom kao nizovi `v` i `k`.
Moduli `SimplexBazNotFeasible`, `Modification`, `AdvModification` predstavljaju redom implementacije algoritma **NoBasicMax**, **ModNoBasicMax** i **AdvModNoBasicMax**.

Kôd funkcije `SimplexMod` je:

```
Public Function SimplexMod(res As Integer) As Double
  Dim i As Integer, j As Integer, p As Integer, from As Integer,
      from1 As Integer, found As Boolean

  from = 1
  frm_Marplex.Status.Text = "Finding first basic solution...."
  Do
    DoEvents
    If work = 1 Then Exit Function
    frm_Marplex.BazNed.Text = val(frm_Marplex.BazNed.Text) + 1
    i = FindI(from)
    If i = -1 Then Exit Do
    from = i
    from1 = 1
    found = False
    Do
      j = FindJ(from1, i)
      If j > 0 Then from1 = j + 1
      If j = -2 Then
        res = Impossible
        Exit Function
      End If
      If j = -1 Then Exit Do
      p = FindPivotRow(j)
      If p = -1 Then
        SolveSystem i, j
        found = True
      ElseIf a(p, n + 1) / a(p, j) >= a(i, n + 1) / a(i, j) Then
        SolveSystem i, j
        found = True
      End If
    Loop Until found
    j = from1 - 1
    If found = False Then
      SolveSystem p, j
    End If
  Loop
  SimplexMod = SolveBasicFeasible(res)
End Function
```

U implementaciji algoritma nisu konstruisani skupovi B i Q već se brojevi i_s i j_p (`i` i `j` u kodu) traže u ciklusu `Do-Loop`. Pri tome se pamte promenljive `from`

i `from1` kao predhodne vrednosti za `i` i `j`. Procedurama `FindI` i `FindJ` traže se brojevi i i j.

Poboljšana modifikacija (algoritam **AdvModNoBasicMax**) je implementirana u funkciji `SimplexAdvMod`. Sledi kod ove funkcije:

```
Public Function SimplexAdvMod(res As Integer) As Double
    Dim bliz As Integer, ii As Integer, i As Integer, j As Integer, k As Integer,
        l As Integer, p As Integer, niter As Integer, pivrow As Integer,
        pivcol As Integer, s() As Double, piv() As Integer, max As Double,
        maxpiv As Double, imaneg As Boolean, negbs As Integer
    frm_Marplex.Status.Text = "Finding first basic solution...."
    niter = 0
    ReDim s(n): ReDim piv(n)
    Do
        If work = 1 Then Exit Function
        DoEvents
        frm_Marplex.BazNed.Text = val(frm_Marplex.BazNed.Text) + 1
        pivrow = -1: maxpiv = 0: pivcol = -1
        For j = 1 To n: s(j) = -1: piv(j) = 0: Next j
        'Pokusavamo da nadjemo negativni pivot
        ii = -1:
        For i = 1 To m
            If a(i, n + 1) < 0 Then
                p = -1: ii = i: imaneg = False
                For j = 1 To n
                    If (a(i, j) < 0) Then
                      If s(j) = -1 Then
                          p = FindPivotRow(i, j)
                          piv(j) = p
                           s(j) = a(p, n + 1) / a(p, j)
                           If (a(p, n + 1) < 0) And (Abs(a(p, j)) > maxpiv) Then
                               pivrow = p
                               pivcol = j
                               maxpiv = a(pivrow, pivcol)
                               Exit For
                           End If
                       Else
                           p = piv(j)
                       End If
                    End If
                Next j
                If p = -1 Then
                    res = Impossible
                    Exit Function
                End If
                If pivrow > 0 Then Exit For
            End If
        Next i
        negbs = 0
        For i = 1 To m
            If a(i, n + 1) < 0 Then negbs = negbs + 1
        Next i
        frm_Marplex.negb.Text = negbs
        If ii = -1 Then Exit Do
        If pivrow > 0 Then
            SolveSystem pivrow, pivcol
```

```
        Else
            'If impossible, we are choosing the positive one
            'and applying anticyclic rules
            'Every time we are choosing last negative b_i
            bliz = 30000
            For j = 1 To n
                If (a(ii, j) < 0) And (nbasicv(j) < bliz) Then
                    bliz = nbasicv(j)
                    pivcol = j
                End If
            Next j
            bliz = 30000
            For l = 1 To m
                If (a(l, n + 1) >= 0) And (a(l, pivcol) > 0) Then
                    If (a(l, n + 1) / a(l, pivcol) = s(pivcol)) _
                    And (basicv(l) < bliz) Then
                        bliz = basicv(l)
                        pivrow = l
                    End If
                End If
            Next l
            SolveSystem pivrow, pivcol
        End If
    Loop
    SimplexAdvMod = SolveBasicFeasible(res)
End Function
```

Promenljiva `negbs` pamti trenutni broj negativnih elemenata b_i. Najpre pokušavamo da nađemo pivot element tako da je uslov Leme 3.2.1 zadovoljen. Ukoliko to nije moguće, biramo pozitivan pivot element i primenjujemo anticiklična pravila. Značenje ostalih promenljivih je isto kao i u prethodnoj proceduri.

Funkcija Simplex koja predstavlja implementaciju algoritma **NoBasicMax** data je sledećim kodom:

```
Public Function Simplex(res As Integer) As Double
    Dim i As Integer, pivcol As Integer, pivrow As Integer, h As Integer
    frm_Marplex.Status.Text = "Finding first basic solution...."
    Do
    If work = 1 Then Exit Function
    DoEvents
    frm_Marplex.BazNed.Text = val(frm_Marplex.BazNed.Text) + 1
    i = FindI
    If i = 0 Then
      Simplex = SolveBasicFeasible(res)
      Exit Do
    End If
    pivcol = FindPivotColNotFeasible(i)
    If pivcol = 0 Then
      res = Impossible
      Exit Do
    End If
    pivrow = FindPivotRowNotFeasible(i, pivcol)
    SolveSystem pivrow, pivcol
  Loop
End Function
```

Uloga pomoćnih procedura `FindI`, `FindPivotColNotFeasible` i `FindPivotRowNotFeasible` je ista kao i kod predhodne funkcije.

Obe ove funkcije posle svog izvršenja pozivaju funkciju `SolveBasicFeasible` kojom se rešava bazično dopustiva Tuckerova tabela.

5.3 Program RevMarPlex

Program RevMarPlex se sastoji od sledećih modula:

`DefaultFunctions Revised Simplex`

U modulu `DefaultFunctions` nalaze se pomoćne funkcije dok su u `RevisedSimplex` modulu funkcije: `SolveBasicFeasible`, `FindBasicNotFeasible`, `FindBasicFeasibleMod` i `RevisedSimplex`. Prve tri funkcije implementiraju odgovarajuće faze revidiranog simpleksa (poslednja implementira modifikaciju) dok je četvrta glavna funkcija i nju korisnik poziva. Njeni parametri su:

- `f_`: Linearna funkcija koju treba optimizovati u simboličkoj formi,
- `A_`: Lista ograničenja u simboličkoj formi,
- `max_`: Logička promenljiva. `True` za maksimizaciju, `False` za minimizaciju.

Funkcija `MakeMatrix` iz modula `DefaultFunctions` vrši konverziju problema iz simboličkog u matrični oblik. Parametri funkcije `FindBasicFeasible` su redom matrica sistema, RHS vektor, kao i indeksi kolona koje čine prvu bazu (prvo bazično rešenje).

Sledi kod funkcije `FindBasicFeasible`.

```
FindBasicFeasible[A_, b_, base1_] :=
    Block[{pom=0,iter=0,bs, bz1, bm, h=0,i=0,j=0,k=0,B={},N={},bz={},ik,row={},
      col={}, nbase={}, base={}, pr=0, pc=0, eps, min=0, negbi, radi},
      {m, n} = Dimensions[A];    base = base1;
      B = PrepareMatrix[A, base];
      For [i = 1, i <= m, i++, nbase = Join[nbase, {i}]];
      nbase = Complement[nbase, base];
      Nb = PrepareMatrix[A, nbase];  eps = 10^-6; bm = {};  radi = True;  k = 0;
      While[True,
        k++;     bs = Sort[base];     bm = Join[bm, {bs}];
        bz = Chop[LinearSolve[Transpose[B], b], eps];
        radi = False; negbi = 0;
        For [h = 1, h <= n, h++,
          If [bz[[h]] < 0,
              radi = True;  i = h; negbi++;
          ];
        ];

        If [Mod[k, 10] == 0,
            Print["Finished ", k, " iterations, ", negbi, " negative b_i-s"]];
        If [radi == False,
            Return[base]
        ];
```

```
      (* Reconstructing Row *)
      ik = Table[0, {n}]; ik[[i]] = 1;
      row = Chop[{LinearSolve[B, ik]}.Transpose[Nb], eps][[1]];

      pc = 0;     bliz = Infinity;
      For [h = 1, h <= m - n, h++,
        If [row[[h]] < 0,
            If [nbase[[h]] < bliz,
              pc = h;          bliz = nbase[[h]];
            ];
            Break[];
        ];
      ];

      If [pc == 0, Return[{}]];

      (*Reconstructing Column*)
      col = Chop[LinearSolve[Transpose[B], A[[nbase[[pc]]]]], eps];

      min = bz[[i]]/row[[pc]];    pr = i;   bliz = 0;
      For [h = 1, h <= n, h++,
        If [(col[[h]] > 0) ,
          If [(bz[[h]]/col[[h]]<min) ||
                   ((bz[[h]]/col[[h]]==min) && (bliz>base[[pr]])),
              min = bz[[h]]/col[[h]];  pr = h; bliz = base[[pr]];
          ];
        ];
      ];
      B[[pr]] = A[[nbase[[pc]]]]; Nb[[pc]] = A[[base[[pr]]]];
      pom = nbase[[pc]];  nbase[[pc]] = base[[pr]];  base[[pr]] = pom;
    ];
    Return[base];
  ];
```

Za rešavanje linearnih sistema jednačina, za rekonstrukciju redova i kolona Takerove tabele koristili smo funkciju `LinearSolve`. Promenljive `B` i `Nb` pamte redom bazičnu i nebazičnu matricu A_B i A_N.

Parametri funkcije `FindBasicFeasibleMod` su isti kao i parametri ranije opisane funkcije `FindBasicFeasible`. Sledi kod funkcije `FindBasicFeasibleMod`.

```
FindBasicFeasibleMod[A_, b_, base1_] :=
    Block[{pom=0, iter=0, bs, bz1, bm, h=0, i=0, j=0, k=0, B={}, N={}, bz={}, ik,
          row={}, col={}, nbase={}, base={}, pr=0, pc=0, eps, min=0, radi, negbi},
      {m, n} = Dimensions[A];     base = base1;
      B = PrepareMatrix[A, base];
      For [i = 1, i <= m, i++, nbase = Join[nbase, {i}]];
      nbase = Complement[nbase, base];
      Nb = PrepareMatrix[A, nbase];
      eps = 10^-6;     bm = {};    radi = True;     k = 0;

      While[True,
        k++;     bs = Sort[base];
        bm = Join[bm, {bs}];     bz = Chop[LinearSolve[Transpose[B], b], eps];
        radi = False;
        For [h = 1, h <= n, h++,
```

```
      If [bz[[h]] < 0,
          radi = True;      i = h;
        ];
      ];
      If [radi == False,  Return[base]      ];

      (* Reconstructing Row *)
      ik = Table[0, {n}]; ik[[i]] = 1;
      row = Chop[{LinearSolve[B, ik]}.Transpose[Nb], eps][[1]];

      pc = 0; negbi = 0;
      bliz = Infinity;
      For [h = 1, h <= m - n, h++,
        If [row[[h]] < 0,
            If [nbase[[h]] < bliz,
              pc = h; negbi++;        bliz = nbase[[h]];
            ];
            Break[];
        ];
      ];

      If [Mod[k, 10] == 0,
        Print["Finished ", k, " iterations, ", negbi, " negative b_i-s"]];

      If [pc == 0, Return[{}]];

      (*Reconstructing Column*)
      col = Chop[LinearSolve[Transpose[B], A[[nbase[[pc]]]]], eps];

      min = bz[[i]]/row[[pc]];
      pr = i;
      bliz = 0;
      For [h = pr + 1, h <= n, h++,
        If [(col[[h]]*bz[[h]] > 0) ,
            If[(bz[[h]]/col[[h]] < min) ||
               ((bz[[h]]/col[[h]] == min) && (bliz > base[[pr]])),
                  min = bz[[h]]/col[[h]]; pr = h; bliz = base[[pr]];
            ];
        ];
      ];
      B[[pr]] = A[[nbase[[pc]]]];    Nb[[pc]] = A[[base[[pr]]]];
      pom = nbase[[pc]];  nbase[[pc]] = base[[pr]];    base[[pr]] = pom;
    ];
    Return[base];
  ];
```

Literatura

[1] E.D. Andersen, J. Gondzio, C. Mészáros and X. Xu *Implementation of interior point methods for large scale linear programming*,Technical report, HEC, Université de Genève, 1996.

[2] Ángel Santos Palomo and Pablo Guerrero García, *Resolviendo una sucesión de sistemas compatibles dispersos*, XXVI Congreso National de Estadística e Investigación Opertiva Úbeda, 6–9 de noviembre de 2001.

[3] Ángel Santos Palomo and Pablo Guerrero García, *Solving a sequence of sparse compatibile Systems*, 19th Biennial Conf. Numerical Analysis, Dundee, Scotland, June 2001.

[4] M.D. Ašić and V.V. Kovačević-Vujčić,, *Ill-conditionedness and interior-point methods,* Univ. Beograd Publ. Elektrotehn. Fak., **11** (2000), 53–58.

[5] D. Bertsimas, J.N. Tsitsiklis, *Introduction to linear optimization*, Athena Scientific, Belmont, Massachusetts, 1997.

[6] A. Ben-Israel and T.N. E. Greville, *Generalized Inverses. Theory and Applications, Second edition*, CMS Books in Mathematics/Ouvrages de Mathmatiques de la SMC, 15. Springer-Verlag, New York, 2003.

[7] M.A. Bhatti, *Practical optimization with MATHEMATICA applications,* Springer Verlag Telos, 2000.

[8] R. Bixby, *Implementing the simplex method; the initial basis*, ORSA Journal on Computing **4** (1992), 267–284.

[9] R.G. Bland, *New finite pivoting rules for the simplex method*, Mathematics of Operations Research **2** (1977), 103–107.

[10] B.D. Bounday, *Basic linear programming*, Edvard Arnold, Baltimore, 1984.

[11] S.L. Campbell and C.D. Meyer, *Generalized inverses of linear transformations*, Pitman, London, San Francisco, Melbourne, 1979.

[12] A. Charnes, *Optimality and degeneracy in linear programming*, Econometrica **20** No.2 (1952)

[13] Richard J. Charon, Tim Traynor, Wang Cheng, *The minimal angles method of Stojkovic and Stanimirovic*, Technical report, University of Windsor, Ontario, Canada 2004.

[14] A.R. Conn, *Linear programming via a nondifferentiable penalty function*, SIAM J. NUMER.ANAL., Vol. **13**, No. 1 (1976), 145–154.

[15] H.W. Corley, Jay Rosenberger, Wei-Chang Yeh and T.K. Sung, *The cosine simplex algorithm*, The International Journal of Advanced Manufacturing Technology **27**, Numbers 9-10, (2006), 1047-1050.

[16] H.W. Corley, Jay Rosenberger and T.K. Sung, *Constraint Optimal Selection Techniques (COSTs) for Nonnegative Linear Programming Problems*, under review in European Journal of Operational Research.

[17] D. Cvetković, M. Čangalović, Đ. Dugošija, V. Kovačević-Vujčić, S. Simić, J. Vuleta, *Kombinatorna optimizacija-Matematička teorija i algoritmi* Društvo operacionih istraživača Jugoslavije DOPIS, Beograd 1996.

[18] Cvetković, D. *Diskretna matematika*, Prosveta, Niš, 1996.

[19] J. Czyzyk, S. Mehrotra and S.J. Wright, *PCx User Guide*, Optimization Thechnology Center, Technical Report 96/01 (1996) 1–21.

[20] G. B. Dantzig, *Programming of Interdependent Activities, Mathematical Model*, Econometrica 17, 1949, 200–211.

[21] G.B. Danzing, *Linear programming and Extensions*, Princeton University Press, Princeton, New Yersy 1963.

[22] A. Dax, *Linear programming via least squares*, Linear Algebra and its Applications **111** (1988), 313–324.

[23] A. Deza, E. Nematollahi, T. Terlaky, *How good are interior point methods? Klee-Minty cubes tighten iteration-complexity bounds.*, AdvOL-Report #2004/20 Advanced Optimization Laboratory, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, December 2004.

[24] J. Egerváry, *Matrixok combinatorius tulájdonsagairol*, Math.Fiz. Lopak, 1931.

[25] J.J. Forrest, D. Goldfarb, *Steepest-edge simplex algorithms for linear programming*, Mathematical Programming 57 (1992), 341–374.

[26] Y.-M. Gao, R.-C. Gan, *The Study of Criteria of Linear Programming with Ineffective Variables*, Journal of Taiyuan University of Technology, Vol.35 No.3 (2004), 371-374.

[27] Y.-M. Gao, R.-C. Gan, *The Study of Criteria of Linear Programming with Ineffective Constraints*, Journal of Taiyuan University of Technology, Vol.35 No.5 (2004), 633-636.

[28] B. Gartner, M. Henk, G.M. Ziegler, *Randomized Simplex Algorithms on Klee-Minty Cubes*, Combinatorica 18 (1998), 349–372.

[29] J. Gondzio, *HOPDM (Version 2.12), A fast LP solver based on a primal-dual interior point method*, European Journal of Operations Research **85**, (1995), 221–225.

[30] Hélcio Vieira Junior, Marcos Pereira Estellita Lins, *An improved intial basis for the simplex algorithm*, Computers and Operations Research Volume 32, Issue 8, (2005), 1983 - 1993.

[31] F. L. Hitchcock, *The distribution of a product from several sources to numerous localitites*, Journal Math. and Phys. 20, 1941, 224–230.

[32] Jian-Feng Hu, *A note on "An improved initial basis for the simplex algorithm"*, Computers and Operations Research 34, (2007), 3397–3401.

[33] J.P. Ignizio, *Linear programming in single-multiple-objective systems,* Englewood Cliffs: Prentice Hall, 1982.

[34] V. Ivanović, *Pravila za proračun potrebnog broja transportnih sredstava*, Vojnoizdavački glasnik, sveska 1-3, 1940, 1–10.

[35] L.V. Kantorovich *Matematicheskie metodi v organizacii i planirovanii proizvodstva*, Izd. LGU, 1939.

[36] N. Karmarkar,*A new polynomial-time algorithm for linear programming*, Combinatorica, 4, 1984, 373-395.

[37] L.G. Khachian, *A Polinomial Algorithm in Linear Programming*, Doklady Akademii Nauk SSSR, Vol. 244, No, 5, 1979, pp. 1093-1096.

[38] V. Klee, G.L. Minty, *How good is the simplex method?*, O. Shisha, ed., Inequalities III, Academic Press, New York, 1972, pp. 159–175.

[39] V. Klee, P. Kleinschmidtm, *The d-step conjecture and its relatives*, Math. Operations Research 12, 1987, pp. 718–755.

[40] M. Kojima, N. Megiddo, T. Noma and A. Yoshise, *A unified approach to interior-point algorithms for linear complementarity problems*, Lecture Notes in Computer Science 538, Springer-Verlag, Berlin, 1991.

[41] T.C.T. Kotiah, D.I. Steinberg, *Occurences of cycling and other phenomena arising in a class of linear programming models*, Communications of the ACM, 20, 1977, pp. 107–112.

[42] V. Kovačević-Vujčić, and M.D. Ašić, *Stabilization of interior-point methods for linear programming*, Computational Optimization and Applications, **14** (1999), 1–16.

[43] S.G. Kulkarni and K.C. Sivakumar, *Applications of generalized inverses to interval linear programs in Hilbert spaces*, Numer. Funct. Anal. and Optimiz. **16**(7 & 8) (1995), 965–973.

[44] S.G. Kulkarni and K.C. Sivakumar, *Explicit solutions of a special class of linear programming problems in Banach spaces*, Acta Sci. Math. (Szeged) **62** (1996), 457–465.

[45] Imran Maqsood, *Development of simulation-and optimization-based decision support methodologies for environmental systems management*, Doctoral thesis University of Regina, August 2004, pp 495

[46] K. Miettinen, *Nonlinear Multiobjective Optimization*, Kluwer Academic Publishers, Boston 1999.

[47] *Microsoft Developer Network (13)*, Microsoft Corporation Inc., 1998.

[48] G.V. Milovanović, *Numerička analiza I deo*, Naučna knjiga, Beograd 1991.

[49] Y. Nesterov and A. Nemirovskii, *Interior point polynomial methods in convex programming*, SIAM, Philadelphia, 1993.

[50] G. Owen, *Game theory*, W.B. Saunders Company, Philadelphia, London, Toronto, 1968.

[51] E. Nering, A. Tucker, *Linear Programs and Related Problems*, Academic Press, New York, 1993.

[52] *www.netlib.org*

[53] P.Q. Pan, *Practical finite pivoting rules for the simplex method*, OR Spektrum 12, (1990), 219–225.

[54] P.Q. Pan, *A simplex - like method with bisection for linear programming*, Optimization 22, (1991), 717–743.

[55] P.Q. Pan, *A variant of the dual pivoting rule in linear programming*, Journal of Information and Optimization Sciences, 15, No 3, (1994), 405–413.

[56] P.Q. Pan, *The most-obtuse-angle row pivot rule for achieving dual feasibility: a computational study*, Europ. J. of Oper.Res. **101** (1997), 167–176.

[57] S. Paulraj, C. Chellappan, T. R. Natesan, *A heuristic approach for identification of redundant constraints in linear programming models*, International Journal of Computer Mathematics, **83** (2006), 675–683.

[58] M.D. Petković, P.S. Stanimirović, N.V. Stojković, *Two modifications of revised simplex metod*, Matematicki Vesnik, 54 (2002), pp. 163–169.

[59] M.D. Petković, P.S. Stanimirović, *Partitioning method for two-variable rational and polynomial matrices*, Mathematica Balcanica vol. 19, 2005, pp. 185–194.

[60] M.D. Petković, P.S. Stanimirović, *Symbolic computation of the Moore-Penrose inverse using partitioning method*, International Journal of Computer Mathematics, 82(March 2005), pp. 355–367.

[61] M.D. Petković, P.S. Stanimirović, *Interpolation algorithm of Leverrier-Faddev type for polynomial matrices*, Numerical Algorithms **42** (2006), 345–361.

[62] M.D. Petković, P.S. Stanimirović, *Interpolation algorithm for computing Drazin inverse of polynomial matrices*, Linear Algebra and its applications, 422 (2007), 526-539.

[63] L.D. Pyle, *The weighted genalized inverse in nonlinear programming - active set selection using a variable-metric generalization of the simplex algorithm* Lecture Notes in Economics and Mathematical System, Austin, Texas, September 13-15, 1977.

[64] L.D. Pyle, *The genalized inverse in linear programming. Basic structure*, SIAM L. appl. Math. **22** (1972), 335–355.

[65] L.D. Pyle and R.E. Cline, *The genalized inverse in linear programming - interior gradient projection methods*, SIAM L. appl. Math. **24** (1973), 511–534.

[66] M. Sakarovitch, *Linear programming,* Springer-Verlag, New York, 1983.

[67] S.K. Sen and A. Ramful, *A direct heuristic algorithm for linear programming*, Proc. Indian Acad. Sci. (Math. Sci.), Vol. 110, No. 1, (2000), 79–101.

[68] Chanin Srisuwannapa, *Coordinate transformations for solving linear programming problems*, Doctoral thesis, Case Western Reserve University, Cleveland, 2007.

[69] W. Stadler (Ed.), *Multicriteria Optimization in Engineering and in the Sciences*, Plenum Press, New York, 1988.

[70] P.S. Stanimirović, N.V. Stojković, M.D. Petković, *Modification and implementation of two phases simplex method*, International Journal of Computer Mathematics, Prihvaćen za štampu.

[71] P.S. Stanimirović, M.D. Petković, *Computation of generalized inverses of polynomial matrices by interpolation*, Appl. Math. Comput., Vol 172/1 (January 2006), pp 508-523.

[72] P.S. Stanimirović, G.V. Milovanović, *Programski paket MATHEMATICA i primene*, Elektronski fakultet u Nišu, Edicija monografije, Niš, 2002.

[73] R. Stanojević, *Linearno Programiranje*, Institut za ekonomiku industrije, Beograd 1966.

[74] N.V. Stojković, *Primal-dual i simpleks metodi za rešavanje problema linearnog programiranja*, Doktorska disertacija, PMF Niš, 2001.

[75] N.V. Stojković and P.S. Stanimirović, *On elimination of excessive constraints in linear programming*, SYMOPIS, (1999), 207–210.

[76] N.V. Stojković and P.S. Stanimirović, *Two direct methods in linear programming*, Europ. J. of Oper.Res. **131(2)** (2001), 417–439.

[77] N.V. Stojković and P.S. Stanimirović, *A method for solving some classes of linear programming*, Proceedings of XIII Conference on Applied Mathematics, Igalo (1998), 153–160.

[78] N.V. Stojković, P.S. Stanimirović, M.D. Petković, *Several Modifications of Simplex Method*, Filomat, (2003), 169–176.

[79] J. Strayer, *Linear Programming and Its Applications*, Springer-Verlag 1989.

[80] M. Tasić, P.S. Stanimirović, I.P. Stanimirović, M.D. Petković, N.V. Stojković, *Some useful MATHEMATICA teaching examples*, Facta Universitatis(Niš) Series Electronics and Energetics, vol. 18, No. 2, August 2005, pp. 329-344.

[81] Ed. T. Terlaky, *Interior point methods of mathematical programming*, Kluwer Acad. Publ., Dodrecht, Boston, London , 1996.

[82] R. J. Vanderbei,*Linear Programming: Foundations and Extensions*, Princeton, N.J. 2001.

[83] R. J. Vanderbei,*LOQO: An interior-point code for quadratic programming, Technical Report SOR-94-15*, Department of Civil Engineering and Operations Research, Princeton University, Princeton, N.J. 1994.

[84] S. Vukadinović, S. Cvejić *Matematičko programiranje,* Univerzitet u Prištini, Priština, 1996.

[85] C.M.Wang, *Comments on Two Direct Methods in Linear Programming*, Master Thesis, Windsor, Ontario, Canada, 2004.

[86] Wang Cheng, *Comments on Two Direct Methods in Linear Programming*, under review in European Journal of Operational Research.

[87] G.Wang, Y.Wei and S. Qiao, *Generalized Inverses: Theory and Computations*, Science Press, Beijing, 2004.

[88] L. Wei, *A Note on Two Direct Methods in Linear Programming*, Europ. J. of Oper.Res. 158, (2004), 262–265.

[89] D.J. White, *A bibliography on the applications of mathematical programming multiple-objective methods*, Journal of the Operational Research Society, 41(8), 1990, pp. 669–691.

[90] S. Wolfram, *Mathematica: a system for doing mathematics by computer*, Addison-Wesley Publishing Co, Redwood City, California, 1991.

[91] S. Wolfram, *Mathematica Book, Version 3.0*, Wolfram Media and Cambridge University Press, 1996.

[92] S. Wolfram, *The Mathematica Book, 4th ed.*, Wolfram Media/Cambridge University Press, 1999.

[93] S.J Wright, *Primal-dual interior-point methods*, SIAM, Philadelphia, 1997.

[94] Y. Zang, *User's guide to LIPSOL*, Department of Mathematics and Statistics, University of Maryland, Baltimore County, Baltimore, Md., July 1995.

Glava 2

Primal-dual metodi unutrašnje tačke

1 Primal-dual algoritmi

Simpleks metod je od 1947. godine bio standardna procedura za rešavanje problema linearnog programiranja. Algoritam je u praktičnim primenama potvrdio svoju efikasnost tako da je Dantzig u [15] tvrdio da broj iteracija retko prelazi dvostruku dimenziju problema što se i eksperimentalno verifikuje na slučajno generisanim problemima, a što se potvrđuje sve do danas na realnim problemima. Uporedo sa razvojem linearnog programiranja, u oblasti računarstva razvija se teorija računske složenosti u okviru koje se definiše pojam polinomijalnog algoritma. Intuitivno, algoritam je polinomijalan ukoliko se broj elementarnih računskih operacija potrebnih za rešavanje proizvoljnog problema iz date klase može ograničiti polinomom po dimenziji problema. Tek 1972. godine je pokazano da postoje primeri [54] na kojima simpleks metod nije polinomijalan. U pomenutom radu je dat jednostavan primer kod koga je dopustivi skup deformisana n-dimenziona kocka sa 2^n temena za čije rešavanje simpleks metodu treba $2^n - 1$ iteracija. Prvi polinomijalan algoritam (metod elipsoida) za linearno programiranje je 1979. godine dao Kačijan [53] i pokazao da on rešava problem linearnog programiranja sa $O(nL)$ aritmetičkih operacija, gde je L broj bitova potrebnih za zapis svih parametara linearnog programa. Međutim, pokazalo se da metod elipsoida u praksi nije konkurentan simpleks metodu jer "često" dostiže gornju granicu složenosti dok to kod simpleks metoda nije slučaj. Štaviše, pokazalo se da pod izvesnim pretpostavkama o raspodeli ulaznih podataka simpleks metod jeste polinomijalan [12]. Prvi praktično primenljiv polinomijalan algoritam je 1984. godine dao Karmarkar [51]. Posle toga je razvoj unutrašnjih metopda veoma ubrzan. Pojavila se čitava familija unutrašnjih metoda, razvijeni su programski paketi koji su pokazali bolje performanse u odnosu na simpleks metod, naročito za probleme velikih dimenzija. Praksa je pokazala da se porastom dimenzije problema povećava superiornost unutrašnjih metoda. Opšte je

mišljenje da će praktičnoj upotrebi ostati oba pristupa linearnom programiranju. Pri rešavanju problema velikih dimenzija, u ranijim iteracijama će se primenjivati unutrašnje metode, a zatim će se postupak završiti simpleks metodom.

Detaljan osvrt na odnos simpleks metoda i unutrašnjih metoda, kao i pravci njihovog daljeg razvoja može se naći u radu Vere Kovačević-Vujčić [63]. Ovde treba napomenuti da su prve ideje o metodima unutrašnje tačke dali u svojim radovima von Neumann 1947. [77], Hoffman 1953. [49], Frish 1955. [35] i Dikin 1967. [18]. Međutim, računski problemi, numerička nestabilnost iteracija i loši eksperimentalni razultati su doveli do stanovišta da algoritmi unutrašnje tačke nisu konkurentni sa simpleks metodom. Polinomijalnost i računska složenost metoda unutrašnje tačke je razmatrana u radovima [31, 32, 37, 40, 50, 55, 76, 120]. Karmarkarov algoritam i metod elipsoida su detaljno opisani na našem jeziku u monografiji [13], tako da se na njima nećemo zadražavati. Takođe je u radu [63] dat opis različitih pristupa metodu unutrašnje tačke. Prema geometrijskoj interpretaciji, postojeći metodi unutrašnje tačke su klasifikovani u tri grupe:

• *Projektivni metodi* (Karmarkar [51], Anstreicher [4], De Chellinck i Vial [16], Todd i Ye [104], Ye [121], Yamashita [116], Gonzaga [41, 43]).

• *Afini metodi* (Dikin [18], Barnes [7], Gonzaga [42], Vanderbei, Monma i Morton [75], Monteiro, Adler i Resende [74], Ye [119]).

• *Metodi koje slede centralnu putanju* (Renegar [80], Gonzaga [44, 45], Roos i Vial [82, 83], Den Hertog, Roos i Terlaky [17], Renegar i Shub [81], Monteiro i Adler [73], Kojima, Mizuno i Yoshise [59]).

U zavisnosti od toga da li rešavaju primalni problem, dualni problem ili simultano oba problema, unutrašnji metodi su u pomenutom radu [63] klasifikovani na sledeći način:

• *Primalni metodi* (Karmarkar [51], Anstreicher [4], De Ghellinck i Vial [16], Ye [117], Gonzaga [44, 45], Den Hertog, Roos i Terlaky [17], Dikin [18], Barnes [7]), Vanderbei, Meketon i Freedman [106]).

• *Dualni metodi* (Yamashita [116], Gonzaga [39], Monma i Morton [75], Renegar [80], Renegar i Shub [81]).

• *Primal-dual metodi* (Monteiro i Adler [73], Mizuno, Tod i Ye [72], Gonzaga [42], Ye [119], Tanabe [102], Todd i Ye [104], Freund [28], Gonzaga i Todd [46], Kojima, Mizuno i Yoshise [56]).

Kako se primal-dual metod unutrašnje tačke u praksi potvrdio kao jedan od najefikasnijih, u ovoj glavi razmatramo teorijske osnove i različite algoritme primal-dual metoda, njihove modifikacije i implementacije. Osnovne ideje primal-dual metoda su razvijene između 1987. i 1991. godine. Za početak teorije primal-dual algoritama se smatra rad Megidda [66] iz 1987. godine. Motivisani tim radom, 1987. godine Kojima, Mizuno i Yoshise [59] razvijaju prvi polinomijalni primal-dual algoritam dugog koraka čija je kompleksnost $O(n \log 1/\varepsilon)$. Odmah zatim isti autori razvijaju primal-dual algoritam kratkog koraka [60] i poboljšavaju kompleksnost na $O(\sqrt{n} \log 1/\varepsilon)$. Slične rezultate dobijaju Monteiro i Adler [73], Mizuno, Tod i Ye [72], Gonzaga [42], Ye [119], Tanabe [102], Todd i Ye [104], Freund [28], Gonzaga

i Todd [46]. Većina savremenih programskih paketa koristi primal-dual algoritam koji je razvio Mehrotra 1992. godine u radu [70] na osnovu radova Megidda [66], Monteira, Adlera i Resendea [74] i Lustiga, Marstena i Shannoa [65].

Napomenimo da se primena primal-dual metoda može proširiti na monotone linearne komplementarne probleme, konveksno programiranje i semidefinitno programiranje, što je razmatrano u [25, 33, 68, 106].

1.1 Teorijske osnove primal-dual metoda

U analizi algoritama primal-dual metoda unutrašnje tačke pogodno je problem linearnog programiranja posmatrati u standardnom obliku:

$$\min c^T x \text{ u odnosu na uslove } Ax = b, \quad x \geq 0, \tag{1.1.1}$$

gde su $c, x \in R^n$, $b \in R^m$, i A je realna matrica tipa $m \times n$.

Dualni problem za (1.1.1) je

$$\max b^T y \text{ u odnosu na uslove } A^T y + s = c, \quad s \geq 0, \tag{1.1.2}$$

gde je $y \in R^m$ i $s \in R^n$. Iz teorije dualnosti je poznato da za dopustive vektore x i (y, s) problema (1.1.1) i (1.1.2) respektivno važi

$$b^T y \leq c^T x, \tag{1.1.3}$$

tj. ciljna funkcija duala je donje ograničenje ciljne funkcije primalnog problema i primal je gornje ograničenje za dual. Jednakost $b^T y^* = c^T x^*$ važi kada je x^* rešenje za (1.1.1) i (y^*, s^*) rešenje za (1.1.2) i zajedničku optimalnu vrednost označavamo sa z^*.

Centralno mesto u primeni metoda unutrašnje tačke imaju Karush-Kuhn-Tucker uslovi optimalnosti. Primena ovih uslova na probleme (1.1.1) i (1.1.2) daje sledeći rezultat:

Teorema 1.1.1 *Vektor $x^* \in R^n$ je rešenje za* (1.1.1) *ako i samo ako postoje vektori $s^* \in R^n$ i $y^* \in R^m$ tako da važi:*

$$\begin{aligned} A^T y^* + s^* &= c, && (1.1.4a)\\ Ax^* &= b, && (1.1.4b)\\ x_i^* s_i^* &= 0, i = 1, \ldots, n, && (1.1.4c)\\ (x^*, s^*) &\geq 0. && (1.1.4d) \end{aligned} \tag{1.1.4}$$

Ako tačka (x, y, s) zadovoljava uslove a, b i d iz prethodne teoreme, tada je $0 \leq s^T x = (c - A^T y)^T x = c^T x - y^T (Ax) = c^T x - b^T y$ odakle sledi (1.1.3). Uslov (1.1.4c) povlači da za svaki indeks $i = 1, \ldots, n$, jedna od vrednosti x_i^* ili s_i^* mora biti jednaka nuli. Taj uslov se naziva *uslov komplementarnosti*, jer povlači da se nenula komponente vektora x i s javljaju na komplementarnim koordinatama.

Važi i dualna teorema:

Teorema 1.1.2 *Vektor* $(y^*, s^*) \in R^m \times R^n$ *je rešenje za* (1.1.2) *ako i samo ako postoji vektor* $x^* \in R^n$ *tako da važe uslovi* (1.1.4).

Direktna posledica ovih teorema je da vektor (x^*, y^*, s^*) rešava sistem (1.1.4) ako i samo je x^* rešenje primalnog problema (1.1.1) i (y^*, s^*) je rešenje dualnog problema. Vektor (x^*, y^*, s^*) se naziva *primal-dual rešenje.* Sa Ω_P i Ω_D označimo skupove optimalnih rešenja primalnog i dualnog problema respektivno:

$$\begin{aligned} \Omega_P &= \{x^* | \, x^* \text{ je rešenje za (1.1.1)}\}, \\ \Omega_D &= \{(y^*, s^*) | \, (y^*, s^*) \text{ je rešenje za (1.1.2)}\}. \end{aligned}$$

Skup rešenja primal-dualnog problema je Dekartov proizvod

$$\Omega = \Omega_P \times \Omega_D = \{(x, y, s) | \text{ važi (1.1.4).}\}$$

Za svako rešenje (x^*, y^*, s^*), važi $x_j^* = 0$ i/ili $s_j^* = 0$ za svako $j = 1, \ldots, n$. Neka je

$$\begin{aligned} \mathcal{B} &= \{j \in \{1, \ldots, n\} \mid x_j^* \neq 0 \text{ za neko } x^* \in \Omega_P\}, \\ \mathcal{N} &= \{j \in \{1, \ldots, n\} \mid s_j^* \neq 0 \text{ za neko } (y^*, s^*) \in \Omega_D\} \end{aligned}$$

Jasno, $\mathcal{B} \cap \mathcal{N} = \emptyset$. Sledeći rezultat je poznat kao Goldman-Tuckerova teorema [36].

Teorema 1.1.3 *Postoji bar jedno primalno rešenje* $x^* \in \Omega_P$ *i jedno dualno rešenje* $(y^*, s^*) \in \Omega_D$ *tako da je* $x^* + s^* > 0$.

Primal-dual rešenje (x, y, s) sa osobinom $x + s > 0$ je *strogo komplementarno rešenje.*

Osnova primal-dual metoda je modifikacija Newtonove metode primenjena na prva tri uslova iz (1.1.4), pri čemu se traženi pravac i korak svake iteracije određuju tako da nejednakost $(x, s) \geq 0$ važi strogo u svakoj iteraciji. Ta stroga nejednakost upravo i dovodi do problema u pravljenju, analizi i primeni algoritama primal-dual metoda.

Uslove optimalnosti (1.1.4) razmatramo kao preslikavanje F iz R^{2n+m} u R^{2n+m}:

$$\begin{aligned} F(x, y, s) = & \begin{bmatrix} A^T y + s - c \\ Ax - b \\ XSe \end{bmatrix} = 0, && (1.1.5a) \\ & (x, s) \geq 0, && (1.1.5b) \end{aligned} \tag{1.1.5}$$

gde je

$$X = \text{diag}(x_1, \ldots, x_n), \quad S = \text{diag}(s_1, \ldots, s_n) \quad \text{i} \quad e = (1, \ldots, 1)^T.$$

Svi primal-dual metodi generišu iteracije (x^k, y^k, s^k) koje zadovoljavaju ograničenje (1.1.5b) striktno, tj. $x^k > 0$ i $s^k > 0$. Na osnovu te osobine se metode koje razmatramo i zovu *metode unutašnje tačke.* Većina metoda unutrašnje tačke zahteva da

iteracije budu *strogo dopustive*, tj. da (x^k, y^k, s^k) zadovoljava linearne jednačine iz primalnog i dualnog problema. Primal dual *dopustiv skup* $\mathcal{F}$ i *strogo dopustiv skup* $\mathcal{F}^0$ definisani su sa

$$\begin{aligned}\mathcal{F} = &\quad \{(x,y,s)|\, Ax = b, A^T y + s = c, (x,s) \geq 0\},\\ \mathcal{F}^0 = &\quad \{(x,y,s)|\, Ax = b, A^T y + s = c, (x,s) > 0\}.\end{aligned}$$

Rešavajući sistem Newtonovih linearnih jednačina

$$J(x,y,s)\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta s\end{bmatrix} = -F(x,y,s),$$

gde je J Jakobijan preslikavanja F, dobijamo traženi pravac $(\Delta x, \Delta y, \Delta s)$. Ako je $(x,y,s) \in \mathcal{F}^0$, Newtonove jednačine postaju

$$\begin{bmatrix}0 & A^T & I\\ A & 0 & 0\\ S & 0 & X\end{bmatrix}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta s\end{bmatrix} = \begin{bmatrix}0\\ 0\\ -XSe\end{bmatrix}. \tag{1.1.6}$$

Pun korak u tom pravcu obično nije dopustiv zbog uslova $(x,s) \geq 0$. Sledeću iteraciju određujemo sa

$$(x,y,s) + \alpha(\Delta x, \Delta y, \Delta s)$$

gde je $\alpha \in (0,1]$ određeno iz uslova pozitivnosti. Nažalost, vrednost parametra α je najčešće vrlo mala ($\alpha \ll 1$), tako da Newtonov pravac ne dozvoljava velike pomeraje ka rešenju. Primal-dual metode modifikuju Newtonovu metodu na dva načina:

1. Podešavaju traženi pravac ka unutrašnjosti nenegativnog kvadranta $(x,s) > 0$ tako je moguć duži korak duž dobijenog pravca pre nego što neka od komponenti (x,s) postane negativna.
2. Ne dozvoljavaju preveliko "približavanje" komponenti (x,s) ka granici nenegativnog kvadranta, jer se u tom slučaju dobija pravac duž koga je moguć samo mali pomeraj ka rešenju.

Centralna putanja

Centralna putanja $\mathcal{C}$ sastoji od strogo dopustivih tačaka i igra važnu ulogu u teoriji primal-dual algoritama. Karakteriše se KKT uslovima u kojima se umesto uslova komplementarnosti (1.1.4c) nameće uslov da proizvodi $x_i s_i$ imaju istu vrednost $\tau > 0$ za svako i, tj. $(x_\tau, y_\tau, s_\tau) \in \mathcal{C}$ ako je rešenje sistema:

$$\begin{aligned}A^T y + s &= c,\\ Ax &= b,\\ x_i s_i &= \tau, \quad i = 1, \ldots, n,\\ (x,s) &\geq 0.\end{aligned}$$

Analogna definicija putanje $\mathcal{C}$ data je sa

$$F(x_\tau, y_\tau, s_\tau) = \begin{bmatrix} 0 \\ 0 \\ \tau e \end{bmatrix}, \quad (x_\tau, s_\tau) > 0.$$

Pokazuje se da je (x_τ, y_τ, s_τ) definisano na jedinstven način za svako $\tau > 0$ ako i samo ako je $\mathcal{F}^0 \neq \emptyset$, tj. cela putnja $\mathcal{C}$ je dobro definisana. Za dokaz egzistencije centralne putanje koristi se sledeće tvrđenje:

Lema 1.1.1 *Pretpostavimo da je $\mathcal{F}^0 \neq \emptyset$. Tada je za svako $K \geq 0$ skup*

$$\{(x,s) | \, (x,y,s) \in \mathcal{F} \text{ za neko } y, \ i \ x^T s \leq K\}$$

ograničen.

Kada τ teži nuli, $\mathcal{C}$ konvergira ka primal-dual rešenju linearnog programa i to preko tačaka u kojima su proizvodi $x_i s_i$ strogo pozitivani i teže ka nuli istom brzinom. Većina primal-dual metoda umesto uobičajenog Newtonovog koraka za F uzima Newtonov korak ka tačkama na $\mathcal{C}$. To se postiže uvođenjem *parametra centriranja* $\sigma \in [0,1]$ i *mere dualnosti* μ definisane sa

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i s_i = \frac{1}{n} x^T s,$$

koja određuje srednju vrednost proizvoda $x_i s_i$. Jednačine za određivanje koraka iteracije su

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -XSe + \sigma \mu e \end{bmatrix}. \tag{1.1.7}$$

Sada je $(\Delta x, \Delta y, \Delta s)$ Newtonov korak ka tački $(x_{\sigma\mu}, y_{\sigma\mu}, s_{\sigma\mu}) \in \mathcal{C}$, u kojoj su proizvodi $x_i s_i$ jednaki $\sigma\mu$.

Za $\sigma = 1$, jednačine (1.1.7) definišu *centralni pravac,* Newtonov korak prema tački $(x_\mu, y_\mu, s_\mu) \in \mathcal{C}$, u kojoj su proizvodi $x_i s_i$ jednaki μ. Centralni pravac je obično jako usmeren ka unutrašnjosti nenegativnog kvadranta i dovodi do malog smanjenja parametra μ. Međutim, u sledećem koraku je tada moguće uzeti relativno duži korak iteracije. Druga ekstremna vrednost za $\sigma = 0$ daje standardni Newtonov korak (1.1.6) koji se naziva *afini pravac.* Većina algoritama koristi srednju vrednost parametra $\sigma \in (0,1)$, tako da istovremeno redukuje vrednost mere dualnosti μ i bira pravac ka centralnoj putanji.

1.2 Osnovna šema primal-dual algoritama

Osnovni primal-dual algoritam je opisan u [112].

Osnovni primal-dual algoritam

Data je tačka $(x^0, y^0, s^0) \in \mathcal{F}^0$

for $k = 0, 1, 2, \ldots$

solve

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix},$$

gde je $\sigma_k \in [0, 1]$ i $\mu_k = (x^k)^T s^k / n$;

postaviti

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + \alpha_k (\Delta x^k, \Delta y^k, \Delta s^k),$$

birajući α_k tako da je $(x^{k+1}, s^{k+1}) > 0$.

end(for).

Većina algoritama za početnu tačku zahteva strogo dopustivu tačku $(x^0, y^0, s^0) \in \mathcal{F}^0$ koja je takođe i u nekoj okolini centralne putanje $\mathcal{C}$. Međutim, često nije lako odrediti takvu tačku. Postoje i problemi kod kojih takva tačka uopšte ne postoji. U problemu

$$\min 2x_1 + x_2 \quad \text{u odnosu na } x_1 + x_2 + x_3 = 5,\ x_1 + x_3 = 5,\ x \geq 0,$$

je primal dopustiv skup $\{(\beta, 0, 5 - \beta) |\, \beta \in [0, 5]\}$. Kako je $x_2 = 0$, strogo dopustiv skup $\mathcal{F}^0$ je prazan. U opštem slučaju, linearni problemi koji se dobijaju svođenjem opšteg problema na standardni oblik često nemaju strogo dopustive tačke. Jedan od načina za prevazilaženje te teškoće je potapanje datog linearnog problema u nešto veći problem za koji je skup $\mathcal{F}^0$ neprazan.

Poznat je algoritam koji ne zahteva da početna tačka bude strogo dopustiva već se samo traži da komponente x i s budu strogo pozitivne. U opštem slučaju sve iteracije (x^k, y^k, s^k) generisane algoritmom su nedopustive, ali je granična tačka dopustiva i optimalna. Algoritam je moguće primeniti i kada je $\mathcal{F}^0 = \emptyset$. Ovakav pristup su dali Kojima [58], Zhang [122], Potra [79] i Wright [112].

Reziduume sistema jednačina

$$r_b - Ax - b, \quad r_c = A^T y + s - c$$

za $k-$tu iteraciju označimo sa r_b^k i r_c^k. Okolinu $\mathcal{N}_{-\infty}(\gamma)$ koja sadrži i nedopustive tačke definišmo sa

$$\begin{aligned} \mathcal{N}_{-\infty}(\gamma, \beta) = \quad & \{(x, y, s) |\, |(r_b, r_c)| \leq [|(r_b^0, r_c^0)|/\mu_0]\beta\mu, \\ & (x, s) > 0, x_i s_i \geq \gamma\mu,\ i = 1, \ldots, n\} \end{aligned}$$

gde su $\gamma \in (0, 1)$ i $\beta \geq 1$ dati parametri i vrednosti (r_b^0, r_c^0) i μ_0 izračunate u početnoj tački (x^0, y^0, s^0). Primetimo da je neophodan uslov

$$\beta \geq 1$$

da bi osigurali da početna tačka (x^0, y^0, s^0) pripada $\mathcal{N}_{-\infty}(\gamma, \beta)$.

Sada navodimo opšti primal-dual algoritam za nedopustivu tačku iz [112].

Algoritam NPD

Dato je $\gamma, \beta, \sigma_{\min}, \sigma_{\max}$, pri čemu je $\gamma \in (0,1)$, $\beta \geq 1$, i $0 < \sigma_{\min} < \sigma_{\max} \leq 0.5$;
izabrati (x^0, y^0, s^0) tako da je $(x^0, s^0) > 0$;

for $k = 0, 1, 2, \ldots$

izabrati $\sigma_k \in [\sigma_{\min}, \sigma_{\max}]$ i rešiti

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} -r_c^k \\ -r_b^k \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix},$$

izabrati za α_k najveću vrednost od $\alpha \in [0,1]$, tako da je

$$(x^k(\alpha), y^k(\alpha), s^k(\alpha)) \in \mathcal{N}_{-\infty}(\gamma, \beta)$$

i da važi sledeći uslov Armija:

$$\mu_k(\alpha) \leq (1 - 0.01\alpha)\mu_k.$$

postaviti

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k(\alpha_k), y^k(\alpha_k), s^k(\alpha_k));$$

end (for).

Neka je ν_k definisano sa

$$\nu_k = \prod_{j=0}^{k-1}(1 - \alpha_j) \quad (\nu_0 = 1).$$

Kako su prve dve komponente funkcije F linearne, sledi

$$(r_b^k, r_c^k) = (1 - \alpha_{k-1})(r_b^{k-1}, r_c^{k-1}) = \cdots = \nu_k(r_b^0, r_c^0),$$

i kako je $(x^k, y^k, s^k) \in \mathcal{N}_{-\infty}(\gamma, \beta)$, dobijamo

$$\nu_k |(r_b^0, r_c^0)|/\mu_k = |(r_b^k, r_c^k)|/\mu_k \leq \beta |(r_b^0, r_c^0)|/\mu_0.$$

Ako pretpostavimo da je $(r_b^0, r_c^0) \neq 0$, tada sledi

$$\nu_k \leq \beta \mu_k / \mu_0.$$

U dopustivom slučaju je $(r_b^0, r_c^0) = 0$ i sve iteracije (x^k, y^k, s^k) su strogo dopustive i tada se Algoritam NPD svodi algoritam za dopustivu tačku. U nastavku pretpostavljamo da je početna tačka (x^0, y^0, s^0) nedopustiva.

Sledeća teorema daje opšti rezultat o kompleksnosti metoda koji slede centralnu putanju.

Teorema 1.2.1 *Neka je* $\varepsilon \in (0,1)$ *dato. Pretpostavimo da algoritam generiše niz iteracija koje zadovoljavaju*

$$\mu_{k+1} \leq \left(1 - \frac{\delta}{n^{\omega}}\right)\mu_k, \quad k = 0, 1, 2, \ldots,$$

za neke pozitivne konstante δ *i* ω*. Pretpostavimo da početna tačka* (x^0, y^0, s^0) *zadovoljava*

$$\mu_0 \leq 1/\kappa$$

za meku pozitivnu konstantu κ*. Tada postoji indeks* K *pri čemu je*

$$K = O(n^{\omega} |\log \varepsilon|)$$

tako da važi

$$\mu_k \leq \varepsilon \quad \textit{za svako } k \geq K.$$

Dakle, ako smanjenje μ u svakoj iteraciji zavisi od n na neki način i ako početna mera dualnosti nije previše velika, algoritam ima polinomijalnu kompleksnost.

1.3 Potencijalno-redukcioni metodi unutrašnje tačke

Karmarkarov algoritam [51] je koristio logaritamsku potencijalnu funkciju oblika

$$\rho \log(c^T x - Z) - \sum_{i=1}^{n} \log x_i,$$

gde je $\rho = n + 1$ i Z donja granica optimalne vrednosti ciljne funkcije. Na razvoj potencijal-redukcionih algoritama posle 1988. godine, važan uticaj ima Tanabe-Todd-Ye [102, 104] potencijalna funkcija definisana sa

$$\Phi_\rho(x, s) = \rho \log x^T s - \sum_{i=1}^{n} \log x_i s_i$$

za neki parametar $\rho > n$. Opisaćemo algoritam koji su dali Kojima, Mizuno i Yoshise [60]. Taj algoritam ima najbolju poznatu granicu kompleksnosti jer zahteva $O(\sqrt{n}|\log \varepsilon|)$ iteracija da bi smanjio razliku između primalnog i dualnog rešenja ispod datog praga $\varepsilon > 0$. Algoritam PR je specijalan slučaj osnovnog primal-dual algoritma. Karakteriše ga izbor konstantne vrednosti $\sigma_k = n/\rho$ za parametar centriranja. Dužina koraka α_k se bira tako da minimizira $\Phi_\rho(\cdot)$ duž dobijenog pravca. Gornja granica dužine koraka od tačke $(x, y, s) \in \mathcal{F}^0$ duž izračunatog pravca $(\Delta x, \Delta y, \Delta s)$ je data sa

$$\alpha_{\max} = \sup\{\alpha \in [0,1] | \, (x, s) + \alpha(\Delta x, \Delta s) \geq 0\}.$$

Algoritam PR
Dato je $\rho > n$ **i tačka** $(x^0, y^0, s^0) \in \mathcal{F}^0$

for $k = 0, 1, 2, \ldots$

staviti $\sigma_k = n/\rho$ i rešiti

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix}, \tag{1.3.1}$$

gde je $\mu_k = (x^k)^T s^k/n$, da bi dobili $(\Delta x^k, \Delta y^k, \Delta s^k)$.

Izračunati $\alpha_{\max}$ iz (1.3.1) i izabrati dužinu koraka α_k iz

$$\alpha_k = \arg \min_{\alpha \in (0, \alpha_{max})} \Phi_\rho(x^k + \alpha \Delta x^k, s^k + \alpha \Delta s^k);$$

postaviti

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + \alpha_k(\Delta x^k, \Delta y^k, \Delta s^k);$$

end (for).

Prvo ćemo pokazati da kada $\Phi_\rho(x^k, s^k) \to -\infty$ tada $\mu_k \to 0$.

Lema 1.3.1 *Važe sledeće nejednakosti:*

(i) $\log(1 + t) \leq t$ *za svako* $t > -1$ *pri čemu jednakost važi samo za* $t = 0$.

(ii) Za svako $t \in R^n$ *za koje je* $\|t\|_\infty \leq \tau < 1$, *važi*

$$-\sum_{i=1}^{n} \log(1 + t_i) \leq -e^t t + \frac{\|t\|^2}{2(1 - \tau)}.$$

Predstavimo potencijalnu funkciju u obliku

$$\begin{aligned} \Phi_\rho(x, s) &= (\rho - n) \log x^T s + \Phi_n(x, s) \\ &= (\rho - n) \log x^T s - \sum_{i=1}^{n} \log \frac{x_i s_i}{x^T s/n} + n \log n. \end{aligned}$$

Sledeća lema pokazuje da je funkcija Φ_n ograničena odozdo.

Lema 1.3.2 *Za* $(x, s) > 0$ *važi*

$$\Phi_n(x, s) \geq n \log n,$$

pri čemu jednakost važi ako i samo ako je $XSe = (x^T s/n)e = \mu e$.

U sledećoj lemi se dokazuje ključni odnos između Φ_ρ i μ.

Lema 1.3.3 *(i) Funkcija* Φ_ρ *je neograničena odozdo na svom domenu.*

(ii) Za svako $(x, y, s) \in \mathcal{F}^0$ *važi*

$$\mu \leq \exp(\Phi_\rho(x, s)/(\rho - n)),$$

gde je $\mu = x^T s/n$.

Iz Leme 1.3.3(ii) sledi da ako generišemo niz iteracija $(x^k, y^k, s^k) \in \mathcal{F}^0$ za koje $\Phi_\rho(x^k, s^k) \to -\infty$, tada $\mu_k \to 0$. Pokazaćemo u nastavku da se Φ_ρ smanjuje za fiksiranu vrednost $\delta > 0$, nezavisno od n, u svakom koraku Algoritma PR, tj.

$$\Phi_\rho(x^{k+1}, s^{k+1}) \leq \Phi_\rho(x^k, s^k) - \delta \quad \text{za svako } k = 0, 1, 2, \ldots \tag{1.3.2}$$

Sledeća teorema daje opšti rezultat za kompleksnost algoritama koji postižu takvo smanjenje potencijalne funkcije.

Teorema 1.3.1 *Neka je data tačka $(x^0, y^0, s^0) \in \mathcal{F}^0$ i pretpostavimo da algoritam generiše niz $(x^k, y^k, s^k) \in \mathcal{F}^0$ koji zadovoljava* (1.3.2) *za neko $\delta > 0$. Tada za svako $\varepsilon \in (0, 1)$ postoji broj K definisan sa*

$$K = \left[\frac{\Phi_\rho(x^0, s^0)}{\delta} + \frac{\rho - n}{\delta} |\log \varepsilon| \right]$$

tako da je

$$\mu_k \leq \varepsilon \quad \text{za svako } k \geq K.$$

U nastavku pokazujemo egzistenciju kvadratne funkcije $q(\alpha)$ koja je gornje ograničenje za Φ_ρ i zavisi od dužine koraka α duž traženog pravca. Za bilo koji pravac $(\Delta x, \Delta y, \Delta s)$ važi

$$A\Delta x = 0, \quad A^T \Delta y + \Delta s = 0. \tag{1.3.3}$$

Odatle sledi

$$\Delta s^T \Delta x = -\Delta x^T A^T \Delta y = -(A\Delta x)^T \Delta y = 0. \tag{1.3.4}$$

Pretpostavili smo da je $\alpha \in (0, \alpha_{\max})$ ali kvadratna procena važi na kraćem intervalu $(0, \alpha_\tau]$ definisanom sa

$$\alpha_\tau \max(\|X^{-1}\Delta x\|_\infty, \|S^{-1}\Delta s\|_\infty) = \tau,$$

gde je $\tau \in (0, 1)$ konstanta koju definišemo kasnije. Iz (1.3.4) sledi

$$\begin{aligned}
&\Phi_\rho(x + \alpha\Delta x, s + \alpha\Delta s) - \Phi_\rho(x, s) \\
&= \rho \log \frac{(x + \alpha\Delta x)^T (s + \alpha\Delta s)}{x^T s} - \sum_{i=1}^n \log \frac{x_i + \alpha\Delta x_i}{x_i} - \sum_{i=1}^n \log \frac{s_i + \alpha\Delta s_i}{s_i} \\
&= \rho \log \left[1 - \alpha \frac{s^T \Delta x + x^T \Delta s}{x^T s} \right] - \sum_{i=1}^n \log \left[1 + \alpha \frac{\Delta x_i}{x_i} \right] - \sum_{i=1}^n \log \left[1 + \alpha \frac{\Delta s_i}{s_i} \right].
\end{aligned}$$

Primenom Leme 1.3.1 dobijamo

$$\begin{aligned}
&\Phi_\rho(x + \alpha\Delta x, s + \alpha\Delta s) \leq \Phi_\rho(x, s) + \rho\alpha \frac{s^T \Delta x + x^T \Delta s}{x^T s} \\
&-\alpha e^T (X^{-1}\Delta x + S^{-1}\Delta s) + \frac{\alpha^2}{2(1 - \tau)} (\|X^{-1}\Delta x\|^2 + \|S^{-1}\Delta s\|^2) \\
&= \Phi_\rho(x, s) + \alpha g_1 + \frac{1}{2}\alpha^2 g_2
\end{aligned} \tag{1.3.5}$$

za svako $\alpha \in (0, \alpha_\tau]$, gde je

$$\begin{aligned} g_1 &= \rho\frac{s^T\Delta x + x^T\Delta s}{x^T s} - e^T(X^{-1}\Delta x + S^{-1}\Delta s), \\ g_2 &= \frac{1}{(1-\tau)}(\|X^{-1}\Delta x\|^2 + \|S^{-1}\Delta s\|^2). \end{aligned}$$

Definišemo kvadratnu aproksimaciju sa

$$q(\alpha) = \Phi_\rho(x,s) + \alpha g_1 + \frac{1}{2}\alpha^2 g_2,$$

i iz (1.3.5) sledi

$$\Phi_\rho(x + \alpha\Delta x, s + \alpha\Delta s) \le q(\alpha) \quad \forall \alpha \in (0, \alpha_\tau].$$

Posmatramo sada pravac $(\Delta x, \Delta y, \Delta s)$ koji zadovoljava jednačinu

$$S\Delta x + X\Delta s = -XSe + \frac{n}{\rho}\mu e \tag{1.3.6}$$

kao i uslove (1.3.3). Za dato $(x, y, s) \in \mathcal{F}^0$ uvedimo oznake

$$\begin{aligned} V &= (XS)^{1/2}, \quad \upsilon = Ve = [(x_i s_i)^{1/2}]_{i=1}^n, \\ \upsilon_{\min} &= \min_i \upsilon_i, \quad r = -\upsilon + \frac{n}{\rho}\mu V^{-1}e, \end{aligned}$$

i neka je $D = X^{1/2}S^{-1/2}$. Važe sledeće jednakosti

$$\|\upsilon\|^2 = x^T s = n\mu, \quad X = VD, \quad S = VD^{-1}. \tag{1.3.7}$$

Uslove (1.3.6) možemo sada posmatrati u nekom od oblika

$$\begin{aligned} S\Delta x + X\Delta s &= Vr, \\ D^{-1}\Delta x + D\Delta s &= r, \\ X^{-1}\Delta x + S^{-1}\Delta s &= V^{-1}r. \end{aligned}$$

Iz (1.3.4) sledi

$$\|r\|^2 = \|D^{-1}\Delta x\|^2 + \|D\Delta s\|^2, \tag{1.3.8}$$

i stoga je

$$\|D^{-1}\Delta x\| \le \|r\|, \quad \|D\Delta s\| \le \|r\|.$$

Iz (1.3.7) i (1.3.8) sledi

$$\|X^{-1}\Delta x\|^2 + \|S^{-1}\Delta s\|^2 \le \frac{1}{\upsilon_{\min}^2}\|r\|^2,$$

odakle je

$$g_2 \leq \frac{1}{1-\tau}\frac{\|r\|^2}{\upsilon_{\min}^2}\|r\|^2.$$

Za koeficijent g_1 važi

$$g_1 = -\frac{\rho}{n\mu}\|r\|^2.$$

Ako je $r \neq 0$, tada je $g_1 < 0$ i kako je $g_2 > 0$, to je grafik funkcije q obrnuta parabola. Pokazaćemo da je njen minimum u intervalu $(0, \alpha_\tau]$.

Lema 1.3.4 *Za svako* $(x, y, s) \in \mathcal{F}^0$ *i za* $\rho > n + \sqrt{n}$, *važi*

$$\|r\|| \geq \frac{\sqrt{3}}{2\upsilon_{\min}}\frac{n\mu}{\rho}.$$

Odredićemo sada vrednost parametra α tako da se postiže konstantno smanjenje $q(\alpha)$ i Φ_ρ u svakoj iteraciji za fiksiranu vrenost $\tau = 0.5$ zbog jednostavnosti. Analogni rezultati se mogu pokazati za svako $\tau \in (0, 1)$.

Teorema 1.3.2 *Neka je* $\tau = 0.5$ *i neka je*

$$\bar{\alpha} = \frac{\upsilon_{\min}}{2\|r\|}.$$

Tada je $q(0) - q(\bar{\alpha}) \leq -0.15$, *i stoga* (1.3.2) *važi za* $\delta = 0.15$.

Napomenimo da se u praksi obično dobijaju veće vrednosti od ove donje granice.

Na osnovu 1.3.1 sledi rezultat o kompleksnosti algoritma.

Posledica 1.3.1 *Neka je* $\rho \geq n + \sqrt{n}$ *i* $\varepsilon > 0$, *i pretpostavimo da početna tačka* $(x^0, y^0, s^0) \in \mathcal{F}^0$ *zadovoljava*

$$\Phi_\rho(x^0, s^0) \leq \|(\rho - n)|\log\varepsilon|$$

za neko $\| > 0$ *nezavisno od n. Ako je K definisano sa*

$$K = \left[\frac{\kappa + 1}{0.15}(\rho - n)|\log\varepsilon|\right] = O((\rho - n)|\log\varepsilon|),$$

tada je

$$(x^k, y^k, s^k) \in \mathcal{F}^0, \quad \mu_k \leq \varepsilon,$$

za svako $k \geq K$.

Ovaj rezultat o kompleksnosti važi za svako ρ koje zadovoljava $\rho \geq n + \sqrt{n}$. Ako fiksiramo $\rho = n + \sqrt{n}$ dobija se najbolja procena kompleksnosti algoritma koja je jednaka $O(\sqrt{n}|\log\varepsilon|)$. Međutim, tada je

$$\sigma = \frac{n}{\rho} = \frac{n}{n + \sqrt{n}} \approx 1 - \frac{1}{\sqrt{n}} \quad \text{za veliko } n,$$

tj. parametar centriranja σ je blizu jedinice tako da nije moguć veliki korak u iteraciji. Manje vrednosti za σ se dobijaju birajući veće vrednosti za $\rho-$ obično je $\rho = 10n$ ili $\rho = n + n^{1.5}$. Time se kompleksnost povećava na $O(n|\log\varepsilon|)$ i $O(n^{1.5}|\log\varepsilon|)$ respektivno, ali se obično dobiju bolje numeričke performanse. Dalje, kako je Φ_ρ nelinearna funkcija višeg reda, nije pogodno izračunati tačnu vrednost njenog minimuma. Sa druge strane, ako je α i poznato, dobija se isuviše mali korak koji ne daje razumne numeričke performanse. U praksi su se bolje pokazale vrednosti $0.99\alpha_{\max}$ i $0.95\alpha_{\max}$.

1.4 Algoritmi koji slede centralnu putanju

Centralnu putanju $\mathcal{C}$ krakterišu KKT uslovi u kojima se umesto uslova komplementarnosti (1.1.4c) nameće uslov da proizvodi $x_i s_i$ imaju istu vrednost $\tau > 0$ za svako i, tj. $(x_\tau, y_\tau, s_\tau) \in \mathcal{C}$ ako je rešenje sistema:

$$\begin{aligned} A^T y + s &= c, && (1.4.1a)\\ Ax &= b, && (1.4.1b)\\ (x,s) &\geq 0. && (1.4.1c)\\ x_i s_i &= \tau, \quad i = 1,\ldots,n, && (1.4.1d) \end{aligned} \tag{1.4.1}$$

Pokazali smo da ovaj sistem ima jedinstveno rešenje (x_τ, y_τ, s_τ) za svako $\tau > 0$ kada je problem dopustiv (iako KKT uslovi mogu za $\tau = 0$ imati i višestruka rešenja). Algoritmi koji slede putanju podešavaju iteraciju duž $\mathcal{C}$ u pravcu opadanja τ ka skupu rešenja Ω. Oni generišu strogo dopustive iteracije (x^k, y^k, s^k) koje zadovoljavaju prva tri KKT uslova i odstupaju od centralne putanje $\mathcal{C}$ samo zato što proizvodi $x_i s_i$ u opštem slučaju nisu identični, tako da uslov (1.4.1d) nije zadovoljen u potpunosti. Odstupanje se meri upoređivanjem proizvoda sa njihovom srednjom vrednošću $\mu = x^T s/n$, koristeći, recimo, normu definisanu sa

$$\frac{1}{\mu}\|XSe - \mu e\|.$$

U literaturi se koriste i 2-norma i ∞-norma u ovoj definiciji. Za obe norme iz $(1/\mu)\|XSe - \mu e\| < 1$ sledi da su x i s strogo pozitivni. Podsetimo da je okolina $\mathcal{N}_2(\theta)$ putanje $\mathcal{C}$ definisana sa

$$\mathcal{N}_2(\theta) = \{(x,y,s) \in \mathcal{F}^0 |\, \|XSe - \mu e\|_2 \leq \theta\mu\}$$

pri čemu je odstupanje od $\mathcal{C}$ manje od $\theta \in (0,1)$, i okolina $\mathcal{N}_{-\infty}(\gamma)$ definisana sa

$$\mathcal{N}_{-\infty}(\gamma) = \{(x,y,s) \in \mathcal{F}^0 |\, x_i s_i \geq \gamma\mu \forall i = 1,\ldots,n\}$$

za neko $\gamma \in (0,1)$.

Algoritam kratkog koraka po putanji

Ovaj najjednostavniji algoritam su uveli Kojima, Mizuno i Yoshise [72] i Monteiro i Adler [73]. Početna tačka je u $\mathcal{N}_2(\theta)$ i koriste se konstantne vrednosti $\alpha_k = 1$ i $\sigma_k = \sigma$.

Algoritam KKP
Dato je $\theta = 0.4$, $\sigma = 1 - 0.4/\sqrt{n}$, i $(x^0, y^0, s^0) \in \mathcal{N}_2(\theta)$;
for $k = 0, 1, 2, \ldots$
staviti $\sigma_k = \sigma$ i rešiti

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix},$$

da bi dobili $(\Delta x^k, \Delta y^k, \Delta s^k)$;
postaviti

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + (\Delta x^k, \Delta y^k, \Delta s^k);$$

end (for).

Uvedimo oznake

$$\begin{aligned} (x(\alpha), y(\alpha), s(\alpha)) &= (x, y, s) + (\alpha)(\Delta x, \Delta y, \Delta s), \quad (1.4.5a) \\ \mu(\alpha) &= x(\alpha)^T s(\alpha)/n. \quad (1.4.5b) \end{aligned} \tag{1.4.2}$$

Lema 1.4.1 *Neka je korak* $(\Delta x, \Delta y, \Delta s)$ *definisan sa* (1.1.7). *Tada je*

$$\Delta x^T \Delta s = 0$$

i

$$\mu(\alpha) = (1 - \alpha(1 - \sigma))\mu.$$

Lema je iskazana u opštem slučaju jer se primenjuje na sve algoritme koji traženi pravac dobijaju iz sistema (1.1.7). Za konkretne vrednosti u algoritmu je

$$\mu_{k+1} = \sigma\mu = \left(1 - \frac{0.4}{\sqrt{n}}\right)\mu_k, \quad k = 0, 1, \ldots, \tag{1.4.3}$$

tako da je konvergencija linearna. Polinomijalna kompleksnost je direktna posledica (1.4.3) i Teoreme 1.2.1

Teorema 1.4.1 *Neka je* $\varepsilon > 0$ *i neka je* $(x^0, y^0, s^0) \in \mathcal{N}_2(0.4)$ *početna tačka Algoritma KKP za koju je* $\mu_0 \leq 1/\varepsilon^\kappa$ *za neku pozitivnu konstantu* κ. *Tada postoji indeks* K *pri čemu je* $K = O(\sqrt{n} \log 1/\varepsilon)$ *tako da je*

$$\mu_k \leq \varepsilon \quad \text{za svako } k \geq K.$$

Dokazaćemo sada da sve iteracije ostaju u istoj okolini.

Lema 1.4.2 *Neka su* u *i* v *vektori u* R^n *za koje važi* $u^T v \geq 0$. *Tada je*

$$\|UVe\| \leq 2^{-3/2}\|u + v\|^2,$$

gde je $U = diag(u_1, \ldots, u_n)$ *i* $V = diag(v_1, \ldots, v_n)$

U Lemi 1.4.1 je pokazano da je $\delta x^T \delta s = \sum \delta x_i \delta s_i = 0$, ali to ne znači da su svi sabirci jednaki nuli. U sledećoj lemi dajemo ograničenje za vektor proizvoda.

Lema 1.4.3 *Ako je* $(x, y, s) \in \mathcal{N}_2(\theta)$ *tada je*

$$\|\Delta X \Delta S e\| \leq \frac{\theta^2 + n(1-\sigma)^2}{2^{3/2}(1-\theta)}\mu.$$

Iz Leme 1.4.1 sledi da μ opada linearno kada se krećemo duž pravca $(\Delta x, \Delta y, \Delta s)$. Sledeća posledica Leme 1.4.3 pokazuje koliko tačka $(x(\alpha), y(\alpha), s(\alpha))$ odstupa od centralne putanje.

Lema 1.4.4 *Neka je* $(x, y, s) \in \mathcal{N}_2(\theta)$. *Tada je*

$$\begin{aligned}
&\|X(\alpha)S(\alpha)e - \mu(\alpha)e\| \\
&\leq |1-\alpha|\|XSe - \mu e\| + \alpha^2 \|\Delta X \Delta S e\| \\
&\leq |1-\alpha|\theta\mu + \alpha^2 \left[\frac{\theta^2 + n(1-\sigma)^2}{2^{3/2}(1-\theta)}\right]\mu.
\end{aligned}$$

Teorema 1.4.2 definiše odnos između θ i σ i pokazuje da i u slučaju punog koraka $\alpha = 1$ duž traženog pravca nova iteracija ostaje u okolini $\mathcal{N}_2(\theta)$.

Teorema 1.4.2 *Neka su parametri* $\theta \in (0,1)$ *i* $\sigma \in (0,1)$ *izabrani tako da zadovoljavaju jednakost*

$$\frac{\theta^2 + n(1-\sigma)^2}{2^{3/2}(1-\theta)} \leq \sigma\theta. \tag{1.4.4}$$

Tada za $(x, y, s) \in \mathcal{N}_2(\theta)$ *važi*

$$(x(\alpha), y(\alpha), s(\alpha)) \in \mathcal{N}_2(\theta)$$

za svako $\alpha \in [0, 1]$.

Lako se proverava da u Algoritmu KKP konkretan izbor parametara $\theta = 0.4$ i $\sigma = 1 - 0.4/\sqrt{n}$ zadovoljava uslove (1.4.4).

Prediktor-korektor metod

U Algoritmu KKP se σ bira strogo između 0 i 1. Time se u jednom koraku popravlja centralnost i smanjuje mera dualnosti μ. Prediktor-korektor metod, Algoritam PK, naizmenično bira dva tipa koraka

- *prediktor* korak sa $\sigma_k = 0$ koji smanjuje μ,
- *korektor* korak $\sigma_k = 1$ koji popravlja centralnost.

Razne varijante ovog algoritma su razmatrali Monteiro i Adler [73], Sonnevend, Stoer i Zhao [85], [86]. U nastavku sledimo jednostavan oblik algoritma koji su dali Mizuno, Todd i Ye [72]. Ovaj algoritam se ponekad naziva "Mizuno-Todd-Ye prediktor-korektor" da bi se razlikovao od prediktor-korektor algoritma Mehrotre koji se zasniva na drugim principima.

Algoritam PK karakteriše par $\mathcal{N}_2(\theta)$ okolina pri čemu je jedna unutar druge. Parne iteracije ostaju u unutrašnjoj okolini a neparne mogu biti i u spoljnoj okolini. Radi jednostavnosti, definisaćemo da je unutrašnja okolina $\mathcal{N}_2(0.25)$ i spoljna okolina $\mathcal{N}_2(0.5)$. Moguć je i drugi izbor parametara pod uslovom da okoline zadovoljavaju određene uslove.

Algoritam PK
Dato je $(x^0, y^0, s^0) \in \mathcal{N}_2(0.25)$;
for $k = 0, 1, 2, \ldots$
if k parno
(*prediktor korak*)
staviti $\sigma_k = 0$ i rešiti

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix},$$

da bi dobili $(\Delta x^k, \Delta y^k, \Delta s^k)$;
izabrati α_k kao najveću vrednost $\alpha \in [0, 1]$ za koju važi

$$(x^k(\alpha), y^k(\alpha), s^k(\alpha)) \in \mathcal{N}_2(0.5)$$

postaviti
$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k(\alpha_k), y^k(\alpha_k), s^k(\alpha_k))$;
else k neparno
(*korektor korak*)
rešiti prethodni sistem za $\sigma_k = 1$ da bi dobili $(\Delta x^k, \Delta y^k, \Delta s^k)$;
postaviti

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + (\Delta x^k, \Delta y^k, \Delta s^k);$$

end (if)
end (for).

Sledeća lema daje donju granicu za dužinu koraka i procenjuje smanjenje parametra dualnosti μ.

Lema 1.4.5 *Neka je* $(x, y, s) \in \mathcal{N}_2(0.25)$, *i neka je* $(\Delta x, \Delta y, \Delta s)$ *korak izračunat iz* (1.1.7) *za* $\sigma = 0$. *Tada je* $(x(\alpha), y(\alpha), s(\alpha)) \in \mathcal{N}_2(0.25)$ *za svako* $\alpha \in [0, \bar{\alpha}]$, *gde je*

$$\bar{\alpha} = \min\left(\frac{1}{2}, \left(\frac{\mu}{8\|\Delta X \Delta S e\|}\right)^{1/2}\right). \tag{1.4.5}$$

Dakle, prediktor korak je dužine minimum $\bar{\alpha}$, i nova vrednost mere dualnosti μ je najviše $(1-\bar{\alpha})\mu$.

Iz Leme 1.4.5 za $\theta = 0.25$ i $\sigma = 0$ sledi

$$\frac{\mu}{8\|\Delta X \Delta S e\|} \geq \frac{0.16}{n}$$

i iz (1.4.5) sledi procena

$$\bar{\alpha} \geq \min\left(\frac{1}{2}, \left(\frac{0.16}{n}\right)^{1/2}\right) = \frac{0.4}{\sqrt{n}}.$$

Odatle je za parne indekse

$$\mu_{k+1} \leq \left(1 = \frac{0.4}{\sqrt{n}}\right)\mu_k, \quad k = 0, 2, 4, \ldots \tag{1.4.6}$$

Sledeća lema pokazuje da korektor korak bilo koju tačku iz $\mathcal{N}_2(0.5)$ vraća u unutrašnju okolinu $\mathcal{N}_2(0.25)$ ne menjajući pri tome vrednost parametra μ.

Lema 1.4.6 *Neka je $(x, y, s) \in \mathcal{N}_2(0.5)$ i neka je $(\Delta x, \Delta y, \Delta s)$ korak izračunat iz (1.1.7) za $\sigma = 1$. Tada je*

$$(x(1), y(1), s(1)) \in \mathcal{N}_2(0.25), \quad \mu(1) = \mu.$$

Iako korektor korak ne menja vrednost μ zbog (1.4.6) se dobija polinomijalna kompleksnost kao i kod Algoritma KKP.

Teorema 1.4.3 *Neka je $\varepsilon > 0$ i neka je $(x^0, y^0, s^0) \in \mathcal{N}_2(0.25)$ početna tačka Algoritma PK za koju je $\mu_0 \leq 1/\varepsilon^\kappa$ za neku pozitivnu konstantu κ. Tada postoji indeks K pri čemu je $K = O(\sqrt{n}\log 1/\varepsilon)$ tako da je*

$$\mu_k \leq \varepsilon \quad \text{za svako } k \geq K.$$

Algoritam PK je poboljšanje Algoritma KKP jer u krajnjim iteracijama prediktor pravac postaje sve bolji, tako da je moguće uzeti dužinu koraka blizu 1. Ustvari, pokazuje se da je konvergencija mere dualnosti μ_k ka nuli superlinearna.

Algoritam dugog koraka po putanji

Algoritam DKP je u sličan prvom polinomijalnom primal-dual algoritmu koji su dali Kojima, Mizuno i Yoshise [59]. Praktično je primenljiviji od Algoritama KKP i PK ali mu je polinomijalnost lošija. Algoritam DKP generiše niz iteracija u okolini $\mathcal{N}_{-\infty}(\gamma)$, koja za male vrednosti γ zahvata veliki deo skupa strogo ostvarljivih tačaka $\mathcal{F}^0$.

Algoritam DKP
Dato je $\gamma, \sigma_{\min}, \sigma_{\max}$, pri čemu je $\gamma \in (0, 1)$, $0 < \sigma_{\min} < \sigma_{\max} < 1$,

i $(x^0, y^0, s^0) \in \mathcal{N}_{-\infty}(\gamma)$;
for $k = 0, 1, 2, \ldots$
izabrati $\sigma_k \in [\sigma_{\min}, \sigma_{\max}]$ i rešiti

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix},$$

da bi dobili $(\Delta x^k, \Delta y^k, \Delta s^k)$;
izabrati za α_k najveću vrednost od $\alpha \in [0, 1]$, tako da je

$$(x^k(\alpha), y^k(\alpha), s^k(\alpha)) \in \mathcal{N}_{-\infty}(\gamma);$$

postaviti

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k(\alpha_k), y^k(\alpha_k), s^k(\alpha_k));$$

end (for).

Sledeća dva tvrđenja daju donju granicu za α_k i odgovarajuću procenu za smanjenje parametra μ u svakoj iteraciji.

Lema 1.4.7 *Neka je* $(x, y, s) \in \mathcal{N}_{-\infty}(\gamma)$, *tada je*

$$\|\Delta X \Delta S e\| \leq 2^{-3/2} \left(1 + \frac{1}{\gamma}\right) n\mu.$$

Teorema 1.4.4 *Za date parametre* $\gamma, \sigma_{\min}$ *i* $\sigma_{\max}$ *u Algoritmu DKP postoji konstanta* δ *nezavisna od* n *tako da je*

$$\mu_{k+1} \leq \left(1 - \frac{\delta}{n}\right) \mu_k$$

za svako $k \geq 0$.

Polinomijalnost Algoritma DKP je direktna posledica prethodne teoreme.

Teorema 1.4.5 *Neka je* $\varepsilon > 0$ *i neka je* $\gamma \in (0, 1)$. *Neka je* $(x^0, y^0, s^0) \in \mathcal{N}_{-\infty}(\gamma)$ *početna tačka Algoritma DKP za koju je* $\mu_0 \leq 1/\varepsilon^\kappa$ *za neku pozitivnu konstantu* κ. *Tada postoji indeks* K *pri čemu je* $K = O(n \log 1/\varepsilon)$ *tako da je*

$$\mu_k \leq \varepsilon \quad \text{za svako } k \geq K.$$

Granične tačke iteracionog niza

Pokazali smo da u razmatranim algoritmima $\mu_k \to 0$, ne govoreći ništa o nizu $\{(x^k, y^k, s^k)\}$. Treba pokazati da niz $\{(x^k, s^k)\}$ ima graničnu vrednost. U slučaju da je $\mathcal{K}$ podniz za koji je

$$\lim_{k \in \mathcal{K}} (x^k, s^k) = (x^*, s^*),$$

tada je za svako $k \in \mathcal{K}$ ispunjeno

$$Ax^k = b, \quad c - s^k \in R(A^T), \quad (x^k, s^k) > 0.$$

Kako je $R(A^T)$ zatvoren i $\mu_k \to 0$, to za (x^*, s^*) važi

$$Ax^* = b, \quad c - s^* \in R(A^T), \quad (x^*, s^*) > 0, \quad (x^*)^T s^* = 0.$$

Dakle, $c - s^* = Ay^*$ za neko y^*. Upoređujući ove uslove sa KKT uslovima (1.4.1) zaključujemo $(x^*, y^*, s^*) \in \Omega$.

Güler i Ye [47] su dokazali da je niz $\{(x^k, s^k)\}$ ograničen i stoga ima bar jedan konvergentan podniz. Dalje, granična vrednost tih podnizova je strogo komplementarno rešenje.

Lema 1.4.8 *Neka je $\mu_0 > 0$ i $\gamma \in (0, 1)$. Tada za sve tačke (x, y, s) za koje važi*

$$(x, y, s) \in \mathcal{N}_{-\infty}(\gamma) \subset \mathcal{F}^0, \quad \mu \leq \mu_0$$

(gde je $\mu = x^T s/n$), postoje konstante C_0 i C_3 tako da je

$$\begin{aligned} \|(x, s)\| &\leq C_0 \\ 0 < x_i \leq \mu/C_3 \, (i \in \mathcal{N}), &\qquad 0 < s_i \leq \mu/C_3 \, (i \in \mathcal{B}), \\ s_i \geq C_3\gamma \, (i \in \mathcal{N}), &\qquad x_i \geq C_3\gamma \, (i \in \mathcal{B}). \end{aligned}$$

Teorema 1.4.6 *Neka je $\{(x^k, y^k, s^k)\}$ niz iteracija generisan Algoritmom KKP, PK ili DKP, i pretpostavimo da $\mu_k \to 0$ kada $k \to \infty$. Tada je niz $\{(x^k, s^k)\}$ ograničen i stoga ima bar jednu graničnu tačku. Svaka granična tačka odgovara strogo komplementarnom primal-dual rešenju.*

Posledica prethodne teoreme je da kada problem ima jedinstveno primal-dual rešenje (x^*, y^*, s^*), tada iteracioni niz bilo kog od razmatranih algoritama konvergira ka toj tački.

1.5 Algoritmi koji startuju iz nedopustive tačke

Prethodno razmatrani algoritmi za početnu tačku zahtevaju strogo dopustivu tačku $(x^0, y^0, s^0) \in \mathcal{F}^0$ koja je takođe i u nekoj okolini centralne putanje $\mathcal{C}$. Međutim, često nije lako odrediti takvu tačku. Postoje i problemi kod kojih takva tačka uopšte ne postoji. U problemu

$$\min 2x_1 + x_2 \quad \text{u odnosu na } x_1 + x_2 + x_3 = 5,\ x_1 + x_3 = 5,\ x \geq 0,$$

je primal dopustiv skup $\{(\beta, 0, 5 - \beta) | \, \beta \in [0, 5]\}$. Kako je $x_2 = 0$, strogo dopustiv skup $\mathcal{F}^0$ je prazan. U opštem slučaju, linearni problemi koji se dobijaju svođenjem opšteg problema na standardni oblik često nemaju strogo dopustive tačke. Jedan od

načina za prevazilaženje te teškoće je potapanje datog linearnog problema u nešto veći problem za koji je skup $\mathcal{F}^0$ neprazan.

Drugi način daje algoritam koji ne zahteva da početna tačka bude strogo dopustiva već se samo traži da komponente x i s budu strogo pozitivne. U opštem slučaju sve iteracije (x^k, y^k, s^k) generisane algoritmom su nedopustive, ali je granična tačka dopustiva i optimalna. Algoritam je moguće primeniti i kada je $\mathcal{F}^0 = \emptyset$. Ovakv pristup su dali Kojima, Megido i Mizuno [58], Zhang [122], Potra [79] i Wright [114].

Rezidume sistema jednačina

$$r_b - Ax - b, \quad r_c = A^T y + s - c$$

za $k-$tu iteraciju označimo sa r_b^k i r_c^k. Okolinu $\mathcal{N}_{-\infty}(\gamma)$ koja sadrži i nedopustive tačke definišmo sa

$$\begin{aligned}\mathcal{N}_{-\infty}(\gamma,\beta) = \quad & \{(x,y,s)|\, \|(r_b,r_c)\| \le [\|(r_b^0,r_c^0)\|/\mu_0]\beta\mu, \\ & (x,s) > 0, x_i s_i \ge \gamma\mu,\ i = 1,\dots,n\}\end{aligned}$$

gde su $\gamma \in (0,1)$ i $\beta \ge 1$ dati parametri i vrednosti (r_b^0, r_c^0) i μ_0 izračunate u početnoj tački (x^0, y^0, s^0). Primetimo da je neophodan uslov

$$\beta \ge 1$$

da bi osigurali da početna tačka (x^0, y^0, s^0) pripada $\mathcal{N}_{-\infty}(\gamma, \beta)$.

Algoritam NPD

Dato je $\gamma, \beta, \sigma_{\min}, \sigma_{\max}$, pri čemu je $\gamma \in (0,1)$, $\beta \ge 1$, i $0 < \sigma_{\min} < \sigma_{\max} \le 0.5$;

izabrati (x^0, y^0, s^0) tako da je $(x^0, s^0) > 0$;

for $k = 0, 1, 2, \dots$

izabrati $\sigma_k \in [\sigma_{\min}, \sigma_{\max}]$ i rešiti

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} -r_c^k \\ -r_b^k \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix},$$

izabrati za α_k najveću vrednost od $\alpha \in [0,1]$, tako da je

$$(x^k(\alpha), y^k(\alpha), s^k(\alpha)) \in \mathcal{N}_{-\infty}(\gamma, \beta) \tag{1.5.1}$$

i da važi sledeći uslov Armija:

$$\mu_k(\alpha) \le (1 - 0.01\alpha)\mu_k \tag{1.5.2}$$

postaviti

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k(\alpha_k), y^k(\alpha_k), s^k(\alpha_k));$$

end (for).

Neka je ν_k definisano sa

$$\nu_k = \prod_{j=0}^{k-1}(1-\alpha_j) \quad (\nu_0 = 1).$$

Kako su prve dve komponente funkcije F linearne, sledi

$$(r_b^k, r_c^k) = (1-\alpha_{k-1})(r_b^{k-1}, r_c^{k-1}) = \cdots = \nu_k(r_b^0, r_c^0), \tag{1.5.3}$$

i kako je $(x^k, y^k, s^k) \in \mathcal{N}_{-\infty}(\gamma, \beta)$, to je

$$\nu_k\|(r_b^0, r_c^0)\|/\mu_k = \|(r_b^k, r_c^k)\|/\mu_k \leq \beta\|(r_b^0, r_c^0)\|/\mu_0.$$

Ako pretpostavimo da je $(r_b^0, r_c^0) \neq 0$ tada je

$$\nu_k \leq \beta\mu_k/\mu_0.$$

U dopustivom slučaju je $(r_b^0, r_c^0) = 0$ i sve iteracije (x^k, y^k, s^k) su strogo dopustive i tada se Algoritam NPD svodi na Algoritam DKP. U nastavku pretpostavljamo da je početna tačka (x^0, y^0, s^0) nedopustiva.

Konvergencija algoritma sledi ako pokažemo egzistenciju konstante $\bar{\alpha}$ takve da je $\alpha_k \geq \bar{\alpha}$ za svako k. Iz uslova (1.5.2) sledi

$$\mu_{k+1} \leq (1 - 0.01\alpha_k) \leq (1 - 0.01\bar{\alpha})\mu_k \quad \text{za svako } k \geq 0,$$

i stoga niz $\{\mu_k\}$ konvergira Q-linearno ka nuli. Iz (1.5.3) takođe sledi

$$\|(r_b^{k+1}, r_c^{k+1})\| \leq (1-\bar{\alpha})\|(r_b^k, r_c^k)\|,$$

odakle sledi da niz rezidualnih normi teži nuli.

Teorema 1.5.1 *Niz $\{\mu_k\}$ generisan Algoritmom NPD konvergira Q-linearno ka nuli, i niz rezidualnih normi $\{\|(r_b^k, r_c^k)\|\}$ konvergira R-linearno ka nuli.*

Polinomijalna kompleksnost algoritma sledi iz činjenice da je donja granica $\bar{\alpha}$ dužine koraka inverzna polinomska funkcija od n ako se početna tačka izabere u obliku

$$(x^0, y^0, s^0) = (\zeta e, 0, \zeta e), \tag{1.5.4}$$

gde je ζ skalar za koji važi

$$\|(x^*, s^*)\|_\infty \leq \zeta, \tag{1.5.5}$$

za neko primal-dual rešenje (x^*, y^*, s^*). Iako se optimalno rešenje ne zna unapred, ovaj uslov je pogodan za praktičnu primenu jer dobro centrirana početna tačka za koju je količnik

$$\|(r_b^0, r_c^0)\|/\mu_0 \tag{1.5.6}$$

mali vodi bržoj konvergenciji nego tačka koja je mnogo bliža skupu rešenja Ω i slabo centrirana. Tačka (1.5.4) je centrirana i količnik (1.5.6) je približno $1/\zeta$.

Teorema 1.5.2 *Neka je $\varepsilon > 0$ dato i neka je data početna tačka $(x^0, y^0, s^0) = (\zeta e, 0, \zeta e)$, gde je ζ definisano u Lemi 1.5.2. Pretpostavimo da važi*

$$\zeta^2 \leq \frac{C}{\varepsilon^\kappa}$$

za neke pozitivne konstante C i κ. Tada postoji

$$K = O(n^2 |\log \varepsilon|)$$

tako da iteracije $\{(x^k, y^k, s^k)\}$ generisane Algoritmom NPD zadovoljavaju

$$\mu_k \leq \varepsilon \quad \text{za svako } k \geq K.$$

Dokazi Teorema 1.5.1 i 1.5.2 slede iz rezultata Kojime [57], Zhanga [122] i Wrighta [113]. Primetimo da iz

$$A\bar{x} = 0, \quad A^T\bar{y} + \bar{s} = 0,$$

sledi

$$\bar{x}^T\bar{s} = -\bar{x}^T A^T \bar{y} = 0.$$

Lema 1.5.1 *Postoji pozitivna konstanta C_1 tako da je*

$$\nu_k \|(x^k, s^k)\|_1 \leq C_1 \mu_k \quad \text{za svako } k \geq 0.$$

Lema 1.5.2 *Pretpostavimo da početna tačka zadovoljava (1.5.4) i (1.5.5). Tada za svaku iteraciju (x^k, y^k, s^k) važi*

$$\zeta \nu_k \|(x^k, s^k)\|_1 \leq 4\beta n \mu_k.$$

Lema 1.5.3 *Postoji pozitivna konstanta C_2 tako da je*

$$\|(D^k)^{-1}\Delta x^k\|_1 \leq C_2 \mu_k^{1/2}, \quad \|D^k \Delta s^k\| \leq C_2 \mu_k^{1/2}$$

za svako $k \geq 0$.

Lema 1.5.4 *Pretpostavimo da početna tačka zadovoljava (1.5.4) i (1.5.5). Tada postoji konstanta ω nezavisna od n tako da je*

$$\|(D^k)^{-1}\Delta x^k\|_1 \leq \omega \mu_k^{1/2}, \quad \|D^k \Delta s^k\| \leq \omega \mu_k^{1/2}. \tag{1.5.7}$$

Lema 1.5.5 *Postoji $\bar{\alpha} \in (0,1)$ tako da sledeća tri uslova važe za svako $\alpha \in [0, \bar{\alpha}]$ i svako $k \geq 0$:*

$$\begin{aligned}
&(x^k + \alpha\Delta x^k)^T (s^k + \alpha \Delta s^k) \geq (1-\alpha)(x^k)^T s^k, \\
&(x_i^k + \alpha \Delta x_i^k)(s_i^k + \alpha \Delta s_i^k) \geq (\gamma/n)(x^k + \alpha \Delta x^k)^T (s^k + \alpha \Delta s^k), \\
&(x^k + \alpha\Delta x^k)^T (s^k + \alpha \Delta s^k) \leq (1 - 0.01\alpha)(x^k)^T s^k.
\end{aligned}$$

Dakle, uslovi (1.5.1) i (1.5.2) su zadovoljeni za svako $\alpha \in [0, \bar{\alpha}]$ i svako $k \geq 0$.

Ako su zadovoljeni uslovi Leme 1.5.4, tada je

$$\bar{\alpha} \geq \bar{\delta}/n^2$$

za neki pozitivan broj $\bar{\delta}$ koji ne zavisi od n.

Teorema 1.5.3 *Neka je (x^k, y^k, s^k) niz generisan Algoritmom NPD. Tada postoji konstanta C_3 tako da za svako dovoljno veliko k važi*

$$0 < x_i^k \le \mu_k / C_3 \, (i \in \mathcal{N}), \qquad 0 < s_i^k \le \mu_k / C_3 \, (i \in \mathcal{B}),$$
$$s_i^k \ge C_3 \gamma \, (i \in \mathcal{N}), \qquad x_i^k \ge C_3 \gamma \, (i \in \mathcal{B}).$$

Dakle, bilo koja granična tačka niza $\{(x^k, s^k)\}$ može da posluži za konstrukciju striktno komplementarnog rešenja.

Teorema 1.5.4 *Neka je $\{(x^k, y^k, s^k)\}$ niz iteracija generisan Algoritmom NPD, i pretpostavimo da $\mathcal{F}^0 \neq \emptyset$. Tada je niz $\{(x^k, s^k)\}$ ograničen i stoga ima bar jednu graničnu taču.*

2 Simbolička implementacija Mehrotraovog algoritma

U ovom odeljku dajemo teorijske osnove primal-dual meotda i opisujemo simboličku implementaciju Mehrotraovog primal-dual metoda u programskom paketu MATHEMATICA. Dajemo i jednostavan algoritam za konstrukciju početne tačke iteracije. Kako je većina dostupnih test primera data u tzv. *MPS* formatu, razvijen je i softver za konverziju fajlova pisanih u *MPS* formatu u odgovarajući *NB* fajl. Implementacija je verifikovana na nekoliko ilustrativnih primera.

Simbolička implementacija primal-dual metoda podrazumeva mogućnost manipulacije formulama od strane kompjutera za vreme implementacije tih metoda. Poznati programski paketi bazirani na metodama unutrašnje tačke su razvijani u programskim jezicima FORTRAN, C i MATLAB. Vanderbeiov program LOQO [105] je pisan u programu C i dostupan je na internet adresi `http://www.sor.princeton.edu-/ rvdb/`. LIPSOL [123] je pisan u programu MATLABU i FORTRANU, i dostupan je na internet adresi `http://pc5.math.umbc.edu/ yzhang/`. Gondziov HOPDM je dostupan u izvornom kodu u FORTRANU [38]. Izvorni fajl je dostupan na `http://ecolu-info.unige.ch/ logilab/software/`. Jedan od popularnih i jakih kodova je PCx [13]. Veći deo PCx [14] koda je pisan u C-u, ali je glavni računski deo - Cholesky faktorizacija - pisan u FORTRANU 77 [2].

U ovom odeljku je razvijen eksperimentalni kod za primal-dual algoritam u programu MATHEMATICA. Cilj nije bio softver sa izuzetnim karakteristikama već predstavlja primenu mogućnosti simboličkog izračunavanja u programu MATHEMATICA. O programu MATHEMATICA videti [108] i [109]. U kodu je iskorišćena činjenica da MATHEMATICA poseduje velike mogućnosti za rešavanje sistema linearnih jednačina.

2.1 Opis Mehrotraovog algoritma

Mehrotraov algoritam generiše niz nedopustivih iteracija (x^k, y^k, s^k) za koje je $(x^k, s^k) > 0$. Traženi pravac se u svakoj iteraciji sastoji od tri komponente:

• afini "prediktor" pravac - Newtonov pravac za funkciju $F(x, y, s)$ definisanu sa (1.1.5),

• izraza za centriranje čija vrednost se određuje na osnovu podesivog izbora parametra za centriranje σ,

• "korektor" pravac koji kompenzuje izvesne nelinearnosti afinog pravca.

Osnovna odlika ovog algoritma je da se parametar centriranja ne određuje unapred kao kod opisanih algoritama. Ukoliko afini pravac obezbeđuje značajno smanjenje mere dualnosti μ pri čemu ostaje $(x, s) > 0$, zaključujemo da je potrebno malo centriranje u toj iteraciji, tako da se za σ uzima mala vrednost. Sa druge strane, ukoliko se u afinom pravcu dopušta samo kratak pomeraj pre nego što se naruši uslov $(x, s) > 0$, zaključujemo da je potrebna veća vrednost parametra za centriranje tako da uzimamo σ bliže jedinici.

Sada razmatramo Mehrotraov primal-dual algoritam [14, 112].

Algoritam MPD

Korak 1. Generisati početnu tačku (x^k, λ^k, s^k), $k = 0$.

Korak 2. Izračunati rezidume

$$r_b = Ax^k - b, \qquad r_c = A^T\lambda^k + s^k - c$$

i proverimo kriterijum za kraj algoritma

$$\frac{|r_b|}{1+|b|} \leq \epsilon, \quad \frac{|r_c|}{1+|c|} \leq \epsilon, \quad \frac{|c^Tx - b^T\lambda|}{1+|c^Tx|} \leq \epsilon.$$

U praksi je retkost da je treći uslov ispunjen i da istovremeno druga dva uslova ne važe. Prema tome, treći uslov je najvažniji i verovatno jedini uslov koji treba proveriti [2]. Najčešće se za granicu greške uzima vrednost $\epsilon = 10^{-8}$ [2].

Ako je kriterijum za kraj ispunjen, izlazni podatak je x^k; u suprotnom, ići na *Korak 3.*

Korak 3. Formirati matrice

$$S = \mathrm{diag}(s_1, \ldots, s_n), \; X = \mathrm{diag}(x_1, \ldots, x_n),$$

koje predstavljaju diajgonalne matrice sa vrednostima $s_1, \ldots, s_n$ i $x_1, \ldots, x_n$ na glavnoj dijagonali, respektivno. Konstruisati vektor e, definisan sa $e=(1, \ldots, 1)^T \in \mathbb{R}^n$.

Korak 4. Neka je $D = S^{-1/2}X^{1/2}$, i $r_{xs} = XSe$ i rešimo sledeći sistem u odnosu na $(\Delta x^{aff}, \Delta\lambda^{aff}, \Delta s^{aff})$:

$$\begin{aligned} rclAD^2A^T\Delta\lambda^{aff} &= -r_b - A(S^{-1}Xr_c - S^{-1}r_{xs}), && (2.1.1)\\ \Delta s^{aff} &= -r_c - A^T\Delta\lambda^{aff}, && (2.1.2)\\ \Delta x^{aff} &= -S^{-1}(r_{xs} + X\Delta s^{aff}). && (2.1.3) \end{aligned}$$

Napomenimo da je sistem ekvivalentan sistemu (1.1.7) pri čemu je parametar $\sigma = 0$. Time se dobija afini pravac iteracije.

Korak 5. Izračunati meru dualnosti $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i s_i$.

Korak 6. Izračunati uslov za nenegativnost iteracione tačke

$$\alpha_{aff}^{pri} = \max\{\alpha \in [0,1] \,:\, x^k + \alpha \Delta x^{aff} \geq 0\}$$
$$\alpha_{aff}^{dual} = \max\{\alpha \in [0,1] \,:\, s^k + \alpha \Delta s^{aff} \geq 0\}.$$

Korak 7. Izračunati $\mu_{aff} = \frac{1}{n}(x^k + \alpha_{aff}^{pri}\Delta x^{aff})(s^k + \alpha_{aff}^{dual}\Delta s^{aff})$ i $\sigma = (\frac{\mu_a ff}{\mu})^3$.

Ako je $\mu aff \ll \mu$, tada afini pravac obezbeđuje veliko smanjenje parametra μ tako da u *Koraku 8* treba izabrati parametar σ bliže nuli, tj. nastaviti iteraciju po afinom pravcu. Ako μaff nije značajno smanjeno u odnsu na μ, u *Koraku 8* treba izabrati parametar σ bliže jedinici. Time se iterativna tačka vraća bliže centralnoj putanji $\mathcal{C}$, tako da je algoritam u boljoj poziciji da sledećoj iteaciji značajno redukuje μ. Kriterijum za izbor parametra σ je maksimalno smanjenje μ. Međutim, kako je pri tome potreban veliiki broj aritmetičkih operacija, Mehrotra je predložio heuristiku $\sigma = (\frac{\mu_a ff}{\mu})^3$ koja se pokazala efikasnom na velikom broju test primera. Time se postiže velika ušteda u procesorskom vremenu potrebnom za izvršenje algoritma.

Korak 8. Izračunati $r_{xs} = -\sigma\mu e + \Delta X^{aff}\Delta S^{aff} e$, gde je

$$\Delta X^{aff} = \mathrm{diag}(\Delta x_1^{aff}, \ldots, \Delta x_n^{aff}), \quad \Delta S^{aff} = \mathrm{diag}(\Delta s_1^{aff}, \ldots, \Delta s_n^{aff}),$$

i rešiti sledeći sistem u odnosu na Δx^{cor}, $\Delta\lambda^{cor}$, i Δs^{cor}:

$$rclAD^2A^T\Delta\lambda^{cor} = AS^{-1}r_{xs}, \tag{2.1.4}$$
$$\Delta s^{cor} = -A^T\Delta\lambda^{cor}, \tag{2.1.5}$$
$$\Delta x^{cor} = -S^{-1}(r_{xs} + X\Delta s^{cor}). \tag{2.1.6}$$

Korak 9. Staviti

$$(\Delta x^k, \Delta\lambda^k, \Delta s^k) = (\Delta x^{aff}, \Delta\lambda^{aff}, \Delta s^{aff}) + (\Delta x^{cor}, \Delta\lambda^{cor}, \Delta s^{cor})$$

Korak 10. Izračunati

$$\alpha_{max}^{pri} = \max\{\alpha \geq 0 \,:\, x^k + \alpha\Delta x^k \geq 0\}$$
$$\alpha_{max}^{dual} = \max\{\alpha \geq 0 \,:\, s^k + \alpha\Delta s^k \geq 0\}.$$

Korak 11. Staviti

$$\alpha_k^{pri} = \min\{0.99\alpha_{max}^{pri}, 1\}$$
$$\alpha_k^{dual} = \min\{0.99\alpha_{max}^{dual}, 1\}.$$

Korak 12. Izračunati sledeću iteraciju

$$x^{k+1} = x^k + \alpha_k^{pri}\Delta x^k,$$
$$(\lambda^{k+1}, s^{k+1}) = (\lambda^k, s^k) + \alpha_k^{dual}(\Delta\lambda^k, \Delta s^k).$$

staviti $(x^k, \lambda^k, s^k) = (x^{k+1}, \lambda^{k+1}, s^{k+1})$, zameniti k sa $k+1$ i preći na *Korak 2.*

2.2 Implementacija Mehrotraovog algoritma

Unutrašnja reprezentacija problema

Sada dajemo opis implementacije Mehrotraovog algoritma u programu MATHEMATICA. Deo ovih rezultata je publikovan u radovima [91, 92]. Razmatraćemo problem linearnog programiranja u opštem obliku. Umesto ulaznog fajla u *MPS* formatu, koristimo *NB* fajl format koji podržava program MATHEMATICA. Na taj način je problem linearnog programiranja predstavljen u intuitivnijoj i prirodnijoj formi, koju zovemo unutrašnja reprezentacija i upisujemo je u obliku *NB* fajla. Unutrašanja reprezentacija problema sadrži dva dela: izraz koji predstavlja ciljnu funkciju i listu datih ograničenja.

Primer 2.2.1 Unutrašnja reprezentacija test problema koji je u literaturi poznat pod nazivom *Afiro* jednaka je

```
-.4x2-.32x13-.6x17-.48x29+10.x32,
{-x1+x2+x3==0,-1.06x1+x4==0,x1<=80.,-x2+1.4x13<=0,
   -x5-x6-x7-x8+x13+x14==0,-1.06x5-1.06x6-.96x7-.86x8+x15==0,
   x5-x9<=80.,x6-x10<=0,x7-x11<=0,x8-x12<=0,-x16+x17+x18+x19==0,
   -.43x16+x20==0,x16<=500.,-x17+1.4x29<=0,
   -.43x21-.43x22-.39x23-.37x24+x31==0,
   x21+x22+x23+x24-x29+x30+x32==44.,
   x21-x25<=500.,x22-x26<=0,x23-x27<=0,x24-x28<=0,
   2.364x9+2.386x10+2.408x11+2.429x12-x19+2.191x25+2.219x26+
        2.249x27+2.279x28<=0,
   -x3+.109x16<=0,-x14+.109x21+.108x22+.108x23+.107x24<=0,
   .301x1-x18<=0,.301x5+.313x6+.313x7+.326x8-x30<=0,
   x4+x20<=310.,x15+x31<=300.}
```

Kako je u programu MATHEMATICA moguća manpulacija formulama, nije veliki problem uzeti kompletnu informaciju o problemu iz takve reprezentacije. Pretpostavimo da je ciljna funkcija označena parametrom *objective*. Takođe, neka je lista ograničenja označena sa *constraints*.

Korak a) U ovom koraku se rešavaju sledeća dva glavna problema:

- generisanje matrice problema a, koja je da data u unutrašnjoj reprezentaciji *objective*, *constraints*, i

- generisanje liste koeficijenata koji predstavljaju ciljnu funkciju.

a1) Transformišemo listu datih ograničenja i podelimo je u tri dela. Deo *les* je generisan posle sledeće transformacije uslova tipa <=.

```
les=Cases[constraints,x_<=y_->x-y];
```

Slično, drugi i treći deo su generisani iz ograničenja tipa >= i ==, respektivno.

```
gre=Cases[constraints,x_>=y_->x-y];
equ=Cases[constraints,x_==y_->x-y];
```

Dužina generisanih listi je korisna informacija u implementaciji.

```
p=Length[les];  q=Length[gre];   m=Length[equ];
```

a2) Lista b datih promenljivih se generiše u sledećem kodu:

```
b={};
For[i=1,i<=p,i++,
   b=Union[b,Map[Variables,les,{1}][[i]] ]
];
For[i=1,i<=q,i++,
   b=Union[b,Map[Variables,gre,{1}][[i]] ]
];
For[i=1,i<=m,i++,
   b=Union[b,Map[Variables,equ,{1}][[i]] ]
];
```

a3) Lista koja sadrži uređeni par čiji prvi element čine transformisana ograničenja a drugi element je lista promenljivih, generiše se u sledećem kôdu:

```
For[i=1,i<=p,i++,
   a=Append[a,List[les[[i]],b]]
];
For[i=1,i<=q,i++,
   a=Append[a,List[gre[[i]],b]]
];
For[i=1,i<=m,i++,
    a=Append[a,List[equ[[i]],b]]
];
```

a4) Sada se matrica a standardnog oblika problema može generisati primenjujući funkciju *Apply* kao funkcional:

```
a=Apply[Coefficient,a,{1}];
```

Konačno, lista ce koja sadrži koeficijente ciljne funkcije se generiše na sledeći način:

```
ce=Coefficient[objective,b];   k=Length[ce];
```

Primer 2.2.2 Za unutrašnju reprezentaciju

```
-2x1+3x2, {3x1+4x2<=14, 3x1+4x2>=5, x1+2x2<=6, x1+3x2==3}
```

imamo $p = 2$, $q = 1$ i $a = \{\{3,4\},\{1,2\},\{3,4\},\{1,3\}\}$. U slučaju minimizacije je $ce = \{-2,3\}$, i $ce = \{2,-3\}$ u slučaju maksimizacije.

Korak b) Sada moramo transformisati problem u standardni oblik i generisati matricu tog problema a. Primenom funkcije $a = form[a,p,q]$ moguće je generisati matricu a standardnog oblika lineranog problema. To se može postići dodavanjem q-dimenzionog vektora oblika $\{0,\ldots,0,\alpha,0,\ldots,0\}$ i-toj vrsti matrice a, gde je $\alpha \in \{-1,0,1\}$ postavljeno na i-tu poziciju.

```
form[B_,p_,q_]:=
 Module[{i,L={},y},
    For[i=1, i<=First[Dimensions[B]], i++,
       If[i<=p,
           y=Join[B[[i]],ReplacePart[Table[0,{q}],1,i]],
           If[i<=q,
```

```
            y=Join[B[[i]],ReplacePart[Table[0,{q}],-1,i]],
            y=Join[B[[i]],Table[0,{q}]]
      ]   ];
      L=Append[L,y]
    ];
    Return[L]
]
```

Sada se matrica a standardnog oblika problema i proširena lista promenljivih u ciljnoj funkciji može generisati sledećim kodom:

```
q=p+q; m=q+m;
a=form[a,p,q];
ce=Join[ce,Table[0,{q}]];
```

Na taj način, prvih p vrsta matrice a, koje odgovaraju ograničenjima tipa $<=$, proširene su vektorima $\{0,\ldots,0,1,0,\ldots,0\}$; vrste matrice a koje odgovaraju ograničenjima tipa $>=$, indeksirane brojevima $p+1,\ldots,q$, proširene su vektorima $\{0,\ldots,0,-1,0,\ldots,0\}$; vrste matrice a koje odgovaraju ograničenjima tipa $==$, indeksirane brojevima $q+1,\ldots,m$, proširene su vektorima $\{0,\ldots,0,0,0,\ldots,0\}$;

Primer 2.2.3 Za skup ograničenja iz Primera 2.2.2 rezultat funkcije $form[a,p,q]$ je lista

$$a = \{\{3,4,1,0,0\},\{1,2,0,1,0\},\{3,4,0,0,-1\},\{1,3,0,0,0\}\}.$$

Takođe, ciljna funkcija je data listom $\{-2,3,0,0,0\}$ (za minimizaciju) ili $\{2,-3,0,0,0\}$ (za maksimizaciju).

Korak c) Generisati listu b slobodnih koeficijenata sa desne strane datih ograničenja.

```
b={};
For[i=1,i<=p,i++,
   If[NumberQ[First[Cases[les[[i]],x_]]],
      b=Append[b,-First[Cases[les[[i]],x_]]],
      b=Append[b,0]
]  ];
For[i=1,i<=q,i++,
    If[NumberQ[First[Cases[gre[[i]],x_]]],
       b=Append[b,-First[Cases[gre[[i]],x_]]],
       b=Append[b,0]
]   ];
For[i=1,i<=m,i++,
    If[NumberQ[First[Cases[equ[[i]],x_]]],
       b=Append[b,-First[Cases[equ[[i]],x_]]],
       b=Append[b,0]
]    ];
```

Kako je najveći broj test problema dostupan u MPS formatu, potreban nam je softver za transkripciju iz *MPS* formata u *NB* format. Taj softver je pisan u programskom jeziku DELPHI, i biće opisan u nastavku.

Primer 2.2.4 Razmotrimo poznati test problem pod nazivom *Blend*. Odgovarajući MPS fajl se transformiše u sledeću unutrašnju reprezentaciju:

```
3.2x1+2.87x2+.0044x8+.07x13+.0378x14+.155x15+.155x16+.155x17+.155x18+.0528x19+
.0528x20+.0528x21+.0528x22+.128x23+.118x24+.0924x36-5.36x37+.0924x44-5.08x45+
.0924x52-4.51x53-2.75x57-4.2x60-3.6x63-2.x70+.04x77+3.x79+.4x80+.0132x82+.01x83,
{-.2931x2+x4==0,-.537x1+x3==0,-.131x1-.117x2+x22+x62==0, -.1155x1-.0649x2+x17
+x21+x61==0, -.0365x1-.1233x2+x15+x19+x67==0, -.143x1-.2217x2+x16+x20+x66==0,
-.037x1+x64==0, -.18x2+x65==0, -.0277x3-.0112x4-.175x5-.271x6-.2836x7-.0571x24
-x25-x54+x55+x74==0, -.0563x3-.0378x4-.27x5-.3285x6-.3285x7-.0185x19-.0185x20
-.0184x21-.0184x22-.0114x24-x26-x27+x35+x43+x51+x56+x73-.5x79==0,
-.199x3-.1502x4-.028x5-.0255x6-.0241x7+x28+x38+x46+x58==0,
-.6873x3-.7953x4+x8+x59==0, -.017x3-.0099x4-.455x5-.2656x6-.2502x7-.0568x19
-.0568x20-.0564x21-.0564x22 +.76x23+.6571x24+x27+x72-.5x79==0,
-x8+x11+x12==0,-x9-x11+x13==0,-x10-x12+x14==0,x7-.0588x13==0,
x6-.0404x14==0,-.8145x13+x30+x40+x48==0,-.8564x14+x31+x41+x49==0,
x5-.3321x15-.3321x16-.2414x17-.2414x18==0,
x9+x10-.5875x15-.5875x16-.6627x17-.6627x18==0,
-.0091x13-.0069x14-.362x15-.362x16-.293x17-.293x18+x29+x39+x47==0,
-.0806x19-.0806x20-.078x21-.078x22+.5714x24+x26+x76==0,
-.0658x19-.0658x20-.0655x21-.0655x22+.5714x23+x25+x75==0,
-.0328x19-.0328x20-.0303x21-.0303x22+x54==0,
-.4934x19-.4934x20-.475x21-.475x22+x32+x42+x50==0,
x18-.2922x19-.2922x20-.305x21-.305x22+x69==0,
-.0096x19-.0096x20+x68==0,-x23+x33==0,-x24+x34==0,
-x28-x29-x30-x31-x32-x33-x34-x35+x37==0,-x38-x39-x40-x41-x42-x43+x45==0,
-x61-x62+x63==0,-x64-x65-x66-x67-x68-x69+x70==0,
-x46-x47-x48-x49-x50-x51+x53==0,-x55-x56+x57==0, -x58-x59+x60==0,
.0042x2+.0327x8-.8239x13-.7689x14+2.3x15+2.3x16+2.3x17+2.3x18+x71==0,
.003x1+.003x2+.01303x3+.01303x4+.01303x5+.01303x6+.01303x7+.0081x13+
.0063x14-.0654x19-.0654x20-.0654x21-.0654x22+.1869x23+.1724x24-x77+x78==0,
.0587x1+.1053x2+.0506x3+.0448x4+.0506x5+.0506x6+.0506x7+.094x8-.
2112x13-.156x14-.2049x15-.2049x16-.1531x17-.1531x18-.2535x19-.2535x20-
.2703x21-.2703x22+.2796x23+.2579x24-.325x71-4.153x72-4.316x73-3.814x74-
3.808x75-4.44x76+1.42x77-x80+x81==0,
.15x1+.185x2+.209x3+.185x4+.209x5+.209x6+.209x7+.045x8+.387x13+.297x14
+.826x15+.826x16+.826x17+.826x18+.632x19+.632x20+.632x21+.632x22+2.241x23
+2.067x24-x82==0,  .302x1+.384x2+.495x3+.721x4+.495x5+.495x6+.495x7
+.793x8+1.03x13+.792x14+14.61x15+14.61x16+14.61x17+14.61x18+.6807x19
+.6807x20+.6807x21+.6807x22+2.766x23+2.552x24-x83==0,
-7.95x28-8.84x29-9.43x30-9.03x31-9.23x32-9.4x33-9.74x34-9.74x35-.493x36
+10.03x37<=0,  -8.7x28-9.45x29-9.57x30-9.32x31-9.22x32-9.85x33-10.1x34
-9.9x35-.165x36+10.03x37<=0, -3.x28-3.x29-3.x30-3.x31-3.x32-3.x33-3.x34
-3.x35+x36<=0, 14.x28+12.x29+3.5x30+3.5x31+6.x32+2.5x33+3.3x34+66.x35
-9.5x37<=0, x28+x29+.233x30+.205x31+.381x32+.39x33+.233x34+x35-.5x37<=0,
-x28-x29-.358x30-.333x31-.509x32-.77x33-.58x34-x35+.5x37<=0,
-.00862x2-7.98x38-8.87x39-9.46x40-9.06x41-9.26x42-9.77x43-.435x44+9.65x45<=0,
-.00862x2-8.58x38-9.33x39-9.45x40-9.2x41-9.13x42-9.78x43-.208x44+9.65x45<=0,
-3.x38-3.x39-3.x40-3.x41-3.x42-3.x43+x44<=0,
14.x38+12.x39+3.5x40+3.5x41+6.x42+66.x43-9.5x45<=0,
x38+x39+.233x40+.205x41+.318x42+x43-.5x45<=0,
-x38-x39-.358x40-.333x41-.509x42-x43+.5x45<=0,
-.0101x2-7.99x46-8.88x47-9.47x48-9.07x49-9.27x50-9.78x51-.426x52+9.05x53<=0,
-.0101x2-8.59x46-9.34x47-9.46x48-9.21x49-9.14x50-9.79x51-.204x52+9.05x53<=0,
-3.x46-3.x47-3.x48-3.x49-3.x50-3.x51+x52<=0,
14.x46+12.x47+3.5x48+3.5x49+6.x50+66.x51-9.5x53<=0,
x46+x47+.233x48+.205x49+.318x50+x51-.5x53<=0,
-x46-x47-.358x48-.333x49-.509x50-x51+.5x53<=0,
```

```
10.1x64+12.63x65+8.05x66+6.9x67+8.05x68+4.4x69-10.1x70<=0,
-14.x58-.8x59+2.x60<=0,14.x58+.8x59-3.x60<=0,
x3+x4+x15+x17+x19+x21+x22+x63+x67<=23.26,
x16+x20+x66<=5.25, x1<=26.32,x2<=21.05,
1.3x13+x14<=13.45,x15+x16+x17+x18<=2.58,x19+x20+x21+x22<=10.,
x23+x24<=10.,.64x37-.36x45-.36x53<=0,.35x37+.35x45-.65x53<=0}
```

Početna tačka algortima

Prvi problem u implementaciji primal-dual algoritma je izbor početne tačke [2, 112, 118]. Poznato je da tačke koje su relativno blizu optimalnog rešenja ali nisu dobro centrirane, često dovode do numeričkih problema. Rezultat [112] Teorema 6.1, garantuje opštu konvergenciju primal-dual algoritma iz bilo koje startne tačke (x^0, λ^0, s^0) pri čemu je $(x^0, s^0) > 0$. Mehrotra [70] predlaže rešavanje jednog problema kvadratnog programiranja za dobijanje početne tačke. Heuristika za određivanje (x^0, λ^0, s^0) u prvom koraku računa (x^*, λ^*, s^*) kao rešenje problema:

$$\begin{aligned} &\min_{x} |x|^2 \quad \text{pod uslovom } Ax = b, \\ &\min_{(\lambda,s)} |s|^2 \quad \text{pod uslovom } A^T\lambda + s = c. \end{aligned} \tag{2.2.1}$$

To znači, x^* i s^* su vektori najmanje norme za koje su rezidumi r_b i r_c jednaki nuli. Startna tačka se zatim određuje sa

$$(x^0, \lambda^0, s^0) = (x^* + \delta_x e, \lambda^*, s^* + \delta_s e),$$

gde su skalari δ_x i δ_s određeni tako da je $(x^0, s^0) > 0$. Vreme potrebno za određivanje početne tačke na taj način je približno jednako vremenu potrebnom za jednu iteraciju metoda unutrašnje tačke [2].

Sada predlažemo dva jednostavna algoritma za određivanje početne tačke. Prvi algoritam određuje tačku koja je dobro centrirana i istovremeno zadovoljava bar jedno ograničenje, ukoliko je to moguće. Pored toga, predloženi algoritmi generišu početnu tačku desetak puta brže od jedne iteracije metoda unutrašnje tačke. Efikasnost predloženih heuristika je proverena na velikom broju test primera. Napomenimo da su rezultati iz ovog odeljka su objavljeni u radovima [75, 76]. Rezultati iz tih radova su citirani u [20, 21, 22] gde se napominje da je izbor početne tačke primenom naše heuristike dao bolje numeričke performanse od početne tačke generisane sa (2.2.1).

Algoritam PocTacka

Korak 1.1. Izračunati vrednost $aux = \max\{q_1, \ldots, q_m\}$, gde je

$$q_i = \begin{cases} \max\left\{\dfrac{b_i}{\sum_{j=1}^n a_{ij}}, 1\right\}, & \sum_{j=1}^n a_{ij} \neq 0; \\ 1, & \sum_{j=1}^n a_{ij} = 0,\ i = 1, \ldots, m. \end{cases} \tag{2.2.2}$$

Korak 1.1.a

Motivisani radom [52], generišemo alternativnu početnu tačku, koja se dobija na sličan način kao i tačka u *Koraku 1.1* i *Koraku 1.2*, koristeći vrednost

$aux = \max\{q_1, \ldots, q_m\}$, gde je

$$q_i = \begin{cases} \max\left\{ \dfrac{b_i}{\sqrt{\sum_{j=1}^{n} a_{ij}^2}}, 1 \right\}, & \sum_{j=1}^{n} a_{ij}^2 \neq 0; \\ 1, & \sum_{j=1}^{n} a_{ij}^2 = 0, \ i = 1, \ldots, m. \end{cases}$$

```
inicp[a_,b_]:=
  Module[{i,aux,k,m,pl={}},
    k=Last[Dimensions[a]];      m=First[Dimensions[a]];
    For[i=1,i<=m,i++,
        aux=N[Sum[a[[i,j]],{j,k}],20];
        If[aux !=0,
           AppendTo[pl,Max[b[[i]]/aux,20],1]]
    ]   ];
    If[pl=={}, aux=1, aux=Max[pl] ];
    Return[pom]
  ]
```

Korak 1.2. Generisati startne tačke x i s, čije su koordinate jednake aux, i startnu tačku l, čije su koordinate jednake nuli.

```
pi=inicp[a,b];
x=Table[pi,{k+q}];      s=x;     l=Table[0,{m}];
```

Pomoćne rutine

Korak 6 je implementiran u sledećoj funkciji *solv1*. Parametar x označava k-tu iteraciju x^k, i parametar dx označava vektor Δx^{aff}. Moguća su sledeća dva slučaja:

A. Ako je $dx[[i]] > 0$ za svako i, tada je $\alpha_{aff}^{pri} = 1$.

B. Ako postoji najmanje jedno i tako da je $dx[[i]] < 0$, stavimo

$$\alpha_{aff}^{pri} = \min\{Abs[x[i]/dx[i]], \ i = 1, \ldots, n\}.$$

```
solv1[x_,dx_]:=
  Module[{i,a=1,al={}},
    For[i=1,i<=First[Dimensions[x]],i++,
       If[dx[[i]]<0,
          al=Append[al,N[Abs[x[[i]]/dx[[i]]],20]]
    ]  ];
    If[Min[al]<1,a=Min[al]];
    Return[a]
  ]
```

Slična funkcija *solv2* može biti iskorišćena u implementaciji *Koraka 10.*

```
solv2[x_,dx_]:=
```

```
Module[{i,a=Abs[Max[]],al={}},
  For[i=1,i<=First[Dimensions[x]],i++,
      If[dx[[i]]<0,
         al=Append[al,Abs[x[[i]]/dx[[i]]]]
  ]   ];
  a=Min[al];    Return[a]
]
```

Glavni algoritam

Glavna funkcija koristi sledeće formalne parametre:
*objective_*: izraz predstavlja ciljnu funkciju.
*constraints_*: je lista datih ograničenja.
*eps_*: izabrana tačnost.
*iter_*: maksimalan broj iteracija.

Korak 1:

```
pi=inicp[a,b];
x=Table[pi,{k+q}];      s=x;  l=Table[0,{m}];
```

Korak 2:

```
at=Transpose[a];
rb=N[a.x-b];
rc=N[at.l+s-ce];
While[(Abs[ce.x-b.l]/(1+Abs[ce.x])>eps),
```

Korak 3:

```
  id=Table[1,{k+q}];
  Xmat=DiagonalMatrix[x];   Smat=DiagonalMatrix[s];
  rxs=Xmat.Smat.id;
```

Korak 4:
Koristimo sledeće pojednostavljenje:

$$S^{-1} = \mathrm{diag}(1/s[[1]], \ldots, 1/s[[k+q]]),$$
$$D^2 = \mathrm{diag}(x[[1]]/s[[1]], \ldots, x[[k+q]]/s[[k+q]]).$$

Sistem u *Koraku 4* može biti rešen korišćenjem MATHEMATICA/ funkcije *LinearSolve*.

```
  is=DiagonalMatrix[1/s];  d=DiagonalMatrix[x/s];
  Dlaff=LinearSolve[a.d.at, -rb-a.(is.Xmat.rc-is.rxs)ad ];
  Dsaff=N[-rc-at.Dlaff,20];
  Dxaff=N[-is.(rxs+Xmat.Dsaff),20];
```

Korak 5:

```
  mi=(x.s)/(k+q);
```

Korak 6:

```
AlfaPa=solv1[x,Dxaff];   AlfaDa=solv1[s,Dsaff];
```

Korak 7:

```
SigCent=N[((((x+AlfaPa*Dxaff).(s+AlfaDa*Dsaff))/(k+q))/mi)^3,20];
```

Korak 8:
Za rešavanje sistema u *Koraku 8* ponovo koristimo funkciju *LinearSolve*.

```
DXaff=DiagonalMatrix[Dxaff];   DSaff=DiagonalMatrix[Dsaff];
rxs=N[-mi*SigCent,20]*id+N[DXaff.DSaff,20].id;
Dlcc=LinearSolve[a.d.at,  a.is.rxs]  ];
Dscc=N[-at.Dlcc,20];
Dxcc=N[-is.(rxs+Xmat.Dscc),20];
```

Korak 9:

```
dx=Dxaff+Dxcc;  dl=Dlaff+Dlcc;   ds=Dsaff+Dscc;
```

Korak 10 i *Korak 11:*

```
AlfaPa=N[Min[0.99*solv2[x,dx],1],20];
AlfaDa=N[Min[0.99*solv2[s,ds],1],20];
```

Korak 12:

```
   x=N[x+AlfaPa*dx,20];l=N[l+AlfaDa*dl,20];s=N[s+AlfaDa*ds,20];
  (* Generate conditions for termination of the cycle:  *)
     rb=N[a.x-b,20];
     rc=N[at.l+s-ce,20];
  ]; (* End of the While cycle *)
  Return[{N[z,20],Take[N[x,20],k]}];
]
```

2.3 Translacija MPS fajla u NB fajl

Većina dostupnih test problema je data u MPS formatu čija je glavna karakteristika zapis matrice problema po kolonama. Cilj nam je da takav format prevedemo u mnogo prirodniji i korisnicima dostupniji NB format koji koristi MATHEMATICA i u kome se problem zapisuje na klasičan način tako da ne zahteva nikakvo posebno predznanje od korisnika. Za taj cilj koristimo programsko okruženje Delphi.

Sažet opis *MPS* formata je dostupan na *ftp://softlib.cs.rice.edu/pub/miplib*. U *MPS* fajl formatu postoji šest sekcija: *NAME, ROWS, COLUMNS, RHS, RANGES, BOUNDS* dok je sedma upravljačka reč *ENDATA*.

Sledi kratak opis tih sekcija:

NAME je prva sedkcija koja se sastoji od naredbe *NAME* i samog imena fajla koje počinje od pete kolone.

U sekciji *ROWS* dat je tip i ime svake vrste matrice ograničenja. Tip vrste (nejednakosti ili jednakosti) dat je u drugoj koloni a ime se nalazi između pete i dvanaeste kolone. Tip kolone je definisan na sledeći način:

tip	značenje	tip	značenje
E	jednakost	N	ciljna funkcija
L	manje ili jednako	N	bez ograničenja
G	veće ili jednako		

U sekciji *COLUMN*, definisana su imena promenljivih zajedno sa koeficijentima ciljne funkcije i nenula elementima matricwe ograničenja. Postoje sledeća polja:

polje	kolona	polje	kolona
ime kolone	5 - 12		
ime vrste	15 - 22	ime vrste	40 - 47
vrednost	25 - 36	vrednost	50 - 61

Poslednja dva polja su opciona.

U *RHS* sekciji su date informacije o slobodnim koeficijentima ograničenja datog problema.

polje	kolona	polje	kolona
rhs ime	5 - 12		
ime vrste	15 - 22	ime vrste	40 - 47
vrednost	25 - 36	vrednost	50 - 61

Poslednja dva polja su opcija kao i kod sekcije *COLUMNS*.

Sekcija *RANGES* je opciona i ređe se koristi, tako da se na njoj nećemo posebno zadržavati.

Sekcija *BOUNDS* je takođe opciona, ali je delimično opisujemo. U toj sekciji se opisuju ograničenja promenljivih.

polje	kolona
tip	2 - 3
ime ograničenja	5 - 12
ime kolone	15 - 22
vrednost	25 - 36

U slučaju da ograničenja nisu data, po definiciji se pretpostavlja($0 \leq x \leq \infty$).

Polje *type* označava tip ograničenja:

tip	značenje		tip	značenje	
LO	donja granica	$x >= b$	FR	slobodna promenljiva	
UP	gornja granica	$x <= b$	MI	donja granica -∞	-$\infty < x$
FX	fiksna promenljiva	$x = b$	BV	binarna promenljiva	$x = 0$ ili 1

U ovoj verziji programa konvertujemo *LO, UP* i *FX* tip ograničenja.

Dajemo *Afiro* problem zapisan u *MPS* formatu radi ilustracije (deo problema je izostavljen zbog veličine fajla):

```
NAME          AFIRO
ROWS
 E  R09
 E  R10
 L  X05
 .
 .
 .
 L  X50
 L  X51
 N  COST
COLUMNS
    X01       X48              .301   R09              -1.
    X01       R10             -1.06   X05               1.
    X02       X21              -1.    R09               1.
    X02       COST             -.4
    X03       X46              -1.    R09               1.
    .
    .
    .

    X36       COST            -.48
    X37       X49              -1.    R23               1.
    X38       X51               1.    R22               1.
    X39       R23               1.    COST             10.
RHS
    B         X50             310.    X51              300.
    B         X05              80.    X17               80.
    B         X27             500.    R23               44.
    B         X40             500.
ENDATA
```

Generalna ideja je: otvoriti *MPS* fajl i prevoditi ga red po red u odgovarajući *NB* format.

U programu se koriste sledeće konstante i promenljive:

```
const
      S0: PChar = 'NAME';     S1: PChar = 'ROWS';
      S2: PChar = 'COLUMNS';  S3: PChar = 'RHS';
      S4: PChar = 'RANGES';   S5: PChar = 'BOUNDS';
      S6: PChar = 'ENDATA';
var
      I, J, K, L, NumVar, NumActualVar: integer;
      S,Name,LinesType,ActualVar,ActualLin,ActualNum,BoundsType:string;
      E, F: TextFile;
      LinesName, Lines, Vars: array of string;
      BoundsM: boolean;
```

Najvažnije su: *LinesName, Lines, Vars*, promenljive tipa dinamički niz. Dinamički niz nema fiksiranu veličinu ili dužinu. Memorija za dinamički niz se dodeljuje kada je data vrednost niza ili se prelazi na proceduru *SetLength*. Kada se dinamički niz zapamti u memoriji, moguć je prelaz na standarne funkcije *Length, High*, and *Low*. Kada se otvara *MPS* fajl i čita red po red, nije nam poznato koliko tačno ograničenja i promenljivih imamo. Ako se ne koristi dinamički niz, potrebno je pročitati fajl nekoliko puta što bi znatno usporilo program.

Promenljiva *LinesName* predstavlja oblast imena vrsta u *ROWS* sekciji. String oblast *Lines* predstavlja ciljnu funkciju i ograničenja. Kada kompletiramo oblast *Lines*, ostaje samo da zapišemo string u novi fajl. Promenljiva *Vars* je niz imena kolona u sekciji *COLUMN*.

Sekcija *NAME* se jednostavno konvertuje sledećim kodom:

```
AssignFile(F, Edit1.Text); Reset(F);
Read(F,S); //  Read the problem name
Name := '(* '+S+' *)';
//  and incorporate it into the MATHEMATICA comment
Readln(F);
```

Zatim se iz *ROWS* sekcije izdvajaju promenljive *LinesType* i *LinesName*. *Lines Type* je string koji predstavlja tip vrste u *ROWS* sekciji. To se postiže sledećim kodom:

```
Read(F,S);
I := 1;   //  I denotes the number of rows (constraints)
while (StrLIComp(PChar(S), S2, 7) <> 0) do
 begin    //  Cycle terminates in the case S='COLUMNS'
   SetLength(LinesType, I);   SetLength(LinesName, I);
   LinesType[I] := S[2];  //  Remember row type
   //  Extract row name, deleting first 4 characters from S
   L := Length(S);
   for J := 1 to (L-4) do S[J] := S[J+4];
   SetLength(S,L-4);      LinesName[I-1] := S;
   Readln(F);  Read(F,S);
   I := I + 1;
 end;
I := I - 1;
SetLength(Lines, I);
```

Zatim se upisuje *COLUMNS* sekcija. Glavna petlja je:

```
Readln(F);  Read(F,S);
NumVar := 0;
while (StrLIComp(PChar(S), S3, 3) <> 0) do
  begin  //  Cycle terminates in the case S='RHS'
    J := 1;
    //  Analysis of the current string S
    Readln(F);   Read(F,S);
```

```
  end;
```

gde je *NumVar* broj promenljivih u *MPS* fajlu. Napomenimo da promenljive u *MPS* formatu mogu imati razna imena, ali ih mi menjamo u standarni oblik $x1, x2, \ldots$ Promenljiva J predstavlja trenutnu poziciju u stringu koji se razmatra. U sledeća četiri koraka opisujemo proces analize tekućeg stringa S.

Korak 1. Promenljiva *ActualVar* predstavlja ime tekuće kolone u *COLUMNS* sekciji. To ime izdvajamo iz tekuće kolone na sledeći način:

```
L := 1;
SetLength(ActualVar, Length(S));
while S[J] = ' ' do J := J + 1;
while S[J] <> ' ' do
  begin
    ActualVar[L] := S[J];     L := L + 1;  J := J + 1;
  end;
SetLength(ActualVar, L-1);
```

Korak 2. Ukoliko je promenljiva (ime kolone) *ActualVar* u *Vars* oblasti, moramo da nađemo njen indeks; ukoliko nije, moramo da je ubacimo u tu oblast. Napomenimo da u nastavku promenljiva *NumActualVar* predstavlja indeks promenljive u oblasti *Vars*.

```
K := 0;
while (K < NumVar) do
  if(StrLIComp(PChar(Vars[K]),PChar(ActualVar),Length(ActualVar))<>0)
      then K:= K + 1
   else break;
if K >= NumVar then
   begin  // Put the column name in array Vars
     NumVar := NumVar + 1;
     SetLength(Vars, NumVar);
     Vars[NumVar-1] := ActualVar;
   end;
NumActualVar := K;
```

Korak 3. Iz *COLUMNS* sekcije preuzimamo *row name* i *value* polja, plus opciono *row name* i *value* polja.

```
while (J < Length(S))do
 begin
   // Finding the content of the row name field
   // and putting it in the ActualLin variable
   while S[J] = ' ' do J := J + 1;
   SetLength(ActualLin, Length(S));  L:=1;
   while S[J] <> ' ' do
     begin
       ActualLin[L] := S[J];  L := L+1;  J := J+1;
     end;
   SetLength(ActualLin, L-1);
```

```
    // Finding the index K of ActualLin in the LinesName array
    K := 0;
    while(StrLIComp(PChar(LinesName[K]),PChar(ActualLin),
                    Length(ActualLin))<>0)
          do K := K + 1;
    // Finding the content of the value field
    // and putting it in the ActualNum variable
    while S[J] = ' ' do J := J + 1;
    SetLength(ActualNum, Length(S));   L:=1;
    while((S[J] <> ' ') and (J <= Length(S))) do
      begin
        ActualNum[L]:=S[J];  L:=L+1;  J:=J+1;
      end;
    SetLength(ActualNum, L-1);
    // Several simplifications
    if((ActualNum[1]<>'+') and (ActualNum[1]<>'-') and (Lines[K]<>''))
        then  ActualNum := '+' + ActualNum;
    if((ActualNum='1') or (ActualNum='1.')  or (ActualNum='1.0'))
        then ActualNum := '';
    if((ActualNum='+1') or (ActualNum='+1.')  or (ActualNum='+1.0'))
        then ActualNum := '+';
    if((ActualNum='-1') or (ActualNum='-1.')  or (ActualNum='-1.0'))
        then ActualNum := '-';
    // Finally we are forming the constraint
    Lines[K]:=Lines[K]+ActualNum+'x'+IntToStr(NumActualVar+1);
    // Skip whitespace characters in the end of line
    while ((S[J] = ' ') and (J <= Length(S))) do J:=J+1;
end;
```

Korak 4. Sekcija *COLUMNS* je ovim završena. Pre učitavanja *RHS* sekcije, dodajemo ==, <= i >= simbole na kraju linija koja predstavljaju ograničenja:

```
for I := 0 to High(Lines) do
  case LinesType[I+1] of
    'E': Lines[I] := Lines[I] + '==';
    'L': Lines[I] := Lines[I] + '<=';
    'G': Lines[I] := Lines[I] + '>=';
  end;
```

Sekcija *RHS* je vrlo slična sekciji *COLUMNS*. Ona ima *rhs name* polje na početku koje je za nas nebitno, a zatim slede četiri polja od kojih su dva opciona. To je analogno *COLUMNS* sekciji, tako da je kod skoro jednak. Potrebno je dodati vrednost desnoj strani ograničenja uz napomenu da, ako podatak nije dat, vrednost desne strane je jednaka nuli.

```
for I := 0 to High(Lines) do
  if Lines[I][Length(Lines[I])]= '='
      then Lines[I]:=Lines[I]+'0';
```

Sekcija *BOUNDS* je opciona i prevodimo je parcijalno. Prevodimo ograničenja tipa *LO, UP* i *FX*. Sledi kod:

```
readln(F);  read(F,S);
while (StrLIComp(PChar(S), S6, 6) <> 0) do
 begin //  Cycle terminates in the case S='ENDATA'
    //  Finding bound type field
    SetLength(BoundsType, 2);
    BoundsType[1] := S[2];   BoundsType[2] := S[3];
    J:=4;  //  Range name starts at fifth position
    //  Skipping bound name field
    while S[J] = ' ' do J := J + 1;

    //  Finding the content of the column name field
    //  and putting it in the ActualVar variable
    //  (like in the examples above)

    //  Finding the index K of the ActualVar in the Vars array
    K := 0;
    while(StrLIComp(PChar(Vars[K]),PChar(ActualVar),Length(ActualVar))<>0)
          do K := K + 1;

    //  Finding the content of the value field
    //  and putting it in the ActualNum variable
    //  (like in the examples above)

    // Finally we are forming the new constraint.
    ActualLin := '';
    if BoundsType='LO' then
       ActualLin:='x'+IntToStr(K+1)+'>='+ActualNum;
    if BoundsType='UP' then
       ActualLin:='x'+IntToStr(K+1)+'<='+ActualNum;
    if BoundsType='FX' then
]       ActualLin:='x'+IntToStr(K+1)+'=='+ActualNum;
    if (ActualLin <> '') then
       begin
         L := Length(Lines);
         SetLength(Lines, L+1);
         Lines[L] := ActualLin;
       end;
    Readln(F);       Read(F,S);
 end;
```

Time su formirani ciljna funkcija i sva ograničenja. Na kraju je potrebno zapamtiti oblast *Lines* kao novi fajl:

```
try
```

```
    AssignFile(E, Edit2.Text);   Rewrite(E);
    //  Objective function must be saved first
    K := 1;
    while LinesType[K] <> 'N' do K := K + 1;
    Writeln(E, WrapText(Lines[K-1]+',', #13#10, ['+','-'], 80));
    Writeln(E);   Writeln(E, Name);    Writeln(E);
    Write(E, '{');
    //  Saving all other element of the array Lines
    //  which represent the constraints
    if ((K-1) = High(Lines)) then
    begin   //  If selected line is last
      for I:=0 to (K-3) do
        Writeln(E,WrapText(Lines[I]+',',#13#10,['+','-'],80));
      Write(E,WrapText(Lines[K-2]+'}',#13#10,['+','-'],80));
    end
    else   //  Otherwise
    begin
      for I := 0 to (K-2) do
        Writeln(E,WrapText(Lines[I]+',',#13#10,['+','-'],80));
      for I := K to (High(Lines)-1) do
        Writeln(E,WrapText(Lines[I]+',',#13#10,['+','-'],80));
      Write(E,WrapText(Lines[High(Lines)]+'}',#13#10,['+','-'],80));
    end;
    CloseFile(E);
  end;
```

Napomenimo da neki *MPS* fajlovi mogu poticati pre 1995. godine. U tom slučaju program ne može da prepozna "Eoln" simbol koji predstavlja kraj linije u tekstualnom fajlu i program ne radi ništa. U tom slučaju je potrebno otvoriti fajl u nekom tekst editoru, NotePadu ili Wordu na primer, a zatim ponovo zapamtiti. Novi fajl tada ima korektne "Eoln" simbole i program sada radi ispravno.

2.4 Numerički primeri

U prvom primeru upoređujemo našu implementaciju sa nekoliko poznatih programa pisanih u različitim programskim jezicima, dok u drugom primeru ispitujemo uticaj perturbacije problema na stabilnost rešenja.

Primer 2.4.1 U ovom primeru pokazujemo visoke numeričke mogućnosti naše implementacije. Za test problem *Afiro* početna tačka ima sve koordinate jednake 500. Uobičajena tačnost u literaturi je 10^{-8}. U ovom slučaju koristimo tačnost 10^{-14}. Vrednosti mere dualnosti $x.s$ generisane tokom iteracija su

$2.27947460213494279\times10^{6}$, 296359., 3921.18, 613.909, 173.229, 42.61,
8.59576, 0.124154, 0.00124166, 0.0000124166, $1.24172039912096354\times10^{-7}$,
$1.24339209599189026\times10^{-9}$, $1.40714556006783997\times10^{-11}$, $1.39058129869647274\times10^{-12}$

i rešenje je

Out[1]= {-464.7531428571419,{$3.19975433444599843\times10^{-13}$,79.9999999999997424,

1.91244772496984012$\times 10^{-14}$,6.39852710910254973$\times 10^{-14}$,6.33933146702376237$\times 10^{-14}$,
5.77426057986956386$\times 10^{-15}$",18.2142857142854701,51.4971318189389215,
73.8941025852179755,25.5000000000002913,499.999999999995914,475.919999999998832,
24.0800000000000169,9.24725339363984843$\times 10^{-13}$,214.999999999999102,
147.740206132740929,3.5320429670772806$\times 10^{-14}$,54.4999999999994688,
3.51617861969817147$\times 10^{-14}$,3.49915810007669492$\times 10^{-14}$,2.05287088556072161$\times 10^{-14}$,
6.92504277492309583$\times 10^{-14}$,6.86331960286415032$\times 10^{-14}$,6.80075262615289499$\times 10^{-14}$,
339.942857142856658,236.202651010115571,63.5282886370786048,
5.32920502835706955$\times 10^{-15}$,84.7999999999996312,69.7114175332243757,
3.19791251290398559$\times 10^{-14}$,3.16007552845632666$\times 10^{-14}$,4.1377910630115311$\times 10^{-14}$

Za problem *Blend* smo postigli relativnu tačnost 10^{-13}, pri čemu su vrednosti mere dualnosti $x.s$ jednake

4329.02, 2295., 935.694, 73.9983, 12.7999, 3.18528,0.8102, 0.124219, 0.00416878,
0.0000455071, 4.55077$\times 10^{-7}$, 4.55079$\times 10^{-9}$, 4.55154$\times 10^{-11}$,4.55699$\times 10^{-13}$. a minimalna vrednost je -30.81214984582697.

Za problem *Adlittle* sa relativnom tačnošću 10^{-14} dobijamo minimalnu vrednost 225494.96316238158 i sledeće vrednosti za meru dualnosti $x.s$:

3.68314$\times 10^{6}$, 1.54606$\times 10^{7}$, 2.48261$\times 10^{7}$, 9.49634$\times 10^{6}$, 1.12582$\times 10^{6}$, 207828,
64916.2, 25548.8, 4728.11, 599.321,95.7551, 8.25261, 0.089176, 0.000891768.,
8.9178725141779207$\times 10^{-6}$, 8.92709050417012406$\times 10^{-8}$, 1.13837804301388501$\times 10^{-9}$,
3.77892830854748851$\times 10^{-10}$, 2.29911431090509088$\times 10^{-10}$,2.2951933148389334$\times 10^{-10}$,
2.0338064157382627$\times 10^{-10}$, 5.17798147396104369$\times 10^{-11}$.

Napomenimo da kod PCx daje minimalnu vrednost 2.25494963×10^5, koja sadrži samo tri decimalne cifre. U sledećoj tabeli dajemo rezultate dobijene primenom naše implementacije pod nazivom MPD na nekim $Netlib$ test problemima. Napomenimo da se rezultati podudaraju sa rezultatima iz [11] i slažu sa rezultatima iz [71] u skoro svim slučajevima.

Problem	*eps*	MPD optimalna vrednost	Br.it.	Najmanje $x.s$	Tačnost
Adlittle	10^{-14}	225494.96316238158	18	2.207476×10^{-11}	5.420766×10^{-15}
Afiro	10^{-14}	-464.753142857419	13	6.882627×10^{-12}	5.247988×10^{-15}
Agg	10^{-10}	-3.599176728426037$\times 10^{7}$	23	0.002223387	8.312113×10^{-11}
Agg2	10^{-12}	-2.0239252355976876$\times 10^{7}$	23	1.391321×10^{-6}	5.926817×10^{-14}
Agg3	10^{-12}	1.0312115935089154$\times 10^{7}$	21	2.253792×10^{-7}	5.599432×10^{-15}
Bandm	10^{-10}	-158.6280184446234	19	1.064918×10^{-8}	6.655994×10^{-11}
Blend	10^{-13}	-30.81214984582697	13	4.55918×10^{-13}	1.116779×10^{-14}
Degen2	10^{-8}	-1435.177999919273	11	6.52524$\times 10^{-7}$	1.81695$\times 10^{-8}$
Israel	10^{-12}	-896644.8218630119	25	3.582831×10^{-8}	5.699723×10^{-14}
Kb2	10^{-14}	-1749.900129906283	19	2.586536×10^{-13}	3.041345×10^{-13}
Lotfi	10^{-12}	-25.26470606187774	20	6.501900×10^{-12}	2.164251×10^{-13}
Recipe	10^{-12}	-266.6159999999579	14	7.790105×10^{-11}	2.480909×10^{-13}
Sc105	10^{-13}	-52.20206121170623	14	3.097975×10^{-12}	4.888131×10^{-14}
Sc205	10^{-13}	-52.20206121170889	18	1.101846×10^{-14}	3.111843×10^{-14}
Sc50a	10^{-13}	-64.5750770585606	13	1.58378×10^{-13}	6.024572×10^{-14}
Sc50b	10^{-14}	-69.9999999999994	12	1.637435×10^{-13}	1.601223×10^{-15}
Scagr7	10^{-13}	-2.3313898243313123$\times 10^{6}$	16	3.031844×10^{-8}	8.868252×10^{-14}
Scfxm1	10^{-8}	18416.759034153343	20	0.00001697	8.904402×10^{-10}
Sctap1	10^{-12}	1412.2500000000337	17	1.011259×10^{-13}	1.378802×10^{-13}
Share2b	10^{-12}	-415.73224071836	14	1.011259×10^{-13}	9.996957×10^{-13}
Stocfor1	10^{-13}	-41131.97621943486	17	1.436246×10^{-10}	4.050751×10^{-14}

Tabela 2.4.1

Najveće dimenzije matrice problema su 444×757 kod primera *Degen*2 i 516×758 kod primera *Agg*2 i *Agg*3.

U sledećoj tabeli dajemo odgovarajuće rezultate dobijene primenom poznatih programa PCx [14], LOQO [105], LIPSOL [123] i MATHEMATICA implementacijom iz [10]. U implementaciji iz [10] koristili smo samo definisanu relativnu tačnost 10^{-4}, zbog ekstremno velikog broja iteracija.

Problem	PCx	LIPSOL	LOQO	BHATTI
Adlittle	2.25494963×10^{5}	2.2549496316×10^{5}	2.2549496×10^{5}	223753
Afiro	-4.64753143×10^{2}	$-4.6475314219\times10^{2}$	-4.6475314×10^{2}	-464.713
Agg	3.59917673×10^{7}	-3.599176727×10^{7}	-3.5991767×10^{7}	-3.59892×10^{7}
Agg2	-2.02392521×10^{7}	$-2.0239252353\times10^{7}$	-2.0239252×10^{7}	$-2.02715e\times10^{7}$
Agg3	1.03121159×10^{7}	1.0312115935×10^{7}	1.0312116×10^{7}	1.02797×10^{7}
Bandm	-1.58628018×10^{2}	$-1.5862801844\times10^{2}$	-1.5862802×10^{2}	-158.628
Blend	-3.08121498×10^{1}	$-3.0812149846\times10^{1}$	-3.0812150×10^{1}	-30.7843
Degen2	-1.43517800×10^{3}	$-1.4351780000\times10^{3}$	-1.4351780×10^{3}	-1435.17
Israel	-8.96644817×10^{5}	$-8.9664482186\times10^{5}$	-8.9664465×10^{5}	-896617
Kb2	-1.74990013×10^{3}	$-1.7499001299\times10^{3}$	-1.7499001×10^{3}	-1749.89
Lotfi	$-2.5264706062\times10^{1}$	$-2.5264706056\times10^{1}$	-2.5264702×10^{1}	-25.2551
Recipe	-2.66616000×10^{2}	$-2.6661600000\times10^{2}$	-2.6661600×10^{2}	–
Sc105	$-5.2202061212\times10^{1}$	$-5.2202061212\times10^{1}$	-5.2202060×10^{1}	-52.1908
Sc205	-5.22020612×10^{1}	$-5.2202061119\times10^{1}$	-5.2202061×10^{1}	-52.1876
Sc50a	$-6.4575077059\times10^{1}$	$-6.4575077059\times10^{1}$	-6.4575077×10^{1}	-64.5683
Sc50b	-7.000000000×10^{1}	$-6.9999999976\times10^{1}$	-7.00000006×10^{1}	-69.993
Scagr7	-2.33138982×10^{6}	$-2.3313898243\times10^{6}$	-2.3313898×10^{6}	-2.33139×10^{6}
Scfxm1	1.84167598×10^{4}	1.8416759028×10^{4}	1.8416759×10^{4}	18416.7
Sctap1	1.41225000×10^{3}	1.4122500000×10^{3}	1.4122500×10^{3}	1412.26
Share2b	$-4.1573224074\times10^{2}$	$-4.1573224024\times10^{2}$	-4.1573224×10^{2}	-415.732
Stocfor1	$-4.1131976219\times10^{4}$	$-4.1131976219\times10^{4}$	-4.1131976×10^{4}	-41132

Tabela 2.4.2

Napomenimo da program PCx u praksi kod većine problema ne može da postigne preciznost veću od 10^{-11}.

U sedećoj tabeli upoređujemo broj potrebnih iteracija u slučaju kada je zadata tačnost 10^{-8} u svim implementacijama, osim kod implementacije iz [10] gde se koristi tačnost 10^{-4} iz već navedenih razloga. Evidentno je da sve implementacije primal dual algoritma imaju približno isti broj iteracija, osim implementacije iz [10] koja i pored znatno manje preciznosti zahteva daleko veći broj iteracija.

Problem	MPD	PCx	LIPSOL	LOQO	BHATI
Adlittle	13	12	12	16	50
Afiro	9	8	7	13	14
Agg	22	19	18	23	96
Agg2	21	23	16	33	50
Agg3	19	22	16	37	60
Bandm	16	17	17	20	86
Blend	11	10	12	17	14
Degen2	11	12	14	18	16
Israel	23	21	22	33	657
Kb2	15	13	15	18	65
Lotfi	18	15	18	21	82

Problem	MPD	PCx	LIPSOL	LOQO	BHATI
Recipe	10	9	9	14	–
Sc105	12	10	10	15	38
Sc205	15	11	11	17	72
Sc50a	10	8	10	15	22
Sc50b	9	6	7	13	25
Scagr7	14	14	12	18	105
Scfxm1	20	17	18	26	172
Sctap1	15	16	17	22	41
Share2b	11	17	12	16	33
Stocfor1	14	12	17	19	44

Tabela 2.4.3

3 Smanjenje dimenzije u Mehrotraovom algoritmu

U ovom odeljku opisujemo modifikaciju Mehrotraovog primal-dual algoritma. Ta modifikacija smanjuje dimenziju problema, poboljšava stabilnost metoda i eliminiše potrebu za svođenjem rešenja na tačno. Takođe je realizovana implementacija modifikacije u programu MATHEMATICA i dati su numerički primeri za poređenje sa originalnim algoritmom. Ovi rezultati su publikovani u radovima [93, 95, 96, 97].

3.1 Primal-dual algortimi i bazično rešenje

Primal-dual algoritmi razmatraju problem linearnog programiranja sa njegovim dualnim problemom (1.1.1) – (1.1.2). Pri tome se generiše niz koji konvergira ka optimalnom rešenju tako da je potrebno konačno rešenje svesti posebnim postupkom na tačno. Napomenimo da se često u praktičnoj primeni linearnog programiranja rešava niz sličnih problema. Pri tome je prethodno optimalno rešenje kod simpleks algoritma moguće upotrebiti za početno rešenje i tako dobiti novo optimalno rešenje brže i jednostavnije. Za unutrašnje metode takva opšta procedura još uvek ne postoji mada postoje takvi pokušaji [27, 29, 30, 2]. Trenutno usvojen pristup je da se niz sličnih problema prvo reši primenom metoda unutrašnje tačke, a zatim se prelazi na simleks metod. Pri tome se koriste prednosti oba metoda. Algoritam za generisanje optimalnog bazičnog rešenja je dat u [67]. On konstruiše optimalno bazično rešenje u manje od n iteracija startujući iz nekog komplementarnog rešenja; dakle, Megiddov algoritam podrazumeva da je poznato tačno optimalno rešenje. Ta pretpostavka u praksi nikad nije ispunjena jer primal-dual algoritam samo generiše niz koji konvergira ka optimalnom rešenju. Kako se rešenje uvek dobija sa unapred zadatom tačnošću, rešenje nije ni tačno dopustivo ni komplementarno. Postupak konačnog završavanja koji su predložili Andersen i Ye [3], [117] se zasniva na prelasku sa iteracije koja prati centralnu putanju na tačno primal-dual rešenje. Algoritam je uspešan ako je iteracija dovoljno bliska optimalnom rešenju tako da indeksni skupovi $\mathcal{B}$ i $\mathcal{N}$ mogu biti tačno određeni. U suprotnom, algoritam daje tačku

koja ne zadovoljva neko od ograničenja u (1.1.1), (1.1.2) i u tom slučaju se algoritam vraća u primal-dual algoritam da bi se kroz nekoliko koraka ponovo pokušalo sa konačnim završavanjem. Smanjenjem dimenzije problema moguće je pojednostaviti postupak konačnog završavanja. Napomenimo da je naglasak modifikacije na tome da se implemtira tako da se dobije bazično optimalno rešenje. Postupak konačnog završavanja je vrlo detaljno razmatran u velikom broju radova [3, 69, 107, 112, 117].

3.2 Redukcija dimenzija problema linearnog programiranja

Deo rezultata iz ovog poglavlja publikovan je u [93].

U nastavku razmatramo problem linearnog programiranja u opštem obliku

$$\begin{aligned} \text{minimizirati} \quad & c_1x_1 + \cdots + c_kx_k \\ \text{pod uslovima} \quad & \sum_{j=1}^{k} a_{ij}x_j \le b_i, \quad i = 1, \ldots, q, \\ & \sum_{j=1}^{k} a_{ij}x_j = b_i, \quad i = q+1, \ldots, m, \\ & x_i \ge 0, \quad i = 1, \ldots, k. \end{aligned} \tag{3.2.1}$$

Standardna forma za (3.2.1) je (sa $k + q = n$)

$$\begin{aligned} \text{minimizirati} \quad & c_1x_1 + \cdots + c_kx_k + c_{k+1}x_{k+1} + \cdots + c_nx_n \\ \text{pod uslovima} \quad & \sum_{j=1}^{k} a_{ij}x_j + x_{k+i} = b_i, \quad i = 1, \ldots, q, \\ & \sum_{j=1}^{k} a_{ij}x_j \qquad\quad = b_i, \quad i = q+1, \ldots, m, \\ & x_i \ge 0, \quad i = 1, \ldots, n. \end{aligned} \tag{3.2.2}$$

Problem (3.2.2) je oblika (1.1.1), gde je $n = k + q$ i

$$c_{k+1} = \cdots = c_{k+q} = 0, \quad A = \begin{bmatrix} a_{11} & \ldots & a_{1k} & 1 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{q1} & \ldots & a_{qk} & 0 & \ldots & 1 \\ a_{q+1,1} & \ldots & a_{q+1,k} & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & \ldots & a_{mk} & 0 & \ldots & 0 \end{bmatrix}. \tag{3.2.3}$$

Dualni problem za (3.2.2) je

$$\begin{aligned} &\text{maksimizirati} \;\; b_1\lambda_1 + \cdots + b_m\lambda_m \\ &\text{pod uslovima} \;\; \sum_{j=1}^{m} a_{ji}\lambda_j = c_i, \;\; i = 1, \ldots, k \\ &\qquad\qquad \lambda_j + s_{k+j} = 0, \;\; j = 1, \ldots, q \\ &\qquad\qquad\qquad\qquad s_i \geq 0, \;\; i = 1, \ldots, k+q. \end{aligned} \tag{3.2.4}$$

Primetimo da ako je $x_i = 0$ za neko i, tada to x_i nema uticaja na konačno rešenje, tako da i-ta kolona iz matrice A može biti izostavljena. Takođe, ako je $s_{k+j} = 0$ za neko j, tada iz (3.2.4) sledi da je $\lambda_j = 0$, tako da j-ta kolona i $(k+j)$-ta vrsta iz matrice A^T može biti izostavljena. Te činjenice povlače sledeća dva tvrđenja. Zbog jednostavnosti koristimo sledeće oznake. Neka su $\alpha = \{\alpha_1, \ldots, \alpha_p\}$ i $\beta = \{\beta_1, \ldots, \beta_q\}$ podskupovi od $\{1, \ldots, m\}$ i $\{1, \ldots, n\}$, respektivno, za neke cele brojeve $1 \leq p \leq m$ i $1 \leq q \leq n$. Sa A^α označavamo $p \times n$ podmatricu od A određenu vrstama koje su indeksirane skupom α. Slično, sa A_β označavamo $m \times q$ podmatricu od A određenu sa kolonama koje su indeksirane skupom β.

Lema 3.2.1 [*93*] *Razmotrimo problem linearnog programiranja* (3.2.2-3.2.4). *Pretpostavimo da je* $x_{i_1} = \cdots = x_{i_d} = 0$, *i razmotrimo skup indeksa* $I = \{i_1, \ldots, i_d\} \subseteq \{1, \ldots, n\}$. *Označimo sa* $\hat{A} \in \mathbb{R}^{m \times (n-d)}$ *matricu* $A_{\{1,\ldots,n\}\setminus I}$, *i neka su* $\hat{x}, \hat{c}, \hat{s} \in \mathbb{R}^{n-d}$, *respektivno, vektori* x, c *i* s *bez* $i-te$ $(i \in I)$ *koordinate. Sa* $\varphi_i(\lambda)$ *označimo linearnu funkciju*

$$\varphi_i(\lambda) = c_i - (a_{1i}\lambda_1 + \cdots + a_{mi}\lambda_m), \quad i \in I.$$

Tada je primal-dualni problem (3.2.2-3.2.4) *ekvivalentan problemu*

$$\begin{aligned} &\textit{minimizirati} \;\; \hat{c}^T\hat{x} \;\; \textit{pod uslovima} \;\; \hat{A}\hat{x} = b, \quad \hat{x} \geq 0, \;\; x_i = 0, \;\; i \in I, \\ &\textit{maksimizirati} \;\; b^T\lambda \;\; \textit{pod uslovima} \;\; \hat{A}^T\lambda + \hat{s} = \hat{c}, \quad \hat{s} \geq 0, \;\; s_i = \varphi_i(\lambda), \;\; i \in I, \end{aligned}$$

gde je $\hat{A}^T$ *transponovana matrica matrice* $\hat{A}$.

Dokaz. Dovoljno je dokazati Lemu za jedan indeks $i \in I$. Pretpostavimo sada da je $x_i = 0$. Koristimo sledeće oznake:

$$\begin{aligned} \hat{A}^i &= [A_1 \ldots A_{i-1} A_{i+1} \ldots A_n], \\ \hat{x}^i &= (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n), \\ \hat{c}^i &= (c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n), \\ \hat{s}^i &= (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n). \end{aligned}$$

Sada je (3.2.2) ekvivalentno sa $x_i = 0$ i

$$\min(\hat{c}^i)^T\hat{x}^i \text{ pod uslovima } \hat{A}^i\hat{x}^i = b, \quad \hat{x}^i \geq 0, \tag{3.2.5}$$

Dualni problem za (3.2.5) je

$$\max b^T y \text{ pod uslovima } (\hat{A}^i)^T y + \hat{s}^i = \hat{c}^i, \quad \hat{s}^i \geq 0. \tag{3.2.6}$$

Sada je (3.2.6) i

$$s_i = c_i - (a_{1i}y_1 + \cdots + a_{mi}y_m) = \varphi_i(y)$$

ekvivalentno sa (3.2.4). □

Lema 3.2.2 [93] *Pretpostavimo da u primal-dual problemu* (3.2.2)-(3.2.4) *važe jednakosti* $s_{k+j_1} = \cdots = s_{k+j_h} = 0$. *Razmotrimo skupove* $J = \{j_1, \ldots, j_h\} \subseteq \{1, \ldots, q\}$ *i* $J^k = \{k + j | \, j \in J\}$. *Označimo sa* $\hat{A}^T \in \mathbb{R}^{(n-h)\times(m-h)}$ *matricu*

$$(A^T)^{\{1,\ldots,n\}\setminus J^k}_{\{1,\ldots,m\}\setminus J} = \left(A^{\{1,\ldots,m\}\setminus J}_{\{1,\ldots,n\}\setminus J^k}\right)^T.$$

Neka su $\hat{x}, \hat{c}, \hat{s} \in \mathbb{R}^{n-h}$, *respektivno, vektori* x, c, s *bez* $(k + j)$*-te* $(j \in J)$ *koordinate, i neka su* $\hat{\lambda}, \hat{b} \in \mathbb{R}^{m-h}$, *respektivno, vektori* λ, b *bez* j*-te* $(j \in J)$ *koordinate. Sa* $\psi_j(\hat{x})$ *označimo linearnu funkciju*

$$\psi_j(\hat{x}) = b_j - (a_{j1}x_1 + \cdots + a_{jn}x_n), \quad j \in J.$$

Tada je primal-dual problem (3.2.2)-(3.2.4) *ekvivalentan sa*

$$\begin{aligned}
&\textit{minimizirati } \hat{c}^T\hat{x} \textit{ pod uslovima } \hat{A}\hat{x} = \hat{b}, \quad \hat{x} \geq 0, \; x_{k+j} = \psi_j(\hat{x}), \; j \in J,\\
&\textit{maksimizirati } \hat{b}^T\hat{\lambda} \textit{ pod uslovima } \hat{A}^T\hat{\lambda} + \hat{s} = \hat{c}, \quad \hat{s} \geq 0, \; s_{k+j} = 0, \; j \in J,
\end{aligned}$$

gde je $\hat{A} = A^{\{1,\ldots,m\}\setminus J}_{\{1,\ldots,n\}\setminus J^k}$.

Dokaz. Dovoljno je dokazati Lemu za jedno $j \in J$. Pretpostavimo sada da je $s_{l+j} = 0$ za neko $1 \leq j \leq m$. Zbog (3.2.3) je $y_j = 0$, i (3.2.4) je ekvivalentno sa

$$\max \hat{b}^T\hat{y} \text{ pod uslovima } \hat{A}^T\hat{y} + \hat{s} = \hat{c}, \quad \hat{s} \geq 0, \tag{3.2.7}$$

gde je

$$\begin{aligned}
(\hat{A}^j)^T &= [A_1^T \cdots A_{j-1}^T A_{j+1}^T \cdots A_m^T],\\
\hat{y}^j &= (y_1, \ldots, y_{j-1}, y_{j+1}, \ldots, y_m),\\
\hat{b}^j &= (b_1, \ldots, b_{j-1}, b_{j+1}, \ldots, b_m),\\
\hat{s}^j &= (s_1, \ldots, s_{l+j-1}, s_{l+j+1}, \ldots, s_n),\\
\hat{c}^j &= (c_1, \ldots, c_{l+j-1}, c_{l+j+1}, \ldots, c_n).
\end{aligned}$$

Primalni problem za (3.2.7) je

$$\min(\hat{c}^j)^T\hat{x}^j \text{ pod uslovima } \hat{A}^j\hat{x}^j = b, \quad \hat{x}^j \geq 0, \tag{3.2.8}$$

gde je

$$\hat{x}^j = (x_1, \ldots, x_{l+j-1}, x_{l+j+1}, \ldots, x_n).$$

Kako je (3.2.2) ekvivalentno sa (3.2.8) i

$$x_{l+j} = b_j - (a_{j1}x_1 + \cdots + a_{jl}x_l) = \psi_j(\hat{x}),$$

dokaz sledi. □

Sledeće tvrđenje objedinjuje prethodne dve leme.

Teorema 3.2.1 [*93*] *Neka su* $x_{i_1} = \cdots = x_{i_d} = 0$ *i* $s_{k+j_1} = \cdots = s_{k+j_h} = 0$, *gde su skupovi* $I = \{i_1, \ldots, i_d\}$ *and* $J = \{j_1, \ldots, j_h\}$ *definisani kao u Lemi 3.2.1 i Lemi 3.2.2, respektivno. Sa oznakama iz Leme 3.2.1 i Leme 3.2.2 primal-dual problem* (3.2.2-3.2.4) *je ekvivalentan sa*

$$\begin{aligned} &\textit{minimizirati} \;\; \hat{c}^T\hat{x} \;\; \textit{pod uslovima} \\ &\qquad \hat{A}\hat{x} = \hat{b}, \hat{x} \geq 0,\ x_i = 0,\ i \in I,\ x_{k+j} = \psi_j(\hat{x}),\ j \in J, \\ &\textit{maksimizirati} \;\; \hat{b}^T\hat{\lambda} \;\; \textit{pod uslovima} \\ &\qquad \hat{A}^T\hat{\lambda} + \hat{s} = \hat{c},\ \hat{s} \geq 0,\ s_{k+j} = 0,\ j \in J,\ s_i = \varphi_i(\hat{\lambda}),\ i \in I, \end{aligned}$$

gde je $\hat{A} \in \mathbb{R}^{(m-h)\times(n-h-d)}$, $\hat{x}, \hat{c}, \hat{s} \in \mathbb{R}^{n-h-d}$, $\hat{\lambda}, \hat{b} \in \mathbb{R}^{m-h}$.

Dokaz. Kako je $I \cap J^k = \emptyset$, za striktno komplementarno rešenje, dokaz sledi direktno iz Leme 3.2.1 i Leme 3.2.2. □

Napomena 3.2.1 *Teorema* 3.2.1 *je uopštenje Teoreme 3.1 iz rada* [*94*], *deo (b).*

Napomenimo da za proizvoljnu tačku $(x, y, s) \in \mathcal{N}_{-\infty}(\gamma)$, možemo proceniti skupove $\mathcal{B}$ i $\mathcal{N}$ na sledeći način [112]:

$$\begin{aligned} \mathcal{B}(x,s) &= \{i \in \{1, \ldots, n\} |\, x_i \geq s_i\}, \\ \mathcal{N}(x,s) &= \{1, \ldots, n\} \backslash \mathcal{B}(x,s), \end{aligned} \tag{3.2.9}$$

gde je

$$\mathcal{N}_{-\infty}(\gamma) = \{(x, y, s) \in \mathcal{F}^0 |\, x_i s_i \geq \gamma\mu \text{ za svako } i = 1, \ldots, n\}$$

pri čemu je $\gamma \in (0, 1)$.

Sledeći algoritam konačnog završavanja (Algotritam FT) je predložio Ye [117].

Algoritam FT
Dato je $\gamma \in (0, 1)$ i $(x, y, s) \in \mathcal{N}_{-\infty}(\gamma)$:
Odrediti skupove $\mathcal{B}(x, s)$ i $\mathcal{N}(x, s)$;
Rešiti sledeći problem:

$$\begin{aligned} \min_{(x^*, y^*, s^*)} &\frac{1}{2}|x^* - x|^2 + \frac{1}{2}|s^* - s|^2, \\ &Ax^* = b, \quad A^T y^* + s^* = c, \\ x_i^* = 0 \text{ for } i \in \mathcal{N}(x,s), &\quad s_i^* = 0 \text{ for } i \in \mathcal{B}(x,s). \end{aligned} \tag{3.2.10}$$

If $x_{\mathcal{B}}^* > 0$ i $s_{\mathcal{N}}^* > 0$

zaokruživanje je uspešno: (x^*, y^*, s^*) je striktno komplementarno rešenje;
else
zaokruživanje je neuspešno i vraćamo se u primal-dual algoritam.

Sledeći rezultat iz [112] pokazuje da je uspešnost Algoritma FT garantovana kada je μ dovoljno malo.

Teorema 3.2.2 *Neka je $\gamma \in (0,1)$ dato. Tada postoji granična vrednost $\bar{\mu}$ tako da za sve (x, y, s) koje zadovoljavaju*

$$(x, y, s) \in \mathcal{N}_{-\infty}(\gamma), \quad 0 < \mu = x^T s/n \leq \bar{\mu},$$

važi

(i) $\mathcal{B}(x, s) = \mathcal{B}$ i $\mathcal{N}(x, s) = \mathcal{N}$; što znači da su konkretni skupovi $\mathcal{B}$ i $\mathcal{N}$ korektno procenjeni postupkom (3.2.9)*;*

(ii) procedura (3.2.10) *daje striktno komplementarno rešenje (x^*, y^*, s^*).*

Sada predlažemo sledeću modifikaciju Algoritma FT. Neka je $\varepsilon > 0$ dato. Koristeći (3.2.9) procenimo skupove $\mathcal{B}$ i $\mathcal{N}$. Stavimo

$$I = \{\, i \,|\, x_i < \varepsilon,\ i \in \mathcal{N}\}, \quad J = \{\, j \,|\, s_{l+j} < \varepsilon,\ l + j \in \mathcal{B}\}.$$

u Teoremi 3.2.1 i ako je

$$x_{l+j} = \psi_j(\hat{x}) > 0,\ j \in J, \text{ i } s_i = \varphi_i(\hat{y}) > 0,\ i \in I,$$

primenimo Algoritam FT na primal-dual problem

$$\min \hat{c}^T \hat{x} \text{ pod uslovom } \hat{A}\hat{x} = \hat{b},$$
$$\max \hat{b}^T \hat{y} \text{ pod uslovom } \hat{A}^T \hat{y} + \hat{s} = \hat{c}.$$

Primetimo da u ovoj varijanti algoritma redukujemo primal-dual problem sa matricom $A_{m\times n}$ na primal-dual problem sa matricom $\hat{A}_{(m-r)\times(n-q)}$. Istu ideju ćemo u nastavku sprovesti u toku same implementacije primal-dual algoritma.

3.3 Redukcija dimenzije i potencijalna funkcija

U ovom odeljku pokazujemo uticaj redukcije dimenzije na primal-dual potencijalnu funkciju. Potencijal-redukcioni metodi koriste logaritamsku potencijanu funkciju kao kriterijum za svaku tačku iz $\mathcal{F}^0$ i pri tome u svakom koraku sledeću iteraciju biraju tako da se potencijalna funkcija smanji za datu fiksiranu vrednost [104]. Primer primal-dual potencijalne funkcije je Tanabe-Todd-Ye potencijalna funkcija

$$\Phi_\rho(x, s) = \rho \log x^T s - \sum_{i=1}^{n} \log x_i s_i \tag{3.3.11}$$

za neki parametar $\rho > n$. Potencijal-redukcioni algoritmi traženi pravac dobijaju rešavanjem (3.3.11) za neko $\sigma_k \in (0,1)$. Dužina koraka se bira tako da aproksimativno minimizira Φ_ρ duž dobijenog pravca.

U sledećoj teoremi ispitujemo uticaj redukcije problema pod uslovima Teoreme 3.2.1 na funkciju Φ_ρ.

Teorema 3.3.1 *Pretpostavimo da iterativni niz ne prati centralnu putanju i pretpostavimo da je $x_j s_j < \varepsilon < 1$ i $x_i s_i \geq \varepsilon$, $i \neq j$. Ako je moguće eliminisati x_j, za proizvoljno j (ili s_j, za neko $j > k$) i eliminisati j-tu kolonu iz matrice A (j-tu vrstu i $(j-k)$-tu kolonu iz matrice A^T), tada potencijal-redukciona funkcija se približno smanjuje za* $\log\left(\dfrac{n}{n-1}\right)^\rho - \log x_j s_j > 0$.

Dokaz. Dokaz Primenom Leme 3.2.1 (Leme 3.2.2) možemo eliminisati x_j, (s_j) i j-tu kolonu iz matrice A (j-tu vrstu i $(j-k)$-tu kolonu iz matrice A^T). Neka je

$$\hat{\Phi}_\rho(\hat{x}, \hat{s}) = \rho \log \hat{x}^T \hat{s} - \sum_{i \neq j} \log x_i s_i$$

potencijal-redukciona funkcija redukovanog problema, gde $\hat{x}$ i $\hat{s}$ označavaju vektore x i s bez j-tih elemenata, respektivno. Tada važi

$$\begin{aligned}
\Phi_\rho(x,s) - \hat{\Phi}_\rho(\hat{x},\hat{s}) &= \rho(\log x^T s - \log \hat{x}^T \hat{s}) - \log x_j s_j \\
&= \rho \log\left(1 + \frac{x_j s_j}{\sum_{i \neq j} x_i s_i}\right) - \log x_j s_j \\
&\geq \rho \log\left(1 + \frac{\varepsilon}{(n-1)\varepsilon}\right) - \log x_j s_j \\
&\geq \log\left(\frac{n}{n-1}\right)^\rho - \log x_j s_j > \log\left(\frac{n}{n-1}\right)^\rho - \log \varepsilon > 0.
\end{aligned}$$

Dokaz je kompletiran. □

4 Modifikovani algoritam i implementacija

U ovom odeljku razmatramo smanjenje dimenzije problema u toku izvršenja Mehrotraovog primal-dual algoritma. Predlažemo sledeće modifikacije *Koraka 12* i *Koraka 2*. Modifikacija *Koraka 12* se bazira na eliminaciji promenljivih x_i i s_i koje konvergiraju ka nuli, koja se zasniva na Teoremi 3.2.1.

4.1 Opis algoritma

Za tačku $(x, \lambda, s) \in \mathcal{N}_{-\infty}(\gamma)$, možemo proceniti skupove $\mathcal{B}$ i $\mathcal{N}$ kada je μ dovoljno malo na osnovu (3.2.9). Takođe je poznato da postoji granična vrednost $\overline{\mu}$ tako da za svako (x, λ, s) koje zadovoljava

$$(x, \lambda, s) \in \mathcal{N}_{-\infty}(\gamma) \subset \mathcal{F}^0, \quad 0 < \mu = x^T s/n \leq \overline{\mu}, \tag{4.1.1}$$

važi $\mathcal{B}(x,s)=\mathcal{B}$ and $\mathcal{N}(x,s)=\mathcal{N}$, tj. indeksni skupovi $\mathcal{B}$ i $\mathcal{N}$ su korektno procenjeni sa (3.2.9) [112].

Potrebni su nam mali realni brojevi $eps1$, $eps2$ and $eps3$ kao kriterijum za primenu redukcije predložene u Teoremi 3.2.1. Motivacija za primenu tih krirterijuma je u proceni (3.2.9) i (4.1.1).

Koristimo vektor *xzero* za pamćenje indeksa eliminisanih elemenata vektora x. Prazna lista je početna vrednost za vektor *xzero*.

Korak 12M. Izračunati sledeću iteraciju

$$x=x^{t+1}=x^t+\alpha_t^{pri}\Delta x^t,$$
$$(l,s)=(\lambda^{t+1},s^{t+1})=(\lambda^t,s^t)+\alpha_t^{dual}(\Delta\lambda^t,\Delta s^t).$$

Ako je uslov $xs=x^Ts<eps3$ zadovoljen, izvršiti dve petlje, *Petlju 1* i *Petlju 2*, koje su definisane na sledeći način.

Petlja 1. Za svako $i=1,\ldots,k$, ako je uslov

$$x_i=x_i^{t+1}<eps1,\quad x_i\le s_i=s_i^{t+1} \tag{4.1.2}$$

zadovoljen, izvršiti sledeće korake 1.1-1.4.

1.1. Obrisati i-ti element iz vektora x, c i s, i obrisati i-tu kolonu iz matrice A.

1.2. Postaviti $k=k-1$.

1.3. Ubaciti indeks i na početak vektora *xzero*.

1.4. Postaviti $i=i-1$.

Odgovarajući kod je:

```
For[i=1,i<=k,i++,
   cut=(x[[i]]<eps1) && (x[[i]]<s[[i]]);
   If[cut,
     x=Delete[x,i]; ce=Delete[ce,i]; s=Delete[s,i];
     a=Transpose[Delete[Transpose[a],i]];
     k--;
     xzero=Prepend[xzero,i];
     i--;
];
```

Petlja 2. za svako $i=k+1,\ldots,k+q=n$ izvršiti korake 2.1-2.4.

2.1. Postaviti $False$ kao vrednost logičke promenljive lg. Ta promenljiva se koristi kao indikator da je neki element iz x obrisan.

2.2. Pod uslovima (4.1.2) izvršiti sledeće:

2.2.1. Postaviti $lg=True$.
2.2.2. Obrisati i-ti element iz x, c, s i i-tu kolonu iz matrice A.
2.2.3. Postaviti $q=q-1$.
2.2.4. Zamenimo $(i-k)$-ti i poslednji element u A, x i $l=\lambda$.

2.3. U slučaju da je $i \leq Length[s]$, ako je uslov

$$lg = False, \quad s_i = s_i^{t+1} < eps2, \quad s_i \leq x_i = x_i^{t+1} \tag{4.1.3}$$

zadovoljen, izvršiti sledeće korake:

2.3.1. Obrisati i-ti element iz x, c i s, obrisati $(i-k)$-ti element iz b i $l = \lambda$, i obrisati $(i-k)$-tu vrstu i i-tu kolonu iz matrice A.

2.3.2. Postaviti $q = q - 1$.

2.4. U slučju da je $lg = True$ postaviti $i = i - 1$.

Konačno, kao i u klasičnom metodu, postaviti $(x^t, \lambda^t, s^t) = (x^{t+1}, \lambda^{t+1}, s^{t+1})$ i preći na *Korak 2M*, modifikaciju *Koraka 2*.

Odgovarajući kôd je:

```
For[i=k+1,i<=k+q,i++,
  lg=False;
  cut=(x[[i]]<eps1) && (x[[i]]<s[[i]]) && (i<=Length[x]);
    If[cut,
      lg=True;
      x=Delete[x,i]; ce=Delete[ce,i]; s=Delete[s,i];
      a=Transpose[Delete[Transpose[a],i]];
      q--;
      xzero=Prepend[xzero,i];
      pp=a[[i-k]]; a=Delete[a,i-k]; AppendTo[a,pp];
      pp=b[[i-k]]; b=Delete[b,i-k]; AppendTo[b,pp];
      pp=l[[i-k]]; l=Delete[l,i-k]; AppendTo[l,pp];
    ];
    If[i<=Length[s],
      cut=(s[[i]]<eps2) && (s[[i]]<x[[i]]);
      If[!lg && cut,
       lg=True;
        x=Delete[x,i]; ce=Delete[ce,i]; s=Delete[s,i];
        a=Delete[a,i-k]; b=Delete[b,i-k]; l=Delete[l,i-k];
        a=Transpose[Delete[Transpose[a],i]];
        xzero=Prepend[xzero,i];
        q--;
    ];];
    If[lg, i--];
];
```

Korak 2M. Proveriti kriterijum za zaustavljanje kao u *Koraku 2*. Ako taj kriterijum nije ispunjen, preći na *Korak 3*. Kada je kriterijum zadovoljen, izvršimo rekonstrukciju vektora x^t, umetanjem nula na odgovarajućim pozicijama vektora x^t, gde su odgovarajući elementi obrisani. Za tu svrhu koristimo listu $xzero$ i primenimo sledeću For petlju:

Za svako $i = 1, \ldots, n = Length[xzero]$ postaviti nulu na i-tu poziciju vectora x^t.

Nakon toga vratimo x kao izlaz.

To se postiže sledećim kodom

```
For[i=1, i<=Length[xzero],i++,
      x=Insert[x,0,xzero[[i]]];
];
```

Napomena 4.1.1 Važan praktičan problem je izbor indikatora $\mathcal{B}(x,s)$ za optimalnu particiju $\mathcal{B}$. Osim kriterijuma (3.2.9) i (4.1.1), moguće je primeniti i druge kriterijume za procenu indeksnih skupova.

1. Indikator (3.2.9) nije invarijantan u odnosu na skaliranje kolona. Alternativni indikator je definisan u [24]:

$$\mathcal{B}(x,s) = \{i \in \{1,\ldots,n\} | \, (|\Delta x_i^{aff}|/x_i) \leq (|\Delta s_i^{aff}|/s_i)\},$$

gde je $(\Delta x_i^{aff}, \Delta s_i^{aff})$ afini pravac. Taj indikator je invarijantan u odnosu na skaliranje kolona [24, 2].

Jedina modifikacija u prethodnom *Koraku 12M* je sledeća. U *Petlji 1* i u *Koraku 2.2 Petlje 2* zamenimo uslov (4.1.2) uslovom

$$\left|\Delta x_i^{aff}\right|/x_i > \left|\Delta s_i^{aff}\right|/s_i, \quad x_i < eps1,$$

što u programu glasi

$$Abs[Dxaff[[i]]]/x[[i]] > Abs[Dsaff[[i]]]/s[[i]].$$

Takođe, u *Koraku 2.3 Petlje 2* zamenimo uslov (4.1.3) sa

$$\left|\Delta x_i^{aff}\right|/x_i < \left|\Delta s_i^{aff}\right|/s_i, \quad s_i < eps2.$$

Primena ovih kriterijuma daje numeričke rezultate koji su vrlo slični rezultatima koji se dobijaju posle primene uslova (4.1.2) i (4.1.3).

2. Umesto kriterijuma $xs = x^T s < eps3$, koji dozvoljava redukciju u *Koraku 12M*, takođe smo koristili alternativni kriterijum $x^T s/x1^T s1 = r < eps3$, gde su $x1$ i $s1$ vrednosti za x i s u prethodnom iterativnom koraku. Naša numerička iskustva pokazuju da većina testiranih problema u poslednjim koracima iterativnog procesa uzima vrednost $r \approx 0.01$. To znači da je superlinearna konvergencija dostignuta za vrednost $eps3 = 0.01$. Ali, neki od tih problema poseduju sasvim zadovoljavajuću konvergenciju čak i za vrednosti $eps3 = 0.1$. Posle primene tog kriterijuma dobili smo gotovo iste rezultate kao i sa prethodnim kriterijumom.

Napomena 4.1.2 Primenom Teoreme 3.2.1 možemo da redukujemo dimenziju primal-dualnog linearnog problema (3.2.2) – (3.2.4). Takođe, time se poboljšava stabilnost i centralnost iterativnog niza. To sledi iz sledećih razmatranja. U *Koraku 4* svake primal-dual iteracije neophodno je rešiti linearni sistem (2.1.1). Takođe,

u *Koraku 8* je neophodno rešiti sistem (2.1.2). Primena Teoreme 3.2.1 redukuje sistem u (2.1.1) sa matricom tipa $A_{m\times n}$ u ekvialentan sistem sa matricom manje dimenzije $\hat{A}_{(m-h)\times(n-h-d)}$. Smanjenje dimenzije problema nam daje pogodnosti u izračunavanjima i potrebnom memorijskom prostoru. Jasno je da je modifikacija posebno efikasna za probleme linearnog programiranja velikih dimanzija. Takođe, poseban problem stabilnosti u (2.1.1) nastaje zbog prisustva veoma malih (i respektivno veoma velikih) dijagonalnih elemenata u D^2 i D^{-2}. U skladu sa Teoremom 3.2.1 moguće je eliminisati neke od tih elemenata i poboljšati stabilnost i centralnost iterativnog niza.

Napomena 4.1.3 Postavljanje $x_i = 0$, $i \in I$, $s_{k+j} = 0$, $j \in J$ u *Koraku 12M* je analogno postavljanju $x_i^* = 0$, $i \in \mathcal{N}(x,s)$, $s_i^* = 0$, $i \in \mathcal{B}(x,s)$ što je poznato iz algoritma konačnog završavanja. Naša strategija se razlikuje od klasičnog algoritma konačnog završavanja jer se ne procenjuju svi elementi iz skupova $\mathcal{N}$ i $\mathcal{B}$. Pored toga, mi primenjujemo tu strategiju sukscesivno tokom iteracija, vršeći pri tome samo jednostavnu verifikaciju.

4.2 Numerički primeri

U prvom primeru upoređujemo odnos procesorskih vremena za modifikovani i originalni metod dok u drugom primeru pokazujemo da postoje problemi malih dimenzija koje modifikacija uspešno rešava za razliku od originalnog metoda koji te probleme ne može da reši sa zadatom tačnošću.

Primer 4.2.1 U ovom primeru razmatramo nekoliko poznatih test problema iz literature i nekoliko slučajno generisanih primera. U Tabeli 4.1.1 su redom dati: naziv problema, zadate tačnosti, broj iteracija MPD, rezultati dobijeni modifikovanim algoritmom $MMPD$, broj iteracija $MMPD$ i odnos vremena izvršenja ova dva algoritma. Napomenimo da je $eps3 = 1$ u svim primerima. Najveće dimenzije matrice problema su 390×740 kod primera $Test12$, 444×757 kod primera $Degen2$ i 516×758 kod primera $Agg2$ i $Agg3$.

Problem	*eps*	$eps1, 2$	Br.it. MPD	MMPD opt. vred.	Br. it.	$TMod/T$
Afiro	10^{-12}	10^{-1}	13	-464.7531428571451	13	0.7135
Blend	10^{-12}	10^{-2}	11	-30.81214984583144	12	0.8846
KbM	10^{-12}	10^{-3}	12	0.	9	0.5599
Kb2	10^{-12}	10^{-3}	19	-1749.900129941934	20	0.7956
Adlittle	10^{-12}	10^{-2}	16	225494.963162381	16	0.7739
Sc50a	10^{-12}	10^{-3}	13	-64.57507705855755	13	0.8283
Sc50b	10^{-12}	10^{-3}	12	-70.00000000000004	12	0.8682
Sc105	10^{-12}	10^{-3}	15	-52.20206121170699	15	0.8792
Sc205	10^{-12}	10^{-4}	18	-52.20206121170724	17	0.8838
Share2b	10^{-12}	10^{-3}	14	-415.7322407415902	14	0.8414
Stocfor1	10^{-12}	10^{-3}	17	-41131.97621943501	17	0.7763
Lotfi	10^{-8}	10^{-4}	21	-25.2647060618722685	22	0.8105
Agg2	10^{-8}	10^{-4}	22	$-2.023925235068718\times10^{7}$	22	0.8974
Agg3	10^{-10}	10^{-4}	23	$1.031211593574227\times10^{7}$	21	0.7579
Test1	10^{-12}	10^{-4}	30	86.9999999999999928	29	0.8547
Test2	10^{-12}	10^{-4}	19	0.799724607645447882	18	0.8730
Test3	10^{-12}	10^{-4}	21	0.790546594605269792	19	0.6855
Test4	10^{-12}	10^{-4}	19	2.91137275484501465	18	0.7162
Test5	10^{-12}	10^{-4}	19	3.05735620828120602	18	0.6792
Test6	10^{-12}	10^{-4}	22	2.24385948689881997	21	0.7172
Test7	10^{-12}	10^{-4}	22	2.24385948689881997	21	0.8139
Test8	10^{-12}	10^{-4}	23	486.128054336368009	22	0.7257
Test9	10^{-12}	10^{-4}	23	495.742433638785229	22	0.7381
Test10	10^{-12}	10^{-4}	24	495.628752528977134	24	0.8684
Test11	10^{-12}	10^{-4}	25	466.272008865721332	24	0.7779
Test12	10^{-8}	10^{-4}	13	44.51259834897959	13	0.6959

Tabela 4.1.1

Napomenimo da *eps* označava kriterijum za zaustavljanje algoritma. Takođe, problem *KbM* se dobija brisanjem gornjih ograničenja iz poznatog problema *Kb2*. Napomenimo da vrednosti promenljivih *eps1*, *eps2* i *eps3* moraju biti pažljivo izabrane. Na primer, koristeći $eps1 = eps2 = 10^{-3}$ u problemu *Blend*, dobijamo loše uslovljen linearni sistem, ali modifikacija i pored toga daje optimalnu vrednost. Međutim, modifikacija ne daje rešenje za izbor $eps1 = eps2 = 10^{-1}$.

Takođe ilustrujemo tvrđenje da je algoritam konačnog završavanja nepotreban u našem algoritmu. Razmotrimo, na primer, test problem *Afiro.* Optimalna tačka generisana modifikovanim algoritmom ima nula koordinate na pozicijama koje su određene sledećim vektorom $xzero$

$$\{17,17,17,12,12,12,12,12,12,11,9,2,2,2,2,1,28\}.$$

Optimalna tačka xm jednaka je

$$\{0,\ 80.00000000000023,0,0,0,0,18.21428571428548,$$
$$51.49692904472464,73.89388764455072,25.49999999999969,$$
$$500.0000000000107,475.9200000000041,24.08000000000006,$$
$$0,214.9999999999961,147.786872899196,0,$$
$$54.50000000000029,0,0,0,0,0,0,339.9428571428572,$$
$$236.1559842436612,63.54835534665428,0,84.79999999999951,$$
$$69.71121475901005,0,0,0\}.$$

Sa druge strane, obični Mehrotraov metod daje optimalnu tačku x, jednaku vektoru

$\{$2.215848062666221$\times 10^{-12}$, 79.99999999999929,
1.312660664489544$\times 10^{-13}$, 4.394145729655261$\times 10^{-13}$,
4.353491369592836$\times 10^{-13}$, 5.774260579869571$\times 10^{-13}$,
18.21428571428262, 51.50061507109443, 73.89779483249926,
25.5000000000001, 500.00000000001, 475.9199999999925,
24.08000000000016, 7.65658212305429$\times 10^{-12}$,
214.9999999999957, 147.7683249031915,
2.425875538820393$\times 10^{-13}$, 54.49999999999952,
2.415068530966149$\times 10^{-13}$, 2.403460673212315$\times 10^{-13}$,
1.409009221948509$\times 10^{-13}$, 4.755857873948911$\times 10^{-13}$,
4.713557464415677$\times 10^{-13}$, 4.670668078581763$\times 10^{-13}$,
339.9428571428506, 236.1745322396589, 63.54037970837263,
3.686831036665461$\times 10^{-14}$, 84.79999999999963,
69.71490078537609, 2.196141063804613$\times 10^{-13}$,
2.170131429848782$\times 10^{-13}$, 1.773714715268836$\times 10^{-13}\}$.

Sličan oblik rešenja je generisan programom PCx.

Mali elementi u x, koji mogu da generišu loše uslovljenu matricu D^{-2}, su eliminisani tokom primene modifikovanog metoda. Slično, mali elementi u s koji mogu da generišu loše uslovljenu matricu D^2, se takođe izbegavaju u izračunavanjima modifikovanog algoritma. Iz tih razloga, modifikovani algoritam daje bolje rešenje u odnosu na originalni algoritam.

Primer 4.2.2 Na sledeća dva mala test problema je ilustrovana je veća numerička stabilnost modifikovanog metoda. Razmotrimo problem

$$\begin{array}{lrcl} \min & 3x_3 + 2x_4 + x_5 & & \\ \text{p.o.} & x_1 + 2x_2 + x_3 & = & 1000.02, \\ & x_1 + x_2 + x_4 & = & 1000.01, \\ & x_1 - x_2 + x_5 & = & 999.99, \\ & x_1,\, x_2,\, x_3,\, x_4,\, x_5 & \geq & 0. \end{array} \tag{4.2.1}$$

Primenom modifikovanog algoritma sa $eps = 10^{-12}$, $eps1 = 10^{-4}$, $eps2 = 10^{-3}$ i $eps3 = 10^{-1}$ dobijamo optimalnu vrednost 0 i ekstremnu tačku

$$\{999.9999999998636, 0.01000000039754838, 0, 0, 0\}.$$

Sa druge strane, običan algoritam nije u stanju da reši problem sa istom preciznošću 10^{-12}. Već smo pokazali da PCx takođe ne može da reši ovaj problem. Napomenimo da je u radu [64] dat interesantan primer dimenzija 18×18, baziran na prethodnom primenru, koji ne rešavaju PCx, HOPDM ni naša modifikacija.

Drugi primer je sledeći test problem

$$\begin{aligned} \max\ & 30x_1 + 60x_2 + 50x_3 \\ \text{p.o.}\ & 3x_1 + 4x_2 + 2x_3 \leq 60, \\ & x_1 + 2x_2 + 2x_3 \leq 30, \\ & 2x_1 + x_2 + 2x_3 \leq 40, \\ & x_1, x_2, x_3 \geq 0. \end{aligned} \tag{4.2.2}$$

Modifikovani algoritam sa $eps = 10^{-12}$, $eps1 = 10^{-3}$, $eps2 = 10^{-3}$ i $x.s < eps3 = 1$ daje optimalnu vrednost 900.0000000000002 i optimalnu tačku $\{0, 15., 0\}$ posle 9 iteracija. Originalni method posle pet iteracija dolazi do loše uslovljene matrice i staje posle sedam iteracija sa sistemom linearnih jednačina koji nema rešenje.

4.3 Poređenja algoritama i zaključne napomene

U ovom odeljku je opisano poboljšanje primal-dual metoda koje se uglavnom zasniva na eliminaciji vrsta i kolona iz matrice problema. Slična istraživanja su započeta u radu [65] gde je testirana samo eliminacija kolona. Takođe je razvijen eksperimentalni kôd u programu MATHEMATICA za implementaciju predložene modifikacije. Naš glavni cilj je da tim kodom uporedimo Mehrotraov primal-dual metod sa našom modifikacijom koja je opisana u odeljku 2.3. Posle primene predložene heuristike u Mehrotraovom algoritmu imamo sledeće prednosti u odnosu na originalni algoritam, koje su pokazane u prethodnim primerima:

- Modifikovani metod daje bolju preciznost

- Modifikacija poboljšava stabilnost i centralnost iterativnog niza.

  Na primer, tokom rešavanja problema *Afiro* modifikovanim metodom, loše uslovljene matrice koje mogu da dovedu do značajnih numeričkih grešaka se ne pojavljuju. Međutim, tokom primene običnog metoda, posle devetog koraka se javlja loše uslovljena matrica. Slična je situacija sa problemom *Blend*, gde običan metod prouzrokuje loše uslovljenu matricu posle deset koraka. Kao što je pokazano u Primeru 4.2.1, moguće pojavljivanje loše uslovljene matrice se može izbeći pažljivim izborom vrednosti za parametre $eps1$, $eps2$ i $eps3$. Takođe, dva ilustrativna test problema su data u Primeru 4.2.2

- Modifikovani metod je brži u odnosu na originalni, iako sadrži dodatne eliminacije nekih vrsta i kolona i na kraju primenjuje rutinu za rekonstrukciju rešenja. Tu činjenicu potvrđuje poslednja kolona u Tabeli 4.3.1.

  Takođe, primetimo da modifikacija redukuje procesorsko vreme potrebno za rešavanje problema *Blend* i *Kb2*, uprkos dodatnog iterativnog koraka koji zahteva modifikacija.

- Modifikacija smanjuje dimenziju problema i zahtev za memorijskim prostorom.

  U sledećoj tabeli sa *Dim[AS]* označavamo startnu dimenziju matrice A i *Dim[AF]* označava dimenziju finalne matrice A. Takođe, maksimalan brojn bajtova memorije koja se koristi za pamćenje inicijalne matrice A je označen sa *ByteCount[AS]*, dok *ByteCount[AF]* označava maksimalan broj bajtova memorije koja se koristi za pamćenje matrice A u finalnom iterativnom koraku.

Problem	Dim[AS]	Dim[AF]	ByteCount[AS]	ByteCount[AF]
Afiro	27×52	21×16	28864	7336
Blend	74×114	60×56	170816	68904
KbM	43×68	27×6	59712	3808
Kb2	52×77	33×33	81768	22864
Adlittle	56×138	46×61	156152	57616
Sc50a	27×52	21×16	28864	7336
Sc50b	74×114	60×56	170816	68904
Sc105	43×68	27×6	59712	3808
Share2b	52×77	33×33	81768	22864
Stockfor1	56×138	46×61	156152	57616

Tabela 4.3.1

Pored toga, navedimo i dodatne prednosti modifikovanog metoda:

- U modifikovanom algoritmu algoritam konačnog zaokruženja je nepotreban, kao što je pokazano u Primeru 4.2.1.
- Modifikaciju je moguće primeniti u bilo kom primal-dual algoritmu.
- Opisana modifikacija je primenljiva na već postojeće primal-dual kodove.
- Kako modifikacija redukuje dimenziju problema za vreme iterativnih koraka, realno je očekivati da će biti posebno efikasna za velike probleme linearnog programiranja (primeri *Agg2, Agg3, T12*).

5 Prošireni i normalni sistem jednačina

U ovom poglavlju upoređujemo dve varijante Mehrotraovog primal dual algoritma koje se baziraju na proširenom i normalnom sistemu jednačina, respektivno razmatrane su u radu [87]. Implementacija odgovarajućih algoritama u programu MATHEMATICA je iskorišćena za poređenje. Na kraju dajemo nekoliko ilustrativnih numeričkih primera.

5.1 Primal-dual algoritam sa proširenim sistemom

Razmatramo opšti problem linearnog programiranja (1.1.1). Poznato je da linearni sistem koji se rešava u svakoj primal-dual iteraciji možemo formulisati na tri ekvivalentna načina. Neredukovan oblik za nedopustiv algoritam unutrašnje tačke je

$$\begin{bmatrix} 0 & A & 0 \\ A^T & 0 & I \\ 0 & S & X \end{bmatrix} \begin{bmatrix} \Delta\lambda \\ \Delta x \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_b \\ -r_c \\ -XSe + \sigma\mu e \end{bmatrix}, \tag{5.1.1}$$

gde je $\mu = x^T s/n$ and $r_b = Ax - b$, $r_c = A^T\lambda + s - c$.

Eliminacijom Δs iz (5.1.1) i koristeći notaciju $D = S^{-1/2}X^{1/2}$, dobijamo prošireni sistem i posle eliminacije Δx iz proširenog sistema dobijamo normalni sistem jednačina. Detalji će biti dati u opisu algoritma. Pomenuti sistemi linearnih jednačina se rešavaju u *Koraku 4* i *Koraku 8*, tako da ostale korake koji su identični odgovarajućim koracima u klasičnom Mehrotrovom algoritmu ne navodimo.

Korak 4. Izračunati $D = S^{-1/2}X^{1/2}$, $r_{xs} = XSe$ i rešiti jedan od sledeća dva sistema u odnosu na $(\Delta x^{aff}, \Delta\lambda^{aff}, \Delta s^{aff})$. U slučaju normalnog sistema rešiti sistem

$$\begin{aligned} AD^2A^T\Delta\lambda^{aff} &= -r_b - A(S^{-1}Xr_c - S^{-1}r_{xs}),\\ \Delta s^{aff} &= -r_c - A^T\Delta\lambda^{aff},\\ \Delta x^{aff} &= -S^{-1}(r_{xs} + X\Delta s^{aff}), \end{aligned}$$

gde je

$$S^{-1} = \mathrm{diag}(1/s_1, \ldots, 1/s_{k+q}), \quad D^2 = \mathrm{diag}(x_1/s_1, \ldots, x_{k+q}/s_{k+q}).$$

U slučaju proširenog sistema rešiti

$$\begin{aligned} \begin{bmatrix} 0 & A \\ A^T & -D^{-2} \end{bmatrix} \begin{bmatrix} \Delta\lambda^{aff} \\ \Delta x^{aff} \end{bmatrix} &= \begin{bmatrix} -r_b \\ -r_c + X^{-1}r_{xs} \end{bmatrix},\\ \Delta s^{aff} &= -r_c - A^T\Delta\lambda^{aff}. \end{aligned} \tag{5.1.2}$$

Napomenimo da je u [112] preporučena sledeća formula za Δs^{aff}:

$$\Delta s^{aff} = -X^{-1}(r_{xs} - A^TS\Delta x).$$

Ali, u tom slučaju radimo sa matricom X^{-1}, koja je loše uslovljena u finalnim iterativnim koracima. Iz tog razloga prednost dajemo reprezentaciji oblika (5.1.2).

Korak 8. Izračunati $r_{xs} = -\sigma\mu e + \Delta X^{aff}\Delta S^{aff}e$, gde je

$$\Delta X^{aff} = \mathrm{diag}(\Delta x_1^{aff}, \ldots, \Delta x_n^{aff}),\ \Delta S^{aff} = \mathrm{diag}(\Delta s_1^{aff}, \ldots, \Delta s_n^{aff}),$$

i rešiti jedan od sledeća dva sistema u odnosu na $(\Delta x^{cor}, \Delta\lambda^{cor}, \Delta s^{cor})$. U slučaju normalnog sistema rešiti sistem

$$\begin{aligned} AD^2A^T\Delta\lambda^{cor} &= AS^{-1}r_{xs},\\ \Delta s^{cor} &= -A^T\Delta\lambda^{cor},\\ \Delta x^{cor} &= -S^{-1}(r_{xs} + X\Delta s^{cor}), \end{aligned}$$

ili u slučaju proširenog sistema rešiti

$$\begin{aligned} \begin{bmatrix} 0 & A \\ A^T & -D^{-2} \end{bmatrix} \begin{bmatrix} \Delta\lambda^{cor} \\ \Delta x^{cor} \end{bmatrix} &= \begin{bmatrix} -r_b \\ -r_c + X^{-1}r_{xs} \end{bmatrix},\\ \Delta s^{cor} &= -A^T\Delta\lambda^{cor}. \end{aligned} \tag{5.1.3}$$

5.2 Implementacija algoritma sa proširenim sistemom

Za generisanje matrice proširenog sistema (5.1.2) i (5.1.3) koristimo sledeću rutinu $FormAug$:

```
LinearAlgebra`MatrixManipulation`

FormAug[a_,d_]:=
   Module[{rez={},z={},dim},
     dim=Dimensions[a][[1]];
     z=ZeroMatrix[dim];
     rez=AppendColumns[AppendRows[z,a],AppendRows[Transpose[a],-d]];
     Return[rez];
   ]
```

Korak 4:

Implementacija rešavanja normalnog sistem u *Koraku 4* je objašnjena ranije. Implementacija koja odgovara proširenom sistemu je data sledećim kodom:

```
dd= DiagonalMatrix[N[s/x,20]];
AugA=FormAug[a,dd];
ls=LinearSolve[Xmat,rxs];       (* Compute X^-1rxs *)
lx=LinearSolve[AugA,Join[-rb,-rc+ls]]; (* Solve the system (2.2A) *)
Dlaff=Take[lx,Dimensions[a][[1]]];
Dxaff=Drop[lx,Dimensions[a][[1]]];
Dsaff=N[-rc-Transpose[a].Dlaff,20];
```

Napomenimo da vrednost $ls = X^{-1}rxs$ može biti izračunata sa $ls = Inverse[Xmat].rxs$. Međutim, izraz $Inverse[Xmat]$ prouzrokuje numeričke probleme zbog malih vrednosti nekih koordinata vektora x koje se nalaza na glavnoj dijagonali matrice $Xmat$.

Korak 8:

Implementacija rešavanja normalnog sistem u *Koraku 8* je objašnjena ranije. Implementacija za prošireni sistem (5.1.3) je data sledećim kôdom:

```
DXaff=DiagonalMatrix[Dxaff];     DSaff=DiagonalMatrix[Dsaff];
rxs=N[-mi*SigCent,20]*Table[1,{k+q}]+N[DXaff.DSaff,20].Table[1,{k+q}];
rb=Table[0,{Dimensions[a][[1]]}];
ls=LinearSolve[Xmat,rxs];  (* Compute ls = X^-1rxs *)
lx=LinearSolve[AugA,Join[-rb,ls]]; (* Solve the system *)
Dlcc=Take[lx,Dimensions[a][[1]]];
Dxcc=Drop[lx,Dimensions[a][[1]]];
Dscc=N[-Transpose[a].Dlcc,20];
```

5.3 Numerički primeri

Primer 5.3.1 Primenili smo obe verzije algoritma sa tačnošću $eps = 10^{-8}$ na poznate test probleme. Koristimo oznake $TimeA$ i $TimeN$ za procesorsko vreme potrebno u slučaju proširenog i normalnog sistema, respektivno. U sledećoj tabeli date su dimenzije problema i količnik potrebnih vremena za rešavanje problema.

Problem	Dimenzije	$TimeA/TimeN$
Adlittle	56×138	0.76
Afiro	27×51	0.95
Agg	488×615	0.52
Agg2	516×758	0.46
Agg3	516×758	0.57
Bandm	305×472	0.63
Blend	74×114	0.75
Israel	174×316	0.69
Kb2	43×68	0.69
Lotfi	153×366	0.64
Sc105	105×163	0.51
Sc205	205×203	0.55
Sc50b	50×78	0.72
Sc50a	50×78	0.66
Scagr7	129×185	0.61
Sctap1	300×660	0.78
Share2b	96×162	0.66
Stocfor1	117×165	0.60

Tabela 5.3.1.

Primetimo da je implementacija proširenog sistema brža u svim primerima iz Tabele 5.3.1. Takođe se može videti da količnik između procesorskih vremena potrebnih za prošireni i normalni sistem opada približno sa porastom dimenzije problema.

Primer 5.3.2 Za primer (4.2.2) primenom proširenog sistema dobijamo optimalnu vrednost `899.999999999` za obe početne tačke. Primenom normalnog sistema dobijamo istu optimalnu vrednost koristeći (2.2.3) startnu tačku, ali koristeći (2.2.2) za startnu tačku posle pet iteracija matrica postaje loše uslovljena.

Primer 5.3.3 U slučaju test primera *Degen2* početna matrica je loše uslovljena i kod proširenog sistema se javlja fenomen oscilovanja vrednosti xs. U prvih 15 iteracija dobijene su sledeće vrednosti:

```
{2944.86, 2936.47, 2941.46, 2948.47, 2946.99, 2954.17, 2960.11, 3014.65,
3027.11, 3026.22, 2965.13, 2954.41, 2055.41, 2970.52, 3015.56}.
```

Sa druge strane normalni sistem uspešno rešava primer *Degen2*. Suprotna situacija je kod test primera *Scfxm1* koji prošireni sistem rešava a normalni sistem ne može da reši.

U ovom poglavlju smo uporedili dve varijante Mehrotraovog primal dual algoritma koje se baziraju na proširenom i normalnom sistemu jednačina. Na primerima je pokazano da je prošireni sistem u opštem slučaju brži u odnosu na normalni sistem. Kako je tačnost slična u oba pristupa, to primena proširenog sistema ima bolje karakteristike. Ovde moramo napomenuti da ovi rezultati zavise od specifičnih metoda za rešavanje proširenog i normalnog sistema u programu MATHEMATICA. U velikom broju poznatih kodova Cholesky faktorizacija [78, 84] se koristi za rešavanje normalnog sistema jer se u mnogim primerima javljaju retke matrice [98]. Napomenimo da postoje i radovi u kojima se za rešavanje sistema predlažu iterativne metode [34, 8, 48]. Kako je napomenuto u [2, 112] prošireni sistem je numerički stabilniji i može se lako prilagoditi za rešavanje problema kvadratnog programiranja. Posebno, problemi koji sadrže slobodne promenljive se lakše rešavaju primenom proširenog sistema, dok primena normalnih jednačina zahteva nezgodnu

reformulaciju problema. Ali, primena proširenog sistema ima i izvesne nedostatke [112]:

- Algoritmi i softver za rešvanje simetričnih neodređenih sistema sa retkom matricom nisu tako dobro razvijeni i široko dostupni kao Cholesky kodovi za sisteme sa retkom matricom.

- U opštem slučaju zahteva više procesorskog vremena (u proseku, 50% -100% više) za izračunavanje iterativnog koraka u odnosu na Cholesky algoritm.

Prva primedba je od manje važnosti jer se veoma brzo razvija softver u toj oblasti [23, 5, 26]. Druga primedba ostaje i pored značajnog napretka u razvoju softvera. Kao što je pokazano, primena proširenog sistema u programu MATHEMATICA je čak i brža, posebno za probleme većih dimenzija. Na kraju napomenimo da program HOPDM u zavisnosti od osobina postavljenog problema određuje koji će se pristup primeniti. Primer 5.3.3 pokazuje da je u nekim slučajevima struktura problema presudna za uspešnu primenu izabranog pristupa.

6 Stabilizacija Mehrotraovog algoritma

U ovom poglavlju primenjujemo stabilizacionu proceduru na Mehrotraov primal dual algoritma, predloženu u [64] i [62]. Implementacija je izvršena u programu MATHEMATICA i dato je nekoliko test primera za komparaciju stabilizacione procedure i originalnog metoda. Rezultati iz ovog poglavlja su objavljeni u radovima [88, 89, 90, 100].

6.1 Teorijski uzroci numeričke nestabilnosti

Ranije smo pokazali da metodi unutrašnje tačke u t-toj iteraciji rešavaju sistem linearnih jednačina sa matricom oblika

$$AD_t^2A^T \text{ ili } \begin{bmatrix} 0 & A \\ A^T & -D_t^{-2} \end{bmatrix}.$$

Jedan od najozbiljnijih izvora numeričkih poteškoća je činjenica da u slučaju kada je skup optimalnih rešenja X^* degenerisan matrica teži da postane ekstremno loše uslovljena kako se približavamo optimalnom rešenju. Postoji više radova u kojima je proučavan sistem oblika

$$(AD_t^2A^T)u = AD_td.$$

Njegovo rešenje $u = (AD_t^2A^T)^{-1}AD_td$ ima zanimljivu osobinu da je uniformno ograničeno i pored moguće loše uslovljenosti matrice $AD_t^2A^T$. Prvi rezultat ovog tipa je dobio Dikin [19], koji je pokazao da je

$$\|u\| \leq \max_{J\in\Omega(A)} \|(A_J^T)^{-1}d_J\|$$

gde je $\Omega(A)$ skup indeksa kolona svih nesingularnih $m \times m$ podmatrica od A i A_J je podmatrica od A sa kolonama čiji indeksi su u J. Isti rezultat su dobili i Stewart [101] i Ben-Tal, Teboulle [9]. Stewart je još pokazao da je

$$\|A^T(AD_t^2A^T)^{-1}AD_t\|$$

uniformno ograničeno u odnosu na t i odredio eksplicitno ograničenje u zavisnosti od singularnih vrednosti matrice A. I pored toga, činjenica da je rešenje ograničeno ne povlači da će izračunavanje tog rešenja biti numerički stabilno. U primal-dual agoritmima se obično za rešavanje sistema koristi Cholesky dekompozicija. Ona se često stabilizuje heuristikom po kojoj se pivot elementi manji od granične vrednosti ε izostavljaju. Pod izvesnim pretpostavkama Wright [115] je pokazao da ta heuristika ima teorijsku osnovu, tj. da je rastojanje između tačnog i približnog rešenja ograničeno sa $\sqrt{\varepsilon}$. Međutim, u radovima [6, 64] je pokazano da u slučaju degeneracije metod unutrašnje tačke može da postane loše uslovljen, čak i u najjačim implementacijama, kao što su PCx [14, 70, 112] i HOPDM [38]. Takođe, u radovima [6, 64] je data stabilizaciona tehnika koja se zasniva na Gaussovoj eliminaciji i prevodi originalni problem u njemu ekvivalentan oblik pri čemu je loša uslovljenost izbegnuta. U nastavku opisujujemo implementaciju stabilizovanog algoritma koja rešava klasu loše uslovljenih test problema iz [6, 64] koje programi PCx i HOPDM nisu uspeli da reše.

Neka je $\{(x^t, \lambda^t, s^t)\}$ niz generisan primal-dual algoritmom i neka je $\{AD_t^2A^T\}$ odgovarajući niz matrica definisanih u *Koraku* 4. Prema Propoziciji 2.1 iz [64], u slučaju kada je optimalna površ X^* degenerisana sledi da $\text{cond}(AD_t^2A^T) \to \infty,\ t \to \infty$. To znači da matrice sistema (2.1.1) i (2.1.2) teže da postanu ekstremno loše uslovljene i da neka stabilizaciona procedura mora biti primenjena. Stabilizaciona procedura koju želimo da primenimo koristi neke pretpostavke o uniformnosti niza $\{(x^t, \lambda^t, s^t)\}$.

Predpostavka 6.1.1 *Neka je* $(x^t, \lambda^t, s^t), t = 1, 2, \ldots$ *niz generisan primal-dual metodom unutrašnje tačke, sa osobinom da sve tačke nagomilavanja od* $\{(x^t, \lambda^t, s^t)\}$ *pripadaju* $riX^* \times riY^*$. *Tada postoje pozitivne konstante* u *i* v *tako da je*

$$u \le (x_i^t/s_i^t)/(x_j^t/s_j^t) \le 1/u, \quad i, j \in \mathcal{B} \quad \textit{and}$$
$$v \le (x_i^t/s_i^t)/(x_j^t/s_j^t) \le 1/v, \quad i, j \in \mathcal{N}.$$

Pretpostavka 6.1.1 je blisko povezana sa pojmom maksimalnog komplementarnog niza (MCS) koji su uveli Guller i Ye u radu [47]. Svaki MCS niz zadovoljava Pretpostavku 6.1.1 [64] tako da to važi za većinu metoda unutrašnje tačke.

Glavna ideja sledeće procedure, preuzete iz [64], je u transformaciji originalnog problema u ekvivalentni oblik u kome se problemi stabilnosti ne javljaju ukoliko se primeni pravilna strategija za izračunavanje traženog pravca.

Procedura 6.1.1

Korak 1. Za dato t neka su $j(1), \ldots, j(n)$ indeksi koji zadovoljavaju uslov $(x_{j(1)}^t/s_{j(1)}^t) \ge \cdots \ge (x_{j(n)}^t/s_{j(n)}^t)$, i neka je $\hat{h} \in \{1, \ldots, n-1\}$ najmanji indeks tako da je $(x_{j(\hat{h}+1)}^t/s_{j(\hat{h}+1)}^t) < 1$.

Korak 2. Urediti kolone matrice A i c^T tako da $j(p)$-ta kolona dođe na p-tu poziciju, $p = 1, \ldots, n$. Set $p = 1, q = 1$.

Korak 3. Odrediti $|a_{i(p),p}| = \max\{|a_{ip}|, i = 1, \ldots, m\}$. Ako je $|a_{i(p),p}| \neq 0$ preći na *Korak 5.*

Korak4. Ako je $|a_{i(p),p}| = 0$ i $p = \hat{h}$ staviti $\hat{r} = q - 1$ i RETURN. U suprotnom zameniti p sa $p + 1$ i preći na *Korak 3.*

Korak 5. Zameniti $i(p)$-tu i q-tu vrstu u A i b. Koristeći a_{qp} kao pivot element eliminisati $a_{ip}, i = q + 1, \ldots, m$ (ako postoje) i c_p.

Korak 6. Ako je $q = \min\{\hat{h}, m\}$ ili $p = \hat{h}$ staviti $\hat{r} = q$ i RETURN. U suprotnom zameniti p sa $p + 1$, q sa $q + 1$ i preći na *Korak 3.*

Ako se Procedura 6.1.1 primenjuje u $\hat{t}$-tom koraku metoda unutrašnje tačke i ako je $\hat{r} < m$, parametare problema transformiše u oblik

$$\hat{A} = \begin{bmatrix} P & Q \\ O & R \end{bmatrix}, \hat{b}, \hat{c}$$

gde je P matrica tipa $\hat{r} \times \hat{h}$, $\text{rank}(P) = \hat{r}$, matrica O je nula matrica tipa $(m-\hat{r}) \times \hat{h}$, Q i R su matrice tipa $\hat{r} \times (n - \hat{r})$ i $(m - \hat{r}) \times (n - \hat{r})$ respektivno. Napomenimo da ova transformacija jednostavno permutuje komponente od x^t i s^t.

Neka je

$$AD_t^2 A^T = d^t \tag{6.1.1}$$

sistem koji treba rešiti u t-toj iteraciji. Za $t \geq \hat{t}$ koeficijenti matrice u (6.1.1) imaju oblik

$$\hat{A}\hat{D}_t^2 \hat{A}^T = \begin{bmatrix} F_t & G_t \\ G_t^T & H_t \end{bmatrix}$$

i F_t, G_t i H_t su respektivno $\hat{r} \times \hat{r}$,$\hat{r} \times (m - \hat{r})$ i $(m - \hat{r}) \times (m - \hat{r})$ matrice. Tada važi sledeće tvrđenje [64].

Teorema 6.1.1 *Pretpostavimo da metod unutrašnje tačke zadovoljava Pretpostavku 6.1.1 i da je Procedura 6.1.1 primenjena periodično, tj. u koracima $t=\hat{t}$, $t=2\hat{t}, \ldots$. Tada postoji konstanta C nezavisna od t tako da je*

$$cond(F_t) \leq C \text{ i } cond(H_t - G_t^T F_t^{-1} G_t) \leq C.$$

Prethodna teorema tvrdi da Procedura 6.1.1 rastavlja loše uslovljenu matricu u dobro uslovljene podmatrice. To nam osigurava stabilnost u daljim numeričkim izračunavanjima.

Rezultati Teoreme 6.1.1 se koriste u Algoritmu 6.1.1 na sledeći način. Za rešavanje sistema (2.1.1)

$$AD^2 A^T \Delta\lambda^{aff} = -r_b - A(S^{-1} X r_c - S^{-1} r_{xs})$$

koristimo sledeću dekompoziciju matrice AD^2A^T:

$$AD^2 A^T = \begin{bmatrix} F_t & G_t \\ G_t^T & H_t \end{bmatrix}. \tag{6.1.2}$$

Ako razbijemo svaki od vektora $\Delta\lambda^{aff}$ i $-r_b - A(S^{-1}Xr_c - S^{-1}r_{xs})$ u dva podesna bloka, označena sa δ_1, δ_2 i v_1, v_2, respektivno, u skladu sa blok dekompozicijom (6.1.2), sistem (2.1.1) može biti zamenjen ekvivalentnim sistemom:

$$\begin{aligned} \delta_1 + F_t^{-1}G_t\delta_2 &= F_t^{-1}v_2, \\ (H_t - G_t^T F_t^{-1}G_t)\delta_2 &= v_2 - G_t^T F_t^{-1}v_1. \end{aligned} \tag{6.1.3}$$

Prema Teoremi 6.1.1 sistem (6.1.3) sadrži samo dobro uslovljene matrice.

Na kraju navodimo teoremu koja daje oblik dualnog problema modifikovanog primalnog problema.

Teorema 6.1.2 *Pretpostavimo da matrica problema linearnog programiranja ima oblik*

$$A = \begin{bmatrix} a_{11} & \dots & a_{1,p-1} & a_{1p} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & a_{q-1,p-1} & a_{q-1,p} & \dots & a_{q-1,n} \\ 0 & \dots & 0 & a_{qp} & \dots & a_{qn} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & a_{mp} & \dots & a_{mn} \end{bmatrix}.$$

Ako upotrebimo a_{qp} kao pivot element za eliminaciju a_{ip} i saglasno transformišemo $b_i, i = q+1, \dots, m$, tada dual transformisanog problema

$$\min \ c^T x \quad \text{p.o. } A'x = b', \quad x \geq 0,$$

ima oblik

$$\max \ b'^T \lambda' \quad \text{p.o. } A'^T \lambda' + s = c, \quad s \geq 0,$$

gde je

$$\begin{aligned} \lambda'_i &= \lambda_i, \quad i \neq q, \\ \lambda'_q &= \lambda_q + \sum_{i=q+1}^{m} \frac{a_{ip}}{a_{qp}} \lambda_i. \end{aligned} \tag{6.1.4}$$

Dokaz. Treba pokazati $b'^T \lambda' = b^T \lambda$ i $A'^T \lambda' = A^T \lambda$. Kako je

$$\begin{aligned} b'_i &= b_i, \quad i = 1, \dots, q, \\ b'_i &= b_i - \frac{a_{ip}}{a_{qp}} b_q, \quad i = q+1, \dots, m, \end{aligned}$$

vrednost dualne ciljne funkcije je

$$\begin{aligned} {b'}^T\lambda' &= \sum_{i=1}^{m} b'_i\lambda'_i \\ &= \sum_{i=1}^{q-1} b_i\lambda_i + b_q\left(\lambda_q + \sum_{i=q+1}^{m}\frac{a_{ip}}{a_{qp}}\lambda_i\right) + \sum_{i=q+1}^{m}\left(b_i - \frac{a_{ip}}{a_{qp}}b_q\right)\lambda_i \quad (6.1.5) \\ &= \sum_{i=1}^{q} b_i\lambda_i + \sum_{i=q+1}^{m}\frac{a_{ip}}{a_{qp}}b_q\lambda_i + \sum_{i=q+1}^{m} b_i\lambda_i - \sum_{i=q+1}^{m}\frac{a_{ip}}{a_{qp}}b_q\lambda_i = b^T\lambda. \end{aligned}$$

Dalje, elementi ${A'}^T\lambda'$ imaju oblik $\sum_{i=1}^{m} a'_{ij}\lambda'_i, \quad j = 1,\ldots,n$. Sada, koristeći (6.1.4) i $a'_{ij} = a_{ij} - \frac{a_{qj}}{a_{qp}}a_{ip}, j = q+1,\ldots,m$, imamo

$$\begin{aligned} &\sum_{i=1}^{m} a'_{ij}\lambda'_i = \sum_{i=1}^{m} a_{ij}\lambda_i, \quad j = 1,\ldots,p-1, \\ &\sum_{i=1}^{m} a'_{ip}\lambda'_i = \sum_{i=1}^{q-1} a_{ip}\lambda_i + a_{qp}\left(\lambda_q + \sum_{i=q+1}^{m}\frac{a_{ip}}{a_{qp}}\lambda_i\right) = \sum_{i=1}^{m} a_{ip}\lambda_i, \\ &\sum_{i=1}^{m} a'_{ij}\lambda'_i = \sum_{i=1}^{q-1} a_{ij}\lambda_i + a_{qj}\left(\lambda_q + \sum_{i=q+1}^{m}\frac{a_{ip}}{a_{qp}}\lambda_i\right) + \sum_{i=q+1}^{m}\left(a_{ij} - \frac{a_{qj}}{a_{qp}}a_{ip}\right)\lambda_i \\ &\qquad = \sum_{i=1}^{m} a_{ij}\lambda_i, \quad j = p+1,\ldots,n. \end{aligned}$$

Prema tome, jednakost ${A'}^T\lambda' = A^T\lambda$ važi. □

6.2 Implementacija stabilizacione procedure

Sada opisujemo implementaciju Procedure 6.1.1 i odgovarajuću modifikaciju Mehrotraovog primal-dual metoda.

Formalni parametri se definišu sa:

$a\_, b\_, lam\_$: data matrica A i vektori b i λ, respektivno.
$xs\_$: vektor x/s čije koordinate su x_i/s_i, $i = 1,\ldots,k+n$.
$c\_, lam\_$: vektor ciljne funkcije i vektor λ u dualnom problemu.

Matrica na koju primenjujemo Gaussovu eliminaciju je oblika

$$[\,A\ b\,] = [\,a\ b\,].$$

```
<<LinearAlgebra‘MatrixManipulation‘
Elimination[a_, b_, xs_, c_, lam_] :=
Module[{an=a, xss={}, pom={}, indper={},n,m,p,q,h,L,maxi,cn=c,l=lam},
```

```
    {m,n}=Dimensions[a];    brelim++; inp = 1;
   (*  Step 1 *)
    indper = Sort[Range[n], xs[[#1]] > xs[[#2]] &];
    xss = xs[[indper]];
    h = 1;
     While[xss[[h]]>=1 && h<n, h++];
    h = h-1;  L = True;
    If[h==n, L=False; q=n];
     (* Step 2 *)
    an=Transpose[Append[Transpose[a][[indper]],b]];
    cn=cn[[indper]];
     p = 1; q=1;
     While[(L && (q<m)),
         (* Step 3 *)
         pom = Drop[pom, q-1];
         maxi = Position[pom, Max[pom]][[1,1]] + q-1;
         If[an[[maxi, p]]==0,
             (* Step 4 *)
             If[p==h,          (* Then *)
                 Return[List[an, indper, q-1, cn, l]],
                 p++;      (* Else *)
         ];
             ,                  (* Else *)
             (* Step 5 *)
             If[maxi != q,
                 pom=an[[maxi]];  an[[maxi]]=an[[q]];   an[[q]]=pom;
                 pom=l[[maxi]];   l[[maxi]]=l[[q]];     l[[q]]=pom;
             ];
            For[i=q+1, i<=m, i++,
                 l[[q]]=l[[q]]+N[ l[[i]]*an[[i,p]]/an[[q,p]], 20];
             ];
             (* Adaptation of the vector $\lambda $*)
             Do[an[[i]]=an[[i]]-N[an[[i, p]], 20]*N[an[[q]], 20]/N[an[[q, p]], 20],
                                {i,q+1,m}    (* Gaussian elimination *)
             ];
             (* Step 6 *)
            L = Not[(q == Min[h, m]) || (p==h)];
             If[L, p++; q++;];
         ];
     ];
     Return[List[an, indper, q, cn, l]];
];  }
```

Rezultat funkcije *Elimination*[] je lista čiji su elementi definisani u nastavku.

`an, cn, l`: matrica a i vektori c i l respektivno, posle stabilizacione procedure,

`indper`: skup indeksa $\{j_1, \ldots, j_n\}$ sortirani u skladu sa *Korakom* 1 Procedure 6.1.1.

Eliminacija se primenjuje u *Koraku* 4. Posle primene eleiminacije, transformisane matrice se koriste za rešavanje sistema u *Koraku* 4 i *Koraku* 8. Stabilizacija se prvi put primenjuje kada je matrica AD^2A^T loše uslovljena. U svim ostalim slučajevima, eliminacija se primenjuje kada je jedna od matrica F_t ili $H_t - G_t^T F_t^{-1} G_t$ loše uslovljena. Za izračunavanje faktora uslovljenosti matrice koristimo funkciju $MatrixConditionNumber[mat]$, koja izračunava faktor uslovljenosti u odnosu na

beskonačnu normu matrice *mat*. Koristimo parametar *eps*3 koji određuje graničnu vrednost od koje primenjujemo eliminaciju, koji je definisan sledećom heuristikom:

```
If[MatrixConditionNumber[N[a.d.Transpose[a]]]<=10^10,
     eps3=10*MatrixConditionNumber[N[a.d.Transpose[a]]],
     eps3=10^16
];
```

Modifikacija *Koraka 4*:
Da bi rešili sistem

$$AD^2A^T\Delta\lambda^{aff} = -r_b - A(S^{-1}Xr_c - S^{-1}r_{xs})$$

koristimo dekompoziciju matrice AD^2A^T na sledeći oblik:

$$AD^2A^T = \begin{bmatrix} F_t & G_t \\ G_t^T & H_t \end{bmatrix}.$$

Ako razdvojimo svaki od vektora $\Delta\lambda^{aff} = Dlaff$ i $-r_b - A(S^{-1}Xr_c - S^{-1}r_{xs}) = vRight$ u dva pogodna bloka, označena sa δ_1, δ_2 i v_1, v_2, respektivno, tada moramo rešiti sledeći linearni sistem:

$$\delta_1 + F_t^{-1}G_t\delta_2 = F_t^{-1}v_2, \tag{6.2.1}$$
$$(H_t - G_t^TF_t^{-1}G_t)\delta_2 = v_2 - G_t^TF_t^{-1}v_1. \tag{6.2.2}$$

Taj korak je implementiran u sledećem kodu. Vrednost $pok = 0$ parametra *pok* označava da eliminacija nije prethodno primenjivana. Sledeća rutina je upotrebljena za početak stabilizacije.

```
is=DiagonalMatrix[N[1/s,20]];   d= DiagonalMatrix[N[x/s,20]];
If[pok==0,
  If[( MatrixConditionNumber[N[a.d.Transpose[a]]]>eps3)&&(NumStep>1),
     pok=1;  (* $Cond(A)$ causes the elimination *)
      pom=Elimination[a,b,x/s,ce,l];
      vR=pom[[3]];  ce=pom[[4]]; l=pom[[5]];
      a=N[Transpose[Drop[Transpose[pom[[1]]],-1]]];
      b=N[Flatten[Take[Transpose[pom[[1]]],-1]]];
      pom=pom[[2]];
      permut=permut[[pom]];
      x=x[[pom]]; s=s[[pom]]; rc=rc[[pom]];
      rb=N[a.x-b,20];  rc=N[Transpose[a].l+s-ce,20];
      Xmat=DiagonalMatrix[x];   Smat=DiagonalMatrix[s];
      rxs=N[Xmat.Smat.Table[1,{n}],20];
      is= DiagonalMatrix[N[1/s,20]];   d= DiagonalMatrix[N[x/s,20]];
   ];,

  If[ ((vR<Dimensions[a][[1]])&&(MatrixConditionNumber[vH-Transpose[vG].vF.vG]>eps3))
        ||( MatrixConditionNumber[N[vF]]>eps3),
      (* $Cond(vF)$ or $cond(vH-vG^T VF^{-1}VG)$ causes the elimination *)
      pom=Elimination[a,b,x/s,ce,l];
      vR=pom[[3]];  ce=pom[[4]]; l=pom[[5]];
      a=N[Transpose[Drop[Transpose[pom[[1]]],-1]]];
```

```
      b=N[Flatten[Take[Transpose[pom[[1]]],-1]]];
      pom=pom[[2]];
      permut=permut[[pom]];
      x=x[[pom]]; s=s[[pom]];
      rb=N[a.x-b,20];   rc=N[Transpose[a].l+s-ce,20];
      Xmat=DiagonalMatrix[x];   Smat=DiagonalMatrix[s];
      rxs=N[Xmat.Smat.Table[1,{n}],20];
      is= DiagonalMatrix[N[1/s,20]];   d= DiagonalMatrix[N[x/s,20]];
   ];
]; (* Criteria for the stabilization starting are verified, *)
  (* and the elimination is possibly started *)  }
```

Sledeća rutina se koristi za proveru da li je rešavan sistem oblika (2.1.1) ili sistem oblika (6.2.1)-(6.2.2). Jednačine (2.1.2) i (2.1.3) ostaju iste u oba slučaja.

```
 If[pok==0,  (* The case when the stabilization is not already applied *)
  If[(vR<Dimensions[a][[1]])&&( MatrixConditionNumber[N[a.d.Transpose[a]]]>eps3)
                        &&(NumStep>1),
       vA=a.d.Transpose[a];
       (* Submatrix F *)   vF=SubMatrix[vA,{1,1},{vR,vR}];   vF=N[Inverse[vF],20];
       (* Submatrix G *)   vG=SubMatrix[vA,{1,vR+1},{vR,Dimensions[vA][[2]]-vR}];
       (* Submatrix H *)
       vH=SubMatrix[vA,{vR+1,vR+1},
                    {Dimensions[vA][[1]]-vR,Dimensions[vA][[2]]-vR}];
       vRight=N[-rb-a.(is.Xmat.rc-is.rxs),20];
       vRes=N[LinearSolve[vH-Transpose[vG].vF.vG,
                  Drop[vRight,vR]-Transpose[vG].vF.Take[vRight,vR]],20];
       Dlaff=N[vF.Take[vRight,vR]-vF.vG.vRes,20];
       Dlaff=Join[Dlaff,vRes];
       hhh=vH-Transpose[vG].vF.vG;
        ,
     Dlaff=N[LinearSolve[a.d.Transpose[a],-rb-a.(is.Xmat.rc-is.rxs)],20 ];
   (* The elimination is applied *);
        ];,  (*pok=1*)
(* The case when the stabilization is already applied *)
   vA=a.d.Transpose[a];
   (* Submatrix F *)  vF=SubMatrix[vA,{1,1},{vR,vR}];  vF=N[Inverse[vF],20];
   (* Submatrix G *)  vG=SubMatrix[vA,{1,vR+1},{vR,Dimensions[vA][[2]]-vR}];
   (* Submatrix H *)
    If[vR<Dimensions[a][[1]],
      vH=SubMatrix[vA,{vR+1,vR+1},{Dimensions[vA][[1]]-vR,Dimensions[vA][[2]]-vR}];
      vRight=N[-rb-a.(is.Xmat.rc-is.rxs),20];
     vRes=N[LinearSolve[vH-Transpose[vG].vF.vG,
                 Drop[vRight,vR]-Transpose[vG].vF.Take[vRight,vR]],10];
      Dlaff=N[vF.Take[vRight,vR]-vF.vG.vRes,20];
     Dlaff=Join[Dlaff,vRes];
      hhh=vH-Transpose[vG].vF.vG;
     ,
     (* Transformation is not applied *)
      Dlaff=N[LinearSolve[a.d.Transpose[a],
      ];
   ];
];
   (* Solve the systems (2.2) and (2.3) *)
Dsaff=N[-rc-Transpose[a].Dlaff,20];
Dxaff=N[-is.(rxs+Xmat.Dsaff),20];   }
```

Modifikacija *Koraka* 8:

Analogno kao u *Koraku* 4, linearan sistem oblika (6.2.1)-(6.2.2) treba rešiti umesto sistema (2.1.4) u *Koraku* 8.

Da bi rešili linearan sistem $AD^2A^T\Delta\lambda^{cor} = AS^{-1}r_{xs}$ koristimo razbijanje matrice AD^2A^T na oblik (6.2.2) kao i odgovarajuće razlaganje vektora $AS^{-1}r_{xs}$. Razlažemo svaki od vektora $\Delta\lambda^{aff} = Dlaff$ i $AS^{-1}r_{xs} = vRight$ u dva odgovarajuća bloka, označena sa δ_1, δ_2 i v_1, v_2, respektivno. Sledeća rutina se koristi za proveru da li je sistem oblika (2.1.4) ili sistem oblika (6.2.1)-(6.2.2) rešavan. Jednačine (2.1.5) i (2.1.6) ostaju iste u oba slučaja.

```
DXaff=DiagonalMatrix[Dxaff];     DSaff=DiagonalMatrix[Dsaff];
rxs=N[-mi*SigCent,20]*Table[1,{n}]+N[DXaff.DSaff,20].Table[1,{n}];
If[pok==0, (* The case when the stabilization is not already applied *)
  If[(vR<Dimensions[a][[1]])&&( MatrixConditionNumber[N[a.d.Transpose[a]]]>eps3)
                    &&(NumStep>1),
    (* Stabilization is used because of $cond(A)$ *)
     vRight=N[a.is.rxs,20];
    vRes=N[LinearSolve[vH-Transpose[vG].vF.vG,
               Drop[vRight,vR]-Transpose[vG].vF.Take[vRight,vR]],20];
     Dlcc=N[vF.Take[vRight,vR]-vF.vG.vRes,20];    Dlcc=Join[Dlcc,vRes];
    ,
     (* Modification is not applied *)
     Dlcc=N[LinearSolve[a.d.Transpose[a], a.is.rxs],20 ];
  ];
  ,(*pok=1*)
(* The case when the stabilization is already applied *)
   (* $Cond(vF)$ of $cond(vH-vG^T VF^{-1}VG$ causes the elimination *)
   If[vR<Dimensions[a][[1]],
      vRight=N[a.is.rxs,20];
      vRes=N[LinearSolve[vH-Transpose[vG].vF.vG,
                 Drop[vRight,vR]-Transpose[vG].vF.Take[vRight,vR]],20];
      Dlcc=N[vF.Take[vRight,vR]-vF.vG.vRes,20];
      Dlcc=Join[Dlcc,vRes];
      ,   (* Stabilization is not applied *)
      Dlcc=N[LinearSolve[a.d.Transpose[a],a.is.rxs],20];
   ];
];           (* Solve systems (2.5) and (2.6)  *)
Dscc=N[-Transpose[a].Dlcc,20];
Dxcc=N[-is.(rxs+Xmat.Dscc),20]; }
```

Takođe, *Korak* 12 je modifikovan prinudnim izlaskom iz while ciklusa u slučaju kada je uslov

```
x.s>x1.s1 && x1.s1<1
```

zadovoljen, gde $x1$ i $s1$ označavaju vrednosti vektora x i s u prethodnom iterativnom koraku.

U slučaju da je while ciklus završen na normalan način, rezultat se izračunava pomoću sledećeg koda.

```
   pom=Range[Length[permut]];
   pom=Sort[pom,permut[[#1]]<permut[[#2]]&];
   x= x[[pom]];      ce=ce[[pom]];
   z=-N[ce.x,20]
   Return[{N[z,20],Take[N[x,20],k1]}]; }
```

U slučaju kada je primenjen prinudan izlazak, koristimo jednostavnu korektivnu proceduru. Izlazna vrednost je tada generisana identičnim kodom, zamenjujući promenljive x i s njihovim prethodnim vrednostima $x1$ i $s1$, respektivno.

6.3 Numerički primeri

U ovom poglavlju dajemo numeričke rezultate primene naše implementacije Mehrotrinog primal-dual algoritma. Dve varijante algoritma su implementirane u programskom paketu MATHEMATICA [108, 109, 110, 111]. Prva verzija je direktna primena Algoritma 2.1. Modifikovana verzija je kombinacija Algoritma 2.1 i Procedure 6.1.1. Ove dve verzije su upoređene sa poznatim kodovima koji se zasnivaju na primal-dual algoritmu PCx [14], MOSEK [1] i LIPSOL [123]. Test primeri sadrže klasu degenerisanih test problema iz [6]. Da bi smo ilustrovali činjenicu da je simpleks metod numerički stabilan i na degenerisanim problemima, isti primeri su rešavani i kodom koji se zasniva na simpleks metodu. Implementacija Mehrotraovog metoda i stabilizacije je takođe primenjena na neke *Netlib* probleme.

Primer 6.3.1 Primenom modifikovanog metoda na problem (4.2.1) dobijamo optimalnu vrednost $4.689819293851613\times10^{-17}$, i optimalnu tačku

$\{1000.0000000000003, 0.009999999999990915, 4.465972084095311 \times 10^{-19},$
$9.290319146020935 \times 10^{-18}, 2.6977763021245666 \times 10^{-17}\}$.

Originalan Mehrotraov primal-dual metod ne može da reši ovaj problem: zaustavlja se posle 10 iteracija sa sistemom linearnih jednačina koji nema rešenja. Još više, interesantno je da PCx daje optimalnu vrednost $1.48461670 \times 10^{-10}$ i nedopustiv status.

Primer 6.3.2 Sledeći primer je test problem (4.2.2). Modifikovan metod, koristeći preciznost 10^{-15}, daje optimalnu vrdnost 899.9999999999999 i optimalnu tačku

$\{1.883068181443366\times10^{-14}, 14.999999999999979, 1.1048249711401044\times10^{-14}\}$

posle 14 iteracija. Originalni metod posle pet iteracija nailazi na loše uslovljenu matricu i staje posle sedam iteracija sa sistemom linearnih jednačina koji nema rešenja.

Primer 6.3.3 U radovima [6], [64], data je klasa loše uslovljenih test problema. U svim slučajevima (osim *Hager 1*) vrednost ciljne funkcije je jednaka nuli. U [6] neki od problema su rešavani sa PCx, HOPDM i solverom iz Microsoft Excel 8.0a. U Tabeli 6.3.1 dajemo vrednosti ciljne funkcije primalnog problema dobijene primenom PCx. U većini slučajeva rezultati nisu tačni. Pored toga, primetimo da rezultati zavise od procesora.

Problem	PCx (Pentium IV)	PCx (Pentium III)	Br. it.	Status
Little	1.48461670e-010	1.48461670e-010	6	Infeasible
Hager	7.75911235e-005	3.16756489e-003	13	Unknown
Hager1	7.85089570e-002	5.64193210e-002	13	Unknown
Hager2	5.54632489e-003	4.11184994e-003	13	Unknown
07-20-02	3.52750000e002	0	11	Infeasible
10-20-06	1.10673751e-014	6.18444502e-010	3	Unknown
10-20-11	-2.33440078e-005	-1.72408763e-005	10	Infeasible
10-20-12	3.73678721e-006	2.69583792e-003	12	Infeasible
10-20-13	2.04945591e-005	1.44417032e-007	12	Unknown
15-30-03	8.30517615e-009	6.22298978e-002	8	Unknown
15-30-04	-1.62595431e004	-1.07999616e004	11	Unknown
15-30-06	3.50522624e-002	1.01731404e-004	8	Unknown
15-30-07	9.72422879e003	2.93026691e003	10	Unknown
15-60-09	-1.20545931e004	-6.55982227e003	7	Unknown
20-40-05	4.13614433e003	4.19202481e000	9	Unknown
30-60-05	-2.58378496e-004	2.37138986e-001	5	Optimal

Tabela 6.3.1

Mnogo bolji rezultati su dobijeni korišćenjem kodova LIPSOL (Tabela 6.3.2) i MOSEK (Tabela 6.3.3). Naime, oba koda koriste rutine za generisanje bazičnog optimalnog rešenja koje se značajno razlikuje od rešenja koje je generisano primal-dual metodom. To se može videti u Tabeli 6.3.3, gde kolone MOSEK IP i MOSEK BAS daju optimalno prima-dual rešenje i vrednost bazičnog rešenja, respektivno. I pored toga, LIPSOL ne može da reši problem 07-20-02, i daje pogrešno rešenje za 10-20-11. MOSEK primal-dual rešenja su tačna samo za neke probleme. Sa druge strane, MOSEK bazično rešenje ima manju tačnost od rešenja iz Tabele 6.3.4 koja su dobijena primenom stabilizacione procedure.

Problem	LIPSOL	Status
Little	3.4177496495e-008	Converged
Hager	3.6578521551e-010	Converged
Hager1	8.0000200580e-002	Converged
Hager2	2.3283064365e-010	Converged
07-20-02	-	Not converged
10-20-06	1.5773489034e-010	Converged
10-20-11	-2.1375811612e-001	Converged
10-20-12	1.0585697416e-020	Likely converged but error $\geq$ tolerance
10-20-13	3.6362725826e-021	Likely converged but error $\geq$ tolerance
15-30-03	1.3295987516e-009	Converged
15-30-04	1.1530965567e-003	Converged
15-30-06	3.2699876586e-009	Converged
15-30-07	-5.9604644775e-007	Converged
15-60-09	-3.2091122637e-004	Converged
20-40-05	3.9942264557e-003	Converged
30-60-05	-5.2619215398e-007	Converged

Tabela 6.3.2

Problem	MOSEK IP	IP Status	MOSEK BAS
Little	0.00000000e+000	Feasible	0.00000000e+000
Hager	1.26560371e-005	Feasible	4.07935617e-012
Hager1	8.00000000e-002	Feasible	8.00000000e-002
Hager2	4.78001311e-007	Feasible	4.07453626e-010
07-20-02	-9.79658597e003	Unknown	-1.44988298e-005
10-20-06	1.09442797e-004	Feasible	3.09810931e-008
10-20-11	2.30185688e-005	Feasible	-4.53250825e-008
10-20-12	1.44992676e007	Unknown	-3.15392512e-004
10-20-13	2.2441842e004	Unknown	-3.00129e-004
15-30-03	2.95747574e-005	Feasible	-7.16417263e-018
15-30-04	1.72796838e004	Unknown	3.15442681e-004
15-30-06	1.478620e004	Unknown	-1.41331e-004
15-30-07	-3.11335992e007	Unknown	3.81857157e-004
15-60-09	-1.50742544e007	Unknown	-4.36755330e-004
20-40-05	-1.68255476e007	Unknown	3.66744919e-004
30-60-05	3.69986004e-005	Feasible	-8.01040996e-008

Tabela 6.3.3

Naša implementacija Mehrotraovog metoda takođe ne rešava ove probleme. Svi ti problemi su rešeni modifikovanim metodom primenom stabilizacione procedure. U Tabeli 6.3.4 dati su odgovarajući rezultati. Kolona *Primal-dual* sadrži rezultate generisane našom implementacijom Mehrotraovog algoritma, i kolona označena sa *Modification* sadrži rezultate dobijene sa stabilisanom verzijom Mehrotraovog algoritma. Kolona MarPlex sadrži rezultate dobijene primenom naše implementaije simpleks metoda u programskom jeziku Visual Basic [99]. Rezultati označeni sa * su dobijeni korišćenjem koda CPLEX.

Problem	Dimensions	Primal-dual	Modification	No. It.	MarPlex
Little	3×5	–	4.68981929e-017	14	0.00000000e000
Hager	18×27	0.01177870e000	2.31595656e-013	19	0.00000000e000
Hager1	18×27	0.04643150e000	0.08000000e000	19	0.08000000e000
Hager2	18×27	983.985462e000	-1.13927731e-012	17	0.00000000e000
07-20-02	9×22	-43326.7103e000	1.71702249e-004	13	-2.12028566e-008
10-20-06	10×20	–	5.21750702e-014	16	0.00000000e000
10-20-11	12×20	–	-1.65960575e-010	26	1.01339538e-007
10-20-12	12×20	0.98274344e000	6.06765507e-015	12	0.00000000e000
10-20-13	12×22	0.62207356e000	6.11841761e-015	12	0.00000000e000
15-30-03	17×32	0.98876311e000	5.43630863e-013	14	-9.41872713e-009
15-30-04	16×31	46.9218065e000	-4.44262662e-004	12	3.63477739e-007
15-30-06	16×31	0.00615304e000	2.02024065e-007	9	3.74187948e-007
15-30-07	17×32	-96148.2245e000	-5.73460485e-005	12	3.63477739e-007
15-60-09	16×61	279069.384e000	6.69293887e-009	15	-4.42993286e-005*
20-40-05	22×42	-89.0574426e000	-1.72587897e-006	11	1.05961369e-004*
30-60-05	30×59	2.65006827e007	-4.23620577e-009	15	0.00000000e000

Tabela 6.3.4

Primer 6.3.4 Naš kod je takođe primenjen na nekim *Netlib* test problemima. Originalni i modifikovani Mehrotraov algoritam daju skoro identične rezultate za skoro sve probleme. Primetimo da naša implementacija Mehrotraov primal-dual algoritma ne može da reši problem $Scfxm1$, ali sa modifikacijom problem $Scfxm1$ je rešen. Takođe, problem *Bandm* nije korektno rešen bez stabilizacije. Sa druge strane, primetili smo mala numerička odstupanja kada se stabilizacija primeni

na probleme *Agg*, *Agg*2, *Blend*, *Sctap*1 i *Share*2*b*. To je verovatno posledica grešaka koje indukuje Gaussova eliminacija. Zaključujemo da je primena stabilizacione procedure razumna samo na loše uslovljenim problemima.

Problem	Primal-dual	Modification
Adlittle	2.2549496316e005	2.2549496316e005
Afiro	-4.6475314286e002	-4.6475314286e002
Agg	-3.5991767284e007	-3.5991767295e007
Agg2	-2.0239252356e007	-2.0239252351e007
Agg3	1.0312115935e007	1.0312115935e007
Bandm	-1.5862801770e002	-1.5862801845e002
Blend	-3.0812149846e001	-3.0812149691e001
Capri	2.6900129138e003	2.6900129138e003
Degen2	-1.4351780000e003	-1.4351780000e003
Israel	-8.9664482186e005	-8.9664482186e005
Kb2	-1.7499001299e003	-1.7499001299e003
Lotfi	-2.5264706062e001	-2.5264706062e001
Recipe	-2.6661600000e002	-2.6661600000e002
Sc105	-5.2202061212e001	-5.2202061212e001
Sc205	-5.2202061212e001	-5.2202061212e001
Sc50a	-6.4575077059e001	-6.4575077059e001
Sc50b	-7.0000000000e001	-7.0000000000e001
Scagr7	-2.3313898243e006	-2.3313898243e006
Scfxm1	–	1.8416759028e004
Sctap1	1.4122500000e003	1.4122501183e003
Share2b	-4.1573224072e002	-4.1573224074e002
Stocfor1	-4.1131976219e004	-4.1131976219e004

Tabela 6.3.5

Na kraju napomenimo da, kako se metod bazira na Gaussovoj metodi eliminacije, česta primena stabilizacione procedure može da prouzrokuje akumulaciju grešaka zaokruživanja. Zbog toga je prikazanu proceduru preporučljivo koristiti periodično u problemima u kojima se javljaju ekstremno loše uslovljene matrice.

7 Modifikovani afini algoritam

U ovom poglavlju predlažemo jednostavan afini algoritam koji se odlikuje velikom numeričkom stabilnošću. Ideja za modifikaciju potiče iz rada [61]. Implementacijom ovog algoritma uspešno su rešeni svi navedeni loše uslovljeni problemi.

7.1 Afini algoritam i implementacija

Korak 1. Generisati početnu tačku x^0.

Korak 2. Izračunati rezidume

$$r_b = Ax^0 - b, \qquad r_c = A^T\lambda^0 + s^0 - c$$

i proveriti kriterijum za kraj algoritma $\frac{|c^T x - b^T \lambda|}{1+|c^T x|} \leq \epsilon$. Ako je kriterijum za kraj ispunjen, izlazni podatak je x^{k+1}; u suprotnom, ići na *Korak 3.*

Korak 3. Formirati matrice S, X i vektor e.

Korak 4. Neka je $D = S^{-1/2}X^{1/2}$, i $r_{xs} = XSe$ i rešimo sledeći sistem

$$\begin{aligned} AD^2A^T\Delta\lambda^{aff} &= -r_b - A(S^{-1}Xr_c - S^{-1}r_{xs}), \\ \Delta s^{aff} &= -r_c - A^T\Delta\lambda^{aff}, \\ \Delta x^{aff} &= -S^{-1}(r_{xs} + X\Delta s^{aff}). \end{aligned}$$

Korak 5. Izračunati uslov za nenegativnost iteracione tačke

$$\begin{aligned} \alpha_{aff}^{pri} &= \max\{\alpha \in [0,1] \,:\, x^k + \alpha\Delta x^{aff} \geq 0\} \\ \alpha_{aff}^{dual} &= \max\{\alpha \in [0,1] \,:\, s^k + \alpha\Delta s^{aff} \geq 0\}. \end{aligned}$$

Korak 6. Staviti $(\Delta x^k, \Delta\lambda^k, \Delta s^k) = (\Delta x^{aff}, \Delta\lambda^{aff}, \Delta s^{aff})$.

Korak 7. Izračunati

$$\begin{aligned} \alpha_{max}^{pri} &= \max\{\alpha \geq 0 \,:\, x^k + \alpha\Delta x^k \geq 0\} \\ \alpha_{max}^{dual} &= \max\{\alpha \geq 0 \,:\, s^k + \alpha\Delta s^k \geq 0\}. \end{aligned}$$

Korak 8. Staviti

$$\begin{aligned} \alpha_k^{pri} &= \min\{0.99\alpha_{max}^{pri}, 1\} \\ \alpha_k^{dual} &= \min\{0.99\alpha_{max}^{dual}, 1\}. \end{aligned}$$

Korak 9. Izračunati sledeću iteraciju

$$\begin{aligned} x^{k+1} &= x^k + \alpha_k^{pri}\Delta x^k, \\ (\lambda^{k+1}, s^{k+1}) &= (\lambda^k, s^k) + \alpha_k^{dual}(\Delta\lambda^k, \Delta s^k). \end{aligned}$$

Zatim primeniti transformaciju

$$x = \overline{x}^{t+1} = x^t + \alpha_k(x^{t+1} - x^t),$$

gde je α_k realan broj koji zadovoljava $0 < \alpha_k < 1$ i preći na *Korak 2.*

U *Koraku 9* primenimo zamenu

```
dx=Dxaff;  dl=Dlaff;   ds=Dsaff;
```

Ostali detalji implementacije su analogni odgovarajućim koracima u Mehrotraovom algoritmu.

7.2 Numerički primeri

Primer 7.2.1 U ovom primeru dajemo vrednosti mere dualnosti xs koje verifikuju numeričku stabilnost i konvergenciju nekih test primera.

Afiro:

{2.09131031484025964$\times 10^{6}$, 717870.2, 178650.3, 3178.164, 802.9825,
365.1796, 120.9447, 41.31498, 26.45259, 3.55117, 0.0674026✓,
0.00327542, 0.000120382, .85961511355211195$\times 10^{-5}$,
.36281499330046743$\times 10^{-6}$,.23982647832721707$\times 10^{-7}$,
1.05699567815953665$\times 10^{-9}$}

Blend:

{14442.41,5493.112, 3476.63, 1815.434, 734.1495, 142.1366, 42.09287,
15.52328, 9.826819, 6.14928, 3.02126, 1.43341, 0.950946, 0.499918,
0.167127, 0.0408819, 0.00494338, 0.000283267✓,
.44466400330290623$\times 10^{-5}$, .7795079018220674$\times 10^{-7}$,
1.71453952334643489$\times 10^{-9}$}

Kb2:

{379788.1, 218215.2, 88425.23, 50764.44, 26809.45, 14529.56, 9044.567,
8458.448, 7488.039, 911.929, 756.411, 705.527, 635.302, 643.668, 565.415,
500.207, 489.106, 427.418, 422.982, 346.793, 294.856, 291.866, 275.545,
222.275, 173.444, 106.912✓, 36.7028, 35.6593, 31.7892, 24.4521,
16.8164, 9.32001, 4.03164, 2.21231, 0.91458, 0.473405, 0.291879,
0.121496, 0.0169711, 0.00196154, 0.000650803, 7.74735416706757629$\times 10^{-6}$}$_{41}$

Adlittle:

{7.3076710216085603$\times 10^{6}$, 4.18579049157574267$\times 10^{6}$, 1.52389518770815923$\times 10^{6}$,
464181.4, 235050.5, 92340.6, 44810.17, 22558.18, 7628.979, 4468.44, 2964.28,
1368.54✓, 896.732, 684.051, 499.263, 398.103, 229.3717, 69.32518,
9.08286, 0.847247, 0.0447238, 0.000599578, 0.0000153751,2.90569345853342007$\times 10^{-7}$}

Sc50a: ($\alpha = 0.98$:)

{97873.41, 6501.212, 847.0143, 64.14744, 48.21975, 40.72946, 36.13577,
22.13138, 9.188619, 3.50789, 1.87901, 1.11681, 0.643128, 0.0786037,
0.0024431, 0.0000392868, 7.83566461175516693$\times 10^{-7}$✓,
3.50005943189752377$\times 10^{-8}$, 5.42850445053744934$\times 10^{-10}$}$_{18}$

Primetimo da je broj iteracija veći od klasičnog algoritma. Međutim, numerička stabilnost algoritma je povećana.

8 Zaključak

U ovoj glavi su razmatrani teorijski i praktični problemi implementacije Mehrotraovog primal dual algoritma. Realizovana je implementacija u programskom paketu MATHEMATICA i izvršeno je poređenje sa već postojećim kodovima. Pokazano je da u odnosu na kod $PrimalDualLP$ iz [10] koji je takođe realizovan u programskom paketu MATHEMATICA, implementacija MPD u svakom pogledu ima znatno bolje karakteristike. U poređenju sa jačim programima (PCx, LIPSOL, LOQO, HOPDM) implementacija MPD je u prednosti kada je u pitanju veća preciznost, rešavanje nekih slabo uslovljenih primera i unošenje podataka u klasičnom obliku. Sa druge strane, kod MPD je sporiji od pomenutih programa i ograničen je po pitanju dimenzije

problema.Većina dostupnih test problema je data u MPS formatu čija je glavna karakteristika zapis matrice problema po kolonama. Takav format prevedimo u mnogo prirodniji i korisnicima dostupniji NB format koji koristi MATHEMATICA i u kome se problem zapisuje na klasičan način tako da ne zahteva nikakvo posebno predznanje od korisnika. Za taj cilj smo koristili programsko okruženje Delphi.

Kod MPD je realizovan sa idejom da se na njemu izvrše sledeća istraživanja:

- uticaj smanjenje dimenzije problema eliminacijom vrsta i kolona iz matrice sistema;
- poređenje normalnog i pridruženog sistema jednačina;
- stabilizacija numeričkog procesa primenom procedure iz [64].

Posle eliminaciji vrsta i kolona iz matrice problema i primene predložene heuristike u Mehrotraovom algoritmu imamo sledeće prednosti u odnosu na originalni algoritam:

- modifikovani metod daje bolju preciznost, modifikacija poboljšava stabilnost i centralnost iterativnog niza,
- modifikovani metod je brži u odnosu na originalni, iako sadrži dodatne eliminacije nekih vrsta i kolona i na kraju primenjuje rutinu za rekonstrukciju rešenja.
- modifikacija redukuje procesorsko vreme potrebno za rešavanje problema *Blend* i *Kb2*, uprkos dodatnog iterativnog koraka koji zahteva modifikacija a takođe modifikacija smanjuje dimenziju problema i zahtev za memorijskim prostorom.

Na primerima je pokazano da je proširen sistem u opštem slučaju brži u odnosu na normalni sistem. Kako je tačnost slična u oba pristupa, to primena proširenog sistema ima bolje karakteristike. Posebno, problemi koji sadrže slobodne promenljive se lakše rešavaju primenom proširenog sistema, dok primena normalnih jednačina zahteva nezgodnu reformulaciju problema.

Primrnom stabilizacione procedure razvijen je kôd koji uspešno rešava loše uslovljene probleme, koje ni poznati LP solveri nisu mogli da reše. Stabilizovani kôd je primenjen na nekim *Netlib* test problemima. Originalni i modifikovani Mehrotraov algoritam daju skoro identične rezultate za skoro sve probleme. Primetimo da naša implementacija Mehrotraov primal-dual algoritma ne može da reši neke probleme, ali sa modifikacijom ti problemi su rešeni. Sa druge strane, primetili smo mala numerička odstupanja kada se stabilizacija primeni na neke probleme. To je verovatno posledica grešaka koje indukuje Gaussova eliminacija. Zaključujemo da je primena stabilizacione procedure razumna samo na loše uslovljenim problemima.

Literatura

[1] E.D. Andersen and K.D. Andersen, *The MOSEK interior-point optimizer for linear programming: an implementation of the homogeneous algoritm*, In H. Frenk, K. Roos, T. Terlaky and S. Zang, editors, *High Perfomance Optimization*, Kluwer Academic Publishers, (2000), 197–232.

[2] E.D. Andersen, J. Gondzio, C. Mészáros and X. Xu *Implementation of interior point methods for large scale linear programming*, Technical report, HEC, Université de Genève, 1996.

[3] E.D. Andersen and Y. Ye *Combining interior-point and pivoting algorithms for linear programming*, Technical report, Department of Menagement Sciences, The University of Iowa, 1994.

[4] K.M. Anstreicher, *A monotonic projective algorithm for fractional linear programming*, Algorithmica **1**, (1985) 483–498.

[5] C. Ashcraft, R.L. Grimes and J.G. Lewis, *Accurate symmetric indefinite linear equation solvers*, SIAM Journal on Matrix Analysis, **20**, Issue 2 (1999), 513 – 561.

[6] M.D. Ašić and V.V. Kovačević-Vujčić, *Ill-conditionedness and interior-point methods*, Univ. Beograd Publ. Elektrotehn. Fak. **11**, (2000), 53–58.

[7] E.R. Barnes, *A variation on Karmarkar's algorithm for solving linear programming problems*, Mathematical Programming **36** (1986), 174–182.

[8] V. Baryamureeba and T. Steihaug, *On the convergence of an inexact primal-dual interior point method for linear programming*, Technical report No.188, Department of informatics, Univesity of Bergen (2000), 1–13.

[9] A. Ben-Tal, M. Teboulle, *A geometric property of the least square solution of linear equations*, Linear Algebra and its Applications **139**, (1990), 165–170.

[10] M.A. Bhatti *Practical optimization methods*, Springer, New York, 2000.

[11] R. Bixby, *Implementing the simplex method; the initial basis*, ORSA Journal on Computing **4** (1992), 267–284.

[12] K.H. Borgwardt, *The simplex method: a probablistic analysis*, Springer, Berlin, 1987.

[13] D. Cvetković, M. Čangalović, Dj. Dugošija, V. Vujčić-Kovačević, S. Simić i J. Vuleta, *Kombinatorna optimizacija*, Društvo Operacionih Istraživača Jugoslavije - DOPIS, Beograd, 1996.

[14] J. Czyzyk, S. Mehrotra and S.J. Wright, *PCx User Guide,* Optimization Thechnology Center, Technical Report 96/01, (1996) 1–21.

[15] G.B. Dantzig, *Lineare Programmierung und Erweiterungen*, Springer-Verlag Berlin Heidelberg New York, 1966.

[16] G. De Ghellinck and J. Ph. Vial, *A polynomial Newton method for linear programming*, Algoritmica **1** (1986), 425–453.

[17] D. Den Hertog, C. Roos and T. Terlaky, *A potential reduction variant of Renegar's short-step path following method for linear programming*, Linear Algebra and its Applications **152** (1991), 43–68.

[18] I.I. Dikin, *Iterative solution of problems of linear and quadratic programming*, Soviet Mathematics Doklady **8** (1967), 674–675.

[19] I.I. Dikin, *On the speed of an iterative process*, Upravlyaemye Sistemy **12**, (1974), 54–60.

[20] Juan Dominguez and María D. Gonzáles-Lima, *A primal-dual interior-point algorithm for quadratic programming*, Universidad Simón Bolívar,(CESMa) Apdo 89000, Caracas 1080-A, Technical Report 2005-01. Venezuela

[21] Juan Dominguez and María D. Gonzáles-Lima, *A primal-dual interior-point algorithm for quadratic programming*, Numer Algor 42 (2006), 1–30.

[22] Juan Javier Dominguez Moreno, *Un algoritmo de puntos interiores para programación cuadrática*, Master thesis, Universidad Simón Bolívar, 2004.

[23] I.S. Duff, *The solution of augmented systems*, in Numerical Analysis, D.F. Griffiths and G.A. Watson, eds., Longman Scientific and Technical, Essex,U.K. (1993), 40–45.

[24] A.S. El-Bakry, R.A.Tapia and Y. Zhang, *A study of indicators for identifying zero variables in interior-point methods*, SIAM Rev. **36(1)** (1994), 45–72.

[25] A. Forsgren and G. Sporre, *On weighted linear least-squeares problems related to interior methods for convex quadratic programming*, Technical report TRITA-MAT-2000-OS11, Department of Mathematics, Royal Institute of Technology, 2000.

[26] R. Fourer and S. Mehrotra, *Solving symmetric indefinite systems in an interior-point method for linear programming*, Mathematical Programming **62** (1993), 15–39.

[27] R.M. Freund, *A potential-function reduction algorithm for solving a linear program directly from an infeasible "warm start"*, Mathematical Programming **52** (1991), 441-446.

[28] R. Freund, *Polynomial-time algorithms for linear programmnig based only on primal scaling and projected gradients of a potential function*, Mathematical Programming **51** (1991), 203-222.

[29] R. Freund, *Theoretical efficiency of a shifted-barier-function algorithm for linear programmnig*, Linear Algebra and its Applications **152** (1991), 19-41.

[30] R.M. Freund, *A potential-function reduction algorithm with user-specified phase I-phase II balance for solving a linear program directly from an infeasible "warm start"*, SIAM J. Optimization Vol. **5**, No. 2 (1995), 247–268.

[31] R.M. Freund, *Following a "balanced" trajectory from an infeasible point to an optimal linear programming solution with polynomial-time algorithm*, Mathematics of Operations Research Vol. **21**, No. 4 (1996), 839–859.

[32] R.M. Freund, *An infeasible-start algorithm for linear programming whose complexity depends on the distance from the starting point to the optimal solution*, Annals of Operations Research **62** (1996), 29–57.

[33] R.W. Freund and F. Jare, *A QMR-based interior-point algorithm for solving linear programs*, Mathematical Programming **76** (1996), 183–210.

[34] R.W. Freund, F. Jare and S. Mizuno, *Convergence of a class of inexact interior-point algorithms for linear programs* Mathematics of Operational Research Programming **24** (1999), 50–71.

[35] K.R. Frish, *The logarithmic potential method of convex programming*, Technical report, University Institute of Economics, Oslo, Norway (1955), 183–210.

[36] A.J. Goldman and A.W. Tucker, *Theory of linear programming*, in Linear Equalities and related Systems, H.W. Kuhn and A.W. Tucker, eds., Princeton University Press, Princeton, N.J. (1956), 53–97.

[37] J. Gondzio and T. Terlaky, *A computational view of interior-point methods for linear programming*, Cahiers de recherche, Technical Report, HEC, Université de Genève 1994.22 (1994).

[38] J. Gondzio, *HOPDM (Version 2.12): A fast LP solver based on a primal-dual interior point method*, European Journal of Operations Research **85** (1995), 221–225.

[39] C.C. Gonzaga, *An algorithm for solving linear programming problems in $O(n^3L)$ operations*, in Progress in Mathematical Programming: Interior-Point and Related Methods, N. Megiddo, ed., Springer-Verlag, New York (1988), 1–28.

[40] C.C. Gonzaga and J.F. Bonnans, *Fast convergence of the simplified largest step path following algorithm*, Mathematical Programming **76** (1996), 95–115.

[41] C.C. Gonzaga, *Conical projection algorithms for linear programming*, Mathematical Programming **43** (1989), 151–173.

[42] C.C. Gonzaga, *Polynomial affine algorithms for linear programming*, Mathematical Programming **49** (1990), 7–21.

[43] C.C. Gonzaga, *Interior point algorithms for linear programming with inequality constraints*, Mathematical Programming **52** (1991), 209–225.

[44] C.C. Gonzaga, *Large-step path following method for linear programming, Part I: barrier function method*, SIAM Journal on Optimization **1** (1991), 268–279.

[45] C.C. Gonzaga, *Large-step path following method for linear programming, Part II: potential reduction method*, SIAM Journal on Optimization **1** (1991), 280–292.

[46] C.C. Gonzaga and M.J. Todd, *An $O(\sqrt{n}L)$-iteration large-step primal-dual affine algorithm for linear programming*, SIAM Journal on Optimization **2** (1992), 349–359.

[47] O. Güler and Y. Ye, *Convergence behavior of interior-point algorithms*, Mathematical Programming **60** (1993), 215–228.

[48] N.I.M. Gould, *Iterative methods for ill-conditioned linear systems from optimization*, Technical report, Rutheford Appelton Laboratory, Oxfordshire (1998), 1–18.

[49] A.J. Hoffman, M. Mannos, D. Sokolowsky and N Wiegman, *Computational experience in solving linear programs*, Journal of the Society for Industrial and Applied Mathematics **1** (1953),17–33.

[50] B. Jansen, C. Ross, T. Terlaky and Y. Ye, *Improved complexity using higher-order correctors for primal-dual Dikin affine scaling*, Mathematical Programming **76** (1996), 117–130.

[51] N. Karmarkar, *A new polynomial-time algorithm for linear programming*, Combinatorica **4** (1984), 373–395.

[52] N.K. Karmarkar and K.G. Ramakrishnan, *Computational results of an interior point algorithm for large scale linear programming*, Mathematical Programming **52** (1991), 555–586.

[53] L.G. Khachiyan, *A polynomial algorithm in linear programming*, Soviet Mathematics Doklady **20** (1979), 191–194.

[54] V. Klee and G.L. Minty, *How good is the simplex method*, In: O. Shisha, ed., Inequalities III, Academic Press, New York (1972), 159–175.

[55] M. Kojima, S. Mizuno and A. Yoshise, *A polynomial-time algorithm for a class of linear complemntarity problems*, Mathematical Programming **44** (1989), 1–26.

[56] M. Kojima, S. Mizuno and A. Yoshise, *A primal-dual interior point algorithm for linear programming*, in Progress in Mathematical Programming: Interior-Point and Related Methods, N. Megiddo, ed., Springer-Verlag, New York **ch. 2** (1989), 29–47.

[57] M. Kojima, *Basic lemmas in polynomial-time infeasible-interior-point methods for linear programs*, Annals of Operations Research **62** (1996), 1–28.

[58] M. Kojima, *A primal-dual infeasible-interior-point algorithm for linear programming*, Mathematical Programming, Series A **61** (1993), 261–280.

[59] M. Kojima, S. Mizuno and A. Yoshise, *A polynomial-time algorithm for a class of linear complementarity problems*, Mathematical Programming **44** (1989), 1–26.

[60] M. Kojima, S. Mizuno and A. Yoshise, *An $O(\sqrt{n}L)$ iteration potential redusvtion algorithm for linear complementarity problems*, Mathematical Programming **50** (1991), 331–342.

[61] V.V. Kovačević-Vujčić, *Improving the rate of convergence of interior-point methods for linear programming*, Mathematical Programming **52** (1991), 467–479.

[62] V.V. Kovačević-Vujčić, *Stabilization of path-following interior-point methods for linear programming*, IX Conference on Applied Mathematics, D. Herceg, Lj. Cvetković, eds. Institute of Mathematics, Novi Sad (1995), 47–53.

[63] V.V. Kovačević-Vujčić, *New approaches to linear programming*, SYMOPIS, Beograd, November 4-6 (1993), 5–15.

[64] V.V. Kovačević-Vujčić and M.D. Ašić, *Stabilization of interior-point methods for linear programming*, Computational Optimization and Applications **14** (1999), 331–346.

[65] I.J. Lustig, R.E. Marsten and D.F. Shanno, *Computationial experience with a primal-dual interior point method for linear programming*, Linear Algebra and Its Applications **152** (1991), 191–222.

[66] N. Megiddo, *Pathways to the optimal set in linear programming*, in Progress in Mathematical Programming: Interior-Point and Related Methods, N. Megiddo, ed., Springer-Verlag, New York **ch. 8** (1989), 131–158.

[67] N. Megiddo, *On finding primal- and dual-optimal bases*, ORSA Journal on Computing, **3**(1991), 63–65.

[68] S. Mehrotra and Y. Ye, *Finding an interior point in the optimal face of linear programs*, Mathematical Programming **62** (1993), 497–515.

[69] S. Mehrotra, *On finding a vertex solution using interior point methods*, Linear Algebra and Its Applications **152** (1991), 233–253.

[70] S. Mehrotra, *On the implementation of a primal-dual interior point method* SIAM J. on Optimization **2** (1992), 575–601.

[71] S. Mehrotra and Y. Ye, *Finding an interior point in the optimal face of linear programs*, Mathematical Programming **62** (1993), 497–515.

[72] S. Mizuno, M. Todd and Y. Ye, *On adaptive step primal-dual interior-point algorithms for linear programming* Mathematics of Operations Research **18** (1993), 964–981.

[73] R.D.C. Monteiro and I. Adler *Interior path-following primal-dual algorithms. Part*I*: Linear Programming* Mathematical Programming **44** (1989), 27–41.

[74] R.D.C. Monteiro, I. Adler and M.G.C. Resende, *A polynomial-time primal-dual affine scaling algorithm for linear convex quadratic programming and its power series extension* Mathematics of Operations Research **15** (1990), 191–214.

[75] C.L. Monma and J. Morton, *Computational experience with a dual affine variant of Karmarkar's method for linear programming*, Operations Research Letters **6** (1987), 261–267.

[76] Y. Nesterov, *Long-step strategies in interior-point primal-dual methods*, Mathematical Programming **76** (1996), 47–94.

[77] J. von Neumann, *On a maximization problem*, Technical report, Institute for Advanced Study (Princeton, NJ, USA), (1947).

[78] E. Ng and B.W. Peyton, *A supernodal Cholesky factorization algorithm for shared-memory multiprocessors*, SIAM J. SCI.COMPUT.,Vol. **14**, No. 4, (1993), 761–769.

[79] F.A. Potra, *A quadratically convergent predictor-corrector method for solving linear programs from infeasible starting points*, Mathematical Programming **67** (1994), 383–406.

[80] J. Renegar, *A polynomial-time algorithm, based on Newton's method, for linear programming*, Mathematical Programming **40** (1988), 59–93.

[81] J. Renegar and M. Shub, *Unified complexity analysis for Newton LP metods*, Mathematical Programming **53** (1992), 1–16.

[82] C. Roos and J.Ph. Vial, *Long steps with the logarithmic penalty barrier function in linear programming*, in: J. Gabszewich, J.F. Richard and L. Wolsey,eds, Economic Decision-making: Games, Economics and Optimization. Contributions in Honour of Jacques H. Dreze (Elsevier Science Publisher, Amsterdam, 1990), 433–441.

[83] C. Roos and J.Ph. Vial, *A polynomial method of approximate centers for linear programming*, Mathematical Programming **54** (1992), 295-306.

[84] E. Rothberg, A. Gupta, E. Ng and B.W. Peyton, *Parallel sparse Cholesky factorization algorithms for shared-memory multiprocessors system*, Proceedings of the 7th IMACS International Conf. on Computer Methods for Partial Differential Equations

[85] G. Sonnevend, J. Stoer and G. Zhao, *On the complexity of following the central path of linear programs by linear extrapolation*, Methods of OPerations Research, **62** (1989), 19–31.

[86] G. Sonnevend, J. Stoer and G. Zhao, *On the complexity of following the central path of linear programs by linear extrapolation II*, Mathematical Programming, **52** (1991), 527–553.

[87] P. Stanimirović, N. Stojković, B. Momcilovic and Z. Jovanovic,*Augmented and Normal Equations System in Mehrotra's Primal-dual Algorithm*, Filomat **15**, Niš (2001), 285–292.

[88] P. Stanimirović, N. Stojković and Vera Kovačević Vujčić, *Stabilization of Mehrotra's primal-dual algorithm and its implementation*, SYMOPIS, Beograd (2001), 343–346.

[89] P. Stanimirović, N. Stojković and Vera V. Kovačević-Vujčić, *Stabilization of Mehrotra's primal-dual algorithm and its implementation,* European Journal of Operational Research, Volume bf 165, Issue 3, (2005), 598-609.

[90] P. Stanimirović, N. Stojković and Vera V. Kovačević-Vujčić, *Some implementation details of modified Mehrotra's primal-dual algorithm*, XVI Conference on Applied Mathematics, N. Krejic, Z. Luzanin, eds. Department of Mathematics and Informatics, Novi Sad, (2004),149–156.

[91] N. Stojković and P. Stanimirović, *About the starting point in primal-dual interior point method*, SYMOPIS, Beograd (2000), 261–264.

[92] N. Stojković and P. Stanimirović, *Initial point in primal-dual interior point method*, Facta Universitatis, Ser. Mech. Vol.3, No 11, (2001), 219–222.

[93] N. Stojković, *On the finite termination in the primal-dual algorithm for linear programing*, YUJOR **11** (2001), 31–40.

[94] N. Stojković and P. Stanimirović , *Two direct methods in linear programming*, European Journal of Operational Research **131**, (2001), 417–439.

[95] N. Stojković and P. Stanimirović, *Transformations of Dual Problem and Decreasing Dimensions in Linear Programming*, Matematičiki Vesnik **54**, (2002), 203–210.

[96] N. Stojković and P. Stanimirović, *Decreasing Dimensions in Mehrotra's Primal-Dual Algorithm*, Zbornik radova SYM-OP-IS 2003,(2003), 343–346.

[97] N. Stojković and P. Stanimirović, *Finite Termination and Decreasing Dimensions in Mehrotra's Primal-Dual Algorithm*, The Sixth International Symposium on Nonlinear Mechanics - Nonlinear Science and Applications - Niš 2003. 24 - 29. August, 2003

[98] N.V. Stojković and P.S. Stanimirović, *Inverse and characteristic polynomial of extended triangular matrix*, Publ. Elect. Fac. Beogr. **11** (2000), 71–78.

[99] N. V. Stojković, Predrag S. Stanimirović and Marko D. Petković, *Several Modifications of Simplex Method*, Filomat 17,(2003), 169–176.

[100] N.V. Stojković, *Primal-dual i simpleks metodi za rešavanje problema linearnog programiranja*, Doktorska disertacija, PMF Niš, 2001.

[101] G.W. Stewart, *On scaled projections and pseudoinverses*, Linear Algebra and its Applications **112**, (1989), 189–193.

[102] K. Tanabe, *Centered Newton method for mathematical programming*, in System Modeling and Optimization: Proceedings of the 13th IFIP conference, Lecture Notes in Control and Information Systems 113, Berlin, August/September 1987, Springer-Verlag, New York, (1988), 197–206.

[103] M.J. Todd, *Potential-reduction methods in linear programming*, Mathematical Programming **76** (1996), 3–45.

[104] M.J. Todd and Y.Ye, *A centered projective algorithm for linear programming*, Mathematics of Operations Research **15** (1990), 508–529.

[105] R.J. Vanderbei, *LOQO: an interior point code for quadratic programming*, Statistics and Operations Research, Princeton University, Technical Report SOR-94-15, 1998.

[106] R.J. Vanderbei, M.S. Meketon and B.A. Freedman, *A modification of Karmarkar's linear programming algorithm*, Algorithmica **1** (1986), 395-407.

[107] P.J. Williams, *Effective finite termination procedures in interior-point methods for linear programming*, Doctoral thesis, Houston, Texas, 1998.

[108] S. Wolfram, *Mathematica: a system for doing mathematics by computer*, Addison-Wesley Publishing Co, Redwood City, California, 1991.

[109] S. Wolfram, *Mathematica Book, Version 3.0*, Wolfram Media and Cambridge University Press, 1996.

[110] Wolfram Research, *Mathematica 4.0 standard add-on packages*, Wolfram Media, (1999).

[111] S. Wolfram, *Mathematica Book, 4th edition*, Wolfram Media/Cambridge University Press, (1999).

[112] S.J Wright, *Primal-dual interior-point methods*, SIAM, Philadelphia, 1997.

[113] S.J. Wright, *A path-following interior-point algorithm for linear and quadratic problems*, Annals of Operations Research **62** (1996), 103–130.

[114] S.J. Wright, *An infeasible-interior-point algorithm for linear complementarity problems*, Mathematical Programming, **67** (1994), 29–52.

[115] S.J Wright, *Modified Cholesky factorizations in interior-point algorithms for linear programming*, SIAM Journal on Optimization **9**, (1999), 1152–1191.

[116] H. Yamashita, *A polynomially and quadratically convergent method for linear programming*, Mathematical System Institute (Tokyo, Japan, 1986)

[117] Y. Ye, *On the finite convergence of interior -point algorithms for linear programming*, Mathematical Programming, **57** (1992), 325–336.

[118] Y. Ye, *On homogeneous and self-dual algorithms for LCP*, Mathematical Programming **76** (1996), 211–221.

[119] Y. Ye, *An $O(n^3L)$ potential reduction algorithm for linear programming*, Mathematical Programming **50** (1991), 239–258.

[120] Y. Ye, *On quadratic and $O(\sqrt{n}L)$ convergence of a predictor-corrector algorithm for LCP*, Mathematical Programming **62** (1993), 537–551.

[121] Y. Ye, *A class of projective transformations for linear programming*, SIAM Journal on Computing (1991), 537–551.

[122] Y. Zhang, *On the convergence of a class of infeasible-interior-point method for the horizontal linear complementarity problem*, SIAM J. on Optimization **4** (1994), 208–227.

[123] Y. Zhang, *Solving large-scale linear programs by interior-point methods under the MATLAB environment*, Optimization Methods and Software **10** (1998), 1–31.

Glava 3

Primena linearnog programiranja

1 Uvod

U okviru ove glave razmatraće se formulacija nekih optimizacionih zadataka koji imaju poseban praktični značaj, a svode se na primenu linearnog programiranja. Međutim, u praksi se najčešće postavlja pitanje: kako formulisati matematičke modele i pripremiti konkretne podatke iz raznih oblasti primene linearnog programiranja, tako da bi se mogli upotrebiti računari i gotove rutine (paketi programa) koje su razvijene specijalno za rešavanje ovih zadataka?

Od odgovora na ovo pitanje zavisi konkretizacija svega onoga što je poznato u primenjenoj matematici, a odnosi se na najrazličitije postupke rešavanja zadataka linearnog programiranja. Zbog toga će se dati formulacija nekih zadataka kod kojih primena linearnog programiranja igra značajnu ulogu sa stanovišta pripremanja i donošenja optimalnih upravljačkih odluka.

Matematički model određenog zadatka koji se svodi na linearno programiranje nikada ne može da obuhvati sve prirodne pojave i da bude verna slika stanja koje se sa njim želi prikazati, jer najčešće obuhvata samo važnije pojave. Zbog toga najkvalifikovaniji i najodgovorniji deo posla leži na onima koji formulišu zadatke. Od njih zavisi izbor karakterističnih pojava najvažnijih za dati problem, koje se kao takve uključuju u odgovarajući matematički model.

Kod formiranja upravljačkih zadataka posebnu pažnju trebalo bi posvetiti na izbor pokazatelja kvaliteta rešenja odgovarajućeg zadatka. Po pravilu, karakteristika koja se optimizira (funkcija cilja) predstavlja pokazatelj kvaliteta rešenja. Dopunske granice karakteristika rešenja određuju se konstrukcijom dodatnih ograničenja.

Sa gledišta izbegavanja većih teškoća u iznalaženju optimalnog plana, poželjno je izvršiti izbor upravljačkih promenljivih, tako da skup ograničenja bude što jednostavniji. To će se postići ako fizički i prirodni uslovi zadatka drugačije ne zahtevaju.

Postavka i rešavanje zadataka koji se svode na LP ne izvode se uvek samo u jednom potezu. To je najčešći slučaj sa zadacima kod kojih odgovarajući matematički model nije dobro formulisan, pa ga treba poboljšati i bolje približiti realnim uslovima. Zato se rešavanju zadataka koji se svode na linearno programiranje nikada ne sme pristupiti jednostrano. Na njima, kao i na drugim zadacima optimizacije, mora raditi tim specijalista koji dobro poznaje prirodu problema i matematičke metode iznalaženja odgovarajućeg optimalnog rešenja.

U prvom poglavlju su opisane neke primene linearnog programiranje koje su zasnovane na transportnom modelu, kao što su transportni zadatak na putnoj mreži, optimizacija železničkog transporta, minimizacija vremena transporta.

Naredno poglavlje izučava neke klasične primene linearnog programiranja, koje su često zastupljene u literaturi. Najpoznatije primene su: optimalni program proizvodnje, optimizacija utroška materijala, izbor sastava mešavine, problem ishrane, neke primene u poljoprivredi, proizvodnji i ishrani. Ove primene su opisane u knigama [5, 6, 7]. Za svaki matematički model opisan u ovom poglavlju data je njegova implementacija u programskom paketu MATHEMATICA.

Poslednja dva poglavlja opisuju neke nestandardne primene linearnog programiranja, koje su retko zastupljene u literaturi. Prva od takvih primena se odnosi na projektovanje teleskopa (problem optike), druga na projektovanje digitalnih FIR filtara koji imaju veliku ulogu u obradi digitalnih signala (problem telekomunikacija), a treća na linearnu regresiju. Modeli L_1 i L_∞ su implemenirani u programskom paketu MATHEMATICA.

2 Primene zasnovane na transportnom modelu

2.1 Transportni zadatak u putnoj mreži

Kod rešavanja određenih transportnih problema, često se za njih vezuje konkretna putna (transportna) mreža u okviru koje se definišu svi punktovi $P_1, \ldots, P_N$ od kojih su neki međusobno povezani komunikacijama K_{ij}. Postojanje komunikacije K_{ij} znači da se iz punkta P_i u punkt P_j može vršiti transport, ali ne i obrnuto, što znači da pojam komunikacije ima značenje jednosmernog kretanja. Dvosmerno kretanje između punktova P_i i P_j podrazumeva postojanje dve komunikacije K_{ij} i K_{ji}. Svaki punkt putne mreže može se okarakterisati brojem p_i $(i = 1, \ldots, N)$, koji ima značenje, na primer, za organizaciju transporta proizvedene ili uskladištene robe, obima proizvodnje ili obima uskladištene robe u punktu P_i. U zavisnosti od znaka veličine p_i, svi punktovi P_i $(i = 1, \ldots, N)$ dele se na punktove proizvodnje (skladištenja) i na punktove potrošnje. Za punktove proizvodnje uzima se da su vrednosti p_i pozitivne, a za punktove potrošnje da su negativne, dok za punktove koji su samo tranzitne stanice uzima se da $p_i = 0$. Svakoj komunikaciji K_{ij} pridružuju se dva karaktcristična broja:

d_{ij} - maksimalna propusna sposobnost komunikacije K_{ij} i

c_{ij} - cena prevoza jedinice mere proizvoda koji se transportuje iz punkta P_i u punkt P_j korišćenjem komunikacije K_{ij}.

Čak i u slučaju kada ne postoji komunikacija K_{ij} između punktova P_i i P_j, može se podrazumevati njeno postojanje, ali pod uslovom da se uzme u obzir da je njena propusna moć $d_{ij} = 0$.

Zadatak se sastoji u sledećem: potrebno je odrediti plan transporta, tj. skup vrednosti x_{ij} koje zadovoljavaju ograničenja:

$$\begin{aligned} \sum_{j=1}^{N} x_{ij} - \sum_{j=1}^{N} x_{ji} &= p_i, \ (i = 1, \ldots, N) \\ 0 \leq x_{ij} &\leq d_{ij}, \end{aligned} \tag{2.1.1}$$

za sve komunikacije, uzimajući u obzir i one koje ne postoje, za koje je $d_{ij} = 0$, a da pri tome funkcija cilja

$$F = \sum_{i=1}^{N} \sum_{j=1}^{N} c_{ij} x_{ij} \tag{2.1.2}$$

dobije minimalnu vrednost. Lako se može zapaziti da količina x_{ij} koju treba transportovati po bilo kojoj komunikaciji putne mreže K_{ij} ulazi u levu stranu izraza (2.1.1) dva puta: jedanput kao količina koju treba transportovati iz punkta P_i sa znakom plus, a drugi put kao količina koju treba transportovati iz punkta P_j sa znakom minus. Zbog toga je potrebno zadovoljenje ravnoteže proizvodnje (skladištenja) i potrošnje, a posledica toga je da imamo

$$\sum_{i=1}^{N} p_i = 0. \tag{2.1.3}$$

Prema tome, za egzistenciju transportnog plana neophodno je, pored uslova (2.1.1), zadovoljenje i uslova (2.1.3). Tako se i ovaj zadatak dovodi u neposrednu vezu sa transportnim zadatkom.

2.2 Optimizacija železničkog transporta

Ekonomično iskorišćenje vagona na železnici predstavlja izuzetno aktuelan i značajan zadatak, koji se sastoji u raspodeli vagona sa ciljem udovođenja određenim zahtevima za transport robe. Kada je reč o raspodeli vagona ne misli se samo na jedan tip, već na različite tipove vagona (zatvoreni, poluvagoni, platforme, itd., sa različitim brojem osovina).

Označimo sa:

x_{ij} - broj vagona j-tog tipa napunjenih i-tom robom;

a_{ij} - normu opterećenja jednog vagona j-tog tipa i-tom robom;

a_i - količinu robe koju treba transportovati u tonama;

b_j - raspoloživi broj vagona j-tog tipa;

c_{ij} - eksploatacione rashode koji nastaju usled transporta i-te robe na jednom vagonu j-tog tipa.

Zadatak optimalnog rasporeda vagona sastoji se u izračunavanju nepoznatih ,x_{ij} koje minimiziraju funkciju cilja

$$F = \sum_{j=1}^{n} \sum_{i=1}^{m} c_{ij} x_{ij},$$

pri ograničenjima

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_{ij} &= a_i, \ (i = 1, \ldots, m), \\ \sum_{i=1}^{n} x_{ij} &\leq b_j, \ (j = 1, \ldots, n), \\ x_{ij} &\leq 0, \ (i = 1, \ldots, n, j = 1, \ldots, n). \end{aligned}$$

Funkcija cilja F određuje ukupne troškove transporta. Prvi skup ograničenja ukazuje da sva roba mora biti transportovana, a drugi skup da je broj vagona pojedinih tipova ograničen. Na ovaj način mogu se rešavati slični problemi rečnog, morskog i drugih vrsta transporta kako robe, tako i putnika.

2.3 Minimizacija vremena transporta

U nizu transportnih zadataka kako vojnih, tako i civilnih, kvalitet organizacije transporta meri se utrošenim vremenom na njegovo sprovođenje (izlazak na vatreni položaj, kampanja pšenice, kukuruza, voća, povrća i drugih proizvoda).

Označimo sa t_{ij} vreme utrošeno na transport proizvoda iz i-tog punkta proizvodnje u j-ti punkt potrošnje. Neka su punktovi proizvodnje označeni sa A_i u kojima su proizvedene količine a_i $(i = 1, \ldots, m)$, a punktovi potrošnje sa B_j u kojima postoje potrebe za količinama b_j $(j = 1, \ldots, n)$, izraženim u istim jedinicama mere kao i vrednosti a_i.

Zadatak optimizacije se sastoji u određivanju plana transporta $\|x_{ij}\|$, tj. skupa vrednosti x_{ij} $(i = 1, 2, ..., m;\ j = 1, 2, ..., n)$ za koje će vreme $t(X)$ najdužeg trajanja prevoza

$$\begin{aligned} t(X) &= \max\ t_{ij}, \\ x_{ij} &> 0. \end{aligned} \tag{2.3.1}$$

postati minimalno pri skupu ograničenja

$$\begin{aligned} \sum_{j=1}^{n} x_{ij} &\leq a_i, \ (i = 1, \ldots, m), \\ \sum_{j=1}^{n} x_{ij} &= b_j, \ (j = 1, \ldots, n), \\ x_{ij} &\geq 0, \ \ (i = 1, \ldots, m, \ j = 1, \ldots, n). \end{aligned}$$

Funkcija cilja (2.3.1) koja se minimizira predstavlja najduže vreme iz skupa vremena t_{ij}- trajanje transporta iz bilo koga punkta A_i u punkt B_j gde su predviđene količine transporta ($x_{ij} > 0$). Ovako formulisani zadatak ne spada u okvire linearnog programiranja, obzirom da je funkcija cilja $t(X)$ nelinearna funkcija opromenljivih x_{ij}. Zadaci ove vrste rešavaju se svođenjem na sukcesivno rešavanje serije običnih transportnih zadataka, ili na rešavanje o maksimalnom protoku gde se koriste poznati, ali veoma složeni algoritmi Hitchcock-a, Forda, Fulkersona.

3 Klasična primena linearnog programiranja

3.1 Optimalni program proizvodnje

Sastaviti optimalni program proizvodnje znači izabrati, između velikog broja proizvoda, onaj asortiman proizvoda koji će obezbediti maksimalno ekonomske efekte iz ograničene količine proizvodnih resursa.

Pretpostavimo da preduzeće može proizvoditi n različitih tipova proizvoda: P_1, P_2, ..., P_n. Za proizvodnju preduzeće koristi m različitih vrsta mašina, r različitih kategorija radnika i g različitih vrsta sirovina u ograničenim količinama. Poznat je dohodak koji se ostvaruje po jedinici svakog proizvoda, pa je problem kako preduzeće da programira proizvodnju da bi ostvarilo najveći mogući dohodak u poslovanju.

Matematički model ćemo formulisati korišćenjem sledećih simbola:

x_j - količina j-tog proizvoda koju treba proizvesti prema optimalnom programu proizvodnje, a koju treba odrediti pomoću matematičkog modela;

c_j - dohodak po jedinici j-tog proizvoda;

a_{ij} - vreme koje je potrebno da se na i-toj mašini proizvede jedinica j-tog proizvoda;

a_{i0} - kapacitet i-te mašine izražen u vremenskim jedinicama;

b_{kj} - vreme potrebno radniku k-te kategorije da obradi, odnosno proizvede, jedinicu j-tog proizvoda;

b_{k0} - raspoloživi fond radnog vremena radnika k-te kategorije;

s_{vj} - količina v-te sirovine koja je potrebna za proizvodnju jedinice j-tog proizvoda;

s_{v0} - raspoloživa količina v-te vrste sirovine;

e_j - količina j-tog proizvoda koja se može prodati na trzištu.

Pomoću uvedenih simbola formulišemo matematički model. On ima funkciju kriterijuma

$$z_0 = \sum_{j=1}^{n} c_j x_j,$$

koja označava dohodak od celokupne proizvodnje, pa treba naći njenu maksimalnu vrednost uz sledeće ograničavajuće faktore:

$$\begin{aligned}
\sum_{j=1}^{n} a_{ij}x_j &\le a_{i0}\ ,\ i = 1, 2, \ldots, m,\\
\sum_{j=1}^{n} b_{kj}x_j &\le b_{k0}\ ,\ k = 1, 2, \ldots, r,\\
\sum_{j=1}^{n} s_{vj}x_j &\le s_{v0}\ ,\ v = 1, 2, \ldots, g,\\
0 \le x_j &\le e_j\ ,\ j = 1, 2, \ldots, n.
\end{aligned} \tag{3.1.1}$$

```
Dodaj[a_,b_]:=
  Module[{m=a,i,n=Length[b]},
      For[i=1,i<=n,i++,m=AppendTo[m,b[[i]] ]  ];
      Return[m]
  ]

OptimalniProgram[c_,a_,a0_,b_,b0_,s_,s0_,e_]:=
  Module[{m=-a,v=-a0,n=Length[c],i},
     m=Dodaj[m,-b];  m=Dodaj[m,-s]; m=Dodaj[m,-IdentityMatrix[n]];
     v=Dodaj[v,-b0]; v=Dodaj[v,-s0];
     For[i=1,i<=n,i++,v=AppendTo[v,-e[[i]]]];
     LinearProgramming[-c,m,v]
  ]
```

Primer 3.1.1 *U ovom primeru uočimo neke konkretne vrednosti za veličine koje su neophodne u matematičkom modelu* (3.1.1).

```
In[1]:= c = {2,6,2,3,5};
        a = {{1,3,2,1,4}, {3,1,2,4,5}, {3,4,5,1,2}};  a0 = {20,23,19};
        b = {{3,2,2,4,1}, {1,1,2,4,1}}; b0 = {12,14};
        s = {{3,3,1,2,3}, {2,4,4,1,5}}; s0 = {221,342};
        e = {23,33,56,47,45};

In[2]:=N[OptimalniProgram[c, a, a0, b, b0, s, s0, e],6]

Out[1]= {0, 3.45714, 0, 0.714286, 2.22857}
```

3.2 Optimizacija utroška materijala

Optimalni utrošak materijala je takav utrošak materijala koji obezbeđuje ostvarenje određene proizvodnje uz najmanji ukupni otpadak. Kod formiranja odgovarajućeg matematičkog modela polazi se od sledećih pretpostavki: preduzeće bi trebalo da proizvede m različitih delova u određenim količinama, za proizvodnju tih delova koristi se materijal istih dimenzija.

Broj varijanti za obradu materijala unapred je poznat, a isto tako poznata je količina otpadaka koja se javlja pri svakoj varijanti, kao i količina pojedinih delova koja se dobija obradom jedinice materijala po svakoj varijanti.

U matematičkom modelu se koriste sledeće oznake:

x_j - količina materijala (broj traka, tabli) koja će biti obrađena po j-toj varijanti, a koje se određuju kao nepoznate vrednosti;

a_j - količina otpadaka od jedinice datog materijala koji je obrađen po j-toj varijanti;

a_{ij} - količina delova i-te vrste koja se dobija od jedne jedinice materijala obrađenog po j-toj varijanti;

b_i - ukupna količina i-tog dela koju treba obezbediti za proizvodnju;

s - raspoloživa količina datog materijala.

Modelom će se odrediti količina materijala date dimenzije koja će biti obrađena po j-toj varijanti, ali tako da se željene količine delova proizvedu uz minimalni ukupni otpadak.

Funkcija kriterijuma modela

$$z_0 = \sum_{j=1}^{n} a_j x_j$$

označava ukupni otpadak pri obradi materijala po svim varijantama, pa je potrebno naći njenu minimalnu vrednost uz sledeći sistem ograničenja:

$$\begin{aligned} &\sum_{j=1}^{n} a_{ij} x_j = b_i \ , \ i = 1, 2, \ldots, m, \\ &\sum_{j=1}^{n} x_j \leq s, \\ &x_j \geq 0 \ , \ j = 1, 2, \ldots, n. \end{aligned}$$

Formirali smo model za slučaj kada preduzeće za proizvodnju koristi materijal istih dimenzija.

```
UtrosakMaterijala[a0_,a_,b_,s_]:=
   Module[{m=a,v=b,n=Length[a0],s0={},i},
      For[i=1,i<=n,i++,
          s0=AppendTo[s0,-1]
      ];
      m=Dodaj[m,{s0}];    v=Dodaj[v,{-s}];
      LinearProgramming[a0,m,v]
   ]
```

Primer 3.2.1 Neka su date sledeće ulazne veličine:

```
In[1]:=a0 = {3,7,9,6,5}; a = {{3,7,1,2,4}, {2,2,4,1,5}}; b = {35,40}; s = 200;
In[2]:=N[UtrosakMaterijala[a0, a, b, s],6]
Out[1]= {2.14286, 0, 0, 0, 7.14286}
```

Pretpostavićemo da preduzeće za proizvodnju koristi isti materijal u k raznih dimenzija. Svaka dimenzija materijala može se obraditi prema raznim varijantama.

Ako sa n_v označimo broj varijanti obrade v-tog materijala, onda će ukupan broj varijanti obrade materijala svih dimenzija biti jednak N, pri čemu je

$$N = \sum_{v=1}^{k} n_v.$$

Sa oznakama i parametrima, čije je značenje isto kao i u prethodnom modelu, formiramo sledeći model.

Potrebno je naći minimalnu vrednost funkcije kriterijuma

$$z_0 = \sum_{j=1}^{N} a_j x_j$$

uz zadovoljenje sledećeg sistema ograničenja:

$$\sum_{j=1}^{N} a_{ij} x_j = b_i \ , \ i = 1, 2, \ldots, m,$$
$$\sum_{j=1}^{n_v} x_j \leq s_v \ , \ v = 1, 2, \ldots, k,$$
$$x_j \geq 0 \ , \ j = 1, 2, \ldots, N.$$

```
UtrosakMaterijalaRazni[a0_,a_,b_,s_,n_]:=
   Module[{m=a,v=b,vn,k=Length[n],i,nula,j,l=Length[a0]},
       vn=Sum[n[[i]],{i,k}];   nula=Table[0,{k},{l}];
       For[i=1,i<=k,i++,
         For[j=1,j<=n[[i]],j++,nula[[i,j]]=-1]
       ];
       m=Dodaj[m,nula];     v=Dodaj[v,-s];
       LinearProgramming[a0,m,v]
   ]
```

3.3 Izbor sastava mešavine

Problem svodi na određivanje količina pojedinih sirovina koje će biti utrošene za proizvodnju gotovog proizvoda odgovarajućeg kvaliteta, ali tako da troškovi nabavke sirovina budu minimalni.

U prvom slučaju, u kome se od više sirovina prozvodi jedan proizvod, polazimo od sledćih pretpostavki: za dobijanje gotovog proizvoda može se koristiti n vrsta sirovina; gotov proizvod mora da sadrži m raznih elemenata u određenim količinama; treba proizvesti ukupno b jedinica gotovog proizvoda; poznate su nabavne cene pojedinih sirovina.

Matematički model ćemo formirati korišćenjem sledećih simbola:

x_j - količina j-te sirovine koja će se utrošiti za proizvodnju b jedinica gotovog proizvoda;

p_j - nabavna cena jedinice j-te sirovine;

a_{ij} - količina i-tog elementa u jedinici j-te sirovine;

a_{i0} - propisana (minimalna ili maksimalna) količina i-tog elementa u gotovom proizvodu;

b - količina gotovog proizvoda koju treba proizvesti;

s_j - dozvoljena količina j-te sirovine u gotovom proizvodu.

Model se sastoji od funkcije kriterijuma

$$\min \quad z_0 = \sum_{j=1}^{n} p_j x_j$$

i ograničavajućih faktora:

$$\sum_{j=1}^{n} a_{ij} x_j = a_{i0} \ , \ i = 1, 2, \ldots, m,$$
$$\sum_{j=1}^{n} x_j = b, \quad 0 \leq x_j \leq s_j \ , \ j = 1, 2, \ldots, n.$$

```
Mesavina[p_,a_,a0_,b_,s_]:=
  Module[{m=a,v=a0,n=Length[p],pom,i},
     pom={};
     For[i=1,i<=n,i++,pom=AppendTo[pom,1]];
     m=Dodaj[m,{pom}];     m=Dodaj[m,-IdentityMatrix[n]];
     v=Dodaj[v,{b}];    v=Dodaj[v,-s];
     LinearProgramming[p,m,v]
  ]

  MesavinaProcenti[p_,a_,a0_,s_]:=Mesavina[p,a,a0,1,s]
```

3.4 Primena u poljoprivredi

Pretpostavimo da poljoprivredno dobro raspolaže sa h hektara obradive površine, da na njoj može zasejati n vrsta poljoprivrednih kultura, da raspolaže sa m vrsta poljoprivrednih mašina i da se za proizvodnju koristi g vrsta semena, zaštitnih sredstava i đubriva. Pored toga, poznati su prosečni prinosi svih poljoprivrednih kultura po jednom hektaru, te prodajna cena po jedinici svake kulture i direktni troškovi obrade jednog hektara pod određenom kulturom.

Potrebno je da se odredi na kojoj površini je potrebno zasejati svaku od poljoprivrednih kultura da bi poljoprivredno dobro ostvarilo maksimalan čist prihod.

Matematički model ćemo formirati korišćenjem sledećih simbola:

x_j - broj hektara na kojima će biti zasejana j-ta poljoprivredna kultura;

q_j - prosečni prinos j-te kulture po jednom hektaru;

p_j - prodajna cena po jedinici j-te kulture;

t_j - direktni troškovi obrade jednog hektara pod j-tom kulturom;

a_{ij} - vreme potrebno i-toj poljoprivrednoj mašini za obradu jednog hektara pod j-tom kulturom;

a_{i0}^k - raspoloživo vreme rada i-te poljoprivredne mašine u k-toj sezoni;

b_j^k - broj radnika koje treba angažovati u k-toj sezoni po jednom hektaru pod j-tom kulturom;

b_0^k - broj radnika sa kojima raspolaže poljoprivredno dobro za k-tu sezonu;

s_{vj} - količina v-tog materijala (semena, zaštitnog sredstva, veštačkog đubriva) potrebna po jednom hektaru pod j-tom kulturom;

s_{v0} - raspoloživa količina v-tog materijala;

h - raspoloživa obradiva površina u hektarima;

$\underline{h}_j$ - minimalna površina pod j-tom kulturom;

$\overline{h}_j$ - maksimalna površina pod j-tom kulturom.

Model se sastoji od funkcije kriterijuma

$$z_0 = \sum_{j=1}^{n} (q_j p_j - t_j) x_j,$$

koja označava ukupan čist prihod poljoprivrednog dobra, pa je potrebno naći njenu maksimalnu vrednost uz sledeće ograničavajuće faktore:

$$\sum_{j=1}^{n} a_{ij} x_j \le a_{i0}^k \ , \ i = 1, 2, \ldots, m, \ k = 1, 2, \ldots, r,$$

$$\sum_{j=1}^{n} b_j^k x_j \le b_0^k \ , \ k = 1, 2, \ldots, r,$$

$$\sum_{j=1}^{n} s_{vj} x_j \le s_{v0} \ , \ v = 1, 2, \ldots, g,$$

$$\sum_{j=1}^{n} x_j = h,$$

$$\underline{h}_j \le x_j \le \overline{h}_j \ , \ j = 1, 2, \ldots, n.$$

```
Poljoprivreda[q_,p_,t_,a_,a0_,b_,b0_,s_,s0_,h_,hmin_,hmax_]:=
   Module[{r=Dimensions[a0][[1]],m={},v={},i,qpt,pom,n=Length[q],vn},
       vn=Dimensions[a0];
       For[i=1,i<=vn[[1]],i++,
           For[j=1,j<=vn[[2]],j++,v=AppendTo[v,-a0[[i,j]]]]];
       qpt=Table[q[[i]] p[[i]]-t[[i]],{i,n}];
       For[i=1,i<=r,i++,m=Dodaj[m,-a]];
       m=Dodaj[m,-b];     m=Dodaj[m,-s];
       pom={};
       For[i=1,i<=n,i++,pom=AppendTo[pom,1]];
       m=Dodaj[m,{pom}]; m=Dodaj[m,IdentityMatrix[n]]; m=Dodaj[m,-IdentityMatrix[n]];
```

```
    v=Dodaj[v,-b0];     v=Dodaj[v,-s0];  v=AppendTo[v,h];
    v=Dodaj[v,hmin];    v=Dodaj[v,-hmax];
    LinearProgramming[-qpt,m,v]
]
```

3.5 Primena u ishrani

Problemi ishrane daju široke mogućnosti primene linearnog programiranja. Pored rešavanja pitanja dijetalne ishrane, koja je predstavljala područje prve praktične primene linearnog programiranja, postoji niz drugih interesantnih pitanja, kao što su: minimizacija troškova ishrane, izbor optimalnih sastava ishrane, itd.

Razni prehrambeni proizvodi sadrže određene sastojke i vitamine u različitim odnosima. Minimalne potrebe ovih ili onih sastojaka u ishrani su poznate. Poznavajući raspoloživu količinu prehrambenih proizvoda i cenu po jedinici mere svakog od njih, može se postaviti pitanje: kako se mogu zadovoljiti potrebe ishrane, a da pri tome nastali troškovi budu minimalni?

Matematička formulacija ovog zadatka sastoji se u sledećem. Poznato je n različitih prehrambcnih proizvoda $P_1, Pz, \ldots, P_j, \ldots, P_n$. Označimo sa x_j nepoznate količine koje predstavljaju dnevne potrebe j-tog prehrambcnog proizvoda. U svakom prehrambenom proizvodu postoje određene količine prehrambenih sastojaka: belančevina, šećera, ugljenih hidrata, kalcijuma, gvožđa, vitamina i drugih, za život potrebnih komponenata koje ćemo označiti sa $K_1, \ldots, K_i, \ldots, K_m$. Neka veličina a_{ij} karakteriše sadržaj j-tog prehrambenog sastojka u i-tom prehrambenom proizvodu ($i = 1, \ldots, m;\ j = l, \ldots, n$). Tada je ukupna količina i-tog prehrambenog sastojka u ishrani jednaka

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n.$$

Ako se minimalna svakodnevna potreba organizma u i-tom prehrambenom sastojku označi sa b_i, može se formirati sistem ograničenja

$$\sum_{j=1}^{n} a_{ij}x_j \geq b_i,\ (i = 1, \ldots, m). \tag{3.5.1}$$

S druge strane, dnevna upotreba svakog prehrambenog prpizvoda može biti ograničena raspoloživim rezervama i mogućnostima njihove popune. Zbog toga postoji dopunski uslovi za nepoznate x_j, koji se mogu napisati u obliku

$$0 \leq x_j \leq a_j,\ (j = l, \ldots, n), \tag{3.5.2}$$

gde su a_j veličine koje se određuju na bazi raspoložive količine prehrambenog proizvoda P_j u određenom vremenskom periodu. Ako se sa c_j označi cena jedinice mere j-tog prehrambenog proizvoda, cena celokupne ishrane može se prikazati funkcijom cilja

$$F(X) = \sum_{j=1}^{n} c_j x_j. \tag{3.5.3}$$

Prema tome, optimalna ishrana biće takav sastav i svakodnevne upotrebe određenih prehrambenih proizvoda $X = (x_1, \ldots, x_n)$, koji daje minimum funkcije cilja (3.5.3), pri skupu ograničenja koja su zadata izrazima (3.5.1) i (3.5.2).

Nekada uslovi (3.5.1), u odnosu na neke prehrambene proizvode, mogu biti zadani sa dvostrukim ograničenjima, tako da umesto izraza (3.5.1) postoje ograničenja oblika

$$b_i^* \geq \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \ (i = 1, \ldots, m).$$

Radi potpunije ilustracije navedimo jedan numerički primer.

Primer 3.5.1 Potrebno je izračunati minimalne troškove ishrane sa tri proizvoda P_1, P_2 i P_3, koji sadrže određene procente pet komponenti K_1, K_2, K_3, K_4, K_5. Brojni podaci dati su u sledećoj tabeli. Sastaviti odgovarajući matematički model.

	K_1	K_2	K_3	K_4	K_5	Cena
P_1	0.05	0.10	0.00	0.15	0.20	6
P_2	0.13	0.18	0.10	0.00	0.25	4
P_3	0.10	0.00	0.15	0.07	0.00	9
Propisani minimun	0.05	0.10	0.20	0.14	0.17	

Tabela 3.6.1.

Iz opšte formulacije matematičkog modela ishrane proizilazi da su nepoznate x_1, x_2, x_3, a da predstavljaju broj kilograma proizvoda P_1, P_2 i P_3, koje treba uzimati u dnevnoj ishrani da bi ona udovoljila propisane uslove i da bi troškovi ishrane bili minimalni.

Prema tome, za dati brojni primer funkcija cilja ima oblik

$$F(x_1, x_2, x_3) = 6x1 + 4x2 + 9x3,$$

dok su ograničenja zadana sistemom nejednačina

$$\begin{aligned} 0.05x_1 + 0.13x_2 + 0.10x_3 &\geq 0.05, \\ 0.10x_1 + 0.18x_2 &\geq 0.10, \\ 0.10x_2 + 0.15x_3 &\geq 0.20, \\ 0.15x_1 + 0.07x_3 &\geq 0.14, \\ 0.20x_1 + 0.25x_2 &\geq 0.17. \end{aligned}$$

Zadatak se sastoji u nalaženju minimuma funkcije cilja pri zadanim ograničenjima.

Ne postoje teškoće da se analogija sa ovog malog primera prenese na opšti slučaj i konstruiše odgovarajuća tablica polaznih podataka oblika kakav ima prethodna tablica. Na taj način postiže se preglednost svih potrebnih polaznih podataka.

3.6 Transport proizvodnje

Ovde će se objasniti matematički model organizacije transporta poljoprivredne proizvodnje, pod pretpostavkom da postoji m - punktova proizvodnje (ekonomije) $A_1, \ldots, A_m$ i n - punktova potrošnje $B_1, \ldots, B_n$.

Neka poljoprivredno gazdinstvo raspolaže sa g_k prevoznih sredstava k-iog tipa ($k = 1, \ldots, s$). Tipovi transportnih sredstava razlikuju se po nosivosti, brzini

kretanja, eksploatacionim karakteristikama i troškovima prevoza po jedinici mere tereta.

U cilju formulacije odgovarajućeg matematičkog modela uvedimo sledeće oznake:

a_i - količina proizvoda izražena jedinicom mere koja se proizvede u punktu A_i $(i = 1, \ldots, m)$,

b_j - količina proizvoda izražena istom jedinicom mere koja je potrebna punktu B_j $(j = 1, \ldots, n)$,

d_k - količina tereta koja se može prevesti prevoznim sredstvom k-tog tipa $(k = 1, \ldots, s)$,

c_{ik} - troškovi koji nastaju usled dolaska praznog prevoznog sredstva k-tog tipa iz mesta njegovog stacioniranja u mesto proizvodnje A_i radi utovara,

c^*_{jk} - troškovi koji nastaju usled vraćanja praznog prevoznog sredstva k-tog tipa iz punkta B_j u mesto njegovog stalnog stacioniranja,

c^{**}_{ijk} - troškovi koji nastaju usled prevoza proizvoda iz punkta A_i u punkt B_j na jednoj mašini k-tog tipa,

x_{ijk} - broj prevoznih sredstava k-tog tipa upućenih u punkt A_i za prevoz tereta u punkt B_j.

Pretpostavimo da usled dužine transporta nije moguća upotreba nijednog prevoznog sredstva više od jedanput (što znači da svako prevozno sredstvo ostvaruje celu turu: mesto stacioniranja - mesto proizvodnje - mesto potrošnje).

Koristeći se uvedenim oznakama, formulisani zadatak svodi se na nalaženje promenljivih x_{ijk} posredstvom traženja minimuma funkcije cilja

$$F(X) = \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{s}(c_{ik} + c^*_{jk} + c^{**}_{ijk})x_{ijk},$$

pri uslovima ograničenja oblika.

$$\begin{aligned}
\sum_{j=1}^{n}\sum_{k=1}^{s} d_k x_{ijk} &\leq a_i,\ (i = 1, \ldots, m)\\
\sum_{i=1}^{m}\sum_{k=1}^{s} d_k x_{ijk} &= b_j,\ (j = 1, \ldots, n)\\
\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ijk} &\leq g_k,\ (k = 1, \ldots, s)\\
x_{ijk} &\geq 0 \quad (i = 1, \ldots, m; j = 1, \ldots, n; k = 1, \ldots, s).
\end{aligned}$$

Prvi skup ograničenja dat je na osnovu raspoloživih količina proizvodnje, drugi na osnovu potreba u punktovima potrošnje, a treći na osnovu ograničenja transportnih sredstava k-tog tipa $(k = 1, \ldots, s)$.

3.7 Primena u planiranju proizvodnje

Osnovni zadatak planiranja optimalne proizvodnje primenom linearnog programiranja sastoji se u pronalaženju takvih količina raznih artikala, koje jedno ili više

udruženih preduzeća mogu da proizvedu uz najpovoljnije korišćenje raspoloživih ili novih resursa (radne snage, mašina za rad, sirovina i materijala), pod uslovom da je obezbeđen plasman na tržištu celog asortimana proizvodnje. Ovde se razmatra više zadataka iz oblasti planiranja i programiranja proizvodnje koji se svode na linearno programiranje.

Izbor asortimana za slučaj ograničenja jedne kategorije resursa

Najpre ćemo razmotriti jedan uprošćen matematički model optimizacije proizvodnje, koji se sastoji u optimalnom korišćenju raspoloživog kapaciteta mašina. Pretpostavimo da se raspolaže sa n mašina $M_1, \ldots, M_j, \ldots, M_n$ na kojima bi trebalo proizvoditi m artikala $A_1, \ldots, A_i, \ldots, A_m$. Neka se svaki i-ti artikal obrađuje na svakoj j-toj mašini za vreme a_{ij}. Poznat je kapacitet svake mašine M_j izražen u određenim jedinicama vremena sa c_j $(j = 1, \ldots, n)$, kao i ukupno vreme obrade bilo kog i-tog artikla na svim mašinama, koje će se označiti sa

$$b_i = \sum_{j=1}^{n} a_{ij}, \ (i = 1, \ldots, m).$$

Odgovarajući matematički model koji obezbeđuje maksimizaciju proizvodnje, za koju je obezbeđen plasman na tržištu, sastoji se u nalaženju vrednosti

$$X = (x_1, \ldots, x_m),$$

tj. količina asortimana, za koje će funkcija cilja

$$F(X) = \sum_{i=1}^{m} b_i x_i, \tag{3.7.1}$$

dobiti maksimalnu vrednost, a da pri tome budu zadovoljena ograničenja

$$\begin{aligned} \sum_{i=1}^{m} a_{ij} x_i &\leq c_j, \ (j = 1, \ldots, n), \\ x_i &\geq 0, \ (i = 1, \ldots, m). \end{aligned} \tag{3.7.2}$$

Sličan zadatak, kao što je zadatak formulisan matematičkim modelom (3.7.1) - (3.7.2), može se formulisati sa ciljem obezbeđenja maksimalnog dohotka. Ako u tom smislu uvedemo dodatne oznake d_i $(i = 1, \ldots, m)$, koje imaju značenje dobiti po komadu i-tog artikla, zadržavajući ista značenja svih napred uvedenih oznaka, zadatak maksimizacije dobiti sastoji se u nalaženju količina x_i $(i = l, \ldots, m)$ pojedinh artikala za koje će funkcija cilja

$$F(X) = \sum_{i=1}^{m} d_i x_i,$$

dobiti maksimalnu vrednost pri istim ograničenjima oblika (3.7.2).

Izbor asortimana za slučaj ograničenja više kategorija resursa

U oba izložena zadatka optimizacije tretirana su ograničenja jedne kategorije resursa (ograničen kapacitet mašina za rad). Međutim, zadatak optimizacije uslova proizvodnje postaje nešto složeniji, ako se uzme u razmatranje ograničenje više različitih kategorija resursa.

Pretpostavimo da je u proizvodnju uključeno n mašina $M_1, \ldots, M_j, \ldots, M_n$, na kojima radi s kategorija radnika $R_1, \ldots, R_k, \ldots, R_s$ i koristi se r vrsta sirovina i materijala $S_1, \ldots, S_j, \ldots, S_r$. Neka se može proizvoditi m artikala $A_1, \ldots, A_i, ..., A_m$.

U cilju formulacije odgovarajućeg matematičkog modela uvedimo sledeće oznake:

x_i - količina i-tog artikla (proizvoda) koju treba odrediti pomoću matematičkog modela optimizacije ($i = 1, \ldots, m$),

c_i - dohodak po jedinici i-tog artikla, a_{ij} - vreme potrebno za obradu i-tog artikla na j-toj mašini ($j = 1, \ldots, n$), a_j - kapacitet j-te mašine izražen u vremenskim jedinica

b_{ki} - vreme potrebno za obradu i-tog artikla od strane radnika k-te kategorije specijalnosti ($k = 1, \ldots, s$),

b_k - raspoloživi fond vremena radnika k-te kategorije specijalnosti,

d_{iv} - količine v-te sirovine (materijala) koja je potrebna za proizvodnju i-tog artikla ($v = 1, \ldots, r$),

d_{iv} - raspoloživa količina v-te vrste sirovine (materijala),

e_i - količine i-tog artikla koja se može prodati na tržištu.

Koristeći se uvedenim oznakama, može se formulisati matematički model u kome se maksimizira funkcija cilja

$$F(X) = \sum_{i=1}^{m} c_i x_i$$

koja predstavlja ukupnu dobit od celokupne proizvodnje, tj. traže se one vrednosti $x_1, \ldots, x_m$ koje zadovoljavaju skup ograničenja

$$\begin{aligned}
\sum_{i=1}^{m} a_{ij} &\leq a_i, \ (j = 1, \ldots, n), \\
\sum_{i=1}^{m} b_{ki} x_i &\leq b_k, \ (k = 1, \ldots, s), \\
\sum_{i=1}^{m} d_{iv} x_i &\leq d_v, \ (v = 1, \ldots, r), \\
0 &\leq x_i \leq e_i, \ (i = 1, \ldots, m),
\end{aligned}$$

a da pri tome definisana funkcija cilja dobije maksimalnu vrednost. Slično kao u prethodnom slučaju i ovde se može formulisati funkcija cilja koja ima drugo značenje (maksimizacija proizvodnje, maksimizacija produktivnosti, tj. odnosa između vrednosti ukupne proizvodnje i angažovane radne snage za ostvarivanje te proizvodnje, itd.), a da pri tome ograničenja ostanu ista, kao što su napred formulisana.

3.8 Upravljanje zalihama

Ovde će se prikazati jedan matematički model upravljanja zalihama koji se svodi na primenu linearnog programiranja. Cilj koji se postavlja sastoji se u planiranju proizvodnje u zavisnosti od potreba tržišta, kako bi troškovi skladištenja gotove robe bili rainimalni.

Pretpostavimo da se proizvodi n artikala na m različitih mašina u s jednakih vremenskih intervala ($i = 1, \ldots, m; j = 1, \ldots, , n; k = 1, \ldots, s$). U cilju formulisanja odgovarajućeg modela, pomoću koga se vrši minimizacija troškova skladištenja, uvedimo sledeće oznake:

a_{ij} - vreme trajanja operacije na i-toj mašini za j-ti artikal, ($i = 1, \ldots, m; j = 1, \ldots, n$),

b_{ik} - kapacitet i-te mašine u k-tom vremenskom periodu ($k = 1, \ldots, s$),

c_{jk} - troškovi skladištenja jedinice j-tog artikla, koji su jednaki za bilo koji k-ti vremenski interval,

x_{jk} - obim proizvodnje j-tog artikla u k-tom vremenskom intervalu,

p_{jk} - tražnja j-tog artikla u k-tom vremenskom intervalu,

$X_{jk} = \sum_{v=1}^{k} x_{jv}$ - kumulativ proizvodnje j-tog artikla za prvih k vremenskih intervala,

$P_{jk} = \sum_{v=1}^{k} p_{jv}$ - kumulativ tražnje j-tog proizvoda za prvih k vremenskih intervala.

Prema tome, zadatak se sastoji u određivanju minimuma funkcije cilja

$$F = \sum_{j=1}^{n} \sum_{k=1}^{s} c_{jk}(X_{jk} - P_{jk}),$$

koja predstavlja troškove skladištenja za sve j-te artikle u svim vremenskim intervalima, pod uslovima da kumulativ proizvodnje nije manji od kumulativa tražnje, tj.

$$X_{jk} \geq P_{jk},$$

kao i da su zadovoljena ograničenja kapaciteta proizvodnje

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_{jk} &\leq b_{ik}, \ (i = 1, \ldots, m; k = 1, \ldots, s), \\ x_{jk} &\geq 0, \ (i = 1, \ldots, m; k = 1, \ldots, s). \end{aligned}$$

Koristeći se napred uvedenim oznakama za X_{jk} P_{jk}, formulisani matematički model može se prikazati u obliku funkcije cilja

$$F = \sum_{j=1}^{n} \sum_{k=1}^{s} c_{jk} \left[\sum_{v=1}^{k} (x_{jv} - p_{jv}) \right],$$

koju je potrebno minimizirati pri ograničenjima oblika

$$\begin{aligned}\sum_{v=1}^{k}(x_{jv}-p_{jv}) &\geq 0,\ (j=1,\ldots,n;k=1,\ldots,s),\\ \sum_{j=1}^{n}a_{ij}x_{jk} &\leq b_{ik},\ (i=1,\ldots,m;k=1,\ldots,s),\\ x_{jk} &\geq 0,\ (j=1,\ldots,n;k=1,\ldots,s).\end{aligned}$$

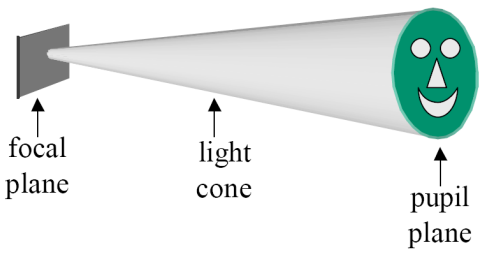
Na bazi rezultata koji se mogu dobiti na osnovu izloženog modela, može se vršiti upravljanje zalihama i proizvodnjom.

4 Primene u astronomiji i elektronici

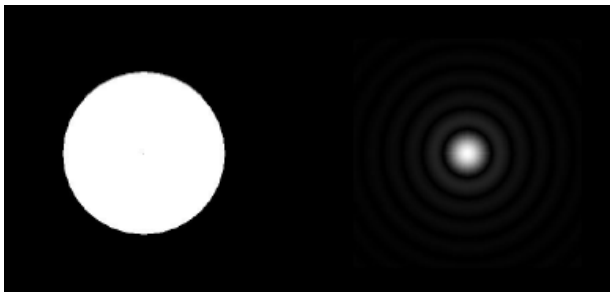
U ovoj glavi ćemo pokazati dve praktične primene prethodno izloženih metoda matematičkog programiranja (linearnog programiranja i višekriterijumske optimizacije), u astronomiji i u telekomunikacijama (elektronici). Linearno programiranje i višekriterijumska optimizacija, kao i ostale mnogobrojne metode matematičkog programiranja imaju primene u skoro svim oblastima nauke, tehnike i uopšte u svakodnevnom životu. Sledeća dva primera ilustruju tu činjenicu.

4.1 Izračunavanje optimalne maske teleskopa

U ovom odeljku razmotrićemo primenu linearnog programiranja na problem projektovanja teleskopa pomoću koga ćemo moći da utvrdimo da li oko neke zvezde kruže planete. Na slici 4.1.1 prikazan je uprošćeni model teleskopa.



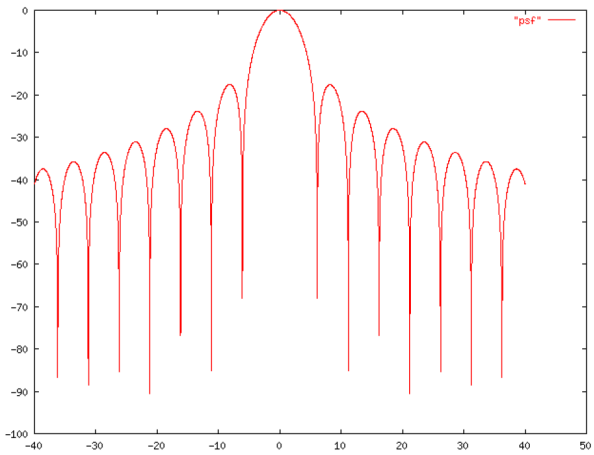
Slika 4.1.1. Principijelna šema teleskopa.



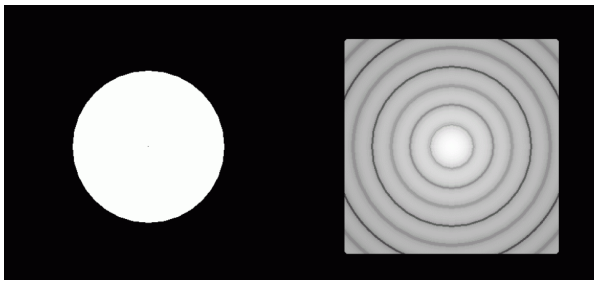
Slika 4.1.2. Profil objektiva i difrakciona slika.

Svetlost sa udaljene zvezde pada na ravan objektiva teleskopa (eng. *pupil plane*), prelama se kroz sočivo objektiva i pada na žižnu ravan (eng. *focal plane*). Svi svetlosni zraci koji pod istim uglom padnu na objektiv, posle prelamanja padaju u istu tačku na žižnoj ravni. Svetlosni zraci koji dolaze sa zvezde su skoro paralelni, pa bi prema tome lik zvezde trebao biti jedna tačka u žižnoj ravni (ako podesimo teleskop tako da se zvezda nalazi na glavnoj optičkoj osi objektiva, onda bi to bila tačka sa koordinatama $(0,0)$). Međutim, usled talasne prirode svetlosti, dolazi do difrakcije na otvoru objektiva i umesto jedne svetle tačke, vidimo mrlju konačne veličine koju okružuju prstenovi (slika 4.1.2).

Ovi prstenovi, iako su tamniji od centralne mrlje, ipak su intenzivniji od svetlosti bilo koje planete koja kruži oko zvezde. Na primer, za posmatrača sa strane, svetlost koja potče od Sunca je 10^{10} puta intenzivnija od svetlosti Zemlje. Postavljanjem maske na objektiv, menjamo jačinu i oblik prstenova na difrakcionoj slici. Treba naći takav oblik maske tako da intenzitet svetlosti u okolini svetle mrlje bude dovoljno mali. Na slici 4.1.3 je prikazan intenzitet svetlosti na koordinatnoj x osi žižne ravni. Intenzitet svetlosti je u decibelima $I[\text{dB}] = 10 \log \frac{I}{I_{ref}}$, pri čemu je za referentni intenzitet I_{ref} uzeta vrednost u koordinatnom početku. Na slici 4.1.4 je difrakciona slika, pri čemu je sada referentni nivo -110dB.



Slika 4.1.3. Intenzitet svetlosti na x koordinatnoj osi.



Slika 4.1.4. Profil objektiva i difrakciona slika, pri čemu je nula na −110dB.

Pretpostavimo da je profil maske iskazan funkcijom $A(x)$, tj da je otvoreni deo objektiva:

$$\left\{(x,y) \mid -\frac{1}{2} \le x \le \frac{1}{2},\ -A(x) \le y \le A(x)\right\} \tag{4.1.1}$$

Naravno, prethodno je izvršena odgovarajuća normalizacija dužina. Intenzitet svetlosti je direktno proporcionalan kvadratu jačine električnog polja E. Sa (x,y) označavaćemo koordinate tačke u ravni objektiva a sa (ξ,ζ) u žižnoj ravni. Ako usvojimo Fraunhoferovu aproksimaciju [3], imamo sledeći izraz za jačinu električnog polja u tački (ξ,ζ) u žižnoj ravni:

$$E(\xi,\zeta) = \int_{-1/2}^{1/2}\int_{-A(x)}^{A(x)} e^{i(x\xi+y\zeta)}dydx = 4\int_0^{1/2}\cos(x\xi)\frac{\sin(A(x)\zeta)}{\zeta}dx$$

Poslednja jednakost važi zbog simetrije oblika maske (4.1.1). Uslov koji ćemo sada nametnuti je da u prethodno zadatoj oblasti $\mathcal{O}$ važi sledeći uslov:

$$-\alpha E(0,0) \le E(\xi,\zeta) \le \alpha E(0,0), \quad (\xi,\zeta) \in \mathcal{O} \tag{4.1.2}$$

Pretpostavimo sada da su tačke skupa $\mathcal{O}$ na x osi (tj. da je $\zeta = 0$). Tada odnos $\frac{\sin(A(x)\zeta)}{\zeta}$ postaje samo $A(x)$ odnosno električno polje je jednako:

$$E(\xi,0) = 4\int_0^{1/2}\cos(x\xi)A(x)dx$$

Tada uslov (4.1.2) postaje:

$$\int_0^{1/2} A(x)\left[\cos(x\xi) - \alpha\right]dx \le 0$$
$$\int_0^{1/2} A(x)\left[\cos(x\xi) + \alpha\right]dx \le 0$$

Cilj nam je da vidljiva površina objektiva bude što je moguće veća, odnosno da maksimizujemo

$$\int_{-1/2}^{1/2} 2A(x)dx = 4\int_0^{1/2} A(x)dx$$

Znači, posmatramo sledeći optimizacioni problem:

$$\begin{aligned} &\max\ 4\int_0^{1/2} A(x)dx \\ &p.o. \int_0^{1/2} A(x)\left[\cos(x\xi)-\alpha\right]dx \le 0 \\ &\qquad \int_0^{1/2} A(x)\left[\cos(x\xi)+\alpha\right]dx \le 0 \\ &\qquad 0 \le A(x) \le \frac{1}{2}. \end{aligned} \tag{4.1.3}$$

Ako sada izvršimo odgovarajuću diskretizaciju integrala i skupa $\mathcal{O}$ dobijamo sledeći problem linearnog programiranja [4, 8]:

$$\begin{aligned} &\max\ 4\sum_{i=1}^{N_x} B_i A(x_i) \\ &p.o. \sum_{i=1}^{N_x} C_i A(x_i)\left[\cos(x_i\xi_j)-\alpha\right] \le 0, \quad j=1,\ldots,N_\xi \\ &\qquad \sum_{i=1}^{N_x} D_i A(x_i)\left[\cos(x_i\xi_j)-\alpha\right] \le 0 \quad j=1,\ldots,N_\xi \\ &\qquad 0 \le A(x_i) \le \frac{1}{2}. \end{aligned} \tag{4.1.4}$$

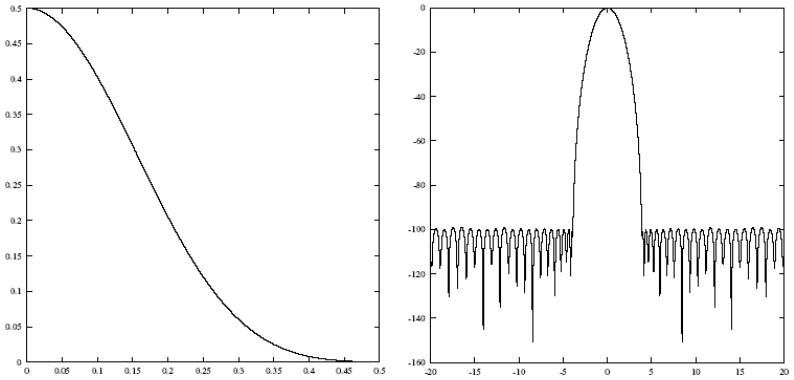
Ovde smo sa N_x i N_ξ označili broj diskretizacionih nivoa redom za promenljive x i ξ. ξ_i su vrednosti diskretizovane promenljive ξ. Ako je skup $\mathcal{O}$ zadat npr. kao interval

$$\mathcal{O} = \{(\xi, 0) \mid \xi_0 < \xi < \xi_1\}$$

onda možemo uzeti ekvidistantnu podelu $\xi_i = a + (i-1)\frac{b-a}{N_\xi - 1}$. Vrednosti x_i, B_i, C_i i D_i su koeficijenti odgovarajućih kvadraturnih formula kojima se aproksimiraju integrali u funkciji cilja i uslovima. U najprostijem slučaju možemo uzeti $B_i = C_i = D_i = \frac{1}{N_x}$ i $x_i = -\frac{1}{2} + \frac{i-1}{N_x - 1}$.

Problem (4.1.4) predstavlja problem linearnog programiranja u simetričnom obliku. Ulogu promenljivih ovde imaju vrednosti funkcije $A(x)$ u tačkama x_i ($A(x_i)$).

Ako je sada $\alpha = 10^{-5}$, $\xi_0 = 4$, $\xi_1 = 40$, $N_x = N_\xi = 1000$, i diskretizacija je uniformna, dobijamo rešenje sa slike 4.1.5 za funkciju $A(x)$ [4].



Slika 4.1.5. Profil optimalne maske $A(x)$ i intenzitet svetlosti na x osi ako se koristi optimalna maska.

Sa slike vidimo, da je intenzitet svetlosti za $\xi > 4$ manji ili jednak -110dB, tako da je u ovom pojasu moguće zapaziti lik neke planete koja kruži oko zvezde.

Slična analiza se može obaviti ukoliko skup $\mathcal{O}$ ima nešto drugačiji oblik, ili se formira više od jedne maske [4].

4.2 Projektovanje FIR filtara

U ovom odeljku razmotrićemo primenu linearnog programiranja u projektovanju digitalnih FIR filtara. FIR filtri, kao i uopšte digitalni filtri imaju veliku ulogu u obradi *digitalnih signala* [2]. Digitalni signali su za razliku od kontinualnih signala definisani samo u diskretnim vremenskim trenucima i predstavljaju se kao niz realnih vrednosti $x(t)$, $t = 1, 2, \ldots$. U analizi digitalnih signala najčešće se koriste diskretna Fouriereova transformacija i z-transformacija. One su date pomoću sledećeg izraza:

$$X(f) = \sum_{n=-\infty}^{+\infty} x(t)e^{2\pi f i n} \qquad X(z) = \sum_{n=-\infty}^{+\infty} x(n)z^{-n} \tag{4.2.1}$$

Fouriereova transformacija $X(f)$ se naziva i *spektar* signala $x(t)$. Napomenimo da je $X(f)$ periodična funkcija sa periodom 1. Nadalje ćemo $X(f)$ posmatrati samo na intervalu $(-\frac{1}{2}, \frac{1}{2})$. Digitalni filtri, kao i analogni, modifikuju ulazni signal tako što propuštaju na izlaz samo one komponente signala čija je frekvencija u tačno određenom opsegu. Frekvencijska karakteristika idealnog filtra može se prikazati pomoću izraza [2]:

$$|H(f)| = \left|\frac{Y(f)}{X(f)}\right| = \begin{cases} 1, & f \in \mathcal{O} \\ 0, & \text{U suprotnom} \end{cases}$$

gde su $Y(f)$, $X(f)$ redom spektri izlaznog i ulaznog signala a $\mathcal{O}$ skup frekvencija koje se propuštaju. Idealni filtar je nemoguće u praksi realizovati, pa se zato pristupa traženju realnog filtra (koji se može realizovati u praksi) koji što bolje aproksimira idealni. Neka je $h(t)$ inverzna diskretna Fouriereova transformacija funkcije $H(f)$, data pomoću:

$$h(t) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(f) e^{2\pi i f t} df.$$

Imamo da su signal na izlazu i signal na ulazu povezani sledećom relacijom:

$$y(t) = \sum_{s=-\infty}^{+\infty} h(s) x(t-s)$$

koji predstavlja *konvoluciju* nizova $x(t)$ i $h(t)$.

Za FIR (*Finite Impulse Response*) filtre važi da postoji broj n_0 takav da je $h(t) = 0$ ako je $|t| > n_0$. Broj n_0 nazivamo *red* filtra. U suprotnom, u pitanju je IIR filtar (*Infinite Impulse Response*). Uvešćemo još jednu pretpostavku u našu analizu, da je $h(t)$ simetrična funkcija po t. Ovaj uslov obezbeđuje linearnost faze kompleksne funkcije $H(f)$. Sada je $|H(f)|$, takođe, simetrična funkcija i važi $|H(f)| = |A(f)|$ gde je $A(f)$ definisana pomoću [2]:

$$A(f) = h(0) + 2 \sum_{t=1}^{n_0} h(t) \cos(2\pi f t). \tag{4.2.2}$$

Nadalje ćemo posmatrati samo vrednosti $H(f)$ (odnosno $A(f)$) za $f > 0$. Neka je $\mathcal{O} = [0, f_0]$ (propusnik niskih frekvencija). Znači, potrebno je minimizirati:

$$\begin{aligned} &\min \int_0^{f_0} |A(f) - 1| df \\ &\min \int_{f_0}^{1/2} |A(f)| df, \end{aligned} \tag{4.2.3}$$

Ovim smo dobili problem bezuslovne višekriterijumske nelinearne optimizacije. Promenljive su u ovom slučaju vrednosti $h(t)$, $t = 0, \ldots, n_0$. Uvedimo sada pomoćne funkcije $\tau(f)$ i $\eta(f)$. Problem (4.2.3) možemo ekvivalentno predstaviti u obliku:

$$\begin{aligned} &\min \int_0^{f_0} \tau(f) df \\ &\min \int_{f_0}^{1/2} \eta(f) df \\ &p.o. \ -\tau(f) \le A(f) - 1 \le \tau(f), \quad f \in \mathcal{O} \\ &\quad -\eta(f) \le A(f) \le \eta(f), \quad f \notin \mathcal{O}. \end{aligned} \tag{4.2.4}$$

Ako sada izvršimo diskretizaciju po frekvenciji, slično kao u prethodnom odeljku, dobijamo problem linearne višekriterijumske optimizacije. U praksi se najčešće

fiksira jedno odstupanje (npr. na skupu $\mathcal{O}^C$) i traži se minimum drugog. Sada problem (4.2.3) svodimo na sledeći problem:

$$\begin{aligned} \min \ & \int_0^{f_0} \tau(f)df \\ p.o. \ & -\tau(f) \le A(f) - 1 \le \tau(f), \quad f \in \mathcal{O} \\ & -\epsilon \le A(f) \le \epsilon, \quad f \notin \mathcal{O} \\ & \tau(f) \ge 0 \end{aligned} \tag{4.2.5}$$

a ako uvedemo diskretizaciju po f direktno dobijamo problem linearnog programiranja u simetričnom obliku:

$$\begin{aligned} \min \ & \sum_{j=1}^{N'_f} B_j \tau(f_j) \\ \text{p.o.} \ & -\tau(f_j) \le h(0) + 2\sum_{t=1}^{n_0} h(t)\cos(2\pi f_j t) - 1 \le \tau(f_j), \quad j = 1, \ldots, N'_f \\ & -\epsilon \le h(0) + 2\sum_{t=1}^{n_0} h(t)\cos(2\pi f_j t) \le \epsilon, \quad j = N'_f + 1, \ldots, N_f + N'_f \end{aligned} \tag{4.2.6}$$

Primetimo da je ovo svođenje isto kao kod metode ϵ ograničenja iz prethodne glave. Kao i u prethodnom odeljku, i ovde su f_j i B_j, $j = 1, \ldots, N'_f$ čvorovi i koeficijenti odgovarajuće kvadraturne formule. Za $j > N'_f$ vrednosti f_j su iz skupa $\mathcal{O}^C$. Promenljive su nam sada $\tau(f_j)$, $j = 1, \ldots, N'_f$ i $h(t)$, $t = 0, \ldots, n_0$.

U radu [1], razmatran je problem reprodukcije signala pomoću tri FIR filtra, pri čemu svaki propušta različiti opseg frekvencija. Imamo da je spektar paralelne veze ovih filtara:

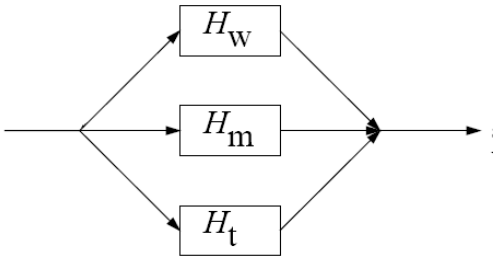
$$H(f) = H_w(f) + H_m(f) + H_t(f)$$

gde su $H_w(f)$, $H_m(f)$ i $H_t(f)$ gde su redom spektri svakog filtra (indeksi potiču od engleskih reči w-*woofer*, m-*midrange*, t-*tweetier*). Propusni opsezi ($\mathcal{O}$) za ove filtre su redom $\mathcal{O}_w = [0, f_w]$,$\mathcal{O}_m = [0, f_{m1}] \cup [f_{m2}, \frac{1}{2}]$ i $\mathcal{O}_t = [f_t, \frac{1}{2}]$. U ovom slučaju nam je cilj da funkcija $A(f)$ bude što bliža jedinici za $f \in [0, \frac{1}{2}]$. Prema tome, imamo sledeći problem:

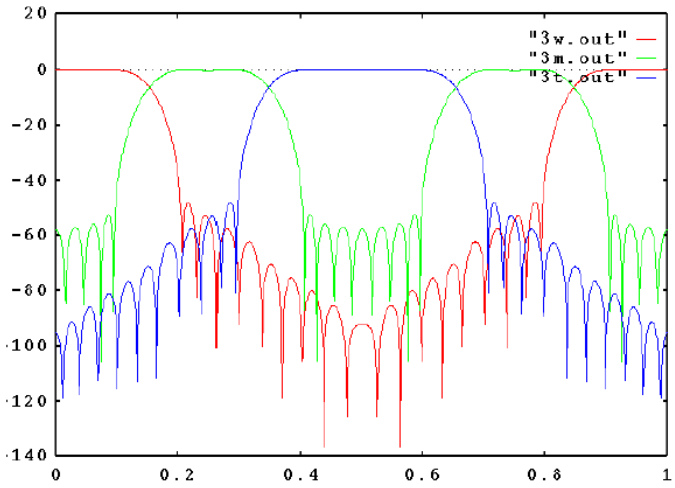
$$\begin{aligned} \min \ & \int_0^{1/2} |A_w(f) + A_m(f) + A_t(f) - 1| \\ p.o. \ & -\epsilon \le A_i(f) \le \epsilon, \quad f \in \mathcal{O}_i, \quad i = w, m, t \end{aligned} \tag{4.2.7}$$

koji na sličan način svodimo na problem linearnog programiranja.

Na slici 4.2.2 su prikazane funkcije $A_w(f)$, $A_m(f)$ i $A_t(f)$ za sledeće vrednosti parametara [1]: $N_f = 1000$, $f_w = 0.2$, $f_t = 0.7$, $f_{m1} = 0.1$ i $f_{m2} = 0.4$.



Slika 4.2.1. Blok šema paralelne veze tri filtra.



Slika 4.2.2. Grafici funkcija $A_w(f)$, $A_m(f)$ i $A_t(f)$.

5 Linearna regresija

U mnogim naukama veoma često postoji potreba određivanja direktne funkcionalne zavisnosti između dve ili više veličina. Oblast matematičke statistike koja se bavi utvrđivanjem i opisivanjem ovih zavisnosti naziva se *regresija*. Ukoliko se utvrđuje linearna zavisnost posmatranih obeležja, reč je o *linearnoj regresiji*. Postoje dva moguća pristupa ovom problemu.

Kod prvog pristupa se posmatra uticaj obeležja (slučajnih veličina) $X_1, \ldots, X_n$ na takođe slučajnu veličinu Y. Pri tom, treba odrediti funkciju $f(x_1, \ldots, x_n)$ takvu da slučajna veličina $f(X_1, \ldots, X_n)$ najbolje aproksimira Y. Kao mera odstupanja ovih dvaju veličina, najčešće se koristi srednjekvadratno odstupanje, tj uslov da je $E(Y - f(X))$ minimalno. Ovim tipom regresije nećemo se baviti.

U drugom pristupu, posmatra se uticaj određenog broja neslučajnih veličina, *kontrolnih faktora* na vrednost odgovarajuće slučajne veličine Y. Ova veza je de-

deterministička (neslučajna). Na veličinu Y takođe utiču i slučajni faktori, kao i drugi neslučajni faktori čiji se uticaj ne može efektivno sagledati. Pretpostavićemo da su ova dva uticaja međusobno nezavisna, aditivna (vrednost promenljive Y je zbir vrednosti determinističke i slučajne komponente) i da je očekivanje slučajne komponente jednako nuli.

U nastavku ćemo detaljnije opisati problem linearne regresije druge vrste i dati rešenja u slučaju L_2, L_1 i L_∞ norme. Pokazaćemo da se slučajevi L_1 i L_∞ norme svode na problem linearnog programiranja.

5.1 Formulacija problema linearne regresije druge vrste

Ukoliko je veza kontrolnih faktora i promenljive Y linearna, radi se o *linearnom modelu regresije druge vrste*. Označimo sa $a_1, \ldots, a_n$ vrednosti kontrolnih faktora, $x_1, \ldots, x_n$ vrednosti koeficijenata linearne veze, a sa ϵ slučajnu komponentu. Imamo:

$$Y = a_1 x_1 + \ldots, a_n x_n + \epsilon. \tag{5.1.1}$$

Potrebno je izvršiti odgovarajuću procenu koeficijenata $x_1, \ldots, x_n$ tako da se relacija (5.1.1) "najbolje" slaže sa dobijenim eksperimentalnim rezultatima. Pri tome se vrši niz eksperimenata sa različitim vrednostima kontrolnih faktora $a^k = (a_{k1}, \ldots, a_{kn})$ $k = 1, \ldots, m$ gde je m ukupan broj eksperimenata. Označimo sa Y_i i ϵ_i, $i = 1, \ldots, m$ dobijene vrednosti veličine Y i slučajne veličine ϵ u i-tom eksperimentu (ponavljanju). Dobija se:

$$Y_i = a_{i1} x_1 + \ldots, a_{in} x_n + \epsilon_i, \quad i = 1, \ldots, m. \tag{5.1.2}$$

Obeležimo sada $\mathbf{Y} = [Y_1 \ \cdots \ Y_m]^T$, $A = [a_{ij}]$, $x = [x_1 \ \cdots \ x_n]$ kao i $\epsilon = [\epsilon_1 \ \cdots \ \epsilon_m]$. Prethodni sistem možemo zapisati u matričnoj formi na sledeći način:

$$\mathbf{Y} = \mathbf{A}\mathbf{x} + \epsilon \tag{5.1.3}$$

Potrebno je, za zadate vrednosti matrice A i vektora $\mathbf{Y}$ naći vrednost vektora x takvu da je norma vektora ϵ minimalna.

$$\min \ \|\epsilon\| = \|\mathbf{Y} - \mathbf{A}\mathbf{x}\|. \tag{5.1.4}$$

Napomena 5.1.1 U izrazu (5.1.1) kontrolni faktori ne moraju biti funkcionalno nezavisni. Takođe, moguće je posmatrati opštiji model:

$$Y = x_1 f_1(a_1, \ldots, a_p) + \ldots + x_n f_n(a_1, \ldots, a_p) + \epsilon$$

gde su sada $a_1, \ldots, a_p$ kontrolni faktori (zavisni ili nezavisni), a $f_1, \ldots, f_n$ funkcije kojima se opisuje zavisnost veličine Y od faktora $a_1, \ldots, a_p$. U ovom slučaju, matricu A određujemo kao $A = [f_j(a_{i1}, \ldots, a_{ip})]$, $i = 1, \ldots, m$, $j = 1, \ldots, n$, a dalji postupak je potpuno isti.

U opštem slučaju, možemo posmatrati bilo koju normu u prethodnom izrazu. Međutim, najčešće su u upotrebi L_1 norma (suma apsolutnih vrednosti komponenata), L_2 (suma kvadrata) kao i L_∞ (maksimalna apsolutna vrednost komponente).

U zavisnosti od toga koja se funkcija norme koristi razlikujemo L_1 *linearnu regresiju*, L_2 *linearnu regresiju* kao i L_∞ *linearnu regresiju*. Detaljno ćemo proučiti ove tri vrste regresije.

Primer 5.1.1 Predpostavimo da se slučajna veličina Y sastoji iz dva dela: konstantne, ali nepoznate vrednosti x i slučajnog odstupanja od te fiksirane vrednosti ϵ. Imamo da važi:

$$Y = x + \epsilon \tag{5.1.5}$$

Ovaj model je sličan problemu regresije pri čemu ovde nemamo kontrolne faktore. Vrednost x je jednaka očekivanju slučajne promenljive Y ($E\epsilon = 0$), pa se prema tome problem svodi na određivanje nepoznatog očekivanja promenljive Y na osnovu vrednosti eksperimenata. Ukoliko posmatramo L_1, L_2 odnosno L_∞ normu, rešenja problema su data sledećim izrazom:

$$\begin{aligned} L_1: &\quad x^*_{L_1} = Y_{(m+1)/2} \\ L_2: &\quad x^*_{L_2} = \overline{Y} = \frac{Y_1 + \ldots + Y_m}{m} \\ L_\infty: &\quad x^*_{L_\infty} = \frac{Y_{(m)} + Y_{(1)}}{2}. \end{aligned} \tag{5.1.6}$$

Pri tome smo sa $Y_{(m+1)/2}$ označili *medijanu* realizovanog uzorka $(Y_1, \ldots, Y_m)$ a sa $Y_{(k)}$ odgovarajuću statistiku poredka (k-ti element po veličini realizovanog uzorka). Sve ove vrednosti na neki način predstavljaju "srednju vrednost" broja ostvarenih poena na ispitu. Koje će se od ova tri rešenja usvojiti zavisi od konkretnog slučaja.

Neka se npr. našim eksperimentom određuje broj poena studenata na nekom ispitu i neka su dobijene sledeće vrednosti:

$$28, 62, 80, 84, 86, 86, 92, 95, 98$$

Imamo da je $x^*_{L_1} = 86$, $x^*_{L_2} = 79$ a $x^*_{L_\infty} = 63$. Vidimo da prema prvoj proceni student koji je osvojio 80 poena je loše uradio test, prema drugoj je prosečno uradio, a prema trećoj dobro.

5.2 L_2 regresija

Sigurno najrasprostanjenija varijanta linearne regresije u praksi je L_2 regresija. Kod ovog tipa minimizira se euklidska L_2 norma vektora ϵ, definisana na sledeći način:

$$\|\epsilon\|_2 = \left(\epsilon_1^2 + \ldots + \epsilon_m^2\right)^{1/2}$$

Ukoliko se umesto broja 2 u predhodnom izrazu stavi broj $p > 1$ dobijamo L_p normu. Kada p teži beskonačnosti, imamo da važi $\lim_{p\to+\infty} \|y\|_p = \|y\|_{+\infty}$. Problem koji posmatramo ima sledeći oblik:

$$\min \ \|\mathbf{Y} - A\mathbf{x}\|_2. \tag{5.2.1}$$

Ukoliko kvadriramo prethodni izraz, dobijamo ekvivalentan problem:

$$\min \ F(x) = \sum_{i=1}^{m} \left(Y_i - \sum_{j=1}^{n} a_{ij} x_j \right)^2. \tag{5.2.2}$$

Ovim smo problem sveli na bezuslovni problem kvadratnog programiranja[1]. Ovaj problem se jednostavno rešava izjednačavanjem parcijalnih izvoda funkcije $F(x)$ sa

[1] Kvadratnim programiranjem se nećemo baviti u ovoj knjizi

nulom. Na taj način dobijamo da optimalno rešenje x^* zadovoljava sledeći linearni sistem jednačina:

$$A^T A\mathbf{x} = A^T\mathbf{Y}. \tag{5.2.3}$$

Ukoliko je matrica $A^T A$ regularna, rešenje sistema (5.2.3) je jedinstveno i jednako:

$$x^* = (A^T A)^{-1} A^T\mathbf{Y}. \tag{5.2.4}$$

Ukoliko matrica $A^T A$ nije regularna, može se pokazati da sistem (5.2.3) (odnosno problem (5.1.4)) ima beskonačno mnogo rešenja i da su opisana na sledeći način:

$$x^* = A^\dagger\mathbf{Y} + (I_m - AA^\dagger)z. \tag{5.2.5}$$

gde je $A^\dagger$ *uopšteni inverz (pseudoinverz)* matrice A, a $z \in R^m$ proizvoljan vektor.

Primer 5.2.1 Najjednostavniji i najčešće primenljivan u praksi regresioni model je onaj u kome je Y linearna funkcija jednog kontrolnog faktora x, tj važi:

$$Y = ax + b + \epsilon$$

Pritom su zadate vrednosti $(x_1, Y_1), \ldots (x_m, Y_m)$. Očigledno, ovaj problem je ekvivalentan provlačenju (ili *fitovanju*) najpribližnije prave kroz zadate tačke (x_i, Y_i). Dobija se:

$$A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix}, \quad A^T A = \begin{bmatrix} \sum_{i=1}^m x_i^2 & \sum_{i=1}^m x_i \\ \sum_{i=1}^m x_i & m \end{bmatrix}, \quad A^T Y = \begin{bmatrix} \sum_{i=1}^m x_i Y_i \\ \sum_{i=1}^m x_i \end{bmatrix}. \tag{5.2.6}$$

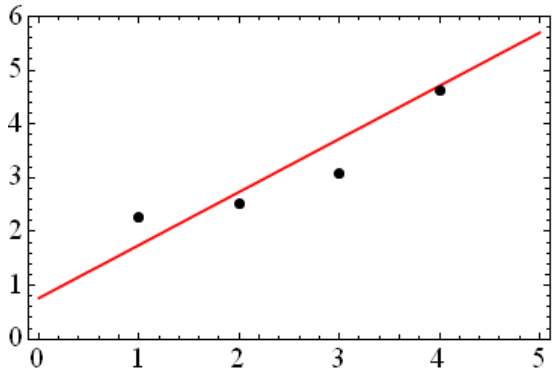
U konkretnom slučaju se primenom jednačine (5.2.4), i na osnovu prethodnih formula dobija rešenje traženog problema $x^* = [a, b]^T$. Na primer, za skup vrednosti:

x_i	1	2	3	4	5
Y_i	2.26	2.52	3.08	4.62	6.15

Imamo da važi:

$$A^T A = \begin{bmatrix} 55 & 15 \\ 15 & 5 \end{bmatrix}, \quad A^T Y = \begin{bmatrix} 65.76 \\ 18.63 \end{bmatrix},$$

pa sada lako dobijamo da je rešenje $x^* = \begin{bmatrix} a & b \end{bmatrix}^T = \begin{bmatrix} 0.98 & 0.77 \end{bmatrix}^T$. Date tačke, kao i fitovana prava prikazane su na slici 5.2.1.



Slika 5.2.1. Grafik funkcije $f(x) = ax + b$ zajedno sa tačkama.

5.3 L_1 i L_∞ regresija

Za razliku od L_2 regresije, u slučaju L_1 regresije ne postoji egzaktna formula, kao što je to (5.2.5) kojom bi se opisala rešenja ovog problema. U ovom slučaju minimizujemo L_1 normu vektora ostatka, odnosno tražimo vektor x^* koji zadovoljava:

$$x^* = \text{argmin}_{x\in\mathbb{R}^n} \|\mathbf{Y} - A\mathbf{x}\|_1 = \text{argmin}_{x\in\mathbb{R}^n} \sum_{i=1}^{m} \left| Y_i - \sum_{j=1}^{n} a_{ij}x_j \right|. \tag{5.3.1}$$

Iako ne postoji egzaktna formula za rešenje problema (5.3.1), on se relativno lako svodi na problem linearnog programiranja. Zaista, ukoliko označimo sa t_i sledeći izraz:

$$t_i = \left| Y_i - \sum_{j=1}^{n} a_{ij}x_j \right|$$

problem (5.3.1) se lako svodi na sledeći problem linearnog programiranja:

$$\begin{aligned} &\min\ t_1 + \ldots + t_m \\ &\text{p.o.}\ -t_i \le Y_i - \sum_{j=1}^{n} a_{ij}x_j \le t_i, \quad i = 1, \ldots, m. \end{aligned} \tag{5.3.2}$$

Rešavanjem problema linearnog programiranja (5.3.2) dobijamo rešenje x^* problema L_1 regresije.

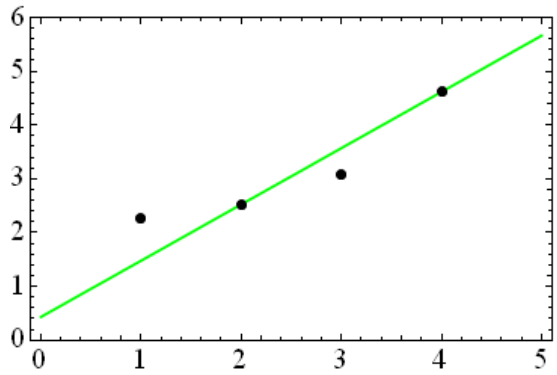
Primer 5.3.1 Posmatrajmo isti problem zadat u primeru 5.2.1. Odgovarajući ekvivalentan problem linearnog programiranja za ovaj problem je:

$$\begin{array}{llllllllll} \min & t_1 & +t_2 & +t_3 & +t_4 & +t_5 & & & & \\ \text{p.o.} & -t_1 & & & & & -a & -b & \le & -2.26 \\ & & -t_2 & & & & -2a & -b & \le & -2.52 \\ & & & -t_3 & & & -3a & -b & \le & -3.08 \\ & & & & -t_4 & & -4a & -b & \le & -4.62 \\ & & & & & -t_5 & -5a & -b & \le & -6.15 \\ & -t_1 & & & & & +a & +b & \le & 2.26 \\ & & -t_2 & & & & +2a & +b & \le & 2.52 \\ & & & -t_3 & & & +3a & +b & \le & 3.08 \\ & & & & -t_4 & & +4a & +b & \le & 4.62 \\ & & & & & -t_5 & +5a & +b & \le & 6.15. \end{array}$$

Rešenje ovog problema linearnog programiranja je:

$$[t_1, t_2, t_3, t_4, t_5, a, b]^T = \begin{bmatrix} 0.78 & 0 & 0.49 & 0 & 0.48 & 1.05 & 0.43 \end{bmatrix}^T$$

odakle dobijamo da su koeficijenti a i b redom jednaki 1.05 i 0.43. Kao i u primeru 5.2.1 i ovde prikazujemo na slici 5.3.1 grafički tačke i aproksimativnu pravu.



Slika 5.3.1. Grafik funkcije $f(x) = ax + b$ zajedno sa tačkama, L_1 regresija.

Na sličan način možemo transformisati i sledeći problem L_∞ regresije:

$$x^* = \text{argmin}_{x\in\mathbb{R}^m} \|\mathbf{Y} - A\mathbf{x}\|_\infty = \max_{1\le i\le m} \left| Y_i - \sum_{j=1}^{n} a_{ij}x_j \right| \tag{5.3.3}$$

na problem linearnog programiranja. U ovom slučaju, potrebno je da uvedemo samo još jednu dodatnu promenljivu t koja će biti upravo jednaka funkciji cilja:

$$\begin{aligned} &\min\ t \\ &\text{p.o.}\ -t \le Y_i - \sum_{j=1}^{n} a_{ij}x_j \le t, i = 1, \ldots, m. \end{aligned} \tag{5.3.4}$$

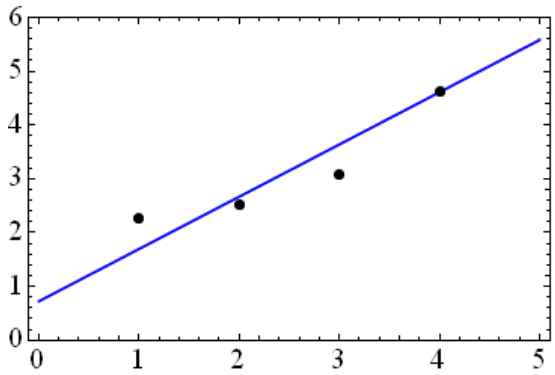
Primer 5.3.2 Sada ćemo rešiti problem iz primera 5.2.1, ali upotrebom L_∞ regresije. Problemu je ekvivalentan sledeći problem linearnog programiranja:

$$\begin{array}{lrrrcl} \min & t & & & & \\ \text{p.o.} & -t & -a & -b & \le & -2.26 \\ & -t & -2a & -b & \le & -2.52 \\ & -t & -3a & -b & \le & -3.08 \\ & -t & -4a & -b & \le & -4.62 \\ & -t & -5a & -b & \le & -6.15 \\ & -t & +a & +b & \le & 2.26 \\ & -t & +2a & +b & \le & 2.52 \\ & -t & +3a & +b & \le & 3.08 \\ & -t & +4a & +b & \le & 4.62 \\ & -t & +5a & +b & \le & 6.15 \end{array}$$

Rešenje ovog problema linearnog programiranja je:

$$x^* = \begin{bmatrix} t & a & b \end{bmatrix}^T = \begin{bmatrix} 0.56 & 0.97 & 0.72 \end{bmatrix}^T.$$

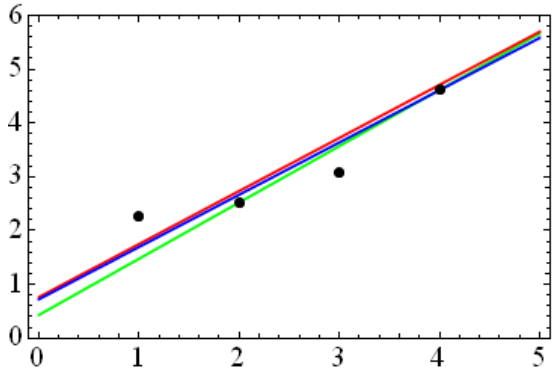
Grafik aproksimativne prave i datih tačaka prikazan je na slici 5.3.2.



Slika 5.3.2. Grafik funkcije $f(x) = ax + b$ zajedno sa tačkama, L_∞ regresija.

5.4 Poređenje L_1, L_2 i L_∞ modela regresije

Prezentovani modeli linearne regresije daju optimalna rešenja na osnovu različitih kriterijuma optimalnosti. Ova rešenja u opštem slučaju ne moraju biti jednaka. Izvršimo upoređivanje rezultata iz primera 5.2.1 dobijenih pomoću L_1, L_2 i L_∞ modela regresije. Na slici 5.4.1 su prikazane tačke zajedno sa aproksimativnim pravama za sva tri modela regresije. Uočavamo da su prave veoma bliske.



Slika 5.4.1. Sva tri modela regresije za problem iz primera 5.2.1.

Prema tome izbor modela regresije na konkretnom primeru nije od naročite važnosti. U tom slučaju je najbolje koristiti L_2 regresiju zato što se optimalno rešenje direktno dobija primenom formula (5.2.5) odnosno (5.2.6).

Međutim za neke specifične primere, L_2 model regresije često daje veoma loše rezultate koji često mogu da dovedu do potpuno pogrešnih zaključaka.

Primer 5.4.1 Posmatrajmo sledeći model, u kome je Y kvadratna funkcija [2] jednog kontrolnog faktora x:

$$Y = ax^2 + bx + c + \epsilon$$

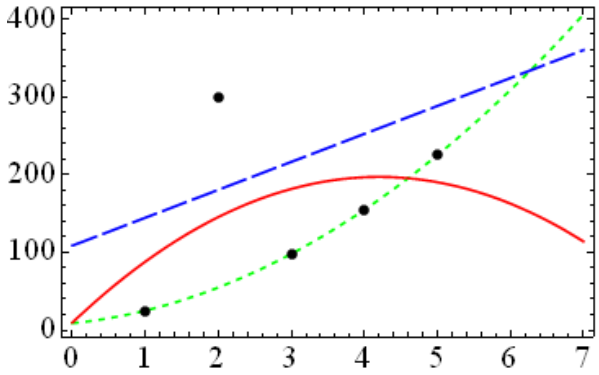
[2]Napominjemo da se ovde takođe radi o linearnom modelu regresije zato što je Y linearna funkcija parametara a, b i c modela.

Na osnovu sledećeg skupa vrednosti od $m = 5$ tačaka potrebno je odrediti parametre a, b i c tako da kvadratna funkcija $ax^2 + bx + c$ najbolje aproksimira vrednosti veličine Y:

x_i	1	2	3	4	5
Y_i	25.36	300	97.151	154.99	225.15

Primetimo da vrednosti promenljive Y rastu sa porastom vrednosti x, osim u slučaju $x = 2$ gde beležimo značajno odstupanje. Ovako velika odstupanja su često posledica sistematskih grešaka. Posmatrajmo sada kako ovo drastično odstupanje utiče na optimalne vrednosti parametara a, b i c kod svakog od modela regresije.

Postupak određivanja optimalnih vrednosti parametara prepuštamo čitaocu. Na slici 5.4.2 prikazane su date tačke i aproksimativne kvadratne funkcije za sva tri modela.



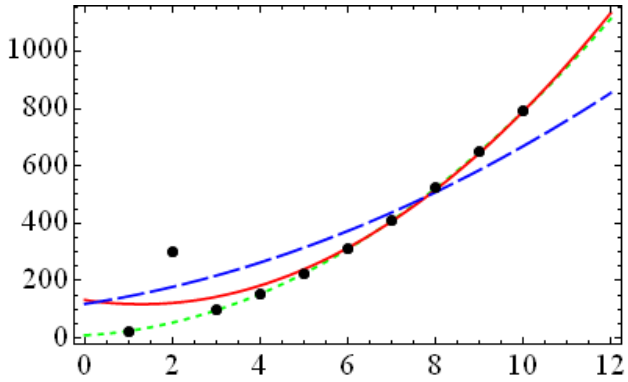
Slika 5.4.2. Sva tri modela regresije za problem kod koga jedna tačka drastično odstupa.

Primećujemo najpre da sve tačke sem druge leže praktično na jednoj paraboli koja je dobijena L_1 aproksimacijom. Prema tome, sistematska greška skoro da uopšte nije imala uticaj na optimalno rešenje kod L_1 modela pa možemo zaključiti da je na ovom primeru L_1 model dao najbolje rezultate. Nasuprot ovome, kod L_2 modela beležimo veliko odstupanje. Kao posledica sistematske greške primećujemo da je optimalna vrednost koeficijenta a u ovom slučaju negativna i da optimalna parabola ima maksimum za $x = 4.2$. Dakle može se potpuno pogrešno zaključiti da je zavisnost $Y(x)$ najpre rastuća a zatim opadajuća, pa zaključujemo da je L_2 regresija dala veoma loše rezultate na ovom primeru. Relativno loše rezultate beležimo i kod L_∞ modela. Primetimo da je optimalna parabola u ovom slučaju prava, tj da važi $a = 0$.

Primer 5.4.2 Dodajmo sada još nekoliko eksperimentalnih tačaka. Dobijamo sledeću tablicu:

x_i	1	2	3	4	5	6	7	8	9	10
Y_i	25.36	300	97.151	154.99	225.15	310.67	409.47	522.43	649.21	790.16

Ukoliko sada dodamo još nekoliko tačaka na paraboli (slika 5.4.3), dobijamo da se L_2 aproksimacija poboljšava i da se za veće vrednosti faktora x izjednačila sa L_1 aproksimacijom koja je i dalje najbolja. Primećujemo da je L_∞ aproksimacija bolja nego pre dodavanja novih vrednosti, ali je odstupanje dalje primetno.



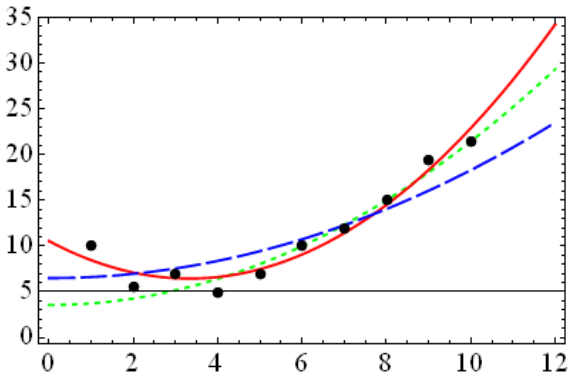
Slika 5.4.3. Poboljšana L_2 aproksimacija.

Na osnovu predhodna dva primera primećujemo da L_1 aproksimacija definitivno najbolje otklanja sistematske greške. Međutim to ponekad može biti loše, što lepo ilustruje naredni primer.

Primer 5.4.3 Posmatrajmo skup vrednosti:

x_i	1	2	3	4	5	6	7	8	9	10
Y_i	10	5.5	6.89	5.01	6.95	10.02	12.05	15	19.45	21.46

Primećujemo da Y najpre opada pa onda raste. Sa slike 5.4.4 možemo da zapazimo da parabola dobijena L_2 aproksimacijom najpre opada do vrednosti $x = 3.33$ a zatim raste, što je vrlo slično ponašanju eksperimentalnih vrednosti funkcije Y. Međutim L_1 aproksimacija ne prati početno opadanje zavisnosti, tj prvih nekoliko tačaka gde ova zavisnost opada nemaju dovoljan uticaj na optimalno rešenje L_1 modela. Slično se ponaša i L_∞ model na ovom primeru.



Slika 4.4.4. L_∞ model na istom primeru.

Na osnovu predhodna tri primera možemo zaključiti da se pri izboru modela regresije mora mnogo voditi računa o samoj prirodi eksperimentalnih rezultata na koje se model primenjuje i na osnovu toga izabrati odgovarajući model.

5.5 Implementacija u paketu MATHEMATICA

Metode L_1 i L_∞ implementirane su u programskom jeziku MATHEMATICA. Izložićemo najvažnije detalje implementacije. Funkcija `L1Reg` formira problem linearnog programiranja oblika (5.3.2), rešava ga i kao rezultat daje x^* komponentu optimalnog rešenja (t^*, x^*). Sledi implementacija ove funkcije:

```
L1Reg[A_, Y_] := Module[{i, m, n, A1, b, sol, c, a},
      {m, n} = Dimensions[A];
      c = Join[Table[1, {i, m}], Table[0, {i, n}]];
      A1 = Join[
          Table[Join[Table[KroneckerDelta[i, j], {j, m}], A[[i]]], {i, m}],
          Table[Join[Table[KroneckerDelta[i, j], {j, m}], -A[[i]]], {i, m}]];
      b = Join[Y, -Y];
      sol = LinearProgramming[c, A1, b];
      Return[Take[sol, -n]];
];
```

Funkcija `LInfReg` formira i rešava problem linearnog programiranja. Kao rezultat se takođe prosleđuje x komponenta optimalnog rešenja (t^*, x^*). Sledi implementacija ove funkcije:

```
LInfReg[A_, Y_] := Module[{i, m, n, A1, b, sol, c},
      {m, n} = Dimensions[A];
      c = Join[{1}, Table[0, {i, n}]];
      A1 = Join[
          Table[Join[{1}, A[[i]]], {i, m}],
          Table[Join[{1}, -A[[i]]], {i, m}]
      ];
      b = Join[Y, -Y];
      sol = LinearProgramming[c, A1, b];
      Return[Take[sol, -n]];
];
```

Literatura

[1] J.O. Coleman, *A Systematic Approach to the Constrained Quadratic Optimization of Embedded FIR Filters*, Conference on Information Sciences and Systems, Princeton, March 1998.

[2] Lj. Milić Z. Dobrosavljević, *Uvod u Digitalnu Obradu Signala*, Elektrotehnički fakultet, Univerzitet u Beogradu, 1999.

[3] D.M. Ivanović, V.M. Vučić, *Fizika II, Elektromagnetika i optika*, Naučna knjiga, Beograd, 1967.

[4] N.J. Kasdin, R.J. Vanderbei, D.N. Spergel, M.G. Littman. *Extrasolar Planet Finding via Optimal Apodized and Shaped Pupil Coronagraphs.* Astrophysical Journal, 582:11471161, 2003.

[5] J.J. Petrić, *Operaciona istraživanja, Knjiga prva*, Savremena administracija, Beograd, 1974.

[6] J.J. Petrić, *Operaciona istraživanja, Knjiga druga*, Savremena administracija, Beograd, 1983.

[7] O. Todorović, *Operaciona istraživanja*, Prosveta, Niš, 1999.

[8] R.J. Vanderbei, N.J. Kasdin, D.N. Spergel, *Rectangular-Mask Coronagraphs for High-Contrast Imaging*, arXiv:astro-ph/0401644 v1 30 Jan 2004.

[9] R.J. Vanderbei, *Linear Programming: Foundations and Extensions*, Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ 08544, 2001.

[10] S. Wolfram, *The Mathematica Book, 4th ed.*, Wolfram Media/Cambridge University Press, 1999.

Glava 4

Dinamičko Programiranje

1 Uvod

Opšte karakterisitke modela dinamičkog programiranja (skraćeno DP) su:

- Globalni model (problem) se rasčlanjuje na etape, i u okviru svake etape se vrši optimizacija ciljne funkcije (maksimizacija dobiti, minimizacija troškova i slično)
- Etapa (faza) se definiše kao uređen skup stanja.
- Prilikom izbora procesa upravljanja (ili dejstva), transformacija svakog stanja tekuće etape (faze) je povezana sa sledećom etapom. Ako se problem DP interpretira kao mrežni model, svaki čvor u mreži odgovara jednom stanju.
- Optimalno upravljanje počinje sa prvom ili poslednjom etapom, u zavisnosti od prirode modela.

Dinamičko programiranje je tehnika za rešavanje problema sa rekurzivnom strukturom koja poseduje sledeće karakteristike:

1. Optimalne podstrukture (Belmanov princip optimalnosti): neko optimalno rešenje na proizvoljnoj instanci sadrži optimalno rešenje svih svojih podinstanci.
2. Mali broj podproblema: ukupan broj podproblema koji će se rešavati je mali.
3. Preklapanje podproblema: u toku rekurzije pojavljuju se isti podproblemi više puta.

Princip "zavadi pa vladaj" (divide-and-conquer) je korišćen u mnogim algoritmima: da bi se rešio veliki problem, on se razbija na manje probleme koji se mogu rešiti nezavisno. U dinamičkom programiranju je ovaj princip usavršen do ekstrema:

kada se ne zna precizno koji se manji problemi rešavaju, jednostavno se reše svi, zatim se rezultati zapamte da bi se upotrebili kasnije u rešavanju većih problema [18, 19]. Postoje dve pincipijelne poteškoće u primeni ove tehnike. Prvo, nije uvek moguće kombinovanje rešenja dva problema da bi se formiralo rešenje jednog većeg. Drugo, može da postoji neprihvatljivo veliki broj malih problema koji se rešavaju.

Nije precizno definisano koji problemi mogu biti efektivno rešeni pomoću dinamičkog programiranja; postoji veliki broj "teških" (hard) problema za koje ovaj metod nije primenljiv [19], kao i mnogo "lakih" problema za koje je ovaj metod manje efikasan u odnosu na standardne algoritme [19]. Međutim, postoji tačno definisana klasa problema za koje je dinamičko programiranje sasvin efektivno [19]. Ovi problemi imaju opšte svojstvo da svaka odluka uključena u nalaženje najboljeg rešenja ostaje dobra odluka čak i kada je podproblem uključen kao deo većeg problema. Neki od takvih problema su opisani u ovoj glavi.

Dinamičko programiranje omogućava optimalno planiranje višeetapnih procesa upravljanja. Mnogi zadaci upravljanja procesima u tehnici, ekonomiji, vojsci, fizici, biologiji, itd., mogu se predstaviti u vidu višeetapnih procesa, na koje se može primeniti metod dinamičkog programiranja, a sa ciljem da se dobije optimalni plan upravljanja. Pri tome se pod optimalnim upravljanjem podrazumeva ono upravljanje koje daje najbolje rešenje.

Za praktičnu primenu dinamičkog programiranja potrebno je da svaki razmatrani proces ima svoj jasno postavljeni matematički model, sa precizno definisanim ciljem (funkcija cilja), koji treba maksimizirati (minimizirati) i ograničenjima koja se moraju uzeti u obzir u toku procesa. Za ovako postavljen zadatak, ukoliko su zadovoljeni uslovi koje zahteva primena metoda DP, potrebno je naći funkcionalne relacije primenom principa optimalnosti. U izvesnim slučajevima moguće je naći analitičko rešenje, ali rešenje se najčešće nalazi numeričkim procesima.

Dinamičko programiranje se obično primenjuje u problemima optimizacije: problem može imati mnogo rešenja, svako rešenje ima vrednost, a traži se rešenje koje ima optimalnu (najveću ili najmanju) vrednost. U slučaju da postoji više rešenja koja imaju optimalnu vrednost, obično se traži bilo koje od njih.

Sama reč "programiranje" ovde se (kao i u linearnom programiranju) odnosi na popunjavanje tabele pri rešavanju problema, a ne na upotrebu kompjutera i programskih jezika. Tehnike optimizacije koje imaju elemente dinamičkog programiranja su bile poznate i ranije, ali tvorcem metoda danas se smatra profesor Ričard E. Belman. Belman je proučavao dinamičko programiranje i dao čvrstu matematičku osnovu za ovaj način rešavanja problema. Uopšteno govoreći, problem se rešava tako što se uoči hijerarhija problema istog tipa, sadržanih u glavnom problemu, i rešavanje se počne od najjednostavnijih problema. Vrednosti i delovi rešenja svih rešenih podproblema se pamte u tabeli, pa se dalje njihovim kombinovanjem dobija rešenja većih podproblema sve do rešenja glavnog problema.

Napomenimo da se u literaturi koja obrađuje dinamičko programiranje [6, 19, 20, 25, 13] sreću dva različita pristupa ovom problemu. Prvi pristup je uglavnom vezan za knjige u kojima se proučavaju problemi operacionih istraživanja i matematičkog progamiranja, dok se drugi pristup prvenstveno javlja u knjigama koje su posvećene

teoriji algoritama. Između ova dva pristupa nema suštinskih razlika i oba pristupa tretiraju potpuno isti problem. U nastavku ove glave izložićemo veći broj problema koji se rešavaju tehnikom dinamičkog programiranja, pri čemu ćemo za neke probleme koristiti prvi a za neke drugi pristup. Konkretno, problemi 2.1, 2.2, 2.3, 5.1, 5.2 i 5.3 formulisani su korišćenjem prvog pristupa dok je za formulaciju ostalih problema korišćen drugi pristup.

Koristićemo sledeće konvencije i oznake. Elemente niza A označavaćemo sa $a_1, \ldots, a_N$, ili sa $A(1), \ldots, A(N)$. U slučaju dvostrukih indeksa, radi bolje čitljivosti nećemo pisati a_{b_c} nego $a(b_c)$, ili $a(b(c))$. Iste primedbe važe i za matrice.

2 Klasični problemi dinamičkog programiranja

U ovom odeljku izložićemo nekoliko klasičnih primena tehnike DP. Svaki pododeljak predstavlja jedan od klasičnih problema koji se ovom tehnikom uspešno rešavaju. Za svaki problem data je najpre formulacija a zatim i detaljno opisano rešenje. Svi navedeni problemi pripadaju problemima optimizacije i efikasno se svode na probleme linearnog, nelinearnog i celobrojnog programiranja a za probleme 2.1, 2.2 i 2.3 date su i ekvivalentne formulacije. Zatim je izloženo rešenje primenom tehnike DP i uradjen jedan ili više praktičnih primera gde su izložena rešenja primenjena na konkretne probleme.

2.1 Prosta raspodela jednorodnog resursa

Predpostavimo da imamo odredjenu količinu nekog resursa i da je potrebno rasporediti svu raspoloživu količinu ovog resursa na n potrošača. Potrošači mogu biti različiti proizvodni procesi. Poznata je efikasnost svakog potrošača koja predstavlja zaradu koju potrošač ostvaruje. Naravno, efikasnost nekog potrošača je funkcija količine resursa dodeljene tom potrošaču. Napomenimo da efikasnost ne mora da predstavlja zaradu, već to može biti neki drugi parametar (potrošnja materijala, vreme, i slično) koji želimo optimizovati. Za svaki potrošač je poznata maksimalna količina resursa koja mu se može dodeliti (kapacitet potrošača). Prema tome, poznate su sledeće veličine:

S - ukupna raspoloživa količina resursa.

$g_j(x)$ - funkcija dobiti koja se ostvaruje kada se količina x raspoloživog resursa dodeli j-tom procesu (mašini, liniji).

Q_j - kapacitet potrošača j.

Potrebno je pronaći količine resursa x_j, $j = 1, \ldots, n$ koje treba dodeliti svakom potrošaču tako da se maksimizira ukupna dobit koju potrošači zajedno ostvaruju.

Rešenje: Označimo sa $F(x)$ ukupnu dobit koju ostvaruju mašine ako se j-toj mašini dodeli količina resursa x_j, gde je $x = (x_1, \ldots, x_n)$. Prema tome, potrebno je naći

$$\max F(x) = \sum_{j=1}^{n} g_j(x_j). \tag{2.1.1}$$

Sledeća dva uslova se dobijaju iz činjenica da je S ukupna raspoloživa količina resursa i da je kapacitet j-tog potrošača jednak Q_j:

$$\begin{aligned} &\sum_{i=1}^{n} x_i \leq S, \\ &0 \leq x_j \leq Q_j,\ j = 1, \ldots, n. \end{aligned} \tag{2.1.2}$$

Ovim smo konstruisali jedan optimizacioni problem kod koga je funkcija cilja u opštem slučaju nelinearna. Ukoliko su g_j linearne funkcije, dobija se problem linearnog programiranja koji je moguće rešavati primenom nekog od metoda izloženih u prvoj i drugoj glavi. Pritom moramo da predpostavimo da su vrednosti x_j po prirodi realni brojevi (tj da se resursi mere količinski, npr. struja, voda, gorivo, itd...). Ukoliko su x_j celi brojevi, problem (2.1.1-2.1.2) je problem celobrojnog linearnog odnosno nelinearnog programiranja. Rešavanje problema linearnog odnosno nelinearnog, a pogotovu celobrojnog programiranja je u opštem slučaju složen problem (potrebni su složeni algoritmi koji dugo rade).

Medjutim, ovaj problem se mnogo efikasnije rešava primenom tehnike DP. Označimo sa $f_i(s)$ maksimalnu dobit koju je moguće ostvariti sa ukupnom količinom resursa s i prvih i potrošača. Potrebno je naći $f_n(S)$. Posmatraćemo složeniji problem kod koga je potrebno naći sve vrednosti $f_i(s)$ za svako $0 \leq s \leq S$ i $1 \leq i \leq n$.

Princip optimalnosti se interpretira na sledeći način: ako sa x_n označimo količinu koja je dodeljena n-tom potrošaču, tada za dalje raspoređivanje preostaje $s - x_n$ jedinica koje bi trebalo **optimalno raspodeliti** na preostale potrošače.

Prema tome, maksimalna dobit $f_n(s)$ koja se ostvaruje raspodelom s jedinica resursa na n procesa se može iskazati sledećim izrazom:

$$f_n(s) = \max_{0 \leq x_n \leq \min\{s, Q_n\}} \{g_n(x_n) + f_{n-1}(s - x_n)\}. \tag{2.1.3}$$

Predhodnu formulu možemo dalje primeniti na izračunavanje vrednosti funkcije $f_{n-1}(s)$ za $0 \leq s \leq S$. U graničnom slučaju, maksimalna dobit na osnovu dodele x_1 jedinica resursa potrošaču 1 jednaka je:

$$f_1(s) = \max_{0 \leq x_1 \leq \min\{s, Q_1\}} \{g_1(x_1)\}, \quad \text{uz uslov} \quad f_1(0) = 0. \tag{2.1.4}$$

Izloženo rešenje je po prirodi rekurzivno i jednostavno za implementaciju ali pati od eksponencijalne složenosti (vreme izvršenja je eksponencijalna funkcija argumenta n). Ovaj problem će biti detaljnije razmotren u narednom odeljku prilikom razmatranja problema ranca. Za sad, dovoljno je da konstatujemo da ukoliko vrednosti funkcija f_i izračunavamo u redosledu $i = 1, \ldots, n$ i pritom pamtimo predhodno izračunate vrednosti vremenska složenost postaje linearna po n.

Prema tome, najpre se izračunavaju vrednosti funkcije $f_1(s)$ na osnovu graničnog uslova (2.1.4), a zatim korišćenjem jednačine (2.1.3) se redom izračunavaju vrednosti funkcija $f_2(s), f_3(s), \ldots, f_n(s)$ pri čemu se predhodno izračunate vrednosti

pamte. Maksimalna dobit u polaznom problemu je jednaka $f_n(S)$. Medjutim, sada je potrebno odrediti i samo optimalno rešenje $x^* = (x_1^*, \ldots, x_n^*)$.

Optimalno rešenje rekonstruišemo u obrnutom redosledu. Na osnovu relacije (2.1.3) imamo da je x_n^* zapravo vrednost promenljive x_n za koju je izraz na desnoj strani maksimalan. Pošto smo vrednosti funkcije f_{n-1} već izračunali, možemo da odredimo x_n^*. Nastavljajući ovaj postupak dalje, dobijamo i ostale vrednosti x_j^* u sledećem redosledu $x_n^*, \ldots, x_2^*, x_1^*$:

$$\begin{aligned} f_n(S) &= \max_{0 \le x_n^* \le \min\{S, Q_n\}} \{g_i(x_n^*) + f_{n-1}(S - x_n^*)\}, \\ f_{i-1}(S - x_i^*) &= \max_{0 \le x_{i-1} \le \min\{S - x_n^*, Q_{n-1}\}} \{g_{n-1}(x_{n-1}) + f_{n-2}\left[(S - x_n^*) - x_{n-1}\right]\}. \end{aligned}$$

.....

Na kraju, za vrednost x_1^* maksimalna dobit u prvom potrošaču je:

$$\begin{aligned} f_1(S - x_n^* - x_{n-1}^* - \cdots - x_2^*) &= \max_{0 \le x_1 \le \min\{S - x_n^* - \cdots - x_2^*, Q_1\}} g_1(x_1) \\ &= g_1\left(S - x_n^* - x_{n-1}^* - \cdots - x_2^*\right). \end{aligned}$$

Napomenimo još jednom da je potrebno izračunati vrednosti funkcija $f_i(s)$ za svaku vrednost argumenta $0 \le s \le S$. Pritom, ukoliko su x_i (odnosno s) po prirodi celi brojevi, onda je i S ceo broj i potrebno je izračunati ukupno S vrednosti svake od funkcija f_i. Medjutim, ukoliko su x_i realni brojevi, tada je potrebno znati vrednosti funkcija f_i na celom intervalu $[0, S]$. Prema tome, da bi metod mogli praktično da primenimo u ovom slučaju, potrebno je naći analitički izraz za svaku od funkcija $f_i(s)$.

Razmotrimo sada dva primera problema proste raspodele jednorodnog resursa. U prvom primeru funkcija cilja je linearna dok je u drugom nelinearna.

Primer 2.1.1 U pekari je potrebno rasporediti jednorodni resurs, vreće brašna, na linije za proizvodnju peciva, kolača i hleba. Ostvarena dobit linearno zavisi od količine dodeljenog resursa i iznosi:

- za pecivo: $d_1 = 4$ n.j./jedinici resursa;
- za kolače: $d_2 = 5$ n.j./jedinici resursa;
- za hleb: $d_3 = 2$ n.j./jedinici resursa.

U razmatranom periodu, pekara će imati na raspolaganju do 8 vreća brašna, dok su kapaciteti svake linije prerada do 5 vreća brašna. Zbog higijensko-tehničkih uslova nije dozvoljeno presipanje brašna, odnosno deljenje jedinice resursa.

Rešenje: U ovom primeru, jedinica resursa jeste vreća brašna. Neka je x_1 broj vreća brašna koje su upotrebljene za pecivo, x_2 broj vreća brašna koje su upotrebljene za kolače, dok je x_3 vreća brašna koje su upotrebljene za hleb.

Matematički model je

$$\begin{aligned} \max \quad & F(x) = 4x_1 + 5x_2 + 2x_3 \\ \text{p.o.} \quad & x_1 + x_2 + x_3 \le 8 \\ & 0 \le x_i \le 5, \ i = 1, 2, 3. \end{aligned}$$

Dodatno ograničenje je zahtev da promenljive budu celobrojne (sadržaj vreća se ne moze deliti i presipati). Neposredne dobiti od raspodele resursa linijama su date u sledećoj tabeli.

S	x_1	$f_1(S)$	x_2	$f_2(S)$	x_3	$f_3(S)$
0	0	0	0	0	0	0
1	1	4	1	5	1	5
2	2	8	2	10	2	10
3	3	12	3	15	3	15
4	4	16	4	20	4	20
5	5	20	5	25	5	25
6	5	20	5	29	5	29
7	5	20	5	33	5	33
8	5	20	5	37	5	37

Tabela 2.1.1.

Vrednosti dobiti u koloni $f_1(S)$ se izračunavaju na osnovu izraza za funkciju cilja, u slučaju da se resurs dodeljuje samo prvoj liniji (pravljenje peciva):

$$f_1(S) = \max_{0 \le x_1 \le 5} 4x_1.$$

U slučaju da se resurs dodeljuje liniji 1 i 2 (za proizvodnju peciva koristi se x_1 vreća i x_2 vreća za proizvodnju kolača), optimalna dobit je prikazana u koloni $f_2(S)$, koja se računa po sledećem izrazu:

$$f_2(S) = \max_{0 \le x_2 \le 5} \{5x_2 + f_1(S - x_2)\}$$

na osnovu čega sledi:

$$f_2(0) = \max_{x_2=0} \{5x_2 + f_1(0 - x_2)\} = \max\{5 \cdot 0 + f_1(0-0)\} = 0 + 0 = 0$$

$$f_2(1) = \max_{\substack{x_2=0 \\ x_2=1}} \{5x_2 + f_1(1 - x_2)\} = \max \left\{ \begin{array}{l} 5 \cdot 0 + f_1(1-0) = 0 + 4 = 4 \\ 5 \cdot 1 + f_1(1-1) = 5 + 0 = 5 \end{array} \right\} = 5$$

$$f_2(2) = \max_{\substack{x_2=0 \\ x_2=1 \\ x_2=2}} \{5x_2 + f_1(2 - x_2)\} = \max \left\{ \begin{array}{l} 5 \cdot 0 + f_1(2-0) = 0 + 8 = 8 \\ 5 \cdot 1 + f_1(2-1) = 5 + 4 = 9 \\ 5 \cdot 2 + f_1(2-2) = 10 + 0 = 10 \end{array} \right\} = 10.$$

.....

$$f_2(8) = \max_{0 \le x_2 \le 5} \{5x_2 + f_1(8 - x_2)\} = \max \left\{ \begin{array}{l} 5 \cdot 0 + f_1(8-0) = 0 + 20 = 20 \\ 5 \cdot 1 + f_1(8-1) = 5 + 20 = 25 \\ 5 \cdot 2 + f_1(8-2) = 10 + 20 = 30 \\ 5 \cdot 3 + f_1(8-3) = 15 + 20 = 35 \\ 5 \cdot 4 + f_1(8-4) = 20 + 16 = 36 \\ 5 \cdot 5 + f_1(8-5) = 25 + 12 = 37 \end{array} \right\} = 37.$$

Ako se u razmatranje uvrsti i raspodela resursa na treću liniju, optimalna dobit se računa po sledećoj formuli:

$$f_3(S) = \max_{0 \le x_3 \le 5} \{2x_3 + f_2(S - x_3)\}$$

$$f_3(0) = \max_{x_3=0} \{2 \cdot x_3 + f_2(0 - x_3)\} = \max\{2 \cdot 0 + f_2(0-0)\} = \max\{2 \cdot 0 + f_2(0-0)\}$$

$$= 0 + 0 = 0$$

$$f_3(1) = \max_{\substack{x_3=0 \\ x_3=1}} \{2 \cdot x_3 + f_2(1 - x_3)\} = \max \left\{ \begin{array}{l} 2 \cdot 0 + f_2(1-0) = 0 + 5 = 5 \\ 2 \cdot 1 + f_2(1-1) = 2 + 0 = 2 \end{array} \right\} = 5$$

$$f_3(2) = \max_{\substack{x_3=0 \\ x_3=1 \\ x_3=2}} \{2 \cdot x_3 + f_2(2 - x_3)\} = \max \left\{ \begin{array}{l} 2 \cdot 0 + f_2(2-0) = 0 + 10 = 10 \\ 2 \cdot 1 + f_2(2-1) = 2 + 5 = 7 \\ 2 \cdot 2 + f_2(2-2) = 4 + 0 = 4 \end{array} \right\} = 10$$

.....

$$f_3(8) = \max_{0\le x_3\le 5}\{2\cdot x_3 + f_2(8-x_3)\} = \max\left\{\begin{array}{l} 2\cdot 0 + f_2(8-0) = 0+37 = 37 \\ 2\cdot 1 + f_2(8-1) = 2+33 = 35 \\ 2\cdot 2 + f_2(8-2) = 4+29 = 33 \\ 2\cdot 3 + f_2(8-3) = 6+25 = 31 \\ 2\cdot 4 + f_2(8-4) = 8+20 = 28 \\ 2\cdot 5 + f_2(8-5) = 10+15 = 25 \end{array}\right\} = 37$$

Optimalno rešenje glasi:

$$x^* = \begin{bmatrix} x_1^* \\ x_2^* \\ x_3^* \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 0 \end{bmatrix},$$

dok je vrednost funkcije cilja

$$f(x^*) = 4x_1 + 5x_2 + 2x_3 = 4\cdot 3 + 5\cdot 5 + 2\cdot 0 = 37.$$

Kako je u pitanju jednostavni slučaj distribucije jednorodnog resursa, problem se može rešiti direktnim poređenjem vrednosti funkcije dobiti, a u zavisnosti od mogućih stanja distribucije resursa, kao što je dato u tabeli 2.1.2.

x_1	0	1	2	3	4	5	5	5	5
$(S-x_1)$	8	7	6	5	4	3	3	3	3
x_2	5	5	5	5	4	3	3	3	3
$(S-x_2)$	3	3	3	3	4	5	5	5	5
x_3	3	2	1	0	0	0	0	0	0
$g_1(x_1)$	0	4	8	12	16	20	20	20	20
$g_2(x_2)$	25	25	25	25	20	15	15	15	15
$g_3(x_3)$	6	4	2	0	0	0	0	0	0
$\sum g_i(x_i)$	31	33	35	37	36	35	35	35	35

Tabela 2.1.2.

I ovim metodom su dobijeni isti rezultati, to jest, optimalno rešenje je

$$x^* = \begin{bmatrix} x_1^* \\ x_2^* \\ x_3^* \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 0 \end{bmatrix},$$

pri čemu je vrednost funkcije cilja:

$$f(x^*) = 4x_1 + 5x_2 + 2x_3 = 4\cdot 3 + 5\cdot 5 + 2\cdot 0 = 37.$$

Primer 2.1.2 Za potrebe rada proizvodnog sistema, potrebno je pamučno platno (jednorodni resurs) raspodeliti na tri moguće proizvodne linije, pri čemu je kapacitet proizvodnih linija limitiran na količinu platna (resursa) $0 \le x_i \le 4$, $i = 1, 2, 3$. Količina platna je takođe u razmatranom periodu ograničena i iznosi $R = 7$ jedinica. Ostvarena dobit na linijama zavisi od kvadrata količine prerađenog platna, i to:

za liniju 1: $d_1 = 3x_1^2$
za liniju 2: $d_2 = 4x_2^2$
za liniju 3: $d_3 = 2x_3^2$.

Ograničenu količinu platna treba tako distribuirati proizvodnim linijama tako da ostvarena dobit od njegove prerade bude maksimalna.

Rešenje: Potrebno je maksimizirati funkciju kriterijuma (cilja):

$$D(x) = 3x_1^2 + 4x_2^2 + 2x_3^2.$$

Ograničenja potiču od ograničene količine resursa i od kapaciteta linija, i glase:

$$\begin{aligned} x_1 + x_2 + x_3 &\le 7 \\ 0 \le x_i &\le 4. \end{aligned}$$

Dobit se može izraziti kao funkcija raspoloživog resursa, i data je u sledećoj tabeli.

R	x_1	$D_1(x_1)$	x_2	$D_2(x_2)$	x_3	$D_3(x_3)$
0	0	0	0	0	0	0
1	1	3	1	4	1	4
2	2	12	2	16	2	16
3	3	27	3	36	3	36
4	4	48	4	64	4	64
5	4	48	4	67	4	67
6	4	48	4	76	4	76
7	4	48	4	91	4	91

Tabela 2.1.3.

Kolona $D_1(x_1)$ u tabeli se računa pomoću izraza:

$$D_1(x_1) = \max_{0 \le x_1 \le 4} \left\{ 3x_1^2 \right\}.$$

Ako se u razmatranje uvede prvi i drugi proces, vrednosti u koloni $D_2(x_2)$ dobijaju se pomoću izraza:

$$D_2(x_2) = \max_{0 \le x_2 \le 4} \left\{ 4x_2^2 + D_1(R - x_2) \right\}.$$

Odatle slede rezultati:

$$\begin{aligned}
D_2(0) &= \max_{x_2=0} \left\{ 4x_2^2 + D_1(0 - x_2) \right\} = \max \left\{ 4 \cdot 0^2 + D_1(0-0) \right\} = 0, \\
D_2(1) &= \max_{\substack{x_2=0 \\ x_2=1}} \left\{ 4x_2^2 + D_1(1 - x_2) \right\} = \max \left\{ \begin{array}{l} 4 \cdot 0^2 + D_1(1-0) = 3 \\ 4 \cdot 1^2 + D_1(1-1) = 4 \end{array} \right\} = 4, \\
D_2(2) &= \max_{\substack{x_2=0 \\ x_2=1 \\ x_2=2}} \left\{ 4x_2^2 + D_1(2 - x_2) \right\} = \max \left\{ \begin{array}{l} 4 \cdot 0^2 + D_1(2-0) = 12 \\ 4 \cdot 1^2 + D_1(2-1) = 7 \\ 4 \cdot 2^2 + D_1(2-2) = 16 \end{array} \right\} = 16,
\end{aligned}$$

$$\begin{aligned}
D_2(3) &= \max_{\substack{x_2=0 \\ x_2=1 \\ x_2=2 \\ x_2=3}} \left\{ 4x_2^2 + D_1(3 - x_2) \right\} = \max \left\{ \begin{array}{l} 4 \cdot 0^2 + D_1(3-0) = 27 \\ 4 \cdot 1^2 + D_1(3-1) = 16 \\ 4 \cdot 2^2 + D_1(3-2) = 19 \\ 4 \cdot 3^2 + D_1(3-3) = 36 \end{array} \right\} = 36, \\
D_2(4) &= \max_{\substack{x_2=0 \\ x_2=1 \\ x_2=2 \\ x_2=3 \\ x_2=4}} \left\{ 4x_2^2 + D_1(4 - x_2) \right\} = \max \left\{ \begin{array}{l} 4 \cdot 0^2 + D_1(4-0) = 48 \\ 4 \cdot 1^2 + D_1(4-1) = 31 \\ 4 \cdot 2^2 + D_1(4-2) = 28 \\ 4 \cdot 3^2 + D_1(4-3) = 39 \\ 4 \cdot 4^2 + D_1(4-4) = 64 \end{array} \right\} = 64,
\end{aligned}$$

$$D_2(5) = \max_{\substack{x_2=0\\x_2=1\\x_2=2\\x_2=3\\x_2=4}} \left\{4x_2^2 + D_1(5-x_2)\right\} = \max \left\{ \begin{array}{l} 4\cdot 0^2 + D_1(5-0) = 48 \\ 4\cdot 1^2 + D_1(5-1) = 52 \\ 4\cdot 2^2 + D_1(5-2) = 43 \\ 4\cdot 3^2 + D_1(5-3) = 48 \\ 4\cdot 4^2 + D_1(5-4) = 67 \end{array} \right\} = 67,$$

$$D_2(6) = \max_{\substack{x_2=0\\x_2=1\\x_2=2\\x_2=3\\x_2=4}} \left\{4x_2^2 + D_1(6-x_2)\right\} = \max \left\{ \begin{array}{l} 4\cdot 0^2 + D_1(6-0) = 48 \\ 4\cdot 1^2 + D_1(6-1) = 52 \\ 4\cdot 2^2 + D_1(6-2) = 64 \\ 4\cdot 3^2 + D_1(6-3) = 63 \\ 4\cdot 4^2 + D_1(6-4) = 76 \end{array} \right\} = 76,$$

$$D_2(7) = \max_{\substack{x_2=0\\x_2=1\\x_2=2\\x_2=3\\x_2=4}} \left\{4x_2^2 + D_1(7-x_2)\right\} = \max \left\{ \begin{array}{l} 4\cdot 0^2 + D_1(7-0) = 48 \\ 4\cdot 1^2 + D_1(7-1) = 52 \\ 4\cdot 2^2 + D_1(7-2) = 64 \\ 4\cdot 3^2 + D_1(7-3) = 84 \\ 4\cdot 4^2 + D_1(7-4) = 91 \end{array} \right\} = 91.$$

Kada se uzme u obzir mogućnost raspodele resursa na sva tri procesa, izraz za maksimalnu dobit u funkciji raspoloživog resursa je:

$$D_3(x_3) = \max_{0\le x_3\le 4} \left\{2x_3^2 + D_2(R-x_3)\right\}.$$

Na osnovu ovog izraza su izračunate vrednosti u koloni $D_3(x_3)$. U ovom slučaju su dobijene maksimalne vrednosti dobiti, pri čemu je uzeta u obzir optimalna dobit drugog i trećeg procesa.

$$D_3(0) = \max_{x_3=0} \left\{2x_3^2 + D_2(0-x_3)\right\} = \max \left\{2\cdot 0^2 + D_2(0-0)\right\} = 0,$$

$$D_3(1) = \max_{\substack{x_3=0\\x_3=1}} \left\{2x_3^2 + D_2(1-x_3)\right\} = \max \left\{ \begin{array}{l} 2\cdot 0^2 + D_2(1-0) = 4 \\ 2\cdot 1^2 + D_2(1-1) = 2 \end{array} \right\} = 4,$$

$$D_3(2) = \max_{\substack{x_2=0\\x_2=1\\x_2=2}} \left\{2x_3^2 + D_2(2-x_3)\right\} = \max \left\{ \begin{array}{l} 2\cdot 0^2 + D_2(2-0) = 12 \\ 2\cdot 1^2 + D_2(2-1) = 7 \\ 2\cdot 2^2 + D_2(2-2) = 16 \end{array} \right\} = 16,$$

$$D_3(3) = \max_{\substack{x_3=0\\x_3=1\\x_3=2\\x_3=3}} \left\{2x_3^2 + D_2(3-x_3)\right\} = \max \left\{ \begin{array}{l} 2\cdot 0^2 + D_2(3-0) = 27 \\ 2\cdot 1^2 + D_2(3-1) = 16 \\ 2\cdot 2^2 + D_2(3-2) = 19 \\ 2\cdot 3^2 + D_2(3-3) = 36 \end{array} \right\} = 36,$$

$$D_3(4) = \max_{\substack{x_3=0\\x_3=1\\x_3=2\\x_3=3\\x_3=4}} \left\{2x_3^2 + D_2(4-x_3)\right\} = \max \left\{ \begin{array}{l} 2\cdot 0^2 + D_2(4-0) = 48 \\ 2\cdot 1^2 + D_2(4-1) = 31 \\ 2\cdot 2^2 + D_2(4-2) = 28 \\ 2\cdot 3^2 + D_2(4-3) = 39 \\ 2\cdot 4^2 + D_2(4-4) = 64 \end{array} \right\} = 64,$$

$$D_3(5) = \max_{\substack{x_3=0\\x_3=1\\x_3=2\\x_3=3\\x_3=4}} \left\{2x_3^2 + D_2(5-x_3)\right\} = \max \left\{ \begin{array}{l} 2\cdot 0^2 + D_2(5-0) = 48 \\ 2\cdot 1^2 + D_2(5-1) = 52 \\ 2\cdot 2^2 + D_2(5-2) = 43 \\ 2\cdot 3^2 + D_2(5-3) = 48 \\ 2\cdot 4^2 + D_2(5-4) = 67 \end{array} \right\} = 67,$$

$$D_3(6) = \max_{\substack{x_3=0\\x_3=1\\x_3=2\\x_3=3\\x_3=4}} \left\{2x_3^2 + D_2(6-x_3)\right\} = \max \left\{ \begin{array}{l} 2\cdot 0^2 + D_2(6-0) = 48 \\ 2\cdot 1^2 + D_2(6-1) = 52 \\ 2\cdot 2^2 + D_2(6-2) = 64 \\ 2\cdot 3^2 + D_2(6-3) = 63 \\ 2\cdot 4^2 + D_2(6-4) = 76 \end{array} \right\} = 76,$$

$$D_3(7) = \max_{\substack{x_3=0\\x_3=1\\x_3=2\\x_3=3\\x_3=4}} \left\{2x_3^2 + D_2(7-x_3)\right\} = \max \left\{ \begin{array}{l} 2\cdot 0^2 + D_2(7-0) = 48 \\ 2\cdot 1^2 + D_2(7-1) = 52 \\ 2\cdot 2^2 + D_2(7-2) = 64 \\ 2\cdot 3^2 + D_2(7-3) = 84 \\ 2\cdot 4^2 + D_2(7-4) = 91 \end{array} \right\} = 91.$$

Maksimalna vrednost dobijena na osnovu treće funkcije predstavlja i rešenje problema, tako da je:

$$maxD(X) = D_3(7) = 91(n.j.).$$

Optimalna vrednost za treću promenljivu je $x_3 = 0$ ($D_3(7) = 91$). Vrednosti ostale dve promenljive se određuju is tabele, tako da je $x_2 = 4$, dok iz ograničenja sledi $x_1 = 3$. Ove vrednosti su i podvučene u sledećoj tabeli.

Pošto je u pitanju jednostavni slučaj distribucije jednorodnog resursa, problem se može rešiti direktnim poređenjem vrednosti funkcije dobiti, a u zavisnosti od mogućih stanja distribucije resursa, kao što je dato u tabeli.

x_1	0	1	2	**3**	4	4	4	4
$R - x_1$	7	6	5	4	3	3	3	3
x_2	4	4	4	**4**	3	3	3	3
$R - x_2$	3	3	3	3	4	4	4	4
x_3	3	2	1	**0**	0	0	0	0
$R - x_3$	4	5	6	7	7	7	7	7
$c_1x_1^2$	0	3	12	27	48	48	48	48
$c_2x_2^2$	64	64	64	64	36	36	36	36
$c_3x_3^2$	18	8	2	0	0	0	0	0
$\sum_{i=1}^{3} c_j x_j^2$	82	75	78	**91**	84	84	84	84

Tabela 2.1.4.

Optimalno rešenje je

$$x^* = \begin{bmatrix} x_1^* \\ x_2^* \\ x_3^* \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix},$$

dok je vrednost funkcije cilja

$$\max D(x) = D(x^*) = 3x_1^{*2} + 4x_2^{*2} + 2x_3^{*2} = 91.$$

2.2 Raspodela poslova na mašine

Problem raspodele različitih poslova na određen broj mašina može se definisati na sledeći način:

U nekom pogonu postoji n jednorodnih mašina koje mogu da obavljaju operacije (ili poslove), označene sa 1 i 2. Ako x mašina u toku jednog razmatranog vremenskog perioda izvršava operaciju 1, to će kompaniji doneti dobit $g(x)$ na kraju tog perioda. Medjutim, jedan deo mašina biće amortizovan, odnosno neće biti za dalju upotrebu tako da se u narednom periodu može računati na $a(x)$ mašina od x mašina koje

su obavljale operaciju 1. Na sličan način, ako y mašina u toku prvog razmatranog perioda izvršava operaciju 2, to će kompaniji doneti dobit $h(y)$ a ukupno $b(y)$ mašina biće i dalje raspoloživo za rad. U svakom vremenskom periodu svaka raspoloživa mašina obavlja jednu od operacija, tj nijedna mašina ne pauzira.

Potrebno je odrediti broj mašina x_i koje treba da obavljaju operaciju 1 u i-tom vremenskom periodu $i = 1, \ldots, N$ (preostale ispravne mašine obavljaju operaciju 2 u tom periodu), tako da ostvarena dobit posle N perioda bude maksimalna.

Rešenje: Za prvi period važe sledeće jednačine:

- za operaciju 1 se koristi x_1 mašina, dok za obavljanje operacije 2 preostaje $y_1 = n_1 - x_1$ ($n_1 = n$= ukupan broj raspoloživih mašina na početku);
- ukupna dobit u prvom periodu iznosi: $g(x_1) + h(y_1)$;
- broj upotrebljivih mašina na kraju prvog perioda (upotrebljivih za rad u drugom periodu) je: $n_2 = a(x_1) + b(y_1)$.

Za drugi period važi:

- broj upotrebljivih mašina je: $n_2 = a(x_1) + b(y_1)$;
- za operaciju 1 se koristi x_2 mašina (od raspoloživih n_2, dok za obavljanje operacije 2 preostaje $y_2 = n_2 - x_2$;
- ukupna dobit u drugom periodu iznosi: $g(x_2) + h(y_2)$;
- broj upotrebljivih mašina na kraju drugog perioda (upotrebljivih za rad u trećem periodu) je: $n_3 = a(x_2) + b(y_2)$.

Na isti način se mogu napisati izrazi do N-tog perioda. Matematički model problema može se iskazati na sledeći način:

$$\begin{aligned} \max F(x,y) \quad &= \sum_{i=1}^{N} [g(x_i) + h(y_i)] \\ \text{p.o.} \quad &x_i + y_i = n_i \\ &n_{i+1} = a(x_i) + b(y_i), \quad i = 1, 2, \ldots, N-1 \\ &0 \le x_i \le n_i, \quad i = 1, 2, \ldots, N. \end{aligned}$$

Kao i kod predhodnog problema, i ovde se u slučaju linearne funkcije cilja $F(x, y)$ i ograničenja problem svodi na problem linearnog programiranja. Izložimo sada rešenje korišćenjem tehnike DP.

Ako se za $f_i(n)$ obeleži maksimalna dobit posle i perioda, pri čemu je n broj raspoloživih mašina na početku prvog perioda, tada se rekurentne jednačine dinamičkog programiranja mogu napisati na osnovu sledećih pravila:

- ukoliko do kraja planskog perioda prestaje samo jedan period (ili se razmatra problem koji ima samo jedan period), tada je:

$$f_1(n) = \max_{0 \le x_1 \le n} \{g(x_1) + h(n - x_1)\};$$

- ukoliko je do kraja planskog perioda je preostalo k perioda, tada važi:

$$f_k(n) = \max_{0 \le x_k \le n} \{g(x_k) + h(n - x_k) + f_{k-1}[a(x_k) + b(n - x_k)]\}, \quad k > 1.$$

Optimalne vrednosti $x_1^*, \ldots, x_n^*$ se izračunavaju na isti način kao u predhodnom problemu.

Primer 2.2.1 Preduzeće raspolaže sa 100 visokoproduktivnih presa. U naredne tri godine ($N = 3$) preduzeće bi trebalo da izrađuje otpreske tipa A i tipa B, uz zamenu alata. U zavisnosti od raspodele mašina, u i-tom periodu se može ostvariti dobit:

$$g(x_i) + h(y_i) = 0.9x_i + 0.5y_i.$$

Amortizacija zavisi od dela koji se izrađuje na mašini (zbog težine, potrebne sile i slično) i iznosi 70% za mašine koje izrađuju otpresak tipa A i 20% za mašine koje izrađuju otpresak B, tako da broj raspoloživih mašina za naredni period iznosi:

$$n_{i+1} = 0.3x_i + 0.8y_i.$$

Matematički model problema:

$$\begin{aligned} \max F(x) = & \sum_{i=1}^{3} (0.9x_i + 0.5y_i) \\ \text{p.o.} \quad & x_i + y_i = n_i \\ & n_{i+1} = 0.3x_i + 0.8y_i \\ & 0 \le x_i \le n_i, \quad n_1 = n. \end{aligned}$$

- Ako se razmatra samo treći period (period u kome je do kraja planskog perioda ostao samo jedan, treći period), maksimalna dobit će iznositi:

$$f_1(n) = \max_{0 \le x_3 \le n_3} \{0.9x_3 + 0.5(n_3 - x_3)\}.$$

Maksimalna dobit se dobija za $x_3 = n_3$, i iznosi $0.9n_3$.

- U drugom koraku ($k = 2$) posmatra se maksimalna dobit za treći i drugi period, pa rekurentna jednačina glasi:

$$f_2(n) = \max_{0 \le x_2 \le n_2} \{0.9x_2 + 0.5(n_2 - x_2) + f_1(0.3x_2 + 0.8(n_2 - x_2))\}$$

ili

$$\begin{aligned} f_2(n) &= \max_{0 \le x_2 \le n_2} \{0.4x_2 + 0.5n_2 + 0.9(0.3x_2 + 0.8(n_2 - x_2))\} \\ &= \max_{0 \le x_2 \le n_2} \{1.22n_2 - 0.05x_2\} \\ &= 1.22n_2 \quad \text{za } x_2 = 0. \end{aligned}$$

Kada se uzmu u obzir sva tri perioda, maksimalna dobit će iznositi:

$$f_3(n) = \max_{0 \le x_1 \le n_1} \{0.9x_1 + 0.5(n_1 - x_1) + f_2(0.3x_1 + 0.8(n_1 - x_1))\}.$$

Na osnovu prethodnih razmatranja, odavde se dobija

$$f_3(n) = \max_{0\le x_1\le n_1} \{0.9x_1 + 0.5(n_1 - x_1) + 1.22\,(0.3x_1 + 0.8(n_1 - x_1))\}\,,$$

ili

$$\begin{aligned} f_3(n) &= \max_{0\le x_1\le n_1} \{0.4x_1 + 0.5n_1 + 1.22(0.8n_1 - 0.5x_1)\}\,, \\ &= \max_{0\le x_1\le n_1} \{1.476n_1 - 0.21x_1\} = 1.476n_1 = 147.6. \end{aligned}$$

Maksimalna vrednost za $f_3(n)$ se dobija za $x_1 = 0$.

Rešenje problema je:

$$\begin{aligned} \max\ F(x) &= f_3(100) = 147.6 \\ x_1 &= 0,\ y_1 = 100 \\ n_2 &= 0.3x_1 + 0.8y_1 = 80, \\ x_2 &= 0,\ y_2 = 80, \\ n_3 &= 0.3x_2 + 0.8y_2 = 64, \\ x_3 &= 64,\ y_3 = 0. \end{aligned}$$

2.3 Optimalna politika zamene opreme

Principi zamene opreme moraju biti u skladu sa produktivnošću opreme, troškovima korišćenja, kao i preostalom vrednošću opreme.

Neka je:

$D(t)$ - dobit koja se ostvaruje upotrebom opreme stare t godina;
$C(t)$ - godišnji troškovi održavanja opreme;
$V(t)$ - preostala vrednost opreme stare t godina;
C - nabavna cena nove opreme.

U razmatranom periodu od N godina (gde je N razmatrani period, život projekta, život proizvoda koji se izrađuju na razmatranoj opremi) potrebno je odrediti optimalni ciklus zamene opreme.

Rešenje: Funkcija cilja $f_N(t)$ predstavlja dobit u razmatranom periodu od N godina koja se ostvaruje eksploatacijom opreme stare t godina.

Da bi se postavila funkcionalna relacija, potrebno je naći vezu između karakterističnih veličina u toku dva susedna perioda.

Periodi na koje se ukupni period N godina rasčlanjuje, računaju se od kraja. Na početku bilo kog (k-tog) perioda raspolaže se opremom starom t godina i moguća su dva alternativna rešenja:

- zadržati staru opremu (staru t godina) i ostvariti dobit:

$$D(t) - C(t) + f_{k-1}(t+1);$$

- nabaviti novu opremu i ostvariti dobit:

$$V(t) - C + D(t) - C(t) + f_{k-1}(t).$$

Optimalna dobit od eksploatacije opreme u preostalih k perioda $f_k(t)$ je:

$$f_k(t) = \max \begin{cases} D(t) - C(t) + f_{k-1}(t+1) \\ V(t) - C + D(t) - C(t) + f_{k-1}(t). \end{cases}$$

Za $k = 1$, odnosno maksimalna dobit koja se ostvaruje u slučaju kada je preostao samo jedan period, korišćenjem opreme stare t godina (pri čemu je $t = 1, \ldots, N$), iznosi:

$$f_1(t) = \max \begin{cases} D(t) - C(t) \\ V(t) - C + D(t) - C(t). \end{cases} \quad , t = 1, \ldots, N.$$

2.4 Problem maksimalnog zbira

Poreklo zadatka: [25].

Neka je dat pravougaonik A sa M vrsta i N kolona, popunjen celim brojevima. Iz svakog polja je dozvoljeno preći samo na polje ispod ili na polje desno od tog polja. Potrebno je izabrati put od gornjeg levog polja do donjeg desnog polja, tako da zbir brojeva u poljima preko kojih se prolazi, bude maksimalan. Ispisati vrednost optimalnog puta, a zatim i put kao niz koordinata polja preko kojih se prolazi.

Rešenje: Generisanje svih mogućih puteva i pamćenje onog puta koji ima najveći zbir, nije dobra ideja, jer broj mogućih puteva raste eksponencijalnom brzinom sa porastom veličine pravougaonika.

Proverimo uslove za primenu dinamičkog programiranja, tj da svaki podproblem polaznog problema ima optimalnu podstrukturu. Neka je dat put P čija polja imaju najveći zbir. Tada svaki deo optimalnog puta spaja polazno i završno polje tog dela puta na optimalan način (inače polazni put ne bi bio optimalan). Pogodno je podprobleme formalno zadati na sledeći način: neka je $P(i, j)$ optimalni put od polja $(1, 1)$ do polja (i, j), a $B(i, j)$ neka je zbir koji se na tom putu postiže. Tada je potrebno da se pronađe put $P(M, N)$ i zbir $B(M, N)$. Već smo ustanovili da problem ima optimalnu podstrukturu. Do polja (I, J) možemo doći samo preko jednog od polja $(I, J-1)$ ili $(I-1, J)$. Zbir $B(I, J)$ je zato jednak jednom (većem) od $B(I, J-1)$ ili $B(I-1, J)$, uvećanom za $A(I, J)$:

$$B(I,J) = \begin{cases} A(1,1), & I = J = 1, \\ B(1, J-1) + A(1,J), & I = 1, J \geq 2, \\ B(I-1,1) + A(I,1), & J = 1, I \geq 2, \\ \max\{B(I,J-1), B(I-1,J)\} + A(I,J), & I \geq 2, J \geq 2. \end{cases}$$

Tada se put $P(I, J)$ dobija dodavanjem polja (I, J) na jedan od puteva $P(I-1, J)$ ili $P(I, J-1)$ koji daje veći zbir.

Prema tome, rešavanje polaznog problema se svodi na rešavanje dva podproblema istog tipa, i njihovo jednostavno kombinovanje - poređenje dva zbira i sabiranje većeg od ta dva zbira sa vrednošću poslednjeg polja. Za rešavanje svih netrivijalnih podproblema važi potpuno isto. Trivijalni problemi su nalaženje elemenata prve vrste i prve kolone matrice B. Kako do polja $(1, J)$ ili $(I, 1)$ postoji samo jedan put, treba samo sabrati polja na tom putu.

Podproblemi se i u ovom problemu preklapaju. Na primer, problem za polje sa koordinatama $(i-1, j-1)$ se pojavljuje u oba potproblema $(I, J-1)$ i $(I-1, J)$. Zato rekurzija nikako nije prihvatljiva za rešavanje globalnog problema. Većina podproblema se rekurzivnim rešavanjem dalje svodi na dva manja podproblema, pa bi rekurzijom ukupno bilo potrebno rešiti eksponencijalno mnogo podproblema umesto samo $M * N$, koliko ih ukupno ima različitih. Stoga ćemo rešiti redom sve različite podprobleme, to jest, koristićemo dinamičko programiranje.

Koordinate polja koja sačinjavaju optimalan put se generišu od poslednjeg polja ka prvom. Da bismo ih ispisali od prvog ka poslednjem, možemo koristiti stek, pomoću koga obrćemo redosled podataka. Međutim, prilikom poziva funkcija i procedura, operativni sistem već koristi stek za smeštanje podataka, pa je primena rekurzije jednostavniji i prirodan način da se ispišu koordinate polja u željenom redosledu.

Napomena: Program koji sledi je dobijen dinamičkim programiranjem, a rekurzija se samo koristi za ispisivanje rešenja. Procedura $ispisi(M, N)$ koja ispisuje put do polja (M, N), poziva se rekurzivno najviše onoliko puta koliko ima polja na putu od polja $(1, 1)$ do polja (M, N), dakle ukupno manje od $M + N$ puta, tako da se ona brzo izvršava. Rekurzija se može upotrebiti zbog toga što je hijerarhija u rekurzivnom pozivanju jednoznačno određena.

Implementacija:

```
program maksimalni_zbir_u_matrici;
  var a,s:array[1..30,1..30] of integer;
      i,j,m,n:integer;
  procedure ispis(i,j:integer);
    var k:integer;
    begin
      if(i>1) and (j>1) then
        begin
          if s[i-1,j]>s[i,j-1] then ispis(i-1,j)
          else ispis(i,j-1);
          writeln(i:3,j:3);
        end
      else if i=1 then  for k:=1 to j do writeln(i:3,k:3)
      else for k:=1 to i do writeln(k:3,j:3);
    end;

  begin
    readln(m,n);
    for i:=1 to m do
      begin
        for j:=1 to n do read(a[i,j]);   readln
      end;
    s[1,1]:=a[1,1];
    for j:=2 to n do s[1,j]:=a[1,j]+s[1,j-1];
    for i:=2 to m do s[i,1]:=a[i,1]+s[i-1,1];
    for i:=2 to m do
      for j:=2 to n do
        if s[i,j-1]<s[i-1,j] then  s[i,j]:=a[i,j]+s[i-1,j]
        else s[i,j]:=a[i,j]+s[i,j-1];
    writeln('Maksimalni zbir je ',s[m,n]);
```

```
    writeln('Postize se na sledeci nacin: ');    ispis(m,n);
  end.
```

Na primer, za matricu oblika

4	3	5	7	5
1	9	4	1	3
2	3	5	1	2
1	3	1	2	0
4	6	7	2	1

dobija se maksimalni zbir 38. Ovaj optimalni put se postiže korišćenjem sledećih polja:

1 1, 1 2, 2 2, 3 2, 4 2, 5 2, 5 3, 5 4, 5 5.

2.5 Problem maksimalnog zbira: još nekoliko varijanti

Problem maksimalnog zbira može se sresti u više srodnih varijanti. U svakoj varijanti podrazumevaćemo (ako se drugačije ne napomene) da je potrebno kretanjem kroz matricu postići najveći zbir i da se na svako polje može stati najviše jednom.

- Kretanjem dole i desno stići od polja $(1,1)$ do polja (m,n). Ovo je varijanta koja je razmatrana i rešena u predhodnom problemu.
- Kretanjem dole, desno i dole-desno stici od polja $(1,1)$ do polja (m,n). U ovoj varijanti elementi matrice B se računaju neznatno drugačije:

$$B(X,Y) = \max\{B(X,Y-1), B(X-1,Y), B(X-1,Y-1)\} + A(X,Y)$$

Detalje oko konstrukcije rekurentne formule u ovoj i narednim varijantama, kao i implementaciju rešenja u narednim varijantama prepuštamo čitaocu.

Implementacija:

```
program suma;
  uses wincrt;
  type matrica=array[1..50,1..50] of integer;
  var a,s:matrica;
      i,j,n,max:integer;
  procedure ispis(i,j:integer);
    var k:integer;
    begin
     if (i>1) and (j>1) and (i>=j) then
      if j<i then
        begin
          if s[i-1,j]>s[i,j-1] then
            if s[i-1,j]>s[i-1,j-1] then ispis(i-1,j)
                                   else ispis(i-1,j-1)
          else if s[i,j-1]>s[i-1,j-1] then ispis(i,j-1)
                                      else ispis(i-1,j-1);
          write('[',i,',',j,'], ');
        end
      else
        begin
           if s[i,j-1]>s[i-1,j-1] then ispis(i,j-1)
                                  else ispis(i-1,j-1);
```

```
          write('[',i,',',j,'], ');
        end
      else if (i>=1) and (j=1) then
         for k:=1 to i do write('[',k,',',1,'], ')
    end;
begin
  writeln('Unesi stranicu trougla');   readln(n);
  writeln('Unesi vrednost svakog clana');
  for i:=1 to n do
    begin
      for j:=1 to i do read(a[i,j]);
      readln;
    end;
  s[1,1]:=a[1,1];
  for i:=2 to n do s[i,1]:=s[i-1,1]+a[i,1];
  for i:=2 to n do
    for j:=2 to i do
      begin
        if j<i then
          begin
            max:=s[i-1,j-1];
            if s[i-1,j]>max then max:=s[i-1,j];
            if s[i,j-1]>max then max:=s[i,j-1];
          end
        else if j=i then
            if s[i,j-1]>=s[i-1,j-1] then max:=s[i,j-1]
                                    else max:=s[i-1,j-1];
        s[i,j]:=max+a[i,j];
      end;
  writeln(s[n,n]);
  ispis(n,n);
end.
```

• Kretanjem gore-desno, dole-desno i desno, stići od bilo kog polja $(x, 1)$ prve kolone, do bilo kog polja (y, n) poslednje kolone. Ovde se matrica B mora popunjavati po kolonama, za elemente prve kolone važi $B(X, 1) = A(X, 1)$, a za ostale:

$$B(X,Y) = \max\{B(X-1,Y-1), B(X,Y-1), B(X+1,Y-1)\} + A(X,Y)$$

pri čemu smatramo da su $B(X-1,0)$ i $B(X-1,M+1)$ jednaki minus beskonačno, tj. za prvi i poslednji element u koloni maksimum se bira od 2, a ne od 3 člana.

• Kretanjem na dole i desno stići od gornjeg levog do donjeg desnog polja matrica A, i pri tome stati na takve susedne brojeve $x_1, x_2, ..., x_k$ u matrici A ($x_1 = a_{11}, x_k = a_{mn}$), da se maksimizira ne njihov zbir, nego zbir apsolutnih razlika uzastopnih brojeva preko kojih se ide, dakle $\sum_{i=1}^{k-1} |x_{i+1} - x_i|$.

Lako je uvideti da za $X > 1$, $Y > 1$ važi relacija

$$B(X,Y) = \max \left\{ \begin{array}{l} B(X,Y-1) + |A(X,Y-1) - A(X,Y)| \\ B(X-1,Y) + |A(X-1,Y) - A(X,Y)| \end{array} \right\}$$

dok se elementi prve vrste i prve kolone izracunavaju prema formuli

$$B(1,1) = 0$$
$$B(1,Y) + B(1,Y-1) + |A(1,Y-1) - A(1,Y)|$$
$$B(X,1) + B(X-1,1) + |A(X-1,1) - A(X,1)|$$

Na osnovu vrednosti popunjene matrice B nije teško odrediti polja preko kojih se prelazi radi maksimiziranja traženog zbira.

• Potrebno je stići od bilo kojeg elementa prve kolone do bilo kog elementa poslednje kolone, krećući se gore, dole i desno. Matricu B popunjavamo po kolonama, tako da je B(X,Y) najveća vrednost koja se može dostići kretanjem kroz prvih Y kolona i zaustavljanjem u polju A(X,Y). Tada imamo:

$$B(X,1) = \max_{1 \le k \le M} \left\{ \sum_{p=k,X} a_{p,1} \right\}, 1 \le X \le M$$

gde p od k do X po potrebi (za $k > X$) ide i unazad, i

$$B(X,Y) = \max_{1 \le k \le M} \left\{ B(k,Y-1) + \sum_{p=k,X} a_{p,Y} \right\}, 1 \le X \le M, 2 \le\ Y \le N.$$

Ukupan broj operacija je u ovom primeru veći i proporcionalan je sa M^2N, a rezultata je određen maksimumom poslednje kolone matrice B.

Osim navedenih, mogu se naći i druge slične varijante ovog problema. Sve one se rešavaju na način vrlo blizak izloženom.

Brojevni trougao

Dat je trougao popunjen brojevima: u prvom redu je jedan broj, a u drugom dva, itd. do N-tog reda sa N brojeva. Ovaj trougao možemo predstaviti donje-trouganom matricom $A(X,Y)$. U sledećoj tabeli je prikazan "brojevni trougao".

```
7
3  8
8  1  0
2  7  4  4
4  5  2  6  5
```

Napisati program koji nalazi najveću sumu brojeva, po putu koji počinje na vrhu, a završava se na nekom polju baze.

- U svakom koraku možemo ići dole ili dijagonalno dole desno.
- Broj redova u trouglu je veći od 1, a najviše 100.
- Brojevi u trouglu su celi, u intervalu od 0 do 99.

Rešenje: Neka je $B(X,Y)$ maksimalna suma koju je moguće ostvariti kretanjem po brojevnom trouglu od polja $(1,1)$ do polja (X,Y). Važi rekurentna formula

$$B(X,Y) = \max\{B(X-1,Y-1), B(X-1,Y)\} + A(X,Y).$$

Implementacija:

```
program brojnitrougao;
  const maxn=100;
  var a,mat:array[1..maxn,1..maxn] of integer;
      poz,s,i,j,k,n:integer;
      ulaz,izlaz:text;

  procedure ulazpod;
    var i,j:integer;
    begin
      assign(ulaz,'c:\tp\zadaci\input.txt'); assign(izlaz,'c:tp\zadaci\output.txt');
      reset(ulaz); rewrite(izlaz);
      readln(ulaz,n);
      for i:=1 to n do
        for j:=1 to i do
          begin
            read(ulaz,mat[i,j]); a[i,j]:=0;
          end;
      a[1,1]:=mat[1,1];
      close(ulaz);
    end;

  procedure nadjizbir;
    var i,j:integer;
    begin
      for i:=2 to n do
        for j:=1 to i do
          begin
            if (j>1) and (a[i,j]<a[i-1,j-1]+mat[i,j]) then
              a[i,j]:=a[i-1,j-1]+mat[i,j];
            if (i>j) and (a[i,j]<a[i-1,j]+mat[i,j]) then
              a[i,j]:=a[i-1,j]+mat[i,j];
          end;
      s:=0;
      for j:=1 to n do
        if a[n,j]>s then s:=a[n,j];
    end;

  procedure stampaj;
    var i:integer;
    begin
      writeln(izlaz,'Najveci zbir je ',s);   close(izlaz);
    end;

  begin
    ulazpod; nadjizbir; stampaj;
  end.
```

Primer: Za sledeći sadržaj ulazne datoteke

```
5
7
3  8
8  1  0
2  7  7  4
4  5  2  6  5
```

dobija se maksimalni zbir 30, pri čemu je optimalni put

$$\{1,1\}, \quad \{2,1\}, \quad \{3,1\}, \quad \{4,2\}, \quad \{5,2\}.$$

Razmotrimo sada još jednu varijantu problema brojevnog trougla.

• Treba stići od broja na vrhu do bilo kog broja na osnovici trougla. Trougao se može smestitiu kvadratnu matricu na više načina. Neka to bude trougao iznad sopstvene dijagonale. Tako dobijemo ekvivalentnu postavku: u kvadratnoj matrici treba kretanjem na dole i desno stići od gornjeg levog ugla do bilo kog broja na sporednoj dijagonali. Zadatak se rešava tako što na isti način kao i ranije formiramo kvadratnu matricu B, ali je popunjavamo samo do sporedne dijagonale. Rešenje je određeno najvećim elementom na sporednoj dijagonali matrice B.

2.6 Najduži zajednički podniz

Poreklo zadatka: [25].

Niz A je podniz niza B, ako se precrtavanjem nekih elemenata niza B može dobiti niz A. Na primer (1,3,3,5) je podniz od (1,2,3,3,4,5), a nije podniz ni od (1,5,2,3,3,4) ni od (1,2,3,4,5).

Data su dva niza: P od M i Q od N elemenata. Naći niz najveće moguće dužine koji je podniz i za P i za Q.

Rešenje: Neka je $NZP(X,Y)$ najduži zajednićki podniz nizova P_X i Q_Y (P_X sadrži prvix X elemenata niza P, dok Q_Y sadrži prvix Y elemenata niza Q). U zadatku se traži $NZP(M,N)$. Ako je $p_M = q_N$, tada $NZP(M,N) = NZP(M-1,N-1) \cup \{p_M\}$ (p_M se dopisuje na kraj niza $NZP(M-1,N-1)$), dok je za $p_M \neq q_N$, $NZP(M,N)$ jednak dužem od $NZP(M-1,N)$, $NZP(M,N-1)$.

Ova relacija omogućava da se izračunaju sve vrednosti $NZP(M,N)$ redom (po vrstama ili po kolonama), znajući da je $NZP(X,0) = NZP(0,Y) = \emptyset$ (prazan niz). Dovoljno je pamtiti dužine najdužih zajedničkih podnizova u matrici B, a $NZP(X,Y)$ se lako rekonstruiše na osnovu matrice B:

$$\begin{aligned} B(X,0) &= B(0,Y) = 0 \\ B(X,Y) &= \begin{cases} B(X-1,Y-1)+1 & \text{ako je } P(X) = Q(Y) \\ \max\{B(X,Y-1), B(X,Y-1)\} & \text{ako je } P(X) \neq Q(Y) \end{cases} \end{aligned}$$

```
program NZP;
  type matrica=array[0..120,0..120]of integer;
  var b:matrica;
      p,q:array[1..120]of integer;
      i,j,m,n:integer;
      used:boolean;
      f:text;

  Function max(a,b:integer):integer;
    begin
      if a>b then max:=a
```

```
      else max:=b;
    end;
  procedure ispis(var b:matrica;i,j:integer);
    begin
      while (b[i-1,j]=b[i,j])and(i>1) do i:=i-1;
      while (b[i,j-1]=b[i,j])and(j>1) do j:=j-1;
      if b[i,j-1]>0 then ispis(b,i,j-1);
      write(f,q[i]:3);
    end;

begin
  assign(f,'input.dat');reset(f);  readln(f,m,n);
  for i:=1 to m do read(f,p[i]);  for i:=1 to n do read(f,q[i]);
  close(f);
  for i:=1 to m do b[i,0]:=0;  for i:=1 to n do b[0,j]:=0;
  for i:=1 to n do
    begin
      used:=false;
      for j:=1 to m do
        if (q[i]=p[j])and(b[0,j]=0)and not used then
          begin
            b[i,j]:=max(b[i-1,j],b[i,j-1])+1;  b[0,j]:=1;
            used:=true;
          end
        else
          b[i,j]:=max(b[i-1,j],b[i,j-1]);
    end;
  assign(f,'output.dat');  rewrite(f);  ispis(b,n,m);   close(f);
end.
```

Ukoliko bi se tražila samo dužina NZP, mogli bismo da uštedimo prostor, tako što umesto matrice B koristimo samo dva niza, koji bi igrali ulogu vrste iz B koja se trenutno formira i predhodne vrste (uz redosled popunjavanja unazad, dovoljan je samo jedan niz). Da bismo rekonstruisali NZP, neophodna nam je cela matrica B, ili dodatno vreme za ponovna računanja izgubljenih informacija.

2.7 Najjeftinija ispravka reči

Poreklo zadatka: [25].

Dozvoljene operacije nad stringom su: umetanje slova, brisanje jednog slova, izmena slova i brisanje svih slova do kraja stringa. Svaka od ovih operacija ima zadatu cenu. Potrebno je odrediti najmanju ukupnu cenu operacija kojima se od datog stringa A dobija dati string B.

Rešenje: Zadatak se rešava vrlo slično kao problem maksimalnog zbira i problem najdužeg zajedničkog podniza, pa je zbog toga detaljnije objašnjenje izostavljeno. Neka je $C(i,j)$ minimalna cena potrebna da se od stringa A_i dobije string B_j. Tada važi sledeća rekurentna formula:

$$C(i,j) = \begin{cases} C(i-1,j-1), & a_i = b_j \\ \min\{C(i-1,j-1)+cZam, C(i-1,j)+cBris, C(i,j-1) \\ +cUmet\} \cup \{C(k,j)+CBrisDoKraja \mid k=1,\ldots,i-1\}, & a_i \neq b_j \end{cases}$$

Pritom su $cZam, cBris, cUmet, cBrisDoKraja$ redom cene zamene slova, umetanja jednog slova, brisanja jednog slova i brisanja svih slova do kraja počev od nekog slova. Startne vrednosti su $C(i,0) = \min\{i \cdot cBris, cBrisDoKraja\}$ i $C(0,j) = j \cdot cUmet$.

Implementacija:

```
program Najeftinija_ispravka_reci;
 var  cena:array[0..50,0..50] of real;
      a,b:string;
      cUmet,cBris1,cZam,cBrisDoKraja:real;
      i,j,k,m,n:integer;
 function min3(a,b,c:real):real;
     var x:real;
      begin
        x:=a;
        if x>b then x:=b;
        if x>c then x:=c;
        min3:=x;
      end;
   begin
     write('Cena umetanja=');readln(cUmet);
     write('Cena brisanja jednog znaka='); readln(cBris1);
     write('Cena zamene='); readln(cZam);
     write('Cena brisanja do kraja stringa='); readln(cBrisDoKraja);
     write('a='); readln(a); m:=length(a);
     write('b='); readln(b); n:=length(b);
      cena[0,0]:=0;
      for i:=1 to m do
       begin
         cena[i,0]:=cena[i-1,0]+cBris1;
         if cena[i,0]>cBrisDoKraja then   cena[i,0]:=cBrisDoKraja;
       end;
      for j:=1 to n do  cena[0,j]:=cena[0,j-1]+cUmet;
      for i:=1 to m do
       for j:=1 to n do
         begin
          if a[i]=b[j] then  cena[i,j]:=cena[i-1,j-1]
           else
             cena[i,j]:=min3(cena[i-1,j-1]+cZam,cena[i,j-1]+cUmet,
                             cena[i-1,j]+cBris1);
          for k:=1 to i-1 do
           if cena[i,j]>cena[k,j]+cBrisDoKraja then
              cena[i,j]:=cena[k,j]+cBrisDoKraja;
         end;
      writeln('Najeftinija prepravka kosta ' ,cena[m,n]:10:3);
 end.
```

2.8 Najbrže stepenovanje

Poreklo zadatka: [25].

Dat je prirodan broj N i promenljiva K. Koristeći operacije množenja i stepenovanja,male zagrade i promenljivu K, napisati izraz koji je jednak K^N, a u kome učetvuje minimalan broj operacija. Množenje se smatra jednom operacijom,

a računanje Q-tog stepena smatra se za $Q-I$ operacija. Prilikom ispisivanja izraza stepenovanje označiti sa dve zvezdice.

Primer: za $N=5$, potrebne su tri operacije: $K^5=(K\cdot K)^2\cdot K$.

Rešenje: Neka se u optimalnom izrazu jednakom K^N (N> 1) poslednja izvršava operacija množenja.Tada je izraz oblika $T_1 \cdot T_2$, gde su T_1 i T_2 optimalni izrazi jednaki K^P i K^{N-P} za neko P.Broj operacija u izrazu jednakom K^N je tada $B(N) = B(P)+B(N-P)+1$.

Ako se u optimalnom izrazu jednakom K^N $(N>1)$ poslednja izvršava operacija stepenovanja, izraz je oblika T^R, gde je N deljivo sa R, a T je optimalan izraz jednak $K^{N/R}$,pa važi $B(N)=B(N/R)+R-1$.

Videli smo u čemu se ogleda optimalnost podstrukture problema. Prilikom rešavanja svih podproblema redom (za X od 1 do N), optimalan izraz jednak K^X naći ćemo tako što svaku moguću operaciju isprobamo kao poslednju, a za podizraze koje treba uvrstiti u izraz, koristimo ranije izračunata rešenja. Za dobijanje prvog stepena K^I, potrebno je 0 operacija, pa imamo:

$$\begin{aligned} B(1) &= 0, \\ B(X) &= \min \begin{cases} \min\limits_{1\le P<X}\{B(P)+B(X-P)+1\} \\ \min\limits_{R\neq 1, R|X}\{B(X/R)+R-1\} \end{cases} , 1<X\le N \end{aligned}$$

Da bi se rekonstruisao izraz sa najmanje operacija, koji je jednak k^N, dovoljno je pri rešavanju svakog podproblema zapamtiti poslednju operaciju. Pored niza B, koji pamti najmanji broj operacija, pamtićemo i niz C iste dužine. Stavićemo $C(X)=P$, ako se K^X najjejftinije dobija kao proizvod optimalnih izraza jednakih K^P i K^{X-P} redom, ili $C(X)=-R$ ako K^X najjejftinije dobijamo kao R-ti stepen optimalnog izraza jednakog $K^{X/R}$.

```
program najstep;
  var B,C:array[1..1000] of integer;
      p,r,m,N:integer;

  procedure ispis(n:integer);
   begin
     if C[n]=0 then write('K')
     else if C[n]<0 then
       begin
         write('('); ispis(n div (-C[n]));  write(')**',-C[n]);
       end
     else begin
            ispis(C[n]); write('*'); ispis(n-C[n]);
          end;
   end;

  begin
    write('N= ? '); readln(N);
    B[1]:=0; C[1]:=0;
    for m:=2 to N do
      begin
        B[m]:=B[1]+B[m-1]+1;       C[m]:=1;
```

```
      for p:=2 to m-1 do
        if B[p]+B[m-p]+1<B[m] then
          begin
             B[m]:=B[p]+B[m-p]+1;    C[m]:=p;
          end;
      for r:= 2 to m div 2 do
        if (m mod r) = 0 then
           begin
             if B[m div r]+r-1 < B[m] then
               begin
                 B[m]:=B[m div r]+r-1; C[m]:=-r;
               end;
           end;
    end;
  writeln('Potreban broj operacija je ',B[N]);
  write('Izraz je: '); ispis(N); writeln;
end.
```

2.9 Red za karte

Poreklo zadatka: [25].

N ljudi stoji pred ulazom blagajne i čeka da kupi karte. $K-$tom čoveku u redu treba t_k vremena da kupi kartu $k = 1, \ldots, N$. Svaki čovek može se udružiti sa sledećim u redu. Vreme potrebno da $k-$ti i $k+1-$vi čovek kupe karte ako se udruže, iznosi p_k, $k = 1, \ldots, N-1$. Kupovina time može a i ne mora da se ubrza. Odrediti takav način udruživanja,da ukupno vreme potrebno da svih N ljudi kupi po kartu,bude minimalno. Ulazni podaci su broj N i nizovi T i P, trebalo bi ispisati redne brojeve onih ljudi koji se udružuju sa sledećim u redu.

Rešenje: Neka je niz A takav da $a_k = t_k + t_{k+1} - p_k$, tj. a_k je ušteda u vremenu ako se udruže k–ti i $k+1-$vi čovek u redu. Primetimo da ne mora biti $p_k < t_k + t_{k+1}$,pa elementi niza A mogu biti i negativni. Kako se k–ti čovek ne može udružiti i sa $k+1-$vim istovremeno, ne možemo istovremeno uštede a_{k-1} i a_k. Preciznije, potrebno je da se odredi podniz niza A, koji ima najveći zbir i u kome nema susednih elemenata, a to je zadatak koji je upravo rešen.

Zadatak se može rešiti i bez svođenja na prethodni: potrebno je formirati niz B, gde je $B(k)$ najmanje potrebno vreme (umesto najveće moguće uštede ,što je učinjeno u prethodnom rešenju) da prvih k ljudi kupi karte. Lako se dobija

$$B(0) = 0, B(I) = a_j, B(X) = \min\{B(X-1) + t_x, B(X-2) + p_{x-1}\}.$$

Nakon što se formira niz B, ispisati redne brojeve onih ljudi koji su se udružili sa sldećim u redu. To se, kao i u drugim sličnim zadacima, najlakše radi pomoću (brze) rekurzivne procedure. Ukoliko je prihvatljiv redosled unazad, može se koristiti i ciklus.

```
program Red_Za_Karte;
 var t,p,a,b:array[0..1000]of integer;
     f:text;
     n,i:integer;
```

```
procedure ispis(n:integer);
 begin
   if n>0 then
     if b[n]=b[n-1] then ispis(n-1)
     else
       begin
         ispis(n-2);    write(f,n:4);
       end;
 end;
function min(a,b:integer):integer;
 begin
   if a>b then min:=a else min:=b;
 end;
begin
 assign(f,'input.dat'); reset(f);  readln(f,n);
 for i:=1 to n do read(f,t[i]);  for i:=1 to n-1 do read(f,p[i]);
 close(f);
 for i:=1 to n-1 do a[i]:=t[i]+t[i+1]-p[i];
 b[0]:=0;   b[1]:=a[1];
 for i:=1 to n do b[i]:=min(b[i-1]+t[i], b[i-2]+p[i-1]);
 assign(f,'output.dat'); rewrite(f); ispis(n);  close(f);
end.
```

2.10 Raspoređivanje mašina na dva posla

Poreklo zadatka: [25].

Dato je N mašina koje mogu obabljati dva posla. Na mašinama postoji jedan jeftin deo (zanemarljive cene) koji se lako kvari, ali rezervnih delova trenutno nema. Ako p mašina obavlja prvi posao, dobit od toga je $D_1(p)$, i pri tome $R_1(p)$ mašina može da nastavi da radi. Slično, ako q mašina obavlja drugi posao, dobit je $D_2(q)$,i ostaje $R_2(q)$ ispravnih mašina. Potrebno je tokom M uzastopnih perioda raspoređivati mašine na ova dva posla, tako da se ostvari maksimalna dobit.

Rešenje: Pretpostavimo da je poznata optimalna raspodela mašina na poslove tokom prvih K perioda. Nije jasno kako se taj podproblem može upotrebiti u rešavanju problema za $K+1$ period. Preciznije, može se dogoditi da se u prethodnim periodima isplati angažovati mašine na drugačiji način, tako da se ostvari manja dobit, ali da ostane više ispravnih mašina za novi period proizvodnje. Tako bi se (možda) mogla nadmašiti dobit bazirana na maksimumu iz prvih K perioda i optimalnom korišćenju preostalih ispravnih mašina u novom periodu. Kako onda rešiti zadatak?

Pokušajmo pre svega da definišemo hijerarhiju problema sa optimalnom podstrukturom. Neka je dato optimalno rešenje problema. Jedini period za koji znamo broj mašina koje učestvuju u njemu je prvi period. Neka je u optimalnom rešenju u prvom periodu p mašina raspoređeno na prvi posao, a $N-p$ mašina na drugi posao. Tada je tokom narednih $M-1$ perioda preostalih $R_1(p)+R_2(N-p)$ mašina moralo biti raspoređeno takođe na optimalan način. Prema tome, veličina problema treba da se zada brojem ispravnih mašina i brojem perioda,a problem za K perioda odnosi se na poslednjih K perioda,a ne na prvih K.

Neka je maksimalna dobit od X mašina u (poslednjih) Y perioda označena sa

$B(X,Y)$. Tada je:

$$\begin{aligned} B(X,1) &= \max_{0\le p\le X}\{D_1(p)+D_2(X-p)\} \\ B(X,Y) &= \max_{o\le p\le X}\{D_1(p)+D_2(X-p)+B(R_1(p)+R_2(X-p),Y-1)\} \end{aligned}$$

U matrici C kao $C(X,Y)$ pamtimo ono p za koje se dostiže maksimum u B. Na osnovu matrice C lako se rekonstruiše raspodela mašina po periodima.

Implementaciju ovog rešenja prepuštamo čitaocu.

3 Problem ranca

Jedan od najpoznatijih problema dinamičkog programiranja jeste **problem ranca** (knapsack problem). Problem ima nekoliko poznatih varijanti. Problem ranca je klasični problem dinamičkog programiranja, ali je zbog njegove važnosti i značaja ovom problemu posvećen ceo jedan odeljak ove glave. U nastavku ćemo izložiti više varijatni ovog problema i svaku detaljno analizirati. Za neke od varijanti dajemo i implementaciju u programskom jeziku PASCAL.

3.1 Prva varijanta problema ranca

Poreklo formulacije problema: [25].

Provalnik sa rancem u koji može da stane N zapreminskih jedinica, upao je u prostoriju u kojoj se čuvaju vredni predmeti. U prostoriji ima ukupno M tipova predmeta, pri čemu je svaki tip predmeta raspoloživ u vrlo velikoj količini (više nego što može da stane u ranac). Za svaki tip predmeta poznata je njegova vrednost $V(k)$ i njegova zapremina $Z(k)$, $k = 1,\ldots,M$. Sve navedene veličine su celobrojne. Provalnik želi da napuni ranac najvrednijim sadržajem. Potrebno je odrediti predmete koje treba staviti u ranac, i njihovu zbirnu vrednost.

Da bi se sagledala suštinu problema, uočimo nekoliko situacija.

- Ranac koji je optimalno popunjen ne mora biti popunjen do vrha, ili preciznije, zbir zapremina uzetih predmeta ne mora biti jednak zapremini ranca. Bitno je da taj zbir nije veći od zapremine ranca, a da u isto vreme zbir vrednosti tih predmeta bude maksimalan. Na primer, ako je $N = 7, M = 3, V = (3,4,8), Z = (3,4,5)$, ranac možemo popuniti do vrha tako što u njega stavimo prvi i drugi predmet jer je $z_1 + z_2 = 3 + 4 = 7 = N$. Međutim, takvo popunjavanje nije optimalno, jer je njegova vrednost $v_1 + v_2 = 3 + 4 = 7$, dok stavljanjem samo trećeg predmeta u ranac, dobijamo ranac vrednijeg sadržaja $v_3 = 8$, pri čemu ranac nije popunjen do vrha ($z_3 = 5 < 7 = N$).

- Na osnovu predhodnog primera, može se steći pogrešan utisak da se do rešenja dolazi tako što se u svakom koraku odredi takvo k, među predmetima koji nisu upotrebljeni, za koje je količnik $V(k)/Z(k)$ najveći, pa se ranac puni predmetom čiji je indeks k. Ovaj pristup nije ispravan, što ćemo dokazati na jednostavnom primeru: neka je $N = 7$, $M = 3$, $V = (3,4,6)$, $Z = (3,4,5)$. Navedena

ideja (grabljivi izbor) nalaže da se odabere treći predmet, kao najvredniji po jedinici zapremine. Time bi u ranac bio stavljen samo jedan od predmeta trećeg tipa (ubuduće: predmet broj 3), a vrednost plena bi bila jednaka 6. Lako je videti da izbor po jednog predmeta 1 i 2 daje plen vrednosti 7, što je bolje (i optimalno) rešenje. Napomenimo samo da bi ideja grabljivog izbora vodila ka rešenju da ne moraju da se uzimaju celi predmeti i da je vrednost dela nekog predmeta srazmerna veličini tog dela.

Dokazaćemo, koristeći se indukcijom, da se svako celobrojno q, $0 \leq q \leq N$, može naći popunjavanje ranca kapaciteta q. Za $q = 0$ optimalno resenje je prazan ranac. Neka su poznata rešenja za sve rance kapaciteta $i < q$, i neka je $B(q)$ zbirna vrednost koja odgovara uzetim predmetima za ranac kapaciteta q, a $C(q)$ niz rednih brojeva predmeta stavljenih u ranac kapaciteta q pri optimalnom izboru. Svaki od M tipova predmeta koji može da stane u prazan ranac kapaciteta q, isprobamo kao poslednji izabrani za ranac kapaciteta q. Ostatak ranca u svakom od ovih slučajeva popunimo na optimalan način, što na osnovu induktivne hipoteze može uraditi. Najveća od svih dobijenih vrednosti ranca kapaciteta q biće optimalna. Ova činjenica sledi direktno na osnovu optimalnosti podstrukture, jer jedan predmet mora biti poslednji, a mi smo svaki isprobali kao poslednji. Prema rečenom, rešenje za ranac kapaciteta q je

$$B(q) = \max_{\substack{1 \leq i \leq M \\ Z(i) \leq q}} \{B(q - z(i)) + V(i)\}.$$

Ako je $B(q) = B(q - z(k)) + V(k)$, tada je

$$C(q) = C(q - Z(k)) \cup \{k\},$$

što označava da k-ti predmet ostvaruje maksimalnu vrednost $B(q)$. Primetimo da ne moramo pamtiti ceo skup $C(q)$ za svaki kapacitet ranca q, dovoljno je kao član niza $C(q)$ zapamtiti poslednji dodati element $x(q)$. Svi elementi se tada mogu naći redom (od poslednjeg ka prvom), i to su:

$$a = C(N),\ b = C(N - Z(a)),\ c = C(N - Z(a - Z(b)),\ \ldots$$

itd. dok se ne dobiju svi predmeti iz ranca.

Analizirajući problem, mozemo primetiti sledeće.

Teorema 3.1.1 *Ako je za optimalno popunjavanje ranca do kapaciteta q izabrani predmet sa rednim brojem i, onda predhodno izabrani predmet na optimalan način popunjava ranac kapaciteta $q - z(i)$.*

Dokaz. Tvrđenje se lako dokazuje svođenjem na kontradikciju. Označimo sa $B(q)$, $q = 0, \ldots, N$ optimalna popunjavanja ranca do kapaciteta q. Neka je

$$B(q) = \max\{B(q - z(i)) + V(i) | z_i \leq q, i = 1, \ldots, M\} = B(q - z(k)) + V(k).$$

Dokažimo da je popunjavanje do kapaciteta $q - z(k)$ optimalno. Pretpostavimo da ranac kapaciteta $q - z(k)$ može bolje da se popuni, za neko i. Neka je vrednost tog

popunjavanja jednaka $R(q - z(k)) > B(q - z(k))$. Popunimo na isti način prvih $q-z(k)$ jedinica zapremine ranca veličine N, i dodamo k-ti predmet. Time dobijamo popunjavanje celog ranca, koje je bolje od polaznog, jer je njegova vrednost

$$R(q - z(k)) + V(k) > B(q - z(k)) + V(k) = B(q).$$

To je kontradikcija, jer je po pretpostavci $B(q)$ maksimalno. □

Prema tome, **optimalno rešenje problema sadrži u sebi optimalna rešenja podproblema istog tipa, sadržanih u glavnom problemu.** Uobičajeno je da se u ovakvom slučaju kaže da problem ima optimalnu podstrukturu. Navedeno svojstvo se naziva i svojstvom Belmana, po autoru metoda dinamičkog programiranja. Ovo je ključna osobina dinamičkog programiranja. Zahvaljujući ovoj osobini, možemo doći do optimalnog rešenja problema, kombinujući optimalna rešenja podproblema.

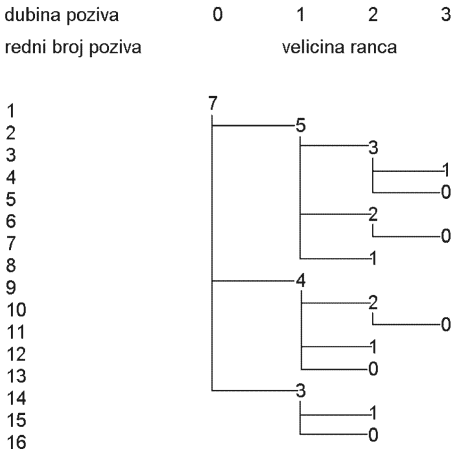
Program za prvu varijantu problema ranca u jeziku PASCAL:

```
program ranac;
  var B,C:array[0..1000] of integer;
      V,Z:array[1..10] of integer;
      N,M,q,i:integer;
  begin
   write('N = ? '); readln(N);    write('M = ? '); readln(M);
   writeln('Vrednosti i zapremine, redom? ');
   for i:=1 to M do read(V[i], Z[i]);
   B[0]:=0;   C[0]:=0;
   for q:= 1 to N do
     begin
       B[q]:=0; C[q]:=0;
       for i:=1 to M do
         if (Z[i]<=q) and (B[q-Z[i]]+V[i]>B[q]) then
           begin
             B[q]:=B[q-Z[i]]+V[i]; C[q]:=i;
           end;
     end;
   writeln('Optimalna vrednost sadrzaja u rancu je: ',B[N]);
   writeln('Redni brojevi izabranih predmeta: ');
   i:=N;
   while(C[i]>0) do
     begin
       writeln(C[i]); i:=i-Z[C[i]];
     end;
   writeln;
 end.
```

Zadatak se može resiti upotrebom rekurzivne procedure $napuni(q, B, C)$, koja za ranac kapaciteta q nalazi njegovu optimalnu vrednost B i redni broj poslednjeg dodatog predmeta C. Smatraćemo da su nizovi V i Z, kao i broj tipova predmeta M globalne veličine. Iz glavnog programa bi se pozivala procedura $napuni(N, B, C)$, a svaki poziv ove procedure može prouzrokovati M novih rekurzivnih poziva. Na primer za $N = 1000, M = 10$ i predmete zapremine od $Zmin = 5$ do $Zmax = 10$, bilo bi između $M^{N/Zmax} = 10^{100}$ i $M^{N/Zmin} = 10^{200}$ rekurzivnih poziva procedure,

i to samo na poslednjem nivou rekurzije. Kada bi svaki poziv trajao nanosekundu ($10^{-9}s$) trebalo bi vise od $10^{91}s > 10^{83}$ godina, a starost kompletne vasione procenjuje se na manje od 10^{12} godina [25]!

Umesto rekurzivno, probleme možemo rešavati i redom od $q = 1$ do $q = N$, popunjavajući pri tome nizove B i C dobijenim vrednostima. Za svako q, potrebno je M prolazaka kroz ciklus da se odredi poslednji izabrani element i najveća vrednost. To znači da je ukupan broj operacija jednak MN. U gornjem primeru to bi bilo deset hiljada ciklusa od po par računarskih koraka, što se na današnjim računarima obavlja za znatno manje od jedne sekunde. Odakle ovako drastična razlika u efikasnosti ponuđenih rešenja? Ispitajmo detaljnije na manjem primeru kako radi rekurzivni algoritam. Neka je $N = 7,\ M = 3,\ Z = (2, 3, 4)$. Vrednosti predmeta nisu od značaja za praćenje toka rekurzivnih poziva. Na sledećoj slici je prikazana hijerarhija rekurzivnih poziva procedure *napuni* u obliku drveta. Čvorovi drveta navedeni su po redosledu nastajanja, tj. po redosledu pozivanja primeraka procedure *napuni*, predstavljenih tim čvorovima. Oznake čvorova predstavljaju vrednosti argumenta q, tj. veličinu ranca koji treba popuniti.



Slika 3.1.1. hijerarhija rekurzivnih poziva procedure *napuni*.

Kao što vidimo, prilikom prolaska po drvetu rekurzivnih poziva, jedan isti podproblem (popunjavanje ranca iste velicine q susreće se više puta, i svaki put se iznova rešava. Drugim rećima, **podproblemi imaju zajedničke podprobleme, odnosno delimično se preklapaju.** Pri tome broj ponovljenih rešavanja po pravilu raste eksponencijalno sa povećanjem dimenzije polaznog problema. Preklapanje podproblema je druga osobina koja opravdava primenu dinamičkog programiranja. Ova osobina nije neophodna da bi se dinamičko programiranje primenilo, ali bez preklapanja podproblema dinamičko programiranje gubi svoju prednost u brzini u odnosu na rekurziju, jer se tada i rekurzijom svaki podproblem kreira i rešava samo jednom. Štaviše, kada nema preklapanja podproblema, u nekim problemima se događa da se rekurzivno rešenje dobije brže i/ili da se pri tome troši znatno manje memorijskog prostora.

Stroga matematička formulacija problema ranca se može naći u [7].

Problem ranca se može posmatrati kao poseban slučaj linearnog celobrojnog programiranja:

$$\begin{aligned} \max \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n, \\ \text{p.o.} \quad & a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b, \\ & x_1, x_2, \ldots, x_n \geq 0,\ x_1, x_2, ..., x_n \in Z. \end{aligned}$$

Pri tome su $a_1, a_2, \ldots, a_n$ prirodni brojevi a b ceo broj. Postoji više metoda za rešavanja problema, ranca. Ovde ćemo se detaljnije pozabaviti rešavanjem problema ranca koje je bazirano na ideji dinamičkog programiranja. U razlaganju problema na etape, ključnu ulogu igra definicija sledeće funkcije za cele brojeve $k \geq 1$ i $y \geq 0$:

$$F_k(y) = \max\{c_1x_1 + \cdots + c_kx_k | a_1x_1 + \cdots + a_kx_k \leq y, x_1, \ldots, x_k \geq 0, x_1, \ldots, x_k \in Z\}.$$

Za ovu funkciju, moguće je dokazati više rekurentnih relacija:

1.

$$F_1(y) = c_1\lfloor y/a_1 \rfloor$$

$$F_k(y) = \max\left\{c_kx_k + F_{k-1}(y - a_kx_k) | x_k \in \{0, 1, ..., \lfloor y/c_k \rfloor\} \text{ za } k \geq 2\right\}.$$

Rekurzija je posledica logičke alternative da su u optimalnom rešenju moguće vrednosti promenljive x_k upravo $0, 1, \ldots, \lfloor y/a_k \rfloor$ a posle njenog fiksiranja preostaje optimalno punjenje ranca zapremine $y - a_kx_k$ ostalim "predmetima".

2.

$$F_k(y) = \max\{F_{k-1}(y), F_k(y - a_k) + c_k\},$$

ukoliko se za $y < 0$ funkcija $F_k(y)$ dodefiniše pomoću $F_k(y) = -\infty$. Rekurzija je posledica logičke alternative: za k-tu koordinatu optimalnog rešenja važi ili $x_k = 0$ (tada je $F_k(y) = F_{k-1}(y)$) ili $x_k \geq 1$ (tada je $F_k(y) = F_k(y - a_k) + c_k$). Naravno, optimalna vrednost je uvek jednaka većoj od ovih vrednosti.

3.

$$F_n(y) = 0 \text{ za } 0 \leq y \leq \min\{a_1, \ldots, a_n\}$$

a za $y \geq \min\{a_1, \ldots, a_n\}$ je

$$F_n(y) = \max\{c_k + F_n(y - a_k) | k = 1, \ldots, n\},$$

ukoliko se dodefiniše $F_n(y) = -\infty$ za $y < 0$. Rekurzija je posledica logičke alternative da, ukoliko nula nije optimalno rešenje, bar jedna njegova koordinata, recimo x_k, je veća ili jednaka od 1 (tada je $F_n(y) = c_k + F_n(y - a_k)$).

Svaka od ovih rekurzija može da posluži za izračunavanje optimalne vrednosti $F_n(b)$ problema kao i odgovarajućeg optimalnog rešenja.

Pri korišćenju prve rekurzije, trebalo bi nepoznatu vrednost $F_n(b)$ svoditi na niže nivoe sve do poznatih vrednosti $F(y)$ a zatim se vraćati unazad. Pri tome se svi rezultati moraju pamtiti. Za određivanje optimalnog rešenja trebalo bi još zapamtiti gde se u pojedinim izračunavanjima dostiže maksimum.

Primer 3.1.1 *Primenom prve rekurzivne formule rešiti problem*

$$\begin{array}{ll} \max & 3x_1 + 4x_2 + 5x_3 + 2x_4, \\ \text{p.o.} & 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 9, \\ & x_l, x_2, x_3, x_4 \geq 0,\ x_1, x_2, x_3, x_4 \in Z. \end{array}$$

Računamo redom:

$$F_4(9) = \max\{2x_4 + F_3(9 - 5x_4) | x_4 \in 0, 1\} = \max\{F_3(9), 2 + F_3(4)\}$$

$$\begin{array}{rcl} F_3(9) & = & \max\{F_2(9), 5 + F_2(5)\} \\ F_3(4) & = & \max F_2(4), 5 + F_2(0) \\ F_2(9) & = & \max\{F_1(9), 4 + F_1(6)\} \\ F_2(5) & = & \max\{F_1(5), 4 + F_1(2)\} \\ F_2(4) & = & max\{F_1(4), 4 + F_1(1)\} \\ F_2(0) & = & 0 \\ F_1(y) & = & 3\lfloor\frac{y}{2}\rfloor. \end{array}$$

Postupkom unazad nalazimo:

$$\begin{array}{rcl} F_2(4) & = & F_1(4) == 6 \\ F_2(5) & = & 4 + F_1(2) = 7 \\ F_2(9) & = & 4 + F_1(6) = 13 \\ F_3(4) & = & F_2(4) = 6 \\ F_3(9) & = & F_2(9) = 13 \\ F_4(9) & = & F_3(9) = 13. \end{array}$$

Optimalno rešenje rekonstruišemo na sledeći način: Iz $F_4(9) = F_3(9)$ *sledi* $x_4^* = 0$. *Iz* $F_3(9) = F_2(9)$ *sledi* $x_3^* = 0$. *Iz* $F_2(9) = 4 + F_1(6)$ *sledi* $x_2^* = 1$. *Iz* $F_1(6) = 9$ *sledi* $x_1^* = 3$.

Druga rekurzija je pogodnija za kompjutersku obradu. Računanje ide unapred, redom po vrstama

$$\begin{array}{llll} F_1(1) & F_1(2) & \cdots & F_1(b) \\ F_2(1) & F2(2) & \cdots & F_2(b) \\ \vdots & & & \\ F_n(1) & F_n(2) & \cdots & F_n(b). \end{array}$$

Pri tome se pamte samo dve uzastopne vrste. Za računanje optimalnog rešenja može se uvesti prateći indeks $i_k(y)$ veličine $F_k(y)$, jednak najvećem indeksu j takvom da je j–ta promenljiva optimalnog rešenja u $F_k(y)$ pozitivna. Ukoliko je nula optimalno rešenje, ovaj indeks se definiše kao nula. Za ove indekse vredi rekurzija

$$i_k(y) = \begin{cases} i_{k-1}(y), & c_k + F_k(y) < F_{k-1}(y) \\ k, & c_k + F_k(y) \geq F_{k-1}(y). \end{cases}$$

Znajući $i_n(y)$ za $y = 0, 1, \ldots, b$, lako detektujemo optimalno rešenje iz smisla indeksa i vrednosti $i_n(b), i_n(b - a_{i_n(b)}), \ldots$.

Primer 3.1.2 *Rešiti zadatak iz Primera 3.1.1 drugom rekurzijom. Popunjavamo redom po vrstama matrice* $F_k(y)$ *i* $i_k(y)$ *za* $k = 1, 2, 3, 4$; $y = 1, 2, ..., 9$.

0	3	3	6	6	9	9	12	12
0	3	4	6	7	9	10	12	13
0	3	4	6	8	9	10	12	13
0	3	4	6	8	9	10	12	13
0	1	1	1	1	1	1	1	1
0	1	2	1	2	1	2	1	2
0	1	1	1	3	1	2	1	2
0	1	1	1	3	1	2	1	2

Primer popunjavanja: $F_2(9) = \max\{F_2(9), F_2(6) + 4\} = \max\{12, 13\} = 13$, $i_2(9) = 2$. *Rekonstrukcija optimalnog rešenja: Iz* $i_4(9) = 2$ *sledi da je u optimalnom rešenju* $x_3 = x_4 = 0$ *i* $x_2 \geq 1$. *Iz* $i_4(9-a_1) = i_4(6) = 1$ *sledi da je u optimalnom rešenju* $x_1 \geq 1$ *i* $x_2 = 1$. *Iz* $i_4(6-a_1) = i_4(4) = 1$ *sledi* $x_1 \geq 2$, *a iz* $i_4(4 - a_1) = i_4(2) = 1$, *sledi* $x_1 \geq 3$. *Zbog* $i_4(2 - a_1) = i_4(0) = 0$, *sledi* $x_1 = 3, x_2 = 1, x_3 = x_4 = 0$.

Treća rekurzija zaliteva najmanje izračunavanja. Kako u optimalnom rešenju može biti više pozitivnih koordinata, na desnoj strani formule mogu da postoje suvišni clanovi. Za dovoljno veliko b oni se mogu ispustiti.

Teorema 3.1.2 *Pretpostavimo da su promenljive u problemu ranca uređene tako da važi*

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \cdots \geq \frac{c_n}{a_n}.$$

Ako je $b > a(a_1 - 1)$, *pri čemu je* $a = \max\{a_1, \ldots, a_n\}$, *tada*

$$F_n(b) = F_n(b - a_1) + c_1.$$

Dokaz. Dovoljno je pokazati da postoji optimalno rešenje sa pozitivnom prvom koordinatom. Ako je $a_1 = 1$, optimalno rešenje linearne relaksacije, a time i problema, jednako je $x_1 = b, x_2 = \cdots = x_n = 0$, pa tvrđenje važi. Ako je $a_1 \geq 2$, uvedimo izravnavajući promenljivu x_{n+1} i posmatrajmo ekvivalentni problem

$$\begin{aligned} \min \quad & -c_1x_1 - \cdots - c_nx_n, \\ a_1x_1 + \cdots + a_nx_n + x_{n+1} \; &= \; b, \; x_1, x_2, ., x_n > 0, x_1, x_2, \ldots, x_n \in Z. \end{aligned}$$

Kako je

$$x_1 = \frac{b - a_2x_2 - \cdots - a_nx_n - x_{n+1}}{a_1},$$

njemu odgovarajući asimptotski problem sa bazičnom podmatricom $A_B = (a_1)$. izgleda

$$\min \sum_{j=2}^{n} \left(\frac{c_1}{a_1} a_j - c_j \right) x_j,$$

$$a_2x_2 + a_3x_3 + \cdots + a_nx_n + x_{n+1} = b (\text{mod } a_1),$$

$$x_2, \ldots, x_{n+1} \geq 0, x_2, ..., x_{n+1} \in Z.$$

Ovaj problem je ekvivalentan problemu najkraćeg puta na grafu koji ima a_1 čvorova. Kako najkraći put ima najviše $a_1 - 1$ grana, asimptotski problem ima optimalno rešenje $(x_2^*, \ldots, x_{n+1}^*)$ za koje važi $x_2^* + \cdots + x_{n+1}^* \leq a_1 - 1$. Za $b > a(a_1 - 1)$ važi

$$x_1^* = \frac{b - \sum_{j=2}^{n} a_j x_j^* - x_{n+1}^*}{a_1} \geq \frac{b - a\sum_{j=2}^{n+1} x_j}{a_1} \geq \frac{b - a(a_1 - 1)}{a_1} > 0,$$

pa je $(x_1^*, x_1^*, \ldots, x_n^*)$ optimalno rešenje polaznog problema sa pozitivnom (i celobrojnom) prvom koordinatom. □

Primer 3.1.3 *Primenimo treću rekurentnu formulu za rešavanje problema iz Primera 3.1.1. Primetimo da su promenljive uređene tako da je* $\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \cdots \geq \frac{c_n}{a_n}$. *i da je* $a = 5$, $a_1 = 2$. *Saglasno Teoremi 3.1.2, za* $y > 5 \cdot 1$ *vredi* $F(y) = F(y - a_1) + c_1$. *Pri tome je* $F(y) = F_4(y)$. *Otuda je:*

$$\begin{aligned}
F(9) &= F(7) + 3;\\
F(7) &= F(5) + 3;\\
F(5) &= \max\{F(3) + 3, F(1) + 4, F(1) + 5, F(0) + 2\};\\
F(3) &= \max\{F(1) + 3, F(0) + 4, -\infty, \infty, \};\\
F(1) &= 0\\
F(0) &= 0.
\end{aligned}$$

Hodom unazad nalazimo redom $F(3) = F(0) + 4 = 4$, $F(5) = F(3) + 3 = 7$, $F(7) = 10, F(9) = 13$. *Optimalno rešenje lako rekonstruišemo: Iz* $F(9) = F(7) + 3$ *sledi* $x_1 \geq 1$. *Iz* $F(7) = F(5) + 3$ *sledi da je za dostizanje optimalne vrednosti* $F(7)$ *ponovo* $x_1 \geq 1$, *dakle* $x_1 \geq 2$. *Iz* $F(5) = F(3) + 3$ *sledi da je za dostizanje* $F(3)$ *opet* $x_1 \geq 1$. *Dakle, ukupno* $x_1 \geq 3$. *Iz* $F(3) = F(0) + 4 > F(1) + 3$, *sledi da je za dostizanje* $F(3)$, $x_1 = 0$ *i* $x_2 = 1$. *Optimalno rešenje je* $x_1 = 3, x_2 = 1, x_3 = 0, x_4 = 0$.

3.2 Druge varijante problema ranca

Problem ranca se može postaviti i tako da se za svaki predmet zada broj raspoloživih primeraka, ili tako da je od svake vrste predmeta na raspolaganju tačno po jedan primerak. Sledeće razmatranje problema ranca za slučaj kada se svaki predmet može uzeti najviše jednom je uzeto iz [25].

Neka je poznato optimalno popunjavanje ranca veličine X. Ako u tom popunjavanju ne učestvuje Y-ti predmet, onda je isto popunjavanje optimalno i za ranac veličine X i prvih $Y - 1$ predmeta. Ako u optimalnom popunjavanju učestvuje i Y-ti predmet, onda ostali predmeti iz ranca čine optimalno popunjavanje za ranac veličine $X - Z(Y)$ i prvih $Y - 1$ predmeta. Zaključujemo da problem i dalje ima optimalnu podstrukturu, ali se sada veličina problema zadaje sa dva parametra. Prvi parametar je kapacitet ranca a drugi je broj predmeta. Označimo sa $B(X, Y)$ najveću vrednost ranca kapaciteta X, popunjavanog nekim od prvih Y predmeta. Tada važe sledeće rekurentne relacije:

$$\begin{aligned}
B(0,0) &= 0\\
B(X,Y) &= \begin{cases} B(X, Y-1), & Z(Y) > X,\\ \max\{B(X, Y-1), B(X - Z(Y), Y-1) + V(Y)\}, & Z(Y) \leq X. \end{cases}
\end{aligned}$$

Prema tome, podproblemi se mogu rešavati sledećim redom: najpre za sve rance i jedan predmet, pa za sve rance i dva predmeta, itd. Kada se, u poslednjoj instanci, reši problem $B(N, M)$, dobija se rešenje polaznog problema.

I ovde je, kao i u predhodnoj varijanti, radi rekonstrukcije rešenja dovoljan dodatni niz C, gde će se u $C(X)$ pamtiti indeks poslednjeg uzetog predmeta pri optimalnom popunjavanju ranca kapaciteta X.

Pri popunjavanju matrice B po kolonama koriste se samo vrednosti iz predhodne kolone (to jest iz kolone $Y - 1$). Zahvaljujući tome, možemo umesto matrice B sa M kolona, koristiti matricu sa samo dve kolone, što je značajna ušteda memorijskog prostora, koja može uticati na primenljivost postupka. Iz relacije po kojoj se računa $B(X, Y)$, može se uočiti da svi potrebni elementi predhodne kolone imaju redni broj vrste manji ili jednak X. Stoga pri popunjavanju Y-te kolone matrice B unazad (od poslednje vrste ka prvoj), možemo sve operacije izvesti u istoj koloni, pa je za čuvanje potrebnih podataka dovoljan niz B [25].

```
program ranac11;
   uses wincrt;
   var b,c:array[0..1000] of longint;
       v,z:array[1..10] of integer;
       N,M,x,y:integer;
   begin
     write('N=? '); readln(N); write('M=? '); readln(M);
     for y:=1 to M do readln(v[y],z[y]);
     for x:=0 to N do
       begin b[x]:=-1; c[x]:=0;  end;
     b[0]:=0;
     for y:=1 to M do
       for x:=N downto 1 do
         if (z[y]<=x) and (b[x-z[y]]>-1) and (b[x]<v[y]+b[x-z[y]]) then
              begin
                b[x]:=v[y]+b[x-z[y]];  c[x]:=y;
              end;

     writeln('Optimalna vrednost sadrzaja ranca je ',b[N]);
     writeln('Izabrani su predmeti: ');
     x:=N;
     while c[x]>0 do
       begin
         write(c[x],'  '); x:=x-z[c[x]];
       end;
   end.
```

Efikasniji algoritam je implementiran u sledećem kôdu:

```
program Knapsack_Problem; var  v, z: array [1..1000] of Integer;
     s: array [1..1000] of Boolean;
     d, p: array [0..12000] of Integer;
     n, m, Sol, ds: Integer;

  procedure Input;
  var f: Text;
      i: Integer;
```

```
  begin
    Assign (f, 'knapsack.in');  Reset(f);  Read(f, n, m);
    for i := 1 to n do Read(f,z[i],v[i]);
    Close (f);
  end;

  procedure Solve;
  var  i, j, x: Integer;
  begin
    FillChar(s, SizeOf(s), False);
    ds:= 0;
    repeat
       for i:= 1 to m do
         begin
             d[i]:=-1;   p[i]:=-1;
         end;
       d [0]:=0;       p[0]:=0;
       for i := 1 to n do
        if (not s [i]) then
           for j:= m downto z[i] do
              if(d[j-z[i]]<>-1) and (d[j-z[i]]+v[i]>d[j]) then
                 begin
                    d [j] := d[j-z[i]]+v[i];p [j] := i;
                 end;
        if (ds = 0) then
           begin
             x := m;
             for i := 0 to m do
                if(d[x]<d[i]) then x := i;
             m := x;
             Sol := d [m];
           end;
        x := p [m];
        while (m > 0) and (not s[p[m]]) and (p[m] <= x) do
          begin
            s[p[m]]:= True;   Inc(ds);  x := p[m];   m := m-z[p[m]];
         end;
    until (m = 0);
  end;

  procedure Output;
  var  f: Text;    i: Integer;
  begin
    Assign (f, 'knapsack.out');  ReWrite (f);  WriteLn(f, Sol);
    for i:= 1 to n do if s[i] then  WriteLn(f, i);
    Close(f);
  end;

begin
  Input; Solve;  Output;
end.
```

Pri rešavanju ove varijante problema (svaki predmet najviše jednom) treba imati na umu još jednu važnu činjenicu: rešavanje problema dinamičkog programiranja se isplati jedino ako je broj predmeta M dovoljno veliki, a kapacitet ranca N relativno mali, tako da se NM operacija (koliko je pribljižno potrebno za ovaj način rešavanja)

izvrši brže nego nalaženje svih 2^N mogućih podnizova i njihovih zbirova vrednosti, odnosno zapremina. Recimo za $N = 10000, M = 100$, dinamičkim programiranjem rešenje dobijamo bez čekanja, dok 2^{100} podnizova prakticno ne možemo formirati. Međutim, u slučaju malog M i velikog N bolje je isprobati sve podnizove [25].

U nastavku je dat program za problem ranca u kome se svaki predmet uzima određeni broj puta. Promenljive imaju sledeći smisao:

n - broj elemenata,
m - zapremina ranca,
$najbolja$ - zapremina pri kojoj se uzima najveća vrednost,
$zapremina[i]$ - zapremine,
$cena[i]$ - cene predmeta,
$broj[i]$ - broj predmeta tipa i,
$ubacen[i]$ ako je indeks predmeta koji je ubačen da bi se dobila zapremina i,
$ubacen[i] = -1$ ukoliko još nismo dobili zapreminu i,
$ostalo[i]$ - koliko je još predmeta tipa onog koji je ubačen ostalo,
$best[i]$ - najbolja cena koja može da se dobije za zapreminu i,
$uzeto[i]$ - koliko je lopov uzeo stvari i-te vrste.

```
var max,i,j,l,n,m,najbolja:integer;
    broj,zapremina,cena,best,ubacen,ostalo,uzeto:array[0..1000]of integer;
    f:text;
begin
   assign(f,'input.txt');reset(f);  read(f,n,m);
   for i:=1 to n do read(f,zapremina[i],cena[i],broj[i]);
   close(f);
   for i:=1 to m do ubacen[i]:=-1;
   fillchar(best,sizeof(best),0);   {popunjava svuda 0}
   ubacen[0]:=0;
   for i:=1 to n do                {i prolazi kroz predmete}
      for j:=0 to m do            {j prolazi kroz zapremine}
         if (ubacen[j]<>-1)and(j+zapremina[i]<=m) then
                     {ako je vec dobijena zapremina j}
            if cena[i]+best[j]>best[j+zapremina[i]] then
                     {ako je povoljno da dodajemo i}
               if (ubacen[j]<>i)or(ostalo[j]>0) then
                  begin
                     {ako moze jos jedan}
                     best[j+zapremina[i]]:=best[j]+cena[i];
                     ubacen[j+zapremina[i]]:=i;  {ubacuje i-ti predmet}
                     if ubacen[j]=i then
                        ostalo[j+zapremina[i]]:=ostalo[j]-1
                     else      {koliko je predmeta i-te vrste ostalo}
                        ostalo[j+zapremina[i]]:=broj[i]-1;
                  end;
   max:=0;      najbolja:=0;
   for i:=1 to m do
     if best[i]>max then
         begin
           max:=best[i];  najbolja:=i;  {nalazi se najbolja zapremina}
         end;
   l:=najbolja;
   fillchar(uzeto,sizeof(uzeto),0);
```

```
    while l>0 do begin
       inc(uzeto[ubacen[l]]);
       l:=l-zapremina[ubacen[l]];          {sada rekonstruise niz}
    end;
    assign(f,'output.txt');rewrite(f);
    writeln(f,'Lopov puni zapreminu ', najbolja,' pri cemu odnosi vrednost ',max);
    writeln(f,'tako sto uzima sledece elemente:');
    for i:=1 to n do
        if uzeto[i]>0 then
            writeln(f,'element broj ',i,' ',uzeto[i],' put(a)');
    close(f);
end.
```

Pomenimo poznatije primere primene varijante problema ranca u kojoj se svaki predmet može uzeti najviše jedanput.

Primer 3.2.1 Poreklo zadatka: [25].

Iz datog niza prirodnih brojeva A, izdvojiti podniz čiji je zbir elemenata dati broj N, ili ispisati poruku da takav niz ne postoji.

Uputstvo. Označimo sa M broj elemenata niza A, a sa S njihovu sumu. Možemo smatrati da su nizom A zadati takvi predmeti, kojima je $v_i = z_i = a_i, i = 1, \ldots, M$. Sada je jasno da se problem svodi na optimalno popunjavanje ranca kapaciteta N, pri čemu rešenja ima ako i samo ako je vrednost optimalnog sadržaja ranca jednaka njegovoj zapremini M, to jest ako i samo ako je ranac napunjen do vrha.

```
program ranac3;
  var b,c:array[0..1000] of longint;
      a:array[1..100] of integer;
      N,M,x,y:integer;
  begin
    write('N=? '); readln(N); write('M=? '); readln(M);
    for y:=1 to M do readln(a[y]);
    for x:=0 to N do
      begin b[x]:=0; c[x]:=0;  end;
    for y:=1 to M do
      for x:=N downto 1 do
        if (a[y]<=x) and (b[x] < a[y]+b[x-a[y]]) then
             begin
               b[x]:=a[y]+b[x-a[y]];  c[x]:=y;
             end;

    if b[N]=N then
      begin
        writeln('Trazeni podniz je ');
        x:=N;
        while c[x]>0 do
          begin
             write(a[c[x]],'  '); x:=x-a[c[x]];
          end;
      end
    else writeln('Ne postoji takav podniz');
  end.
```

Primer 3.2.2 Poreklo zadatka: [25].

Brojeve iz datog niza prirodnih brojeva A podeliti u dve grupe, tako da je razlika zbirova elemenata u pojedinim grupama minimalna.

U ovom primeru niz A ponovo ima ulogu i niza Z i niza V. Neka S i N imaju ista značenja kao i u predhodnom primeru. Rešenje problema se sastoji u sledećem: predmete koje čine optimalno popunjavanje ranca kapaciteta $S/2$ (deljenje je celobrojno) trebalo bi svrstati u jednu, a sve ostale predmete u drugu grupu. Ako je ranac popunjen do vrha, grupa će za parno S imati jednake zbirove. U protivnom prva grupa ima manji zbir, adruga veći, ali ti zbirovi su najbliži vrednosti $S/2$, a time i jedan drugom.

```
Program TegoviDin;
  var b,c:array[0..4000] of longint;
      t:array[1..50] of longint;
      q,n,y,ind,max:longint;
      s,min:real;
  begin
     write('Unesite broj tegova: ');read(n);
     s:=0;max:=0;
     for q:=1 to n do
       begin
         write('unesite ',q,'. teg: ');   read(t[q]);
         s:=s+t[q]; max:=max+t[q];
       end;
     min:=s; s:=s/2;
     for q:= 0 to (max div 2) do
       begin
         b[q]:=0; c[nq]:=0;
       end;
     for y:=1 to n do
       for q:=(max div 2) downto 1 do
         if t[y]<=q then
           if b[q]< t[y]+b[q-t[y]] then
             begin
               b[q]:=t[y]+b[q-t[y]];   c[q]:=y;
             end;
     for q:=1 to (max div 2) do
       if abs(b[q]-s)<min then
         begin
           min:=abs(b[q]-s);    ind:=q;
          end;
     writeln('jedan tas: ',b[ind]:2,', drugi tas: ',max-b[ind]);
     q:=ind;
     writeln('redni brojevi tegova na jednom od tasova:');
     while c[q]>0 do
       begin
         write(c[q]:3);      q:=q - t[c[q]];
       end;
  end.
```

Vrednost optimalno popunjenog tasa (ranca) je uvek u opsegu od 1 do (`max div 2`). To važi zato što po popunjavanju jednog tasa drugi tas ima tačno utvrđenu vrednost koja se sigurno može dobiti kombinacijom datih tegova, kao i vrednost drugog tasa. Neformalni dokaz se sastoji u sledećem. Neka je q vrednost za koju su tasovi u ravnoteži (ako je `max` neparno, tasovi su u najboljoj ravnotezi kada se njihove vrednosti razlikuju za jedan, ukoliko je `max` parno, q= `max div 2`. Vrednost `max` predstavlja zbir vrednosti svih tegova. Pretpostavimo da je optimalno popunjen tas tezine x. Tada drugi tas ima vrednost $max - x$. Pretpostavimo dalje da je

vrednost optimalno popunjenog tasa

$$x \leq max\ div\ 2.$$

Ukoliko bi postojalo bolje rešenje za koje je

$$max\ div\ 2 - x > |max\ div\ 2 - a|,$$

gde je a bolje popunjen tas i $a >$*max div 2* moralo bi da vazi i da je drugi tas popunjen vrednošću *max*$-a <$ *max div 2*, pa bi se ova vrednost sigurno našla kao optimalno rešenje. Kada su i x i a manja ili jednaka `max div 2`, dokaz je trivijalan.

Primer 3.2.3 Poreklo zadatka: [25].

Dat je izraz $a_1 - a_2 - ... - a_n$, gde su svi brojevi u nizu a celi. Postaviti zagrade u ovaj izraz, tako da vrednost izraza bude jednaka datom broju X. **Uputstvo:** Problem se ekvivalentno može postaviti ovako: brojeve iz datog niza celih brojeva A, podeliti u dve grupe, tako da a_1 obavezno pripadne prvoj, a a_2 drugoj grupi, i da razlika zbirova elemenata prve i druge grupe bude jednaka X. Ova formulacija je ekvivalentna, jer zagrade uvek mogu da se postave tako da ispred brojeva iz prve grupe ima paran broj minusa, a ispred brojeva iz druge grupe neparan broj minusa koji se na te brojeve odnose. Ovaj postupak postavljanja zagrada je verovatno lakše izvesti, nego precizno opisati, pa prepuštamo čitaocu da nakon formiranja dveju grupa brojeva zagrade postavi sam.

Neka su S_1 i S_2 sume brojeva prve i druge grupe, a S suma svih N brojeva, tako da je $X = S_1 - S_2 = 2S_1 - S$. Rešenje preformulisanog zadatka se sada svodi na izbor nekih od brojeva $a_3, a_4, ..., a_n$, (pazite: a_1 i a_2 su izostavljeni), tako da je zbir izabranih brojeva jednak $M = (X + S)/2 - a_1$, a taj zadatak je već razmatran.

U problemima koji se svode na problem ranca, varijanta "svaki predmet najviše jednom", do rešenja se može doći na potpuno isti načini u slučaju da su elementi niza celi brojevi. Uslov da su elementi niza pozitivni dat je samo zato što u protivnom interpretacija gubi fizički smisao: morale bi se uvesti negativne zapremine i negativne vrednosti. Osobina niza koja je suštinski bitna u prethodna tri primera je da su vrednosti elemenata niza, kao i njihov ukupan broj celobrojne veličine čije vrednosti nisu velike. Za niz od N celih brojeva koji su svi po apsolutnoj vrednosti manji od G, sume i/ili razlike elemenata svih podnizova (svaki element može da se uzme direktno, sa promenjenim predznakom ili da se izostavi), mogu da imaju manje od od $2NG$ različitih vrednosti, bilo pozitivnih bilo negativnih. Tipično, za $N = G = 100$, dinamičkim programiranjem se lako ispituje za svaki od mogućih 20000 brojeva, da li se može pojaviti kao suma ili razlika nekih elemenata niza, pozitivnih ili negativnih. Ovakav postupak rešavanja se ne bi mogao primeniti u slučaju da mogućih kandidata za zbir (ili razliku) nekih (ili svih) elemenata niza ima za nekoliko redova veličine više, a to se dešava ako je ograničenje elemenata niza mnogo veće, ili ako se dopuštaju realni brojevi. Tada bi morali da se isprobaju svi podnizovi, što znači da efikasnog postupka u tom slučaju i nema [25].

Sumirajmo na kraju šta je sve bilo potrebno da bismo svaku od varijanti problema ranaca (ili bilo kog drugog problema) rešili dinamičkim programiranjem [25]:

- Analiziramo strukturu optimalnog rešenja. Uočimo da su izvesni delovi jednog optimalnog rešenja problema upravo optimalna rešenja manjih problema istog tipa. Odredimo parametre koji zadaju veličinu problema i podproblema.

- Vrednost optimalnog rešenja zadamo rekurentno, tj. koristeći vrednosti optimalnih rešenja nekih manjih problema.

- Nađemo efikasan način da na osnovu vrednosti optimalnih rešenja podproblema dobijemo vrednost optimalnog rešenja problema (tipično koristeći jedan ciklus, ponekad samo par naredbi).

- Nađemo i tabeliramo vrednosti rešenja za najjednostavnije podprobleme, a zatim kombinujući vrednosti rešenja manjih podproblema nalazimo vrednost rešenja većih podproblema, sve do rešenja samog problema. Pri tome tabeliramo vrednosti rešenja i delimične informacije o načinu dobijanja rešenja za sve podprobleme.

- Na osnovu tabelarnih informacija konstruišemo traženo optimalno rešenje.

3.3 Različite varijante problema ranca

U disertaciji [18] razmatra se familija problema kombinatotne optimizacije koji su vezani za problem ranca. Svi ti problemi su NP-hard, i razmatraju se egzaktni algoritmi koji imaju razumno vreme za rešavanje problema. Slično ponašanje ima simpleks metod, koji uprkos eksponencijalnoj složenosti u najlošijem slučaju pokazuje razumno vreme za rešavanje svih realnih problema.

Problem ranca zahteva podskup datih artikala koji se odabiraju tako da se maksimizira ciljna funkcija bez prekoračenja kapaciteta ranca (ranaca). Najpoznatiji slučajevi problema ranca su sledeći.

- U $0-1$*problemu ranca* ($0-1$ knapsack problem) svaki artikal može biti odabran najviše jedanput.

- U *ograničenom problemu ranca* (bounded knapsack problem), postoji ograničena količina za svaki tip artikla.

- Problem ranca sa višestrukim izborom (multiple-choice knapsack problem) se pojavljuje kada se artikli mogu odabirati iz disjunktnih klasa.

- Kada se nekoliko ranaca može istovremeno, dobija se *višestruki problem ranca* (multiple knapsack problem).

- Najopštiji oblik problema ranca je *višestruko ograničen problem ranca* (multiconstrained knapsack problem) koji je u suštini opšti oblik celobrojnog programiranja sa pozitivnim koeficijentima.

U matematičkim modelima koji slede dati su neki artikli sa profitom p_j i težinama w_j koji se pakuju u jednom ili više ranaca kapaciteta c. Pretpostavlja se da su koeficijenti p_j, w_j, c pozitivni celi brojevi.

0− problem ranca je problem izbora podskupa od n artikala tako da je odgovarajući profit maksimalan.

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} p_j x_j \\ \text{p.o.} \quad & \sum_{j=1}^{n} w_j x_j \leq c \\ & x_j \in \{0,1\},\ j = 1,\ldots,n, \end{aligned}$$

gde je x_j binarna promenljiva koja uzima vrednost 1 ako se artikal j uključuje u ranac, a inače 0.

Ukoliko je na raspolaganju ograničen iznos m_j od svakog tipa j artikala, ograničeni problem ranca ima model

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} p_j x_j \\ \text{p.o.} \quad & \sum_{j=1}^{n} w_j x_j \leq c \\ & x_j \in \{0,1,\ldots,m_j\},\ j = 1,\ldots,n, \end{aligned}$$

Neograničeni problem ranca (unbounded knapsack problem) je generalizacija ograničenog problema ranca u kome je broj artikala svakog tpa neograničen:

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} p_j x_j \\ \text{p.o.} \quad & \sum_{j=1}^{n} w_j x_j \leq c \\ & x_j \geq 0 \text{ ceo broj},\ j = 1,\ldots,n, \end{aligned}$$

Suštinski, svaka promenljiva x_j u neograničenom problemu ranca je ograničena kapacitetom c, jer je težina svakog artikla najmanje 1.

Sledeća generalizacija $0-1$ problema ranca je mogućnost izbora tačno jednog artikla tipa j iz svake od k klasa N_i, $i = 1,\ldots,k$ tako da se profit maksimizira. Na

taj način se dobija problem ranca sa višestrukim izborom:

$$\begin{aligned}
\max \quad & \sum_{i=1}^{k} \sum_{j \in N_i} p_{ij} x_{ij} \\
\text{p.o.} \quad & \sum_{i=1}^{k} \sum_{j \in N_i} w_{ij} x_{ij} \leq c \\
& \sum_{j \in N_i} x_{ij} = 1, \ i = 1, \ldots, k, \\
& x_{ij} \in \{0, 1\}, \ i = 1, \ldots, k, \ j \in N_i.
\end{aligned}$$

U ovom modelu binarna promenljiva $x_{ij} = 1$ označava da je artikal tipa j odabran iz klase i. Ograničenje $\sum_{j \in N_i} x_{ij} = 1$, $i = 1, \ldots, k$ osigurava da se odabira tačno jedan artikal iz svake klase.

Ukoliko profit p_j postane jednak težini w_j za svaki artikal u $0 - 1$ problemu ranca, dobija se *Subset–sum Problem*:

$$\begin{aligned}
\max \quad & \sum_{j=1}^{n} w_j x_j \\
\text{p.o.} \quad & \sum_{j=1}^{n} w_j x_j \leq c \\
& x_j \in \{0, 1\}, \ j = 1, \ldots, n.
\end{aligned}$$

Ime problema označava da se problem sastoji u izboru nekog podskupa vrednosti $w_1, \ldots, w_n$ tako da je je suma što veća i da ne prevazilazi c.

Sada zamislite kasira koji želi da dobije iznos novca c koristeći najmanji mogući broj novčanica $w_1, \ldots, w_n$. Takav problem se naziva *Change–making Problem*:

$$\begin{aligned}
\min \quad & \sum_{j=1}^{n} x_j \\
\text{p.o.} \quad & \sum_{j=1}^{n} w_j x_j = c \\
& x_j \geq 0 \text{ ceo broj}, \ j = 1, \ldots, n.
\end{aligned}$$

U ovom modelu w_j predstavlja vrednost novčanice tipa j koja se koristi sa x_j jedinica.

Ako se odabira n predmeta i pakuje u m ranaca (moguće) različitih kapaciteta

c_i tako da da se dobije najveći mogući profit, dobija se višestruki problem ranca.

$$\begin{aligned}
\max \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} p_j x_{ij} \\
\text{p.o.} \quad & \sum_{j=1}^{n} w_j x_{ij} \leq c_i,\ i = 1, \ldots, m \\
& \sum_{i=1}^{m} x_{ij} \leq 1,\ j = 1, \ldots, n, \\
& x_{ij} \in \{0, 1\},\ i = 1, \ldots, m,\ j = 1, \ldots, n.
\end{aligned}$$

U ovom slučaju $x_{ij} = 1$ označava da artikal tipa j bi trebalo da bude pakovan u ranac i, dok ograničenje $\sum_{j=1}^{n} w_j x_{ij} \leq c_i$ osigurava da je kapacitativno ograničenje ispunjeno. Ograničenje $\sum_{i=1}^{m} x_{ij} \leq 1$ osigurava da je svaki artikal odabran najviše jedanput.

Vrlo koristan model je *problem binarnog pakovanja* (Bin–packing Problem) u kome u kome bi trebalo da se svaki od n artikala pakuje u kutije jednakih veličina, tako da je broj upotrebljenih kutija minimalan:

$$\begin{aligned}
\max \quad & \sum_{i=1}^{n} y_i \\
\text{p.o.} \quad & \sum_{j=1}^{n} w_j x_{ij} \leq c y_i,\ i = 1, \ldots, n \\
& \sum_{i=1}^{n} x_{ij} = 1,\ j = 1, \ldots, n, \\
& x_{ij} \in \{0, 1\},\ i = 1, \ldots, m,\ j = 1, \ldots, n \\
& y_i \in \{0, 1\},\ i = 1, \ldots, n.
\end{aligned}$$

U ovom modelu y_i da li se kutija tipa i koristi, dok x_{ij} da artikal tipa j bi trebalo da se pakuje u kutiju tipa i. Ograničenje $\sum_{i=1}^{n} x_{ij} = 1$ obezbeđuje da se svaki artikal pakuje tačno jednom, dok nejednakost $\sum_{j=1}^{n} w_j x_{ij} \leq c y_i$ osigurava da se kapacitativna ograničenja ispunjavaju za sve upotrebljene kutije.

Najopštiji oblik problema ranca jeste višestruko ograničen problem ranca. On je u suštini opšti celobrojni problem u kome su svi koeficijenti p_j, w_{ij}, c_i nenegativni

celi brojevi.

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} p_j x_j \\ \text{p.o.} \quad & \sum_{j=1}^{n} w_j x_{ij} \leq c y_i, \ i = 1, \ldots, n \\ & \sum_{j=1}^{n} w_{ij} x_j \leq c_i, \ i = 1, \ldots, m, \\ & x_j \geq 0 \text{ ceo broj }, \ j = 1, \ldots, n. \end{aligned}$$

4 Ne baš klasični problemi dinamičkog programiranja

U ovom odeljku izložićemo veći broj primera problema koji se veoma efikasno rešavaju primenom tehnike DP. Problemi su uglavnom formulisani u obliku zadataka. Za svaki problem data je formulacija, kratko uputstvo za rešavanje, kao i detaljno rešenje. Rešenja su detaljno obrazložena i data je analiza složenosti kao i memorijskih zahteva za svako rešenje. Za pojedina rešenja data je i implementacija u programskom jeziku PASCAL. Savetujemo čitaocu da pre nego što pogleda rešenje svakog zadatka, najpre prouči formulaciju i pokuša da sam konstruiše rešenje primenom tehnike DP. Neki od zadataka su vrlo slični nekom od klasičnih problema DP (odeljak 2) što će čitaoc najverovatnije bez problema uočiti. Tek nakon što iscrpe sve svoje ideje čitaoc se savetuje da najpre pročita uputstvo i ponovo razmisli o problemu a tek onda da pogleda rešenje. To je najbolji način da se ovlada ovom zaista korisnom tehnikom za rešavanje problema.

Za neke od zadataka namerno nije data implementacija već se čitaoc upućuje da sam implementira izloženo rešenje. Tako će čitaoc biti u mogućnosti da proveri da li je zaista razumeo u potpunosti izloženo rešenje. Ovo pravilo je generalno i važi za sve tehnike, metode i algoritme u programiranju.

4.1 Z-Cifre

Poreklo zadatka: Z-trening [28].

Koliko ima $2N$-tocifrenih brojeva ($1 \leq N \leq 500$) tako da je zbir prvih N cifara jednak proizvodu poslednjih N cifara.

Uputstvo: Primetimo da je najveći mogući zbir prvih N cifara posmatranih brojeva jednak $9N \leq 4500$. Prema tome, dovoljno je odrediti koliko ima N-tocifrenih brojeva čiji je zbir, odnosno proizvod jednak i (označimo ih sa $s(i)$ odnosno $p(i)$). Sada je rešenje problema $s(1)p(1) + \ldots + s(4500)p(4500)$. Brojeve $v(i)$ odnosno $p(i)$ određujemo primenom rekurentne formule za dvodimenzionalne nizove $s(i, N)$ i $p(i, N)$ koji označavaju koliko ima N-tocifrenih brojeva čiji je zbir odnosno proizvod jednak i.

Rešenje: Neka je $x = \overline{x_1, \ldots, x_N, x_{N+1}, \ldots, x_{2N}}$ proizvoljan broj koji zadovoljava uslove date u formulaciji problema i neka su $x^{(1)} = \overline{x_1 \cdots x_N}$ i $x^{(2)} = \overline{x_{N+1} \cdots x_{2N}}$ brojevi sastavljeni od prvih N odnosno poslednjih N cifara broja x. Očigledno da broj x na jedinstven način određuje brojeve $x^{(1)}$ i $x^{(2)}$. Pritom, broj $x^{(1)}$ mora imati tačno N cifara (prva cifra brojeva $x^{(1)}$ i x je x_1 i ona mora biti različita od nule) dok $x^{(2)}$ mora imati najviše N cifara, tj. može počinjati i sa nulom [1]. Primetimo da važi i obratno, da svaka dva N-tocifrena broja $x^{(1)}$ i $x^{(2)}$ takvi da je zbir cifara prvog jednak proizvodu cifara drugog broja jednoznačno određuju broj x.

Označimo sa $s(i)$, odnosno $p(i)$ broj N-tocifrenih prirodnih brojeva čiji je zbir odnosno proizvod jednak i. Imamo da je $s(i)$ odnosno $p(i)$ broj načina da se odredi broj $x^{(1)}$ odnosno $x^{(2)}$. Prema tome, rešenje problema dato je izrazom: $\sum_i s(i)p(i)$. Sada se pitanje granica u predhodnoj sumi. Donja granica je očigledno 1, dok je gornja određena maksimalnom mogućom vrednošću zbira odnosno proizvoda N-tocifrenog broja. Ove vrednosti su $9N \leq 9 \cdot 500 \leq 4500$ kao i 9^N. Očigledno, prva granica je manja, pa nju možemo uzeti kao gornju granicu sume.

Preostaje nam samo još da odredimo $s(i)$ i $p(i)$ za svako $i = 1, \ldots, 4500$. Ovaj problem je veoma sličan problemu ranca. Neka je $s(i, j)$ odnosno $p(i, j)$ broj j-tocifrenih brojeva takvih da je zbir odnosno proizvod cifara jednak i. Neka je $y = \overline{y_1 \cdots y_{j-1} y_j}$ jedan takav broj. Očigledno je zbir cifara broja y (označimo ga sa $S(y)$) jednak: $S(y) = S(y') + y_j$ gde je $y' = \overline{y_1 \cdots y_{j-1}}$. Prema tome, ukoliko odaberemo zadnju cifru y_j, broj y' možemo izabrati na $s(i - y_j, j - 1)$ načina. Naravno, mora da važi da je $y_j \leq i$, kao i $y_j \leq 9$ (y_j je cifra). Prema tome, važi sledeća rekurentna formula:

$$s(i, j) = \sum_{1 \leq y_j \leq \min\{9, i\}} s(i - y_j, j - 1) \tag{4.1.1}$$

Na sličan način se dobija i odgovarajuća rekurentna formula za niz $p(i, j)$:

$$p(i, j) = \sum_{y_j \mid i, 1 \leq y_j \leq 9} p\left(\frac{i}{y_j}, j - 1\right) \tag{4.1.2}$$

Startne vrednosti za $s(i, j)$ i $p(i, j)$ su:

$$s(i, 1) = p(i, 1) = \begin{cases} 1, & 1 \leq i \leq 9 \\ 0, & \text{u suprotnom} \end{cases}$$

Vrednosti $s(i, j)$ i $p(i, j)$ računamo za svako $i = 1, \ldots, 9N$ i $j = 2, \ldots, N$ primenom rekurentnih formula (4.1.1) i (4.1.2). Na kraju uzimamo vrednosti $s(i) = s(i, N)$ i $p(i) = p(i, N)$ za svako $i = 1, \ldots, 9N$.

Očigledno, izrazi (4.1.1) i (4.1.2) zahtevaju maksimalno po 9 aritmetičkih operacija. Pritom moramo da izračunamo ukupno $9N \cdot N = 9N^2$ elemenata nizova $s(i, j)$ i $p(i, j)$, pa je prema tome ukupan broj potrebnih aritmetičkih operacija maksimalno $2 \cdot 9 \cdot 9N^2 = 162N^2 = \mathcal{O}(N^2)$.

[1] Međutim, ukoliko je $x_{N+1} = 0$, tada je proizvod cifara broja $x^{(2)}$ jednak nuli, odnosno imamo da je zbir cifara broja $x^{(1)}$ jednak nuli, što je nemoguće

Primetimo da se u rekurentnim izrazima (4.1.1) i (4.1.2) prilikom izračunavanja vrednosti j-te kolone matrica s i p koriste samo vrednosti u $j-1$-voj koloni, kao kod problema ranca, pa je i ovde, u implementaciji dovoljno pamtiti samo poslednju kolonu ovih matrica. Prema tome, ovo rešenje zahteva samo dva niza (s i p) od po 4500 elemenata.

4.2 Z-Menadžer

Poreklo zadatka: Z-trening [28]

Mali Z je dobio zadatak da direktoru firme u kojoj radi snimi filmove. Direktor je malom Z-u dao spisak filmova, i za svaki od filmova: vreme kada počinje, vreme kada se završava, i koliko ce Z dobiti para ako snimi taj film.

Mali Z ima na raspolaganju samo jedan video rekorder, pa ukoliko se dva filma vremenski preklapaju (dva filma se preklapaju i u slučaju kada jedan počinje u trenutku kada se drugi završava), on može da snimi samo jedan od njih. Odrediti koje filmove mali Z treba da snimi tako da tako da zaradi što je moguće više novca.

Predpostaviti da važe sledeća ograničenja: $1 <= N <= 100000$ gde je N ukupan broj filmova, $1 \leq p(i) < k(i) \leq 2000000000$ i $1 \leq c(i) \leq 1000$ gde su $p(i), k(i)$ i $c(i)$ redom vremena početka i kraja i-tog filma $1 \leq i \leq N$ a $c(i)$ zarada koju mali Z dobija ako snimi i-ti film.

Uputstvo: Sortirati filmove po rastućim vrednostima vremena završetka $k(i)$. Neka je $z(i)$ maksimalna zarada koja može biti ostvarena ukoliko je poslednji snimljen i-ti film. Ako je predzadnji j-ti film, onda važi $k(j) < p(i)$ i $z(i) = z(j) + c(i)$. Sada treba naći j tako da je $z(j)$ maksimalno.

Rešenje: Sortirajmo sve filmove tako da važi $k(1) \leq k(2) \leq \cdots \leq k(n)$. Neka je $z(i)$ maksimalna zarada koja može biti ostvarena ukoliko je poslednji snimljen i-ti film. Neka je j-ti film snimljen predposlednji. Tada iz uslova zadatka dobijamo da važi $k(j) > p(i)$. Zarada ostvarena filmovima

Neka su $i_1 < i_2 < \cdots < i_k$ filmovi koje treba snimiti da bi se maksimizovala zarada, pri čemu je poslednji snimljen i-ti film (tj $i_k = i$). Iz uslova problema imamo da važi $p(i) = p(i_k) > k(i_{k-1})$, pa je ukupna zarada ostvarena filmovima $i_1, \ldots, i_{k-1}$ manja ili jednaka $z(i_{k-1})$. Prema tome, imamo da važi:

$$z(i) = c(i_1) + \ldots + c(i_{k-1}) + c(i_k) \leq z(i_{k-1}) + c(i_k) \tag{4.2.1}$$

Ukoliko u izrazu (4.2.1) važi stroga nejednakost, tada možemo odabrati filmove $j_1, \ldots, j_{l-1}, j_l = i_{k-1}$ koji se međusobno ne preklapaju, pri čemu je ukupna zarada ostvarena ovim filmovima jednaka $z(i_{k-1})$. Tada se snimanjem filmova $j_1, \ldots, j_l, i$ dobija zarada

$$z(i) = z(i_{k-1}) + c(i_k) \tag{4.2.2}$$

veća od $z(i)$ što je u suprotnosti sa definicijom broja $z(i)$. Prema tome zaključujemo da u izrazu (4.2.1) važi jednakost, odnosno $z(i) = z(i_{k-1}) + c(i_k)$. Ostaje nam još samo da za zadato i odredimo i_{k-1} (tj film koji predhodno treba snimiti). Pošto je $z(i)$ maksimalna zarada koja se može ostvariti, ako je i-ti film poslednji snimljen,

sledi da i_{k-1} treba izabrati tako da je $z(i_{k-1})$ maksimalno, pri čemu važi uslov zadatka $p(i_k) > k(i_{k-1})$. Prema tome, važi sledeća rekurentna formula za $z(i)$:

$$z(i) = c(i) + \max\{z(j) \mid 0 \leq j < i, p(i) > k(j)\} \tag{4.2.3}$$

Startna vrednost za računanje $z(i)$ je $z(0) = 0$ (ukoliko se ne snimi nijedan film, zarada je jednaka nuli). Krajnje rešenje problema dobijamo kao $\max_{1 \leq i \leq N} z(i)$.

Što se tiče memorijskih zahteva, ovo rešenje zahteva samo jedan niz z od N elemenata. Međutim situacija je nešto složenija kada je u pitanju složenost algoritma. Naime, za izračunavanje vrednosti svakog elementa $z(i)$ direktnom primenom izraza (4.2.3), potrebno je ispitati ukupno $i - 1$ različitih mogućnosti za j, pa je složenost ovakve implementacije $\mathcal{O}(N^2)$. Pokažimo sada kako se ova vrednost može poboljšati. Prva ideja je da uporedo za nizom $z(i)$ računamo i niz maksimuma: $mz(i) = \max_{1 \leq k \leq n} z(i)$. Primetimo takođe, da ako je $p(i) > k(j)$, onda je $p(i) > k(j) > k(l)$ za svako $l = 1, \ldots, j$. Prema tome, dovoljno je naći:

$$j_0 = \text{argmin}\{k(j) \mid k(j) < p(i), j = 1, \ldots, i\}$$

i onda je:

$$\begin{aligned} z(i) &= c(i) + mz(j_0) \\ mz(i) &= \max\{mz(i-1), z(i)\} \end{aligned} \tag{4.2.4}$$

Pošto je niz $k(j)$ sortiran, vrednost j_0 možemo naći primenom tehnike binarnog pretraživanja. Primenom ove modifikacije smanjujemo ukupnu složenost izračunavanja jedne vrednosti $z(i)$ na $\mathcal{O}(\log N)$ operacija, a celog algoritma na $\mathcal{O}(N \log N)$ (sortiranje možemo obaviti koristeći neki od algoritama koji rade takođe u vremenu $O(N \log N)$, na primer QuickSort, MergeSort, HeapSort, itd.). Sada se ovaj algoritam bez problema može primeniti za velike vrednosti broja N ($N \leq 100000$).

Implementacija:

```
program Z-Menadzer;

Const MaxN=100005; type tniz=array [0..MaxN] of longint;

var x,y,v:tniz;
    n:longint;

procedure sort(l,r: longint); var i,j,xp,yp,tmp: longint; begin
  i:=l; j:=r;
  xp:=x[(l+r) DIV 2]; yp:=y[(l+r) div 2];
  repeat
    while (y[i]<yp) or ((y[i]=yp) and (x[i]<xp)) do i:=i+1;
    while (yp<y[j]) or ((y[j]=yp) and (xp<x[j])) do j:=j-1;
    if i<=j then
    begin
      tmp:=y[i]; y[i]:=y[j]; y[j]:=tmp;
      tmp:=x[i]; x[i]:=x[j]; x[j]:=tmp;
      tmp:=v[i]; v[i]:=v[j]; v[j]:=tmp;
      i:=i+1; j:=j-1;
    end;
  until i>j;
```

```
  if l<j then sort(l,j);
  if i<r then sort(i,r);
end;

function binpret(el:longint):longint; var l,d,sr  :longint; begin
  l:=1;  d:=n;
  while d-l>1 do begin
    sr:=(l+d) div 2;
    if y[sr]>=el then d:=sr else l:=sr;
  end;
  if el>y[l] then binpret:=l else binpret:=l-1;
  if el>y[d] then binpret:=d;    {new}
end;

var max:tniz;
    i,res,ind:longint;

begin
  readln(n);
  for i:=1 to n do
    readln(x[i],y[i],v[i]);
  fillchar(max,sizeof(max),0);

  sort(1,n);

  max[0]:=0;  max[1]:=v[1];
  for i:=2 to n do begin
    ind:=binpret(x[i]);
    if max[ind]+v[i]>max[i-1] then max[i]:=max[ind]+v[i]
                              else max[i]:=max[i-1];
  end;

  res:=max[n];
  writeln(res);
end.
```

4.3 Igra

Mirko i Slavko igraju sledeću igru: na stolu se nalaze n brojeva poredanih u niz. Prvo Mirko uzme jedan broj, sa leve ili sa desne strane niza. Zatim Slavko uzme jedan broj sa leve ili desne strane preostalog niza, i tako naizmenično dok ne pokupe sve brojeve sa stola. Napisati program koji izracunava, pod pretpostavkom da i Mirko i Slavko igraju optimalno, koliki je maksimalni zbir brojeva koji Slavko može da skupi.

Sa standardnog ulaza se ucitava u jednom redu broj n ($n \leq 128$), a u drugom redu n brojeva, u opsegu od 0 do 100.

Na standardni izlaz treba ispisate samo jedan broj, maksimalan zbir brojeva koje Mirko može da osvoji.

Uputstvo: Označimo sa $m(i, j)$ odnosno $s(i, j)$ maksimalan zbir brojeva koje može da sakupi Mirko odnosno Slavko ukoliko obojica igraju optimalno, i ako su na tabli zapisani brojevi $a_i, a_{i+1}, \ldots, a_{j-1}, a_j$.

Rešenje: Pošto je Mirko prvi na potezu, on bira koji broj će uzeti, a_i ili a_j. Ukoliko odabere broj a_i, na tabli ostaju brojevi $a_{i+1}, \ldots, a_j$ i je Slavko na potezu. Primetimo da je situacija ista kao pre Mirkovog poteza, samo što su Mirko i Slavko zamenili uloge (sada je Slavko prvi a Mirko drugi igrač). Prema tome, ukoliko obojica igraju optimalno do kraja igre, zbir brojeva koje uzima Mirko je $a_i + s(i+1, j)$ a Slavko $m(i+1, j)$. Slično, ukoliko se Mirko odluči za a_j u prvom potezu, njegov zbir je $a_j + s(i, j-1)$ dok je Slavkov $m(i, j-1)$. Prema tome, dobijamo sledeće rekurentne izraze:

$$\begin{aligned} m(i,j) &= \max\{a_i + s(i+1,j), a_j + s(i,j-1)\} \\ s(i,j) &= \begin{cases} m(i+1,j) & , a_i + s(i+1,j) > a_j + s(i,j-1) \\ m(i,j-1) & , \text{u suprotnom} \end{cases} \end{aligned} \tag{4.3.1}$$

Kao startne vrednosti možemo uzeti $m(i,i) = a_i$ i $s(i,i) = 0$ (ukoliko je na tabli zapisan samo jedan broj, u prvom potezu ga Mirko uzima i završava igru). Konačno rešenje problema je naravno $s(1,n)$.

Za implementaciju ovog rešenja potrebno je pamtiti dva dvodimenzionalna niza $m(i,j)$ i $s(i,j)$, što zahteva $\mathcal{O}(n^2)$ memorije. Kao i kod problema ranca, i ovde je dovoljno pamtiti samo jednu vrstu matrica $s(i,j)$ i $m(i,j)$, pa su sada memorijski zahtevi $\mathcal{O}(n)$. Složenost ovog rešenja je $\mathcal{O}(n^2)$.

Implementacija:

```
program Igra;

function max(a,b:integer):integer; begin
  if a>b then max:=a else max:=b;
end;

var m,s:array [1..130,1..130] of integer;
    a:array [1..130] of integer;
    i,j,n,r,zbir:integer;

begin
  readln(n);
  for i:=1 to n do
    read(a[i]);

  fillchar(m,sizeof(m),0);
  fillchar(s,sizeof(s),0);

  for i:=1 to n do begin
    m[i,i]:=a[i]; s[i,i]:=0;
  end;

  for i:=n-1 downto 1 do begin
    for j:=i+1 to n do begin
       m[i,j]:=max((a[i]+s[i+1,j]),(a[j]+s[i,j-1]));
       zbir:=0;
       for r:=i to j do zbir:=zbir+a[r];
       s[i,j]:=zbir-m[i,j];
    end;
  end;
```

```
  writeln(m[1,n]);
end.
```

4.4 Bankomati

U novoj eri elektronskog poslovanja, bankomati sve više postaju svakodnevna pojava. Banke žele da se klijenti što krace zadržavaju kod bankomata, kako bi opslužili što više ljudi, i zbog toga zahtevaju da se traženi iznos novca klijentima isplacuje sa što manje novčanica. Medutim, bankomate su programirali lenji programeri koji su upotrebili najjednostavniji algoritam tako da se u svakom njegovom koraku bira najveća novčanica koja je manja ili jednaka od preostalog iznosa. Na primer, za sistem sa novčanicama od 1, 7 i 10 dinara, iznos od 14 dinara biće isplaćen pomoću jedne novčanice od 10 i 4 novčanice od 1 dinara, ukupno 5 novčanica. Ovo ne odgovara zahtevu banke, jer se 14 dinara može isplatiti pomoću samo dve novčanice od 7 dinara.

Vaš zadatak je da napišete program koji ce za dati sistem od $n \leq 50$ novčanica, koji uvek sadrži i apoen od jednog dinara, naći najmanji iznos koji se može isplatiti u manje novčanica od broja novčanica koji bi bankomat iskoristio. Vrednost najveće novčanice je manja ili jednaka 500000.

Uputstvo: Neka su $n_1, \ldots, n_k$ ($n_1 < n_2 < \cdots < n_k$ iznosi na novčanicama koje poseduje bankomat. Takođe neka je $o(x)$ optimalan broj novčanica potreban da se isplati suma x a $g(x)$ broj novčanica koje isplaćuje bankomat. Pronaćićemo rekurentne formule na osnovu kojih se računaju brojevi $o(x)$ i $g(x)$, a zatim ćemo naći minimalni broj x takav da je $g(x) > o(x)$. Dokazaćemo da ukoliko je $g(x) = o(x)$ za svako $x \leq M$, za neki prirodan broj M (koji zavisi od $n_1, \ldots, n_k$), da je tada $g(x) = o(x)$ za svako $x \in \mathbb{N}$.

Rešenje: Pronađimo najpre rekurentnu formulu za $o(x)$. Posmatrajmo optimalnu isplatu $x = p_1 + \ldots + p_l$ sume x pri čemu je $p_i \in \{n_1, \ldots, n_k\}$ za svako $i = 1, \ldots, l$. Neka je $p_l = n_j$. Tada je $x - n_j = p_1 + \ldots + p_{l-1}$ optimalna isplata sume $x - n_j$ sa ukupno $l - 1$ novčanica (ukoliko bi postojao način da se suma $x - n_j$ isplati pomoću manje od $l - 1$ novčanica, tada bi dodavanjem nočanice n_j dobili isplatu sume x sa manje od l novčanica, što je nemoguće). Prema tome, imamo da važi $l = o(x) = o(x - n_j) + 1$. Slično kao i u predhodnim primerima dobijamo da indeks j treba izabrati tako da je zbir na desnoj strani predhodne jednakosti minimalan. Prema tome, imamo sledeću rekurentnu jednačinu:

$$o(x) = 1 + \min_{n_j \leq x} o(x - n_j) \tag{4.4.1}$$

Startna vrednost je u ovom slučaju $o(0) = 0$. Pošto je $n_1 = 1$ prema uslovu zadatka, matematičkom indukcijom lako pokazujemo da primenom startnog uslova $o(0) = 0$ i rekurentne relacije (4.4.1) možemo izračunati vrednosti $o(x)$ za svako $x \in N$.

Konstuišimo sada rekurentnu formulu za $g(x)$. Prema uslovu zadatka, bankomat u bira najveću novčanicu koja je manja ili jednaka datom iznosu. Neka je to nočanica

n_j. Nadalje bankomat postupa na isti način, ali sa iznosom $x - n_j$. Prema tome važi sledeća rekurentna formula:

$$g(x) = g(x - n_j) + 1, \quad n_j = \max_{n_i \leq x} n_i \tag{4.4.2}$$

Startna vrednost je i u ovom slučaju $g(0) = 0$.

Prema tome, primenjujući izraze (4.4.1) i (4.4.2) možemo da računamo vrednosti $g(x)$ i $o(x)$ redom za $x = 1, 2, 3, 4, \ldots$ sve dok ne pronađemo broj x_0 takav da je $o(x_0) < g(x_0)$. Međutim, može se desiti da takav broj ne postoji, pa prema tome moramo znati do kog broja treba vršiti pretragu. Konkretno, treba naći takav broj M (M naravno zavisi od $n_1, \ldots, n_k$) takav da ukoliko je $g(x) = o(x)$ za svako $x \leq M$, da je onda i $g(x) = o(x)$ za svako $x \in \mathbb{N}$.

Dokažimo da $M = 2n_k$ zadovoljava ovaj uslov. Predpostavimo suprotno, neka je x_0 minimalan broj takav da je $g(x_0) > o(x_0)$ i neka važi $x_0 > M$. Imamo da je $g(x) = o(x)$ za svako $x = 0, 1, \ldots, x_0 - 1$. Na osnovu (4.4.1) i (4.4.2) imamo da je:

$$o(x_0) = o(x_0 - n_j) + 1, \quad g(x_0) = g(x_0 - n_k) + 1 \tag{4.4.3}$$

Iz (4.4.3) direktno dobijamo da je $o(x_0 - n_j) < g(x_0 - n_k)$ Ukoliko je $j = k$ iz $o(x_0 - n_k) = g(x_0 - n_k)$ dobijamo da je $o(x) = g(x)$, što je kontradikcija. Pretpostavimo sada da je $j \neq k$. Pošto je $o(x_0 - n_j) = g(x_0 - n_j)$ i važi

$$x_0 - n_j \geq x_0 - n_{k-1} > M - n_{k-1} = n_k,$$

imamo da je vrednost prve novčanice u optimalnoj isplati sume $x_0 - n_j$ jednaka n_k. Prema tome važi:

$$o(x_0 - n_j) = o(x_0 - n_j - n_k) + 1$$

Na osnovu (4.4.1) imamo da je:

$$o(x_0 - n_k) \leq o(x_0 - n_j - n_k) + 1 = o(x_0 - n_j) < g(x_0 - n_k)$$

što predstavlja kontradikciju sa predpostavkom o minimalnosti broja x_0. Ovim smo dokazali da je dovoljno izračunati $g(x)$ i $o(x)$ za svako $x = 1, \ldots, M$ da bi mogli da utvrdimo da li postoji broj x_0 takav da je $g(x_0) > o(x_0)$.

Za implementaciju izloženog rešenja potrebna su nam dva niza g i o od po $M = n_k + n_{k-1}$ elemenata. Međutim, primetimo da je u izrazima (4.4.1) i (4.4.2) za računanje x-tog clana nizova g i o potrebno tačno n_k prethodnih članova svakog niza. Prema tome, dovoljno je pamtiti po n_k članova svakog niza.

Kako je za primenu rekurentne formule (4.4.1) potrebno maksimalno k upoređivanja, dok za (4.4.2) je potrebno samo jedno sabiranje sledi da je ukupna složenost ovog metoda jednaka $\mathcal{O}(kM) = \mathcal{O}(k(n_k + n_{k-1}))$.

Implementacija:

```
program Bankomati;

Const MaxN=1000000;
```

```
var d1,g1,m,ind:array [0..MaxN] of longint;
    tmp,a:array [1..51] of longint;
    n:longint;

procedure sort(l,r: longint); var i,j,x,y: longint; begin
  i:=l; j:=r; x:=tmp[(l+r) DIV 2];
  repeat
    while tmp[i]<x do i:=i+1;
    while x<tmp[j] do j:=j-1;
    if i<=j then
    begin
      y:=tmp[i]; tmp[i]:=tmp[j]; tmp[j]:=y;
      i:=i+1; j:=j-1;
    end;
  until i>j;
  if l<j then sort(l,j);
  if i<r then sort(i,r);
end;

var i,k,l,v,res,maxp:longint;
    bl,nema:boolean;

begin
  readln(n);
  for i:=1 to n do read(a[i]);
  readln;

  fillchar(ind,sizeof(ind),0);

  for i:=1 to n do tmp[i]:=a[i];
  for i:=1 to n do ind[tmp[i]]:=i;

  sort(1,n);
  for i:=1 to 2*tmp[n] do d1[i]:=i;

  bl:=true;

  m[0]:=1;

  for i:=1 to 2*tmp[n] do
    if ind[i]=0 then m[i]:=m[i-1] else m[i]:=a[ind[i]];

  for i:=1 to 2*tmp[n] do
    g1[i]:=g1[i-m[i]]+1;

  d1[0]:=0;
  g1[0]:=0;

  for v:=0 to 2*tmp[n] do begin
    for l:=1 to n do
      if (a[l]<=v) then if (d1[(v-a[l])]+1<=d1[v]) then d1[v]:=d1[(v-a[l])]+1;
    bl:=g1[v]=d1[v];
    if not bl then break;
  end;
  if not bl then writeln(v) else writeln(0);
end.
```

Napomena: Ukoliko se izostavi minimalnost broja x_0, postoji rešenje ovog problema u vremenu $\mathcal{O}(k^2)$. Posmatrajmo isplatu sume n_t koju vrši bankomat pomoću novčanica $n_1, \ldots, n_{t-1}$. Neka je $b_{t,j}$ broj nočanica n_j koje učestvuju u toj isplati $(1 \leq j \leq t-1)$. Tada važi:

$$a_t = b_{t,1}n_1 + \ldots + b_{t,t-1}n_{k-1}$$

Posmatrajmo brojeve:

$$P_{t,j} = n_j + \sum_{i=j}^{t-1} b_{t,i}n_i \tag{4.4.4}$$

Može se pokazati [2] da ukoliko postoji broj x_0 takav da je $g(x_0) > o(x_0)$, onda bar jedan od brojeva $P_{k,j}$ zadovoljava da je isplata sume $P_{k,j}$ koju daje bankomat lošija od isplate definisane izrazom (4.4.4). Prema tome, ovaj broj zadovoljava uslove zadatka. Za generisanje brojeva $P_{k,j}$ kao i za proveru da li neki od njih zadovoljava uslove zadatka, potrebno je ukupno $\mathcal{O}(k^2)$ aritmetičkih operacija.

Implementaciju ovog rešenja problema prepuštamo čitaocu.

4.5 Cveće i vaze

Želite da aranžirate izlog u prodavnici cveća na najbolji mogući način. Imate F vrsta cveća i V vaza, pri čemu je $V \geq F$. Vaze su pričvršćene na stalak i numerisane redom brojevima $1, 2, \ldots, V$. Ukoliko vrstu cveća sa rednim brojem i stavite u vazu sa rednim brojem j, estetski izgled koji na ovaj način postižete jednak je a_{ij} (pritom ovaj može biti i negativan). Cveće svake vrste može biti postavljeno najviše u jednu vazu i ukoliko je cveće i-te vrste postavljeno u k-tu vazu, cveća $i+1$-ve,...,n-te vrste moraju biti postavljena u neku od vaza $k+1, \ldots, v$. Odrediti način na koji treba rasporediti cveće po vazama tako da ukupan estetski izgled bude maksimalan.

Uputstvo: Označimo sa $E(i, j)$ maksimalni estetski izgled koji je moguće dobiti sa prvih i vrsta cveća i prvih j vaza. Odredićemo rekurentnu formulu za niz $E(i, j)$.

Rešenje: Razlikovaćemo dva slučaja, ukoliko cveće i-te vrste stavljamo u j-tu vazu i ukoliko to nije slučaj. U prvom slučaju, imamo da nam ova kombinacija povećava ukupan estetski izgled za a_{ij}. Sada je potrebno cveće $1, \ldots, i-1$ rasporediti u vaze $1, \ldots, j-1$ i pritom maksimizovati ukupni estetski izgled. Ovaj maksimalni estetski izgled jednak je $E(i-1, j-1)$.

U drugom slučaju, ukoliko se cveće i-te vrste ne stavi u j-tu vazu, onda ili j-ta vaza ostaje prazna, ili cveće vrste i ne stavljamo ni u jednu od predhodnih vaza. Maksimalni estetski izgledi u ova dva slučaja jednaki su redom $E(i, j-1)$ i $E(i-1, j)$. Prema tome, imamo da važi sledeća rekurentna formula:

$$E(i, j) = \max\{E(i-1, j-1) + a_{ij}, E(i-1, j), E(i, j-1)\} \tag{4.5.1}$$

Startne vrednosti su $E(i, 0) = E(0, j) = 0$ za svako $1 \leq i \leq F$ i $1 \leq j \leq V$. Za rekonstrukciju optimalnog rasporeda, dovoljno je u niz $P(i, j)$ pamtiti koji je od tri broja sa desne strane izraza (4.5.1) najveći.

[2] *The Fifth Tournament of Towns*, Moscow, 1994

Ukoliko se traži rekonstrukcija optimalnog rasporeda, ovo rešenje zahteva pamćenje dva dvodimenzionalna niza (E i P) od po FV elemenata. Ukoliko to nije slučaj, onda pošto se u izrazu (4.5.1) prilikom izračunavanja $E(i,j)$ koriste vrednosti iz i-te i $i-1$-ve kolone matrice E, dovoljno je pamtiti samo poslednju vrstu niza E, odnosno ukupno V elemenata.

Složenost izloženog rešenja je u oba slučaja (bilo da se traži rekonstrukcija optimalnog rasporeda ili ne) jednaka $\mathcal{O}(FV)$.

Implementaciju izloženog rešenja problema prepuštamo čitaocu.

4.6 Z-Draguljčići

Poreklo zadatka: Z-trening [28].

U nekoj azbuci postoji S slova ($1 \leq S \leq 100$). Sva slova ove azbuke se sastavljaju pomoću dragulja, i pritom je broj dragulja potreban za sastavljanje i-tog slova jednak d_i.

Potrebno je pronaći ukupan broj reči (reč je bilo koji niz slova) koje se mogu sastaviti pomoću N dragulja ($N \leq 10000$).

Uputstvo: Neka je $a(i,j)$ broj reči koji može da se sastavi pomoću i dragulja, pri čemu je j-to slovo poslednje u reči. Potrebno je pronaći rekurentnu formulu za računanje elemenata niza $a(i,j)$.

Rešenje: Pošto je j-ta reč poslednja, ostatak reči je potrebno formirati pomoću $i-d_j$ dragulja (naravno, uz predpostavku da je $i \geq d_j$, u suprotnom je trivijalno $a(i,j)=0$). Ukupan broj reči koje je moguće formirati pomoću $i-d_j$ dragulja je $\sum_{k=1}^{S} a(i-d_j,k)$ (poslednje slovo ostatka reči, tj predposlednje slovo reči može biti bilo koje). Prema tome, važi sledeća rekurentna formula:

$$a(i,j) = \sum_{k=1}^{S} a(i-d_j,k) \tag{4.6.1}$$

uz startne vrednosti $a(0,0)=1$ i $a(0,j)=0$ za $1 \leq j \leq S$. Ove startne vrednosti obuhvataju slučaj prazne reči (postoji samo jedna reč koja se može sastaviti bez dragulja, a to je prazna reč). Primetimo za je izrazu (4.6.1) suma na desnoj strani funkcija od $i-d_j$. Ovaj izraz predstavlja sumu elemenata $i-d_j$-te vrste niza $a(i,j)$, i označićemo ga sa $s(i-d_j)$. Sada je:

$$\begin{aligned} a(i,j) &= \begin{cases} s(i-d_j), & i \geq d_j \\ 0, & \text{u suprotnom} \end{cases} \\ s(i) &= \sum_{j=1}^{S} a(i,j) \end{aligned} \tag{4.6.2}$$

Najpre se za konkretno i izračunaju sve vrednosti $a(i,j)$, a zatim se računa njihova suma $s(i)$. Konačno rešenje se dobija kao: $s(1)+\ldots+s(N)$.

Primetimo da ovo rešenje zahteva pamćen je nizova $a(i,j)$ i $s(i)$, za šta je potrebno ukupno $\mathcal{O}(NS)$ memorije. Međutim moguće je izbeći pamćenje niza $a(i,j)$ i direktno računati elemente niza $s(i)$ pomoću sledećeg izraza koji se dobija iz (4.6.2):

$$s(i) = \sum_{d_j \leq i} s(i - d_j) \tag{4.6.3}$$

Kod ove varijante potrebno nam je samo $O(N)$ memorije. Takođe primetimo da su $s(i)$, kao i rešenje zadatka veoma veliki, čak i za male vrednosti ulaznih parametara. Zato je neophodno implementirati neku od struktura za rad sa velikim brojevima. Npr. ukoliko je $S = 100$, $N = 10000$ i $d_i = 1$ za svako $i = 1, \ldots, S$ (slučaj u kome se dobija najveći broj reči), dobijamo da je ukupan broj reči jednak približno $1.995 \cdot 10^{3010}$.

Što se tiče složenosti ovog rešenja, bilo da se koriste izrazi (4.6.2) ili izraz (4.6.3), složenost je jednaka $\mathcal{O}(NS)$.

Da bi izbegli korišćenje strukture podataka za manipulaciju sa velikim brojevima, u implementaciji ćemo rešenje određivati po modulu 8192.

Implementacija:

```
program Z-Draguljcici Const Modul=8191; Const MaxM=105; Const
MaxN=10005;

type tniz=array [0..MaxM] of longint; type tmat=array [0..MaxN] of
tniz;

var a:tmat ;
    d,s:array [0..MaxN] of longint;
    n,m,i,j,res :longint;

begin
  readln(n,m);
  for i:=1 to m do readln(d[i]);

  fillchar(s,sizeof(s),0);
  fillchar(a,sizeof(a),0);

  s[0]:=1;  a[0][0]:=1;

  for i:=1 to n do begin
    for j:=1 to m do
      if d[j]<=i then a[i][j]:=s[i-d[j]];
    for j:=1 to m do begin
      s[i]:=s[i]+a[i][j];
      s[i]:=s[i] and Modul;
    end;
  end;

  res:=0;
  for i:=1 to n do begin
    res:=res+s[i];
    res:=res and Modul;
  end;
```

```
  res:=res and Modul;
  writeln(res);
end.
```

4.7 Pljačka

Tri lopova su ukrala n dijamanata i treba da ih podele. Pritom je svaki lopov različito procenio svaki dijamant, tako da i-ti dijamant vredi p_{li} dinara, $1 \leq p_{li} \leq P$ prema proceni l-tog od lopova ($l = 1, 2, 3$, $i = 1, 2, \ldots, n$. Izvršiti podelu dijamanata tako da je ukupna vrednost dijamanata koje dobije l-ti lopov jednaka bar s_l dinara (prema njegovoj proceni), $1 \leq s_l \leq Pn$ kao i da je ukupan zbir procenjenih zarada maksimalan. Smatrati da je $P = 40$ a $n \leq 25$.

Uputstvo: Neka je $d(z_1, z_2, k)$ maksimalna zarada koju može da ostvari treći lopov pomoću prvih k dijamanata a da pritom prvi i drugi redom zarade z_1 odnosno z_2 dinara. Pošto je $p_{li} \leq P$ sledi da je $z_1, z_2 \leq Pn$. Prema tome, niz d ima maksimalno P^2n različitih elemenata. Pronađimo rekurentnu formulu za niz $d(z_1, z_2, k)$.

Rešenje: Ukoliko k-ti dijamant uzme prvi lopov, onda on mora zaraditi tačno $z_1 - a_k$ dinara, a drugi z_2 dinara uzimajući neke od predhodnih dijamanata. U tom slučaju maksimalna zarada trećeg lopova biće $d(z_1 - p_{1k}, z_2, k - 1)$. Slično dobijamo da ukoliko drugi lopov uzme k-ti dijamant, maksimalna zarada trećeg je $d(z_1, z_2 - p_{2k}, k - 1)$. Napokon ukoliko ovaj dijamant uzme treći lopov, prva dva moraju zaraditi z_1 odnosno z_2 dinara uzimajući prvih $k - 1$ dijamanata dok je zarada trećeg $d(z_1, z_2, k-1) + p_{3k}$. Prema tome, imamo da važi sledeća rekurentna formula:

$$\begin{aligned} d(z_1, z_2, k) = \max\{&d(z_1, z_2, k-1) + p_{3k}, \\ &d(z_1 - p_{1k}, z_2, k-1), \\ &d(z_1, z_2 - p_{2k}, k-1)\} \end{aligned} \tag{4.7.1}$$

Startne vrednosti za niz d su:

$$d(z_1, z_2, 0) = \begin{cases} 0, & (z_1, z_2) = (p_{11}, 0) \\ 0, & (z_1, z_2) = (0, p_{21}) \\ p_{31}, & z_1 = z_2 = 0 \\ -\infty, & \text{u suprotnom} \end{cases} \tag{4.7.2}$$

Primetimo da je u izrazu (4.7.2) $d(z_1, z_2, 0) = -\infty$ kada ne postoji podela tako da prvi lopov zaradi z_1 a drugi z_2 dinara. Ovaj zaključak se može proširiti i na $d(z_1, z_2, k)$, i upravo zato je vrednost $-\infty$ odabrana za ovaj slučaj. Naravno, u implementaciji bi trebalo koristiti negativan broj dovoljno veliki po apsolutnoj vrednosti (npr. $-Pn$, pošto je maksimalna moguća zarada jednaka Pn). Vrednosti niza $d(z_1, z_2, k)$ potrebno je izračunati za sve $0 \leq z_1, z_2 \leq Pn$ kao i $k = 1, 2, \ldots, n$.

Za konačno rešenje problema potrebne su vrednosti $d(z_1, z_2, n)$ za $z_1 \geq s_1$ i $z_2 \geq s_2$. Maksimalan zbir procenjenih zarada lopova lako dobijamo pomoću:

$$\begin{aligned} D = \max\{&z_1 + z_2 + d(z_1, z_2, n) \mid s_1 \leq z_1 \leq Pn, \\ &s_2 \leq z_2 \leq Pn,\ d(z_1, z_2, n) \geq s_3\} \end{aligned} \tag{4.7.3}$$

Ukoliko je $D = -\infty$ (ili je skup čiji se maksimum određuje na desnoj strani izraza (4.7.3) prazan), sledi da ne postoji podela koja bi zadovoljila sva tri lopova po pitanju zarade. U suprotnom, ukoliko je $D > -\infty$, raspodela dijamanata pri kojoj se dostiže maksimum D može se rekonstruisati rekurzivno. U svakom koraku se ispituje koja je od tri vrednosti na desnoj strani izraza (4.7.1) maksimalna, odnosno koji lopov uzima k-ti dijamant.

Ako se ne zahteva rekonstrukcija raspodele dijamanata, onda je dovoljno pamtiti vrednosti $d(z_1, z_2, k-1)$ da bi se izračunale odgovarajuće vrednosti $d(z_1, z_2, k)$, dakle samo P^2 vrednosti. U suprotnom je potrebno pamtiti ceo niz d, odnosno $P^2 n$ elemenata. Alternativno, možemo pamtiti u nizu $p(z_1, z_2, k)$ koji lopov dobija k-ti dijamant. Složenost ovog rešenja je $\mathcal{O}(P^2 n)$.

Implementaciju izloženog rešenja problema prepuštamo čitaocu.

4.8 Krug

Na kružnici je dato $2k$ tačaka $A_1, \ldots, A_{2k}$ pri čemu su raspoređene bas tim redosledom u smeru kazaljke na satu. Krug je podeljen tetivama čija su temena date tačke, pri čemu je svaka tačka teme tačno jedne tetive. Ove tetive dele krug na određeni broj delova P. Naći broj načina N na koji možemo konstruisati tetive tako da je broj delova P minimalan.

Uputstvo: Minimalni broj delova P se dobija kada se nikoje dve tetive ne seku i on je jednak $P = k + 1$. Neka je $N(k)$ traženi broj kombinacija. Uočićemo tačku A_1 i utvrditi koje su sve mogućnosti za drugo teme tetive čije je prvo teme tačka A_1.

Rešenje: Dokažimo najpre da je minimalan mogući broj delova na koje se razbija krug konstruisanim tetivama jednak $P = k + 1$ i da se ovaj broj dostiže akko se nikoje dve tetive ne seku. Konstruišimo tetive redom, jednu po jednu i posmatrajmo kako se menja broj delova na koje se krug razbija. Na početku je broj delova jednak 1 (ceo krug bez tetiva). Kad konstrušemo prvu tetivu, broj delova postaje jednak $2 = 1 + 1$. Predpostavimo sada da smo konstruisali i tetiva i konstruišimo $i+1$-vu. Ukoliko ona ne seče nijednu drugu tetivu, sledi da cela pripada samo jednom delu. Ona ovaj deo deli na dva, pa smo na ovaj način dobili još jedan deo. Ukoliko tetiva seče l drugih tetiva, ona prolazi kroz ukupno $l+1$ delova, i svaki deli na dva. Prema tome dobili smo još $l + 1$ novih delova, što je više nego u prvom slučaju. Znači najmanji broj delova, $P = k + 1$ dobija se ukoliko se tetive ne seku.

Pronađimo sada rekurentnu formulu za broj $N(k)$ načina da se krug podeli na $k + 1$ delova, ili ekvivalentno da se povuku k tetiva koje se međusobno ne seku. Neka je tačka A_1 spojena sa tačkom A_p. Primetimo da tetiva A_1A_p deli krug na dva dela, pri čemu su tačke $A_2, \ldots, A_{p-1}$ u jednom a $A_{p+1}, \ldots, A_{2k}$ u drugom delu. Jasno je da ne može postojati tetiva čije je jedno teme u jednom a drugo u drugom delu. Dalje primetimo da ukoliko je p neparan broj, u prvom delu ima ukupno $p-2$ tačke što je neparan broj, pa je odgovarajuće spajanje tetivama nemoguće. Ukoliko je $p = 2i$ paran broj, tetive u svakom od delova je moguće povući redom na $N(i-1)$

odnosno $N(k-i)$ načina. Broj i, odnosno tačku A_p možemo odabrati proizvoljno. Prema tome, imamo da važi sledeća rekurentna formula:

$$N(k) = \sum_{i=1}^{k} N(k-i)N(i-1) \tag{4.8.1}$$

Startna vrednost za niz $N(k)$ je $N(0) = 1$.

Ovo rešenje zahteva pamćenje jednog niza N dužine k. Uočimo da $N(k)$ jako brzo raste kad k raste. Već za $k = 20$ dobijamo $N(20) = 6564120420$ što izlazi iz opsega 32-bitnih celih brojeva. Znači, za izračunavanje vrednosti $N(k)$ za $k \geq 20$ potrebno je korišćenje neke strukture za operacije sa velikim brojevima. Složenost ovog rešenja je $\mathcal{O}(k^2)$.

Implementacija:

```
Const MaxN=300; type tniz=array [0..MaxN] of int64;

var s:tniz;
    k,n,i,j:longint;
    sum:int64;

begin
  readln(k);
  n:=2*k;  s[0]:=1;  i:=2;

  while i<=n do begin
    sum:=0;
    for j:=1 to (i div 2) do
      sum:=sum+s[i-2*j]*s[2*(j-1)];
    s[i]:=sum;
    i:=i+2;
  end;

  writeln(s[n],' ',k+1);
end.
```

Napomena: Može se dokazati da je opšti član niza $N(k)$ jednak:

$$N(k) = \frac{1}{k+1}\binom{2k}{k}$$

Ovi brojevi su poznati kao Katalanovi brojevi.

4.9 Lift

U nekoj zgradi koja ima ukupno M spratova postoji stari i dotrajali lift. Pretpostavimo da kompanija koja je vlasnik zgrade ima n radnika i da u jednom trenutku svi oni žele da se popnu na podkrovlje (sprat iznad M-tog sprata). Pritom je i-ti radnik u posmatranom trenutku na s_i-tom spratu a lift je u prizemlju. U zgradi inače postoje i stepenice, ali se do potkrovlja može doći isključivo liftom. Ukupan broj stajanja lifta na putu do potkrovlja je S. Ukoliko lift ne stane na sprat s_i, i-ti

radnik mora stepenicama da dođe do najbližeg sprata na koji lift staje. Odrediti na kojih S spratova lift treba da stane tako da je broj spratova koje radnici trebaju preći stepenicama minimalan.

Uputstvo: Neka je $m(i,j)$ minimalan broj spratova koje bi radnici trebalo da pređu stepenicama pri čemu lift staje na ukupno i spratova a j je poslednji sprat na koji lift staje. Pronađimo rekurentnu formulu za $m(i,j)$

Rešenje: Posmatrajmo optimalno stajanje lifta tako da on staje i puta i da su predposlednji i poslednji sprat na koje lift staje redom k-ti i j-ti. Radnici koji se nalaze na spratovima ispod k-tog sigurno neće ući u lift na j-tom spratu jer im je k-ti sprat bliži. Radnici koji su iznad k-tog sprata, ićiće stepenicama ili do k-tog, ili do j-tog, zavisno koji im je sprat bliži, odnosno svi radnici p za koje važi $s(p) \leq \frac{k+j}{2}$ ićiće do k-tog a ostali do j-tog sprata. Neka je

$$st(i,j) = \sum_{i<s(p)\leq j} \min\{s(p)-i, j-s(p)\} \tag{4.9.1}$$

Izraz $st(i,j)$ predstavlja ukupan broj spratova koji pešače radnici koji su između i-tog i j-tog sprata da bi došli do bližeg od ta dva sprata. Uočimo da je ukupan broj spratova koji prelaze radnici koji su ispod k-tog sprata jednak $m(i-1,k)-st(k,+\infty)$. Radnici koji su iznad k-tog sprata prelaze ukupno $st(k,i) + st(i,+\infty)$ spratova pešaka. Broj k naravno treba odabrati tako da se ukupan broj pređenih spratova minimizuje, pa prema tome važi sledeća rekurentna jednačina:

$$m(i,j) = \min_{k=i-1,i,\ldots,j}\{m(i-1,k) - st(k,+\infty) + st(k,i) + st(i,+\infty)\} \tag{4.9.2}$$

Startne vrednosti za niz $m(i,j)$ su $m(0,k) = st(0,+\infty)$ za svako $k = 0,1,\ldots,M$ (pošto nema stajanja, svi radnici moraju najpre da siđu u prizemlje a zatim da se ukrcaju u lift). Konačno rešenje problema se dobija kao:

$$\max_{0\leq t\leq M} m(S,t) = m(S,t_0)$$

Poslednji sprat na koji lift staje je t_0. Rešenje nadalje možemo rekonstruisati obrnutom primenom rekurentne formule (4.9.2) odnosno nalaženjem indeksa k takvog da se izraz na desnoj strani formule (4.9.2) minimizuje. Neka je taj indeks k_0, i on predstavlja indeks predhodnog sprata na koji lift staje. Na isti način se rekonstruišu i ostali spratovi. Alternativno, možemo u nizu $p(i,j)$ pamtiti indeks k_0, čime uprošćavamo rekonstrukciju rešenja.

Što se tiče memorijskih zahteva, ovo rešenje zahteva pamćenje niza $m(i,j)$ i eventualno niza $p(i,j)$. Znači potrebna količina memorije je $\mathcal{O}(SM)$. Pritom, ukoliko rekonstrukcija rešenja nije potrebna, dovoljno je pamtiti poslednju izračunatu kolonu matrice $m(i,j)$, tj dovoljno je $\mathcal{O}(M)$ memorije. Pošto se pri svakoj primeni izraza (4.9.2) izračunava vrednost funkcije st na osnovu (4.9.1), ukupna složenost metoda je $\mathcal{O}(SM^2n)$. Pritom, vrednosti ove funkcije možemo prekalkulisati i smestiti u odgovarajući niz, za šta nam je potrebno $\mathcal{O}(SMn)$ operacija i još $\mathcal{O}(SM)$ memorije. Međutim, ukupna složenost rešenja u ovom slučaju redukuje se na $\mathcal{O}(SMn)$. Implementaciju izloženog rešenja problema prepuštamo čitaocu.

4.10 Sečenje štapića

Potrebno je iseći štapić dužine L na nekoliko delova. Radionica u kojoj se vrši sečenje naplaćuje svako pojedinačno sečenje srazmerno dužini štapića koji se seče. Lako se vidi da različiti načini sečenja različito koštaju.

Na primer, ako sečemo štapić dužine 10 na četiri dela čije su dužine redom $2, 3, 4$ i 1 i ako prvo presečemo štapić na dužini 5 od levog kraja a zatim svaki deo ponovo sečemo, ukupna cena koju plaćamo je $10 + 5 + 5 = 20$. Međutim, ukoliko sečemo najpre na dužini 2 od levog kraja, zatim veći štapić na dužini 3, itd... cena koju plaćamo je $10 + 8 + 5 = 23$.

Ako je data dužina štapa L, $L \leq 1000$ kao i rastojanja od levog kraja štapa na kojima treba izvršiti rez $d_1, \ldots, d_n$, $d_1 < d_2 < \cdots < d_n < L$, $n \leq 50$ naći minimalnu potrebnu cenu kao i način sečenja da bi se ta cena ostvarila.

Uputstvo: Neka je $c(i, j)$ minimalna cena potrebna da se iseče deo štapića od i-tog do j-tog mesta. Pronaćićemo rekurentnu formulu za niz $c(i, j)$.

Rešenje: Predpostavimo da se deo štapića najpre seče na k-tom mestu. Cena ovog sečenja je jednaka dužini dela štapića $d_j - d_i$. Posle sečenja imamo nova dva dela štapa, od i-tog do k-tog mesta i od k-tog do j-tog. Koje moramo nadalje da sečemo. Minimalna cena potrebna da se iseku ova dva dela štapića na odgovarajući način je $c(i, k) + c(k, j)$. Na osnovu ovoga, na sličan način kao u predhodnim problemima dobijamo da je:

$$c(i, j) = \min_{i<k<j} \{c(i, k) + c(k, j)\} + d_j - d_i \tag{4.10.1}$$

Startne vrednosti za niz $c(i, j)$ su $c(i, i+1) = 0$. Izračunavanje vrednosti elemenata matrice c pomoću jednačine (4.10.1) vrši se po dijagonalama paralelnim glavnoj dijagonali ove matrice.

Složenost ovog rešenja je $\mathcal{O}(n^3)$ a potrebna količina memorije $\mathcal{O}(n^2)$ ili $\mathcal{O}(n)$ ukoliko se ne zahteva rekonstrukcija. U drugom slučaju, u jednom nizu se pamti samo poslednja izračunata dijagonala matrice c.

Implementaciju izloženog rešenja problema prepuštamo čitaocu.

4.11 Particije prirodnih brojeva

Neka je dat prirodan broj n kao i brojevi $n_1, \ldots, n_k$ takvi da važi $n_1 + \ldots + n_k = n$ i $n_1 \geq n_2 \geq \ldots \geq n_k$. Tada uređenu k-torku $(n_1, \ldots, n_k)$ nazivamo *particijom* prirodnog broja n. Neka je $p(n)$ ukupan broj particija broja n. Na primer, za $n = 6$ imamo ukupno $p(6) = 11$ particija i one su:

$$\begin{aligned}&(6), (5, 1), (4, 2), (3, 3), (4, 1, 1), (3, 2, 1), (2, 2, 2), (2, 2, 1, 1),\\ &(2, 1, 1, 1, 1), (1, 1, 1, 1, 1, 1)\end{aligned} \tag{4.11.1}$$

Za unet prirodan broj $n \in \mathbb{N}$, $n \leq 1000$ naći broj particija $p(n)$.

Uputstvo: Označimo sa $p(n, m)$ broj particija broja n čiji je najveći član (n_1) jednak m.

Rešenje: Pokazaćemo da je broj particija broja n sa najvećim članom jednakim m jednak broju particija broja $n-m$ sa najvećim članom manjim ili jednakim m. Posmatrajmo jednu particiju $(n_1, \ldots, n_k)$ broja n pri čemu je $n_1 = m$. Tada je očigledno $(n_2, \ldots, n_k)$ particija prirodnog broja $n-m$. Takođe imamo da je $m = n_1 \geq n_2$. Prema tome, na ovaj način smo definisali jedno preslikavanje. Potrebno je dokazati da je ovo preslikavanje bijekcija. Pošto iz $(n_1, \ldots, n_k) \neq (n'_1, \ldots, n'_k)$ i $n_1 = n'_1 = m$ sledi da je $(n_2, \ldots, n_k) \neq (n'_2, \ldots, n'_k)$, posmatrano preslikavanje je $1-1$. Sa druge strane, ukoliko je $(n'_2, \ldots, n'_k)$ proizvoljna $k-1$-člana particija broja $n-m$ takva da je $n'_2 \leq m$, neposrednom proverom dobijamo da je $(m, n'_2, \ldots, n'_k)$ particija broja n sa najvećim članom jednakim m.

Na osnovu ovog razmatranja možemo zaključiti da niz $p(n,m)$ zadovoljava sledeću rekurentnu relaciju:

$$p(n,m) = \sum_{k=1}^{\min\{n-m,m\}} p(n-m,k) \tag{4.11.2}$$

Gornja granica u predhodnoj sumi je takođe ograničena brojem $n-m$ zato što očigledno nijedan član (pa ni najveći) ne može biti veći od ukupnog zbira svih članova $n-m$. Startne vrednosti u formuli (4.11.2) su:

$$p(m,m) = 1, \quad m \geq 1$$

Broj $p(n)$ na kraju dobijamo kao $p(n) = \sum_{m=1}^{n} p(n,m)$. Opisani metod očigledno radi u složenosti $\mathcal{O}(n^3)$ (pošto desnu stranu relacije (4.11.2) računamo u složenosti $\mathcal{O}(n)$) i zahteva $\mathcal{O}(n^2)$ memorije (niz $p(n,m)$).

Primetimo da desnu stranu relacije (4.11.2) računamo više puta za istu vrednost $n-m$. Da bi izbegli ova suvišna izračunavanja, označimo izraz sa desne strane (4.11.2) sa $\hat{p}(n-m,m)$. Drugim rečima, neka je:

$$\hat{p}(n,m) = \sum_{k=1}^{\min\{m,n\}} p(n,k) \tag{4.11.3}$$

Prema tome, na osnovu (4.11.2) važi $p(n,m) = \hat{p}(n-m,m)$ za svako $n, m \in \mathbb{N}$ pri čemu je $m \leq n$. Pri tome, primetimo da pod istim uslovima očigledno važi i sledeća rekurentna relacija $\hat{p}(n,m) = p(n,m) + \hat{p}(n,m-1)$:

$$\hat{p}(n,m) = \hat{p}(n-m,m) + \hat{p}(n,m-1) \tag{4.11.4}$$

Relacija (4.11.4) može se takođe za izračunavanje broja $p(n)$, pri čemu je sad $p(n) = \hat{p}(n,n)$. Startne vrednosti za primenu ove relacije su $\hat{p}(n,1) = p(n,1) = 1$ (postoji samo jedna particija broja n čiji je najveći član jednak n i ona je sastavljena od svih jedinica).

I u ovom slučaju potrebno nam je $\mathcal{O}(n^2)$ memorije, pri čemu je složenost izračunavanja sada redukovana na $\mathcal{O}(n^2)$. Uz ograničenje $n \leq 1000$, ova složenost je prihvatljiva za praktična izračunavanja.

Napomenimo samo da je $p(1000) = 2.40615 \cdot 10^{31}$ pa prema tome potrebno je koristiti neku od struktura za rad sa velikim brojevima [3].

Implementaciju izloženog rešenja problema prepuštamo čitaocu.

4.12 Različiti podnizovi

Poreklo zadatka: [20]

Neka su data dva niza a i b prirodnih brojeva redom dužina m i n. Kažemo da je niz a podniz niza b ukoliko se a dobija tako što se iz b izbrišu neki elementi, tj ukoliko postoji niz indeksa $p_1 < p_2 < \cdots < p_m$ takav da je $a(t) = b(p_t)$ za svako $t = 1, \ldots, m$. Niz indeksa $p_1, \ldots, p_m$ ne mora biti jedinstven. Naći broj pojavljivanja niza a u nizu b kao podniz. Drugim rečima, naći broj različitih nizova indeksa p takvih da važe uslovi $p_1 < p_2 < \cdots < p_m$ i $a(t) = b(p_t)$ za svako $t = 1, \ldots, m$.

Uputstvo: Ovaj problem je veoma sličan problemu nalaženja najdužeg zajedničkog podniza dva data niza. Neka je $B(i, j)$ broj pojavljivanja niza $(a(1), \ldots, a(i))$ u nizu $(b(1), \ldots, b(j))$. Pronađimo rekurentnu formulu za niz $B(i, j)$.

Rešenje: Neka je $1 \leq p_1 < \cdots < p_i \leq j$ niz indeksa takav da je $a(t) = b(p_t)$ za svako $t = 1, \ldots, i$. Razmotrićemo dva slučaja.

Najpre neka je $p_t = j$. Tada je $a(i) = b(j)$, pri čemu niz indeksa $p_1, \ldots, p_{i-1}$ predstavlja jedno pojavljivanje niza $(a(1), \ldots, a(i-1))$ u nizu $(b(1), \ldots, b(j-1))$. Obrnuto ako je $p_1, \ldots, p_{i-1}$ indeksni niz nekog pojavljivanja niza $(a(1), \ldots, a(i-1))$ u nizu $(b(1), \ldots, b(j-1))$ i ako je $a(i) = b(j)$, tada ukoliko definišemo $p_i = j$ dobijamo pojavljivanje niza $(a(1), \ldots, a(i))$ u nizu $(b(1), \ldots, b(j))$. Ovakvih pojavljivanja ima $B(i-1, j-1)$.

Ukoliko je $p_t < j$ odnosno $p_t \leq j-1$ dobijamo da je u ovom pojavljivanju ceo niz $(a(1), \ldots, a(i))$ sadržan u $(b(1), \ldots, b(j-1))$ pri čemu je p odgovarajući indeksni niz. Pritom i ovde važi obrnuti smer, proizvoljno pojavljivanje pojavljivanje (sa bilo kojim indeksnim nizom p) niza $(a(1), \ldots, a(i))$ u $(b(1), \ldots, b(j-1))$ je ujedno i pojavljivanje u nizu $(b(1), \ldots, b(j))$. Ovaj zaključak važi nezavisno od toga da li je $a(i) = b(j)$ ili je $a(i) \neq b(j)$.

Prema tome, imamo da važi sledeća rekurentna formula:

$$B(i,j) = \begin{cases} B(i-1, j-1) + B(i, j-1) & , a(i) = b(j) \\ B(i, j-1) & , a(i) \neq b(j) \end{cases} \tag{4.12.1}$$

Startne vrednosti su $B(0, 0) = 1$ kao i $B(i, 0) = 0$ za $i \geq 1$. Rešenje problema je naravno $B(m, n)$.

Ovo rešenje zahteva $\mathcal{O}(mn)$ memorije ukoliko se pamti ceo niz B ili $\mathcal{O}(n)$ ukoliko se pamti samo poslednja izračunata kolona matrice B. Složenost rešenja je naravno $\mathcal{O}(mn)$.

Napomenimo samo da vrednosti niza B za određene ulazne podatke mogu biti vrlo veliki brojevi (npr. ako je $a(i) = 1$ i $b(j) = 1$ za svako $i = 1, \ldots, m$ i $j = 1, \ldots, n$

[3] Pošto je broj cifara međurezultata maksimalno 31, ukupan broj **elementarnih operacija** na računaru je jednak 31000000 za šta je savremenim računarima potrebno manje od 1 sec.

imamo da je $B(i,j) = \binom{i}{j}$), pa je potrebno koristiti neku od struktura za rad sa velikim brojevima.

Implementaciju izloženog rešenja prepuštamo čitaocu.

4.13 Ukrcavanje trajekta

Poreklo zadatka: [20]

Trajekt se sastoji od dve trake dužine l metara ($l \leq 100$) na koje mogu da se parkiraju vozila. Pošto je širina trake jednaka širini vozila, ona moraju biti parkirana jedno iza drugog. U redu za ukrcavanje čeka n vozila ($n \leq 1000$), pri čemu je dužina i-tog vozila a_i centimetara ($a_i \in \mathbb{N}$, $a_i \leq A = 3000$). Vozila se moraju ukrcavati redom, u redosledu u kome čekaju. Radnik na trajektu može svako vozilo uputiti u jednu od dve trake, pri čemu je cilj ukrcati što više vozila. Pritom svako vozilo koje dođe na red mora biti ukrcano, ukoliko za to postoje mogućnosti (tj ima mesta na nekoj od traka). Naći maksimalan broj k takav da je moguće na opisani način ukrcati u trajekt prvih k vozila. Odrediti jedno od mogućih ukrcavanja ovih vozila.

Uputstvo: Neka je $V(l_1, i)$ logički niz koji ima vrednost $\top$ ukoliko je moguće ukrcati prvih i vozila tako da na prvoj traci ostane slobodno l_1 centimetara, a u suprotnom vrednost $\bot$. Rešenje problema se sada dobija kao

$$\max\{k \mid V(t,k) = \top, \text{ za neko } t \in \{1,2,\ldots,l\}\}.$$

Rešenje: Predpostavimo da postoji način za ukrcavanje vozila $1, 2, \ldots, i$ u trajekt. Razmotrićemo dva slučaja.

Neka se i-to vozilo ukrcava u 1. traku. Pošto je nakon ukrcanja ovog vozila slobodno l_1 centimetara, pre ukrcavanja mora biti slobodno $l_1 + a_i$ centimetara. Naravno, pre i-tog ukrcali smo sva vozila do $i-1$-vog, pa je prema tome $V(l_i + a_i, i-1) = \top$. Važi i obrat, ukoliko je $V(l_i + a_i, i-1) = \top$ sledi da i-to vozilo možemo ukrcati u 1. traku pa je i $V(l_1, i) = \top$.

Neka se sada i-to vozilo ukrcava u 2. traku. U drugoj traci je ukupno zauzeto $l_2 = \sum_{j=1}^{i} a_i - l_1$ centimetara. Ovaj broj mora biti ne veći od l. Pre ukrcavanja i-tog vozila, u 1. traci je bilo takođe l_1 slobodnih centimetara pa zaključujemo da mora da važi $V(l_1, i-1) = \top$. Važi i obrat. Ukoliko je $l_2 \leq l$ i $V(l_1, i-1) = \top$, i-to vozilo možemo ukrcati u drugu traku pa dobijamo da je i $V(l_1, i) = \top$.

Prema tome, ovim smo pokazali da važi sledeća rekurentna formula za niz $V(l_1, i)$:

$$V(l_1, i) = V(l_1 + a_i, i-1) \wedge (V(l_1, i-1) \vee l_{2i} \leq l) \tag{4.13.1}$$

gde je:

$$l_{2i} = s(i) - l_1, \quad s(i) = \sum_{j=1}^{i} a_i. \tag{4.13.2}$$

Startne vrednosti za $V(l_1, i)$ su $V(l, 0) = \top$ i $V(l_1, 0) = 0$ za $l_1 < l$ (ukoliko nemamo nijedno vozilo, prva traka je cela prazna). Sada samo treba pronaći maksimalnu vrednost indeksa i takvu da je $V(l_1, i) = \top$ bar za jednu vrednost l_1. Ova maksimalna vrednost rešenje problema.

Da bi rekonstruisali rešenje, potrebno je pamtiti u nizu $P(l_1, i)$ koji član u disjunkciji sa desne strane izraza (4.13.1) ima vrednost $\top$. Ukoliko su oba člana $\perp$ onda je $P(l_1, i) = 0$. Primetimo da je vrednost $V(l_1, i)$ potpuno određena pomoću vrednosti $P(l_1, i)$, pa je dovoljno pamtiti samo niz P. Rekonstrukcija se naravno obavlja rekurzivno kao i kod predhodnih problema.

Dakle ovo rešenje zahteva $\mathcal{O}(nl)$ memorije i ima složenost takođe $\mathcal{O}(nl)$. Ukoliko je potreban samo broj vozila a nije bitan raspored, onda je dovoljno pamtiti poslednju izračunatu kolonu niza V (niz P uopšte nije potrebno pamtiti), pa su memorijski zahtevi redukovani na $\mathcal{O}(l)$.

Implementaciju izloženog rešenja prepuštamo čitaocu.

4.14 Novopečeni bogataš

Lokalni novopečeni bogataš, kupio je poveliko kvadratno parče zemlje, na njemu želi da izgradi što je moguće veću kuću i to tako da njena osnova bude kvadratnog oblika sa stranama paralelnim stranama placa. Mapa imanja je sastavljena iz kvadratića dimenzija 1×1. Jedno drvo zauzima tačno jedan kvadratić. Pošto je imanje veliko, vrlo je teško naći najveći kvadrat čija je strana najduža na kome nema stabala. Dužina strane je broj kvadratića 1×1 duž njegove strane.

Potrebno je izračunati dužinu stranice najvećeg kvadrata koji ne sadrži ni jedno drvo. Predpostaviti da je veličina imanja $n \leq 100$ kvadrata.

Uputstvo: Neka je $K(i, j)$ maksimalna dimenzija kvadrata koji ne sadrži nijedno drvo i čiji je donji desni ugao kvadratić sa koordinatama (i, j). Potrebno je naći rekurentnu formulu za niz $K(i, j)$.

Rešenje: Ukoliko je na polju (i, j) locirano drvo, onda je trivijalno $K(i, j) = 0$. Predpostavimo sada da na polju (i, j) nema drveta. Posmatrajmo maksimalne kvadrate koji ne sadrže drvo i čiji su donji desni uglovi $(i-1, j)$, $(i, j-1)$ kao i $(i-1, j-1)$. Neka su dužine njihovih stranica redom k_1, k_2 i k_3. Tada je očigledno $K(i, j) \leq k_1, k_2, k_3 + 1$ jer bi u suprotnom odgovarajući kvadrat čija je stranica k_i, $i = 1, 2, 3$ mogao biti veći na račun kvadrata sa temenom (i, j). Neka je $k = \min\{k_1, k_2, k_3 + 1\}$. Dokažimo da je $K(i, j) \geq k$. Zaista, pošto je $k \leq k_1$ sva polja sa koordinatama $(i - p - 1, j - q)$ gde je $p, q \leq k$ ne sadrže drvo. Isto važi i za polja $(i - p, j - q - 1)$. Napokon, pošto je $k \geq k_3 + 1$ imamo da je i polje $(i - k, j - k)$ slobodno. Time smo dokazali da je $K(i, j) \geq k$. Prema tome važi sledeća rekurentna formula:

$$K(i,j) = \begin{cases} \min\{K(i-1,j), K(i,j-1), K(i-1,j-1)+1\}, & \text{na polju } (i,j) \\ & \text{nije drvo} \\ 0, & \text{u suprotnom} \end{cases}$$

Startne vrednosti za niz $K(i,j)$ su $K(1,j)$ i $K(i,1)$. Ukoliko je drvo locirano na nekoj od ovih lokacija, odgovarajuća vrednost elementa niza K je 0, u suprotnom je 1. Konačno rešenje dobijamo jednostavno kao $\max_{1\leq i,j\leq n} K(i,j)$.

Što se tiče memorijskih zahteva, ovo rešenje zahteva $\mathcal{O}(n^2)$ memorije (niz kojim se identifikuje da li je na polju (i,j) drvo i niz K). Medjutim, ukoliko se predpostavi da je broj drveća na plantaži jednak $\mathcal{O}(n)$, izloženo rešenje se može modifikovati tako da zahteva ukupno $\mathcal{O}(n)$ memorije. Modifikacija se postiže pamćenjem samo poslednje izračunate vrste niza K. U oba slučaja složenost rešenja je $\mathcal{O}(n^2)$.

Implementacija:

```
program sumadija;
 type matrica=array[1..100,1..100] of integer;
 var  a:matrica;
      n,i,p,j,x,y:integer;
function  min(a,b,c:integer):integer;
 var p:integer;
 begin
   if a>b then p:=b else p:=a;
   if c>p then min:=p else min:=c;
 end;
 begin
   write('Unesi velicinu imanja');     readln(n);
   write('Unesite broj drveca na imanju');   readln(p);
   write('unesite koordinate drveca');
   for i:=1 to n do
       for j:=1 to n do   a[i,j]:=1;
   for i:=1 to p do
         begin
           readln(x,y);      a[x,y]:=0;
         end;
   x:=0;
   for i:=2 to n do for j:=2 to n do
     begin
         a[i,j]:=a[i,j]+a[i,j]*min(a[i-1,j],a[i-1,j-1],a[i,j-1]);
         if a[i,j]>x then x:=a[i,j];
     end;
   writeln('Duzina najvece dimenzije kuce je P= ',x);
 end.
```

4.15 Maksimalna suma nesusednih u nizu

Poreklo zadatka: [25].

Dat je niz A od N celih brojeva. Odrediti podniz niza A čiji zbir elemenata je maksimalan, a u kome nema susednih elemenata niza A. Smatrati da prazan niz ima zbir elemenata nula.

Rešenje: Neka je za niz A poznat jedan takav podniz $P = P(A)$. Ako element a_N ne pripada podnizu P, onda je P optimalni podniz i za niz A_{N-1}. Ako element a_M pripada podnizu P, onda je podniz P bez poslednjeg elementa (tj. bez a_N) optimalan za niz A_{N-2}. U protivnom bi bilo moguće poboljšati optimalni podniz P niza A, tako što se na bolji podniz niza A_{N-2} dopiše broj a_N. Ustanovili smo optimalnost podstrukture problema.

Rešenje za nizove A_0 i A_1 su trivijalna. Za X od 2 do N, $P(A_X)$ je onaj od nizova $P(A_{X-1})$ i $P(A_{X-2}) \cup \{a_X\}$, koji ima veći zbir.

Formiraćemo niz B, tako da je $B(X)$ zbir elemenata optimalnog podniza $P(A_X)$. Prema predhodnom, sledi

$$B(0) = 0,\ B(1) = \max(0, a_1),$$
$$B(X) = \max\{B(X-1), B(X-2) + a_X\},\ X = 2, \ldots, N.$$

Na osnovu niza B jednostavno se pronalazi podniz P.

Implementacija:

```
Program Maksimalna_Suma_Nesusednih;
  var a,b:array[0..1000]of integer;
      n,i:integer;
      f:text;

  procedure ispis(n:integer);
    begin
      if n>0 then
      if b[n]=b[n-1] then ispis(n-1)
      else
        begin
          ispis(n-2);  write(a[n]:4); write(f,a[n]:4);
        end;
    end;

  begin
    assign(f,'input.dat');
    reset(f);
    readln(f,n);
    for i:=1 to n do readln(f,a[i]);
    close(f);
    b[0]:=0;
    if a[1]>0 then b[1]:=a[1] else b[1]:=0;
    for i:=2 to n do
      if b[i-2]+a[i]>b[i-1] then b[i]:=b[i-2]+a[i]
    else b[i]:=b[i-1];
    assign(f,'output.dat');
    rewrite(f);
    ispis(n);
    writeln(f);writeln(f,b[n]);
    close(f);
    writeln;  writeln(b[n]);
  end.
```

4.16 Svi najjeftiniji putevi

Poreklo zadatka: [25].

U jednoj zemlji ima N gradova. Svi gradovi su povezani putevima, mada ne obavezno direktnim. Nema gradova koji su spojeni sa više od jednog direktnog puta. Za svako putovanje od grada do grada poznata je (pozitivna) cena,koja ne mora biti srazmerna dužini puta. Cene su date matricom D od N vrsta i N kolona. Ako između gradova i i j ne postoji put, cena takvog puta je beskonačna. Za

praktične potrebe najpogodnije je predstaviti da je tada u matrici cena $D(i,j)$ upisan zbir cena svih postojećih puteva ili još neki veći broj. Treba odrediti najniže cene putovanja za svaki par gradova, kao i odgovarajuće maršute.

Rešenje: Ovaj problem se rešava vrlo poznatim algoritmom Flojda,koji na prvi pogled nema veze sa dinmičkim programiranjem,zbog neočekivanog načina zadavanja podproblema. Prema do sada izloženim problemima i rešenjima, moglo bi se očekivati da se veličina podproblema zada brojem gradova između kojih treba naći najkraće puteve. Podproblem veličine X bi bio nalaženje najjeftinijih puteva između prvih X gradova u nekom odabranom redosledu. Tada bi novi grad trebalo pokušati umetnuti na svaki od puteva, na svakom od mesta u tom putu, a posebne teškoće zadalo bi nalaženje najkraćih puteva od novog grada do ostalih gradova. Sve to zahteva mnogo operacija. Kako onda upotrebiti dinamičko programiranje?

Konstatujmo najpre nekoliko činjenica:

Na najjeftinijem putu od grada u do grada v gradovi mogu da se pojave najvše jednom. U protivnom bi bilo moguće pjeftiniti put izbacivanjem petlje, što je u kontradikciji sa pretpostavkom da je taj put najjeftiniji.

Ako je put P najjeftiniji put od u do v, i put P sadrži čvor W, onda je deo puta P od u do w najjejftiniji put od u do w a deo puta P od w do v je najjeftiniji put od w do v, što se jednostavno dokazuje svpđenjem na kontradikciju. U ovom tvrđenju se ogleda optimalnost podstrukture problema. Formulišimo isto zapažanje na nešto pogodniji način: neka je određen optimalan (najjeftiniji) put za svaki par gradova. Neka optimalan put P od u do v ne sasdrži grad N. Tada je taj put od u do v optimalan put koji sadrži samo (neke od) prvih $N-1$ gradova za presedanje. Ako optimalan put P od u do v sadrži grad N, onda su putevi od u do N, i od N do v optimalni putevi koji koriste samo prvih $N-1$ gradova za presedanje (jer se na putu P grad N može pojaviti samo jednom (jer je put P optimalan), pa su optimalni i njegovi delovi). Sada je jasno da veličina podproblema treba da se zada brojem gradova preko kojih se sme presedati. Podproblem veličine K je nalaženje (za svaki par gradova) optimalnih puteva, koristeći samo u prvih K gradova.

Dalje rešavanje je jednostavno. Neka je D_k matrica dužina optimalnih putevakoji kao među čvorove koriste samo prvi K čvorova. Tada je

$$\begin{aligned} D_0(i,j) &= D(i,j) \\ D_K(i,j) &= \min\{D_{K-1}(i,j), D_{K-1}(i,K) + D_{K-1}(K,j)\}, K = 1 \ldots N \end{aligned}$$

Tražena matrica najjeftinijih puteva je D_N. Da bismo mogli da rekonstruišemo i same puteve,koristićemo i matricu C. Na početku $C(i,j) = 0$ za sve i,j. Kada se pri nekom K promeni $D(i,j)$, stavimo $C(i,j) = K$, čime smo zapamtili da optimalan put (u K-tom podproblemu) od i do j ide preko K. Rekurzivna procedura ispis (i,j) ispisuje brojeve svih gradova na putu od i do j (osim samih i i j).

Na ovom primeru se vidi da iako su problemi koje rešavamo slični, ne treba ih rešavati po inerciji. U nekim problemima mogu se uočiti hijerarhije podproblema raznih vrsta. Potrebno je, međutim (najčešće kroz analizu strukture optimalnog rešenja) doći do takvih podproblema, čija rešenja su sadržana u rešenju većeg problema, i čijim se kombinovanjem rešenje većeg problema može dobiti.

Što se tiče memorijskih zahteva, ovo rešenje zahteva $\mathcal{O}(n^2)$ memorije. Složenost rešenja je $\mathcal{O}(n^3)$.

Implementacija:

```
program Svi_najjeftiniji_putevi;
 uses crt,dos;
 var d,c:array[1..50,1..50] of integer;
     n,i,j,k:integer;
 procedure ispis(a,b:integer);
    begin
     if c[a,b]>0 then
       begin
        ispis(a,c[a,b]);write(c[a,b]:5);ispis(c[a,b],b);
       end;
    end;
   begin
     textcolor(1);write('broj gradova?');textcolor(2);read(n);
     for i:=1 to n do
      for j:=1 to n do
      begin
        textcolor(3);write('od',i,'do',j,':');textcolor(6);read(d[i,j]);  c[i,j]:=0;
      end;
     readln;
     for k:=1 to n do
      for i:=1 to n do
       for j:=1 to n do
        if d[i,j]>d[i,k]+d[k,j] then
        begin
         d[i,j]:=d[i,k]+d[k,j];    c[i,j]:=k;
        end;
     textcolor(9);writeln('Optimalne cene puteva:');
     for i:=1 to n do
     begin
      for j:=1 to n do  begin
                          textcolor(14);   write(d[i,j]:5);
                        end;
        writeln;
     end;
     textcolor(4); writeln('Koji put zelite da rekonstruisete u celosti:');
     textcolor(13);write('i= ');textcolor(12);readln(i);
     textcolor(13);write('j=');textcolor(12);readln(j);
     textcolor(7);writeln(i:5);
     textcolor(8);ispis(i,j);
     textcolor(11);writeln(j:5);
 end.
```

4.17 Roman

Poreklo zadatka: [25].

U romanu ima N glava. U i-toj glavi ima a_i stranica. Treba izdati roman u K tomova tako da broj stranica najdebljeg toma bude minimalan. Glave se ne mogu deliti. Naći broj glava u svakom tomu, kao i broj stranica najdebljeg toma.

Rešenje: Označimo sa $B(X, Y)$ broj stranica najdebljeg toma u optimalnom rastavu prvih X glava na Y tomova. Tako je, na primer $B(X, 1)$ jednako zbiru prvih

X elemenata niza A, a $B(N,K)$ je vrednost koja se traži.

Predpostavimo da je dato jedno optimalno rastavljanje N glava na K tomova, i da u optimalnom rastavu romana, u poslednji tom ulazi g glava.Tada je broj strana najdebljeg toma $B(N,K)$ jednak sumi brojeva stranica poslednjih g glava ako je poslednji tom najdeblji, a najdebljem od preostalih $K-1$ tomova inače. Primetimo da tada (ako poslednji tom nije najdeblji) rastavljanje prvih $K-1$ tomova na $N-g$ glava mora takođe biti optimalno, inače bi se moglo popraviti rešenje glavnog zadatka. Zbog toga je debljina najdebljeg od prvih $K-1$ tomova jednaka $B(n-g,K-1)$. Kraće zapisano, važi:

$$B(N,K) = \max\left\{\sum_{i=N-g+1}^{N} A_i, B(N-g,K-1)\right\}$$

Primetimo da ako je poslednji tom u optimalnom rešenju za K tomova najdeblji, prethodni tomovi u optimalnom rešenju NE MORAJU biti formirani na optimalan način. Dovoljno je da ni jedan tom ne prelazi debljinu K-tog, a ne da najdeblji $K-1$ prvih bude što tanji. Međutim, ova konstatacija ne utiče na optimalnost podstrukture! Da bi problem mogao da se rešava dinamičkim programiranjem, potrebno je da se jedno optimalno rešenje problema može dobiti kombinovanjem optimalnih rešenja podproblema, a ne da je svako optimalno rešenje takvo. Jasno je da i u slučaju kada je u optimalnom rešenju poslednji tom najdeblji, prethodni tomovi MOGU biti formirani na optimalan način. Dinamičkim programiranjem pronaći ćemo ono optimalno rešenje koje u sebi sadrži optimalna rešenja podproblema.

Do rešenja dolazimo tako sto izračunamo sve $B(X,Y)$ redom: najpre za $Y=1$ imamo

$$B(X,1) = \sum_{i=1}^{x} A_i zaX = 1,N$$

za $X=1,N$ (jer sve glave idu u jedan tom), a zatim za $Y=1$ probamo za svako smisleno g (takvo da za svaki tom preostane bar jedna glava) da stavimo g glava u poslednji tom, tražeći pri tome ono g za koje se dobija najmanja debljina najdebljeg toma. Tako dobijemo:

$$\begin{aligned} B(X,Y) &= \min_{1\le g\le X+1-Y}\left\{\max\left\{B(Y-g,Y-1), \sum_{i=X-g+1}^{X} A_i\right\}\right\}, \\ & 2\le Y\le K, 1\le X\le N. \end{aligned}$$

Popunjavajući matricu B redom, dobićemo traženu debljinu kao $B(N,K)$. Da bismo znali i broj glava u pojedinim tomovima, potrebno je da zapamtimo za koje g je dobijen navedeni minimum. U tu svrhu formiraćemo matricu C, iste veličine kao B, tako da je $C(X,Y)$ jednako onom g koje ostvaruje navedeni minimum. Nakon formiranja matrice C, treba rekonstruisati brojeve glava u pojedinim tomovima. To se najlakše postiže rekurzivnom procedurom, koja se poziva ukupno K puta (onoliko puta koliko ima tomova):

Primetimo i to, da kad već vršimo rekonstrukciju na osnovu matrice C nije nam neophodna cela matrica B, već samo poslednja cela kolona i kolona koja se formira. Ako bismo kolone popunjavali s kraja (za X od N do 1), B se može zameniti jednim nizom, čime bismo zauzeli znatno manje memorije.

Implementacija:

```
program roman;
 uses crt,dos;
  var
   b,c:array[1..150,1..20] of integer;
   a:array[1..150] of integer;
   k,n,x,y,g,f:integer;
    function max(a,b:integer):integer;
     begin
      if a>b then max:=a
             else max:=b
     end;
    procedure ispis(x,y:integer);
     begin
      if y>0 then
       begin
        ispis(x-c[x,y],y-1);
        writeln('U tom  ' , y , ' ide ' ,c[x,y],  'gl.');
       end;
     end;
    begin
     textcolor(1);write('broj glava n=');textcolor(2);readln(n);
     textcolor(1);write('broj tomova k=');textcolor(2);readln(k);
     for x:=1 to n do
      begin
       textcolor(3);write('a[',x,']=');  textcolor(5);readln(a[x]);
      end;
     b[1,1]:=a[1];   c[1,1]:=1;
     for x:=2 to n do
     begin
      b[x,1]:=b[x-1,1]+a[x];    c[x,1]:=x;
     end;

     for y:=2 to k do
      for x:=1 to n do
       begin
        b[x,y]:=max(b[x-1,y-1],a[x]);   c[x,y]:=1;
        for g:=2 to x+1-y do
        begin
         f:=max(b[x-g,y-1],b[x,1]-b[x-g,1]);
         if b[x,y]>f then
         begin
          b[x,y]:=f;  c[x,y]:=g;
         end;
        end;
       end;
    textcolor(4);writeln('Broj strana u najdebljem tomu je',b[n,k],'.');
    textcolor(14);ispis(n,k);
 end.
```

5 Apsolutno neklasični problemi dinamičkog programiranja

U ovom odeljku razmotrićemo nekoliko primena tehnike DP u problemima koji na prvi pogled nemaju nikakve sličnosti sa klasičnim problemima u kojima se ova tehnika primenjuje. Ova konstatacija se naročito odnosi na problem izračunavanja generalisanih inverza matrica.

5.1 Primena dinamičkog programiranja na rešavanje problema trgovačkog putnika

Graf se predstavlja uređenim parom $G=(N,L)$, gde je $N=\{1,\ldots,n\}$ skup čvorova, i $L \subseteq \{(i,j)|\ i \in N, j \in N\}$ skup grana. Ako se elementima čvorova N i/ili elementima skupa grana L konačnog grafa $G=(N,L)$ pridruže određeni brojevi ili funkcije, takav graf se naziva *mreža*. Dužinu svake grane (i,j) ćemo označavati sa c_{ij}. Oznaka $\Gamma(i)$ predstavlja skup čvorova koji slede čvor i, tj. $\Gamma(i)=\{j \in N|\ (i,j) \in L\}$, dok oznaka $\Gamma^{-1}(i)$ predstavlja skup čvorova koji prethode čvoru i, tj. $\Gamma^{-1}(i)=\{j \in N|\ (j,i) \in L\}$. Početni i završni čvor se obeležavaju sa s i t, redom (videti, na primer [17, 26, 7]).

Zadatak nalaženja najkraće Hamiltonove konture na zadatoj mreži naziva se **problem trgovačkog putnika** (Traveling Salesman Problem, **TSP**). Hamiltonova kontura je put koji kroz sve čvorove grafa prolazi jednom i samo jednom i završava se u početnom čvoru:

$$hk=(i_1,i_2,\ldots,i_{n-1},i_n,i_1),\ i_p \neq i_k,\ \forall p,k \in \{1,\ldots,n\}, p \neq k.$$

Nalaženje najkraće Hamiltonove konture liči na realni zadatak trgovačkog putnika koji planira obilazak višše gradova, pri čemu polazi iz jednog grada i vraća se u isti.

Matematički model problema se formira na sledeći način.

Data je mreža $G=(N,L)$ sa dužinama grana c_{ij}, za svako $(i,j) \in L$, dok su $c_{ij}=\infty$ za $(i,j) \notin L$. Bez umanjenja opštosti, možemo usvojiti da je čvor 1 polazni i završni čvor. Definisaćemo skup $Q \subseteq N\backslash\{1\}$, kao bilo koji podskup čvorova grafa koji ne sadrži čvor 1. Označićemo sa $f(Q,j)$, $j \notin Q$ dužinu najkraćeg puta koji polazi od čvora 1, prolazi kroz sve čvorove iz skupa Q i završava u čvoru j. Po definiciji Hamiltonove konture, najkraća Hamiltonova kontura imaće dužinu $f(N\backslash\{1\},1)$ tj. biće jednaka dužini najkraćeg puta koji polazi od prvog čvora, prolazi kroz sve čvorove osim prvog i završava se u prvom čvoru.

Na osnovu ovoga se može definisati sledeća rekurentna formula dinamičkog programiranja [17, 26]:

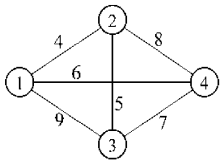
$$\begin{aligned}(\forall Q \subseteq N\backslash\{1\})(\forall j \in N\backslash Q)\quad f(Q,j) &= \min_{i \in Q}\{f(Q\backslash\{i\},i)+c_{ij}\},\\ f(\emptyset,j) &= c_{1j}.\end{aligned}$$

Na osnovu prethodnog, optimalno rešenje dobija se dobija kao rešenje sledećeg problema:

$$f(N\backslash\{1\},1) = \min_{i\in N\backslash\{1\}} \{f(N\backslash\{i,1\},i) + c_{i1}\}.$$

Sledeći primer je preuzet iz [26].

Primer 5.1.1 Odrediti najkraću Hamiltonovu konturu za zadatu mrežu.



Slika 2.4.1. Prikaz zadate mreže.

Rešenje: S obzirom da je nulta etapa trivijalna, počećemo od prve:

$$\begin{aligned}
f(\{2\},3) &= f(\emptyset,2) + c_{23} = c_{12} + c_{23} = \underline{4+5} = 9\\
f(\{3\},2) &= f(\emptyset,3) + c_{32} = c_{13} + c_{32} = 9 + 5 = 14\\
f(\{2\},4) &= f(\emptyset,2) + c_{24} = c_{12} + c_{24} = 4 + 8 = 12\\
f(\{4\},2) &= f(\emptyset,4) + c_{42} = c_{14} + c_{42} = 6 + 8 = 14\\
f(\{3\},4) &= f(\emptyset,3) + c_{34} = c_{13} + c_{34} = 9 + 7 = 16\\
f(\{4\},3) &= f(\emptyset,4) + c_{43} = c_{14} + c_{43} = \underline{6+7} = 13.
\end{aligned}$$

Druga etapa se može izraziti sledećim jednačinama:

$$\begin{aligned}
f(\{2,3\},4) &= \min_{i\in\{2,3\}} \left\{ \begin{array}{c} f(\{3\},2) + c_{24} \\ f(\{2\},3) + c_{34} \end{array} \right\} = \min \left\{ \begin{array}{c} 14+8 \\ \underline{9+7} \end{array} \right\} = 16\\
f(\{2,4\},3) &= \min_{i\in\{2,4\}} \left\{ \begin{array}{c} f(\{4\},2) + c_{23} \\ f(\{2\},4) + c_{43} \end{array} \right\} = \min \left\{ \begin{array}{c} 14+512+7 \end{array} \right\} = 19\\
f(\{3,4\},2) &= \min_{i\in\{3,4\}} \left\{ \begin{array}{c} f(\{4\},3) + c_{32} \\ f(\{3\},4) + c_{42} \end{array} \right\} = \min \left\{ \begin{array}{c} \underline{13+5} \\ 16+8 \end{array} \right\} = 18
\end{aligned}$$

I na kraju treća, završna etapa u kojoj dobijamo konačno rešenje, opisuje se sledećim jednakostima:

$$\begin{aligned}
f^* &= f(2,3,4,1) = \min_{i\in\{2,3,4\}} f(N\backslash\{1,i\},i) + c_{i1}\\
&= \min \left\{ \begin{array}{c} f(\{3,4\},2) + c_{21} \\ f(\{2,4\},3) + c_{31} \\ f(\{2,3\},4) + c_{41} \end{array} \right\} = \min \left\{ \begin{array}{c} \underline{18+4} \\ 19+9 \\ \underline{16+6} \end{array} \right\} = 22.
\end{aligned}$$

Najkraća Hamiltonova kontura zadate mreže ima dužinu 22. Redosled čvorova se dobija prateći unazad dobijene minimalne vrednosti (koje su podvučene). U zadatku imamo dva najkraća puta i to: (1, 2, 3, 4, 1) i (1, 4, 3, 2, 1).

Napomena 5.1.1 *U simetričnim mrežama se uvek dobija paran broj najkraćih Hamiltonovih kontura. Razlog je jasan: jednu konturu je moguće obići u oba smera.*

Osim problema nalaženja najkraće Hamiltonove konture, isti postupak se može primeniti i na nalaženje najkraćeg Hamiltonovog puta kroz mrežu. Hamiltonov put se definiše kao put koji polazi od početnog, prolazi kroz ostale čvorove samo jedanput, i završava se u nekom završnom čvoru:

$$hp = (s, i_1, \ldots, i_{n-1}, t),\ i_p \neq i_k,\ \forall p,k \in \{1,\ldots,n\}, p \neq k.$$

Postoje različite varijante problema najkraćeg Hamiltonovog puta kroz mrežu. Poznata je varijanta u kojoj su zadati početni i završni čvor, ili samo jedan od njih. Ove varijante se mogu rešiti opisanim postupkom dinamičkog programiranja uz male modifikacije.

5.2 Dinamički transportni problem

Određenu klasu transportnog problema (nelinearni problemi) čine problemi kod kojih se prevoz tereta obavlja transportnim sredstvom koje snabdeva robom više odredišta iz samo jednog ishodišta. Kod ovih problema kriterijum optimalnosti može biti definisan preko minimalne ukupne dužine puta, minimalnih troškova ili vremena putovanja. Primena dinamičkog programiranja u problemima transporta dala je mogućnost optimizacije troškova prevoza i u slučaju nelinearnih problema transporta. Nelinearni transportni problemi spadaju u grupu problema koji se moraju dekomponovati u niz etapa kako bi se problem mogao rešiti dinamičkim programiranjem. Korišćenje diskretnog dinamičkog programiranja omogućava i rešavanje onih transportnih problema kod kojih se može u prevozu javiti i nehomogeni teret. U ovakvim slučajevima postavlja se zahtev prevoza maksimalne vrednosti (cena, prioritet) različitog tereta, uz što veće iskorišćenje kapaciteta transportnog sredstva. U nastavku će biti razmotrena upravo ova klasa problema.

Neka je poznato transportno sredstvo (kamion, vagon, avion, brod i slično) nosivosti (kapaciteta) Q mernih jedinica. U ovo sredstvo potrebno je utovariti n različitih tipova predmeta sa različitom "vrednošću" (cenom, prioritetom). U razmatrano transportno sredstvo potrebno je utovariti predmete maksimalne "vrednosti", pri čemu P_i predstavlja težinu jednog predmeta i-tog tipa ($i = 1, 2, \ldots, n$), C_i "vrednost" predmeta i-tog tipa, a x_i je broj predmeta i-tog tipa.

Matematički zapis modela problema transporta je sledeći:

$$\max \quad F(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} C_i x_i$$

$$\text{p.o.} \quad \sum_{i=1}^{n} P_i x_i \leq Q, \; x_i = 0, 1, 2, 3, \ldots$$

Pri utovaru u transportno sredstvo predmeta samo prvog tipa maksimalna "vrednost" utovara je:

$$f_1(Q) = \max\{C_1 x_1\}$$

uz ograničenje:

$$P_1 x_1 \leq Q, \; x_1 = 0, 1, 2, 3, \ldots$$

Broj predmeta prvog tipa utovarenih u transportno sredstvo, dat je najvećim celim brojem koji ne prelazi Q/P_1:

$$x_1 = \lfloor Q/P_1 \rfloor$$

Pri određivanju maksimalne "vrednosti" problema polazi se od relacije:

$$f_1(Q) = \lfloor Q/P_1 \rfloor C_1.$$

Pri utovaru u transportno sredstvo predmeta prvog i drugog tipa maksimalna "vrednost" utovara se označava sa $f_2(Q)$. Ako je u ovom slučaju utovareno x_2 predmeta drugog tipa, tada težina predmeta prvog tipa ne sme preći težinu $Q-P_2x_2$ dok je "vrednost" jednaka

$$C_2x_2 + f_1(Q - P_2x_2).$$

Maksimalna "vrednost" utovara može se izračunati upotrebom rekurzivne relacije:

$$f_2(Q) = \max_{0\leq x_2\leq \lfloor \frac{Q}{P_2} \rfloor} \{C_2x_2 + f_1(Q - P_2x_2)\}.$$

Nastavljanjem ovog procesa dolazi se nakon n koraka do rekurzivne relacije:

$$f_n(Q) = \max_{0\leq x_n\leq \lfloor \frac{Q}{P_n} \rfloor} \{C_nx_n + f_{n-1}(Q - P_nx_n)\}$$

pri čemu je:
$f_n(Q)$ - maksimalna vrednost utovara sastavljenog upotrebom n tipova predmeta;
C_nx_n - "vrednost" koja odgovara upotrebi x_n predmeta n-tog tipa
$f_{n-1}(Q-P_nx_n)$ - maksimalna "vrednost" utovara sastavljena od $n-1$ tipova predmeta čija težina nije veća od $Q - P_nx_n$.

Ovako definisan model transporta može se koristiti u prevozu homogenog i nehomogenog tereta kada je poznato:
- kapacitet transportnog sredstva Q,
- težina P_j za jedinični prevoz j-tog tipa tereta,
- broj x_i predmeta tipa i,
- "vrednost" ("cena", "prioritet") po jedinici predmeta tipa i, te gabariti pojedinog tipa predmeta.

Poznavanje ovih parametara omogućava postizanje maksimalne "vrednosti" (cene, prioriteta) prevoza uz maksimalno korišćenje kapaciteta (nosivosti) transportnog sredstva. Primena opisanog modela biće pokazana primeru koji je preuzet iz [3].

Primer 5.2.1 Zadate su četiri vrste roba pakovane u sledećim težinskim ili drugim jedinicama: $P_1 = 11, P_2 = 9, P_3 = 7$ i $P_4 = 6$ (tona, m^3, komada i sl.). "Prioritet" ("vrednost") ovih predmeta (npr. paleta) ocenjen je od stručnih lica i iznosi: $C_1 = 40, C_2 = 38, C_3 = 26$ i $C_4 = 19$ (bodova, poena). Zadatak se sastoji u maksimiziranju "vrednosti" prevoza uz što veće korišćenje kapaciteta (nosivosti) transportnog sredstva. Transportno sredstvo ima nosivost od 25 težinskih jedinica.

Rešenje: Neka se u transportno sredstvo utovaraju predmeti prvog tipa, tj. roba koja je pakovana (paletizovana) u pakete težine $P_1 = 11$ težinskih jedinica. Ovih predmeta moguće je utovariti u količini $[Q/P_1] = [25/11] = 2$, pa je $x_1 = 0; 1; 2$ predmeta. Budući da je težina tereta $P_1 = 11$ težinskih jedinica, to kod kapaciteta transportnog sredstva $0 \leq Q \leq 10$ težinskih jedinica nije moguće utovariti ni jedan predmet prvog tipa. Jedan predmet je moguće utovariti kod kapaciteta transportnog sredstva $11 \leq Q \leq 21$ težinskih jedinica, a dva predmeta kod kapaciteta $22 \leq Q \leq 25$ težinskih jedinica. Izvedeni proračun za Q, $f_1(Q)$ i x_1 unet je u sledeću tabelu.

Q	$f_1(Q) = \lfloor Q/P_1 \rfloor C_1$	$x_1 = \lfloor Q/P_1 \rfloor$
0 - 10	0	0
11 - 21	40	1
22 - 25	80	2

Tabela 2.5.1.

Neka se u sledećoj etapi u transportno sredstvo utovaraju predmeti prvog i drugog tipa. Kako je $P_2 = 9$ težinskih jedinica, to x_2 ima moguće vrednosti 0;1;2 jer je $\lfloor Q/P_2 \rfloor = \lfloor 25/9 \rfloor = 2$. Kapacitet transportnog sredstva bi trebalo podeliti na intervale u zavisnosti od vrednosti parametara P_1 i P_2, a zatim korišćenjem relacije

$$f_2(Q) = \max_{0 \le x_2 \le \lfloor \frac{Q}{P_2} \rfloor} \{C_2 x_2 + f_1(Q - P_2 x_2)\}.$$

i činjenice da je $x_2 = 0; 1; 2$ formirati vrednosti funkcije $f_2(Q)$ za utovar predmeta prvog i drugog tipa. Pri kapacitetu $0 \le Q \le 8$ nije moguće u transportno sredstvo utovariti predmete ni prvog ni drugog tipa. Kod kapaciteta $9 \le Q \le 10$ primena formule za $f_2(Q)$ daje maksimalnu vrednost 38 (poena, bodova):

$$\begin{aligned} f_2(9) &= \max_{0 \le x_2 \le \lfloor \frac{9}{9} \rfloor} \{38x_2 + f_1(9 - 9x_2)\} \\ &= \max \left\{ \begin{array}{l} 38 \cdot 0 + f_1(9 - 9 \cdot 0) = 0 \\ 38 \cdot 1 + f_1(9 - 9 \cdot 1) = 38 \end{array} \right\} = 38, \end{aligned}$$

što znači da se pri ovom kapacitetu utovaruje samo jedan predmet drugog tipa.

Urađen je sličan proračun za svaki interval do nosivosti od 25 težinskih jedinica. Kod kapaciteta $11 \le Q \le 17$ moguće je utovariti samo jedan predmet prvog tipa, tako da primena formule za $f_n(Q)$, za $n = 1$, daje maksimalnu vrednost 40 (poena, bodova):

$$\begin{aligned} f_n(Q) &= \max_{0 \le x_n \le \lfloor \frac{Q}{P_n} \rfloor} \{C_n x_n + f_{n-1}(Q - P_n x_n)\} \\ f_1(Q) &= \max_{0 \le x_1 \le \lfloor \frac{Q}{P_1} \rfloor} \{C_1 x_1\} \\ f_1(11) &= \max_{0 \le x_1 \le \lfloor \frac{11}{11} \rfloor} \{40x_1\} = \max \left\{ \begin{array}{l} 40 \cdot 0 = 0 \\ 40 \cdot 1 = 40 \end{array} \right\} = 40. \end{aligned}$$

Kod kapaciteta $18 \le Q \le 19$ moguće je utovariti dva predmeta drugog tipa, tako da primena formule za $f_n(Q)$, za $n = 2$, daje maksimalnu vrednost 76 (poena, bodova):

$$\begin{aligned} f_n(Q) &= \max_{0 \le x_n \le \lfloor \frac{Q}{P_n} \rfloor} \{C_n x_n + f_{n-1}(Q - P_n x_n)\} \\ f_2(Q) &= \max_{0 \le x_2 \le \lfloor \frac{Q}{P_2} \rfloor} \{C_2 x_2 + f_{2-1}(Q - P_2 x_2)\} \\ f_2(18) &= \max_{0 \le x_2 \le \lfloor \frac{18}{9} \rfloor} \{38x_2 + f_1(18 - 9x_2)\} \\ &= \max \left\{ \begin{array}{l} 38 \cdot 0 + f_1(18 - 9 \cdot 0) = 0 + f_1(18) = 0 + 40 = 40 \\ 38 \cdot 1 + f_1(18 - 9 \cdot 1) = 38 + f_1(9) = 38 + 0 = 38 \\ 38 \cdot 2 + f_1(18 - 9 \cdot 2) = 76 + f_1(0) = 76 + 0 = 76 \end{array} \right\} = 76. \end{aligned}$$

Kod kapaciteta $20 \le Q \le 21$ moguće je utovariti jedan predmet prvog i jedan drugog tipa, tako da primena formule za $f_n(Q)$, za $n = 2$, daje maksimalnu vrednost 78 (poena, bodova):

$$\begin{aligned} f_2(Q) &= \max_{0 \le x_2 \le \lfloor \frac{Q}{P_2} \rfloor} \{C_2 x_2 + f_{2-1}(Q - P_2 x_2)\} \\ f_2(20) &= \max_{0 \le x_2 \le \lfloor \frac{18}{9} \rfloor} \{38x_2 + f_1(18 - 9x_2)\} \\ &= \max \left\{ \begin{array}{l} 38 \cdot 0 + f_1(20 - 9 \cdot 0) = 0 + f_1(20) = 0 + 40 = 40 \\ 38 \cdot 1 + f_1(20 - 9 \cdot 1) = 38 + f_1(11) = 38 + 40 = 78 \\ 38 \cdot 2 + f_1(20 - 9 \cdot 2) = 76 + f_1(2) = 76 + 0 = 76 \end{array} \right\} = 78. \end{aligned}$$

Kod kapaciteta $22 \le Q \le 25$ moguće je utovariti dva predmeta prvog tipa, tako da primena

formule za $f_n(Q)$, za $n = 1$, daje maksimalnu vrednost 80 (poena, bodova):

$$\begin{aligned} f_n(Q) &= \max_{0 \le x_n \le \lfloor \frac{Q}{P_n} \rfloor} \{C_n x_n + f_{n-1}(Q - P_n x_n)\} \\ f_1(Q) &= \max_{0 \le x_1 \le \lfloor \frac{22}{11} \rfloor} \{40 x_1\} \\ f_1(22) &= \max_{0 \le x_1 \le \lfloor \frac{11}{11} \rfloor} \{40 x_1\} = \max \left\{ \begin{array}{l} 40 \cdot 0 = 0 \\ 40 \cdot 1 = 40 \\ 40 \cdot 2 = 80 \end{array} \right\} = 40. \end{aligned}$$

Kako se za svaki interval izvršio proračun dolazi se do rezultata da se najveća vrednost $f_2(25)$ dobija pri utovaru u transportno sredstvo dva predmeta prvog tipa, kao što je prikazano u tabeli.

Q	$F_2(Q)$	x_2	x_1
0 - 8	0	0	0
9 - 10	38	1	0
11 - 17	40	0	1
18 - 19	76	2	0
20 - 21	78	1	1
22 - 25	80	0	2

Tabela 2.5.2.

U trećoj i četvrtoj etapi razmatrajući mogućnosti utovara predmeta prvog, drugog i trećeg tipa, odnosno prvog, drugog, trećeg i četvrtog tipa dolazi se do rezultata datih u naredne dve tabele. Vrednosti funkcija $f_3(Q)$ i $f_4(Q)$ se dobijaju korišćenjem sledećih rekurzivnih relacija:

$$\begin{aligned} f_3(Q) &= \max_{0 \le x_3 \le \lfloor \frac{Q}{P_3} \rfloor} \{C_3 x_3 + f_2(Q - P_3 x_3)\}, \ x_3 = 0, 1, 2, 3 \\ f_4(Q) &= \max_{0 \le x_4 \le \lfloor \frac{Q}{P_4} \rfloor} \{C_4 x_4 + f_2(Q - P_4 x_4)\}, \ x_4 = 0, 1, 2, 3, 4 \end{aligned}$$

U narednoj tabeli prikazane su vrednosti utovara za predmete prvog, drugog i trećeg tipa.

Q	$f_3(Q)$	x_3	x_2	x_1
0 - 6	0	0	0	0
7 - 8	26	1	0	0
9 - 10	38	0	1	0
11 - 13	40	0	0	1
14 - 15	52	2	0	0
16 - 17	64	1	1	0
18 - 19	76	0	2	0
20 - 21	78	3	1	1
22	80	0	0	2
23 - 24	90	2	1	0
25	102	1	2	0

Tabela 2.5.3.

U tabeli ispod prikazane su vrednosti utovara za predmete sva četiri tipa.

Q	$f_4(Q)$	x_4	x_3	x_2	x_1
0 - 5	0	0	0	0	0
6	19	1	0	0	0
7 - 8	26	0	1	0	0
9 - 10	38	0	0	1	0
11 - 12	40	0	0	0	1
13	45	1	1	0	0
14	52	0	2	0	0
15	57	1	0	1	0
16 - 17	64	0	1	1	0
18 - 19	76	0	0	2	0
20 - 21	78	0	0	1	1
22	83	1	1	1	0
23	90	0	2	1	0
24	95	1	0	2	0
25	102	0	1	2	0

Tabela 2.5.4.

Optimalno rešenje postavljenog problema prevoza, u vrednosti prioriteta od 102 (boda, poena), dobija se pri prevoženju jednog predmeta trećeg tipa i dva predmeta drugog tipa. U ovom slučaju, kapacitet transportnog sredstva potpuno je iskorišćen jer je 1*7+2*9 = 25 težinskih jedinica. Analiza rezultata optimizacije pokazuje da dobijeno optimalno rešenje nije dato samo za transportno sredstvo nosivosti 25 težinskih jedinica, već i za ona sredstva čiji se kapacitet kreće u intervalu $0 \leq Q \leq 25$ težinskih jedinica, što znači da je rešen jedan skup sličnih problema. Takođe je moguće uočiti i kolika je "vrednost" transporta i koliko je iskorišćenje kapaciteta transportnog sredstva ako se ne donese odluka koja nije optimalna.

5.3 Dinamičko programiranje i uopšteni inverzi

Dinamičko programiranje se može koristiti za generisanje uopštenih inverza matrice. Rezultati ovog odeljka su preuzeti iz rada [4].

Poznato je da generalisani inverzi matrica imaju važnu ulogu u najmanjim srednje-kvadratnim problemima (least squares problems) [1, 27]. Kada se minimizira veličina $q = (Ax - b)^T(Ax - b)$, rešenje je dato sa $x = A^\dagger b$, gde je $A^\dagger b$ MoorePenroseov inverz matrice A. Osim toga, ako rešenje nije jedinstveno, tada je $x = A^\dagger b$ rešenje minimalne dužine (solution of minimal length), koje je jedinstveno. Generalisani inverzi se koriste u mnogim drugim disciplinama. Oni su presudni u teoriji linearnih asocijativnih memorija (linear associative memories) [12], u regresionoj analizi (regression analysis) [2], i u teoriji ograničenog mehaničkog kretanja (theory of constrained mechanical motion) [23].

Postoji veliki broj metoda za izračunavanje generalisanih inverza. Između ostalih, poznati su metodi Decella and Grevillea [5, 24]. Istaknutu ulogu ima rastavljanje matrice pomoću singularno vrednosne dekompozicije (singular value decomposition) [23]. Osim toga, pristup dinamičkog programiranja koji koristi funkcionalne jednačine se takođe može upotrebiti, zato što se matrica može posmatrati kao konstrukcija kojoj se pridodaje po jedna kolona u nekom trenutku vremena. Dinamičko programiranje se koristi u rešavnju problema najmanjeg srednje-kvadratnog rešenja [6]. Međuim, prvi put je primena dinamičkog programiranja u izračunavanju generalisanih inverza objavljena u radu [8]. U radu [8] je to urađeno

pokazivanjem veze između dinamičkog programiranja i Greville-ovog sekvencijalnog metoda uvedenog u [9].

Izvođenje $\alpha Q \beta R$ algoritma [10, 11]

Ovaj dinamički program za izračunavanje generalisanog inverza je inspirisan $\alpha Q \beta R$ algoritmom koji traži vektor x posredstvom dinamičkog programa tako da je norma $|Ax - b|^2$ najmanja, i da je dužina vektora x minimalna. Pri tome je A matrica dimenzija $m \times n$, b je vektor-kolona dimenzije m, i x je vektor-kolona dimenzije n. Pogledajmo prvo ukratko kako $\alpha Q \beta R$ algoritam izračunava najmanje kvadratno rešenje. Dve ciljne funkcije su uvedene u ovom algoritmu. Prva je

$$f_k(b) = \text{najmanji kvadrat dužine vektora } A_k x^k, \tag{5.3.1}$$

gde je A_k matrica koja se sastoji od prvih k kolona od A, i x_k je vektor-kolona dimenzije k. Druga funkcija je data izrazom

$$g_k(b) = \text{najmanji kvadrat dužine vektora } x_k, \tag{5.3.2}$$

u kome x^k odgovara ograničenju

$$|A_k x^k - b|^2 = \min. \tag{5.3.3}$$

Kolone matrice A su označene sa $a_1, a_2, \ldots a_n$. Pretpostavimo da su vrednosti $f_{k-1}(b)$ i $g_{k-1}(b)$ poznate. Tada su vrednosti $f_k(b)$ i $g_k(b)$ dobijene korišćenjem principa optimalnosti [6]. Procedura dinamičkog programiranja za izračunavanje $f_k(b)$ i $g_k(b)$ bazirana na poznavanju funkcija $f_{k-1}(b)$ i $g_{k-1}(b)$ zavisi od toga da li je a_k linearno zavisan od $a_1, a_2, \ldots, a_{k-1}$. Ako je a_k linearno zavisan od $a_1, a_2, \ldots, a_{k-1}$, tada je

$$f_k(b) = f_{k-1}(b); \tag{5.3.4}$$

jer linearna kombinacija vektora $a_1, a_2, \ldots a_{k-1}$; a_k ne može da aproksimira vektor b bolje od linearne kombinacije vektora $a_1, a_2, \ldots a_{k-1}$. Takođe je

$$g_k(b) = \min_{x_k} \left[x_k^2 + g_{k-1}(b - x_k a_k) \right], \tag{5.3.5}$$

gde je x_k skalar, k-ta komponenta vektora x^k. To je zbog toga što kada je x_k izabrano, tada kazna iznosi x_k^2. Ostale komponente, $x_1, x_2, \ldots, x_{k-1}$, moraju biti izabrane tako da je linearna kombinacija vektora $a_1, a_2, \ldots, a_{k-1}$ što je moguće bliža novom ciljnom vektoru $b - x_k a_k$. Osim toga, suma $x_1^2 + x_2^2 + \cdots + x_{k-1}^2$ mora biti minimalna.

U slučaju kada je a_k linearno nezavisan od $a_1, a_2, \ldots, a_{k-1}$, sa bilo kojim izborom sklara x^k moramo da što približnije aproksimiramo novi ciljni vektor $b - x_k a_k$ pomoću sume $x_1 a_1 + \cdots + x_{k-1} a_{k-1}$. Prema tome,

$$f_k(b) = \min_{x_k} f_{k-1}(b - x_k a_k) \tag{5.3.6}$$

Ako je minimizirajuća vrednost skalara x_k jednaka x_k^*, tada

$$g_k(b) = (x_k^*)^2 + g_{k-1}(b - x_k^* a_k). \tag{5.3.7}$$

Razmotrimo sada funkciju $f_k(b) =$ kvadrat dužine reziduuma $(A_k x^k - b)$, pri čemu je vektor x^k izabran optimalno, gde je $\dim b = m$, i A_k se sastoji od prvih k kolona od A. Za $k = 1$ imamo

$$f_1(b) = \min(a_1 x_1 - b)^T (a_1 x_1 - b) \tag{5.3.8}$$

Uslov minimiziranja je

$$a_1^T a_1 x_1 - a_1^T b = 0, \tag{5.3.9}$$

tako da je

$$x_1^* = \frac{a_1^T b}{a_1^T a_1} = a_1^\dagger b, \tag{5.3.10}$$

gde je $a_1^\dagger$ generalisani inverz vektora a_1 (uz pretpostavku $a_1 \neq 0$). To povlači da je

$$f_1(b) = b^T \left[I - a_1 a_1^\dagger \right] b. \tag{5.3.11}$$

Funkcija $f_1(b)$ je kvadratna po b što može da se zapiše sa

$$f_1(b) = b^T Q_1 b, \tag{5.3.12}$$

gde je Q_1 simetrična pozitivno semi-definitna $m \times m$ matrica. Da bi pokazali da je $f_k(b)$ kvadratna matrica po b, to jest

$$f_k(b) = b^T Q_k b, \tag{5.3.13}$$

gde je Q_k simetrična pozitivno definitna $m \times m$ matrica, posmatramo dva zasebna slučaja, zavisno od toga da li je vektor a_k linearno zavisan ili nije linearno zavisan od prethodnih vektora.

Pretpostavimo prvo da je a_k nezavisan od $a_1, a_2, \ldots, a_{k-1}$ i da je k-ti element vektora x označen sa s. Iz jednačina (5.3.6) imamo

$$f_k(b) = \min_s f_{k-1}(b - a_k s), \ k = 2, 3, \ldots, n. \tag{5.3.14}$$

Pretpostavimo da

$$f_{k-1}(b) = b^T Q_{k-1} b. \tag{5.3.15}$$

Tada se dobija

$$f_k(b) = \min_s \left\{ f(b - sak)^T Q_{k-1} (b - sak) \right\}. \tag{5.3.16}$$

Funkcija koju treba optimizirati je

$$\begin{aligned} \{Q_{k-1}(b - sa_k)\} &= (b^T Q_{k-1} - s a_k^T Q_{k-1}(b - sa_k) \\ &= b^T Q_{k-1} b - s a_k^T Q_{k-1} b - b^T Q_{k-1} s a_k \\ &\quad + s^2 a_k^T Q_{k-1} a_k. \end{aligned} \tag{5.3.17}$$

Da bismo dobili optimalno rešenje, stavimo $(\frac{d\{\}}{ds}) = 0$, tako da je

$$\frac{d\{\}}{ds} = 2sa_k^T Q_{k-1} a_k - aTkQ_{k-1}b - b^T Q_{k-1} a_k = 0 \tag{5.3.18}$$

Primetimo da kako je $a_k^T Q_{k-1} b$ skalar, a_k^T

$$Q_{k-1}b = (a_k^T Qk - 1b)^T = b^T Q_{k-1} a_k.$$

Prema tome, optimalno rešnje je

$$s^* = \frac{b^T Q_{k-1} a_k}{a_k^T Q_{k-1} a_k} = \frac{a^T k Q_{k-1} b}{a_k^T Q_{k-1} a_k}. \tag{5.3.19}$$

Primenimo dva oblika optimalnog rešenja s, i označimo $a_k = Q_{k-1}a_k$. Jednačinu (5.3.17) možemo sada zapisati sa

$$f_k(b) = b^T Q_{k-1} b - 2b^T \alpha_k \frac{\alpha_k^T b}{\alpha^T k \alpha_k} + \frac{b^T \alpha_k}{\alpha_k^T \alpha_k} (\alpha_k^T \alpha_k) \frac{\alpha_k^T b}{\alpha_k^T \alpha_k}. \tag{5.3.20}$$

Premo tome,

$$f_k(b) = b^T \left(Q_{k-1} - \frac{\alpha_k \alpha_k^T}{a_k^T \alpha_k} \right) b = b^T Q_k b. \tag{5.3.21}$$

Rekurzivna veza za Q_{k-1} je

$$Q_k = Q_{k-1} - \frac{\alpha_k \alpha_k^T}{\alpha_k^T a_k}. \tag{5.3.22}$$

Pretpostavimo sada da je a_k linearno zavisno od $a_1, a_2, \ldots, a_{k-1}$. Tada je ispunjeno $f_k(b) = f_{k-1}(b)$, jer linearna kombinacija od $a_1, a_2, \ldots, a_{k-1}$; a_k ne može biti bliža vektoru b od linearne kombinacije vektora $a_1, a_2, \ldots, a_{k-1}$. Premo tome, u ovom slučaju važi $Q_k = Q_{k-1}$.

Razmotrimo sada drugu funkciju $g_k(b)$ = najmanji kvadrat dužine vektora x^k, za koju je $|A_k x^k - b|$ minimalno.

Za $k = 1$ sledi, iz prethodnih razmatranja, da je u jednoj fazi procesa

$$x_1^* = \frac{a_1^T b}{a_1^T a_1} = a_1^\dagger b, \tag{5.3.23}$$

gde je $a_1^\dagger$ generalisani inverz vektora a_1 (pretpostavljajući da je $a_1 \neq 0$). To povlači

$$g_1(b) = b^T (a_1^\dagger)^T a_1^\dagger b. \tag{5.3.24}$$

Funkcija $g_1(b)$ je kvadratana po b, što možemo zapisati sa

$$g_1(b) = b^T R_1 b, \tag{5.3.25}$$

gde je R_1 smetrična pozitivno semi-definitna $m \times m$ matrica. Da bi pokazali da je $g_k(b)$ takođe kvadratna po b, tj.

$$g_k(b) = b^T R_k b, \tag{5.3.26}$$

razmatramo dva zasebna slučaja, zavisno od toga da li je vektor a_k linearno zavisan od prethodnih vektora ili ne. Pretpostavimo da je a_k nezavisan od $a_1, a_2, \ldots, a_{k-1}$. Iz jednačina (5.3.19) zaključujemo

$$s^* = \frac{\alpha_k^T b}{\alpha_k^T \alpha_k}. \tag{5.3.27}$$

Primenimo s^* na jednačinu (5.3.7), koja je oblika

$$g_k(b) = (s^*)^2 + g_{k-1}(b - s^* a_k)$$

dobija se

$$\begin{aligned} g_k(b) &= (s^*)^2 + (b - s^* a_k)^T R_{k-1}(b - s^* a_k) \\ &= \frac{b^T \alpha_k}{\alpha_k^T a_k} \frac{\alpha_k^T b}{\alpha_k a_k} + b^T \left[I - \frac{a_k \alpha^T k}{\alpha_k^T a_k} \right] R_{k-1} \left[I - \frac{a_k \alpha^T k}{\alpha_k^T a_k} \right] b \\ &= b^T \left\{ \left[I - \frac{\alpha_k a_k^T}{\alpha_k^T a_k} \right] R_{k-1} \left[I - \frac{a_k \alpha_k^T}{\alpha_k^T a_k} \right] + \frac{\alpha_k \alpha_k^T}{(\alpha_k^T a_k)^2} \right\} b. \end{aligned} \tag{5.3.28}$$

Pretpostavimo sada da a_k zavisi od $a_1, a_2, \ldots, a_{k-1}$. Iz prethodnog dela znamo da rešenje jednačine (5.3.3) nije jednistveno, i da jedno od tih rešenja daje minimnalnu dužinu vektora x^k. Ako je $k$ti element rešenja x označen sa s, dobija se rekurzivna relacija

$$g_k(b) = \min_s \{s^2 + g_{k-1}(b - s a_k)\}. \tag{5.3.29}$$

Uz pretpostavku

$$g_{k-1}(b) = b^T R_{k-1} b, \tag{5.3.30}$$

dobijamo

$$g_k(b) = \min_s \{s^2 + (b - s a_k)^T R_{k-1}(b - s a_k)\}, \tag{5.3.31}$$

$$\begin{aligned} \{\ldots\} &= s^2 + (b^T - s a_k^T) R_{k-1}(b - s a_k) \\ &= s^2 + (b^T R_{k-1} - s a_k^T R_{k-1})(b - s a_k) \\ &= s^2 + b^T R_{k-1} b - 2 s a^T k R_{k-1} b + s^2 a_k^T R_{k-1} a_k. \end{aligned} \tag{5.3.32}$$

Označimo $R_{k-1} a_k$ sa β_k, i stavimo $\frac{d\{\}}{ds} = 0$. Tada je

$$2s - 2\beta_k^T b + 2 s a_k^T \beta_k = 0. \tag{5.3.33}$$

Premo tome, dobija se optimalno rešenje

$$s^* = \frac{b^T \beta_k}{1 + a_k^T \beta_k}. \tag{5.3.34}$$

Kako je s skalar, takođe imamo

$$s = \frac{\beta^T k b}{1 + a_k^T \beta k}.$$

Primenimo jednačinu (5.3.34) i poslednju jednačinu na funkciju $g_k(b)$. Vidi se da je

$$\begin{aligned} g_k(b) &= \frac{b^T \beta_k}{1 + a_k^T \beta_k} \left(1 + a_k^T \beta_k\right) \frac{\beta_k^T b}{1 + a_k^T \beta_k} + b^T R_{k-1} b - 2 \frac{b^T \beta_k}{1 + a_k^T \beta_k} \beta_k^T b \\ &= b^T \left(R_{k-1} - \frac{\beta_k \beta_k^T}{1 + a_k^T \beta_k} \right) b. \end{aligned} \tag{5.3.35}$$

Prema tome, time smo dokazali da je $g_k(b)$ takođe kvadratna po b; to jest

$$g_k(b) = b^T R_k b \tag{5.3.36}$$

gde je $R_k = R_{k-1} - (\beta_k \beta_k^T / (1 + a_k^T \beta_k))$.

U prethodnoj proceduri, potrebno je odrediti kada je a_k linearno zavisan od $a_1, a_2, \ldots, a_{k-1}$, a kada nije. Dakle, potrebno je pokazati da važi

$$a_k = \alpha_k + \text{lin. kom. od } \alpha_1, \alpha_2, \ldots, \alpha_k, \tag{5.3.37}$$

gde je $\alpha_k = Q_{k-1} a_k$. Takođe, pokazano je da je α_k ortogonalno na $a_1, a_2, \ldots, a_{k-1}$ [4]. Prema tome, umesto direktne provere da li a_k linearno zavisi od $a_1, a_2, \ldots, a_{k-1}$, možemo proveriti da li je a_k nula vektor ili nije nula vektor. Dakle, ako je

$$\alpha_k = Q_{k-1} a_k = 0,$$

tada je a_k linearno zavisan od $a_1, a_2, \ldots, a_{k-1}$; u suprotnom, ako je $a_k = Q_{k-1} a_k \neq 0$, tada je a_k linearno nezavisan od $a_1, a_2, \ldots, a_{k-1}$. U sekciji 4 ćemo pokazati da su svi podaci potrebni za dobijanja generalisanog inverza dobijeni u prethodnoj proceduri. Algoritamski koraci su:

$$\alpha_1 = a_1, \tag{5.3.38}$$

$$Q_1 = I - \frac{\alpha_1 \alpha_1^T}{\alpha_1^T \alpha_1} = I - a_1 \alpha_1^\dagger, \tag{5.3.39}$$

$$R_1 = (a_1^\dagger)^T a_1^\dagger. \tag{5.3.40}$$

Tada za svaku vrednost k, $k = 2, 3, \ldots, n$, imamo dva slučaja. Ako je

$$\alpha_k = Q_{k-1} a_k = 0, \tag{5.3.41}$$

tada je

$$Q_k = Q_{k-1}, \tag{5.3.42}$$

$$\beta_k = R_{k-1} a_k, \tag{5.3.43}$$

$$R_k = R_{k-1} - \frac{\beta_k \beta_k^T}{1 + a_k^T \beta_k}. \tag{5.3.44}$$

Ako je, sa druge strane važi

$$\alpha_k = Q_{k-1}a_k \neq 0, \tag{5.3.45}$$

tada je

$$\alpha_k^\dagger = \frac{\alpha_k^T}{\alpha_k^T\alpha_k}, \tag{5.3.46}$$

$$Q_k = Q_{k-1} - \alpha_k\alpha_k^\dagger, \tag{5.3.47}$$

$$R_k = (\alpha_k^\dagger)^T\alpha_k^\dagger + (I - a_k\alpha_k^\dagger)^T R_{k-1}(I - a_k\alpha_k^\dagger). \tag{5.3.48}$$

Izvođenje Grevilleovog algoritma

Zbog svoje velike primenljivosti, Grevilleov rezultat je široko rasprostranjen, ali bez konstruktivnog dokaza zbog kompleksnosti tehnike rešavanja. Jednostavano izvođenje su dali Udwadia i Kalaba [5]. U nastavku slede detalji dokaza.

Pretpostavimo da je poznat MP inverz matrice A, označen sa $A^\dagger$. Želimo da rešimo problem najmanjeg kvadrata $Bx \cong b$. Matrica B je podeljena sa $[A\ a]$, i vektor x je podeljen sa $\begin{bmatrix} z \\ s \end{bmatrix}$, gde je z vektor dimenzije $k = 1$, i s je skalar. Prema tome, možemo pisati $Bx \cong b$ kao

$$[Aa]\begin{bmatrix} z \\ s \end{bmatrix} s \cong b.$$

Tada

$$Az^\dagger as \cong b, \tag{5.3.49}$$

$$Az \cong b - as. \tag{5.3.50}$$

Kada znamo $A^\dagger$, rešenje linearnog sistema je

$$z = A^\dagger(b - as). \tag{5.3.51}$$

Zamenom jednačine (5.3.51) u jednačinu (5.3.50) dobija se

$$AA^\dagger(b - as) \cong b - as, \tag{5.3.52}$$

$$(I - AA^\dagger)as \cong (I - AA^\dagger)b. \tag{5.3.53}$$

Označimo $(I - AA^\dagger)a$ sa c. Ako je $c \neq 0$, imao

$$s = \frac{a^T(I - AA^\dagger)}{a^T(I - AA^\dagger)(I - AA^\dagger)a}(I - AA^\dagger)b. \tag{5.3.54}$$

Kako je $(I - AA^\dagger)$ simetrična i idempotentna matrica, što se može pokazati pomoću osobina generalisanog inverza, imamo

$$s = \frac{a^T(I - AA^\dagger)}{a^T(I - AA^\dagger)a}b, \tag{5.3.55}$$

$$s = c^\dagger b. \tag{5.3.56}$$

Sa druge strane, znamo da je rešenje $[A\ a]\begin{bmatrix} z \\ s \end{bmatrix} \cong b$ dato sa

$$\begin{bmatrix} z \\ s \end{bmatrix} = [A\ a]^\dagger b. \tag{5.3.57}$$

Zamenom rešenja za z i s, koja su ranije dobijena, dobijamo

$$[A\ a]^\dagger b = \begin{bmatrix} A^\dagger - A^\dagger a c^\dagger \\ c^\dagger \end{bmatrix} b. \tag{5.3.58}$$

Kako je vektor b proizvoljan,

$$[A\ a]^\dagger = \begin{bmatrix} A^\dagger - A^\dagger a c^\dagger \\ c^\dagger \end{bmatrix}, \text{ ako je } c \neq 0. \tag{5.3.59}$$

Razmotrimo sada slučaj kada je $c = 0$; zapravo, c je m-dimenzionalni nula vektor. Iz jednačine (5.3.53), znamo da kada je $c = 0$, skalar s može biti proizvoljan, što znači da vektor koji je rešenje nije jedinstven. Umesto da biramo proizvoljno rešenje tako da važi $[A\ a]\begin{bmatrix} z \\ s \end{bmatrix} \cong b$, tražimo rešenje minimalne dužine, to jest

$$z^T z + s^2 = \min. \tag{5.3.60}$$

Označimo vektor $(A^\dagger b)$ sa u a vektor $(A^\dagger a)$ sa v. Sada imamo

$$z = A^\dagger(b - as) = u - vs. \tag{5.3.61}$$

Prema tome

$$(u - vs)^T(u - vs) + s^2 = \min, \tag{5.3.62}$$

$$u^T u - 2sv^T u + v^T v s^2 + s^2 = \min. \tag{5.3.63}$$

Vrednost s koja minimizira dužinu vektora $z^T z + s^2$ u (5.3.60) je dobijena iz

$$(1 + v^T v)s = v^T u = v^T A^\dagger b, \tag{5.3.64}$$

što daje

$$s = \frac{v^T A^\dagger}{1 + v^T v} b, \tag{5.3.65}$$

tako da je

$$[A\ a]^\dagger = \begin{bmatrix} A^\dagger - A^\dagger a \frac{v^T A^\dagger}{1 - v^T v} \\ \frac{v^T A^\dagger}{1 - v^T v} \end{bmatrix}, \text{ if } c = 0. \tag{5.3.66}$$

Jednačine (5.3.59) i (5.3.66) određuju rekurzivnu proceduru.

Generalisani inverzi i $\alpha Q \beta R$

Ponovimo da se rešenje minimalne norme problema najmanjeg kvadrata $(Ax - b)^T(Ax - b) = \min$ takođe može dobiti iz $x = A^\dagger b$, gde je $A^\dagger$ pseudo-inverz od A. Prirodno je tražiti $A^\dagger$ primenom $\alpha Q \beta R$ algoritma.

Grevilleov algoritam pokazuje kako da znajući pseudoinverz $A_{k-1}^\dagger$ matrice A_{k-1}, koja se sastoji od prvih $k-1$ kolona matrice A, izračunamo pseudoinverz $A_k^\dagger$ matrice A_k, koja se sastoji od prvih k kolona matrice A. Vektor c igra ključnu ulogu u ovom postupku. Razmotrimo šta konkretno vektor c predstavlja. Pretpostavimo da matrica A sadrži n kolona, i da vektor kolona w sadrži n skalara $w_1, w_2, \ldots, w_n$. Oni su označeni sa

$$A = [a_1 \ a_2 \ \ldots \ a_n], \quad w = \begin{bmatrix} w_1 \\ w_2 \\ \ldots \\ w_n \end{bmatrix}.$$

Prema tome, Aw je linearna kombinacija kolona matrice A. Kako je $c = (I - AA^\dagger)a$,

$$c^T Aw = a^T (I - AA^\dagger)^T Aw. \tag{5.3.67}$$

Zato što je $I - AA^\dagger$ simetrična matrica, važi

$$\begin{aligned} c^T Aw &= a^T (I - AA^\dagger) Aw, \\ c^T Aw &= a^T (Aw - AA^\dagger Aw). \end{aligned} \tag{5.3.68}$$

Podsetimo na jednu od osobina MP generalisanog inverza, $AA^\dagger A = A$. Zaključujemo da je

$$c^T Aw = 0, \tag{5.3.69}$$

što znači da je vektor c ortogonalan na svaku linearnu kombinaciju kolona matrice A. Još više, iz $c = (I - AA)^\dagger a$ imamo

$$a = c + AA^\dagger a, \tag{5.3.70}$$

gde $A^\dagger a$ možemo posmatrati kao neki vektor w. Prema tome, a je linearna kombinacija prethodnih kolona matrice A plus vektor c koji je ortogonalan na sve te kolone matrice A. Dakle, nova kolona, a, je određena kao nezavisna od svih prethodnih kolona, kada c nije nula vektor.

Razmotrimo sada vektor α_k u $\alpha Q \beta R$ algoritmu. Može se pokazati da je vektor α_k ortogonalan na vektore $a_1, a_2, \ldots, a_{k-1}$. Umesto primene vektora c kao u Grevilleovom algoritmu, prelazak sa $A_{k-1}^\dagger$ na $A_k^\dagger$ može se izvršiti po članovima ili α_k ili β_k, zavisno od toga da li je $\alpha_k \neq 0$ ili $\alpha_k = 0$.

Razmotrimo prvo slučaj u kome je $c \neq 0$. To znači da vektor a_k ima nenula komponentu koja je ortogonalna na vektore $a_1, a_2, \ldots, a_{k-1}$. Dakle, to je slučaj u kome α_k nije linearno zavisan od $a_1, a_2, \ldots a_{k-1}$. Grevilleova rekurzija je data sa

$$A_k^\dagger = \begin{bmatrix} A_{k-1}^\dagger - A_{k-1}^\dagger a_k c^\dagger \\ c^\dagger \end{bmatrix}, \tag{5.3.71}$$

koja zahteva da znamo vektor c, pored $A_{k-1}^{\dagger}$ i a_k. Ali $\alpha Q \beta R$ algoritam daje vektor α_k, koji je komponenta od a_k koja je ortogonalna na $a_1, a_2, \ldots a_{k-1}$, kao što je pokazano ranije. Dakle, kada su $A_{k-1}^{\dagger}$, a_k i α_k poznati, možemo odrediti $A_k^{\dagger}$ sa

$$A_k^{\dagger} = \begin{bmatrix} A_{k-1}^{\dagger} - A_{k-1}^{\dagger} a_k \alpha_k^{\dagger} \\ \alpha_k^{\dagger} \end{bmatrix}. \tag{5.3.72}$$

Drugi slučaj je $c = 0$. U tom slučaju je $a_k = A_{k-1} A_{k-1}^{\dagger} a_k$, pa je vektor a_k linearno zavisan od kolona matrice A_{k-1}. Grevilleova rekurzija je data formulom

$$A_k^{\dagger} = \begin{bmatrix} A_{k-1}^{\dagger} - A_{k-1}^{\dagger} a_k v^T \\ v^T \end{bmatrix}, \tag{5.3.73}$$

gde je

$$v = \frac{(A_{k-1}^{\dagger})^T A_{k-1}^{\dagger} a_k}{1 + a_k^T (A_{k-1}^{\dagger})^T A_{k-1}^{\dagger} a_k}. \tag{5.3.74}$$

Za interpretaciju tih relacija sa stanovišta $\alpha Q \beta R$ algoritma, ponovimo jednačine (5.3.2) i (5.3.14):

$$g_k(b) = \text{najmanji kvadrat dužine vektora } x^k$$

$$\text{koji minimizira } |A_k x_k - b|^2 = b^T R_k b.$$

Kako znamo da je rešenje tog problema jednako

$$x_k = A_k^{\dagger} b, \tag{5.3.75}$$

možemo pisati

$$g_k(b) = b^T (A_k^{\dagger})^T A_k^{\dagger} b. \tag{5.3.76}$$

Prema tome, vidimo da je

$$R_k = (A_k^{\dagger})^T A_k^{\dagger}, \tag{5.3.77}$$

što bi trebalo da bude saglasno sa R_k koje se dobija u pethodnim jednačinama (5.3.21) i (5.3.25). To omogućava da pišemo

$$v = \frac{R_{k-1} a_k}{1 + a_k^T R_{k-1} a_k}, \tag{5.3.78}$$

$$v = \frac{\beta_k}{1 + a_k^T \beta_k}, \tag{5.3.79}$$

što prestavlja vektor v preko izraza β_k. Ove relacije pokazuju kako se prelazak sa A_{k-1} na A_k može izvršiti preko α_k ili β_k, zavisno od toga da li je $\alpha_k = 0$ ili $\alpha_k \neq 0$. Dodatno, znamo i rešenje problema $|A_k x^k - b|^2 = \min$ is $x^k = A_k^{\dagger} b$. Dakle

$$\begin{aligned} f_k(b) &= (A_k A_k^{\dagger} b - b)_T (A_k A_k^{\dagger} b - b), \\ f_k(b) &= b^T (A_k A_k^{\dagger} - I)(A_k A_k^{\dagger} - I) b, \\ f_k(b) &= b^T (A_k A_k^{\dagger} - A_k A_k^{\dagger} - A_k A_k^{\dagger} + I) b, \\ f_k(b) &= b^T (I - A_k A^{\dagger})_k^{\dagger} b. \end{aligned} \tag{5.3.80}$$

To pokazuje da je

$$Q_k = I - A_k A_k^{\dagger}, \tag{5.3.81}$$

gde je Q_k izračunat tokom $\alpha Q \beta R$ algoritma, A_k se sastoji od prvih k kolona matrice A, i $A_k^{\dagger}$ je generalisani inverz od A_k.

U radovima [14, 15, 16, 21, 22] upotrebljen je Grevileov metod pregrađivanja za izračunavanje uopštenih inverza racionalnih i polinomijalnih matrica.

6 Memoizacija

6.1 Transformacija stringa

Poreklo zadatka: [25].

Data su dva niza: niz X dužine N se sastoji od slova A i B, a niz Y dužine $M > N$ od cifara 0 i 1. Da li se iz prvog niza dobiti drugi, koristeći sledeće operacije:
1. zameniti bilo koje A nepraznim nizom cifara 0, ili
2. zameniti bilo koje B nepraznim nizom nula ili nepraznim nizom jedinica.

Rešenje: Tražena transformacija nije moguća ako se niz X završava slovom A, a niz Y jedinicom. Pretpostavimo zato da se niz X završava slovom A, a niz Y sekvencom od K nula,ili da se niz X završava slovom B, a niz Y sekvencom od K nila ili K jedinica. Transformacija niza X u niz Y će tada biti moguća ako is samo ako je moguće transformisati niz X_{N-1} (X bez poslednjeg elementa) u neki od nizova $Y_P, M - K \leq P < M$.

Dovoljno je rešiti sve probleme $Q(I, J)$ transformacije niz X_I, u niz $Y_J, 1 \leq I \leq N, 1 \leq J \leq M$. Odgovore možemo smestiti u logičku matricu, koju ćemo popunjavati po vrstama ili po kolonama. Odgovor na postavljeno pitanje je u polju (N, M) te logičke matrice.

Pretpostavimo da je $X = (ABA)$, $Y = (0001110000)$. Dinamičko programiranje rešava svih $N \cdot M = 3 \cdot 10 = 30$ problema, dok memoizacija mořešavati lanac $Q(3, 10) \to Q(2, 6) \to Q(1, 3) \to Q(0, 0)$, nakon čega neće stići (a neće ni morati) da postavi ostaje podprobleme (iz $Q(3, 10)$, to su $Q(2, 7), Q(2, 8)$ i $Q(2, 9)$: a iz $Q(2, 6)$ to su $Q(1, 4)$ i $Q(1, 5)$ i njihovi dalji pozivi u dubinu).

6.2 Pobednička strategija

Poreklo zadatka: [25].

Postoji mnogo igara za dva igrača koje imaju sledeće zajedničke osobine: dat je konačan skup legalnih pozicija S i konačan skup dozvoljenih poteza M, kojima se iz jedne legalne pozicije prelazi u drugu. Igrači vuku poteze naizmenično. Jedan podskup F skupa svih pozicija S čini završne pozicije. To su pozicije u kojima nije moguće odigrati ni jedan legalan potez. Kada se stigne do neke završne pozicije, igra je završena, a igrač koji je odigrao poslednji potez je pobednik. Ni jedna pozicija se ne može ponoviti tokom igre, tako da se svaka igra neminovno završava u konačnom broju poteza i nerešen ishod ne postoji. Cilj svakog igrača je da pobedi kad god je to moguće. Problem koji se u ovakvim igrama obično postavlja je da za datu

poziciju ustanovimo da li igrač koji je na potezu može da pobedi, i koji potez treba radi toga da odigra.

Navešćemo jedan postupak koji u ovakvim igrama uvek možemo da primenimo. Prema uslovima zadatka, igra se može predstaviti konačnim acikličnim grafom, tako da su čvorovi grafa pozicije, a grane grafa su potezi. Obojimo sve čvorove koji odgovaraju završnim pozicijama u crno. U tim pozicijama, igrač koji je na potezu je izgubio. Nakon toga, obojene čvorove iz kojih se nekim potezom može preći u crni čvor obojimo u belo (u takvoj poziciji igrač koji je na potezu može da pobedi tako što pređe na poziciju kojoj odgovara crni čvor), a neobojene čvorove iz kojih se svakim mogućim potezom prelazi u beli čvor obojimo u crno. Postupak bojenja je konačan i svaki čvor na opisani način dobija crnu ili belu boju. Pobednička strategija se (kao što smo već rekli) sastoji u tome da igrač koji je na potezu pređe na poziciju kojoj pdgovara crni čvor. Na taj način protivnik zatiče ili završnu poziciju pa je odmah izgubio, ili poziciju iz koje mora preći u neku kojoj odgovara beli čvor. Prema tome, prvi igrač može da nastavi da sprovodi istu strategiju, pa će na taj način povući poslednji potez i pobediti.

U igri se koristi N kuglica. Svaki igrač neizmenično uzima po izvestan broj kuglica. U prvom potezu igrač može uzeti najviše P kuglica ($P < N$), a u svakom sledećem potezu ne višeod P i ne više od za Q više od onoga koliko je uzeo protivnik u prethodnom potezu ($Q < P$). Na primer, za $N = 100, P = 10, Q = 5$, ako bi prvi igrač uzeo 3 kuglice, drugi može uzeti od 1 do 8 kuglica, a ako bi prvi uzeo 6 kuglica, legalni potezi grugog su 1 do 10 kuglica. Cilj je uzeti poslednju kuglicu. Odrediti pobedničku strategiju.

Rešenje: Pozicija se može zadati brojem preostalih kuglica na stolu i maksimalnim brojem kuglica kije igrač može uzeti. Tako je polazna pozicija $(100, 10)$, a u gornjem primeru iz nje nastaju pozicije $(97, 8)$ i $(94, 10)$. Možemo popuniti matricu B veličine $N * P$, tako da je $B(X, Y)$ broj kuglica koje prema pobedničkoj strategiji treba uzeti da bi se pobedilo, ili 0 ako pobedu nije moguće garantovati. Podproblemi se rešavaju po vrstama sleva na desno, dakle za svako X od 0 do N, a pri fiksiranom X za svako Y od $Q + 1$ do P (drugi broj u uređenom paru koji opisuje tekuću poziciju, predstavlja gornje ograničenje broja kuglica koje se mogu uzeti, a ono se u zavisnosti od prethodnog poteza kreće od $Q + 1$ do P).

Memoizacija je ovde nešto bolje rešenje, jer se ni ovde ne moraju rešavati svi podproblemi. Čim ustanovimo da je pozicija $(91, 10)$ gubitnička, sledi da je pozicija $(100, 10)$ pobednička, pa nećemo imati potrebe da rešavamo problem za pozicije 92,93,94,95,96,97,98,99 i 99 kuglica. Takvih ušteda ima mnogo tokom igre, a ne samo na prvom koraku (odozgo na dole). Pri mekim vrednostima N, p, q,(na primer 35,20,15) može se desiti da se stvarno reši manje od 30% ukupnog broja problema, što znači da u tim slučajevima memoizacija radi oko tri puta brže od klasičnog pristupa.

```
program Pobednicka_Strategija;
  uses crt,dos;
   var  b:array[0..100,0..100] of integer;
        n,p,q,x,y:integer;
```

```
function min(g,h:integer):integer;
  begin
   if g<h then min:=g    else min:=h
  end;
procedure resi(x,y:integer);
   var  k:integer;
   begin
      if b[x,y]=-1 then
       begin
        if x<=y then b[x,y]:=x;
        k:=y;
        while (k>0) and (b[x,y]=-1) do
         begin
          resi(x-k,min(k+q,p));
          if b[x-k,min(k+q,p)]=0 then b[x,y]:=k;
          k:=k+1;
         end;
        if b[x,y]=-1 then  b[x,y]:=0;
       end;
   end;
begin
    readln(n); readln(p); readln(q);
     for x:=0 to n do
        for y:=0 to p do  b[x,y]:=-1;
     resi(n,p);   writeln(b[n,p]);
end.
```

6.3 Maksimalna suma nesusednih u drvetu

Svečani prijem

U jednoj velikoj firmi vlada hijerarhija nadređenosti u obliku drveta. Direktor je koren tog drveta, a svaki drugi radnik ima jedinstvenog neposrednog šefa. Osim toga, svaki radnik ima zvanični rejting (pozitivan realan broj), koji odražava važnost tog radnika. U firmi se sprema svečani prijem i vlasnik firme je u dilemi koje radnike da pozove. Vlasnik želi da postigne dva cilja: da se niko ne bi osećao neprijatno, pozive se ili radnik ili njegov neposredni šef (ili nijedan), ali nikako obojica. Osim toga, vlasnik želi da zbir rejtinga prisutnih radnika bude maksimalan.

Za svakog od N radnika numerisanih od 1 do N, dat je redni broj P_u neposrednog šefa (za direktora je data 0), i rejting T_u tog radnika. Načiniti optimalan izbor radnika koji će biti pozvani na prijem.

Rešenje: Matematčka formulacija istog zadatka je sledeća: dato je drvo G. Svaki čvor u drveta G ima realnu pozitivnu težinu T_u. Naći podskup čvorova drveta G čiji je zbir težina najveći, a u kome nema susednih čvorova drveta G.

Pod naslednicima ili potomcima nekog čvora u drvetu podrazumevaćemo njegove neposredne naslednike. Neka je $P(G)$ optimalan podskup čvorova za drvo G neka je čvor u koren drveta i neka su $v_1, \ldots, v_K$ svi naslednici čvora u, a $w_1, \ldots, w_M$ svi naslednici čvorova $v_1, \ldots, v_K$.

Tada važi sledeće:

• Ako čvor u ne ulazi u optimalan podskup $P(G)$, onda su delovi optimalnog podskupa P koji pripadaju poddrvetima sa korenima $v_1, \ldots, v_K$ redom, optimalni podskupovi za ta poddrveta.

• Ako čvor u ulazi u optimalan podskup $P(G)$, onda su delovi optimalnog podskupa P koji pripadaju redom poddrvetima iz čvorova $w_1, \ldots, w_M$, optimalni podskupovi za ta drveta.

Ovim je ustanovljena optimalnost podstrukture problema. Za svako poddrvo potrebno je da se reši isti problem, počevši od najjednostavnijih drveta, odnosno listova polaznog drveta. Kako nije sasvim jednostavno poređati podprobleme u potreban redosled (najpre listovi, pa njihovi prethodnici, itd. do korena, koji je poslednji), možemo zadatak rešavati memoizacijom. Rešenje za drvo koje ima samo jedan čvor, je taj čvor. Za netrivijalna drveta podprobleme rešavamo na isti način kao i za glavni problem, i tako dobijamo i zbir težina $B(u)$ čvorova optimalnog podskupa za drvo koje počinje iz čvora u. Odnosno za ceo graf:

$$B(u) = \max \left\{ T(u) + \sum_{i=1}^{M} B(w_i), \sum_{i=1}^{K} B(v_i) \right\}$$

Za rekonstrukciju optimalnog podskupa dovoljan je logički niz C, gde je $C(u) =$ True ako i samo ako je čvor u uvršten podskup poddrveta čiji je koren u. Pažnja: to ne znači da u konačan izbor čvorova ulaze oni čvorovi za koje je u nizu C zapisana vrednost True. Traženi izbor ćemo rekonstruisati rekurzivnom procedurom, koristeći niz C.

Ulazni podaci su, prema rečenom zadati pomoću dva niza: niz P, gde je p_i redni broj prethodnika u drvetu za čvor i, i niz T, u kome je t_i težina čvora broj i u drvetu. Međutim, niz prethodnika nije pogodan za rešavanje zadataka. Potrebno je da za dati čvor brzo odredimo njegove sledbenike (potomke), a ne prethodnika. Zato ćemo formirati niz *potomci*, čiji k-ti član će biti lista potomaka čvora k. Kada pročitamo da je prthodnik čvora i čvor a, umesto da pamtimo $p_i = a$, dodaćemo čvor i u listu potomaka čvora a.

Kada za neki čvor v izračunamo sumu težina optimalnih podskupova svih drveta koja počinju od njegovih potomaka, možemo je zapamtiti kao $bb(v)$, jer će biti ponovo potreban pri rešavanju problema za prethodnika u čvora v. Težina optimalnog podskupa za drvo koje počinje iz v je uobičajeno označena sa $b(v)$.

```
program Svecani_Prijem;
  uses crt,dos;
  type PokClanListe=^Clanliste;
       ClanListe= record
                    vred:integer;
                    sled:PokClanListe;
                  end;
  var
    potomci:array[0..100] of PokClanListe;
    b,bb,t:array[1..100] of integer;
    c:array[1..100] of boolean;
    pom:PokClanListe;
    n,i,a,k,u:integer;
```

```
procedure resi(u:integer);
  var PokV:PokClanListe;
      sumv,sumw,v:integer;
begin
    if b[u]=-1 then
     begin
       sumv:=0;   sumw:=0;   PokV:=potomci[u];
       while PokV<>nil do
        begin
         v:=PokV^.vred;    resi(v);
         sumv:=sumv+b[v];    sumw:=sumw+bb[v];    PokV:=PokV^.sled;
        end;
       bb[u]:=sumv;
       if sumw+t[u]>sumv then
         begin
          b[u]:=sumw+t[u];    c[u]:=true;
         end
        else
          begin b[u]:=sumv;c[u]:=false; end
       end;
     end;

procedure ispis(u:integer);
   var
    PokV,PokW:PokClanListe;
    v,w:integer;
   begin
    if not c[u] then
    begin
     PokV:=potomci[u];
     while PokV<>nil do
     begin
      v:=PokV^.vred;   ispis(v);  PokV:=PokV^.sled;
     end;
    end
    else
     begin
      writeln(u);   PokV:=potomci[u];
      while PokV<>nil do
       begin
        v:=PokV^.vred;    PokW:=potomci[v];
        while PokW<>nil do
          begin
           w:=PokW^.vred;  ispis(w);  PokW:=PokW^.sled;
          end;
        PokV:=PokV^.sled;
       end;
     end;
   end;

begin
 potomci[0]:=nil;
 textcolor(1);write('broj cvorova:'); textcolor(2); readln(N);
 for k:=1 to n do
  begin
   potomci[k]:=nil;     b[k]:=-1;
  end;
```

```
 for i:=1 to n do
  begin
   textcolor(4);write('prethodnik i tezina cvora',i,':');
   textcolor(5);readln(a,t[i]);
   new(pom);   pom^.vred:=i;    pom^.sled:=potomci[a];
   potomci[a]:=pom;
  end;
   u:=potomci[0]^.vred;
   resi(u);     writeln(b[u]);     ispis(u);
end.
```

7 Veza između memoizacije, rekurzije i dinamičkog programiranja

Dinamičko programiranje se odnosi na tehniku dolaska do rešenja, a ne na samo programiranje. Pisanje programa moze da se uradi u bilo kom programskom jeziku. Konkretno u slučaju dinamičkog programiranja obično se popunjava jedna tabela rešenja (bilo niz ili matrica). Kod dinamičkog programiranja do rešenja glavnog problema dolazi se kombinacijom jednostavnijih podproblema istog tipa. Problem se dakle rešava počev od najednostavnijih problema pa dolazeći do glavnog, tako sto se prvo uoči hijearhija problema, onda se rešavaju jednostavniji problemi, njihove vrednosti se upisuju u tabelu rešenja i onda se kombinacijom dobijenih rešenja iz tabele rešavaju složeniji podroblemi sve do rešenja glavnog problema. Na kraju u tabeli se pročita traženo rešenje i često se ista tabela može iskoristiti i za traženje puta dolaska do rešenja.

Dinamičkim programiranjem rešenje se dobija znatno efikasnije u odnosu na rekurziju. Najbitniji uslov za mnogo veću efikasnost dinamičkog programiranja od rekurzije je to sto ovde podproblemi imaju zajedničke podprobleme odnosno delimično se preklapaju. Ovaj uslov nije neophodan za primenu dinamičkog programiranja, ali bez njega dinamičko programiranje gubi svoju veliku prednost u odnosu na rekurziju. Ovo se dešava zbog toga što se primenom rekurzije kao tehnike dolaska do rešenja jedan isti problem susreće više puta i svaki put se nepotrebno iznova rešava (pri tome se broj ponovljenih rešavanja po pravilu povešava eksponencijalno sa povećavanjem dimenzije glavnog problema). Kada ne bi bilo ovog preklapanja podproblema rekurzija ne bi bila manje efikasna jer bi se i tada rešavanje podproblema javlja samo jednom i čak u nekim problemima može se desiti da rekurzivno rešenje radi brže i troši znatno manje memorijskog prostora. Kao rešenje problema neefikasnosti rekurzije kod određenih tipova problema javlja se rekurzija sa memoizacijom ili kraće samo memoizacija. To se može ostvariti na sledeći način: formira se tabela rešenja kao u rešavanju problema dinamičkim programiranjem. Na početku se svi elementi tabele postave na neku besmislenu vrednost (koja se ne može dobiti rešavanjem problema, obično 0 ili -1), čime se označava da je vrednost rešenja nedefinisana, odnosno da problem nije rešavan. Rekurzivna procedura ili funkcija koja rešava zadatak trebalo bi najpre da potraži rešenje u tabeli. Ako nađe rešenje vraća to rešenje (ako se radi o funkciji) i završava sa radom, a ako ga ne nađe traži ga rekurzivno, upisuje u tabelu i na kraju vraća dobijeno rešenje (ako se radi

o funkciji) i završava sa radom. Memoizacija se može smatrati modifikacijom dinamičkog programiranja, jer se i dalje koristi tabela za pamćenje vrednosti rešenja rešenih problema. Razlika je u tome što se memoizacijom problemi (kao i uvek rekurzijom) rešavaju odozgo na dole (kod dinamičkog programiranja se rešavaju odozdo na gore). Ova razlika za efikasnost i brzinu i nije bitna, ono što je najbitnije je da se podproblemi ne rešavaju više puta. Zbog toga problemi rešeni tehnikom dinamičkog prtogramiranja mogu efikasno da se reše i memoizacijom. Po efikasnosti memoizacija je vrlo bliska dinamičkom programiranju, a najčešće je neznatno sporija jer se izvesno vreme gubi na prenos parametara i druge aktivnosi koje zahteva rekurzija. Postoji međutim problemi gde je memoizacija skoro uvek efikasnija od dinamičkog programiranja. To su problemi u kojima nije potrebno rešiti sve podprobleme (što se češće dešava) već samo neke. Dinamičko programiranje ne može unapred znati da li će neki problem biti kasnije upotrebljen u rešavanju većeg problema. Memoizacija nema taj nedostatak jer nepotrebne podprobleme neće ni postavljati. Najbolji način za proveravanje efikasnosti ovih tehnika programiranja je kroz konkretne probleme. Problemi su uzeti iz knjige [25]. Napominjem da su vremena izražena u milisekundama i da ovom prilikom sama izmerena vremena rada nisu bitna već njihov relativni odnos. Lepo se vidi da su rezultati uglavnom očekivani: da su memoizacija i dinamiško programiranje vrlo bliski po efikasnosti dok u većini problema rekurzija je znatno sporija tehnika programiranja i to konkretno u onim problemima gde, kako je navedeno, dolazi do preklapanja podproblema pa se oni nepotrebno više puta izvršavaju. Ovo znatno kod rekurzije usporava algoritam dok se kod memoizacije taj problem lako rešava prethodnim proveravanjem da li je neki problem već rešavan i ukoliko jeste on se ne rešava ponovo.

Problem ranca:

Obim problema	10	40	60	80	1000	100000
Dinamičko programiranje	0	0	10	10	110	13910
Rekurzija	0	45	41069	Dugo	Večno	Večno
Memoizacija	0	0	10	10	110	13830

Maksimalna suma nesusednih u nizu:

Obim problema	10	100	500	5000
Dinamičko programiranje	0	40	220	2354
Rekurzija	0	40	231	2284
Memoizacija	0	40	221	2323

Najduži lanac deljivosti:

Obim problema	50	100	200
Dinamičko programiranje	10	10	50
Rekurzija	20	4116	Dugo
Memoizacija	10	10	20

Roman:

Obim problema	10:5	20:10	30:10	50:5	50:10	200:10
Dinamičko programiranje	10	10	10	10	10	20
Rekurzija	10	2143	54899	170000	Dugo	Dugo
Memoizacija	10	10	10	10	10	20

Literatura

[1] A. Ben-Israel and T.N.E. Grevile, *Generalized inverses, Theory and applications, Second edition*, Canadian Mathematical Society, Springer, New York, 2003.

[2] G.W. Bohrnstedt, D. Knoke, *Statistics for Social Data Analysis, third ed.*, F.E. Peacock, Itasca, IL, 1994.

[3] S. Borović i Mili' c, Milićević, *Zbirka zadataka iz odabranih oblasti operacionih istraživanja*, Biblioteka Vojne akademije, Beograd 2001.

[4] Y. Fan, R. Kalaba, *Dynamic programming and pseudo-inverses*, Appl. Math. Comput., **139** (2003) 323-342.

[5] R. Kalaba, N. Rasakhoo, *Algorithms for generalized inverses, Journal of Optimization Theory and Applications*, **48** (1986) 427-435.

[6] R. Bellman, R. Kalaba, *Dynamic Programming and Modern Control Theory*, McGraw-Hill, New York, 1965.

[7] D. Cvetković, M. Čangalović, Đ. Dugošija, V. Kovačević – Vujčić, S. Simić, J. Vuleta, *Kombinatorna optimizacja*, DOPIS, Beograd, 1996.

[8] Y. Fan, R. Kalaba, *Dynamic programming and pseudo-inverses*, Applied Mathematics and Computations **139** (2003) 323-342.

[9] T.N.E. Grevile, *Some applications of the pseudo-inverse of matrix*, SIAM Rev., **3** (1960), 15–22.

[10] R. Kalaba, R. Xu, W. Feng, *Solving shortest length least-squares problems via dynamic programming*, Journal of Optimization Theory and Applications **85** (1995) 613-632.

[11] R. Kalaba, H. Natsuyama, *Dynamic programming and minimal norm solutions of least squares problems*, submitted for publication.

[12] T. Kohonen, *Self-Organization & Associative Memory*, Springer-Verlag, New York, Inc, 1989.

[13] S. Opricović, *Optimizacija sistema*, Nauka, Beograd, 1992.

[14] M.D. Petković, P.S. Stanimirović, P.S. *Partitioning method for two-variable rational and polynomial matrices*, Math. Balkanica, **19** (2005), 185–194.

[15] M.D. Petković, P.S. Stanimirović, *Symbolic computation of the Moore-Penrose inverse using partitioning method*, International Journal of Computer Mathematics, **82** (2005), 355–367.

[16] M.D. Petković, P.S. Stanimirović, and Tasić, M. *Effective Partitioning Method for Computing Weighted Moore-Penrose Inverse*, Computers and Mathematics with Applications, to appear.

[17] R. Petrović, *Specijalne metode u optimizaciji sistema*, Tehnička Knjiga, Beograd, 1978.

[18] D. Pisinger, *Algorithms for knapsack problem*, Ph.D. thesis, February 1995, Dept. of Computer Science, University of Copenhagen.

[19] R. Sedgewick, *Algorithms*, Addison-Wesley Publishing Company, Reading, Massachusetts, Menlo Park, California London, Amsterdam, Don Mills, Ontario, Sydney, 1984.

[20] S. Skiena, M. Revilla, *Programming Challenges*, Springer, 2002.

[21] P.S. Stanimirović, M. Tasić, *Partitioning method for rational and polynomial matrices*, Applied Mathematics and Computation **155** (2004), 137–163.

[22] Tasić,M.B., Stanimirović, P.S. and Petković, M.D. *Symbolic computation of weighted Moore-Penrose inverse using partitioning method*, Appl. Math. Comput. **189** (2007), 615–640.

[23] F. Udwadia, R. Kalaba, *Analytical Dynamics*, Cambridge University Press, London, 1996.

[24] F. Udwadia, R. Kalaba, *An alternative proof of the Greville formula*, Journal of Optimization Theory and Applications 94 (3) (1997) 23-28.

[25] M. Vugdelija, *Dinamičko programiranje*, Društvo matematičara Srbije, Beograd, 1999.

[26] M. Vujošević, *Metode optimizacije*, Društvo operacionih istraživača, Beograd, 1996.

[27] G.Wang, Y.Wei and S. Qiao, *Generalized Inverses: Theory and Computations*, Science Press, Beijing, 2004.

[28] www.z-trening.com

Glava 5

Višekriterijumska optimizacija

Modeli za nalaženje optimuma jedne kriterijumske funkcije su obično samo aproksimacija realnih problema u kojima donosilac odluke mora da vodi računa o više ciljeva. Postoje matematički modeli i metodi u kojima donosilac odluke analizira i bira rešenja na osnovu više kriterijuma koji se istovremeno razmatraju. Pritom, kao i u slučaju jednokriterijumske optimizacije, donosilac odluke implicitno zadržava slobodu da prihvati, promeni ili odbaci rešenje dobijeno na osnovu matematičkog modela optimizacije. Metode koje od samog početka formiranja matematičkog modela za određeni realni problem vode računa o više ciljeva istovremeno razvijaju se u oblasti višekriterijumske optimizacije (VKO). Ovaj deo matematičkog programiranja svoj buran razvoj ima od kraja sedamdesetih godina dvadesetog veka.

Postoji više razloga koji utiču na to da su problemi VKO suštinski drugačiji u odnosu na probleme jednokriterijumske optimizacije. Osnovni je u tome što se svi faktori koji utiču na odluku, odnosno svi ishodi koje bi imalo eventualno rešenje, posmatraju kao kriterijumi čije bi vrednosti trebalo da budu optimalne. Dakle, potrebno je naći rešenje koje je najbolje po svim razmatranim kriterijumima istovremeno a činjenica je da su neki od njih u skoro svim problemima odlučivanja međusobno konfliktni. Pored toga, razmatrani kriterijumi mogu po svojoj prirodi biti veoma raznorodni i izraženi u različitim mernim jedinicama, od novčanih jedinica, preko jedinica fizičkih veličina, do verovatnoća ili subjektivnih procena datih po nekoj skali koja se formira za konkretni problem. Sve ovo ukazuje da konačno rešenje ne može da se odredi bez učešća donosioca odluke.

Zadatke višekriterijumske optimizacije u slučajevima kada se razmatraju važne odluke kao što su odluke u vezi sa kapitalnim ulaganjima u opremu, karakteriše relativno veliki broj kriterijuma. Što je broj kriterijuma veći, zadaci višekriterijumske analize su složeniji i teži. U ovakvim situacijama, u odlučivanju učestvuje veći broj pojedinaca ili grupa i svi oni favorizuju svoje sisteme vrednosti. Radi efikasnijeg analiziranja odluke i pronalaženja pogodnog rešenja vrši se grupisanje kriterijuma.

Uobičajene su sledeće grupe kriterijuma:

- ekonomski,
- tehnički,
- tehnološki,
- socijalni i
- ekološki.

1 Uvod

Iako je linearno programiranje veoma primenljivo u praksi, mnoge probleme iz prakse je nemoguće adekvatno linearizovati a da se pritom drastično ne izgubi na tačnosti. U tom slučaju primenjuju se metodi nelinearnog programiranja. Osim nelinearnosti, u mnogim problemima je potrebno naći optimum više od jedne funkcije cilja. U tom slučaju, moramo rešavati ***problem višekriterijumske optimizacije***. Ukoliko sve funkcije cilja imaju optimum u istoj tački, problem je trivijalan i direktno se svodi na problem nelinearnog ili linearnog programiranja. U praksi je ova situacija veoma retka. Postoji više metoda za rešavanje problema višekriterijumske optimizacije [19]. Zajedničko svim tim metodima je da se polazni problem na odgovarajući način svodi na problem linearnog i nelinearnog programiranja.

Višekriterijumska optimizacija se može posmatrati kao nastavak istraživanja u klasičnoj (jednokriterijumskoj) optimizaciji, uz izvesna proširenja. Formalno, osnovno proširenje je uvođenje vektorske kriterijumske funkcije, što dovodi do problema vektorskog maksimuma. Suštinski, potrebno je da se proširi koncept optimalnosti. Razmatrajući problem vektorskog maksimuma koncept optimalnosti se zamenjuje konceptom neinferiornosti (Pareto optimalnosti). Može se uvesti pojam opšteg (jedinstvenog) kriterijuma optimizacije, koji uključuje kriterijumske funkcije i preferenciju donosioca odluke. Rešenje zadatka višekriterijumske optimizacije koje se dobija prema takvom kriterijumu je optimalno. U tom slučaju pojam optimalnog rešenja iz klasične optimizacije može se zadržati i u višekriterijumskoj. Međutim, teškoće se upravo javljaju pri pokušaju formalizacije takvog jedinstvenog kriterijuma. Zato se u višekriterijumskoj optimizaciji koriste dve faze ili etape. U prvoj fazi se određuje skup "boljih" rešenja na osnovu vektorske kriterijumske funkcije, a u drugoj se na osnovu preferencije donosioca odluke usvaja konačno rešenje, koje se može nazvati optimalnim. Skup rešenja koji se prezentira donosocu odluke trebalo bi da sadrži mali broj rešenja, koja su neinferiorna prema datim kriterijumskim funkcijama. Problem višekriterijumske optimizacije se najčešće javlja u planiranju složenih sistema; na primer, regionalni razvoj, razvoj vodoprivrednih ili elektroprivrednih sistema, urbano planiranje i očuvanje prirodne okoline [21]. Višekriterijumski problem se javlja u ekonomiji kao problem određivanja tržišne ravnoteže ili ekonomija blagostanja u decentralizovanoj ekonomiji [21]. Sličan problem se javlja i kao problem ravnoteže u teoriji igara [21]. U teoriji igara razmatraju

se igre sa više igrača, što se u teoriji odlučivanja javlja kao "grupno odlučivanje" ili odlučivanje sa više donosioca odluke.

U ovoj glavi ćemo izložićemo problem višekriterijumske optimizacije, kao i metode za njeno rešavanje. Najpre ćemo dati definiciju problema višekriterijumske optimizacije kao i neophodnih pojmova za kasnija razmatranja. Teorijski ćemo obraditi i dati implementaciju nekoliko klasičnih metoda višekriterijumske optimizacije. Svaki od opisanih metoda biće ilustrovan na jednom ili više primera. Razmatranja vezana za implementaciju metoda su originalna i preuzeta su iz radova [25], [27] kao i iz monografije [26].

Opšta formulacija višekriterijumske optimizacije (VKO) poseduje opšti oblik

$$\begin{array}{ll} \max & Q(\mathbf{x}) = Q_1(\mathbf{x}), \ldots, Q_l(\mathbf{x}), \quad \mathbf{x} \in \mathbf{R}^n \\ \text{p.o.} & f_i(\mathbf{x}) \leq 0,\ i = 1, \ldots, m \\ & h_i(\mathbf{x}) = 0,\ i = 1, \ldots, k. \end{array}$$

gde su:$Q_1(\mathbf{x}), \ldots, Q_l(\mathbf{x})$, $f_1(x), \ldots, f_m(x), g_1(x), \ldots, g_m(x)$ realne funkcije od n promenljivih koje su sadržane u vektoru $\mathbf{x} = (x_1, \ldots, x_n)$.

U ovom zadatku traži se rešenje $\mathbf{x}$ koje maksimizira svih l funkcija cilja. Zato se zadatak višekriterijumske optimizacije (VKO) naziva i zadatak vektorske optimizacije. Radi jednostavnosti ovde se razmatraju samo problemi maksimizacije. Poznato je da se zadatak minimizacije jednostavno prevodi u zadatak maksimizacije množenjem kriterijumske funkcije sa -1. Sve nadalje izložene definicije i metode moguće je prilagoditi da važe za rešavanje zadatka minimizacije.

Kažemo da je $\mathbf{X} \subseteq R^n$ *skup dopustivih rešenja* ako važi

$$\mathbf{X} = \{\mathbf{x} | f_i(\mathbf{x}) \leq 0,\ i = 1, \ldots, m;\ h_i(\mathbf{x}) = 0,\ i = 1, \ldots, k\}.$$

Svakom dopustivom rešenju $\mathbf{x} \in \mathbf{X}$ odgovara skup vrednosti kriterijumskih funkcija, tj. vektor $Q(\mathbf{x}) = (Q_1(\mathbf{x}), Q_2(x), \ldots, Q_l(\mathbf{x}))$. Na taj način se skup dopustivih rešenja preslikava u *kriterijumski skup*, tj. $S = \{Q(\mathbf{x}) | \mathbf{x} \in \mathbf{X}\}$.

U daljem tekstu biće korišćeni sledeci pojmovi:

- *Marginalna rešenja* zadatka VKO se određuju optimizacijom svake od funkcija cilja pojedinačno nad zadatim dopustivim skupom, tj. rešavanjem l jednokriterijumskih zadataka

$$\begin{array}{ll} \max & Q_j(\mathbf{x}), \quad \mathbf{x} \in \mathbf{R}^n \\ \text{p.o.} & f_i(\mathbf{x}) \leq 0,\ i = 1, \ldots, m \\ & h_i(\mathbf{x}) = 0,\ i = 1, \ldots, k. \end{array}$$

Marginalna rešenja ćemo obeležavati sa $\mathbf{x}^{(j)^*} = (x_1^{(j)^*}, x_2^{(j)^*}, \ldots, x_n^{(j)^*})$, gde je $x^{(j)^*}$ optimalno rešenje dobijeno optimizacijom j-te funkcije cilja nad zadatiom dopustivim skupom $\mathbf{X}$.

- *Idealne vrednosti funkcija cilja*, označene sa Q_j^* jesu vrednosti funkcija cilja za marginalna rešenja

$$Q_j^* = Q_j(\mathbf{x}^{(j)^*}),\ j = 1, \ldots, l.$$

- Idealne vrednosti funkcija cilja određuju *idealnu tačku* u kriterijumskom prostoru, tj. idealnu vrednost vektorske funkcije

$$Q^* = (Q_1^*, Q_2^*, \ldots, Q_l^*).$$

- Ako postoji rešenje $\mathbf{x}^*$ koje istovremeno maksimizira sve funkcije cilja, tj.

$$\mathbf{x}^* = \{\mathbf{x} | Q_j(\mathbf{x}) = Q_j^*,\ j = 1, \ldots, l\},$$

onda se takvo rešenje naziva *savršeno resenje*.

U najvećem broju slučajeva marginalna rešenja se razlikuju i savršeno rešenje ne postoji. Kada savršeno rešenje postoji, tada se u suštini ne radi o problemu VKO.

Veoma je važno imati u vidu da su u realnim problemima ciljevi gotovo uvek u koliziji, što znači da ne mogu svi biti dostignuti u potpunosti. Zbog toga najčečešće nije moguće strogo definisati optimum niti za svaka dva rešenja formalno odrediti koje je bolje od drugog. Iz tog razloga proces dobijanja rešenja zahteva učešće donosioca odluke (u daljem tekstu DO). To je najčešće neko ko ima dublji uvid u problem i po čijem se zahtevu pristupa rešavanju. Donosilaca odluke može biti i više i tada se problem može dodatno iskomplikovati zbog njihovih različitih ciljeva, udela u odlučivanju i stepena odgovornosti koji su spremni da preuzmu.

2 Osnovni pojmovi i definicije

2.1 Donosilac odluke i njegove preferencije

Na nivou odlučivanja ključnu ulogu ima **donosilac odluke** (skraćeno **DO**). Prema normativnoj teoriji odlučivanja, donosilac odluke je savršeno racionalan pojedinac koji uvek zna šta hoće i nastoji da to realizuje. Mada se ciljevi koje pred sebe postavlja razlikuju po formulaciji, sadržini, složenosti i značaju, svi oni u osnovi sadrže zajedničku komponentu. To je želja donosioca odluke da poveća dobitke, odnosno, smanji ili izbegne gubitke. Pri tome se racionalni donosilac odluke rukovodi principom maksimizacije lične dobrobiti. Nesumnjivo da svako od nas, u manjoj ili većoj meri, odstupa od ovog "ideala" koji ne razmatra izbore iz navike, brzoplete, nepromišljene ili hirovite odluke. Ipak, pažnja sa kojom se odluke donose obično raste sa njihovim značajem, tako da se pri važnim izborima pažljivo vrši procena prednosti i nedostataka pojedinih alternativa. Dobrobit koju pružaju alternative ocenjuje se na osnovu subjektivnih kriterijuma (želje, interesi, uverenja, moralni principi, ukusi i slično). Kao proizvod manje ili više saglasnih ili nesaglasnih uticaja svih ovih faktora formiraju se individualne preferencije, na osnovu kojih donosilac odluke izgrađuje stav prema alternativama i opredeljuje se za jednu od njih.

Da bi doneta odluka bila racionalna, preferencije moraju da zadovolje neke polazne pretpostavke. Za precizno definisanje ovih pretpostavki neophodno je upoznavanje sa osnovnim relacijama preferencije i indiferencije.

Relacije preferencije i indiferencije

Pretpostavimo da donosilac odluke raspolaže skupom akcija (alternativa) koje imaju samo po jedan poznat ishod, tako da se preferencije između akcija određuju na osnovu preferencija između njihovih ishoda. U opštem slučaju akcije se obeležavaju sa $A_i, i = 1, \ldots, m$. Zbog jednostavnosti izlaganja biće korišćene oznake $A_1 = x$, $A_2 = y, \ldots$.

Ako se prilikom poređenja dve alternative, x i y, smatra da je alternativa x bolja od alternative y, onda je x (strogo) preferirano u odnosu na y, tj.

$$xPy \text{ ili } x \succ y,$$

U slučaju slobodnog izbora, donosilac odluke bira alternativu x, i biće razočaran ako bude prinuđen da prihvati alternativu y. Ako su alternative x i y podjednako dobre, tada je donosilac odluke indiferentan između x i y, ili

$$xIy \text{ ili } x \sim y.$$

Prilikom izbora između x i y svejedno je koja će alternativa biti izabrana, odnosno, donosilac odluke će biti podjednako zadovoljan ili nezadovoljan da dobije x ili y. (Ovde je važno imati na umu da donosilac odluke nije indiferentan prema alternativama, već prema izboru između njih, tj. svejedno mu je koju će alternativu dobiti, a ne da li će je dobiti ili ne.)

Uslovi racionalnosti

Da bi se donela racionalna odluka neophodno je da preferencije ispune nekoliko uslova koji se nazivaju *uslovima racionalnosti* ili *uslovima logičke konzistentnosti* i formalno su izraženi u vidu sledećih osobina.

Asimetričnost - Za bilo koje dve alternative, x i y, važi

$$\text{ako } xPy, \text{ onda nije } yPx.$$

Kao neposredne posledice ove relacije, dobijaju se sledeće osobine

$$\text{ako } xPy, \text{ onda nije } xIy,$$

$$\text{ako } xIy, \text{ onda nije } xPy \text{ i nije } yPx.$$

Nezavisno od toga kakve su preferencije između dve alternative, tj. da li se x smatra boljom od y ili obrnuto, ili su jednako dobre, pretpostavlja se da su one *relativno stabilne*, tj. *da se ne menjaju u periodu između izbora akcije i njene realizacije*. Relativna stabilnost preferencija, međutim, nikako ne znači da će se pri ponovljenim izborima između x i y uvek birati ista opcija. Postoji relativno mali broj alternativa među kojima se donosilac odluke uvek i bezuslovno opredeljuje za jednu od njih. Takve su *striktne* ili *bezuslovne preferencije* i one su određene etičkim principima, religijom, zdravstvenim razlozima i slično. U ostalim slučajevima preferencije se formiraju pod uticajem brojnih faktora i njihovih različitih kombinacija, zbog

čega ih karakteriše fleksibilnost, tj. *relativna nestabilnost*, kao i tendencija promene tokom vremena.

Kompletnost - Za bilo koje dve alternative, x i y, ili se x preferira u odnosu na y, ili y preferira u odnosu na x, ili postoji indiferentnost između njih, tj.

$$xPy \text{ ili } yPx \text{ ili } xIy.$$

Kompletnost zahteva da je donosilac odluke, bez obzira na stepen sličnosti ili različitosti alternativa među kojima bira, uvek u stanju da odredi preferencije. Mada deluje kao veoma blag, ovaj uslov ne može se uvek zadovoljiti. Neodlučnost se može javiti pri izboru između dve veoma povoljne ili nepovoljne alternative, ili kada se alternative među sobom toliko razlikuju da se ocenjuju na osnovu potpuno različitih kriterijuma. Ali, to nije isto što i svesno odlaganje odluke. Ako se konačan izbor odloži sa namerom da se preispitaju preferencije, donosi se specifična odluka koja štiti od brzopletog izbora neoptimalne opcije; u tom slučaju, odlaganje odluke može se smatrati racionalnim izborom.

Tranzitivnost - Za bilo koje tri opcije x, y, z, važi da ako se x preferira u odnosu na y i y preferira u odnosu na z, onda se x preferira u odnosu na z, tj.

$$\text{ako } xPy \text{ i } yPz, \text{ onda } xPz$$

i ako postoji indiferentnost između x i y, i između y i z, onda postoji indiferentnost i između x i z, tj.

$$\text{ako } xIy \text{ i } yIz, \text{ onda } xIz.$$

Ordinalna funkcija korisnosti

Opcijama rangiranim po preferencijama moguće je pridružiti brojeve koji odražavaju njihov relativan značaj ili korisnosti, a koje se nazivaju ordinalne funkcije korisnosti. Preciznije rečeno, ordinalna funkcija korisnosti je funkcija koja strukturu preferencija preslikava u skup realnih brojeva. Naziva se ordinalnom jer brojevi otkrivaju samo poredak opcija po preferencijama, pri čemu se veći broj pridružuje bolje rangiranoj alternativi.

2.2 Pareto optimalnost

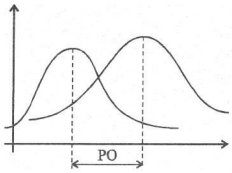
Činjenica da zadaci VKO po pravilu nemaju savršeno rešenje upućuje na preispitivanje koncepta optimalnosti i definicije optimalnog rešenja. Ključnu ulogu u tome ima koncept *Pareto optimalnosti*. To je proširenje poznatog koncepta optimalnosti koj se koristi u klasičnoj jednokriterijumskoj optimizaciji.

Pareto optimum se definiše na sledeći način

- Dopustivo rešenje $\mathbf{x}^*$ predstavlja *Pareto optimum* zadatka VKO ako ne postoji neko drugo dopustivo rešenje $\mathbf{x}$ takvo da važi

$$Q_j(\mathbf{x}) \geq Q_j(\mathbf{x}^*) \quad \forall j = 1, \ldots, l$$

pri čemu bar jedna od nejednakosti prelazi u strogu nejednakost $>$. Drugim rečima, $\mathbf{x}$ je Pareto optimum ako bi poboljšanje vrednosti bilo koje funkcije cilja prouzrokovalo pogoršanje vrednosti neke druge funkcije cilja. Za Pareto oprimum postoje sledeći sinonimi: *efikasno, dominantno i nedominirano rešenje.*



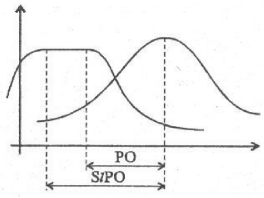
Slika 2.2.1. Geometrijska interpretacija Pareto optimalnih rešenja

Pored Pareto optimuma definišu se slabi i strogi (jaki) Pareto optimumi.

Dopustivo rešenje $\mathbf{x}^*$ je *slabi Pareto optimum* ako ne postoji neko drugo dopustivo rešenje x takvo da važi

$$Q_j(\mathbf{x}) > Q_j(\mathbf{x}^*) \quad \forall j = 1, \ldots, l.$$

Drugim rečima $\mathbf{x}^*$ je slabi Pareto optimum ako nije moguće istovremeno poboljšati sve funkcije cilja.



Slika 2.2.2. Geometrijska interpretacija Pareto optimalnih i Slabo Pareto optimalnih rešenja

Pareto optimalno rešenje $\mathbf{x}^*$ je *strogi Pareto optimum* ako postoji broj $\beta > 0$ takav da za svaki indeks $j \in \{1, \ldots, l\}$ i za svako $\mathbf{x}$ koje zadovoljava uslov

$$Q_j(\mathbf{x}) > Q_j(\mathbf{x}^*)$$

postoji bar jedno $i \in \{1, \ldots, l\} \setminus \{j\}$ takvo da je

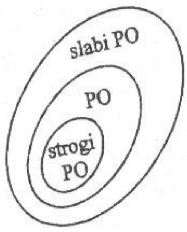
$$Q_i(\mathbf{x}) > Q_i(\mathbf{x}^*)$$

i da važi

$$\frac{Q_j(\mathbf{x}) - Q_j(\mathbf{x}^*)}{Q_i(\mathbf{x}^*) - Q_i(\mathbf{x})} \geq \beta.$$

Strogi Pareto optimum izdvaja ona Pareto rešenja čije promene ne prouzrokuju prevelike relativne promene u funkcijama cilja.

Odnos između opisanih optimuma je takav da svaki skup strožijih Pareto optimuma predstavlja podskup slabijih optimuma, tj. svaki Pareto optimum je istovremeno i slabi Pareto optimum, a svaki strogi Pareto optimum je i Pareto optimum. Odnos svih skupova dat je na slici.



Slika 2.2.3. Odnos između Pareto optimalnih rešenja

Primer 2.2.1 Za sledeći matematički model naći rešenje koristeći se metodima višeciljnog odlučivanja.

$$\begin{array}{ll} \max & f_1(\mathbf{x}) = 2x_1 + 3x_2 \\ \max & f_2(\mathbf{x}) = x_1 - 5x_2 \\ \max & f_3(\mathbf{x}) = -3x_1 + 4x_2 \\ \text{p.o.} & g1(\mathbf{x}): \quad 2x_1 + 0x_2 \leq 10 \\ & g_2(\mathbf{x}): \quad 0x_1 + 3x_2 \leq 9 \\ & g_3(\mathbf{x}): \quad 4x_1 + 0x_2 \leq 32 \\ & g_4(\mathbf{x}): \quad 5x_1 + 0x_2 \geq 5. \end{array}$$

Rešenje. Najpre se određuju marginalna rešenja, koja predstavljaju optimalna rešenja za pojedine kriterijume, a potom i idealne vrednosti funkcija svakog od kriterijuma.

$$\begin{array}{rcl} 2x_1 + 0x_2 \leq 10 & \Rightarrow & x1 \leq 5 \\ 0x_1 + 3x_2 \leq 9 & \Rightarrow & x2 \leq 3 \\ 4x_1 + 0x_2 \leq 32 & \Rightarrow & x1 \leq 8 \\ 5x_1 + 0x_2 \geq 5 & \Rightarrow & x1 \geq 1 \\ x_1, x_2 \geq 0 & \Rightarrow & x_1, x_2 \geq 0 \end{array}$$

U koordinatnom sistemu $(\mathbf{x}) = (x_1, x_2)$ određene su tačke (1,0), (5,0), (5,3), (1,3), koje predstavljaju marginalna rešenja za funkcije kriterijuma.

Idealna vrednost funkcije kriterijuma $f_1(\mathbf{x})$

$$\begin{array}{ll} \max & f_1(\mathbf{x}) = 2x_1 + 3x_2 \\ \text{p.o} & g_1(\mathbf{x}): \quad 2x_1 + 0x_2 \leq 10 \\ & g_2(\mathbf{x}): \quad 0x_1 + 3x_2 \leq 9 \\ & g_3(\mathbf{x}): \quad 4x_1 + 0x_2 \leq 32 \\ & g_4(\mathbf{x}): \quad 5x_1 + 0x_2 \geq 5 \\ & x_1, x_2 \geq 0 \end{array}$$

$$\begin{aligned} f_1(1,0) &= 2 \cdot 1 + 3 \cdot 0 = 2 \\ f_1(5,0) &= 2 \cdot 5 + 3 \cdot 0 = 10 \\ f_1(5,3) &= 2 \cdot 5 + 3 \cdot 3 = 19 \\ f_1(1,3) &= 2 \cdot 1 + 3 \cdot 3 = 11 \end{aligned}$$

Idealna vrednost funkcije kriterijuma $f_1(\mathbf{x})$ jednaka je maksimalnoj vrednosti funkcije $f_1^*(\mathbf{x}) = 19$ za marginalno rešenje $\mathbf{x}^{(1)*} = (5, 3)$.

Idealna vrednost funkcije kriterijuma $f_2(\mathbf{x})$

$$\begin{aligned} \max \quad & f_2(\mathbf{x}) = x_1 - 5x_2 \\ \text{p.o} \quad & g_1(\mathbf{x}): \quad 2x_1 + 0x_2 \leq 10 \\ & g_2(\mathbf{x}): \quad 0x_1 + 3x_2 \leq 9 \\ & g_3(\mathbf{x}): \quad 4x_1 + 0x_2 \leq 32 \\ & g_4(\mathbf{x}): \quad 5x_1 + 0x_2 \geq 5 \\ & x_1, x_2 \geq 0 \end{aligned}$$

$$\begin{aligned} f_2(1,0) &= 1 \cdot 1 - 5 \cdot 0 = 1 \\ f_2(5,0) &= 1 \cdot 5 - 5 \cdot 0 = 5 \\ f_2(5,3) &= 1 \cdot 5 - 5 \cdot 3 = -10 \\ f_2(1,3) &= 1 \cdot 1 - 5 \cdot 3 = -14 \end{aligned}$$

Idealna vrednost funkcije kriterijuma $f_2(\mathbf{x})$ jednaka je maksimalnoj vrednosti funkcije $f_2^*(x) = 5$ za marginalno rešenje $\mathbf{x}^{(2)*} = (5, 0)$.

Idealna vrednost funkcije kriterijuma $f_3(\mathbf{x})$.

$$\begin{aligned} \max \quad & f_3(\mathbf{x}) = -3x_1 + 4x_2 \\ \text{p.o} \quad & g_1(\mathbf{x}): \quad 2x_1 + 0x_2 \leq 10 \\ & g_2(\mathbf{x}): \quad 0x_1 + 3x_2 \leq 9 \\ & g_3(\mathbf{x}): \quad 4x_1 + 0x_2 \leq 32 \\ & g_4(\mathbf{x}): \quad 5x_1 + 0x_2 \geq 5 \\ & x_1, x_2 \geq 0 \end{aligned}$$

$$\begin{aligned} f_3(1,0) &= -3 \cdot 1 + 4 \cdot 0 = -3 \\ f_3(5,0) &= -3 \cdot 5 + 4 \cdot 0 = -15 \\ f_3(5,3) &= -3 \cdot 5 + 4 \cdot 3 = -3 \\ f_3(1,3) &= -3 \cdot 1 + 4 \cdot 3 = 9 \end{aligned}$$

Idealna vrednost funkcije kriterijuma $f_3(\mathbf{x})$ jednaka je maksimalnoj vrednosti funkcije $f_3^*(x) = 9$ za marginalno rešenje $\mathbf{x}^{(3)*} = (1, 3)$.

Tabelarni prikaz posledice marginalnih rešenja na funkcije kriterijuma i skup ograničenja. Prve dve kolone tabele sadrže vrednosti promenljivih, zatim sledeće tri kolone sadrže ostvarene vrednosti kriterijumskih funkcija, dok poslednje četiri kolone sadrže ostvarene vrednosti ograničenja.

x_1	x_2	f_1	f_2	f_3	$g_1 \leq 10$	$g_2 \leq 9$	$g_3 \leq 32$	$g_4 \geq 5$
5	3	**19**	-10	-3	10	9	20	25
5	0	10	**5**	-15	10	0	20	25
1	3	11	-14	**9**	2	9	4	5

3 Metodi za rešavanje zadataka VKO

Pre izlaganja različitih metoda za rešavanje problema VKO napomenimo da se one uglavnom zasnivaju na svođenju problema na problem jednokriterijumske optimizacije (JKO).

Prilikom razmatranja metoda trebalo bi obratiti pažnju na sledeća pitanja [21]:

1. Da li je metoda kvantitativna ili kvalitativna?

2. Kako se izbor donosioca odluke uključuje u algoritam?
3. Da li donosilac odluke kontroliše izbor konačnog rešenja?
4. Kakve informacije i odgovori se očekuju od donosioca odluke?
5. Koje rezultate metoda pruža kao pomoć odlučivanju?
6. Na koju vrstu problema može biti primenjena?
7. Računski aspekti (programiranje, memorija i dr.)

Prema pristupu rešavanju zadatka metodi VKO dele se u sledeće tri grupe:

1) *A posteriori metodi* u kojima se donosilac odluke (DO) informiše o dominantnim (Pareto optimalnim) rešenjima matematičkog modela, a on na osnovu njih donosi konačnu odluku.

2) *A priori metodi* u kojima se informacije o odnosu DO prema kriterijumima ugrađuju u matematički model ili metodu *a priori*, tj. pre bilo kakvog rešavanja modela, a konačna odluka se odnosi na osnovu tako dobijenog rešenja.

3) *Interaktivni metodi* u kojima DO aktivno učestvuje tokom rešavanja modela. U njima se iterativno kombinuju metodi iz prethodne dve grupe, tj. DO prvo daje preliminarne informacije o svojim preferencijama, a zatim kada dobije rešenje, može promeniti informacije ili rešenje. Ovaj postupak se iterativno ponavlja sve dok DO ne bude konačno zadovoljan dobijenim rešenjem.

Od a priori metoda obradićemo relaksirani leksikografski metod i metod ε ograničenja i objasniti princip grupe metoda nazvanih metodi rastojanja. Kao predstavnika interaktivnih metoda opisaćemo metod interaktivnog kompromisnog programiranja.

Svi do sada razvijeni metodi višeciljnog odlučivanja (VCO) imaju sledeće karakteristike:

- skup ciljeva koji mogu biti kvantifikovani,
- skup dobro definisanih ograničenja, i
- proces dobijanja informacija (eksplicitnih ili implicitnih) o identifikovanim ciljevima.

Poslednja osobina je posebno značajna. Naime, većinu realnih ciljeva je vrlo teško kvantifikovati, pa je za korišćenje metoda iz ove grupe potrebno raspolagati procesom koji bi bio u stanju da obezbedi određeni nivo kvantifikacije svih ciljeva.

Poznati svetski autori saglasni su da se većina realnih problema može rešavati primenom interaktivnih metoda, kada donosilac odluke aktivno učestvuje u kreiranju i analizi efikasnih rešenja, a na kraju procesa bira konačno, najprihvatljivije rešnje.

U nastavku sledi opis najvažnijih metoda višekriterijumske optimizacije kao i opis njihove simboličke implementacije u programskom paketu MATHEMATICA. Opis implementacije je baziran na radu [25].

3.1 Metod globalnog kriterijuma

Metod globalnog kriterijuma izuzetno je jednostavan i ne zahteva preferencije o kriterijumima. Nakon određivanja idealnih vrednosti kriterijuma formira se pomoćni jednokriterijumski model sa ograničenjima kao u modelu i funkcijom kriterijuma

$$\min f_r(\mathbf{x}) = \sum_k \left(\frac{f_k^*(\mathbf{x}) - f_k(\mathbf{x})}{f_k^*(\mathbf{x})} \right)^r, \ r \geq 1.$$

Rešenje problema jednako je minimalnoj vrednosti jednokriterijumskog modela, a predstavlja odabranu varijantu zbira normalizovanih odstojanja ostvarenih vrednosti od idealnih vrednosti kriterijuma.

Primer 3.1.1 Razmatra se napred prikazani primer i najjednostavnija varijanta formiranja funkcije kriterijuma u pomoćnom modelu metoda globalnog kriterijuma kada je $r = 1$.

$$\begin{array}{lll} \max & f_1(\mathbf{x}) = 2x_1 + 3x_2 \\ \max & f_2(\mathbf{x}) = x_1 - 5x_2 \\ \max & f_3(\mathbf{x}) = -3x_1 + 4x_2 \\ \text{p.o.} & g1(\mathbf{x}): & 2x_1 + 0x_2 \leq 10 \\ & g_2(\mathbf{x}): & 0x_1 + 3x_2 \leq 9 \\ & g_3(\mathbf{x}): & 4x_1 + 0x_2 \leq 32 \\ & g_4(\mathbf{x}): & 5x_1 + 0x_2 \geq 5. \end{array}$$

Rešenje. Idealne vrednosti kriterijuma

$$f_1^*(\mathbf{x}) = 19, \ f_2^*(\mathbf{x}) = 5, \ f_3^*(\mathbf{x}) = 9.$$

Globalna funkcija (za $r = 1$) jednaka je

$$\begin{aligned} \min f_{r=1}(x) &= \frac{f_1^*(\mathbf{x}) - f_1(\mathbf{x})}{f_1^*(\mathbf{x})} + \frac{f_2^*(\mathbf{x}) - f_2(\mathbf{x})}{f_2^*(\mathbf{x})} + \frac{f_3^*(\mathbf{x}) - f_3(\mathbf{x})}{f_3^*(\mathbf{x})} \\ &= \frac{19 - (2x_1 + 3x_2)}{19} + \frac{5 - (x_1 - 5x_2)}{5} + \frac{9 - (-3x_1 + 4x_2)}{9} \end{aligned}$$

$$\min f_{r=1}(x) = 3 + 0.03x_1 0.40x_2.$$

x_1	x_2	$\min f_{r=1}(\mathbf{x}) = 3 + 0.03x_1 + 0.40x_2$
5	3	4.3
5	0	**3.1**
1	3	4.2

Odatle se dobija $\min f_{r=1}(\mathbf{x}) = 3 + 0.03x_1 0.40x_2 = 3.1$ za $\mathbf{x}^* = (5, 0)$.

3.2 Metod težinskih koeficijenata

Metod težinskih koeficijenata je najstariji metod za VKO-u koja je korišćena. Po ovom metodu uvode se težinski koeficijenti w_i za sve kriterijumske funkcije $Q_i(\mathbf{x})$, $i = 1, \ldots, l$, pa se problem optimizacije svodi na sledeću skalarnu optimizaciju

$$\begin{array}{ll} \max & Q(\mathbf{x}) = \sum_{i=1}^{l} w_i Q_i(\mathbf{x}) \\ \text{p.o.} & \mathbf{x} \in \mathbf{X}, \end{array} \tag{3.2.1}$$

gde težine w_i, $i=1,\ldots,l$ ispunjavaju sledeće uslove

$$\sum_{i=1}^{l} w_i = 1, \quad w_i \geq 0,\ i = 1,\ldots,l.$$

Često se koristi metod težinskih koeficijenata tako što se zadaju vrednosti ovih koeficijenata. Međutim, to uvek izaziva određene teškoće i primedbe na ovakav postupak, jer se unosi subjektivan uticaj na konačno rešenje preko zadatih vrednosti težinskih koeficijenata.

Glavna ideja u metodu težinskih koeficijenata je da se odaberu težinski koeficijenti w_i koji odgovaraju ciljnim funkcijama $Q_i(\mathbf{x})$, $i = 1,\ldots,l$. Mnogi autori su razvili sistematske pristupe u selektovanju težina, čiji se pregled može naći u [10], [11] i [28]. Jedna od poteškoća ovog metoda je da variranje težina konzistentno i neprekidno ne mora uvek da rezultuje u tačnoj i kompletnoj reprezentaciji Pareto optimalnog skupa. Ovaj nedostatak je diskutovan u [8].

Teorema 3.2.1 *Ako su svi težinski koeficijenti w_i pozitivni, onda je rešenje problema* (3.2.1) *Pareto optimalno rešenje polaznog problema VKO.*

Dokaz. Neka je $\mathbf{x}^*$ rešenje problema (3.2.1), i neka su svi težinski koeficijenti strogo pozitivni. Pretpostavimo da ono nije Pareto optimalno, tj. da postoji $\mathbf{x} \in S$ tako da za $i = 1,\ldots,l$ važi $Q_i(\mathbf{x}) \geq Q_i(\mathbf{x}^*)$, pri čemu važi bar jedna stroga nejednakost (recimo za indeks j). Kako je $w_i > 0$ za svako i, važi

$$\sum_{i=1}^{l} w_i Q_i(\mathbf{x}) > \sum_{i=1}^{l} w_i Q_i(\mathbf{x}^*)$$

pa dobijamo kontradikciju sa pretpostavkom da je $\mathbf{x}^*$ rešenje težinskog problema (3.2.1). Sledi da je $\mathbf{x}^*$ Pareto optimalno. □

Teorema 3.2.2 *Ako je za svako $i \in \{1,\ldots,l\}$ ispunjen uslov $w_i \geq 0$, onda je rešenje problema* (3.2.1) *slabi Pareto optimum polaznog problema VKO.*

Dokaz. Neka je $\mathbf{x}^*$ rešenje problema (3.2.1) i da je ispunjen uslov $w_i \geq 0$. Pretpostavimo da ono nije slabo Pareto optimalno, tj. da postoji $\mathbf{x} \in S$ tako da za $i = 1,\ldots,l$ važi $Q_i(\mathbf{x}) > Q_i(\mathbf{x}^*)$. Svi koeficijenti w_i su nenegativni i bar jedan je strogo veći od nule (zbog $\sum_{i=1}^{l} w_i = 1$), pa važi

$$\sum_{i=1}^{l} w_i Q_i(\mathbf{x}) > \sum_{i=1}^{l} w_i Q_i(\mathbf{x}^*)$$

pa dobijamo kontradikciju sa pretpostavkom da je $\mathbf{x}^*$ rešenje težinskog problema. Dakle, $\mathbf{x}^*$ je slabi Pareto optimum. □

Teorema 3.2.3 *Ako je rešenje problema* (3.2.1) *jedinstveno, ono je onda i Pareto optimalno.*

Dokaz. Neka je $\mathbf{x}^*$ jedinstveno rešenje problema (3.2.1). Pretpostavimo da ono nije Pareto optimalno rešenje problema VKO, tj. da postoji $\mathbf{x} \in S$ tako da za $i = 1, \ldots, l$ važi $Q_i(x) \geq Q_i(x^*)$, pri čemu važi bar jedna stroga nejednakost (recimo za indeks j). Primetimo da to znači $\mathbf{x} \neq \mathbf{x}^*$.

Kako je $w_i \geq 0$ za svako i, važi

$$\sum_{i=1}^{l} w_i Q_i(\mathbf{x}) \geq \sum_{i=1}^{l} w_i Q_i(\mathbf{x}^*)$$

Ako bi važila stroga nejednakost, onda $\mathbf{x}^*$ ne bi bilo rešenje problema (3.2.1). Dakle, važi jednakost. To znači da postoje dva različita rešenja $\mathbf{x}$ i $\mathbf{x}^*$ problema (3.2.1), što je kontradikcija. □

Pokazaćemo sada jedno jače tvrđenje.

Teorema 3.2.4 *Ako su svi $w_i > 0$, $i \in \{1, \ldots, l\}$, tada je rešenje problema* (3.2.1) *strogi Pareto optimum problema VKO.*

Dokaz. Neka je $\mathbf{x}^*$ rešenje težinskog problema. Pokazali smo da je ono Pareto optimalno. Dokažimo da je to rešenje takođe i strogi Pareto optimum sa konstantom

$$M = (k-1) \max_{i,j} \frac{w_j}{w_i}.$$

Pretpostavimo suprotno, da postoje $\mathbf{x} \in S$ i indeks i takvi da je $Q_i(\mathbf{x}) > Q_i(\mathbf{x}^*)$ pri čemu za svako j za koje je $Q_j(\mathbf{x}^*) > Q_j(\mathbf{x})$ važi $Q_i(\mathbf{x}^*) - Q_i(\mathbf{x}) < M(Q_j(\mathbf{x}) - Q_j(\mathbf{x}^*))$. Zamenom

$$M = \frac{(k-1)w_j}{w_i}$$

dobijamo

$$w_i \frac{Q_i(\mathbf{x}^*) - Q_i(\mathbf{x})}{k-1} < w_j \left(Q_j(\mathbf{x}) - Q_j(\mathbf{x}^*)\right) > 0 \ .$$

Dakle, za svako $j \neq i$ za koje je $Q_j(\mathbf{x}^*) > Q_j(\mathbf{x})$ važi prethodna nejednakost. Za one indekse $j \neq i$ za koje je $Q_j(\mathbf{x}^*) \leq Q_j(\mathbf{x})$ gornja nejednakost svakako važi. Dakle, za svako $j \neq i$ važi

$$w_i \frac{Q_i(\mathbf{x}^*) - Q_i(\mathbf{x})}{k-1} < w_j \left(Q_j(\mathbf{x}) - Q_j(\mathbf{x}^*)\right),$$

pa sabiranjem ovih nejednakosti za $j = 1, \ldots, i-1, i+1, \ldots, l$ dobijamo

$$w_i \left(Q_i(\mathbf{x}^*) - Q_i(\mathbf{x})\right) < \sum_{j=1, j\neq i}^{l} w_j \left(Q_j(\mathbf{x}) - Q_j(\mathbf{x}^*)\right)$$

tj.

$$\sum_{j=1}^{l} w_j Q_j(\mathbf{x}^*) < \sum_{j=1}^{l} w_j Q_j(\mathbf{x}) \ .$$

Dobijamo da $\mathbf{x}^*$ nije rešenje težinskog problema tj. dolazimo do kontradikcije. Zaključujemo da je $\mathbf{x}^*$ zaista strogi Pareto optimum polaznog problema. □

Težinski koeficijent na neki način treba da predstavi značaj kriterijumske funkcije kojoj je dodeljen. Da bismo to postigli najpre moramo da normalizujemo kriterijumske funkcije tj. da ih promenimo tako da imaju približno jednake vrednosti, a pri tome da zadrže sve bitne osobine. Na primer, ako je kriterijumska funkcija linearna $Q_j(\mathbf{x}) = \sum_{i=1}^{n} a_i x_i$, tada će normalizovan oblik ove funkcije biti

$$\frac{Q_j(\mathbf{x})}{\sum_{i=1}^{n} a_i}\,.$$

Ako je kriterijumska funkcija ograničena tada se za njen normalizovan oblik može uzeti sledeća funkcija

$$\overline{Q_i}(\mathbf{x}) = \frac{Q_i(\mathbf{x}) - Q_i^*}{\max_{\mathbf{x}\in S} Q_i(\mathbf{x}) - Q_i^*}\,,$$

čiji je kodomen $[0, 1]$.

Ukoliko donosilac odluke sam definiše težinske koeficijente, onda ovaj metod spada u grupu a priori metoda. Međutim, najčešće se malo zna o tome kako koeficijente treba izabrati. Zato se obično težinski problem rešava za razne vrednosti vektora $(w_1, \ldots, w_l)$ i na taj način dobijaju različita rešenja među kojima DO bira ono koje mu najviše odgovara. Ako se koristi ovakav pristup, onda ovaj metod postaje a posteriori metod.

Glavna mana ove metode je teškoća određivanja težinskih koeficijenata kada nemamo dovoljno informacija o problemu.

Sledi opis implementacije metoda težinskih koeficijenata.

Funkcijom `Compositions[n,l]` možemo odrediti "l-dimenzionalne tačke", čiji zbir koordinata daje n. Kada tu listu podelimo sa brojem n, možemo dobiti tačke u intervalu $[0, 1]$ čije koordinate mogu predstavljati koeficijente w_i. Na taj način smo obezbedili automatsko generisanje koeficijenata w_i potrebnih za realizaciju metoda. Ostavljena je i mogućnost da korisnik sam izabere koeficijente w_i. To se postiže tako što pri pozivu funkcije `MultiW`, kojom se implementira metod težinskih koeficijenata, poslednji parametar zadamo kao nepraznu listu, tj. zadamo koeficijente w_i u obliku liste.

Sledi program kojim je implementiran metod težinskih koeficijenata [25].

Ulazne veličine:

q_,var_List - ciljna funkcija i lista njenih parametara;

constr_List - lista ograničenja;

*var_* - korak podele intervala [0,1];

w1_List - lista težinskih koeficijenata u intervalu [0,1]. Prazna lista kao vrednost parametra $w1$ označava da će težinski koeficijentu biti generisani pomoću funkcije *Compositions*. Inače, pretpostavlja se da je svaki element $w1[[i]], 1 \leq 1 \leq Length[w1]$ liste $w1$ jedan mogući skup koeficijenata w_j, $j = 1, \ldots, l$: $w[[j]] = w1[[i, j]], j = 1, \ldots, l$.

Lokalne promenljive:
fun - formirana funkcija za jednodimenzionalnu optimizaciju.

```
MultiW[q_, constr_List, var_List, w1_List] :=
 Module[{i=0,k,l=Length[q],res={},W={},fun,sk={},qres={},mxs={},m,ls={}},
   If[w1=={},
    k=Input["Initial sum of weighting coefficients?"]; W=Compositions[k,l]/k,
    W=w1;
   ];
   Print["Weighting Coefficients: "]; Print[W];
   Print["Single-objective problems: "];
   k = Length[W];
   For[i=1, i<=k, i++,
       fun = Simplify[Sum[W[[i,j]]*q[[j]], {j,l}]];  (* 3V *)
       AppendTo[res, Maximize[fun, constr, var]];    (* 1V *)
   ];
   Print["Solutions of single-objective problems: "];  Print[res];
   For[i=1, i<=k, i++,
      AppendTo[qres, q/.res[[i, 2]] ];     AppendTo[mxs, res[[i, 1]] ];
   ];
   Print["Choose the best solution: "];
   m=Max[mxs];
   For[i=1, i<=Length[mxs], i++,
      If[m==mxs[[i]],  AppendTo[ls, {qres[[i]], res[[i,2]]}]; ]
   ];
   Return[ls];
 ]
```

Za pozitivne težine i konveksni problem, optimalna rešenja jednokriterijumskog problem jesu Pareto optimalna [33], tj. minimiziranje odgovarajućeg jednokriterijumskog problema je dovoljno za Pareto optimalnost. Međutim, formulacija ne obezbeđuje neophodan uslov za Pareto optimalnost [34]. Kada je višekriterijumski problem konveksan, primena funkcije `W=Compositions[k,l]` produkuje b Pareto optimalnih rešenja, gde integer b ispunjava $1 \leq b \leq \binom{k+l-1}{l-1}$.

Primer 3.2.1 Maksimizirati funkcije koje se nalaze u listi:

$$Q = \{Q_1(x,y) = x + y, \;\; Q_2(x,y) = 2x - y\}$$

prema ograničenjima

$$-3x + 5y \leq 9, \; 3x + 2y \quad \leq \quad 12, \; 5x - 4y \leq 9, \; x \geq 0, \; y \geq 0.$$

Prvo se koristi prazna lista za parametar $w1$, i na taj način se koeficijenti w_i generišu pomoću funkcije *Compositions*:

```
In[1]:=MultiW[{x+y,2x-y},{-3x+5y<=9,3x+2y<=12,5x-4y<=9,x>=0,y>=0},{x,y},{}]
```

U slučaju $k = 5$, izraz `w=Compositions[k,l]/k` produkuje

w={{0, 1},{$\frac{1}{5}$, $\frac{4}{5}$}, {$\frac{2}{5}$, $\frac{3}{5}$},{$\frac{3}{5}$, $\frac{2}{5}$}, {$\frac{4}{5}$, $\frac{1}{5}$}, {1, 0}}

Takođe, jednokriterijumski problemi optimizacije dati su sledećim unutrašnjim reprezentacijama.

Za $\{w_1, w_2\}$={0, 1}:
{2x-y,{$-3x + 5y <= 9, 3x + 2y <= 12, 5x - 4y <= 9, x >= 0, y >= 0$},{$x, y$}}

Za $\{w_1, w_2\}$={$\frac{1}{5}$, $\frac{4}{5}$}:
{$\frac{3}{5}(3x - y)$, {$-3x + 5y <= 9, 3x + 2y <= 12, 5x - 4y <= 9, x >= 0, y >= 0$},{$x, y$}}

Za $\{w_1, w_2\}$=$\{\frac{2}{5}, \frac{3}{5}\}$:
$\{\frac{1}{5}(8x - y)$, $\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x, y\}\}$

Za $\{w_1, w_2\}$=$\{\frac{3}{5}, \frac{2}{5}\}$:
$\{\frac{1}{5}(7x + y)$, $\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x, y\}\}$

Za $\{w_1, w_2\}$=$\{\frac{4}{5}, \frac{1}{5}\}$:
$\{\frac{3}{5}(2x + y)$, $\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x, y\}\}$

Za $\{w_1, w_2\}$=$\{1, 0\}$:
$\{x + y$, $\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x, y\}\}$

Elementi liste *res*, koji odgovaraju rešenjima jednokriterijumskih problema, jednaki su

$\{\{\frac{9}{2}$,$\{x->3$,$y->\frac{3}{2}\}\}$,$\{\{\frac{9}{2}$,$\{x->3$,$y->\frac{3}{2}\}\}$,$\{\{\frac{9}{2}$,$\{x->3$,$y->\frac{3}{2}\}\}$,
$\{\{\frac{9}{2}$,$\{x->3$,$y->\frac{3}{2}\}\}$,$\{\{\frac{9}{2}$,$\{x->3$,$y->\frac{3}{2}\}\}$,$\{\{5$,$\{x->2$,$y->3\}\}\}$

a rezultat je lista koja sadrži vrednosti ciljnih funkcija u tački $\{x-> 2, y-> 3\}$, koja odgovara maksimalnoj vrednosti težinske funkcije (koja je jednaka 5):

Out[1]=$\{\{\{5$,$1\}$, $\{x-> 2, y-> 3\}\}$

Sada se vrednosti koeficijenata w_i definišu od strane korisnika. Vrednosti težinskih koeficijenata sadržane su u svakom elementu liste
$\{\{1, 0\}, \{0.9, 0.1\}, \{0.875, 0.125\}, \{0.8, 0.2\}, \{0, 1\}\}$.

```
In[2]:=MultiW[{x+y,2x-y},{-3x+5y<=12,5x-4y<=9,x>=0,y>=0},{x,y},
                {{1,0},{0.9,0.1},{0.875,0.125},{0.8,0.2},{0,1}}]
```

Sledeći problemi se rešavaju u ciklusu:

Za $\{w_1, w_2\}$=$\{1, 0\}$:
$\{x + y$,$\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x$,$y\}\}$
Za $\{w_1, w_2\}$=$\{0.9, 0.1\}$:
$\{1.1x + 0.8y$,$\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x$,$y\}\}$
Za $\{w_1, w_2\}$=$\{0.875, 0.125\}$:
$\{1.125x + 0.75y$,$\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x$,$y\}\}$
Za $\{w_1, w_2\}$=$\{0.8, 0.2\}$:
$\{1.2x + 0.6y$,$\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x$,$y\}\}$
Za $\{w_1, w_2\}$=$\{0, 1\}$:
$\{2x - y$,$\{-3x + 5y <= 9$,$3x + 2y <= 12$,$5x - 4y <= 9$,$x >= 0$,$y >= 0\}$,$\{x$,$y\}\}$

Dobija se sledeća vrednost za listu *res* koja odgovara rešenjima ovih problema:

$\{\{5$,$\{x->2$,$y->3\}\}$,$\{4.6$,$\{x->2.$,$y->3.\}\}$,$\{4.5$,$\{x->3.$,$y->1.5\}\}$,
$\{4.5$,$\{x->3.$,$y->1.5\}\}$,$\{\frac{9}{2}$,$\{x->3$,$y->\frac{3}{2}\}\}\}$

a rezultat je

Out[2]= $\{\{\{5,1\},\{x-> 2,y-> 3\}\}\}$

3.3 Metod sa funkcijom korisnosti

Za primenu ovog metoda neophodno je znati preferencije za kriterijume koje se unapred uključuju u model. Preferencije se određuju na osnovu analize značajnosti kriterijuma za svaki konkretan slučaj, i to je najvažnija, kardinalna informacija o

problemu. Najpre se formira pomoćni jednokriterijumski model u vidu funkcije korisnosti $U(f)$

$$\max U(f) = U\{f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_l(\mathbf{x})\}$$

koja se rešava uz primenu postojećih ograničenja (p.o.)

$$g_i(\mathbf{x}) \leq 0,\ i \in I$$
$$x_j \geq 0,\ \text{za svako } j \in J.$$

Potom se nađeno optimalno rešenje uvodi u svaki od kriterijuma i određuju njihove konkretne vrednosti. Funkcija korisnosti zavisi od prirode rešavanog problema. Najčešće se koriste sledeće separabilne funkcije od zadatih funkcija modela višeciljnog odlučivanja:

$$\begin{aligned} U(f) &= \sum_k f_k(\mathbf{x}) \\ U(f) &= \prod_k f_k(\mathbf{x}) \\ U(f) &= \sum_k t_k f_k(\mathbf{x}). \end{aligned}$$

Definicija 3.3.1 Funkcija $U : \mathbf{R}^k \to \mathbf{R}$ kojoj je domen kriterijumski prostor i koja odražava težnje donosioca odluke zove se vrednosna funkcija ili funkcija korisnosti.

Boljoj odluci odgovara veća vrednost vrednosne funkcije. Iz tog razloga, vrednosna funkcija bi trebalo da bude strogo opadajuća po svakoj promenljivoj. Dakle, problem se svodi na problem jednokriterijumske optimizacije

$$\max\ U(f(\mathbf{x}))\ p.o.\ \mathbf{x} \in S. \tag{3.3.1}$$

Definicija 3.3.2 *Funkcija $f : \mathbf{R}^k \to \mathbf{R}$ je strogo rastuća (opadajuća) ako za proizvoljne $\mathbf{x} = (x_1, \ldots, x_k)$ i $\mathbf{y} = (y_1, \ldots, y_k)$ iz $\mathbf{R}^k$ važi, redom*

$$(x_i < y_i \text{ za svako } i = 1, \ldots, k) \Rightarrow f(\mathbf{x}) < f(\mathbf{y}).$$

$$(x_i \leq y_i \text{ za svako } i = 1, \ldots, k) \Rightarrow f(\mathbf{x}) > f(\mathbf{y})).$$

Definicija 3.3.3 *Funkcija $f : \mathbf{R}^k \to \mathbf{R}$ je jako rastuća (opadajuća) ako za proizvoljne $\mathbf{x} = (x_1, \ldots, x_k)$ i $\mathbf{y} = (y_1, \ldots, y_k)$ iz $\mathbf{R}^k$ važi, redom*

$$(x_i \leq y_i \text{ za svako } i = 1, \ldots, k \text{ i } x_j < y_j \text{ za neko } j) \Rightarrow f(\mathbf{x}) < f(\mathbf{y}).$$

$$(x_i \leq y_i \text{ za svako } i = 1, \ldots, k \text{ i } x_j < y_j \text{ za neko } j) \Rightarrow f(\mathbf{x}) > f(\mathbf{y}).$$

Teorema 3.3.1 *Za jako opadajuću funkciju U rešenje problema* (3.3.1) *je Pareto optimalno rešenje početnog višekriterijumskog problema.*

Dokaz. Pretpostavimo suprotno, tj. da neko rešenje problema (3.3.1) (označimo ga sa $\mathbf{x}^*$) nije Pareto optimalno. To znači da postoji $x \in S$ za koje važi da je $f_i(\mathbf{x}) \leq f_i(\mathbf{x}^*)$, pri čemu postoji indeks j za koji važi stroga nejednakost. Kako je funkcija U jako opadajuća, sledi $U(f(\mathbf{x})) > U(f(\mathbf{x}^*))$. Dobijamo kontradikciju sa pretpostavkom da se maksimum funkcije $U \circ f$ dostiže u tački $\mathbf{x}^*$. □

Metod funkcije korisnosti je jednostavan i jako dobar u slučajevima kada ga je moguće primeniti. Međutim, u praksi su česti slučajevi kada je neprekidnom stabilnom funkcijom jako teško ili nemoguće izraziti želje donosioca odluke. Osim toga može se desiti da donosilac odluke nije u stanju da strogo matematički izrazi svoje težnje ili da ih delimično promeni nakon što bude upoznat sa rezultatima.

Primer 3.3.1 Rešava se problem

$$\begin{aligned}
\max \quad & f_1(\mathbf{x}) = 2x_1 + 3x_2 \\
\max \quad & f_2(\mathbf{x}) = x_1 - 5x_2 \\
\max \quad & f_3(\mathbf{x}) = -3x_1 + 4x_2 \\
\text{p.o.} \quad & g1(\mathbf{x}): \quad 2x_1 + 0x_2 \leq 10 \\
& g_2(\mathbf{x}): \quad 0x_1 + 3x_2 \leq 9 \\
& g_3(\mathbf{x}): \quad 4x_1 + 0x_2 \leq 32 \\
& g_4(\mathbf{x}): \quad 5x_1 + 0x_2 \geq 5.
\end{aligned}$$

sa sledećim težinama za kriterijume $t_1 = 0.6, t_2 = 0.3$ i $t_3 = 0.1$.

Uvođenjem vrednosti težinskih koeficijenata u funkciju korisnosti dobija se

$$\max U(f) = t_1 \cdot f_1(\mathbf{x}) + t_2 \cdot f_2(x) + t_3 \cdot f_3(x)$$
$$\max U(f) = 0.6(2x_1 + 3x_2) + 0.3(x_1 - 5x_2) + 0.1(-3x_1 + 4x_2),$$

ili posle sređivanja

$$\max U(f) = 0.9x_1 0.7x_2.$$

Maksimum se određuje iz sledeće tabele

x_1	x_2	$\max U(f) = 0.9x_1 0.7x_2$	f_1	f_2	f_3
5	3	2.4			
5	0	**4.5**	5	10	-15
1	3	-1.2			

Rešenje. $\max f(\mathbf{x}) = 0.9x_1 0.7x_2 = 4.5$ za $\mathbf{x}* = (5, 0)$.

3.4 Metod ograničavanja kriterijuma

Metod ograničavanja kriterijuma jedan je od najstarijih metoda rešavanja modela višeciljnog odlučivanja, čiji se elementi direktno ili indirektno uključuju u procedure drugih metoda. Po ovom metodu vrši se optimizacija najznačajnijeg kriterijuma, neka je to s-ti kriterijum, dok se ostali prevode u ograničenja sa zahtevima da se ostvare željene vrednosti tih kriterijuma. Kao i kod metoda sa funkcijom korisnosti, i u ovom metodu se preferencije o kriterijumima zadaju unapred (a priori) i to je

najvažnija, kardinalna informacija. Prvo se definiše jednokriterijumski pomoćni model koji odgovara polaznom modelu višeciljnog odlučivanja

$$\begin{aligned} \max \quad & f_s(\mathbf{x}) \\ \text{p.o.} \quad & g_i(\mathbf{x}) \leq 0,\ i = 1, 2, ..., l \\ & f_k(\mathbf{x}) \begin{cases} \geq L_k \\ \leq H_k \end{cases}, k \neq s, k = 1, 2, ..., l \\ & x_j \geq 0,\ j = 1, 2, ..., n. \end{aligned}$$

gde veličine L_k i H_k predstavljaju donju i gornju granicu, respektivno, za k-te kriterijume prevedene u ograničenja, $k \neq s$, $k = 1, 2, ..., l$. Određivanjem optimalnog rešenja modela dobija se uslovljena optimalna vrednost s-tog kriterijuma, koja se uvodi u ostale kriterijume radi proračuna njihovih vrednosti. U tu svrhu mogu se koristiti odgovarajuće granice i izravnavajuće promenljive pripradajućih ograničenja. Za razliku od prethodnih metoda višeciljnog odlučivanja koji zahtevaju određivanje maksimalnih vrednosti, ovaj metod se sprovodi uvođenjem samo donjih granica za k-te kriterijume u ograničenja. Pri tome se mora imati na umu koje su moguće vrednosti tih kriterijuma, što se određuje na osnovu analize tabele sa idealnim vrednostima kriterijuma i posledicama marginalnih rešenja na kriterijume.

Primer 3.4.1 Neka se u napred razmatranom primeru smatra da je prvi kriterijum ($s = 1$) od najvećeg značaja. Potrebno je odrediti maksimalnu vrednost tog kriterijuma sa zahtevom da se ostalim kriterijumima ostvare najmanje 60% idealnih vrednosti.

$$0.60 f_2^*(\mathbf{x}) = 0.60 \cdot 5 = 3.0$$
$$0.60 f_3^*(\mathbf{x}) = 0.60 \cdot 9 = 5.4$$

x_1	x_2 f_1	$g_1 \leq 10$	$g_2 \leq 9$	$g_3 \leq 32$	$g_4 \geq 5$	$g_5 \geq 3$	$g_6 \geq 5.4$	
5	3	**19**	10	9	20	25	-10	-3
5	0	10	10	0	20	25	5	-15
1	3	11	2	9	4	5	-14	9

Optimalno Rešenje je $f_1(\mathbf{x}) = 19$ za $\mathbf{x}^{(1)*} = (5, 3)$.

3.5 Leksikografska višekriterijumska optimizacija

Često se do optimalnog rešenja dolazi posle uzastopnog donošenja odluka. Prvo se nađe optimalno rešenje za najvažniju funkciju cilja. Ako je optimalno rešenje jedinstveno, tada je problem rešen. Međutim, ako optimalno rešenje nije jedinstveno, tada se na skupu svih optimalnih rešenja optimizuje funkcija cilja koja je druga po važnosti. Ako je optimalno rešenje sada jedinstveno, problem je rešen; ako nije, optimizuje se funkcija cilja treća po važnosti na skupu optimalnih rešenja prve i druge funkcije cilja itd.

Dakle posmatrajmo problem minimizacije date uređene sekvence ciljnih funkcija

$$Q_1(x), \ldots, Q_l(x)$$

i skup ograničenja

$$\mathbf{x} \in \mathbf{X}.$$

Trebalo bi da se reši sledeći skup konveksnih uslovnih nelinearnih programa

$$\begin{array}{ll} \max & Q_i(\mathbf{x}) \\ \text{p.o.} & \mathbf{x} \in \mathbf{X} \\ & Q_j(\mathbf{x}) \geq a_j, \quad j = 1, \ldots, i-1,\ i \geq 2, \end{array}$$

gde su $a_j = Q_j(\mathbf{x_j}^*)$, $j = 1, \ldots, i-1$ optimalne vrednosti prethdono postavljenih jednokriterijumskih problema na nivoima prioriteta $j = 1, \ldots, i-1$, $i \geq 2$.

Nejednakosti u poslednjem problemu mogu biti zamenjene jednakostima [24]. leksikografski metod spada u grupu apriornih metoda.

Pretpostavimo da su kriterijumske funkcije $Q_1, \ldots, Q_l$ poređane od najvažnije do najmanje važne. Polazni problem višekriterijumske optimizacije se rešava sledećim algoritmom

1. $S_0 := S,\ i := 1$
2. Rešava se problem:

   $$\text{Minimizirati } Q_i(\mathbf{x}) \text{ pod uslovom } \mathbf{x} \in S_{i-1}$$

   Neka je rešenje ovog problema $\mathbf{x}^{(i)^*}$.
3. Ako je $\mathbf{x}^{(i)^*}$, dobijeno u koraku 2, jedinstveno rešenje, ono se proglašava za rešenje višekriterijumskog problema i algoritam se završava.
4. Formira se skup $S_i := \{\mathbf{x} \in S_{i-1} \mid Q_i(\mathbf{x}) \geq Q_i(\mathbf{x}^{(i)^*})\}$
5. Ako je $i = l$ skup S_l se proglašava za skup rešenje problema višekriterijumske optimizacije, i algoritam se završava.
6. Stavlja se $i := i + 1$ i vraćamo se na korak 2.

Jedan od načina za implementaciju višekriterijumske optimizacije dat je funkcijom *MultiLex* [25].

```
MultiLex[q_List,constr_List,var_List]:=
  Module[{res={},s,f=constr,l=Length[q] ,i},
     For[i=1,i<=l,i++,
         s=Maximize[q[[i]],f,var];       (* 2V *)
         If[i<l, AppendTo[f,q[[i]]>=First[s]]  ];   (* 3V,2V *)
         AppendTo[res,{q/.Last[s],Last[s]}];
     ];
     Print[res]; Return[{q/.Last[s],Last[s]}];
  ]
```

Primer 3.5.1 Problem maksimizacije funkcija koje su sadržane u listi
$\{8x1 + 12x2, 14x1 + 10x2, x1 + x2\}$
prema ograničenjima
$\{8x1 + 4x2 <= 600, 2x1 + 3x2 <= 300, 4x1 + 3x2 <= 360, 5x1 + 10x2 >= 600,$

$x1 >= 0, x2 >= 0\}$
može biti rešen koristeći izraz

```
In[3]:=MultiLex[{8x1+12x2,14x1+10x2,x1+x2},
   {8x1+4x2<=600,2x1+3x2<=300,4x1+3x2<=360,5x1+10x2>=600,x1>=0,x2>=0},
   {x1,x2}]
```

Unutrašnje reprezentacije jednokriterijumskih problema su:

za $i = 1$:

$8x1 + 12x2,$
$\{8x1 + 4x2 <= 600,\ 2x1 + 3x2 <= 300,\ 4x1 + 3x2 <= 360,\ 5x1 + 10x2 >= 600,$
$x1 >= 0, x2 >= 0\},$
$\{x1, x2\}$

za $i = 2$:

$14x1 + 10x2,$
$\{8x1 + 4x2 <= 600,\ 2x1 + 3x2 <= 300,\ 4x1 + 3x2 <= 360,\ 5x1 + 10x2 >= 600,$
$x1 >= 0,\ x2 >= 0,\ 8x1 + 12x2 >= 1200\},$
$\{x1, x2\}$

za $i = 3$:

$x1 + x2,$
$\{8x1 + 4x2 <= 600,\ 2x1 + 3x2 <= 300,\ 4x1 + 3x2 <= 360,\ 5x1 + 10x2 >= 600,$
$x1 >= 0,\ x2 >= 0,\ 8x1 + 12x2 >= 1200,\ 14x1 + 10x2 >= 1220\},$
$\{x1, x2\}$

Rešenja uzastopnih optimizacionih problema su zapamćena u listi res, koja je jednaka

```
{{1200,{x1->0,x2->100}},{1220,{x1->30,x2->80}},{110,{x1->30,x2->80}}}
```

Teorema 3.5.1 *Rešenje dobijeno leksikografskom metodom je Pareto optimalno rešenje problema višekriterijumske optimizacije.*

Dokaz. Označimo sa $\mathbf{x}^*$ rešenje dobijeno leksikografskim metodom. Pretpostavimo da ono nije Pareto optimalno tj. da postoji neko $x \in S$ tako da je $Q_i(x) \geq Q_i(x^*)$ za svako $i = 1, \ldots, l$ pri čemu za neko k važi $Q_k(\mathbf{x}) > Q_k(\mathbf{x}^*)$. Mogu nastati dva slučaja. Prva mogućnost je da je $\mathbf{x}^*$ jedinstveno rešenje i da su iskorišćeni svi kriterijumi do j-tog (to znači da je na kraju prethodno opisanog algoritma $i = j$). Kako je $\mathbf{x} \in S = S_0$ i $Q_1(\mathbf{x}) \geq Q_1(x^*)$ sledi da je $Q_1(\mathbf{x}) = Q_1(\mathbf{x}^*)$, kao i $x \in S_1$. Kako je sada $x \in S_1$ i $Q_2(x) \geq Q_2(x^*)$ sledi $Q_2(x) = Q_2(x^*)$ i $\mathbf{x} \in S_2$. Nastavljajući ovakvo rezonovanje zaključujemo $Q_{j-1}(\mathbf{x}) = Q_{j-1}(\mathbf{x}^*)$ i $\mathbf{x} \in S_{j-1}$. Kako je $\mathbf{x}^*$ jedinstveno rešenje problema

$$\text{Minimizirati } Q_j(\mathbf{x}) \text{ pod uslovom } \mathbf{x} \in S_{j-1}$$

a pri tome važi $Q_j(\mathbf{x}) \geq Q_j(\mathbf{x}^*)$ i $\mathbf{x} \in S_{j-1}$ zaključujemo da mora biti $\mathbf{x} = \mathbf{x}^*$ pa je $Q_k(\mathbf{x}) = Q_k(\mathbf{x}^*)$ odakle dobijamo kontradikciju sa pretpostavkom $Q_k(\mathbf{x}) > Q_k(\mathbf{x}^*)$.

Druga mogućnost je da su iskorišćeni svi kriterijumi (tj. u prethodnom algoritmu je $i = l$). Potpuno analogno kao u prvom slučaju se pokazuje da $\mathbf{x} \in S_k$ i da za svako $i = 1, \ldots, k$ važi $Q_i(\mathbf{x}) = Q_i(\mathbf{x}^*)$, što je kontradikcija sa $Q_k(\mathbf{x}) > Q_k(\mathbf{x}^*)$. □

Iz ovog dokaza se može zaključiti da se ograničenje oblika $Q_j(x) \leq Q_j(\mathbf{x}^{(i)*})$ može zameniti ograničenjem $Q_j(\mathbf{x}) = Q_j(\mathbf{x}^{(i)*})$.

Mane leksikografske metode su očigledne:

- U praksi je često teško odrediti koji je kriterijum važniji od drugog.
- Najčešće se dešava da se jedinstveno rešenje dobije pre nego što se iskoriste svi kriterijumi, ili čak nakon korišćenja samo jednog kriterijuma. Na taj način, neki kriterijumi uopšte ne učestvuju u donošenju odluke, što je krajnje nepoželjno.

Zbog toga klasična leksikografski medod ima jako ograničenu primenu.

3.6 Relaksirani leksikografski metod

Hijerarhijski metod je modifikacija leksikografskog metoda, u kome se koriste ograničenja oblika [22]

$$Q_j(\mathbf{x}) \geq \left(1 + \frac{\delta_j}{100}\right) Q_j(\mathbf{x_j}^*).$$

Relaksacija se sastoji u povećanju desne strane ograničenja za procenat $Q_j(\mathbf{x_j}^*) \cdot \delta_j$. Variranjem parametra δ_j mogu se generisati različite Pareto optimalne tačke [18]. Još jedna varijacija leksikografskog metoda je uvedena u [30], gde su ograničenja jednaka

$$Q_j(\mathbf{x}) \geq Q_j(\mathbf{x_j}^*) - \delta_j.$$

U ovom slučaju, $\delta_j \leq 100$ jesu pozitivne tolerancije definisane od strane DO.

Relaksirani leksikografski metod je iterativni postupak u kome se rešavaju jednokriterijumski zadaci optimizacije. Pretpostavlja se da su od strane DO dati prioriteti kriterijuma i da su u skladu sa njima dodeljeni indeksi kriterijumima. U relaksiranoj leksikografskoj metodi se po svakom od p kriterijuma rešava jednokriterijumski zadatak optimizacije. Pri tome se u narednoj iteracijine postavlja kao ograničenje zahtev da rešenje bude optimalno po kriterijumu višeg prioriteta, već se ono relaksira tako da se zahteva da rešenje bude u okolini optimalnog rešenja dobijenog u prethodnoj iteraciji. Nataj način, svaki kriterijum utiče na konačno rešenje.

DO zadaje redosled kriterijuma po značajnosti. Pored toga, svakom kriterijumu, uzimajući poslednji, dodeljuje se vrednost δ_k , $k = 1, \ldots, l-1$, za koju kriterijum višeg prioriteta sme da odstupi od svoje optimalne vrednosti. Metoda obuhvata izvršavanje sledećih l koraka

Korak 1. Rešiti problem

$$\begin{array}{ll} \max & Q_1(\mathbf{x}) \\ \text{p.o.} & \mathbf{x} \in \mathbf{X} \end{array}$$

Rešenje ovog problema označimo sa Q_1^*.

Korak 2. Rešiti problem

$$\begin{array}{ll} \max & Q_2(\mathbf{x}) \\ \text{p.o.} & \mathbf{x} \in \mathbf{X} \\ & Q_1(\mathbf{x}) \geq Q_1^* - \delta_1 \end{array}$$

Korak p**.** Za svako $3 \leq p \leq l$ rešiti jednokriterijumski problem

$$\begin{array}{ll} \max & Q_p(\mathbf{x}) \\ \text{p.o.} & \mathbf{x} \in \mathbf{X} \\ & Q_j(\mathbf{x}) \geq Q_j^* - \delta_j,\ j = 1, \ldots, l-1 \end{array}$$

Za rešenje polaznog modela se usvaja rezultat dobijen u poslednjem koraku, a vrednosti funkcija cilja za dobijeno rešenje se moraju posebno računati.

Rešenje dobijeno ovim metodom obezbeđuje slabi Pareto optimum, a ako je rešenje jedinstveno, ono je i Pareto optimalno. Ako je dopustiva oblast konveksna, podešavanjem parametara $\delta_k, k = 1, \ldots, l-1$, može se dobiti bilo koje Pareto optimalno rešenje.

Teorema 3.6.1 *Rešenje dobijeno relaksiranim leksikografskim metodom je slabi Pareto optimum problema višekriterijumske optimizacije.*

Dokaz. Pretpostavimo suprotno, da rešenje $\mathbf{x}^*$ dobijeno relaksiranim leksikografskim metodom nije slabo Pareto optimalno. To znači da postoji neko $\mathbf{x} \in S$ tako da za svako $i = 1, \ldots, l$ važi $Q_i(\mathbf{x}) \geq Q_i(x^*)$. Kako je $Q_i(\mathbf{x}^*) = Q_i^*$ tim pre važi $Q_i(\mathbf{x}) \geq Q_i^* - \delta_i,\ i = 1, \ldots, k-1$, pa kako je $Q_k(\mathbf{x}) > Q_k(\mathbf{x}^*)$ dobijamo kontradikciju sa pretpostavkom da je $\mathbf{x}^*$ rešenje dobijeno relaksiranim leksikografskom metodom. □

Teorema 3.6.2 *Ako se relaksiranim leksikografskim metodom dobija jedinstveno rešenje, ono je Pareto optimalno.*

Dokaz. Pretpostavimo suprotno, da jedinstveno rešenje $\mathbf{x}^*$ dobijeno relaksiranim leksikografskim metodom nije Pareto optimalno. To znači da postoji neko $\mathbf{x} \in S$ tako da za svako $i = 1, \ldots, l$ važi $Q_i(\mathbf{x}) \geq Q_i(\mathbf{x}^*)$ pri čemu za neko $j \in \{1, \ldots, l\}$ važi stroga nejednakost. Razlikujemo dva slučaja:

1) $Q_k(\mathbf{x}) > Q_k(\mathbf{x}^*)$. Kako iz pretpostavljenog sledi da je $Q_i(\mathbf{x}) \geq Q_i^* - \delta_i$, $i = 1, \ldots, k-1$, zaključujemo da $\mathbf{x}^*$ nije rešenje dobijeno u koraku k gore opisanog algoritma tj. da $\mathbf{x}^*$ nije rešenje dobijeno relaksiranim leksikografskim metodom, što je kontradikcija.

2) $Q_k(\mathbf{x}) = Q_k(\mathbf{x}^*)$. Kao i pod 1), važi $Q_i(\mathbf{x}) \geq Q_i^* - \delta_i,\ i = 1, \ldots, k-1$, pa dobijamo da je $\mathbf{x}$ rešenje dobijemo relaksiranim leksikografskim metodom. Međutim, kako je $\mathbf{x}^*$ jedinstveno rešenje zaključujemo da mora biti $\mathbf{x} = \mathbf{x}^*$. Tada je i $Q_j(\mathbf{x}) = Q_j(\mathbf{x}^*)$, što je kontradikcija.

Dakle, $\mathbf{x}^*$ je zaista Pareto optimalno rešenje. □

U praktičnoj primeni leksikografskog metoda vrednosti δ_i moraju biti pažljivo odabrane. Ukoliko se ovim metodom dobije više rešenja bilo bi dobro smanjiti te vrednosti.

Primer 3.6.1 Primenom relaksiranog leksikografskog metoda rešiti sledeći zadatak VKO (funkcije cilja su relaksirane po prioritetu)

$$\begin{array}{ll} \max & \{Q_1(\mathbf{x}), Q_2(\mathbf{x}), Q_3(\mathbf{x})\} \\ \text{p.o.} & 2x_1 + x_2 \leq 6,\ x_1 + 3x_2 \leq 6,\ x_1 \geq 0,\ x_2 \geq 0. \end{array}$$

gde je

$$\begin{array}{lcll} Q_1(\mathbf{x}) & = & x_1 + 4x_2, & \delta_1 = 1 \\ Q_2(\mathbf{x}) & = & x_1, & \delta_2 = 1 \\ Q_3(\mathbf{x}) & = & x_1 + x_2. & \end{array}$$

Rešenje.
Korak 1. Rešiti problem

$$\begin{array}{ll} \max & Q_1(x) = x_1 + 4x_2 \\ \text{p.o.} & 2x_1 + x_2 \leq 6,\ x_1 + 3x_2 \leq 6,\ x_1, x_2 \geq 0. \end{array}$$

Ovaj zadatak ima jedinstveno optimalno rešenje u tački $x^{1^*} = (0, 2)$, pri čemu je $Q_1^* = 8$.

Korak 2. Rešiti problem

$$\begin{array}{ll} \max & Q_2(\mathbf{x}) = x_1 \\ \text{p.o.} & 2x_1 + x_2 \leq 6,\ x_1 + 3x_2 \leq 6,\ x_1, x_2 \geq 0,\ x_1 + 4x_2 \geq 7. \end{array}$$

Dobija se rešenje $x^{2^*} = (2.43, 1.14), Q_2^* = 2.43$
Korak 3. Rešiti problem

$$\begin{array}{ll} \max & Q_3(\mathbf{x}) = x_1 + 4x_2 \\ \text{p.o.} & 2x_1 + x_2 \leq 6,\ x_1 + 3x_2 \leq 6,\ x_1, x_2 \geq 0,\ x_1 + 4x_2 \geq 7, x_1 \geq 1.43. \end{array}$$

Rešenje ovog zadatka se usvaja kao konačno. To je rešenje $x_1^* = 3.6,\ x_2^* = 1.2$. Vrednosti funkcija cilja u ovoj tački su $Q^* = (7.2, 2.4, 3.6)$.

Relaksirani leksikografski metod je veoma osetljiv na izbor koeficijenata δ_j, tako da se "pogrešnim" izborom njegove vrednosti mogu dobiti neprihvatljiva rešenja. Tako u prethodnom primeru imamo slučaj da se konačno resenje poklapa sa marginalnim rešenjem funkcije cilja koja ima najniži prioritet, dok je vrednost najznačajnijeg kriterijuma smanjena. Kod primene ove metode se preporučuje da DO kritički preispita dobijena rešenja, uporedi ih sa marginalnim i da po potrebi koriguje zadate koeficijente.

Primer 3.6.2 Vlasnik hotela pre početka sezone odlučuje o opremanju soba nameštajem. Hotel raspolaže sa ukupno 58 soba, od kojih su 16 male, tako da mogu da budu samo jednokrevetne, dok ostale mogu biti jednokrevetne, dvokrevetne ili trokrevetne. Podaci o ceni opremanja soba i sezonskoj zaradi po sobi su dati u sledećoj tabeli

	Jednokrevetna	Dvokrevetna	Trokrevetna
Cena opremanja	20000	40000	60000
Sezonska zarada	15000	25000	30000

Tabela 4.6.1.

Za opremanje soba, vlasnik može da izdvoji 2.5 miliona dinara. On želi da postigne dva cilja

1. da ostvari što veću sezonsku zaradu hotela i
2. da omogući primanje što većeg broja gostiju kako ne bi došlo do toga da gostima bude uskraćeno gostoprimstvo.

a) Formulisati matematički model za određivanje optimalnog opremanja soba potrebnim nameštajem (broj kreveta) ako se žele ostvariti oba postavljena kriterijuma.

b) Rešiti zadatak ako je prvi kriterijum prioritetan i ako je vlasnik spreman da "žrtvuje" 50.000 din. svoje sezonske zarade da bi poboljšao drugi kriterijum.

Rešenje.

a) Promenljive x_1, x_2 i x_3 će predstavljati broj jednokrevetnih, dvokrevetnih i trokrevetnih soba u hotelu, respektivno. Tada matematički model ima sledeći oblik

$$\begin{array}{ll} \max & [Q_1(\mathbf{x}), Q_2(\mathbf{x})] \\ \text{p.o.} & x_1 + x_2 + x_3 \leq 58, \\ & 20.000x_1 + 40.000x_2 + 60.000x_3 \leq 2.500.000, \\ & x_1 \geq 16,\ x_2 \geq 0,\ x_3 \geq 0. \end{array}$$

gde su

$$\begin{aligned} Q_1(\mathbf{x}) &= 15.000x_1 + 25.000x_2 + 30.000x_3 \\ Q_2(\mathbf{x}) &= x_1 + x_2 + 3x_3 \end{aligned}$$

b) Iako nije eksplicitno naglašeno, na osnovu zahteva zadatka je očigledno da se traži rešenje dobijeno relaksiranim leksikografskim metodom, s tim da je najvažniji prvi kriterijum, a zadati koeficijent $\alpha_1 = 50.000$. U prvom koraku rešavamo model

$$\begin{array}{ll} \max & Q_1(\mathbf{x}) = 15.000x_1 + 25.000x_2 + 30.000x_3 \\ \text{p.o.} & \mathbf{x} \in \mathbf{X}, \end{array}$$

čije je rešenje $Q_1^* = 1.415.000$ din, $x_1^{1^*} = 16$, $x_2^{1^*} = 17$, $x_3^{1^*} = 25$. U drugoj iteraciji rešavamo model u koji smo dodali ograničenje koji obezbeđujemo da se prvi kriterijum ne "pokvari" za više od 50.000 ($1.415.000 - 50.000 = 1.365.000$)

$$\begin{array}{ll} \max & Q_2(\mathbf{x}) = x_1 + 2x_2 + 3x_3 \\ \text{p.o.} & \mathbf{x} \in \mathbf{X},\ 15.000x_1 + 25.000x_2 + 30.000x_3 \geq 1.365.000. \end{array}$$

Rešenje ovog zadatka je konačno rešenje problema. Ono, međutim, nije jedinstveno. Štaviše, simpleks metodom se mogu generisati četiri rešenja:

$$x^{*I} = (23, 0, 34),\ x^{*II} = (16, 7, 31.67),\ x^{*III} = (24.5, 0, 33.5),\ x^{*IV} = (16, 17, 25).$$

Sva optimalna rešenja se mogu dobiti konveksnom kombinacijom ova četiri. Grafički posmatrano, u trodimenzionalnom prostoru se dobija da su optimalna rešenja deo ravni oivičen trapezoidom čija su temena date četiri tačke. U svim tačkama koje se nalaze unutar tog trapezoida, vrednost druge funkcije cilja iznosi 125, dok vrednost prve varira između 1.365.000 i 1.415.000. Dakle, dobili smo skup tačaka koje predstavljaju slaba Pareto rešenja. Jedino tačka x^{*IV}, koja je optimalna i po prvom kriterijumu, je Pareto optimalno rešenje.

Zaključak. Da bi vlasnik hotela na što bolji način zadovoljio oba kriterijuma, najbolje je da opremi 16 jednokrevetnih, 17 dvokrevetnih i 25 trokrevetnih soba i tako ostvari zaradu od 1.415.000 din. po sezoni. Pri tome će hotel moći da prima 125 gostiju.

Jedan od načina za implementaciju relaksiranog leksikografskog metoda dat je funkcijom **MultiRelax**[] [25]:

Ulazne veličine:

q - lista ciljnih funkcija;

parcf - parametri ciljnih funkcija;

constr - lista ograničenja.

```
MultiRelax[q_List, constr_List, var_List, delta_List] :=
   Module[{res={}, s, f=constr, l=Length[q], i},
      For[i=1, i<=l, i++,
         s = Maximize[q[[i]], f, var];      (* 2V *)
         If[i<l, AppendTo[f, q[[i]]>=First[s]-delta[[i]] ]  ]; (* 3V,2V *)
         AppendTo[res, s];
      ];
      Print[res]; Return[{q/.Last[s],Last[s]}];
   ]
```

Primer 3.6.3 Problem vlasnika hotela rešava se izrazom

```
MultiRelax[{15000x1+25000x2+30000x3,x1+2x2+3x3},
  {x1+x2+x3<=58, 20000x1+40000x2+60000x3<=2500000,x1>=16,x2>=0,x3>=0},
  {x1,x2,x3},{50000}]
```

Dobija se sledeće rešenje:

```
{{1365000,125},{x1->23,x2->0,x3->34}}
```

3.7 Metod e-ograničenja

U ovom metodu DO izdvaja kriterijum $Q_p(\mathbf{x})$, $1 \leq p \leq l$ koji ima najviši prioritet i njega maksimizira, dok ostale funkcije cilja ne smeju imati vrednosti manje od unapred zadatih ε_k, $k = 1, \ldots, l$, $k \neq p$. Sve druge ciljne funkcije se koriste da se kreiraju dodatna ograničenja oblika $l_i \leq Q_i(\mathbf{x}) \leq \varepsilon_i$, $i = 1, \ldots, l$, $i \neq p$ [12]. U ovoj relaciji, l_i i ε_i jesu donja i gornja granica ciljne funkcije $Q_i(\mathbf{x})$, respektivno.

Haimes u [9] je uveo *e-constraint* (ili trade-off) metod (metod e-ograničenja), u kome su sve vrednostoi l_i isključene. Sistematsko variranje vrednosti ε_i produkuje skup Pareto optimalnih rešenja [12]. Međutim, nepodesna selekcija vrednosti $\varepsilon \in \mathbf{R}^k$ može da proizvede rešenje koje ne ispunjava ograničenje. Opšte pravilo za selektovanje ε_i je sledeće ([2])

$$Q_p(\mathbf{x_i}^*) \leq \varepsilon_i \leq Q_i(\mathbf{x_i}^*),$$

gde je $\mathbf{x_i}^*$ tačka koja maksimizira ciljnu funkciju $Q_i(\mathbf{x})$.

Neophodno je da se reši sledeći problem

$$\begin{array}{ll} \max & Q_p(\mathbf{x}) \\ \text{p.o.} & x \in \mathbf{X}, \\ & Q_i(\mathbf{x}) \geq \varepsilon_i,\ i = 1, \ldots, l,\ i \neq p, \end{array}$$

gde je $Q_p(\mathbf{x})$ odabrani kriterijum sa najvećim prioritetom i ϵ_i su odabrani realni brojevi. Ako optimalno rešenje ne postoji, neophodno je da se smanji vrednost za ε_i.

Teorema 3.7.1 *Rešenje problema ε ograničenja je slabo Pareto optimalno rešenje početnog problema VKO.*

Dokaz. Pretpostavimo da je $\mathbf{x}^*$ rešenje problema ε ograničenja, i da ono nije slabi Pareto optimum. Dakle, postoji $\mathbf{x} \in S$ tako da je $Q_j(\mathbf{x}) > Q_j(x^*) \wedge Q_i(\mathbf{x}) > Q_i(\mathbf{x}^*) \geq \varepsilon_i$ za svako $i = 1, \ldots, l$, $i \neq j$, odakle sledi da $\mathbf{x}^*$ nije rešenje problema ε ograničenja, što je kontradikcija. Zaključujemo da je rešenje problema ε ograničenja slabi Pareto optimum. □

Teorema 3.7.2 *Tačka $\mathbf{x}^* \in S$ je Pareto optimalno rešenje polaznog problema VKO akko za svako $j = 1, \ldots, l$ tačka $\mathbf{x}^*$ jeste rešenje problema ε ograničenja pri čemu je $\varepsilon_i = Q_i(\mathbf{x}^*)$, $i = 1, \ldots, l$, $i \neq j$.*

Dokaz. Uslov je dovoljan jer iz njega sledi da ni za jedno $j = 1, \ldots, l$ ne postoji $\mathbf{x} \in S$ tako da je $Q_j(\mathbf{x}) > Q_j(\mathbf{x}^*) \wedge Q_i(\mathbf{x}) \geq \varepsilon_i = Q_i(\mathbf{x}^*)$, $i \neq j$, tj. sledi da je $\mathbf{x}^*$ Pareto optimum.

Pretpostavimo, sa druge strane, da je $\mathbf{x}^*$ Pareto optimum, a da za neko j ono nije rešenje problema ε ograničenja sa $\varepsilon_i = Q_i(\mathbf{x}^*)$, $i \neq j$. Dakle, postoji $\mathbf{x} \in S$ tako da važi $Q_j(\mathbf{x}) > Q_j(\mathbf{x}^*)$ i $Q_i(\mathbf{x}) \geq Q_i(\mathbf{x}^*) = \varepsilon_i$, $i \neq j$. Dolazimo u kontradikciju sa pretpostavkom da je $\mathbf{x}^*$ Pareto optimalno rešenje. Dakle, uslov je i potreban. □

Primetimo da iz ove teoreme sledi da se metodom ε ograničenja može dobiti proizvoljno Pareto optimalno rešenje.

Teorema 3.7.3 *Neka je vektor ε proizvoljan. Tada, ako problem ε ograničenja ima jedinstveno rešenje onda je ono Pareto optimalno rešenje polaznog problema VKO.*

Dokaz. Neka je $\mathbf{x}^*$ jedinstveno rešenje problema ε ograničenja. Pretpostavimo da $\mathbf{x}^*$ nije Pareto optimalno tj. da postoji $\mathbf{x}'$ iz S tako da važi $Q_i(\mathbf{x}') \geq Q_i(\mathbf{x}^*)$ $i = 1, \ldots, l$, pri čemu važi stroga jednakost za $i = l$. Ako je $l = j$, tada dobijamo $Q_j(\mathbf{x}') > Q_j(\mathbf{x}^*) \wedge Q_i(\mathbf{x}') \geq Q_i(\mathbf{x}^*) \geq \varepsilon_i$, $i = 1, \ldots, l$, $i \neq j$, tj. dobijamo kontradikciju sa pretpostavkom da je $\mathbf{x}^*$ rešenje problema ε ograničenja. Ako je $l \neq j$, tada je $Q_i(\mathbf{x}') \geq Q_i(\mathbf{x}^*) \geq \varepsilon_i$, $i = 1, \ldots, l$, $i \neq j$ i $Q_j(x') \leq Q_j(x^*)$. Međutim kako je x^* jedinstveno rešenje te iz $Q_i(\mathbf{x}') \geq \varepsilon_i$, $i = 1, \ldots, l$, $i \neq j$ sledi da mora biti $Q_j(\mathbf{x}') < Q_j(\mathbf{x}^*)$. Ponovo dolazimo do kontradikcije pa zaključujemo da je $\mathbf{x}^*$ zaista Pareto optimalno rešenje polaznog problema VKO. □

U slučaju da problem ε ograničenja nema rešenja trebalo bi povećati vrednosti granica ε_i. Naravno, nijedno ε_i ne sme biti manje od idealne vrednosti funkcije cilja Q_i. U slučaju da DO sam izabere prioritetnu funkciju Q_j i konstante ε_i na ovaj metod se može gledati kao na a priori metod. Međutim, on u tom slučaju ne može biti siguran da će biti zadovoljan dobijenim rešenjem. Zbog toga je pristup obično drugačiji. Postupak rešavanja problema ε ograničenja se ponavlja za razne vrednosti granica ε_i i za razne funcije Q_j kao prioritetne, sve dok DO ne bude zadovoljan dobijenim rešenjem.

Da bismo dokazali da je rešenje dobijeno metodom ograničenja Pareto optimalno moramo dokazati da je ono jedinstveno ili da je rešenje l različitih problema za svaku od funkcija Q_j, što nije nimalo jednostavno. Ovo je glavna mana ove metode.

Sledi opis implementacije prema [25].

Ulazni parametri:

$q\_$, $constr\_$, $var\_$: unutrašnja forma problema;
$p\_$: redni broj ciljne funkcije sa najvećim prioritetom.

```
MultiEps[q_List, constr_List, var_List, p_Integer] :=
 Module[{s, f = constr, i, j, k, l = Length[q], eps, lb = {}, ub = {}},
   For[i = 1, i <= l, i++, (* Find bounds in (3.4)  *)
     xi = Maximize[q[[i]], constr, var];
     AppendTo[lb,q[[p]]/.First[Rest[xi]]];
     AppendTo[ub,q[[i]]/.First[Rest[xi]]];
   ];
   For[i = 1, i <= l, i++,
      For[eps = lb[[i]], eps <= ub[[i]], eps = eps + 0.1,
         f = constr;
         For[k = 1, k <= l, k++,
            If[k != p, AppendTo[f, q[[k]]>=eps ]  ]  (* 3V *)
         ];
         s = Maximize[q[[p]], f, var];   (* 2V, 1V *)
      ]
   ]
 ]
```

Primer 3.7.1 Rešiti problem:

$$\begin{array}{ll} \max: & \{Q_1(\mathbf{x}) = x_1,\ Q_2(\mathbf{x}) = x_2\} \\ \text{p.o.:} & 0 \le x_1 \le 1,\ 0 \le x_2 \le 1. \end{array}$$

Pretpostavimo da prvi kriterijum ima najveći prioritet. Ovaj problem se rešava sledećim pozivom funkcije *MultiEps*:

In[5]:= `MultiEps[{x1,x2},{x1>=0,x2>=0,x1<=1,x2<=1},{x1,x2},{0.5},1]`

U slučaju kada drugi kriterijum uzima vrednosti veće ili jednake od 0.5 ($eps = 0.5$) neophodno je da se reši sledeći jednokriterijumski problem:

$$\begin{array}{ll} \max & Q_1(\mathbf{x}) = x_1 \\ \text{p.o.} & 0 \le x_1 \le 1,\ 0 \le x_2 \le 1,\ x_2 \ge 0.5. \end{array}$$

Njegova unutrašnja forma, korišćena u MATHEMATICA funkciji `Maximize` je

$x1,\ \{x1>=0, x2>=0, x1<=1, x2<=1, x2>=0.5\},\ \{x1, x2\}$

Rešenje problema je

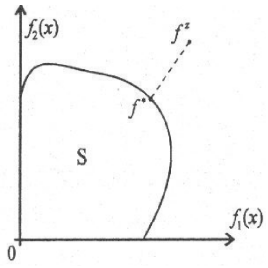
`{{1.,0.5},{x1->1., x2->0.5}}`

i nije Pareto optimalno. Pareto optimalno rešenje se generiše u slučaju $eps = 1$, i jednako je

`{{1.,1},{x1->1, x2->1}}`

3.8 Metodi rastojanja

Metodi rastojanja čine grupu za rešavanje zadataka VKO čija je osnovna ideja da se u kriterijumskom prostoru traži tačka koja je najbliža nekoj unapred određenoj tački koja se želi dostići ili ka kojoj treba težiti ako ona nije dopustiva. Drugim rečima, minimizira se rastojanje između željene tačke i dopustive oblasti. Razlike između metoda ove grupe potiču od toga kako se željena tačka određuje, na koji način se rastojanje od nje "meri", da li se uvode težinski koeficijenti, itd.



Slika 3.8.1. Tačka najbliža željenoj vrednosti f^z.

U opsťem slučaju, DO za svaki od l kriterijuma zadaje željene vrednosti ili određuje način kako će se one izračunati. Na taj način se u l -dimenzionalnom kriterijumskom prostoru definiše tačka

$$f^z = (f_1^z, ..., f_l^z)$$

koja po pravilu ne pripada dopustivoj oblasti S. U slučaju

$$f^z \in S$$

zadatak se rešava rešavanjem sistema jednačina koje su definisane skupom ograničenja.

U metodu rastojanja rešava se sledeći opšti zadatak

$$\begin{array}{ll} \min & d(f^z - f(x)) \\ \text{p.o.} & \mathbf{x} \in \mathbf{x} \end{array}$$

gde $d(*.*)$ označava rastojanje definisano pogodnom metrikom.

U kontekstu merenja rastojanja u kriterijumskom prostoru ovde ćemo ponoviti definicije metrika koje se najčešće koriste u metodima rastojanja

$$l_1 \ (pravougaona) \ metrika : d_{l_1}(f^z, f(\mathbf{x})) = \sum_{k=1}^{l} |f_k^z - f_k(\mathbf{x})| \,,$$

$$l_2 \ (Euklidova) \ metrika : d_{l_2}(f^z, f(\mathbf{x})) = \sqrt{\sum_{k=1}^{l} (f_k^z - f_k(\mathbf{x}))^2},$$

$$l_\infty \ (Čebiševljeva) \ metrika : d_{l_\infty}(f^z, f(\mathbf{x})) = \max_{1 \le k \le l} \{|f_k^z - f_k(\mathbf{x})|\} \,.$$

Kriterijumima je moguće dodeliti težinske koeficijente, tako da prethodne formule za rastojanje između željenog i traženog rešenja dobijaju oblik

$$d_{l_1}(f^z, f(x)) = \sum_{k=1}^{l} w_k \, |f_k^z - f_k(\mathbf{x})|$$

za pravougaonu metriku,

$$d_{l_2}(f^z, f(x)) = \sqrt{\sum_{k=1}^{l} w_k (f_k^z - f_k(x))^2}$$

za Euklidovu metriku,

$$d_{l_\infty}(f^z, f(\mathbf{x})) = \max_{1 \le k \le l} \{w_k \, |f_k^z - f_k(\mathbf{x})|\} .$$

za Čebiševljevu metriku.

Bez obzira na oblik polaznih funcija cilja, zadaci minimizacije rastojanja su problemi nelinearnog programiranja, za koje, u opštem slučaju, nije jednostavno pronaći optimalno rešenje. Izuzetno se u slučaju l_1 metrike i linearnih funkcija cilja i ograničenja zadatak minimizacije rastojanja može formulisati kao problem LP što ćemo objasniti na sledećem primeru.

Napomena 3.8.1 *Objašnjene transformacije su osnova za formulaciju zadatka višekriterijumskog linearnog programiranja (VLP) i oblik koji se naziva ciljno programiranje. Prema tome, zadatak ciljnog programiranja je poseban oblik zadatka metoda rastojanja u VKO. Dodatno je moguće kriterijumima dodeliti prioritete dostizanja željenih vrednosti i težinske koeficijente.*

Rešenje zadatka VKO dobijeno metodom rastojanja, kada skupu vrednosti kriterijuma ne pripada željna vrednost, predstavlja slabi Pareto optimum, a ako je jedinstveno, onda je i Pareto optimalno.

Metod rastojanja (ciljno programiranje ili goal programming method) je razvijen u [4], [5] [13]. Za svaku od l kriterijumskih funkcija odabiraju se ciljne vrednosti, ili se izračunavaju pomoću funkcije *Maximize*. Na taj način, generiše se l-dimenzionalna tačka $b(\mathbf{x}) = (b_1, \ldots, b_l)$, koja sadrži ciljne vrednosti za ciljne funkcije. Potom se minimizira rastojanje između tačke $b(\mathbf{x})$ i $Q(\mathbf{x}) = (q_1(\mathbf{x}), \ldots, q_l(\mathbf{x}))$. Drugim rečima, rešava se sledeći opšti zadatak

$$\begin{array}{ll} \min & d(b - Q(\mathbf{x})) \\ \text{p.o.} & \mathbf{x} \in \mathbf{X}, \end{array}$$

gde je $d(*.*)$ metrika koja definiše rastojanje između tačaka b i $Q(\mathbf{x})$. Uzmimo, na primer metriku definisanu L_1-normom

$$d(b - Q(\mathbf{x})) = \sum_{i=1}^{l} |b_i - q_i(\mathbf{x})|.$$

U ovom slučaju, prirodno je da se koriste zamene $y_i = b_i - q_i(\mathbf{x})$, $i = 1, \ldots, l$. Koristeći dodatne smene $y_i = \$[i]^- - \$[i]^+$, $i = 1, \ldots, l$, u kojima je $\$[i]^-, \$[i]^+ \ge 0$,

(i tada je $|y_i| = \$[i]^+ + \$[i]^-,\ i = 1, \ldots, l$), dobija se sledeći jednokriterijumski optimizationi problem

$$\begin{array}{ll} \min & \displaystyle\sum_{i=1}^{l}(\$[i]^+ + \$[i]^-) \\ \text{p.o.} & \mathbf{x} \in \mathbf{X} \\ & Q_i(\mathbf{x}) - \$[i]^+ + \$[i]^- = b_i,\ i = 1, \ldots, l \\ & \$[1]^+ \geq 0, \ldots, \$[l]^+ \geq 0, \$[1]^- \geq 0, \ldots, \$[l]^- \geq 0. \end{array}$$

Lista ograničenja se transformiše ciklusom:

```
For[i=1,i<=l,i++,
   AppendTo[constr,q[[i]]-$[2i-1]+$[2i]==b[[i]]];
];
For[i=1,i<=2l,i++, AppendTo[constr,$[i]>=0]  ];
```

Odgovarajuća jednokriterijumska ciljna funkcija se generiše i minimizira izrazom

```
s=Minimize[Array[$,2l,1,Plus],constr,Union[Variables[Array[$,2l,1]],var]];
```

Standardna funkcija `Variables[expr]` daje listu nezavsinih promenljivih u izrazu `expr`. Izraz `Union[l1,l2,...]` daje sortiranu listu različitih elemenata koji se pojavljuju u bilo kom od izraza l_i [32].

Primer 3.8.1 Dat je zadatak VLP

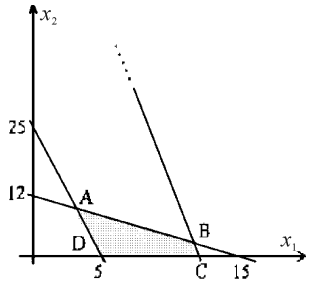
$$\begin{array}{ll} \max & [Q_1(\mathbf{x}), Q_2(\mathbf{x}), Q_3(\mathbf{x})] \\ \text{p.o.} & 35x_1 + 7x_2 \geq 175 \\ & 136x_1 + 170x_2 \leq 2400 \\ & 20x_1 + 3x_2 \leq 240 \\ & x_1, x_2 \geq 0 \end{array}$$

gde su

$$Q_1(\mathbf{x}) = 4x_1 + 5x_2,\ Q_2(\mathbf{x}) = x_1 + x_2,\ Q_3(\mathbf{x}) = 40x_1 + 6x_2.$$

a) Rešiti zadatak metodom rastojanja sa pravougaonom metrikom, ako se želi dostići idealna tačka.

b) Koja rešenja predstavljaju Pareto, a koja slabi Pareto optimum polaznog zadatka.



Slika 3.8.2. Skup dopustivih rešenja u primeru f^z.

Rešenje. a) U ovom zadatku DO je odredio da su željene vrednosti kriterijumskih funkcija idealne vrednosti tih funkcija. Zato je najpre potrebno odrediti marginalna rešenja i odgovarajuće idealne vrednosti funkcije

$f_1^* = 60$ za višestruko rešenje koje pripada duži **AB** sa krajnjim tačkama

$$x_1^{(1)^*} = 3.09,\ x_2^{(1)^*} = 9.52$$

i

$$x_1^{(1)^{**}} = 11.59,\ x_2^{(1)^{**}} = 2.73.$$

$f_2^* = 14.32$ za : $x_1^{(2)^*} = 11.59$, $x_2^{(2)^*} = 2.73$ (tačka **B**),
$f_3^* = 480$ za višestruko rešenje koje pripada duži **BC** sa krajnjim tačkama

$$x_1^{(3)^{**}} = 11.59,\ x_2^{(3)^{**}} = 2.73$$

i

$$x_1^{(3)^{**}} = 12,\ x_2^{(3)^{**}} = 0.$$

Zadatak VKO prilagođen za rešavanje metodom rastojanja sada ima oblik

$$\min\ \Phi(\mathbf{x}) = |f_1^* - f_1(\mathbf{x})| + |f_2^* - f_2(\mathbf{x})| + |f_3^* - f_3(\mathbf{x})|$$

pri istim ograničenjima.

Posto sve kriterijume trebamo maksimizirati, a po definiciji marginalnih rešenja je $f_k^* \geq f_k(x)$, može se staviti da je $|f_k^* - f_k(\mathbf{x})| = f_k^* - f_k(\mathbf{x})$, $k = 1, 2, 3$. Tada se umesto polaznog zadatka može rešavati zadatak minimizacije funkcije $\Phi(\mathbf{x}) = 554,32 - 45x_1 - 12x_2$, pri istim ograničenjima. Pokazaćemo kako se ovaj zadatak rešava opštim postupkom koji koristi definiciju prebačaja i podbačaja. U tom slučaju matematički model ima oblik

$$\begin{aligned}
\min\quad & F(x,y) = y_1^+ + y_1^- + y_2^+ y_2^- + y_3^+ + y_3^- \\
\text{p.o.}\quad & 4x_1 + 5x_2 - y_1^+ + y_1^- = 60 \\
& x_1 + x_2 - y_2^+ + y_2^- = 14,32 \\
& 40x_1 + 6x_2 - y_3^+ + y_3^- = 480 \\
& 35x_1 + 7x_2 \geq 175 \\
& 136x_1 + 170x_2 \leq 2400 \\
& 20x_1 + 3x_2 \leq 240 \\
& x_1, x_2, y_1^+, y_1^-, y_2^+, y_2^-, y_3^+, y_3^- \geq 0.
\end{aligned}$$

Rešavanjem ovog zadatka dobija se jedinstveno rešenje $F^* = 0$, $x_1^* = 11.59$ i $x_2^* = 2.73$.

To što je vrednost funkcije cilj jednako nuli ukazuje da je dobijeno rešenje savršeno jer nema ni prebačaja ni podbačaja vrednosti funkcija cilja od idealne tačke. Ovo je očigledno i na grafiku. Vidimo da u tački **B** sve funkcije cilja dostižu svoju najveću vrednost.

b) Jedino je tačka **B**(11.59, 2.73) Pareto optimalna. Sve tačke na duži **AB** na jednu stranu i **BC** na drugu stranu su slabi Pareto optimumi.

Sledi funkcija **MultiDist** kojom se daje implementacija metode rastojanja. Kao pomoćna funkcija koristi se funkcija **ideal** [25].

```
ideal[q_,constr_,var_]:=
  Module[{res={},i,l=Length[q]},
     For[i=1,i<=l,i++,
       AppendTo[res,First[Maximize[q[[i]],constr,var]]];
     ];
     Return[res];
  ]
```

```
MultiDist[q_,constr_,w_List,var_]:=
 Module[{con=constr,point={},l=Length[q],s},
   If[w=={}, point=ideal[q,con,var], point=w];
   For[i=1,i<=l,i++,
      AppendTo[con,q[[i]]-$[2i-1]+$[2i]==point[[i]]]; (* 3V,2V *)
   ];
   For[i=1,i<=2l,i++, AppendTo[con,$[i]>=0]  ];
   s=Minimize[Array[$,2l,1,Plus],con,Union[Variables[Array[$,2l,1]],var]];
   (* 3V,2V *)
   Return[{q/.Last[s],Last[s]}]
  ];
```

Primer 3.8.2 Rešiti problem

$$\begin{aligned} \max \quad & \{Q_1(\mathbf{x}) = x_1,\ Q_2(\mathbf{x}) = x_2\} \\ \text{p.o.} \quad & 0 \le x_1 \le 1,\ 0 \le x_2 \le 1. \end{aligned}$$

Ovaj problem se rešava izrazom

In[5]:= `MultiDist[{x1,x2},{x1>=0,x2>=0,x1<=1,x2<=1},{},{x1,x2}]`

Pozivom funkcije *ideal*, dobija se idealna tačka $\{2, 3\}$. Posle toga, dobija se sledeći

$$\begin{aligned} \min \quad & \Phi(\mathbf{x}) = |2 - x_1| + |3 - x_2| \\ \text{p.o.} \quad & x_1 \le 1,\ x_2 \le 1\ x_1 \ge 0,\ x_2 \ge 0. \end{aligned}$$

Posle zamena $y_1 = 2 - x1 = \$[1]^- - \$[1]^+$, $y_2 = 3 - x2 = \$[2]^- - \$[2]^+$, koristeći $|y_1| = \$[1]^+ + \$[1]^-$, $|y_2| = \$[2]^+ + \$[2]^-$, s obzirom na (2.7), polazni problem se transformiše u sledeći

$$\begin{aligned} \min \quad & \$[1]^+ + \$[1]^- + \$[2]^+ + \$[2]^- \\ \text{p.o.} \quad & x_1 - \$[1]^+ + \$[1]^- = 2,\ x_2 - \$[2]^+ + \$[2]^- = 3, \\ & x_1 \le 1,\ x_2 \le 1, \\ & x_1 \ge 0,\ x_2 \ge 0,\ \$[1]^+ \ge 0,\ \$[1]^- \ge 0,\ \$[2]^+ \ge 0,\ \$[2]^- \ge 0. \end{aligned}$$

Unutrašnja reprezentacija ovog problema je

$\$[1] + \$[2] + \$[3] + \$[4]$,
$\{x1 >= 0,\ x2 >= 0,\ x1 <= 1,\ x2 <= 1,$
$x1 - \$[1] + \$[2] == 2,\ x2 - \$[3] + \$[4] == 3,$
$\$[1] >= 0, \$[2] >= 0, \$[3] >= 0, \$[4] >= 0\}$,
$\{x1, x2, \$[1], \$[2], \$[3], \$[4]\}$

dok je njegovo optimalno rešenje

$\{3,\ \{x1 -> 1, x2 -> 1, \$[1] -> 0, \$[2] -> 1, \$[3] -> 0, \$[4] -> 2\}\}$

Prema tome, optimalno rešenje polaznog problema je

Out[5]= `{{1,1},{x1->1, x2->1, $[1]->0,$[2]->1,$[3]->0,$[4]->2}}`

3.9 Metod PROMETHEE

Metod **PROMETHEE** (Preference Ranking Organization METHods for Enrichment Evaluation) je metod višekriterijumske analize i služi za rangiranje konačnog broja alternativa. Postoje četiri varijante ovog metoda: PROMETHEE I, II, III i IV (poslednja predstavlja proširenje za neprekidne skupove). Opisaćemo metod II, koja daje potpuni poredak alternativa [21].

Cilj nam je da, od N tačaka $x^{(1)}$,..., $x^{(N)}$ u skupu dopustivih rešenja S izaberemo onu koja daje najbolju vrednost rešenja.

Uvedimo funkciju preferencije $P_i(x^{(1)}, x^{(2)})$ za alternative $x^{(1)}$ i $x^{(2)}$ u odnosu na kriterijum Q_i

$$P_i(x^{(1)}, x^{(2)}) = \begin{cases} 0, & \text{ako je } Q_i(x^{(1)}) \geq Q_i(x^{(2)}) \\ P_i(f_i(x^{(2)}) - f_i(x^{(1)})), & \text{ako je } Q_i(x^{(1)}) < Q_i(x^{(2)}) \end{cases}$$

Uvodimo oznaku $d = Q_i(x^{(2)}) - Q_i(x^{(1)})$.
Predloženo je šest tipova funkcije preferencije:

1. Jednostavan kriterijum

$$P_i(x^{(1)}, x^{(2)}) = \begin{cases} 0, & \text{ako je } d \leq 0 \\ 1, & \text{ako je } d > 0 \end{cases}$$

2. Kvazi-kriterijum

$$P_i(x^{(1)}, x^{(2)}) = \begin{cases} 0, & \text{ako je } d \leq q \\ 1, & \text{ako je } d > q \end{cases}$$

3. Kriterijum sa linearnom preferencijom

$$P_i(x^{(1)}, x^{(2)}) = \begin{cases} 0, & \text{ako je } d \leq 0 \\ \frac{d}{p,} & \text{ako je } 0 < d \leq p \\ 1, & \text{ako je } d > p \end{cases}$$

4. Nivoski kriterijum

$$P_i(x^{(1)}, x^{(2)}) = \begin{cases} 0, & \text{ako je } d \leq q \\ \frac{1}{2,} & \text{ako je } q < d \leq p \\ 1, & \text{ako je } d > p \end{cases}$$

5. Kriterijum sa linearnom preferencijom i oblašću indiferentnosti

$$P_i(x^{(1)}, x^{(2)}) = \begin{cases} 0, & \text{ako je } d \leq q \\ \frac{d-q}{p-q,} & \text{ako je } q < d \leq p \\ 1, & \text{ako je } d > p \end{cases}$$

6. Gausov kriterijum

$$P_i(x^{(1)}, x^{(2)}) = \begin{cases} 0, & \text{ako je } d \leq 0 \\ 1 - e^{\frac{-d^2}{2\sigma^2}}, & \text{ako je } d > 0 \end{cases}$$

Pri tome paremetre p, q, σ treba zadati za svaku kriterijumsku funkciju.

Definišemo višekriterijumski indeks preferencije alternative $x^{(1)}$ nad $x^{(2)}$

$$\Pi(x^{(1)}, x^{(2)}) = \sum_{i=1}^{k} w_i P_i(x^{(1)}, x^{(2)})$$

gde je w_i težina i-tog kriterijuma. Podrazumeva se $\sum_{i=1}^{k} w_i = 1$.

Uvodimo zatim tokove preferencije:

$$\begin{aligned}
\phi_i^+(x^{(i)}) &= \sum_{m=1}^{N} \Pi(x^{(i)}, x^{(m)}) \quad \text{(pozitivni tok)} \\
\phi_i^-(x^{(i)}) &= \sum_{m=1}^{N} \Pi(x^{(m)}, x^{(i)}) \quad \text{(negativni tok)} \\
\phi_i(x^{(i)}) &= \phi_i^+(x^{(i)}) - \phi_i^-(x^{(i)}) \quad \text{(neto tok).}
\end{aligned}$$

Neto tok je funkcija pomoću koje rangiramo alternative. Alternativa $x^{(i)}$ je višekriterijumski bolja od $x^{(j)}$ ako je $\phi_i(x^{(i)}) > \phi_j(x^{(j)})$.

Time smo dobili potpuno uređenje skupa alternativa na osnovu koga možemo izabrati najbolju.

3.10 Metod ELECTRE

Metod ELECTRE I (ELimination and ET Choice Translating REality) prvi put je objavio Roy sa svojim saradnicima (1971.). Za određivanje delimičnih poredaka alternativa najčešće se koristi metod ELECTRE I, a za potpuno uređenje skupa alternativa metod ELECTRE II. Ovi metodi omogućavaju parcijalno uređenje skupa rešenja na osnovu preferencija donosioca odluke, a pogodne su za diskretne probleme i raznorodne kriterijumske funkcije. Modeli dozvoljavaju uključivanje subjektivnih procena, bilo kroz vrednosti kriterijumskih funkcija, bilo kroz relativne važnosti pojedinih kriterijuma. Metodi ELECTRE III i IV su metodi "višeg" ranga.

Metod **ELECTRE** je metod višekriterijumske analize pomoću koga se može dobiti delimično uređenje skupa alternativa. Na osnovu dobijene relacije delimičnog uređenja ρ konstruiše se graf G u kome čvorovi predstavljaju alternative. Grana grafa se usmerava od čvora $x^{(i)}$ prema čvoru $x^{(j)}$ ako je $x^{(i)} \rho x^{(j)}$. Na osnovu tako konstruisanog grafa dobija se parcijalna rang lista (nepotpuno uređen skup) alternativa, a može se desiti da neke alternative ostanu izolovane.

Dakle, metod ELECTRE je razvijen za analizu odnosa među alternativama, a ne za potpuno uređenje skupa alternativa.

Opisaćemo ukratko noviju varijantu ELECTRE II [21].

Označimo sa $I = \{1, \ldots, l\}$ skup indeksa kriterijumskih funkcija. Za svaki par rešenja $(x^{(1)}, x^{(2)})$ definišemo

$$I^+(x^{(1)}, x^{(2)}) := \{i \in I \mid Q_i(x^{(1)}) < Q_i(x^{(2)})\}$$

$$I^{=}(x^{(1)}, x^{(2)}) := \{i \in I \mid Q_i(x^{(1)}) = Q_i(x^{(2)})\}$$
$$I^{-}(x^{(1)}, x^{(2)}) := \{i \in I \mid Q_i(x^{(1)}) > Q_i(x^{(2)})\}$$

Donosilac odluke zadaje težine kriterijuma w_i, $i = 1 \ldots, k$. Zatim se određuje

$$\begin{aligned} W^{+}(x^{(1)}, x^{(2)}) &= \sum_{i \in I^{+}} w_i \\ W^{=}(x^{(1)}, x^{(2)}) &= \sum_{i \in I^{=}} w_i \\ W^{-}(x^{(1)}, x^{(2)}) &= \sum_{i \in I^{-}} w_i \\ W &= W^{+} + W^{=} + W^{-}. \end{aligned}$$

Definišemo indeks saglasnosti

$$C(x^{(1)}, x^{(2)}) = \frac{W^{+} + W^{=}}{W}$$

Uslov saglasnosti za par $(x^{(1)}, x^{(2)})$ je ispunjen ako važi

$$C(x^{(1)}, x^{(2)}) \geq q \;\text{ i }\; \frac{W^{+}}{W^{-}} > 1$$

gde je q parametar čijim variranjem ćemo dobijati različite rezultate.

Da bismo definisali indeks nesaglasnosti moramo uvesti intervalnu skalu S koja će omogućiti poređenje funkcija raznorodnih vrednosti. Tako se dobijaju funkcije s_i koje predstavljaju surogat kriterijumskih funkcija Q_i.

Indeks nesaglasnosti je

$$d(x^{(1)}, x^{(2)}) = \frac{1}{S} \max_{i \in I^{-}} |s_i(x^{(1)}) - s_i(x^{(2)})|$$

Uslov nesaglasnosti za par $(x^{(1)}, x^{(2)})$ je ispunjen ako važi

$$d(x^{(1)}, x^{(2)}) \leq r$$

gde variranjem parametra r dobijamo različite rezultate.

Alternativa $x^{(j)}$ je superiorna nad $x^{(k)}$ ako su ispunjeni odgovarajući uslovi saglasnosti i nesaglasnosti. U tom slučaju grana $(x^{(j)}, x^{(k)})$ ulazi u rezultantni graf.

Povećanjem vrednosti parametra q i smanjivanjem vrednosti parametra r smanjuje se broj grana rezultantnog grafa. U praksi je dobro ispitati vrednosti $0.5 \leq q \leq 1$ i $0 \leq r \leq 0.5$.

Iz rezultantnog grafa možemo identifikovati alternative nad kojima nema superiornih. Naravno, neke alternative mogu ostati izolovane. Treba imati na umu da se metodom ELECTRE pre mogu dobiti korisne informacije o alternativama nego što se može potpuno rešiti problem multikriterijumske analize.

4 Interaktivni metodi

U interaktivnim metodima DO aktivno učestvuje tokom rešavanja problema. Najpre DO daje preliminarne informacije o svojim preferencijama na osnovu kojih analitičar generiše neki skup rešenja ili neki skup novih korisnih informacija. Kada dobije ove podatke DO obezbeđuje nove informacije o svojim zahtevima i postupak se iterativno ponavlja sve dok DO ne bude konačno zadovoljan dobijenim rešenjem. Prednost ovakvog pristupa reăvanju VKO je taj što se generiše samo deo Pareto optimalnog skupa i što DO iskazuje i menja svoje odluke i preferencije tokom samog procesa rešavanja problema.

Postoji mnogo interaktivnih metoda a mi ćemo navesti ukratko jednu od njih.

4.1 Metod referentne tačke

U raznim metodima VKO se koristi takozvana *funkcija ostvarenja* (achievement function). Uočimo referentnu tačku $\overline{z} \in \mathbf{R}^k$ čije su koordinate željene vrednosti kriterijumskih finkcja. Tačka $\overline{z}$ može biti ili ne biti dostižna. Funkcija ostvarenja je funkcija $s_{\overline{z}}$ koja zavisi od $\overline{z}$,

$$s_{\overline{z}} : Z \to \mathbf{R}$$

Posmatrajmo problem

$$\text{Minimizirati } s_{\overline{z}}(Q(\mathbf{x})) \text{ pod uslovom } \mathbf{x} \in S \tag{4.1.1}$$

Teorema 4.1.1 *Ako je funkcija ostvarenja $s_{\overline{z}} : Z \to \mathbf{R}$ strogo rastuća, tada je rešenje problema* (4.1.1) *slabo Pareto optimalno rešenje početnog problema VKO. Ako je izvršna funkcija $s_{\overline{z}} : Z \to \mathbf{R}$ jako rastuća, tada je rešenje problema* (4.1.1) *Pareto optimalno rešenje početnog problema VKO.*

Dokaz. Pretpostavimo suprotno, da je $\mathbf{x}^* \in S$ rešenje Problema (4.1.1), a da ono nije slabo Pareto optimalno. Sledi da postoji neko $\mathbf{x} \in S$ za koje važi da je $Q_i(\mathbf{x}) < Q_i(\mathbf{x}^*)$ za svako $i = 1, \ldots, k$. Kako je $s_{\overline{z}}$ strogo rastuća funkcija, dobijamo $s_{\overline{z}}(Q(\mathbf{x})) < s_{\overline{z}}(Q(\mathbf{x}^*))$, odakle dobijamo kontradikciju sa pretpostavkom da je $\mathbf{x}^*$ rešenje problema (4.1.1).

Drugi deo teoreme se dokazuje slično. Pretpostavimo da je $\mathbf{x}^*$ rešenje problema (4.1.1), a da ono nije Pareto optimalno. Dakle, postoji $\mathbf{x} \in S$ za koje važi $Q_i(\mathbf{x}) \leq Q_i(\mathbf{x}^*)$ za svako $i = 1, \ldots, k$, pri čemu za neko $j \leq k$ važi $Q_j(\mathbf{x}) < Q_j(\mathbf{x}^*)$. Kako je $s_{\overline{z}}$ jako rastuća funkcija dobijamo $s_{\overline{z}}(f(\mathbf{x})) < s_{\overline{z}}(Q(\mathbf{x}^*))$, odakle sledi da $\mathbf{x}^*$ nije rešenje problema (4.1.1), što je kontradikcija. □

U metodu referentne tačke generisanje Pareto optimalnog skupa je bazirano na referentnoj tački, a ne na vrednosnoj funkciji ili težinskim koeficijentima. Najpre se donosiocu odluke, ako je moguće, dostave neke informacije o problemu (idealna tačka, maksimalne vrednosti kriterijumskih funkcija u odnosu na dopustivi skup i slično). Takođe, potrebno je odrediti odgovarajuću funkciju ostvarenja. Na osnovu prethodne teoreme se može zaključiti da je poželjno da izvršna funkcija bude jako rastuća. Metod se sastoji iz sledećih koraka [15]:

1. Donosiocu odluka se predstave informacije o problemu. Stavi se $h = 1$.

2. DO navodi referentnu tačku $\overline{z}^h$ (koordinate ove tačke su željeni nivoi kriterijumskih funkcija).

3. Minimizira se funkcija ostvarenja pod uslovom $\mathbf{x} \in S$ i dobije se Pareto optimalno rešenje $\mathbf{x}^h$ i odgovarajuće z^h. Rešenje z^h se predstavi DO.

4. Minimizira se k funkcija ostvarenja $s_{\overline{z}(i)}$ sa referentnim tačkama $\overline{z}(i) = \overline{z}^h + ||\overline{z}^h - z^h||e^i$, gde je e^i i-ti jedinični vektor, $i = 1, \ldots, k$. Na ovaj način se dobije k novih Pareto optimalnih rešenja. Dakle, ukupno imamo $k + 1$ dominantnih rešenja, tj. $k + 1$ odgovarajućih tačaka u kriterijumskom skupu Z.

5. Ako među ovih $k + 1$ tačaka DO izabere neku kao zadovoljavajuću, onda je odgovarajuće x^h konačno rešenje. U suprotnom, DO bira (među dobijenih $k + 1$ tačaka skupa Z) novu referentnu tačku $\overline{z}^{h+1}$. Postavljamo $h = h + 1$ i prelazimo na korak 3.

Prednost ovog metoda je što DO neposredno upravlja procesom rešavanja problema i što je u mogućnosti da menja svoje mišljenje tokom procesa rešavanja. Nedostatak je što to može dugo da traje i što DO ne može biti siguran da je u koraku 5 koraku dobro izabrao novu referentnu tačku.

Literatura

[1] H.P. Benson, *Existence of efficient solution for vector maximization problems*, Journal of Optimization Theory and Applications, **28** (1978) 569-580.

[2] D.G. Carmichael, *Computation of Pareto optima in structural design*, Int. J. Numer. Methods Eng. 15, (1980) 925-952.

[3] V. Chankong, Y. Haimes, *Multiobjective decision making: Theory and methodology Series, Volume 8*, North-Holland, New York, Amsterdam, Oxford, 1983.

[4] A. Charnes, W.W. Cooper, *Management Models and Industrial Applications of Linear Programming*, New York: John Wiley and Sons, 1961.

[5] A. Charnes, W.W. Cooper, *Goal programming and multiple objective optimization; part 1*, Eur. J. Oper. Res. 1, (1977) 39-54.

[6] C.A. Coello, *A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques*,

Internet lokacija http://www.lania.mx/∼ccoello/EMOO/informationfinal.ps.gz

[7] M. Ćupić, R.V.M. Tummala, M. Suknović M., *Odlučivanje: Formalni pristup*, FON, Beograd, 2001.

[8] I. Das, J.E Dennis, *A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems*, Struct. Optim. **14** (1997) 63-69.

[9] Y.Y. Haimes, L.S. Lasdon, D.A. Wismer, *On a bicriterion formulation of the problems of integrated system identification and system optimization*, IEEE Trans. Syst. Man Cybern. SMC-1, (1971) 296-297.

[10] B.F. Hobbs, *A comparison of weighting methods in power plant siting.*, Decis. Sci., **11** (1980) 725-737.

[11] C.L. Hwang, K. Yoon, *Multiple attribute decision making methods and applications: a state-of-the-art survey.*, In: Beckmann, M.; Kunzi, H.P. (eds.) Lecture Notes in Economics and Mathematical Systems, No. 186. Berlin: Springer-Verlag, 1981.

[12] C.L. Hwang, Md. Masud, A.S., in collaboration with Paidy, S.R. and Yoon, K. *Multiple objective decision making, methods and applications: a state-of-the-art survey* In: Beckmann, M.; Kunzi, H.P. (eds.) Lecture Notes in Economics and Mathematical Systems, No. 164. Berlin: Springer-Verlag, 1979.

[13] Y. Ijiri, *Management Goals and Accounting for Control*, Amsterdam: North-Holland, 1965.

[14] J.P. Ignizio, *Linear programming in single-multiple-objective systems*, Englewood Cliffs: Prentice Hall, 1982.

[15] K. Miettien *Nonlinear Multiobjective Optimization*, Kluver Academic Publishers, Boston, London, Dordrecht, 1999.

[16] K.R. Mac-Crimmon, *An overview of multiple objective decision making, in J.L. Cochrane and M. Zeleny (Eds), Multiple criteria decision making*, University of South Carolina Press, Columbia, 1973.

[17] R. Maeder, *Programming in Mathematica, Third Edition* Redwood City, California: Adisson-Wesley, 1996.

[18] R.T. Marler, *Survey of multi-objective optimization methods for engineering* Struct. Multidisc. Optim. **26** (2004) 369–395.

[19] K. Miettinen, L. Kirilov, *Interactive reference direction approach using implicti parametrization for nonlinear multiobjective optimization*, MCDM 2004, Whistler, B. C. Canada August 6-11, 2004.

[20] I. Nikolić, S. Borović., *Višekriterijumska optimizacija: metode, primena u logistici*, Centar vojnih škola Vojske Jugoslavije, Beograd, 1996.

[21] S. Opricović, *Optimizacija sistema*, Građevinski fakultet Univerziteta u Beogradu, 1992.

[22] A. Osyczka, *Multicriterion Optimization in Engineering with Fortran Programs*, New York: John Wiley and Sons, 1984.

[23] K. Schittkowski, *Multicriteria Optimization -User's Guide-*, http://www.klaus-schittkowski.de, November 2004.

[24] W. Stadler, *Fundamentals of multicriteria optimization*, In: Stadler,W. (ed.) Multicriteria Optimization in Engineering and in the Sciences, pp. 125. New York: Plenum Press, 1988.

[25] P.S. Stanimirović and I.P. Stanimirović, *Implementation of polynomial multi-objective optimization in MATHEMATICA*, Structural Multidisciplinary Optimization, DOI 10.1007/s00158-007-0180-9.

[26] P.S. Stanimirović, G.V. Milovanović, *Programski paket MATHEMATICA i primene*, Elektronski fakultet u Nišu, Edicija monografije, Niš, 2002.

[27] P.S. Stanimirović, N.V. Stojković, M.D. Petković, *Run-time transformations in implementation of linear multi-objective optimization*, XVI Conference on Applied Mathematics, N. Krejic, Z. Luzanin, eds. Department of Mathematics and Informatics, Novi Sad, (2004),157–165.

[28] H. Voogd, *Multicriteria Evaluation for Urban and Regional Planning*, London: Pion, 1983.

[29] M. Vujošević , M. Stanojević , N. Mladenović , *Metode optimizacije*, Društvo operacionih istraživača Jugoslavije, Beograd, 1996.

Internet lokacija http://www.laboi.fon.bg.ac.yu/download/MetOpt/uvoduvko.pdf

[30] F.M. Waltz *An engineering approach: hierarchical optimization criteria*, IEEE Trans. Autom. Control AC-12, (1967) 179-180.

[31] R.E. Wendell, D.N. Lee, *Efficiency in multiple objective optimization problems*, Mathematical Programming, **12** (1977) 406-414.

[32] S. Wolfram, *The Mathematica Book, 4th ed.*, Wolfram Media/Cambridge University Press, 1999.

[33] L.A. Zadeh, *Optimality and non-scalar-valued performance criteria*, IEEE Trans. Autom. Control AC-8, (1963) 5960.

[34] S. Zionts, *Multiple criteria mathematical programming: an updated overview and several approaches.*, In: Mitra, G. (ed.) Mathematical Models for Decision Support, 1988, pp. 135167, Berlin: Springer-Verlag

[35] K. Zotos, *Performance comparison of Maple and Mathematica*, Appl. Math. Comput. (2006), doi:10.1016/j.amc.2006.11.008.

Index