Univerzitet u Nišu
Prirodno-matematički fakultet

MODIFIKACIJE METODA MATEMATIČKOG PROGRAMIRANJA I PRIMENE

Diplomski rad

Mentor:
Dr Predrag S. Stanimirović,
redovni profesor

Kandidat:
Marko D. Petković
br. indeksa: 2642

Niš, Avgust 2006

U svakodnevnom životu, kao i u mnogim naučnim oblastima često se nameće potreba nalaženja optimalne (maksimalne ili minimalne) vrednosti neke veličine pri čemu su zadovoljeni odredjeni uslovi. Ovi problemi su bili osnova za nastanak matematičke oblasti Operaciona istraživanja. Naravno, kako se uvek teži uprošćavanju problema, veličina koju treba optimizovati predstavlja se kao linearna funkcija nekih veličina (promenljivih), a takodje se predpostavlja da su i ograničenja koja moraju biti zadovoljena linearne funkcije. Tako je nastao problem Linearnog programiranja.

Takodje, često nam je potrebno da maksimizujemo (minimizujemo) više veličina. Na taj način dobijamo problem višekriterijumske optimizacije.

Ovim problemima su se bavili mnogi naučnici tako da je konstruisan veliki broj algoritama različitih performansi.

Ovaj Diplomski rad je rezultat mog šestogodišnjeg bavljenja problemima matematičkog programiranja, što je dovelo do objavljivanja više radova kako na domaćim i medjunarodnim konferencijama tako i u vodećim časopisima iz ove oblasti. Kao rezultat mog bavljenja ovom problematikom nastalo je nekoliko programa za rešavanje problema linearnog programiranja od kojih je najznačajniji program MarPlex *koji je u jednom delu konkurentan vodećim svetskim programima iz ove oblasti.*

Sa posebnim zadovoljstvom zahvaljujem se mom mentoru, Prof. Dr Predragu Stanimiroviću, ne samo na pomoći pri izradi ovog rada, već i na velikoj pažnji i vremenu koje mi je posvetio uvodeći me u probleme matematičkog programiranja, a time i u naučni rad.

Zahvaljujem se i Dr. Nebojši Stojkoviću na nesebičnoj pomoći kako pri izradi ovog rada, tako i uopšte. Zahvalnost takodje dugujem i profesoru Dr. Predragu Rajkoviću koji mi je takodje pružio nesebičnu pomoć i uveo me u jednu drugu oblast matematike (teoriju polinoma i polinomnih identiteta), što je takodje dovelo do publikovanja više radova u časopisima i na konferencijama.

Najtoplije se zahvaljujem članovima moje porodice, koji su u granicama svojih mogućnosti takodje doprineli izradi ovog rada.

Sadržaj

1. Uvod

Matematičko programiranje je deo široke naučne oblasti koja se poslednjih decenija razvija veoma brzo i koja je poznata pod nazivom "Operaciona istraživanja". Operaciona istraživanja su se pojavila uoči drugog svetskog rata. Prvi rad, kako iz matematičkog programiranja, tako i iz operacionih istraživanja [18] objavio je ruski matematičar Kantorovič. U tom radu on definiše transportni zadatak. Transportni zadatak se smatra i prvim problemom matematičkog programiranja. Američki matematičar F. Hičkok je, 1941. godine, strogo formulisao ovaj problem i dao metod za njegovo rešavanje [14]. Često se transportni zadatak naziva i zadatkom Hičkoka.

Interesantno je da je još 1940. godine, posle Kantoroviča ali pre Hičkoka, kod nas objavljen rad "Pravila za proračun potrebnog broja transportnih sredstava" [17] pukovnika Vlastimira Ivanovića. U ovom radu, pukovnik Ivanović je pokazao kako se izračunava minimalan broj vozila za prevoz date količine materijala korišćenjem "Principa najveće ekonomije". Pukovnik Ivanović je bio prvi kod nas i medju prvima u svetu koji se bavio matematičkim programiranjem.

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 rednika, kapacitet mašina, radno vreme). Naučnik pravi matematički model fizičkog procesa koji najbolje opisuje odredjenu fizičku pojavu, a na raspolaganju ima konačni broj mernih rezultata. Takodje model ne sme biti suviše komplikovan.

U svim ovim situacijama možemo da indetifikujemo tri zajedničke stvari [2]:

1. Postoji globalna veličina (cilj) koja se želi optimizovati (profit, razlika izmedju predvidjanja 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 odredjene 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.

U ovom radu detaljno ćemo proučiti dve klase problema matematičkog programiranja: ***linearno programiranje*** i ***višekriterijumsku optimizaciju***. Upravo ove dve klase problema su najzastupljenije u praktičnim zadacima. Linearno programiranje se zbog dobro razradjenih metoda za rešavanje često koristi i kada je posmatrani model po svojoj prirodi nelinearan. Takodje, često je potrebno optimizovati više od jedne veličine (na primer, hoćemo da maksimizujemo profit od prodaje automobila koji razvijamo, a takodje i da minimizujemo verovatnoću kvara, udesa, itd...).

Linearno programiranje je jedan od najefikasnijih pristupa formulisanju i rešavanju složenih problema donošenja odluka i kao takvo predstavlja osnovnu disciplinu matematičkog programiranja [5, 15]. Linearno programoranje se javlja pri rešavanju praktičnih zadataka kao što su odredjeni transportni zadaci, planiranje ekonomskog razvoja, kao i u drugim oblastima planiranja.

Zadatak linearnog programiranja je da odredi maksimum (minimum) linearne funkcije koja zavisi od više promenljivih pod uslovom da su neke od ovih promenljivih nenegativne i da zadovoljavaju linearna ograničenja u obliku jednačina i/ili nejednačina. Linearna funkcija koja se optimizira naziva se ***ciljna funkcija*** ili ***funkcija cilja***.

Postoji više metoda za rešavanje problema linearnog programiranja [50, 2, 5, 15, 44]. Geometrijski metod je najjednostavniji medju njima, medjutim njegova mana 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 G. B. Dancig 1947. godine [9] . On je takodje formulisao opšti oblik problema linearnog programiranja i dao algoritam za njegovo rešavanje, poznat kao ***simpleks metod*** [5, 15, 50]. Dancigov rad je više godina kružio medju stručnjacima i poslužio kao osnova svim narednim razmatranjima problema linearnog programiranja.

Za linearni problem koji ima ograničenu oblast dopustivih rešenja, dopustiva oblast je definisana sa m linearno nezavisnih ograničenja. Ovaj sistem ograničenja je ekvivalentan sistemu od m linearnih jednačina sa n promenljivih. Suština je da se izmedju $\binom{n}{m}$ bazičnih rešenja pronadje ono koje maksimizira ili minimizira linearnu ciljnu funkciju.

Geometrijskom metodom se proveravaju vrednosti funkcije cilja u svakoj rubnoj tački oblasti dopustivih rešenja. Kako ovaj metod radi samo za navedene specijalne slučajeve, rubnih tačaka je malo, pa i sam postupak traje kratko. Ali ako posmatramo problem gde ima više promenljivih i uslova, broj rubova veoma brzo raste, a u najgorem slučaju je jednak $\binom{n}{m}$, gde je m broj ograničenja a n broj promenljivih. Ispitivanje ovolikog broja potencijalnih ekstremuma funkcije cilja trajalo bi veoma dugo. Dancigovim metodom se ovaj broj drastično redukuje [10], čime je on našao velike primene u praksi. Medjutim, V. Kli i G.L. Minti su 1972. godine [22] pokazali da simpleks metod nije polinomijalan.

Hačijan je 1979. godine otkrio prvi polinomijalni algoritam za rešavanje problema linearnog programiranja [21]. On je dokazao da njegov metod elipse radi u vremenu $\mathcal{O}(n^4L)$ gde je L broj bitova potrebnih za zapis svih parametara problema. Ovaj metod se u praksi pokazao mnogo sporijim od simpleks metoda jer je skoro uvek dostizao maksimalnu složenost.

Karmarkar je 1984. godine došao do polinomijalnog algoritma koji rešava problem linearnog programiranja i koji je praktično primenljiv [19]. Posle Karmarkarovog metoda po-

javila se čitava familija polinomijalnih algoritama poznatih pod nazivom ***interior point methods*** (metodi unutrašnje tačke). Danas su ***primal-dual*** metodi unutrašnje tačke dominantni za rešavanje problema linearnog programiranja [8]. 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 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.

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

HOPDM je napisan u programskom jeziku FORTRAN [13].

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 [54].

LOQO je napisan u programskom jeziku C, a opisan je u [48].

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 [8].

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.

MarPlex je naš program. Napisan je u programskom jeziku Visual Basic 6.0 i koristi simpleks metod, odnosno modifikacije prezentovane u trećoj glavi ovog rada. 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 takodje naš program i predstavlja revidiranu verziju programa MarPlex napisanu u programskom jeziku MATHEMATICA.

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 [2, 27, 15]. Zajedničko svim tim metodima je da se polazni problem na odgovarajući način svodi na problem linearnog i nelinearnog programiranja.

U drugoj glavi ovog rada detaljno 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 prvi i najjednostavniji metod za rešavanje problema linearnog programiranja i pokazati kako se on može implementirati. Rezultati vezani za implementaciju ovog metoda su originalni i preuzeti

su iz našeg rada [47]. Dalje ćemo proučiti simpleks metod za rešavanje problema linearnog programiranja u opštem obliku i detaljno ćemo opisati sve njegove faze. Takodje ćemo razmotriti još jednu verziju simpleks metoda, revidirani simpleks metod, kojom se rešava problem numeričke stabilnosti klasičnog simpleks metoda. Na kraju ove glave, ukazaćemo na problem cikliranja kao i na dva načina da se taj problem prevazidje. Takodje ćemo pokazati da je simpleks algoritam, i pored svojih odličnih osobina na praktičnim problemima, eksponencijalne složenosti.

Sledeća glava predstavlja naše originalne rezultate preuzete iz radova [41, 34, 40], i bavi se modifikacijama i poboljšanjima pojedinih faza simpleks metoda. Na kraju ove glave detaljno ćemo opisati implementaciju simpleks i revidiranog simpleks metoda, kao i rezultate testiranja naša dva programa MarPlex i RevMarPlex. Pokazaćemo da su prikazane modifikacije i u praksi bolje od klasičnih algoritama.

U čevrtoj glavi izložićemo problem višekriterijumske optimizacije, kao i metode za njegovo 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. Rad svake od metoda pokazaćemo na jednom ili više primera. Razmatranja vezana za implementaciju metoda su originalna i preuzeta su iz našeg rada [43]. Na kraju ove glave, posebno ćemo razmotriti problem linearne višekriterijumske optimizacije. Ukazaćemo na primenu simpleks algoritma i teorije generalisanih inverza prilikom rešavanja, kako opšteg, tako i specijalnih problema linearne višekriterijumske optimizacije. Takodje ćemo ukazati na moguću primenu naših rezultata iz teorije generalisanih inverza [35, 36, 42, 37].

U pretposlednjoj glavi razmotrićemo dve praktične primene metoda matematičkog programiranja. Prva se odnosi na projektovanje teleskopa (problem optike), a druga na projektovanje digitalnih FIR filtara koji imaju veliku ulogu u obradi digitalnih signala (problem telekomunikacija).

U poslednjoj glavi sumiraćemo sve prikazane rezultate i razmotrićemo pravce u kojima se mogu kretati dalja istraživanja iz ove oblasti.

U dodatku ćemo dati kod najvažnijih procedura programa MarPlex, RevMarPlex i GEOM uz detaljno objašnjenje implementacionih detalja.

2. Problem linearnog programiranja

U ovoj glavi ćemo detaljno proučiti problem linearnog programiranja kao i metode za njegovo rešavanje. Najpre ćemo dati osnovne definicije pojmova i osnovne teoreme vezane za skup dopustivih rešenja. Nakon toga ćemo opisati i na primerima ilustrovati dva metoda za rešavanje problema linearnog programiranja: geometrijski i simpleks metod.

2.1 Definicija i oblici problema linearnog programiranja

Problem linearnog programiranja se zadaje na sledeći način [15, 44]: Potrebno je odrediti vrednosti promenljivih $x_1, \dots x_n \in \mathbb{R}$ za koje je ispunjen sistem linearnih jednačina i nejednačina:

$$\begin{aligned} N_i^{(1)}: &\quad \sum_{j=1}^{n} a_{ij}x_j \le b_i, \quad i = 1, \dots, p \\ N_i^{(2)}: &\quad \sum_{j=1}^{n} a_{ij}x_j \ge b_i, \quad i = p+1, \dots, q \\ J_i: &\quad \sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i = q+1, \dots, m \\ &\quad x_j \ge 0, \quad j \in \mathcal{J} = \{1, \dots, s\}, \quad s \le n \end{aligned} \tag{2.1.1}$$

tako da linearna ciljna funkcija

$$f(x) = f(x_1, \dots, x_n) = c_1x_1 + \dots + c_nx_n \tag{2.1.2}$$

ima ekstremum, tj. minimum ili maksimum. Pri tome su a_{ij}, b_i, c_j poznati realni brojevi, $i = 1, \dots, m$, $j = 1, \dots, n$.

Promenljive $x_{s+1}, \dots x_n$ za koje nije ispunjen uslov nenegativnosti u (2.1.1) nazivamo *slobodne promenljive*.

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

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

Dalje, $y \ge 0$ znači $y_1 \ge 0, \dots, y_k \ge 0$.

Rešenja $x = (x_1, \ldots, x_n)$ sistema (2.1.1) nazvaćemo *dopustivim rešenjima* a njihov skup označićemo sa Ω_P. U geometrijskoj interpretaciji, svako rešenje $x \in \Omega_P$ predstavlja tačku n-dimenzionalnog prostora $\mathbb{R}^n$.

Za skup Ω_P pretpostavljamo da nije prazan i da sadrži najmanje jedan element. Optimalno rešenje $x^* = (x_1^*, \ldots, x_n^*) \in \Omega_P$ zadatka linearnog programiranja je ono dopustivo rešenje za koje funkcija cilja (2.1.2) dostiže maksimum (minimum). U slučaju maksimizacije funkcije cilja ispunjen je uslov

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

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

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

Problem minimuma se može transformisati u problem maksimuma (i obratno) jednostavnim množenjem ciljne funkcije sa -1, koristeći $\min(f(x)) = -\max(-f(x))$.

Problem linearnog programiranja ima rešenje ako veličina $f(x^*)$ ima konačnu vrednost na skupu Ω_P dopustivih rešenja.

Neka je $A = [a_{ij}]_{m \times n} = [V_1^T \; V_2^T \; \ldots \; V_m^T]^T$ data matrica sa vrstama $V_1, \ldots, V_m \in \mathbb{R}^n$, neka su $b \in \mathbb{R}^m$ i $c \in \mathbb{R}^n$ dati vektori i neka je $x \in \mathbb{R}^n$ nepoznat vektor. U matričnom obliku problem (2.1.1)-(2.1.2) može se zapisati na sledeći način [7]:

$$\begin{aligned} &\min c^T x, \\ &\text{p.o. } V_i^T x \le b_i, \quad i \in I_1, \\ &\qquad V_i^T x = b_i, \quad i \in I_2, \\ &\qquad V_i^T x \ge b_i, \quad i \in I_3, \\ &\qquad x_j \ge 0, \qquad j \in \mathcal{J} = \{1, \ldots, s\}, \quad s \le n \end{aligned} \tag{2.1.3}$$

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 $s = n$ (nema slobodnih promenljivih), problem (2.1.3) se svodi na tzv. *standardni oblik* problema linearnog programiranja:

$$\begin{aligned} &\min c^T x, \\ &Ax = b, \\ &x \ge 0, \end{aligned} \tag{2.1.4}$$

dok se u slučaju $I_3 = \{1, \ldots, m\}$ i $s = n$ svodi na tzv. *simetrični oblik*:

$$\begin{aligned} &\min c^T x, \\ &Ax \ge b, \\ &x \ge 0. \end{aligned} \tag{2.1.5}$$

Simetrični oblik se često naziva i *kanonski oblik* [2, 50, 46, 45]. Dopustivo rešenje x^* je *optimalno* ako je:

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

za svako drugo dopustivo rešenje $x \in \Omega_P$.

Većina metoda za rešavanje problema linearnog programiranja koristi standardni ili simetrični oblik problema. Kao što ćemo kasnije videti, simpleks metod koristi simetrični, dok revidirani simpleks i primal-dual metodi standardni oblik. Ovi oblici se lako prevode jedan u drugi, a takodje i problem zadat u opštem obliku (2.1.1)-(2.1.2), odnosno (2.1.3) moguće je transformisati u standardni ili simetrični oblik.

Ako se uvedu nove nepoznate $x_{n+1}, \ldots, x_{n+q}$ i jednačine tipa $N^{(1)}$ i $N^{(2)}$ transformišu u ekvivalentne jednačine:

$$J^{(1)}: \quad \sum_{j=1}^{n} a_{ij}x_j + x_{n+i} = b_i, \quad i = 1, \ldots, p$$

$$J^{(2)}: \quad \sum_{j=1}^{n} a_{ij}x_j - x_{n+i} = b_i, \quad i = p+1, \ldots, q$$

Ciljna funkcija je sada

$$f'(x_1, \ldots, x_{n+q}) = c_1x_1 + \ldots + c_nx_n + c_{n+1}x_{n+1} + \ldots + c_{n+q}x_{n+q}$$

gde je $c_{n+1} = \ldots = c_{n+q} = 0$. Promenljive $x_{n+1}, \ldots, x_{n+q}$ nazivamo *dodatne* (ili *slack*) promenljive. Ostaje nam još samo da eliminišemo slobodne promenljive. Svaku slobodnu promenljivu $x_j, j \in \{s+1, \ldots, n\}$ možemo zameniti u ciljnoj funkciji i svim ograničenjima sa $x_j^+ - x_j^-$, pri čemu je $x_j^+ \geq 0, x_j^- \geq 0$,. Na taj način dobijamo ukupno $n' = n + q + (n - s) = 2n + q - s$ promenljivih, pri čemu sve moraju biti nenegativne i zadovoljavati m jednačina. Na ovaj način smo konstruisali problem u standardnom obliku ekvivalentan sa (2.1.1), odnosno (2.1.3).

Za svodjenje opšteg oblika (2.1.1) na simetričan oblik (2.1.5) dovoljno je iz jednačina J_i izraziti $m-q$ promenljivih $x_1, \ldots, x_n$ preko ostalih. Ako sad ove promenljive zamenimo u nejednačine iz (2.1.1) i odgovarajuće uslove nenegativnosti, dobijamo simetričan oblik sa q promenljivih i najviše m uslova. Slobodne promenljive se izražavaju na isti način kao i u prethodnom slučaju.

Napomena 2.1.1 *Razmotreno svodjenje na simetričan oblik važi uz pretpostavku da je sistem jednačina $\{J_i\}_{i=q+1,\ldots,n}$ konzistentan i potpunog ranga. U suprotnom prvo treba eliminisati suvišne jednačine (npr. Gauss-ovim metodom), odnosno formirati ekvivalentan sistem potpunog ranga.*

Na kraju ovog odeljka razmotrimo jedan primer problema linearnog programiranja

Primer 2.1.1 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, 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:

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

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 (2.1.5). Ako sada uvedemo slack promenljive x_3 i x_4 dobijamo ekvivalentan problem u standardnom obliku:

$$\begin{aligned} \max f(x) = 3500x_1 + 4800x_2 \\ 2x_1 + 4x_2 - 24 = -x_3 \\ 4x_1 + 2x_2 - 24 = -x_4 \\ x_1 \geq 0, x_2 \geq 0. \end{aligned}$$

odnosno u matričnoj formi (2.1.4):

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

2.2 Osobine skupa dopustivih rešenja

Neka je problem linearnog programiranja zadat u standardnom obliku (2.1.4). Skup $\Omega_P = \{x \mid Ax = b,\ x \geq 0\}$ na kome je definisana funkcija $f(x) = c^T x$ 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 $f(x)$ definisane na skupu Ω_P.

Teorema 2.2.1 *Skup $\Omega_P = \{x \mid Ax = b,\ x \geq 0\}$ 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 $x \in \Omega_P$. □

Svaka jednačina iz standardnog oblika predstavlja u prostoru $\mathbb{R}^n$ jednu hiperravan. Skup Ω_P je presek svih m hiperravni. Označimo sa $K_i, i = 1, \ldots, n$ kolone matrice A.

Sada ćemo definisati pojam bazičnog rešenja sistema jednačina u (2.1.4) koji je od fundamentalnog značaja u problemu linearnog programiranja.

Definicija 2.2.1 *Rešenje x, sistema $Ax = b$ je osnovno (bazično) ako su u jednačini*

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

vektori K_i za koje je $x_i \neq 0$ linearno nezavisni.

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

Definicija 2.2.2 *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.*

Ukoliko ne postoje degenerisana rešenja, tada svako bazično rešenje x jedinstveno odredjuje odgovarajuće linearno nezavisne kolone matrice A. Ovaj pojam je veoma važan za simpleks metod, tj. za pojavu cikliranja, koja će kasnije biti detaljno objašnjena.

Definicija 2.2.3 *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 B bazična matrica. Sada sistem iz (2.1.4) možemo napisati kao:

$$A_B x_B + A_N x_N = b \Longrightarrow x_B = A_B^{-1} b - A_B^{-1} A_N x_N \tag{2.2.6}$$

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 odredjuje odgovarajuću bazičnu matricu (ukoliko je nedegenerisano, onda je ta matrica jedinstvena a u suprotnom nije). Ovim smo opravdali naziv "bazična" u definiciji (2.2.3).

Definicija 2.2.4 *Tačka x je ekstremna tačka konveksnog skupa Ω_P ako za nju važi:*

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

Definicija 2.2.5 *Bazično rešenje x sistema $Ax = b$ za koje važi i uslov $x \geq 0$ je bazično dopustivo rešenje.*

Važna činjenica da svaki dopustivi problem linearnog programiranja (2.1.4) ima bazično dopustivo rešenje dokazuje sledeća teorema.

Teorema 2.2.2 *Ako je $\Omega_P = \{x \mid Ax = b,\ x \geq 0\} \neq \emptyset$, tada on sadrži bar jedno osnovno rešenje.*

Dokaz. Neka je $x = (x_1, \ldots, x_n) \in \Omega_P$ rešenje za koje je broj strogo pozitivnih koordinata p minimalan. 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. Pretpostavimo sada da 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

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

Izaberimo k tako da je

$$\frac{x_k}{\lambda_k} = \min \left\{ \frac{x_j}{\lambda_j} | \, \lambda_j > 0 \right\}$$

Tada je vektor

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

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 $x_1 \in \Omega_P$. Medjutim, 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. □

U nastavku ćemo pokazati da su bazična rešenja ključna u problemu linearnog programiranja, jer upravo u njima funkcija cilja dostiže ekstremnu vrednost. Sledeća teorema daje vezu izmedju ekstremnih tačaka konveksnog skupa Ω_P i bazičnih rešenja.

Teorema 2.2.3 *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 $x = x^1 = x^2$ tj. x je ekstremna tačka skupa Ω_P.

Dokažimo obratno. 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. □

Posledica 2.2.4 *Skup Ω_P ima konačno mnogo ekstremnih tačaka.*

Dokaz. Svaka bazična matrica B odredjuje tačno jedno bazično rešenje (na osnovu (2.2.6)), i kako se za svako bazično rešenje može naći odgovarajuća matrica B tako da važi (2.2.6). 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 $f(x)$ redukuje sa (u opštem slučaju) beskonačnog skupa Ω_P na konačan skup temena koji ima maksimalno $\binom{n}{m}$ elemenata. Ciljna funkcija $f(x)$ 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 $f(x)$ ekstremna. Ovaj broj potencijalnih ekstremuma veoma brzo raste sa m i n.

Na kraju ovog odeljka, dokažimo da se ekstemum funkcije cilja $f(x)$ (ako postoji) nalazi upravo u ekstremnim tačkama skupa Ω_P, a na osnovu teoreme 2.2.3 u bazično dopustivim rešenjima sistema $Ax = b$.

Teorema 2.2.5 *Neka je skup Ω_P ograničen. Tada postoji $\inf_{x \in \Omega_P} f(x)$, i on se dostiže u ekstremnoj tački x^* skupa Ω_P. Skup $\Omega_P^* = \{x | \, x \in \Omega_P, \, f(x) = f(x^*)\}$ 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

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

što dokazuje da z dostiže minimum u x^*.

Dokažimo sada da je skup Ω_P^* konveksan. Neka je su $x^1, x^2 \in \Omega_P^*$, odnosno $f(x^1) = f(x^2) = f(x^*)$. Tada je

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

za svako $0 \leq \lambda \leq 1$. □

Prema tome, ako funkcija cilja ima konačnu ekstremnu vrednost na Ω_P, onda se ona dostiže u ekstremnoj tački, odnosno bazično dopustivom rešenju. U suprotnom, skup Ω_P je neograničen, kao ni funkcija cilja. Sledeća teorema, koju nećemo dokazivati, daje potreban uslov da funkcija $f(x)$ nema ekstemum. Dokaz teoreme sledi iz samog simpleks metoda.

Teorema 2.2.6 *Postoji bazična matrica $A_B = [K_{i_1} \cdots K_{i_m}]$ i kolona K_p matrice A tako da je $A_B^{-1} K_p \leq 0$, akko je Ω_P neograničen skup.*

Ako je skup Ω_P ograničen tada je svako $x \in \Omega_P$ konveksna kombinacija osnovnih (bazičnih) rešenja. Znači, dovoljno je znati samo bazična rešenja. Upravo na ovoj činjenici se zasnivaju i geometrijski i simpleks metod. Kod oba metoda, polazeći od početnog bazičnog rešenja, konstruisemo niz novih rešenja, tako da funkcija cilja monotono raste (opada). Posle konačnog broja koraka ili dolazimo do optimalne tačke ili do zaključka da je funkcija cilja neograničena, a samim tim i skup Ω_P.

2.3 Geometrijski metod

Geometrijski metod se može iskoristiti kod problema koji sadrže $n = 2$ ili $n = 3$ promenljive. Problem linearnog programiranja dat u standardnom obliku (2.1.4) koji ispunjava uslov $n - m = 2$ (ili $n - m = 3$) takodje se može rešavati geometrijskim metodom. Geometrijski metod, iako je primenljiv samo u specijalnim slučajevima, koristi se jer nam olakšava pristup opštoj algebarskoj metodi. Takodje, geometrijski metod ima velike primene u edukativne svrhe [47].

Neka je dat linearni problem u simetričnom obliku:

$$\begin{aligned}
&\max f(x) = c_1 x_1 + \ldots + c_n x_n \\
&a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \leq b_1 \\
&a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n \leq b_2 \\
&\cdots\cdots\cdots \\
&a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n \leq b_m.
\end{aligned}$$

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

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

odredjuje 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 odredjena jednakošću:

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

gde se podskupovi $D_{m+1}, ..., D_{m+n}$ odnose na uslove nenegativnosti promenjivih $x_1 \geq 0, \ldots, x_n \geq 0$. Prema tome, u n-dimenzionalnom prostoru $\mathbb{R}^n$, skup Ω_P predstavlja poliedar (simplicijalni kompleks). Skup tačaka u kojima funkcija cilja $f(x)$ ima vrednost d predstavlja takodje jednu hiperravan $\mathcal{H}_{f,d} = \{x \in \mathbb{R}^n \mid f(x) = d\}$, 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 $\mathcal{H}_{f,d} \cap \Omega_P \neq \emptyset$. 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 2.2.5, 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.

Kod slučaja $n = 2$, ako se ograničenja grafički predstave u koordinatnom sistemu $x_1 x_2$ dobija se konveksan poligon, na čijim se temenima (ekstremnim tačkama) nalaze moguća rešenja datog problema linearnog programiranja. Teme najudaljenije od prave odredjene funkcijom cilja predstavlja optimalno rešenje datog problema.

Geometrijski metod, za slučaj $n = 2$ smo implementirali u programskom jeziku MATHEMATICA [53]. Tako je nastao program GEOM [47], koji za unet problem linearnog programiranja u simetričnom obliku od dve promenljive pronalazi optimalno rešenje geometrijskim metodom i pri tome grafički prikazuje sve medjukorake. Za grafičko prikazivanje skupa Ω_P dopustivih rešenja, koristili smo sledeće funkcije programskog jezika MATHEMATICA: `InequalityPlot`, `InequalitySolve`, `FindInstance`, itd. Ove funkcije su iz paketa `Graphics‘InequalityGraphics` i `Algebra’InequalitySolve`. Kompletan kod programa GEOM kao i detalji implementacije prikazani su u dodatku, a mogu se naći i u našem radu [47]. Razmotrimo sada rad programa GEOM na sledećem primeru:

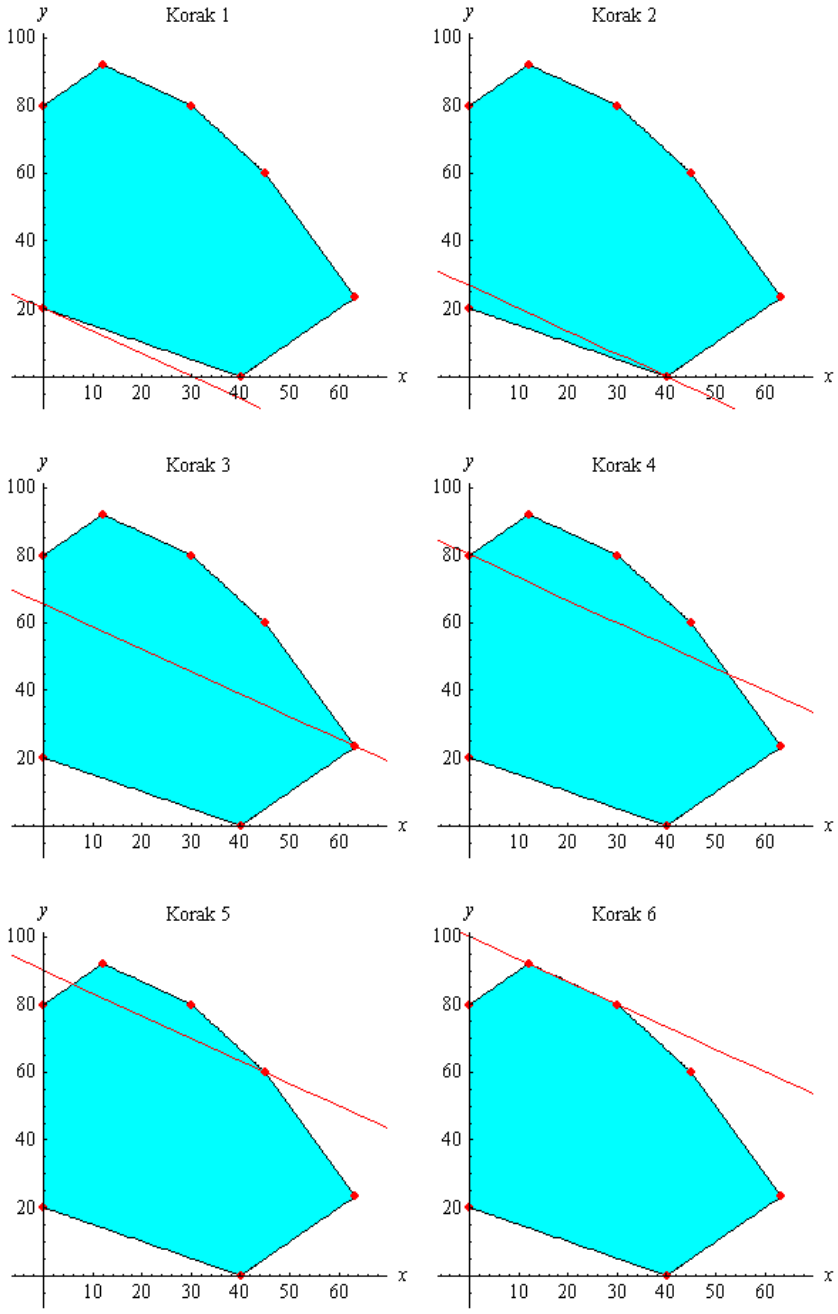
Primer 2.3.1 Rešiti problem linearnog programiranja:

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

Problem rešavamo sledećom naredbom:

```
Geom[8x+12y, {8x+4y<=600, 2x+3y<=300, 4x+3y<=360, 5x+10y>=600, x-y>=80, x-y<=40,
x>=0, y >= 0}]
```

Program nam daje grafički prikaz skupa dopustivih rešenja Ω_P, kao i prave koja odgovara ciljnoj funkciji ($\mathcal{H}_{f,d}$). 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 $\mathcal{H}_{f,d}$ kada prolazi kroz temena poligona (ekstremne tačke):



Slika 2.3.1. Vizuelizacija geometrijskog metoda.

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

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

2.4 Simpleks metod

Sada ćemo izložiti algebarski metod koji rešava opšti problem linearnog programiranja, a potiče od G. B. Danciga [9]. Danas se uglavnom koristi modifikovana varijanta simpleks metoda koja potiče od A. Takera [32, 50].

Koristićemo problem linearnog programiranja u simetričnom obliku:

$$\begin{aligned} \min \quad & f(x) = \sum_{i=1}^{n} c_i x_i + d \\ \text{p.o} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, m, \\ & x_j \geq 0, \quad j = 1, \ldots, n. \end{aligned} \tag{2.4.7}$$

Definicija 2.4.1 *Problem (2.4.7) 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 promenljive kao u (2.1.6) dobijamo:

$$\begin{aligned} \min \quad & f(x) = \sum_{i=1}^{n} c_i x_i + d \\ \text{p.o} \quad & \sum_{j=1}^{n} a_{ij} x_j - b_i = -x_{n+i}, \quad i = 1, \ldots, m \end{aligned} \tag{2.4.8}$$

Oblik (2.4.8) 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 redom označavati sa $x_{B,1}, \ldots, x_{B,m}$ i $x_{N,1}, \ldots, x_{N,n}$. 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 takodje da je matrica sistema u standardnom obliku (2.4.8) $A' = [A|I_m]$ gde je I_m jedinična matrica. I_m je bazična matrica koja odgovara upravo konstruisanom bazičnom rešenju.

Ako je problem zadat u standardnom obliku (2.1.4)

$$\begin{aligned} & \min c'^T x, \\ & A'x = b', \\ & x \geq 0, \end{aligned} \tag{2.4.9}$$

možemo konstruisati kanonski oblik direktno, odabiranjem m promenljivih za bazične $(x_{B,1}, \ldots, x_{B,m})$, i izražavanjem tih promenljivih preko ostalih. Pri tome je $x_{B_i} = x_{v_{B,i}}$ za neke vrednosti $v_{B,1}, \ldots, v_{B,m}$ tako da su kolone $K_{v_{B,i}}$ matrice A linearno nezavisne.

Naredne dve leme pokazuju dva karakteristična slučaja u simpleks metodu.

Lema 2.4.1 *Neka u bazično dopustivom kanonskom obliku (2.4.8) važi $c_j \leq 0$ za svako $j = 1, \ldots, n$. Tada je bazično rešenje $x^* = (0, \ldots, 0, b_1, \ldots, b_m)$ optimalno.*

Dokaz. Pošto je $x_B^* = b' \geq 0$ a $x_N^* = 0$, sledi 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

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

pa je x^* optimalno rešenje. □

Lema 2.4.2 *Neka u bazično dopustivom kanonskom obliku (2.4.8)* $b_i \geq 0$, $i = 1, \ldots, m$ *i za neko* $k \in \{1, \ldots, n\}$ *je ispunjeno* $c_k > 0$ *i* $a_{ik} \leq 0$, $i = 1, \ldots, m$. *Tada je ciljna funkcija na dopustivom skupu neograničena odozgo.*

Dokaz. Pošto je $a_{ik} \leq 0$, tada će za proizvoljno $t > 0$ rešenje $x = (0, \ldots, 0, t, 0, \ldots, 0, b_1, \ldots, b_m)$ biti bazično dopustivo. Medjutim, tada vrednost funkcije cilja $f(x) = c_k t$ neograničeno raste, kada $t \to +\infty$, pa prema tome $f(x)$ nije ograničena odozgo. □

Ukoliko problem (2.4.8) zadovoljava uslove jedne od dve navedene leme, tada ili direktno dobijamo optimalno rešenje, ili dolazimo do zaključka da je funkcija cilja neograničena i da problem nema rešenja. Nadalje ćemo pokazati kako se opšti bazično dopustivi kanonski problem svodi na problem koji zadovoljava jednu od dve navedene leme.

Neka je, u opštem slučaju zadat bazično dopustiv problem (2.4.8) kod koga je $c_j > 0$ za neko $j \in \{1, \ldots, n\}$ i $a_{ij} > 0$ za neko $i \in \{1, \ldots, m\}$.

Pošto je $c_j > 0$, 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 $f(x)$ 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

$$\frac{b_p}{a_{pj}} = \min \left\{ \frac{b_i}{a_{ij}} \mid a_{ij} > 0,\ 1 \leq i \leq n \right\}. \tag{2.4.10}$$

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 $x_{N,j}$ medju 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:

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

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 leme (2.4.1) i (2.4.2). Ovo intuitivno razmatranje je ključno za simpleks metod, i biće u sledećem odeljku formalnije sprovedeno. Takodje, uvešćemo pojam Takerove tabele koji u mnogome pojednostavljuje upravo opisan postupak konstruisanja novog bazičnog rešenja. Ovakav pristup se najčešće koristi prilikom implementacije simpleks metoda.

2.5 Takerove tabele i Simpleks metod za bazično dopustive kanonske oblike

Neka je zadat jedan kanonski oblik problema linearnog programiranja:

$$\begin{aligned}
&\max f(x) = c_1 x_{N,1} + \ldots + c_n x_{N,n} + d \\
&a_{11}x_{N,1} + a_{12}x_{N,2} + \ldots + a_{1n}x_{N,n} - b_1 = -x_{B,1} \\
&a_{21}x_{N,1} + a_{22}x_{N,2} + \ldots + a_{2n}x_{N,n} - b_2 = -x_{B,2} \\
&\cdots\cdots\cdots \\
&a_{m1}x_{N,1} + a_{m2}x_{N,2} + \ldots + a_{mn}x_{N,n} - b_m = -x_{B,m}.
\end{aligned} \tag{2.5.11}$$

Problem možemo tabelarno prikazati na sledeći način:

$$\begin{array}{cccccc}
x_{N,1} & x_{N,2} & \cdots & x_{N,n} & -1 & \\
a_{11} & a_{12} & \cdots & a_{1n} & b_1 & = & -x_{B,1} \\
a_{21} & a_{22} & \cdots & a_{2n} & b_2 & = & -x_{B,2} \\
\vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\
a_{m1} & a_{m2} & \cdots & a_{mn} & b_m & = & -x_{B,m} \\
c_1 & c_2 & \cdots & c_n & d & = & f
\end{array} \tag{2.5.12}$$

Ovu tabelu nazivamo ***Takerovom tabelom*** za problem (2.5.11). Uvedimo oznake $a_{m+1,j} = c_j$ za $j = 1, \ldots, n$ kao i $a_{i,n+1} = b_i$ za $i = 1, \ldots, n$. Takodje, neka je $a_{m+1,n+1} = d$. Sada nam je za opisivanje kanonskog oblika linearnog programiranja dovoljna *proširena Takerova 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.

Na kraju prethodnog odeljka bilo je potrebno izraziti nebazičnu promenljivu $x_{N,j}$ iz p-te jednačine sistema (2.5.11) 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 sad šta se dešava sa matricom sistema A' odgovarajućeg standardnog oblika (2.4.9). U matrici A' kolone koje odgovaraju bazičnim promenljivima formiraju jediničnu matricu dok ostale formiraju Takerovu 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:

$$V_q^1 = V_q - \frac{a_{qj}}{a_{pj}} V_p, \qquad (a')_{ql}^1 = a'_{ql} - \frac{a'_{pl} a'_{q,v_{B,j}}}{a'_{p,v_{B,j}}} \tag{2.5.13}$$

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

$$A^1 = \begin{array}{ccccccccccc} x_{N,1} & x_{N,2} & \cdots & x_{N,j-1} & x_{B,p} & x_{N,j+1} & \cdots & x_{N,n} & -1 & & \\ a^1_{11} & a^1_{12} & \cdots & a^1_{1,j-1} & a^1_{1j} & a^1_{1,j+1} & \cdots & a^1_{1n} & b^1_1 & = & -x_{B,1} \\ a^1_{21} & a^1_{22} & \cdots & a^1_{2,j-1} & a^1_{2j} & a^1_{2,j+1} & \cdots & a^1_{2n} & b^1_2 & = & -x_{B,2} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a^1_{p-1,1} & a^1_{p-1,2} & \cdots & a^1_{p-1,j-1} & a^1_{p-1,j} & a^1_{p-1,j+1} & \cdots & a^1_{p-1,n} & b^1_{p-1} & = & -x_{B,p-1} \\ a^1_{p1} & a^1_{p2} & \cdots & a^1_{p,j-1} & a^1_{pj} & a^1_{p,j+1} & \cdots & a^1_{pn} & b^1_p & = & -x_{B,j} \\ a^1_{p+1,1} & a^1_{p+1,2} & \cdots & a^1_{p+1,j-1} & a^1_{p+1,j} & a^1_{p+1,j+1} & \cdots & a^1_{p+1,n} & b^1_{p+1} & = & -x_{B,p+1} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a^1_{m1} & a^1_{m2} & \cdots & a^1_{m,j-1} & a^1_{mj} & a^1_{m,j+1} & \cdots & a^1_{mn} & b^1_m & = & -x_{B,m} \\ c^1_1 & c^1_2 & \cdots & c^1_{j-1} & c^1_j & c^1_{j+1} & \cdots & c^1_n & d^1 & = & f \end{array} \tag{2.5.14}$$

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

$$\begin{aligned} a^1_{pj} &= \frac{1}{a_{pj}}; \\ a^1_{pl} &= \frac{a_{pl}}{a_{pj}}, && l \neq j; \\ a^1_{qj} &= -\frac{a_{qj}}{a_{pj}}, && q \neq p; \\ a^1_{ql} &= a_{ql} - \frac{a_{pl} a_{qj}}{a_{pj}}, && q \neq p,\ l \neq j; \end{aligned} \tag{2.5.15}$$

Naravno, ovde podrazumevamo da je $q = 1, \ldots, m+1$ i $l = 1, \ldots n+1$. Izrazi (2.5.15) dobijaju se direktno iz izraza (2.5.13) ako se uzme u obzir struktura matrice A'.

Označimo ovu transformaciju sa $\mathcal{T}_{p,j}$. Element a_{pj} nazivamo ***ključnim*** ili ***pivot*** elementom. Transformaciju $\mathcal{T}_{p,j}$ izvršava sledeći algoritam:

Algoritam 1 Replace - *Zamena promenljivih $x_{N,j}$ i $x_{B,p}$*

- **Korak 1.** *Izračunati nove vrednosti koeficijenata a_{ql} prema formulama (2.5.15).*

- **Korak 2.** *Zameniti promenljive $x_{N,j}$ i $x_{B,p}$.*

Posmatrajmo sada transformaciju Takerove tabele $\mathcal{T}_{p,j}$, pri čemu je j takvo da je $c_j > 0$ a p izabrano pomoću (2.4.10).

Lema 2.5.1 *Neka je $\mathcal{T}_{p,j}$ transformacija Takerove tabele, pri čemu je $c_j > 0$ a p izabrano pomoću (2.4.10). Novodobijena Takerova tabela je bazično dopustiva, pri čemu vrednost funkcije cilja u odgovarajućem bazičnom rešenju veća ili jednaka odgovarajućoj vrednosti za polaznu tabelu. Takodje je $c_j^1 < 0$.*

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

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

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

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

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

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

Posle smene je

$$c_j^1 = -\frac{c_j}{a_{pj}} < 0.$$

Na kraju, imamo da je nova vrednost funkcije cilja u bazično dopustivom rešenju:

$$d^1 = d - \frac{c_j b_p}{a_{pj}} \geq d$$

čime je lema dokazana. □

Sada možemo da formulišimo simpleks metod za bazično dopustive kanonske probleme linearnog programiranja:

Algoritam 2 BasicMax

- **Korak 1.** *Ako je $c_1, \ldots, c_n \leq 0$, STOP. Bazično dopustivo rešenje koje odgovara Takerovoj tabeli je optimalno.*
- **Korak 2.** *Izabrati proizvoljno $c_j > 0$.*
- **Korak 3.** *Ispitati da li je $a_{ij} \leq 0$ za svako $i = 1, \ldots, m$. Ako jeste, STOP. Ciljna funkcija je na dopustivom skupu neograničena odozgo, tj. maksimum je $+\infty$.*
- **Korak 4.** *Izračunati*

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

i zameniti nebazičnu promenljivu $x_{N,j}$ i bazičnu promenljivu $x_{B,p}$, koristeći Algoritam Replace i preći na Korak 1.

Na kraju ovog odeljka, razmotrimo prethodno opisanu teoriju i algoritme na jednom primeru:

Primer 2.5.1 Posmatrajmo sledeći problem:

$$\begin{aligned} \max \quad & 5x_1 + 4x_2, \\ \text{p.o.} \quad & x_1 + x_2 \leq 80, \\ & 3x_1 + x_2 \leq 180, \\ & x_1 + 3x_2 \leq 180, \\ & x_1 \geq 0,\ x_2 \geq 0 \end{aligned}$$

odgovara Takerova tabela T_0 : (pivot elementi su zbog preglednosti uokvireni)

$$T_0 = \begin{array}{cccc} x_1 & x_2 & -1 & \\ 1 & 1 & 80 & = & -x_3 \\ \boxed{3} & 1 & 180 & = & -x_4 \\ 1 & 3 & 180 & = & -x_5 \\ 5 & 4 & 0 & = & f \end{array}$$

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

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

odakle odredjujemo $p = 2$. 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:

$$T_1 = \begin{array}{cccc} x_4 & x_2 & -1 & \\ \frac{1}{3} & \boxed{\frac{2}{3}} & 20 & = & -x_3 \\ \frac{1}{3} & \frac{1}{3} & 60 & = & -x_1 \\ -\frac{1}{3} & \frac{8}{3} & 120 & = & -x_5 \\ -\frac{5}{3} & \frac{7}{3} & -300 & = & f \end{array}$$

$$\begin{aligned} a^1_{21} &= a^1_{pj} = \frac{1}{a_{pj}} = 1, \\ a^1_{22} &= a^1_{pl} = \frac{a_{pl}}{a_{pj}} = \frac{a_{22}}{a_{21}} = \frac{1}{3}, \\ a^1_{11} &= a^1_{qj} = -\frac{a_{qj}}{a_{pj}} = -\frac{a_{11}}{a_{21}} = -\frac{1}{3}, \\ a^1_{31} &= a^1_{qj} = -\frac{a_{qj}}{a_{pj}} = -\frac{a_{31}}{a_{21}} = -\frac{1}{3} \\ a^1_{32} &= a^1_{ql} = a_{ql} - \frac{a_{pl}a_{qj}}{a_{pj}} = a_{32} - \frac{a_{22}a_{31}}{a_{21}} = \frac{8}{3}. \end{aligned}$$

$$\begin{aligned} b^1_1 &= b^1_q = b_q - \frac{b_p a_{qj}}{a_{pj}} = b_1 - \frac{b_2 a_{11}}{a_{21}} = 80 - \frac{80 \cdot 1}{3} = 20, \\ b^1_2 &= b^1_p = \frac{b_p}{a_{pj}} = \frac{b_2}{a_{21}} = \frac{180}{3} = 60, \\ b^1_3 &= b^1_q = b_q - \frac{b_p a_{qj}}{a_{pj}} = b_3 - \frac{b_2 a_{31}}{a_{21}} = 180 - \frac{180 \cdot 1}{3} = 120. \end{aligned}$$

$$\begin{aligned} c^1_1 &= c^1_j = -\frac{c_j}{a_{pj}} = -\frac{c_1}{a_{21}} = -\frac{5}{3} = -\frac{5}{3}, \\ c^1_2 &= c^1_l = c_l - \frac{c_j a_{pl}}{a_{pj}} = c_2 - \frac{c_1 a_{22}}{a_{21}} = 4 - \frac{5 \cdot 1}{3} = \frac{7}{3}, \\ d^1 &= d - \frac{b_2 c_1}{a_{21}} = 0 - \frac{180 \cdot 5}{3} = -300. \end{aligned}$$

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

$$b^1_1/a^1_{12} = 30,\ b^1_2/a^1_{22} = 180,\ b^1_3/a^1_{32} = 45,$$

odakle proizilazi $p = 1$. Znači, potrebno je da se izvrši zamena $x_2 \leftrightarrow x_3$. Primenom algoritma **Replace** dobija se:

$$T_2 = \begin{array}{cccc} x_4 & x_3 & -1 & \\ -\frac{1}{2} & \frac{3}{2} & 30 & = -x_2 \\ \frac{1}{2} & -\frac{1}{2} & 50 & = -x_1 \\ 1 & -4 & 40 & = -x_5 \\ -\frac{1}{2} & -\frac{7}{2} & -370 & = f \end{array}$$

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 $z = 370$.

2.6 Simpleks metod za nalaženje prvog bazično dopustivog rešenja

U ovom odeljku razmotrićemo problem pronalaženja prvog bazično dopustivog rešenja (kanonskog oblika). Najčešće su u upotrebi dva metoda kojima se rešava ovaj problem.

Kod prvog metoda [7] koristimo problem linearnog programiranja u standardnom obliku. Neka je dat problem linearog programiranja u standardnom obliku

$$\begin{array}{ll} \min & c^T x, \\ \text{p.o.} & Ax = b, \\ & x \geq 0. \end{array} \tag{2.6.16}$$

Jasno je da bez umanjenja opštosti možemo pretpostaviti da je u standardnom obliku $b \geq 0$ (u suprotnom pomnožimo odgovarajuće jednačine sa -1). Problemu (2.6.17) pridružimo pomoćni problem linearnog programiranja:

$$\begin{array}{ll} \min & e^T w, \\ \text{p.o.} & Ax + w = b, \\ & x \geq 0,\ w \geq 0, \end{array} \tag{2.6.17}$$

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 (2.6.18) neprazan jer mu sigurno pripada tačka ($x = 0$, $w = b$). Takodje je jasno da je ciljna funkcija na tom skupu odozdo ograničena sa nulom. Dopustivu bazu problema (2.6.18) čine kolone koje odgovaraju promenljivim $w_1, \ldots, w_m$, a kanonski oblik problema (2.6.17) se dobija eliminacijom w iz ciljne funkcije pomoću jednačine $w = b - Ax$. Problemi (2.6.17) i (2.6.18) su povezani sledećom teoremom:

Lema 2.6.1 *Skup dopustivih rešenja problema (2.6.17) je neprazan ako i samo ako je optimalna vrednost ciljne funkcije problema (2.6.18) jednaka nuli.*

Dokaz. Neka je $\bar{x}$ dopustivo rešenje problema (2.6.17). Tada je $(\bar{x}, 0)$ dopustivo rešenje problema (2.6.18), pri čemu je vrednost ciljne funkcije jednaka nuli. S obzirom da je nula

donja granica za ciljnu funkciju problema (2.6.17), 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 (2.6.17) 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 (2.6.17). □

Na prethodnoj teoremi se zasniva tzv. *dvofazna modifikacija simpleks metoda*. U jednoj varijanti ovog metoda [7, 44], pomoćni problem se formira na sledeći način:

$$\begin{aligned} \min \quad & c^T x + Me^T w, \\ \text{p.o.} \quad & Ax + w = b, \\ & x \geq 0,\ w \geq 0, \end{aligned} \tag{2.6.18}$$

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 neodredjenost oko izbora konstante M.

Razmotrimo sada metod iz [50] kod koga nema povećanja dimenzija problema. Posmatraćemo jedan kanonski oblik (2.5.11) problema linearnog programiranja:

$$\begin{aligned} & \max f(x) = c_1 x_{N,1} + \ldots + c_n x_{N,n} + d \\ & a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n - b_1 = -x_{B,1} \\ & a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n - b_2 = -x_{B,2} \\ & \ldots\ldots\ldots \\ & a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n - b_m = -x_{B,m}. \end{aligned} \tag{2.6.19}$$

pri čemu nisu svi $b_i \geq 0$, tj postoji neko $i \in \{1, \ldots, m\}$ takvo da je $b_i < 0$. Sada odgovarajuće bazično rešenje $x = (0, \ldots, 0, b_1, \ldots, b_n)$ nije dopustivo, tj ne pripada skupu Ω_P. Zadatak koji rešavamo u ovom odeljku je nalaženje ekvivalentnog bazično dopustivog problema (2.6.19), odnosno nalaženje jednog bazično dopustivog rešenja. Dokažimo najpre jednu pomoćnu lemu.

Lema 2.6.2 *Ako je $a_{i1}, \ldots, a_{in} \geq 0$ i $b_i < 0$ tada i-ta jednačina u sistemu (2.6.19) nema rešenja, tj. sistem je nedopustiv.*

Dokaz. Pretpostavimo da postoji jedno rešenje x problema (2.6.19). Uz navedene pretpostavke dobijamo:

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

odnosno $x_{B,i} < 0$. Kontradikcija. □

Znači, ukoliko je polazni problem dopustiv, moguće je izabrati neko $a_{ij} < 0$, za prethodno izabrano $b_i < 0$. Ako bi sada izvršili transformaciju $\mathcal{T}_{i,j}$, odnosno zamenili promenljive $x_{N,j}$ i $x_{B,i}$, nova vrednost za b_i bi bila pozitivna. Medjutim, može da se desi da je za neko t vrednost b_t bila nenegativna pre transformacije a da je posle postala negativna. Sledeći algoritam [50, 46] isključuje tu mogućnost za $t > i$.

Algoritam 3 NoBasicMax *Algoritam simpleks metoda za probleme koji nisu bazično dopustivi.*

- **Korak 1.** *Ako su* $b_1, b_2, \ldots, b_m \geq 0$*, preći na Korak 5.*
- **Korak 2.** *Izabrati* $b_i < 0$*, tako da je* i *maksimalno.*
- **Korak 3.** *Ako važi* $a_{i1}, a_{i2}, \ldots, a_{in} \geq 0$*, STOP. Problem linearnog programiranja je nedopustiv (na osnovu leme 2.6.2). U suprotnom izabrati* $a_{ij} < 0$*.*
- **Korak 4.** *Ako je* $i = m$*, izabrati za ključni element* a_{mj}*, izvršiti transformaciju i preći na Korak 1. U suprotnom, ako je* $i < m$*, izabrati* $a_{ij} < 0$ *i izračunati*

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

Izabrati za ključni element a_{pj}*, izvršiti transformaciju, i preći na Korak 1.*

- **Korak 5.** *Primeniti simpleks algoritam za bazično dopustive probleme, algoritam* **BasicMax***.*

Primetimo, da ako je $b_i < 0$ za svako $i = 1, \ldots, m$, uzimanjem proizvoljnog elementa a_{ij} za pivot dobijamo novu Takerovu tabelu koja ima bar jedan pozitivan b_i. U ovom algoritmu, biramo poslednji negativan b_i. Ako u koraku 4 bude $p = i$, u sledećoj iteraciji b_i postaje pozitivno, ako je $p > i$, b_i ostaje negativno. U oba slučaja svi $b_{i+1}, \ldots, b_m$ ostaju pozitivni.

Takodje, mogu se na sličan način, kao za algoritam (**BasicMax**) uvesti anticiklična pravila (samo što će sad ulogu ciljne funkcije imati i-ta vrsta Takerove tabele, odnosno i-ta jednačina). U sledećoj glavi, pokazaćemo da algoritam **NoBasicMax** ima nekoliko nedostataka i izložićemo načine za otklanjanje tih nedostataka [41, 34, 40].

Primer 2.6.1 Primenimo prethodni algoritam na sledeći primer zadat Takerovom tabelom.

$$T_0 = \begin{array}{cccc} x_1 & x_2 & -1 & \\ -1 & -2 & -3 & = & -x_3 \\ 1 & 1 & 3 & = & -x_4 \\ \boxed{1} & 1 & 2 & = & -x_5 \\ -2 & 4 & 0 & = & f \end{array}$$

1. Prelazimo na korak 2 jer je $b_1 = -3$ negativno.
2. Biramo $b_1 = -3$.
3. Prelazimo na korak 4, jer su oba koeficijenta $a_{11} = -1$ i $a_{12} = -2$ negativna.
4. Možemo da izaberemo $a_{11} = -1$ ili $a_{12} = -2$. Radi odredjenosti 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} i posle primene algoritma **Replace** dobijamo sledeću tabelu:

$$T_1 = \begin{array}{cccc} x_5 & x_2 & -1 & \\ -1 & -1 & -1 & = & -x_3 \\ -1 & 0 & 1 & = & -x_4 \\ 1 & 1 & 2 & = & -x_1 \\ 2 & 6 & 4 & = & z \end{array}$$

Prelazimo na korak 1.

1. Prelazimo na korak 2, jer je $b_2 = -1$ negativno.

2. Moramo da izaberemo $b_2 = -1$.

3. Prelazimo na korak 4, jer je $a_{12} = -1$ negativno.

4. 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}. Nova tabela je:

$$T_2 = \begin{array}{cccc} x_5 & x_3 & -1 & \\ -1 & \boxed{-1} & 1 & = & -x_2 \\ -1 & 0 & 1 & = & -x_4 \\ 2 & 1 & 1 & = & -x_1 \\ 8 & 6 & -2 & = & z \end{array}$$

Prelazimo na korak 1.

1. $b_1, b_2, b_3 \geq 0$, tj. tabela je bazično dopustiva. Sada možemo primeniti algoritam **BasicMax**

2.7 Eliminacija jednačina i slobodnih promenljivih

Još na početku ovog rada, kada smo definisali oblike problema linearnog programiranja (odeljak 2.1), pokazali smo kako se opšti oblik problema linearnog programiranja svodi na standardni, odnosno simetrični oblik. Ovde ćemo pokazati kako se mogu iskoristiti Takerove tabele i zamena promenljivih (algoritam **Replace**) za svodjenje opšteg problema linearnog programiranja na kanonski oblik. Znači, posmatrajmo sada opšti oblik problema

linearnog programiranja 2.1.1.

$$\begin{aligned} & \max f(x) = c_1x_1 + \ldots + c_nx_n + d \\ N_i^{(1)}: \quad & \sum_{j=1}^{n} a_{ij}x_j \le b_i, \quad i = 1, \ldots, p \\ N_i^{(2)}: \quad & \sum_{j=1}^{n} a_{ij}x_j \ge b_i, \quad i = p+1, \ldots, q \\ J_i: \quad & \sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i = q+1, \ldots, m \\ & x_j \ge 0, \quad j \in \mathcal{J} = \{1, \ldots, s\}, \quad s \le n \end{aligned} \tag{2.7.20}$$

Pomnožimo sve nejednačine tipa $N^{(2)}$ sa -1 i uvedimo dodatne promenljive $x_{n+1}, \ldots, x_{n+q}$ kao kod svodjenja 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.

$$\begin{aligned} & \max f(x) = c_1x_{N,1} + \ldots + c_nx_{N,n} + d \\ & a_{11}x_{N,1} + a_{12}x_{N,2} + ... + a_{1n}x_{N,n} - b_1 = -x_{B,1} \\ & a_{21}x_{N,1} + a_{22}x_{N,2} + ... + a_{2n}x_{N,n} - b_2 = -x_{B,2} \\ & \ldots\ldots\ldots \\ & a_{q1}x_{N,1} + a_{q2}x_{N,2} + ... + a_{qn}x_{N,n} - b_m = -x_{B,q} \\ & a_{q+1,1}x_{N,1} + a_{q+1,2}x_{N,2} + ... + a_{q+1,n}x_{N,n} - b_{q+1} = -0 \\ & \ldots\ldots\ldots \\ & a_{m1}x_{N,1} + a_{m2}x_{N,2} + ... + a_{mn}x_{N,n} - b_m = -0 \end{aligned} \tag{2.7.21}$$

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 Takerova tabela ima sledeći izgled:

$$\begin{array}{ccccccc} x_{N,1} & x_{N,2} & \cdots & x_{N,n} & -1 & & \\ a_{11} & a_{12} & \cdots & a_{1n} & b_1 & = & -x_{B,1} \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 & = & -x_{B,2} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{q1} & a_{q2} & \cdots & a_{qn} & b_q & = & -x_{B,q} \\ a_{q+1,1} & a_{q+1,2} & \cdots & a_{q+1,n} & b_{q+1} & = & -x_{B,q+1} \equiv 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m & = & -x_{B,m} \equiv 0 \\ c_1 & c_2 & \cdots & c_n & d & = & f \end{array} \tag{2.7.22}$$

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 sad $a_{mj} \neq 0$ za neko j.

Ukoliko zamenimo promenljive $x_{B,m}$ i $x_{N,j}$ dobijamo:

$$\begin{array}{cccccccc} x_{N,1} & \cdots & x_{B,m} \equiv 0 & \cdots & x_{N,n} & -1 & & \\ a_{11} & \cdots & a_{1j} & \cdots & a_{1n} & b_1 & = & -x_{B,1} \\ \vdots & & \vdots & & \vdots & \vdots & & \vdots \\ a_{m1} & \cdots & a_{mj} & \cdots & a_{mn} & b_m & = & -x_{N,j} \\ c_1 & \cdots & c_j & \cdots & c_n & d & = & f \end{array} \tag{2.7.23}$$

Očigledno, sada nezavisna promenljiva $x_{B,m} \equiv 0$ ne utiče ni na funkciju cilja ni na uslove. Sledi da j-tu kolonu tabele (2.7.23) 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. Prema tome imamo:

Algoritam 4 ElJed *(Eliminacija jednačina)*

- **Korak 1.** *Ako ne postoji nijedna bazična promenljiva identički jednaka nuli, primeniti algoritam* **NoBasicMax**.
- **Korak 2.** *Neka je $x_{B,i} \equiv 0$. Nadjimo $a_{ij} \neq 0$. 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. Takodje, 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 5 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 imamo 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 2.7.1 Neka je data sledeća tabela:

$$T_0 = \begin{array}{cccccc} x_1 & \boxed{x_2} & x_3 & -1 & & \\ 1 & 1 & 1 & 6 & = & -0 \\ 1 & 1 & 0 & 1 & = & -x_4 \\ 1 & 2 & 1 & 0 & = & f \end{array}$$

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'.

$$T_1 = \begin{array}{cccccc} x_1 & 0 & x_3 & -1 & & \\ 1 & 1 & 1 & 6 & = & -\boxed{x_2} \\ 0 & -1 & -1 & -5 & = & -x_4 \\ -1 & -2 & -1 & -12 & = & f \end{array} \qquad T_1' = \begin{array}{cccccc} x_1 & x_3 & -1 & & \\ 0 & -1 & -5 & = & -x_4 \\ -1 & -1 & -12 & = & f \end{array}$$

Primetimo da ako sada izaberemo (prema algoritmu **NoBasicMax**) a_{12} za pivot, dobijamo optimalno rešenje.

2.8 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 Takerove 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 Takerove tabele računaju pomoću polazne matrice problema. To se postiže revidiranim simpleks metodom.

Ovde posmatramo problem linearnog programiranja u standardnom obliku (2.1.4). Neka je dato jedno bazično rešenje (dobijeno rešavanjem sistema $Ax = b$ ili metodom eliminacije jednačina) x^*. Posmatrajmo sada polazni problem u kanonskom obliku:

$$\begin{array}{ll} \max & (c^*)^T x_N - d \\ p.o. & Tx_N - b^* = -x_B \\ & x = (x_B, x_N) \geq 0 \end{array} \tag{2.8.24}$$

Označimo sada sa n i m redom broj nebazičnih i bazičnih promenljivih a sa T matricu tipa $m \times n$ koja predstavlja Takerovu tabelu. Vektori b^* i c^* su odgovarajući slobodni vektor i vektor funkcije cilja u kanonskom obliku. Sistem jednačina u (2.8.24) 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 i (2.8.24) dobijamo direkno da važi:

$$T = A_B^{-1} A_N \qquad b^* = A_B^{-1} b$$

Znači, ako znamo indekse bazičnih promenljivih $v_{B,1}, \dots v_{B,m}$ ($x_{B,i} = x_{v_{B,i}}$) možemo da rekonstruišemo Takerovu tabelu pomoću formula (2.8). Pri tome je $A_B = [K_{v_{B,1}} \cdots K_{v_{B,m}}]$.

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 $x_c = (x_1, \dots, x_{n+1})$. Takodje je i $(A_c)_B = \begin{bmatrix} A_B & 0 \\ c_B & 1 \end{bmatrix}$. Znači, ako sada primenimo formule (2.8) za prošireni sistem $A_c x_c = \begin{bmatrix} b^* \\ d \end{bmatrix}$, u potpunosti možemo da rekonstruišemo Takerovu tabelu.

Kao što možemo videti, kod upravo izloženog metoda rekonstrukcije Takerove tabele nema nagomilavanja numeričkih grešaka, pa je ovaj metod stabilniji od klasičnog simpleks metoda. Medjutim, 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.

Takodje, u algoritmima simpleks metode nije potrebno da znamo celu Takerovu tabelu. Dovoljno je da znamo (da rekonstruišemo) pojedine redove ili kolone Takerove tabele. U nastavku ćemo prezentirati metod kod koga smo izračunavanje potrebnog dela Takerove 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 [2, 34, 44]:

$$T_{i\bullet} = (A_B^{-1})_{i\bullet} A_N \tag{2.8.25}$$

Vektor $(A_B^{-1})_{i\bullet}$ odredjujemo iz sledeće jednačine:

$$(A_B^{-1})_{i\bullet} A_B = (I_m)_{i\bullet} \Longrightarrow A_B^T (A_B^{-1})_{i\bullet}^T = (I_m)_{i\bullet}^T \tag{2.8.26}$$

što takodje predstavlja sistem linearnih jednačina koji treba rešiti.

Napomenimo da ako za rešavanje sistema linearnih jednačina (2.8.25) i (2.8.26) koristimo metode kao što su Gausova eliminacija, LR faktorizacija, itd. [30] 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 6 RevBasicMax *(Revidirani simpleks metod za bazično dopustive kanonske oblike [44]) 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* $(A_B)_c^T((A_B)_c^{-1})^T_{m+1\bullet} = (I_{m+1})^T_{m+1\bullet}$, *rekonstruisati* $n+1$*-vi red Takerove 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 pronadjen u literaturi, tako da je delimično i originalan. Medjutim, korišćenje predhodnog alogritma bez sledećeg je besmisleno, jer se već u algoritmu **NoBasicMax** numeričke greške nagomilavaju.

Algoritam 7 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 *(Takerove 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 Takerove 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:

- Pošto 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.

2.9 Pojam cikliranja i anticiklična pravila

U lemi 2.5.1 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 [50]:

Primer 2.9.1 Posmatrajmo sledeći problem linearnog programiranja u simetričnom obliku:

$$\begin{aligned}\max f=&\frac{3}{4}x_1-20x_2+\frac{1}{2}x_3-6x_4\\ &\frac{1}{4}x_1-8x_2-x_3+9x_4\le 0\\ &\frac{1}{2}x_1-12x_2-\frac{1}{2}x_3+3x_4\le 0\\ &x_3\le 1\\ &x_1,x_2,x_3,x_4\ge 0.\end{aligned}\tag{2.9.27}$$

Formirajmo odgovarajući kanonski oblik, Takerovu tabelu i primenimo algoritam **BasicMax** 7 puta:

$$\begin{array}{cccccccc} x_1 & x_2 & x_3 & x_4 & -1 & & \\ \frac{1}{4} & -8 & -1 & 9 & 0 & = & -x_5 \\ \frac{1}{2} & -12 & -\frac{1}{2} & 3 & 0 & = & -x_6 \\ 0 & 0 & 1 & 0 & 1 & = & -x_7 \\ \frac{3}{4} & -20 & \frac{1}{2} & -6 & 0 & = & f \end{array} \tag{2.9.28}$$

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 [50, 46]. Medjutim još 1977. godine su Kotiah i Steinberg [24] otkrili čitavu klasu "neveštačkih" problema koji cikliraju. Takodje, 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***. Obradćemo dve vrste anticikličnih pravila: ***Leksikografsku metodu*** 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 Takerovoj 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 kao:

- **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 2.4.9, 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 2.9.2 *U slučaju Takerove 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 Takerove tabele T_4 sa novom vrednošću ciljne funkcije i cikliranje se izbegava.

Napomena 2.9.1 *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 2.9.1 *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^k_i, $i = 1, \ldots, m$, matrice A' ostaju leksikografski pozitivne. Za $i = p$ je $V^1_p = \frac{V_p}{a_{pj}}$ pa $a_{pj} > 0$ i $V_p \overset{lex}{>} 0$ povlači $V_p \overset{lex}{>} 0$. Za $i \neq p$ je

$$V^1_i = V_i - \frac{a_{ij}}{a_{pj}} V_p = a_{ij}\left(\frac{V_i}{a_{ij}} - \frac{V_p}{a_{pj}}\right) \overset{lex}{>} 0,$$

gde stroga nejednakost važi na osnovu prethodne napomene. Za $i \neq p$ i $a^k_{ij} \leq 0$ je $V^1_i = V_i + \frac{|a_{ij}|}{a_{pj}} V_p \overset{lex}{\geq} V_i \overset{lex}{>} 0$. Za $m+1$-vu vrstu važi:

$$V^1_{m+1} = V_{m+1} - \frac{a_{m+1,j}}{a_{pj}} V_p = V_{m+1} + \frac{|a_{m+1,j}|}{a_{pj}} V_p \overset{lex}{>} V_{m+1}$$

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 2.9.2 *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* $1, \ldots, m$

Razmotrimo sada ***Blandova pravila***. Po Blandu [4], treba primenjivati sledeće dve modifikacije koraka 2 i 4:

- **Korak 2'.** *Izabrati $c_j > 0$ tako da je indeks odgovarajuće nebazične promenljive $v_{N,j}$ najmanji.*
- **Korak 4'.** *Izračunati*

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

U slučaju jednakih vrednosti izabrati ono p takvo da je indeks odgovarajuće bazične promenljive $v_{B,p}$ 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 Takerovu tabelu. Ove oznake su iste kao u odeljku 2.8.

Teorema 2.9.2 *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 $y = (y_1, \ldots, y_n, y_{n+1})$ i $v = (v_1, \ldots, v_n, v_{n+1})$ 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 $y_1, \ldots, y_{q-1} \geq 0$ 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:

$$A_c v = [(A_c)_B \ (A_c)_N] \begin{bmatrix} T''_{\bullet t} \\ -(I_m)_{\bullet t} \end{bmatrix} = ((A_c)_N)_{\bullet t} - ((A_c)_N)_{\bullet t} = 0$$

Znači v pripada jezgru $\mathcal{N}(A_c)$ matrice A_c. Primetimo takodje da je vektor y baš $n+1$-va vrsta matrice $(A_c)_B^{-1}A_c$. Odavde sledi da važi $y^T v = 0$. 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 $y_j v_j > 0$.

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 $j \leq q$. Takodje, 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 $y_q v_q < 0$ pa je $1 \geq j < q$. Pošto je $y_j \geq 0$ sledi da je i $v_j \geq 0$.

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 takodje da se tokom cikliranja poslednja kolona Takerove tabele T (vektor b^*) ne menja. Naime, pošto se vrednost funkcija cilja $T_{m+1,n+1}$ ne menja, na osnovu formula (2.5.15) 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 [50].

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. Takodje postoji opasnost od izbora pivot elemenata koji su bliski nuli i koji prouzrokuju velike numeričke greške.

2.10 Složenost simpleks metoda i Minti-Kli poliedri

U uvodu smo napomenuli da i pored dobrih osobina koje je pokazao u praksi, simpleks algoritam nije polinomijalan. To tvrdjenje su prvi dokazali Minti i Kli u radu [22], još 1970. godine uz pretpostavku da se za pivot kolonu uzima prva kolona kod koje je $c_j < 0$. Kasnije je dokazano [23] 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 2.10.1 *Posmatrajmo sledeći problem linearnog programiranja zadat u kanonskom obliku i preko Takerove tabele:*

$$\min \epsilon^{n-1}x_1 + \epsilon^{n-2}x_2 + \ldots + \epsilon x_{n-1} + x_n$$

$$\begin{array}{rcccccccc}
x_1 & \leq t & x_1 & x_2 & \cdots & x_n & -1 & & \\
2\epsilon x_1 + x_2 & \leq t^2 & 1 & 0 & \cdots & 0 & t & = & -x_{n+1} \\
2\epsilon^2 x_1 + 2\epsilon x_2 + x_3 & \leq t^3 & 2\epsilon & 1 & \cdots & 0 & t^2 & = & -x_{n+2} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\
 & & 2\epsilon^{n-1} & 2\epsilon^{n-2} & \cdots & 1 & t^m & = & -x_{n+m} \\
2\epsilon^{n-1}x_1 + 2\epsilon^{n-2}x_2 + \ldots + 2\epsilon x_{n-1} + x_n & \leq t^n & \epsilon^n & \epsilon^{n-1} & \cdots & 1 & 0 & = & f \\
x \geq 0 & & & & & & & &
\end{array}$$

Označimo ovaj problem sa $\mathcal{P}_n(\epsilon, t)$ i nazovimo ga uopštenim problemom Minti-Kli-a dimenzije n.

Minti i Kli su u svom radu [22] 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 2.10.1 *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 Minti-Kli problema.

Lema 2.10.2 *Neka važe uslovi teoreme 2.10.1. 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) Takerove tabele posle l transformacija, dobijamo Takerovu 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 $l = 0$ tvrdjenje trivijalno važi, s obzirom da je $c_{ij}^0 = 0$ za svako $(i, j) \in \{1, \ldots, m+1\} \times \{1, \ldots, n\}$. Pretpostavimo da tvrdjenje 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_{ip}^k i t_{pj}^k tako da je $i \geq p$ a $p \geq j$. Prema tome, ako bar jedan od ova dva uslova ne važi, imamo da je $t_{ij}^{k+1} = t_{ij}^k$ i $c_{ij}^{k+1} = c_{ij}^k$, jer $p \notin [j, i)$ pa tvrdjenje leme važi.

Za $i = p$ imamo da je

$$t_{pj}^{k+1} = \frac{t_{pj}^k}{t_{pp}^k} = t_{pj}^k = (-1)^{c_{pj}^k} t_{pj}^0 = (-1)^{c_{pj}^{k+1}} t_{pj}^0$$

jer je $c_{pj}^{k+1} = c_{pj}^k$, za $j \geq p$. Za $j = p$ imamo da je

$$t_{ip}^{k+1} = -\frac{t_{ip}^k}{t_{pp}^k} = -t_{ip}^k = -(-1)^{c_{ip}^k} t_{ip}^0 = (-1)^{c_{ip}^{k+1}} t_{ip}^0$$

jer je $c_{ip}^{k+1} = c_{ip}^k + 1$ za $i \leq p$.

Za $j < p$ i $p < i$ imamo:

$$t_{ij}^{k+1} = t_{ij}^k - t_{pj}^k t_{ip}^k = (-1)^{c_{ij}^k} t_{ij}^0 - (-1)^{c_{ip}^k + c_{pj}^k} t_{ip}^0 t_{pj}^0$$

Pošto je $t_{ij}^0 = 2\epsilon^{i-j}$ i $c_{ip}^k + c_{pj}^k = c_{ij}^k$ imamo da važi:

$$t_{ij}^{k+1} = (-1)^{c_{ij}^k} \left(t_{ij}^0 - t_{ip}^0 t_{pj}^0\right) = (-1)^{c_{ij}^k} \left(2\epsilon^{i-j} - 4\epsilon^{i-p+p-j}\right) = -(-1)^{c_{ij}^k} 2\epsilon^{i-j} = (-1)^{c_{ij}^{k+1}} t_{ij}^0$$

U poslednjoj jednakosti smo iskoristili da je $c_{ij}^{k+1} = c_{ij}^k + 1$ jer je $i > p$ a $p > j$. 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 Takerovoj 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 2.10.1:

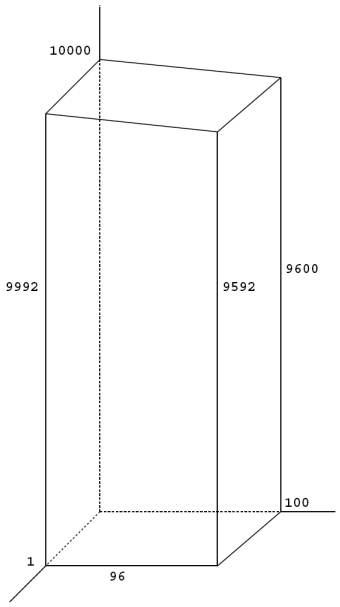
Lema 2.10.3 *U svakoj iteraciji algoritma* **BasicMax**, *primenjenog na problem* $\mathcal{P}_n(\epsilon, t)$ *važiće* $p = j$, *tj. pivot element biće na glavnoj dijagonali Takerove tabele.*

Poliedar dopustivih rešenja Ω_P za problem $\mathcal{P}(\epsilon, t)$ nazivamo *Minti-Kli poliedar*. Napomenimo, da ako izvršimo odgovarajuće smene promenljivih, Mini-Kli poliedar možemo opisati

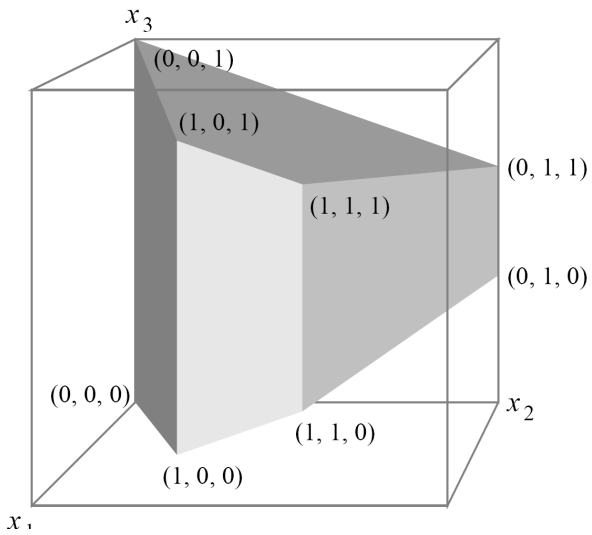
i sledećim sistemom nejednačina

$$\begin{array}{lll} \min & x_n & \\ \text{p.o.} & & x_1 \leq 1 \\ & \epsilon x_1 \leq & x_2 \leq 1 - \epsilon x_1 \\ & \epsilon x_2 \leq & x_3 \leq 1 - \epsilon x_2 \\ & \vdots & \\ & \epsilon x_{n-1} \leq & x_n \leq 1 - \epsilon x_{n-1} \\ & x \geq 0 & \end{array}$$

Geometrijski, poslednji sistem predstavlja deformisanu n-dimenzionalnu jediničnu hiperkocku. Na sledećoj slici su prikazani Minti-Kli poliedar za problem $\mathcal{P}_3(10, 100)$, kao i poliedar koji odgovara sistemu (2.10.29). Primetimo da putanja koju prolazi simpleks metod odgovara Hamiltonovom putu u odgovarajućem grafu poliedra.



Slika 2.10.1. Skup dopustivih rešenja za primer $\mathcal{P}_3(10, 100)$



Slika 2.10.2. Minti-Kli poliedar za $\epsilon = \frac{1}{3}$

Mini-Kli poliedri i njihove osobine proučavani od strane velikog broja autora. Uz višestruko ponavljanje odgovarajućih nejednakosti, u radu [11] je pokazano da odredjena klasa interior-point metoda takodje ima eksponencijalnu složenost u najgorem slučaju. U radu [12] 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 [1] (algoritama **Replace**, **NoBasicMax**, **ElJed**, **ElSl**). Ukazaćemo na nekoliko nedostataka odgovarajućih algoritama i predložićemo modifikacije kod kojih nema odgovarajućih nedostataka.

3.1 Dva poboljšanja simpleks metoda

U Algoritmu **Replace** opisan je postupak kojim se Takerova tabela transformiše u ekvivalentnu tabelu zamenom promenljivih. U praksi, Takerove 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 Takerovoj 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. Takodje, 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 akko su projekcije na ključnu vrstu a_{pl} i a_{qj} različite od 0.

Znači, bilo koji element iz Takerove tabele se menja akko su obe njegove projekcije različite od 0. Konstruišimo sada skupove V i K na sledeći način:

$$\begin{aligned} V &= \{l \mid a_{pl} \neq 0,\ l = 1, \ldots, n+1\}, \\ K &= \{q \mid a_{qj} \neq 0,\ q = 1, \ldots, m+1\}. \end{aligned} \tag{3.1.2}$$

Znači, bilo koji element iz Takerove tabele se menja akko su mu koordinate redom u skupovima V i K.

[1] Rezultati izloženi u ovoj glavi su preuzeti iz naših radova [41, 34, 40]

Algoritam 8 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

Posmatraćemo kanonički oblik problema linearnog programiranja (kao i u odeljku 2.5) zadatog pomoću Takerove tabele.

$$\begin{array}{cccccc} x_{N,1} & x_{N,2} & \cdots & x_{N,n} & -1 & \\ a_{11} & a_{12} & \cdots & a_{1n} & b_1 & = & -x_{B,1} \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 & = & -x_{B,2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m & = & -x_{B,m} \\ c_1 & c_2 & \cdots & c_n & d & = & f \end{array} \tag{3.2.3}$$

U ovom odeljku razmotrićemo modifikaciju metoda za nalaženje početnog bazično dopustivog rešenja (algoritma **NoBasicMax**). Ovi rezultati su preuzeti iz našeg rada [41]. 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

$$x_{B,t} = b_t^1 = \frac{b_t}{a_{tj}} \geq 0.$$

Takodje, 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 i neka je* $b_{i_1}, \ldots, b_{i_q} < 0$ *i* $I = \{i_1, \ldots, i_q\}$. *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

$$x_{B,m}^1 = b_m^1 = \frac{b_m}{a_{ms}} \geq 0.$$

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 [40].

Pošto je i_s fiksirano u koracima 4, 5 i 6, algoritam **ModNoBasicMax** može da ciklira samo ako je $b^1_{i_s} = b_{i_s}$. 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 9 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 staviti $p = 1$.

- **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 dodje do bazično dopustivog problema, ali da posle algoritam **BasicMax** napravi mnogo više iteracija nego kada se primeni algoritam **NoBasicMax**.

3.3 Poboljšana modifikacija Simpleks metoda za bazično nedopustive probleme

U našem radu [40], 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 $\frac{b_l}{a_{lj}} \geq \frac{b_i}{a_{ij}}$. Medjutim, 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 biti
$$\frac{b_i}{a_{ij}} \leq \min\left\{\frac{b_k}{a_{kj}} \,|\, b_k > 0, a_{kj} > 0\right\}$$
- 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 [40]:

Algoritam 10 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*
$$\begin{aligned} M(j_p) &= \min\left\{\frac{b_k}{a_{k,j_p}} \,|\, b_k > 0, a_{k,j_p} > 0\right\}. \\ p' &= \operatorname{argmin}\left\{\frac{b_k}{a_{k,j_p}} \,|\, b_k < 0, a_{k,j_p} < 0\right\}, \end{aligned}$$

  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 $x_{v_{N,j}} = x_{N,j}$). Odredjujemo 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). Takodje, 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 2.8 posmatramo problem linearnog programiranja u standardnom obliku:

$$\begin{aligned} &\min c^T x + d, \\ &Ax = b, \\ &x \geq 0, \end{aligned} \tag{3.4.5}$$

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 Takerovu tabelu na raspolaganju već samo jedan njen deo. U ovom odeljku ćemo opisati modifikaciju algoritma **RevNoBasicMax** baziranu na već obradjenim modifikacijama kod klasičnog simpleks metoda.

Ovde je glavna ideja da nadjemo 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 (3.4.1) 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 3.4.5 dopustiv i neka je x bazično nedopustivo rešenje sa q negativnih koordinata. Tada postoji $T_{ij} < 0$. Takodje, u sledeća dva slučaja:*

a) $q = m$,

b) $q < m$ *and*

$$Neg = \left\{ \frac{b_k^*}{T_{kj}} \mid b_k^* < 0, T_{kj} < 0, k = 1, \ldots, m \right\},$$
$$Pos = \left\{ \frac{b_k^*}{T_{kj}} \mid b_k^* > 0, T_{kj} > 0, k = 1, \ldots, m \right\},$$
$$\min(Neg \cup Pos) = \frac{b_r^*}{T_{rj}} \in Neg$$

moguće je naći novo bazično rešenje sa najviše $q - 1$ *negativnih koordinata, ako odaberemo* T_{rj} *za pivot elemenat.*

Dokaz. Prvi deo teoreme koji je vezan za postojanje elementa $T_{ij} < 0$ je ekvivalentan lemi 2.6.2.

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 (3.4.1). 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 (3.4.1) 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 11 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 [29]. Tako je nastao program MarPlex koji, kako ćemo videti, vrlo uspešno rešava široku klasu problema linearnog programiranja.

Ulazni podaci se zadaju obliku MPS fajla. Ovaj tip fajla predstavlja svetski standard za zadavanje problema linearnog programiranja u vidu simpleks matrica (tablica). Format je dat sledećom "tabelom":

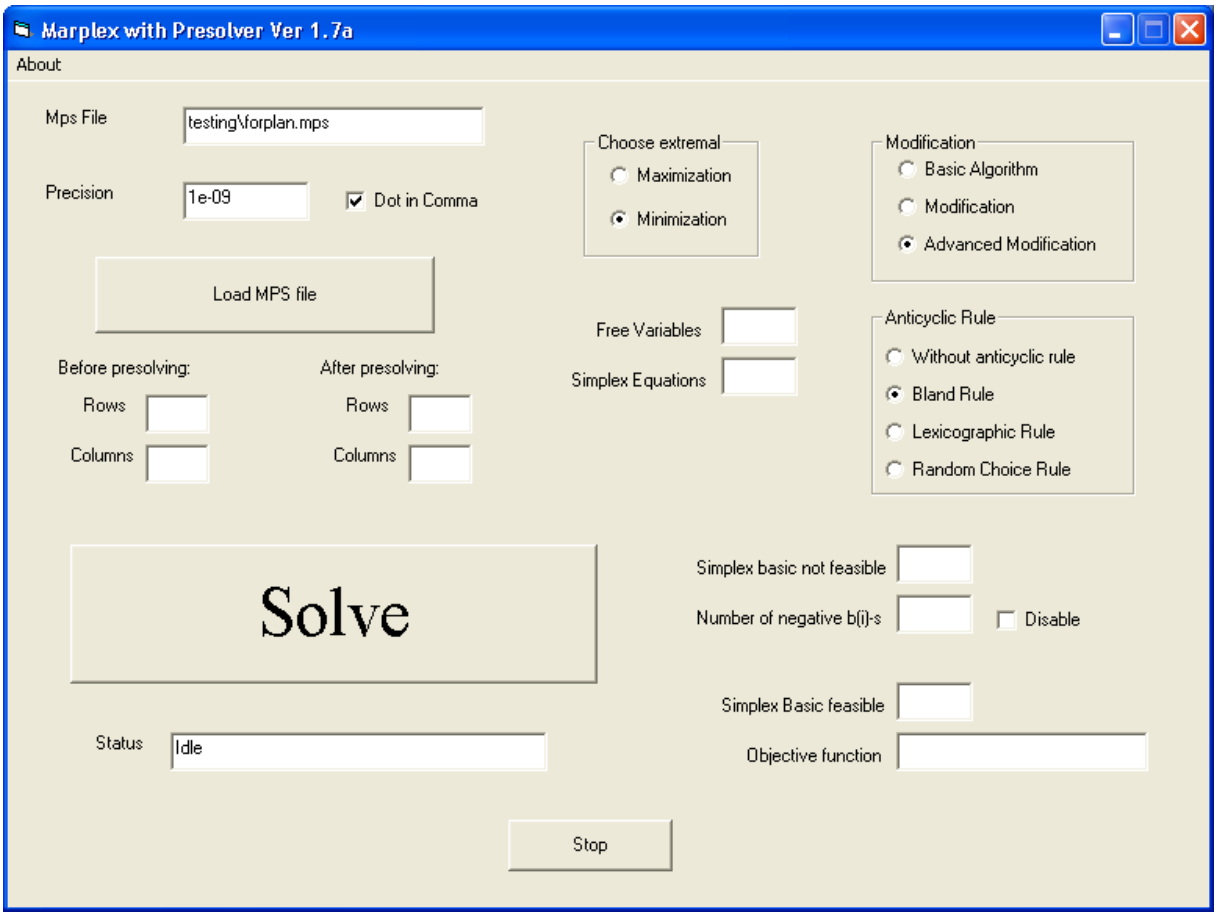
```
------------------------------------------------------------------
Field:        1          2          3        4        5        6
Columns:    2-3       5-12      15-22    25-36    40-47    50-61
          NAME   problem name
          ROWS
           type     name
          COLUMNS
                  column      row      value    row      value
                  name        name             name
          RHS
                   rhs        row      value    row      value
                   name       name             name
          RANGES
```

```
                range       row       value     row       value
                name        name                name
          BOUNDS

           type     bound       column    value
                    name        name
          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 [33]. MarPlex je besplatan program i dostupan je na internetu na:
`tesla.pmf.ni.ac.yu/people/pecko/linprog/marplex.rar`

Interface MarPlex-a je prvenstveno prilagodjen korisniku i omogućava udoban rad i za razliku od drugih sličnih programa ne opterećuje korisnika brojnim opcijama koje se mahom ne koriste.



Slika 3.5.1. Interface programa MarPlex

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. Medjutim, ukoliko je ova vrednost i suviše 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.

- **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. Ponudjene 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 [8] (gde je opisan presolver koji koristi program PCx. Presolver koji smo implementirali u MarPlex-u koristi modifikovanu verziju onog iz [8]. 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 [8]. 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.	Sum	Obj. 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	
blend	1	732	733	−30.769485006264	−3.08121498e+001
	1	732	733	−30.769485006264	
	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	

Name	Bf.	Sim.	Sum	Objective value	PCx
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	
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	

Name	Bf.	Sim.	Sum	Objective value	PCx
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	
	100	374	474	1793324.53797035	
ship04s	14	172	186	1798714.70044539	1.79871471e+006
	116	185	301	1798714.70044539	
	57	194	251	1798714.70044539	
ship08l	320	530	850	1909055.21138913	1.90905521e+006
	461	364	825	1909055.21138913	
	144	631	775	1909055.21138913	
ship08s	54	258	312	1920098.21053462	1.92009821e+006
	169	239	408	1920098.21053462	
	67	272	339	1920098.21053462	
ship12l	49	1019	1068	1470187.91932926	1.47018797e+006
	938	1908	2846	1470187.91932926	
	232	1711	1943	1470187.91932926	
ship12s	55	439	494	1489236.13440613	1.48923613e+006
	429	556	985	1489236.13440613	
	166	486	632	1489236.13440613	
sierra	75	326	401	15394362.1836319	1.53943622e+007
	65	325	490	15381546.3836319	
	82	310	392	15394362.1836319	
stair	2450	44	2494	−251.26695098074	−2.51266951e+002
	686	33	719	−251.266951192317	
	11066	168	11234	–	
standata	21	138	159	1257.6995	1.25769951e+003
	76	98	174	1257.6995	
	146	116	262	1257.69949999999	
standmps	131	109	240	1406.0175	1.40601750e+003
	260	155	415	1406.01749999996	
	752	72	824	1406.0175	
stocfor1	1	17	18	−41131.9762194364	−4.11319762e+004
	1	17	18	−41131.9762194364	
	1	17	18	−41131.9762194364	
vtp.base	69	55	124	129831.462637412	1.29831463e+005
	179	71	250	129831.462461362	
	430	47	477	129831.464051472	

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 takodje da su modifikacija i poboljšana modifikacija skoro izjednačene. Upravo ova činjenica opravdava razmatranje originalne verzije modifikacije iz našeg rada [41] u ovom radu, a i takodje dodatno zakomplikuje situaciju oko izbora algoritma za odredjivanje početnog bazično dopustivog rešenja. Medjutim, 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`, [53]). 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. Medjutim, 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. Takodje, MarPlex koristi poboljšanje algoritma **Replace**, tj koristi sparse strukturu Takerove 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

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 [1, 25]. 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 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

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 $(10^{15}, 10^{20})$.

4. Višekriterijumska optimizacija

Proces istovremene optimizacije više ciljnih funkcija naziva se ***višekriterijumska optimizacija*** (VKO) ili ***vektorska optimizacija***. U ovoj glavi ćemo proučiti nekoliko metoda za rešavanje problema VKO. Takodje, pokazaćemo kako se u specijalnim slučajevima, ovaj problem svodi na problem nalaženja generalisanih inverza.

4.1 Definicija i osnovna svojstva

Teorija odlučivanja i višekriterijumska optimizacija imaju značajnu ulogu u mnogim oblastima nauke i tehnike, i uopšte u životu. Većina praktičnih problema teorije odlučivanja mogu da se formulišu kao problemi odlučivanja po više kriterijuma, tj. kao problemi višekriterijumske optimizacije. Primeri takvih modela su: izbor portfolia, problemi planiranja proizvodnje, dizajn oblika kao i u mnogim inženjerskim problemima [39, 52]. Mnogi fenomeni i pojave mogu biti verno opisane pomoću modela nelinearne višekriterijumske optimizacije. Zato je razvoj metoda za rešavanje tih problema veoma značajan [26, 27].

Definicija 4.1.1 *Problem višekriterijumske optimizacije može na sledeći način da se definiše na sledeći način:*

$$\begin{gathered} \max\left[f_1(x), \ldots, f_l(x)\right] \\ x \in \Omega_P \end{gathered} \tag{4.1.1}$$

gde su $f_i : R^n \Rightarrow R$ *funkcije cilja a* $\Omega_P \subseteq \mathbb{R}^n$ *skup dopustivih rešenja za problem višekriterijumske optimizacije definisan skupovnom jednakošću:*

$$\Omega_P = \{x \in \mathbb{R}^n \mid g_i(x) \leq 0,\ i = 1, ..., m;\ x_j \geq 0,\ j = 1, ..., n\}.$$

Funkcije $g_i(x)$ *se nazivaju funkcijama ograničenja.*

I ovde se radi jednostavnosti razmatraju samo problemi maksimizacije, jer se problem minimizacije jednostavno prevodi u problem maksimizacije množenjem kriterijumske funkcije sa -1.

Kao i kod linearnog programiranja, i ovde elemente skupa Ω_P nazivamo dopustivim rešenjima. Lako se dokazuje i sledeća teorema:

Teorema 4.1.1 *Ako su sve funkcije* $g_i(x)$ *konveksne, tada je i skup* Ω_P *konveksan*

Svakom dopustivom rešenju $x \in \Omega_P$ odgovara skup vrednosti kriterijumskih funkcija, koje su grupisane u vektoru $f(x) = (f_1(x), f_2(x), ..., f_l(x))$. Na taj način se skup dopustivih rešenja preslikava u *kriterijumski skup* $S = \{f(x) \mid x \in \Omega_P\}$.

U nastavku je dato nekoliko osnovnih definicija pojmova koji su značajni u višekriterijumskoj optimizaciji.

Definicija 4.1.2 *Marginalna rešenja zadatka VKO se odredjuju optimizacijom svake od funkcija cilja pojedinačno nad zadatim dopustivim skupom, tj. rešavanjem l jednokriterijumskih zadataka:*

$$\begin{array}{ll} (\max) & f_k(x), \\ \text{p.o.} & x \in \Omega_P, \end{array}$$

za svako $k = 1, \ldots, l$.

Marginalna rešenja ćemo obeležavati sa $x^{(k)} = (x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)})$, tj. $x^{(k)}$ je optimalno rešenje dobijeno optimizacijom $k-$te funkcije cilja nad zadatim dopustivim skupom Ω_P.

Definicija 4.1.3 *Idealne vrednosti za funkcije cilja f_k definišu se kao vrednosti funkcija cilja za marginalna rešenja: $f_k^* = f_k(x^{(k)}),\ k = 1, ..., l$.*

Idealne vrednosti funkcija cilja odredjuju idealnu tačku u kriterijumskom prostoru, tj. idealnu vrednost funkcije $f^* = (f_1^*, f_2^*, ..., f_l^*)$.

Definicija 4.1.4 *Savršeno rešenje je takvo rešenje koje istovremeno maksimizira sve funkcije cilja, tj. $x^* = \{x \mid f_k(x) = f_k^*,\ k = 1, ..., l\}$.*

U najvećem broju slučajeva marginalna rešenja se razlikuju i savršeno rešenje ne postoji. Zbog toga se pojavila potreba da se uvedu novi koncepti optimalnosti. Najvažniji medju njima je koncept ***Pareto optimalnosti***.

Definicija 4.1.5 *Dopustivo rešenje x^* predstavlja Pareto optimum problema VKO ako ne postoji neko drugo dopustivo rešenje x takvo da važi:*

$$f_k(x) \geq f_k(x^*), \forall k = 1, ..., l$$

pri čemu bar jedna od nejednakosti prelazi u strogu nejednakost:

$$f_{k_i}(x) > f_{k_i}(x^*), \forall i = 1, ..., m,\ m < l. \tag{4.1.2}$$

Definicija 4.1.6 *Dopustivo rešenje x^* je slabi Pareto optimum ako ne postoji neko drugo dopustivo rešenje x tako da važi:*

$$f_k(x) > f_k(x^*), \forall k = 1, ..., l.$$

Ukoliko ne postoji savršeno rešenje, problem višekriterijumske optimizacije nije potpuno definisan. Tada moramo iz skupa "jednako dobrih" rešenja izabrati ono koje je u konkretnom slučaju najbolje. Taj izbor vrši donosilac odluke (DO). Najčešće se problem višekriterijumske optimizacije svodi na matematički dobro definisane probleme jednokriterijumske optimizacije. Na taj način se formira takozvani ***skalarni problem*** (ili više njih) čije se rešenje proglašava optimalnim i za problem višekriterijumske optimizacije. Većina klasičnih metoda višekriterijumske optimizacije [15] koji će biti u nastavku izloženi koriste ovaj pristup. Takodje ovaj pristup se sreće i kod novijih metoda (npr. [31]).

U nastavku ćemo izložiti nekoliko metoda za rešavanje problema višekriterijumske optimizacije. Za svaku metodu navešćemo odgovarajuće skalarne probleme i dokazati pod

odredjenim uslovima Pareto optimalnost dobijenih rešenja. Za svaki metod je data implementacija u programskom jeziku MATHEMATICA. Sve date implementacije se baziraju na našem radu [43].

U implementacijama ćemo koristiti sledeće funkcije programskog jezika MATHEMATICA koje rešavaju probleme jednokriterijumske optimizacije.

Funkcija `Maximize[f, constr, var]` nalazi maksimum ciljne funkcije `f` pri ograničenjima `constr`, a funkcija `Minimize[f, constr, var]` minimum funkcije `f` pri ograničenjima `constr`. Ciljna funkcija može biti i linearna i nelinearna.

Funkcija `LinearProgramming[c, A, b]` nalazi vektor x koji daje minimum funkcije $c^T x$ po ograničenjima $Ax \geq b$ i $x \geq 0$. Modifikacija `LinearProgramming`$[\mathtt{c}, \mathtt{A}, \{\{\mathtt{b}_1, \mathtt{s}_1\}, \{\mathtt{b}_2, \mathtt{s}_2\}, ...\}]$ iste funkcije nalazi vektor x koji minimizira funkciju $c^T x$, uz uslov $x \geq 0$ i prema ograničenjima koja zavise od matrice A i uredjenih parova b_i, s_i. Za svaki red a_i matrice Aa, odgovarajuće ograničenje je $A_{i\bullet}x \geq b_i$ za $s_i = 1$, ili $A_{i\bullet}x = b_i$ za $s_i = 0$, ili $A_{i\bullet}x \leq b_i$ za $s_i = -1$.

4.2 Metoda težinskih koeficijenata

Metoda težinskih koeficijenata koristi funkciju korisnosti donosioca odluka da bi omogućila svodjenje problema VKO na problem jednokriterijumske optimizacije. Donosilac odluka zadaje nenegativne težinske koeficijente w_i za svaku od ciljnih funkcija $f_i(x), i = 1, ..., l$. Formira se nov jednokriterijumski problem:

$$\begin{aligned} \max \quad & f^M(x) = \sum_{k=1}^{l} w_k f_k^o(x) \\ \text{p.o.} \quad & x \in \Omega_P, \end{aligned} \tag{4.2.3}$$

gde je $w_k \geq 0$ i f_k^o je normalizovana k-ta funkcija cilja $f_k(x), k = 1, \ldots, l$.

Ako su ciljne funkcije linearne i date u obliku: $f_k(x) = \sum_{i=1}^{n} a_{ki} x_i$, normalizovane ciljne funkcije su definisane izrazima

$$f_k^o(x) = \frac{f_k(x)}{S_k} = \frac{a_{k1}}{S_k} x_1 + \frac{a_{k2}}{S_k} x_2 + \ldots + \frac{a_{kn}}{S_k} x_n,$$

u kojima su S_k zadati realni brojevi. Oni se mogu zadati na sledeći način: $S_k = \sum_{j=1}^{n} |a_{kj}| \neq 0$.

Često, u praktičnim problemima, funkcije cilja se izražavaju u različitim mernim jedinicama (npr. f_1 se izražava u kilogramima, f_2 u sekundama, itd), pa da bi težine w_i bile neimenovane veličine vrši se normalizacija funkcija cilja f_i.

U sledećoj lemi dajemo praktičan kriterijum za detekciju podskupa pareto optimalnih rešenja.

Lema 4.2.1 *Rešenje problema višekriterijumske optimizacije dobijeno primenom metode težinskih koeficijenata je pareto optimalno ukoliko važe uslovi $S_k > 0$ i $w_k > 0$ za svako $k \in \{1, \ldots, l\}$.*

Dokaz. Označimo sa x^* rešenje dobijeno maksimizacijom funkcije $f^M(x) = \sum_{k=1}^{l} w_k f_k^o(x)$. Primetimo da važi: $f^M(x^*) \geq f^M(x), \forall x \in \Omega_P$. Dalje važi:

$$\begin{aligned} & \sum_{k=1}^{l} w_k f_k^o(x^*) \geq \sum_{k=1}^{l} w_k f_k^o(x), \forall x \in \Omega_P \\ \Leftrightarrow \quad & \sum_{k=1}^{l} w_k (f_k^o(x^*) - f_k^o(x)) \geq 0, \forall x \in \Omega_P \\ \Leftrightarrow \quad & \sum_{k=1}^{l} \tfrac{w_k}{S_k} (f_k(x^*) - f_k(x)) \geq 0, \forall x \in \Omega_P \end{aligned} \tag{4.2.4}$$

Pretpostavimo da rešenje x^* problema (2.1) nije pareto optimalno. Tada postoji neko dopustivo rešenje x' tako da važi: $f_k(x') \geq f_k(x^*)$, odnosno $f_k(x^*) - f_k(x') \leq 0$ za svako $k \in \{1, \ldots, l\}$, pri čemu postoji bar jedno k_i za koje nejednakost prelazi u strogu nejednakost. Odatle dobijamo

$$\sum_{k=1}^{l} (f_k(x^*) - f_k(x')) < 0.$$

S obzirom na pretpostavku da su sve veličine S_k i w_k pozitivne, važi

$$\sum_{k=1}^{l} \frac{w_k}{S_k} (f_k(x^*) - f_k(x')) < 0.$$

odnosno $f^M(x^*) < f^M(x')$, što je kontradikcija jer je x^* maksimalna vrednost funkcije $f^M(x)$ pod datim ograničenjima. □

Implementacija metode težinskih koeficijenata predstavlja funkciju `MTK[f_List, g_List, w1_List]` čiji su parametri:

- `f_List`: Lista ciljnih funkcija.
- `g_List`: Lista ograničenja.
- `w1_List`: Lista težinskih koeficijenata.

Poslednji parametar funkcije `MTK` može biti prazna lista. U tom slučaju, od korisnika se traži da unese prirodan broj k i zatim se generišu sve n-torke celobrojnih nenegativnih težinskih koeficijenata čiji je zbir jednak k. To se postiže pomoću funkcije `Compositions[k,n]` iz paketa `Combinatorica`. Sledi kod funkcije `MTK`:

```
MTK[f_List, g_List, w1_List] :=
Module[{l = Length[f], fun = 0, w = {}, s = {}},
    var = Variables[f];
    If[w1 == {},
      k = Input["Koliko puta?"]; w = Compositions[k,l],
      k = Length[w1]-1; w = w1;
    ];
```

```
    For[i = 1, i <= l, i++,
        cfs = Coefficient[f[[i]], var];
        AppendTo[s, Sum[cfs[[j]], {j, Length[cfs]}]];
    ];
    For[i = 1, i <= k + 1, i++,
        fun = Simplify[Sum[w[[i, j]]*f[[j]]/s[[j]], {j, l}]];
        res = First[Rest[Maximize[fun, g, var]]];
        Print[fun, {ReplaceAll[f, res], res}];
    ];
]
```

U lokalnoj promenljivoj *res* čuvaju se trenutni rezultati, a promenljiva fun predstavlja funkciju $f^M(x)$ koju treba optimizirati. Lokalna promenljiva *var* je lista promenljivih.

Primer 4.2.1 Rešiti problem VKO:

$$\begin{array}{ll} (\max) & [40x + 10y, x + y] \\ \text{p.o.} & 2x + y \leq 6 \\ & x + y \leq 5 \\ & x \leq 2 \\ & x, y \geq 0 \end{array}$$

Odaberimo metod konstrukcije koeficijenata w_i primenom funkcije *Compositions*:

```
MTK[{40x + 10y, x + y},
{2x + y<= 6, x + y<= 5, x<= 2, x>= 0, y>= 0}, {}]
```

Za slučaj $k = 5$ imamo $w = \{\{0,5\},\{1,4\},\{2,3\},\{3,2\},\{4,1\},\{5,0\}\}$. Potrebno je rešiti šest problema linearnog programiranja, za svaku podlistu liste w.

Program prvo ispisuje listu problema linearnog programiranja koje rešava a onda odgovarajuća rešenja.

$$\begin{array}{l} (\max)\ \dfrac{5(x+y)}{2},\ \{\{80,5\},\{x \to 1, y \to 4\}\} \\ (\max)\ \dfrac{14x+11y}{5},\ \{\{80,5\},\{x \to 1, y \to 4\}\} \\ (\max)\ \dfrac{31x+19y}{10},\ \{\{80,5\},\{x \to 1, y \to 4\}\} \\ (\max)\ \dfrac{17x+8y}{5},\ \{\{100,4\},\{x \to 2, y \to 2\}\} \\ (\max)\ \dfrac{37x+13y}{10},\ \{\{100,4\},\{x \to 2, y \to 2\}\} \\ (\max)\ 4x + y,\ \{\{100,4\},\{x \to 2, y \to 2\}\} \end{array}$$

Uzimajući za težinske koeficijente $w_1 = 1$ i $w_2 = 4$, rešavamo problem maksimizacije funkcije $\frac{14x+11y}{5}$ na skupu ograničenja $2x + y <= 6, x + y <= 5, x <= 2, x >= 0, y >= 0$.

Dobijamo rešenje $x = 1$ i $y = 4$. U toj tački prva funkcija cilja ima vrednost 80, a druga vrednost 5. Rešenje $\{x = 1, y = 4\}$ je Pareto optimalno, jer ispunjava uslove Leme 1.

Ukoliko težinski koeficijenti imaju vrednost $w_1 = 3$ i $w_2 = 2$, tada maksimizacijom funkcije $\frac{17x+8y}{5}$ dobijamo rešenje $\{x = 2, y = 2\}$, koje je takodje pareto optimalno.

Sledeći primer pokazuje da rešenje koje daje metoda težinskih koeficijenata ne mora biti Pareto optimalno ukoliko nisu zadovoljeni uslovi leme 4.2.1.

Primer 4.2.2 Rešiti problem VKO:

$$\begin{array}{ll} (\max) & [8x + 12y, 14x + 10y, x + y] \\ \text{p.o.} & 2x + y \leq 150 \\ & 2x + 3y \leq 300 \\ & 4x + 3y \leq 360 \\ & x + 2y \leq 120 \\ & x, y \geq 0 \end{array}$$

Koristimo vrednosti koeficijenata w_i koje determiniše korisnik:

```
MTK[{8x + 12y, 14x + 10y, x + y},
{2x + y<= 150, 2x + 3y<= 300, 4x + 3y<= 360, x + y<= 120, x>= 0, y>= 0},
{{1, 0, 0}, {0.9, 0.3, 0.1}, {2, 0.5, 1}}]
```

Za svaku listu koeficijenata rešava se odgovarajući zadatak jednokriterijumske optimizacije:

$$\begin{array}{l} (\max)\ \frac{1}{5}(2x + 3y),\ \{\{1200, 1000, 100\}, \{x \to 0, y \to 100\}\} \\ (\max)\ 0.585x + 0.715y,\ \{\{1200, 1220, 110\}, \{x \to 30, y \to 80\}\} \\ (\max)\ 1.59167x + 1.90833y,\ \{\{1200, 1220, 110\}, \{x \to 30, y \to 80\}\} \end{array}$$

Rešenje problema je $\{\{1200, 1220, 110\}, \{x \to 30, y \to 80\}\}$.

Medjutim, rešenje $\{\{1200, 1000, 100\}, \{x \to 0, y \to 100\}\}$ nije pareto optimalno i ne zadovoljava uslove Leme 4.2.1.

4.3 Leksikografska metoda

U leksikografskoj metodi DO zadaje prioritete kriterijumskim funkcijama u vidu strogo definisanog redosleda značajnosti funkcija. Pretpostavićemo da su kriterijumi već poredjani i indeksirani tako da kriterijum f_1 ima najviši prioritet, f_2 sledeći niži, itd. sve do kriterijuma f_l koji ima najniži prioritet. Rešenje problema višekriterijumske optimizacije dobijamo tako što redom rešavamo sledećih l problema jednokriterijumske optimizacije.

$$\begin{array}{ll} (\max) & f_k(x) \\ \text{p.o.} & f_i(x) = f_i(x^{(i)}),\ i = 1, \ldots,\ k - 1,\ k \geq 2, \\ & x \in \Omega_P, \end{array}$$

za $k = 1, \ldots, l$. Pritom su $x^{(i)}$, $i = 1, \ldots, k-1$, respektivno, rešenja problema na nivoima $i = 1, \ldots, k-1$, $k \geq 2$. Rešenje $x^{(l)}$ uzima se kao konačno rešenje problema VKO. Kod leksikografske metode posle svakog koraka smanjuje se dimenzija oblasti dopustivih rešenja.

Lema 4.3.1 *Rešenje $x^{(l)}$ koje daje leksikografska metoda je pareto optimalno.*

Dokaz. Primetimo da važi $f_k(x^{(l)}) = f_k(x^{(k)})$ za svako $k = 1, \ldots, l-1$. Pretpostavimo da rešenje $x^{(l)}$ nije pareto optimalno. Tada postoji neko dopustivo rešenje x^* tako da važi

$$f_k(x^*) \geq f_k(x^{(l)}), \forall k = 1, \ldots, l$$

gde makar jedna nejednakost prelazi u strogu nejednakost. Neka recimo, za indeks t važi $f_t(x^*) > f_t(x^{(l)})$. S obzirom da je $f_t(x^{(l)}) = f_t(x^{(t)})$, važi nejednakost $f_t(x^*) > f_t(x^{(t)})$, pa sledi $x^* \notin \Omega_{Pt}$, gde je

$$\Omega_{Pt} = \Omega_P \cap \{x \in \mathbb{R}^n \mid f_1(x) \geq f_1(x^{(1)}), \ldots, f_{t-1}(x) \geq f_{t-1}(x^{(t-1)})\}.$$

Dalje sledi da $x^* \in \Omega_P \setminus \Omega_{Pt}$, pa odatle sledi da za neko $i \in \{1, \ldots, k-1\}$ važi $f_i(x^*) < f_i(x^{(i)}) = f_i(x^{(l)})$. Kontradikcija. Ovim je dokaz završen. □

Implementacija leksikografske metode predstavlja funkcija `Leksik[f_List, g_List]`. Značenje parametara ove funkcije je isto kao i kod funkcije `MTK`.

Kada je nadjeno optimalno rešenje f_k^* nivoa k, izraz $f_k(x) \geq f_k^*$ dodaje se na kraj liste uslova korišćenjem funkcije `AppendTo`. Tada se funkcija $f_{k+1}(x)$ uzima za novu ciljnu funkciju.

```
Leksik[f_List, g_List] :=
  Module[{l = Length[f], res = {0}, h = g, r},
    var = Variables[f];
    For[i = 1, i <= l, i++,
        rez = Maximize[f[[i]], h, var];
        AppendTo[h, f[[i]] >= First[rez]];
        Print[rez, h];
        If[First[rez] > First[res], res = rez];
    ];
    r = First[Rest[res]];
    Return[{ReplaceAll[f, r], r}];
]
```

Primer 4.3.1 U ovom primeru se rešava problem višekriterijumske optimizacije iz prethodnog primera leksikografskom metodom.

Za rešavanje unosimo sledeću naredbu:

```
Leksik[{8x + 12y, 14x + 10y, x + y},
{2x + y<= 150, 2x + 3y<= 300, 4x + 3y<= 360, x + 2y>= 120, x>= 0, y>= 0}]
```

Rešavaju se tri problema jednokriterijumske optimizacije.
Za $k = 1$ optimalno rešenje funkcije f_1 je

$$\{1200, \{x \to 0,\ y \to 100\}\},$$

dok su nova ograničenja jednaka

$$\{2x + y <= 150,\ 2x + 3y <= 300,\ 4x + 3y <= 360,\ x + 2y >= 120,\\ x >= 0,\ y >= 0,\ 8x + 12y >= 1200\}.$$

Za $k = 2$ optimalno rešenje funkcije f_2 je

$$\{1220, \{x \to 30, y \to 80\}\}$$

a novi skup ograničenja je

$$\{2x + y <= 150,\ 2x + 3y <= 300,\ 4x + 3y <= 360,\ x + 2y >= 120,\\ x >= 0,\ y >= 0,\ 8x + 12y >= 1200,\ 14x + 10y >= 1220\}.$$

Za $k = 3$ optimalno rešenje funkcije f_3 je

$$\{110, \{x \to 30, y \to 80\}\}$$

dok je novi skup ograničenja jednak

$$\{2x + y <= 150,\ 2x + 3y <= 300,\ 4x + 3y <= 360,\ x + 2y >= 120,\\ x >= 0,\ y >= 0,\ 8x + 12y >= 1200,\ 14x + 10y >= 1220,\ x + y >= 110\}$$

Rešenje problema je:

$$\{\{1200, 1220, 110\}, \{x \to 30, y \to 80\}\}.$$

4.4 Relaksirana leksikografska metoda

Ova metoda je modifikacija leksikografske metode. Razlika je u sledećem: u relaksiranoj leksikografskoj metodi se svakom kriterijumu, sem poslednjeg, dodeljuje realni parametar $\alpha_k \geq 0, k = 1, ..., l-1$, koja pokazuje dozvoljeno odstupanje od svoje optimalne vrednosti. Preciznije, metod sadrži l koraka, za svako $k = 1, ..., l$:

$$\begin{array}{ll} (\max) & f_k(x) \\ \text{p.o.} & f_i(x) \geq f(x^{(i)}) - \alpha_i,\ i = 1, \ldots, k-1,\ k \geq 2, \\ & x \in \Omega_P, x \geq 0, \end{array} \tag{4.4.5}$$

gde su $f_1^*, ..., f_{k-1}^*$ optimalne vrednosti u predhodnim koracima (kao i u leksikografskoj metodi).

Lema 4.4.1 *Rešenje dobijeno relaksiranom leksikografskom metodom je u opštem slučaju slabo Pareto optimalno.*

Dokaz. Neka je $x^{(l)}$ rešenje dobijeno relaksiranom leksikografskom metodom. Pretpostavimo suprotno, tj. da postoji neki vektor $x' \in \mathbb{R}^n$ takav da je $f_k(x') > f_k(x^*)$. Neka je Ω_{Pk} skup dopustivih rešenja k-tog problema (4.4.6). Pošto je $x^* \in \Omega_{Pl}$, znači da je $f_k(x') > f_k(x^*) > f_k(x^{(k)}) - \alpha_k$, odnosno $x' \in \Omega_{Pl}$. To znači da je $f_l(x^*) \geq f_l(x')$ što je kontradikcija. □

Funkcija `Relax[f_List, g_List, a_List]` implementira ovu metodu. Kao ulazni parametar ovde se pojavljuje i lista `a`. To je lista koeficijenata α_k. Kada je u implementaciji nadjeno optimalno rešenje f_k^* nivoa k, izraz $f_k(x) \geq f_k^* - \alpha_k$ dodaje se na kraj liste uslova, kao kod leksikografske metode. Pritom sa $x^{(i)},\ i = 1, \ldots,\ k-1$, respektivno, označavamo rešenja problema na nivoima $i = 1, \ldots,\ k-1,\ k \geq 2$. Rešenje $x^{(l)}$ uzima se kao konačno rešenje problema VKO.

```
Relax[f_List, g_List, a_List] :=
  Module[{l = Length[f], res = {}, h = g},
      var = Variables[f];
      For[i = 1, i < l, i++,
          rez = Maximize[f[[i]], h, var];
          AppendTo[h, f[[i]] >= First[rez] - a[[i]]];
          Print[rez, h];
      ];
      res = Maximize[f[[l]], h, var];
      r = First[Rest[res]];
      Return[{ReplaceAll[f, r], r}];
  ]
```

Primer 4.4.1 Rešiti problem VKO:

$$\begin{array}{ll} (\max) & [x+4y, x, x+y] \\ \text{p.o.} & 2x+y \leq 6 \\ & x+3y \leq 6 \\ & x, y \geq 0, \end{array}$$

ako je zadato $\alpha_1 = 1, \alpha_2 = 1$.

Problem rešavamo izrazom:

```
Relax[{x + 4y, x, x + y}, {2x + y<= 6, x + 3y<= 6, x>= 0, y>= 0}, {1, 1}]
```

U prvom koraku rešavamo problem:

$$\begin{array}{ll} (\max) & x+4y \\ \text{p.o.} & \{2x+y \leq 6, x+3y \leq 6, x \geq 0, y \geq 0\} \end{array}$$

Problem ima jedinstveno rešenje: $\{8, \{x \to 0, y \to 2\}\}$.

U drugom koraku rešava se problem:

$$\begin{array}{ll} (\max) & x \\ \text{p.o.} & \{2x+y \leq 6, x+3y \leq 6, x \geq 0, y \geq 0, x+4y \geq 7\} \end{array}$$

Optimalno rešenje problema je $\{\frac{17}{7}, \{x \to \frac{17}{7}, y \to \frac{8}{7}\}\}$.

Na kraju, poslednji problem koji se rešava je:

$$\begin{array}{ll} (\max) & x + y \\ \text{p.o.} & \{2x + y \leq 6, x + 3y \leq 6, x \geq 0, y \geq 0, x + 4y \geq 7, x \geq \dfrac{10}{7}\} \end{array}$$

Rešenje ovog problema jednokriterijumske optimizacije je $\{\frac{18}{5}, \{x \to \frac{12}{5}, y \to \frac{6}{5}\}\}$.

Konačno rešenje je jednako:

$$\{\{\frac{36}{5}, \frac{12}{5}, \frac{18}{5}\}, \{x \to \frac{12}{5}, y \to \frac{6}{5}\}\}$$

4.5 Metoda ε ograničenja

U ovoj metodi donosilac odluka izdvaja kriterijum $f_q(x)$ koji ima najviši prioritet i koji treba maksimizirati, dok ostale funkcije cilja ne smeju imati vrednosti manje od unapred zadatih realnih parametara $\varepsilon_k, k = 1, ..., l, k \neq q$. Rešava se sledeći jednokriterijumski zadatak:

$$\begin{array}{ll} (\max) & f_q(x) \\ \text{p.o.} & g_i(x) \leq 0, \quad i = 1, ..., m \\ & f_k(x) \geq \varepsilon_k, \quad k = 1, ..., l, \ k \neq q \\ & x_j \geq 0, \quad j = 1, ..., n \end{array}$$

čije se rešenje usvaja kao rešenje polaznog zadatka VKO. Ako postavljeni zadatak nema dopustivo rešenje, potrebno je smanjiti vrednosti ε_k.

Vrlo slično kao u prethodnom odeljku i ovde dokazujemo da je rešenje slabo Pareto optimalno.

Lema 4.5.1 *Rešenje x^* dobijeno metodom ε ograničenja je slabo Pareto optimalno.*

Implementacija ovog metoda data je u sledećoj funkciji `Epsilon[f_List, g_List, e_List, q_]`. Dodatni ulazni parametri su:

- `e_List`: lista parametara ε_k.
- `q_`: redni broj ciljne funkcije najvišeg prioriteta.

Sledi kod funkcije `Epsilon`.

```
Epsilon[f_List, g_List, e_List, q_] :=
  Module[{l = Length[f], h = g},
    var = Variables[f];
    For[i = 1, i <= l, i++,
        If[i != q, AppendTo[h, f[[i]] >= e[[i]]]]
    ];
    res = Maximize[f[[q]], h, var];
```

```
    r = First[Rest[res]];
    Return[{ReplaceAll[f, r], r}];
]
```

Ako za primer (4.2.2) odredimo da je $\varepsilon_1 = 1000, \varepsilon_2 = 1000$ i da treća funkcija ima najviši prioritet, imamo sledeći izraz:

```
Epsilon[{8x + 12y, 14x + 10y, x + y}, {2x + y<= 150, 2x + 3y<= 300,
4x + 3y<= 360, x + 2y>= 120, x>= 0, y>= 0}, {1000, 1000, 0}, 3]
```

Neophodno je rešiti sledeći problem jednokriterijumske optimizacije:

$$\begin{aligned} (\max) \quad & x + y \\ \text{p.o.} \quad & 2x + y \leq 150 \\ & 2x + 3y \leq 300 \\ & 4x + 3y \leq 360 \\ & x + 2y \leq 120 \\ & x, y \geq 0 \\ & 8x + 12y \geq 1000 \\ & 14x + 10y \geq 1000 \end{aligned}$$

Optimalno rešenje ovog problema se uzima kao rešenje polaznog problema:

$$\{\{1200, 1220, 110\}, \{x \to 30, y \to 80\}\}$$

4.6 Metode rastojanja

Metode rastojanja čine grupu metoda za rešavanje zadataka VKO čija je osnovna ideja da se u kriterijumskom prostoru traži tačka koja je najbliža nekoj unapred odredjenoj tački koja se želi dostići ili ka kojoj treba težiti ako ona nije dopustiva. Drugim rečima, minimizira se rastojanje izmedju željene tačke i dopustive oblasti.

Ovde nudimo dva načina za odredjivanje željene tačke $f^z = (f_1^z, ..., f_l^z)$. U prvom slučaju, DO za svaki od kriterijuma f_i^z zadaje željenu vrednost. U drugom slučaju, za f^z se može uzeti tačka koja za koordinate ima optimalne vrednosti kriterijumskih funkcija redom u dozvoljenoj oblasti, odnosno marginalna rešenja. Na taj način se u l - dimenzionalnom kriterijumskom prostoru definiše tačka f^z koja ne mora da pripada dopustivoj oblasti Ω_P. U slučaju da je $f^z \in \Omega_P$ zadatak se rešava rešavanjem sistema jednačina koje su definisane skupom ograničenja.

U metodi rastojanja rešava se sledeći zadatak jednokriterijumske optimizacije:

$$\begin{aligned} (\min) \quad & d(f^z, f(x)) \\ \text{p.o.} \quad & g_i(x) \leq 0, \quad i = 1, ..., m \end{aligned}$$

gde $d(f^z, f(x))$ označava rastojanje definisano pogodnom metrikom.

Najčešće se u ovoj metodi koriste sledeće tri metrike:

$$l_1 \text{ (Pravougaona) metrika:} \quad d_{l_1}(f^z, f(x)) = \sum_{k=1}^{l} |f_k^z - f_k(x)|$$

$$l_2 \text{ (Euklidova) metrika:} \quad d_{l_2}(f^z, f(x)) = \sqrt{\sum_{k=1}^{l} (f_k^z - f_k(x))^2}$$

$$l_\infty \text{ (Čebiševljeva) metrika:} \quad d_{l_\infty}(f^z, f(x)) = \max_{1 \le k \le l} |f_k^z - f_k(x)|$$

Kriterijumima je moguće dodeliti težinske koeficijente, tako da prethodne formule za rastojanje izmedju željenog i traženog cilja dobijaju oblik:

$$d_{l_1}(f^z, f(x)) = \sum_{k=1}^{l} w_k |f_k^z - f_k(x)|, \quad \text{za pravougaonu metriku,}$$

$$d_{l_2}(f^z, f(x)) = \sqrt{\sum_{k=1}^{l} w_k (f_k^z - f_k(x))^2}, \quad \text{za Euklidovu metriku,}$$

$$d_{l_\infty}(f^z, f(x)) = \max_{1 \le k \le l} w_k |f_k^z - f_k(x)|, \quad \text{za Čebiševljevu metriku.}$$

Do kraja ovog odeljka koristićemo i razmatrati samo pravougaonu metriku. Sledi lema koja dokazuje da je rešenje dobijeno primenom ove metode zaista pareto optimalno.

Lema 4.6.1 *Rešenje dobijeno primenom metode rastojanja, korišćenjem pravougaone metrike, je Pareto optimalno, ukoliko važi $w_k > 0$, za svako $k = 1, \ldots, l$, i ukoliko je $f^z = (f_1^z, ..., f_l^z)$ idealna vrednost funkcije f.*

Dokaz. Označimo sa $F(x) = \sum_{k=1}^{l} w_k |f_k^z - f_k(x)|$. Neka je x' rešenje dobijeno minimizacijom funkcije $F(x)$ na skupu Ω_P.

Pretpostavimo da rešenje x' nije pareto optimalno. Tada postoji neko dopustivo rešenje $x^* \in \Omega_P$ tako da važi

$$f_k(x^*) \ge f_k(x'), \forall k = 1, \ldots, l$$

gde makar jedna nejednakost prelazi u strogu nejednakost. Iz prethodnog proizilazi

$$f_k^z - f_k(x^*) \le f_k^z - f_k(x'), \forall k = 1, \ldots, l$$

odnosno

$$|f_k^z - f_k(x^*)| \le |f_k^z - f_k(x')|, \forall k = 1, \ldots, l.$$

Poslednje nejednakosti su tačne zato što su f_k^z idealne vrednosti funkcija f_k, što povlači $f_k^z - f_k(x^*) \ge 0$ i $f_k^z - f_k(x') \ge 0$, $k = 1, \ldots, l$. Množenjem svake nejednakosti pozitivnim koeficijentom i njihovim sumiranjem dobijamo:

$$\sum_{k=1}^{l} w_k |f_k^z - f_k(x^*)| < \sum_{k=1}^{l} w_k |f_k^z - f_k(x')|$$

odnosno $F(x^*) < F(x')$. S obzirom na pretpostavke, važi $F(x') \le F(x), \forall x \in \Omega_P$, pa sledi da $x^* \notin \Omega_P$, što predstavlja kontradikciju. Dakle, rešenje problema VKO x' mora biti Pareto optimalno. □

Implementacija ove metode data je funkcijom `PravM[f_List, g_List, fo_List, w_List]`, koja koristi pravougaonu metriku za odredjivanje rastojanja.

Dodatni ulazni parametri su:

- `fo_List`: Lista koordinata tačke kojoj se teži.
- `w_List`: Lista težinskih koeficijenata kriterijumskih funkcija.

Sledi kod implementacije metode rastojanja:

```
PravM[f_List, g_List, fo_List, w_List]:=
  Module[{l = Length[f], fz = {}},
    var = Variables[f];
    If[fo == {},
        For[i = 1, i <= l, i++,
          AppendTo[fz, First[Maximize[f[[i]], g, var]]];
        ],
        fz=fo;
    ];
  d = Sum[w[[i]]*Abs[fz[[i]] - f[[i]]], {i, l}];
  res = Minimize[d, g, var];
  r = First[Rest[res]];
  Print[ReplaceAll[f, r], r];
]
```

Ukoliko je $fo = \{\}$, program generiše listu koordinata kao optimalne vrednosti kriterijumskih funkcija.
Primer (4.2.1) može se rešiti sledećim izrazom, ukoliko su težinski koeficijenti $w_1 = 3$ i $w_2 = 1$:

```
PravM[{40x + 10y, x + y},
    {2x + y<= 6, x + y<= 5, x<= 2, x>= 0, y>= 0}, {}, {3, 1}]
```

Maksimum prve funkcije $f_1 = 40x + 10y$ u dozvoljenoj oblasti iznosi 100, a funkcije $f_2 = x + y$ je 5. Tačka kojoj se teži jeste $fz = \{100, 5\}$. Zadatak koji treba rešiti jeste:

$$\begin{aligned} (\min) \quad & F(x) = 3|100 - 40x - 10y| + |5 - x - y| \\ \text{p.o.} \quad & 2x + y \leq 6 \\ & x + y \leq 5 \\ & x \leq 2 \\ & x, y \geq 0 \end{aligned}$$

Rešenje ovog problema je:

$$\{\{100, 4\}\, \{x \to 2, y \to 2\}\}$$

Dobijeno rešenje je Pareto optimalno, jer ispunjava uslove prethodne leme.

4.7 Linearna višekriterijumska optimizacija

Do sada smo razmatrali opšti oblik problema višekriterijumske optimizacije gde su u opštem slučaju sve funkcije $f_1, \ldots, f_l$ kao i $g_1, \ldots, g_m$ bile nelinearne. U ovom odeljku smatraćemo da su sve navedene funkcije linearne. Neka je:

$$g_i(x) = \sum_{j=1}^{n} a_{ij}x_j - b_i, \qquad f_i(x) = \sum_{j=1}^{n} c_{ij}x_j + d_i$$

Ako označimo matrice $A = [a_{ij}]$, $C^T = [c_{ij}]$ formata $m \times n$ i $l \times n$ dobijamo *problem linearne višekriterijumske optimizacije* u *simetričnom obliku*:

$$\begin{aligned} &\max\ C^T x + d \\ &\ p.o.\ Ax \leq b \quad x \geq 0 \end{aligned} \tag{4.7.1}$$

Ako sada uvedemo *slack* promenljive dobijamo *standardni oblik* problema linearne višekriterijumske optimizacije:

$$\begin{aligned} &\max\ C^T x + d \\ &\ p.o.\ Ax = b \quad x \geq 0 \end{aligned} \tag{4.7.2}$$

Primetimo da ako je polazni problem višekriterijumske optimizacije linearan, da su onda linearni i odgovarajući jednokriterijumski problemi kod svih prethodno razmatranih metoda, osim metode rastojanja. Kod leksikografske i relaksirane leksikografske metode rešavali smo l jednokriterijumskih problema optimizacije, pri čemu su se uzastopni razlikovali u funkciji cilja i u jednom uslovu. Ova činjenica je vrlo zgodna za primenu simpleks metoda za rešavanje odgovarajućih jednokriterijumskih problema. Neka je T^k optimalna Takerova tabela k-tog problema i neka je dodatni uslov:

$$r_1x_1 + \ldots + r_nx_n - e \leqq 0 \tag{4.7.3}$$

Uslov možemo napisati i u obliku $(r_B^k)^T x_B^k + (r_N^k)^T x_N^k - e \leqq 0$. Za bilo koje dopustivo rešenje k-tog problema imamo da važi $x_B + T^k x_N = b^k$. Ako sada x_B zamenimo u (4.7.3) dobijamo:

$$\left((r_N^k)^T - (r_B^k)^T T^k\right) x_N^k + (r_B^k)^T b^k - e \leqq 0$$

Sada možemo da formiramo startnu Takerovu tabelu i RHS vektor za $k+1$-vi problem na sledeći način:

$$\tilde{T}^{k+1} = \begin{bmatrix} T^k \\ (r_N^k)^T - (r_B^k)^T T^k \end{bmatrix} \quad \tilde{b}^{k+1} = \begin{bmatrix} b^k \\ -(r_B^k)^T b^k + e \end{bmatrix}$$

Na identičan način se formira i novi vektor ciljne funkcije $\tilde{c}^{k+1}$. U zavisnosti od vrste novog uslova, kao i od znaka izraza $-(r_B^k)^T b^k + e$ potrebno je izvršiti odgovarajuće faze simpleks metoda, pri čemu je početno rešenje najčešće vrlo "blizu" optimalnom. Ovo naravno ne mora uvek da bude slučaj. U većini slučajeva, ovim adaptivnim metodom se znatno redukuje ukupan broj iteracija.

Razmotrimo sada metodu rastojanja u slučaju linearnog problema višekriterijumske optimizacije. Neka je odabrana metrika euklidska i neka su sve težine $w_k = 1$. Sada

rešavamo imamo sledeći problem kvadratnog programiranja:

$$\begin{aligned} \min \sqrt{\sum_{i=1}^{l} (C_{i\bullet}x - f_i^z)^2} &= \|Cx - f^z\| \\ p.o\ Ax &= b \\ x &\geq 0 \end{aligned} \tag{4.7.4}$$

Ovaj problem se može rešavati standardnim metodama kvadratnog programiranja [32], kojima se u ovom radu nećemo baviti. Medjutim, posmatrajmo na šta se svodi ovaj problem ako pretpostavimo da ograničenja ne postoje. Tada rešavamo bezuslovni problem linearne višekriterijumske optimizacije koji svodimo na nalaženje $\min_{x\in\mathbb{R}^n} \|Cx - f^z\|$ odnosno odgovarajuće tačke u kojoj se dostiže minimum. Pretpostavićemo da smo f^z izabrali tako da ne pripada slici operatora C.

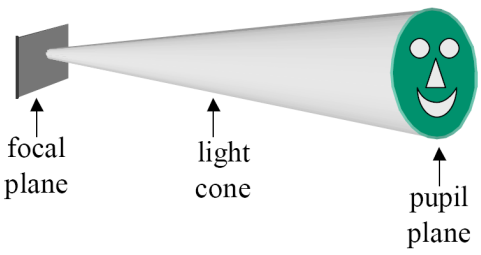
U tom slučaju, rešenje se dobija kao $x^* = C^\dagger f^z$ gde je $C^\dagger$ *Moore-Penrose*-ov pseudoinverz matrice C. Više o generalisanim inverzima i svojstvima vezanim za njih može se naći u monografijama [51, 3]. Tu su takodje opisani mnogi direktni i iterativni metodi za numeričko i simboličko izračunavanje istih. U našim radovima [35, 36, 42, 37] prikazani su metodi za simboličko izračunavanje Moore-Penrose-ovog i drugih pseudoinverza polinomijalnih matrica. U radovima [35, 36] je opisana modifikacija Grevile-ovog partitioning metoda [51] za racionalne i polinomijalne matrice. U [42, 37] bavili smo se primenom interpolacionih metoda. Na taj način smo za ceo red veličine smanjili složenost Leverrier-Faddev metoda primenjenog na polinomijalne matrice. Ova istraživanja su značajna za višekriterijumsku optimizaciju jer omogućavaju da se odjednom reši celokupna klasa optimizacionih problema koja je zadata polinomijalnom matricom $C(s)$.

5. Primene linearnog programiranja i višekriterijumske optimizacije

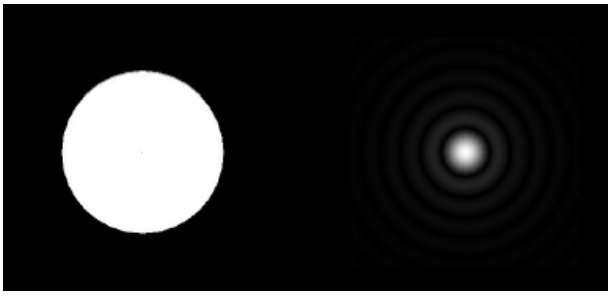
U ovoj glavi ćemo pokazati dve praktične primene prethodno izloženih metoda matematičkog programiranja (linearnog programiranja i višekriterijumske optimizacije), u astronomiji i u telekomunikacijama (elektronici). Linearno programiranje i višekriterijumska optimizacija, kao i ostale mnogobrojne metode matematičkog programiranja imaju primene u skoro svim oblastima nauke, tehnike i uopšte u svakodnevnom životu. Sledeća dva primera lepo ilustruju tu činjenicu.

5.1 Izračunavanje optimalne maske teleskopa

U ovom odeljku razmotrićemo primenu linearnog programiranja na problem projektovanja teleskopa pomoću kog ćemo moći da utvrdimo da li oko neke zvezde kruže planete. Na slici 5.1.1 prikazan je uprošćeni model teleskopa.



Slika 5.1.1. Principijelna šema teleskopa.

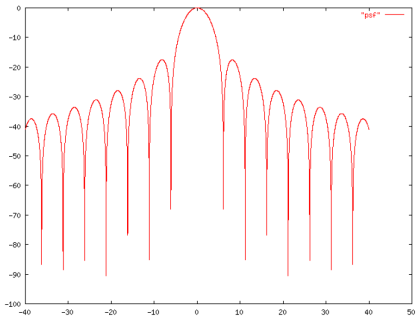


Slika 5.1.2. Profil objektiva i difrakciona slika.

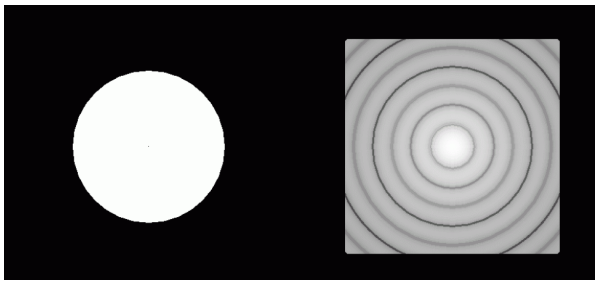
Svetlost sa udaljene zvezde pada na ravan objektiva teleskopa (eng. *pupil plane*), prelama se kroz sočivo objektiva i pada na žižnu ravan (eng. *focal plane*). Svi svetlosni zraci koji pod istim uglom padnu na objektiv, posle prelamanja padaju u istu tačku na žižnoj ravni. Svetlosni zraci koji dolaze 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 $(0, 0)$).

Medjutim, usled talasne prirode svetlosti, dolazi do difrakcije na otvoru objektiva i umesto jedne svetle tačke, vidimo mrlju konačne veličine koju okružuju prstenovi (slika 5.1.2).

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 5.1.3 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 5.1.4 je difrakciona slika, pri čemu je sada referentni nivo -110dB.



Slika 5.1.3. Intenzitet svetlosti na x koordinatnoj osi.



Slika 5.1.4. 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:

$$\left\{(x,y) \mid -\frac{1}{2} \leq x \leq \frac{1}{2},\ -A(x) \leq y \leq A(x)\right\} \tag{5.1.1}$$

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 [16], 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 (5.1.1). Uslov koji ćemo sada nametnuti je da u prethodno zadatoj oblasti $\mathcal{O}$ važi sledeći uslov:

$$-\alpha E(0,0) \leq E(\xi,\zeta) \leq \alpha E(0,0), \quad (\xi,\zeta) \in \mathcal{O} \tag{5.1.2}$$

Pretpostavimo sada da su tačke skupa $\mathcal{O}$ na x osi (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 je uslov (5.1.2):

$$\int_0^{1/2} A(x)\left[\cos(x\xi) - \alpha\right] dx \leq 0$$
$$\int_0^{1/2} A(x)\left[\cos(x\xi) + \alpha\right] dx \leq 0$$

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:

$$\begin{aligned} &\max\ 4\int_0^{1/2} A(x)dx \\ &p.o. \int_0^{1/2} A(x)\left[\cos(x\xi) - \alpha\right] dx \leq 0 \\ &\int_0^{1/2} A(x)\left[\cos(x\xi) + \alpha\right] dx \leq 0 \\ &0 \leq A(x) \leq \frac{1}{2} \end{aligned} \tag{5.1.3}$$

Ako sada izvršimo odgovarajuću diskretizaciju integrala i skupa $\mathcal{O}$ dobijamo sledeći problem linearnog programiranja [20, 49]:

$$\begin{aligned} &\max\ 4\sum_{i=1}^{N_x} B_i A(x_i) \\ &p.o. \sum_{i=1}^{N_x} C_i A(x_i)\left[\cos(x_i\xi_j) - \alpha\right] \leq 0, \quad j = 1, \ldots, N_\xi \\ &\sum_{i=1}^{N_x} D_i A(x_i)\left[\cos(x_i\xi_j) - \alpha\right] \leq 0 \quad j = 1, \ldots, N_\xi \\ &0 \leq A(x_i) \leq \frac{1}{2} \end{aligned} \tag{5.1.4}$$

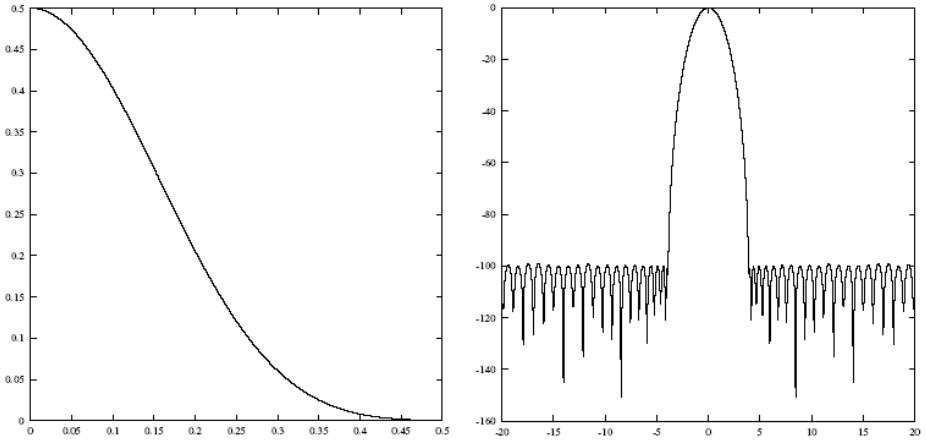
Ovde smo sa N_x i N_ξ označili broj diskretizacionih nivoa redom za promenljive x i ξ. ξ_i su vrednosti diskretizovane promenljive ξ. Ako je skup $\mathcal{O}$ zadat npr. kao interval

$$\mathcal{O} = \{(\xi, 0) \mid \xi_0 < \xi < \xi_1\}$$

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 (5.1.4) 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 5.1.5 za funkciju $A(x)$ [20].



Slika 5.1.5. 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 [20].

5.2 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* [28]. 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 Furijeova transformacija i z-transformacija. One su date pomoću sledećeg izraza:

$$X(f) = \sum_{n=-\infty}^{+\infty} x(n)e^{2\pi fin} \qquad X(z) = \sum_{n=-\infty}^{+\infty} x(n)z^{-n} \tag{5.2.5}$$

Furijeova 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 odredjenom opsegu. Frekvencijska karakteristika idealnog filtra može se prikazati pomoću izraza [28]:

$$|H(f)| = \left|\frac{Y(f)}{X(f)}\right| = \begin{cases} 1, & f \in \mathcal{O} \\ 0, & \text{U suprotnom} \end{cases}$$

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 Furijeova 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 obezbedjuje linearnost faze kompleksne funkcije $H(f)$. Sada je $|H(f)|$, takodje, simetrična funkcija i važi $|H(f)| = |A(f)|$ gde je $A(f)$ definisana pomoću [28]:

$$A(f) = h(0) + 2 \sum_{t=1}^{n_0} h(t) \cos(2\pi f t) \tag{5.2.6}$$

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 minimizovati:

$$\begin{aligned} &\min \int_0^{f_0} |A(f) - 1| df \\ &\min \int_{f_0}^{1/2} |A(f)| df \end{aligned} \tag{5.2.7}$$

Ovim smo dobili problem bezuslovne višekriterijumske nelinearne optimizacije. Promenljive su nam, u ovom slučaju vrednosti $h(t)$, $t = 0, \ldots, n_0$. Uvedimo sada pomoćne funkcije $\tau(f)$ i $\eta(f)$. Problem (5.2.7) možemo ekvivalentno predstaviti u obliku:

$$\begin{aligned} &\min \int_0^{f_0} \tau(f) df \\ &\min \int_{f_0}^{1/2} \eta(f) df \\ &p.o. \ -\tau(f) \leq A(f) - 1 \leq \tau(f), \quad f \in \mathcal{O} \\ &\qquad -\eta(f) \leq A(f) \leq \eta(f), \quad f \notin \mathcal{O} \end{aligned} \tag{5.2.8}$$

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 (5.2.7) svodimo na sledeći problem:

$$\begin{aligned} &\min \int_0^{f_0} \tau(f) df \\ &p.o. \ -\tau(f) \leq A(f) - 1 \leq \tau(f), \quad f \in \mathcal{O} \\ &\qquad -\epsilon \leq A(f) \leq \epsilon, \quad f \notin \mathcal{O} \\ &\qquad \tau(f) \geq 0 \end{aligned} \tag{5.2.9}$$

a ako uvedemo diskretizaciju po f direktno dobijamo problem linearnog programiranja u simetričnom obliku:

$$\begin{aligned} \min \ & \sum_{j=1}^{N'_f} B_j \tau(f_j) \\ \text{p.o.} \ & -\tau(f_j) \le h(0) + 2\sum_{t=1}^{n_0} h(t)\cos(2\pi f_j t) - 1 \le \tau(f_j), \quad j = 1, \dots, N'_f \\ & -\epsilon \le h(0) + 2\sum_{t=1}^{n_0} h(t)\cos(2\pi f_j t) \le \epsilon, \quad j = N'_f + 1, \dots, N_f + N'_f \end{aligned} \tag{5.2.10}$$

Primetimo da je ovo svodjenje isto kao kod metode ϵ ograničenja iz prethodne glave. Kao i u prethodnom odeljku, i ovde su f_j i B_j, $j = 1, \dots, N'_f$ čvorovi i koeficijenti odgovarajuće kvadraturne formule. Za $j > N'_f$ vrednosti f_j su iz skupa $\mathcal{O}^C$. Promenljive su nam sada $\tau(f_j)$, $j = 1, \dots, N'_f$ i $h(t)$, $t = 0, \dots, n_0$.

U radu [6], razmatran je problem reprodukcije signala pomoću tri FIR filtra, pri čemu svaki propušta različiti opseg frekvencija. Imamo da je spektar paralelne veze ovih filtara:

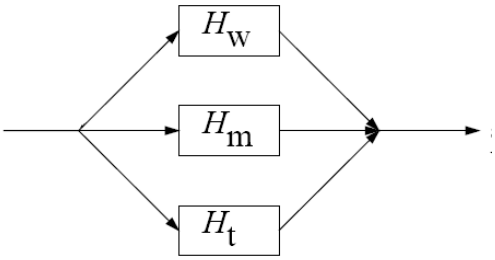
$$H(f) = H_w(f) + H_m(f) + H_t(f)$$

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:

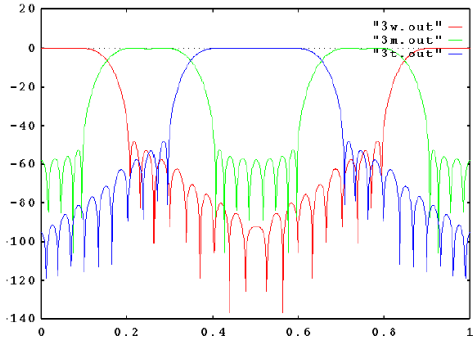
$$\begin{aligned} \min \ & \int_0^{1/2} |A_w(f) + A_m(f) + A_t(f) - 1| \\ p.o. \ & -\epsilon \le A_i(f) \le \epsilon, \quad f \in \mathcal{O}_i, \quad i = w, m, t \end{aligned} \tag{5.2.11}$$

koji na sličan način svodimo na problem linearnog programiranja.

Na slici 5.2.2 su prikazane funkcije $A_w(f)$, $A_m(f)$ i $A_t(f)$ za sledeće vrednosti parametara [6]: $N_f = 1000$, $f_w = 0.2$, $f_t = 0.7$, $f_{m1} = 0.1$ i $f_{m2} = 0.4$.



Slika 5.2.1. Blok šema paralelne veze tri filtra.



Slika 5.2.2. Grafici funkcija $A_w(f)$, $A_m(f)$ i $A_t(f)$.

6. Zaključak

U ovom radu analizirana su dva problema matematičkog programiranja: linearno programiranje i višekriterijumska optimizacija. Definisan je i detaljno obradjen problem linearnog programiranja kao i dva metoda za rešavanje ovog problema: geometrijski i simpleks metod. Veća pažnja je posvećena simpleks metodu zbog toga što geometrijski metod rešava samo specijalan slučaj problema linearnog programiranja. Detaljno su proučene sve faze simpleks metoda. Za svaku fazu je dat i odgovarajući primer na kome se pokazuje kako ona funkcioniše. Izloženo je nekoliko originalnih modifikacija i poboljšanja simpleks metoda koje su takodje detaljno obradjene.

Detaljno je obradjen problem višekriterijumske optimizacije kao i metode za njegovo rešavanje. Uočena je i veza izmedju višekriterijumske optimizacije sa jedne strane, i simpleks metoda i generalisanih inverza sa druge.

Primene prethodno izloženih metoda ilustrovane su na dva primera iz sasvim različitih oblasti: iz astronomije (optike) i iz elektronike (telekomunikacija).

Kratak pregled svih izloženih rezultata kao i neke ideje za dalja istraživanja biće prikazani u ovoj, zaključnoj glavi.

A. Detaljno smo proučili problem linearnog programiranja. Celine koje smo pritom obradili, kao i rezultati i zaključci do kojih smo došli, mogu se sistematizovati na sledeći način:

A.1. Definisali smo matematički model i osnovne oblike problema linearnog programiranja [15, 5, 50, 44]. Pokazali smo kako se problem iz jednog oblika može transformisati u drugi.

A.2. Definisali smo i proučili osnovne osobine skupa dopustivih rešenja [15, 5]. Definisali smo pojmove bazičnog i bazično dopustivog rešenja [50]. Pokazali smo da svako bazično dopustivo rešenje predstavlja ekstremnu tačku skupa dopustivih rešenja, kao i obratno, da je svaka ekstremna tačka bazično dopustivo rešenje. Dokazali smo da funkcija cilja ima optimum upravo u ekstremnoj tački, odnosno u bazično dopustivom rešenju.

A.3. Formulisali smo geometrijski metod za rešavanje problema linearnog programiranja [44]. Pokazali smo kako u specijalnim slučajevima izgleda skup dopustivih rešenja.

A.4. Prezentovali smo našu originalnu implementaciju geometrijskog metoda [47] u programskom jeziku MATHEMATICA [53]. Rad programa GEOM, koji je na taj način nastao, pokazali smo na jednom primeru. Time smo na jedan lep način vizuelno predstavili prethodno razmatranu teoriju.

B. Proučili smo sve faze simpleks metoda linearnog programiranja [2, 50, 44].

B.1. Posmatrali smo kanonski oblik i nekoliko slučajeva koji mogu da nastupe u zavisnosti od znaka elemenata vektora funkcije cilja. U prvom slučaju, dokazali smo da je odgovarajuće bazično rešenje optimalno, u drugom da je funkcija cilja neograničena, dok je u trećem bilo moguće konstruisati novo bazično dopustivo rešenje tako da se vrednost funkcije cilja povećava.

B.2. Definisali smo pojam Takerove tabele [50] i proširene Takerove tabele. Pokazali smo kako se vrši zamena bazične i nebazične promenljive u Takerovoj tabeli i formulisali algoritam zamene. Zatim smo formulisali simpleks metod za bazično dopustive kanonske oblike. Na primerima smo pokazali rad oba algoritma.

B.3. Formulisali smo dva metoda za nalaženje prvog bazično dopustivog rešenja. U prvom metodu [7] se dodaju tzv. veštačke promenljive i time se povećavaju dimenzije problema. Kod drugog metoda [50] to nije slučaj, pa smo isti detaljnije izložili i ilustrovali na primeru.

B.4. Pokazali smo kako se efektivno vrši svodjenje polaznog problema u opštem obliku na kanonski. Formulisali smo dva algoritma kojima se to obavlja, algoritam za eliminaciju jednačina i algoritam za eliminaciju slobodnih promenljivih. Kao i za prethodne algoritme i ovde je dat ilustrativan primer na kome smo pokazali rad algoritama.

B.5. Ukazali smo na problem nagomilavanja računske greške kod simpleks metoda [2, 44]. Da bi rešili ovaj problem formulisali smo revidirani simpleks metod. Iskoristivši vezu izmedju Takerove tabele, bazične i nebazične matrice formulisali smo metod za rekonstrukciju Takerove tabele kao i pojedninačnih vrsta ili kolona. Sve prethodno opisane algoritme, formulisali smo jezikom revidiranog simpleks metoda. Iako kod revidiranog simpleks metoda nema nagomilavanja greške, značajno je povećana složenost jedne iteracije, tj. značajno se gubi na brzini.

B.6. Proučili smo problem cikliranja simpleks metoda. Najpre smo dali primer na kome se cikliranje uočava. Zatim smo proučili dva metoda pomoću kojih se cikliranje može izbeći.

Formulisali smo leksikografsko anticiklično pravilo [7]. Formalno smo dokazali, da se primenom ovog pravila vrsta koja odgovara funkciji cilja u ekvivalentnoj matrici standardnog oblika leksikografski povećava, pa je zato cikliranje nemoguće. Mana ovog metoda je povećanje složenosti jedne iteracije simpleks metoda.

Drugo anticiklično pravilo koje smo proučili je Blandovo pravilo [50, 4]. Kao i kod leksikografskog pravila, i ovde smo formalno dokazali da je uz primenu Blandovih pravila cikliranje nemoguće. Ovaj metod ne povećava složenost iteracije, ali može značajno povećati broj iteracija i smanjiti tačnost izračunavanja.

B.7. Dokazali smo da je složenost simpleks metoda eksponencijalna [22]. Konstruisali smo primer za koji je simpleks metodu potrebno $2^n - 1$ iteracija (n je dimenzija problema) da dodje do optimalnog rešenja. Pokazali smo da skup dopustivih rešenja u konstruisanom primeru ima oblik deformisane hiperkocke i definisali smo pojam Minti-Kli poliedra. Dali smo i nekoliko procena za očekivani broj iteracija simpleks metoda ukoliko se pivot element bira slučajno [12].

C. Prezentovali smo nekoliko naših originalnih modifikacija simpleks [41, 40] i revidiranog simpleks metoda [34]. Najvažniji rezultati u tom pogledu su sledeći:

C.1. Formulisali smo poboljšanje algoritma za zamenu promenljivih [41] uzimajući u obzir da matrica sistema i Takerova tabela u većini praktičnih primera sadrže mali broj elemenata različitih od 0. Našim metodom modifikuju se samo oni elemente Takerove tabele koji menjaju vrednost, čime se značajno ubrzava algoritam za zamenu.

C.2. Naš **glavni rezultat** u ovom radu predstavljaju modifikacije algoritma za traženje prvog bazično dopustivog rešenja [41, 40]. Izložene su tri verzije ove modifikacije, dve za simpleks i jedna za revidirani simpleks metod.

Najpre smo uočili dva nedostatka klasičnog algoritma za nalaženje prvog bazično dopustivog rešenja [41]. Suština uočenih nedostataka je mogućnost da se broj negativnih koordinata bazičnog rešenja povećava.

Teorijskim razmatranjima smo zatim došli do metoda koji nema ove nedostatke. Formulisali smo i dokazali glavnu lemu ovog dela rada, na osnovu koje smo zatim konstruisali metod kod koga se broj negativnih koordinata bazičnog rešenja ne povećava.

Pokazali smo kako se prethodno izložena modifikacija može poboljšati [40]. Za razliku od prve modifikacije, poboljšana modifikacija može u nekim slučajevima smanjiti broj negativnih koordinata bazičnog rešenja za više od 1.

Na sličan način, uočili smo i nedostatke revidiranog simpleks metoda i teorijski došli do metoda kojim se oni prevazilaze. Formulisali smo odgovarajući algoritam koji predstavlja modifikaciju revidiranog simpleks metoda [34].

C.3. Sve prethodno izložene algoritme simpleks metoda kao i originalne modifikacije implementriali smo u programskom jeziku Visual Basic 6.0 [29]. Tako je nastao naš originalni softver MarPlex koji rešava probleme linearnog programiranja. Program je testiran na referentnim svetskim test primerima, i pritom smo pokazali kako se primenom izloženih modifikacija sa manjim brojem iteracija dolazi do optimalnog rešenja. Rešenja dobijena našim programom, uporedili smo sa rešenjima koje je dao jedan od najjačih svetskih solvera PCx. Oba programa su na svim primerima dala približno jednaka rešenja, pri čemu je MarPlex na većini primera imao **bolju preciznost**.

C.4. Revidirani simpleks metod smo implementirali u programskom jeziku MATHEMATICA [53]. Tako je nastao još jedan naš originalni softver RevMarPlex. U ovom slučaju, MATHEMATICA je bila pogodniji jezik zbog svojih integrisanih funkcija za rešavanje sistema linearnih jednačina. RevMarPlex smo takodje testirali na svetskim test primerima. Rezultati su i ovde pokazali da je primenom naše originalne modifikacije potreban manji broj iteracija za nalaženje optimalnog rešenja.

C.5. Oba naša programa smo testirali na još jednoj klasi veoma loše uslovljenih test primera koje program PCx nije bio u stanju da reši. I u ovom slučaju su oba programa dali korektna rešenja, pri čemu je preciznost RevMarPlex-a bila znatno veća. Ovime smo na najbolji način pokazali prednost simpleks metoda nad metodima unutrašnje tačke (prvenstveno primal-dual metodima) kod loše uslovljenih primera.

D. Proučili smo problem višekriterijumske optimizacije [27, 15, 31] koji je takodje, kao problem linearnog programiranja, veoma primenljiv u praksi.

D.1. Definisali smo problem višekriterijumske optimizacije (VKO) kao i osnovne pojmove vezane za njega [15]. Definisan je i pojam Pareto optimalnog rešenja koji je fundamentalan kod problema VKO.

D.2. Formulisali smo nekoliko klasičnih metoda za rešavanje problema VKO. Za svaki metod smo dali originalnu implementaciju [43] u programskom jeziku MATHEMATICA. Takodje, svaki metod smo ilustrovali na po jednom primeru.

D.3. Razmatrali smo neke specijalne slučajeve problema VKO i pokazali smo kako se, pod odredjenim uslovima, na njih mogu primeniti simpleks metod i teorija generalisanih inverza. Takodje smo ukazali na moguću primenu naših originalnih rezultata [35, 36, 42, 37] iz teorije generalisanih inverza u tim slučajevima problema VKO.

E. Pokazali smo dve moguće primene problema linearnog programiranja i višekriterijumske optimizacije. Prva primena je u astronomiji (optici) a druga u telekomunikacijama (elektronici).

E.1. Prva primena se odnosi na projektovanje optimalne maske objektiva [16, 20, 49]. Formulisali smo problem i pokazali kako se on svodi, prvo na problem VKO, a onda i na problem linearnog programiranja. Grafički su prikazani rezultati dobijeni u konkretnom slučaju.

E.2. Druga primena se odnosi na projektovanje FIR filtara [28, 6]. I ovde smo formulisali problem i sveli ga na problem linearnog programiranja. Razmotrili smo i modifikovanu verziju problema koju smo takodje sveli na problem linearnog programiranja, a zatim smo prikazali rezultate u jednom konkretnom slučaju.

Na samom kraju ovog rada, napomenimo da bi dalja istraživanja na ovu temu mogla da se odvijaju u sledećim pravcima:

1. Prezentovane modifikacije metoda za nalaženje prvog bazično dopustivog rešenja mogle bi se primeniti na algoritme za eliminaciju jednačina i slobodnih promenljivih. Jedna takva modifikacija odnosila bi se na izbor pivot vrste ili kolone koji kod ovih algoritama nisu u potpunosti odredjeni. Uvodjenjem dodatnih uslova prilikom izbora, mogao bi se ubrzati dalji tok algoritma.

2. Pokazali smo da teorija generalisanih inverza ima primenu kod problema VKO. Ovu vezu dve fundamentalno različite oblasti bi trebalo detaljnije proučiti.

3. U delu vezanom za složenost simpleks metoda ukazali smo na očekivan broj iteracija simpleksa na uopštenom Minti-Kli primeru [12], ukoliko se pivot element bira slučajno. Sličan rezultat mogao bi da se izvede i za ostale izložene faze simpleks metoda kao i za modifikacije.

A. Dodatak: Kodovi i implementacioni detalji programa

U ovom dodatku biće prikazani delovi kodova naših originalnih programa uz odgovarajuća objašnjenja.

A.1 Program GEOM

U prvom delu funkcije `GEOM` traži se skup dopustivih rešenja, odnosno njegove ekstremne tačke.

Najpre se prikazuje skup ograničenja koristeći funkciju `InequaltyPlot` koja se nalazi u paketu `Graphics‘InequalityGraphics`. Rešavajući sistem nejednačina odredjujemo oblast dopustivih rešenja `h`. Pri tome koristimo funkciju `InequalitySolve` iz paketa `Algebra‘InequalitySolve`. Ekstremne tačke pamtimo u `res` i dobijamo ih u preseku svake dve prave dobijene prevodjenjem nejednačina iz liste ograničenja u jednačine. Sada sortiramo skup extremnh tačaka `res` prema udaljenosti od prave odredjene 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]]]];
     ];
   ];
```

A.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 kod 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 Takerovu tabelu.
3. Nizovi `nbasicv` i `basicv` koji predstavljaju prvu vrstu i zadnju kolonu matrice Takerove tabele.
4. Četiri konstante (`Infinity, Impossibile, Solution, Error`) jedna logička promenljiva `maksim` koja označava da li je problem 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 metoda. 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
Dim nv As Integer
Dim nk As Integer
Dim h As Integer
Dim j As Integer
Dim 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**.

Kod funkcije `SimplexMod` je:

```
Public Function SimplexMod(res As Integer) As Double
Dim i As Integer
Dim j As Integer
Dim p As Integer
Dim from As Integer
Dim from1 As Integer
Dim 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
Dim l As Integer, p As Integer, niter As Integer
Dim pivrow As Integer, pivcol As Integer, s() As Double, piv() As Integer
Dim 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 nadjemo 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
Dim pivcol As Integer
Dim pivrow As Integer
Dim 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 Takerova tabela.

A.3 Program RevMarPlex

Program `RevMarPlex` se sastoji od sledećih modula:

```
DefaultFunctions
Revised Simplex
```

U modulu `DefaultFunctions` nalaze se pomoćne funkcije dok su u modulu `RevisedSimplex` 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 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] M.D. Ašić and V.V. Kovačević-Vujčić,, *Ill-conditionedness and interior-point methods,* Univ. Beograd Publ. Elektrotehn. Fak., **11** (2000), 53–58.

[2] M.A. Bhatti, *Practical optimization with MATHEMATICA applications,* Springer Verlag Telos, 2000.

[3] 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.

[4] R.G. Bland, *New finite pivoting rules for the simplex method*, Mathematics of Operations Research **2** (1977) 103–107.

[5] B.D. Bounday, *Basic linear programming,* Edvard Arnold, Baltimore, 1984.

[6] J.O. Coleman, *A Systematic Approach to the Constrained Quadratic Optimization of Embedded FIR Filters*, Conference on Information Sciences and Systems, Princeton, March 1998.

[7] D. Cvetković, M. Čangalović, Dj. Dugošija, V. Kovačević-Vujčić, S. Simić, J. Vuleta, *Kombinatorna optimizacija-Matematička teorija i algoritmi* Društvo operacionih istraživača Jugoslavije DOPIS, Beograd 1996.

[8] J. Czyzyk, S. Mehrotra and S.J. Wright, *PCx User Guide*, Optimization Thechnology Center, Technical Report 96/01 (1996) 1–21.

[9] G. B. Dantzig, *Programming of Interdependent Activities, Mathematical Model*, Econometrica 17, 1949, 200–211.

[10] G.B. Danzing, *Linear programming and Extensions*, Princeton University Press, Princeton, New Yersy 1963.

[11] 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.

[12] B. Gartner, M. Henk, G.M. Ziegler, *Randomized Simplex Algorithms on Klee-Minty Cubes*, http://citeseer.ist.psu.edu/427730.html.

[13] 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.

[14] F. L. Hitchcock, *The distribution of a product from several sources to numerous localitites*, Journal Math. and Phys. 20, 1941, 224–230.

[15] J.P. Ignizio, *Linear programming in single-multiple-objective systems,* Englewood Cliffs: Prentice Hall, 1982.

[16] D.M. Ivanović, V.M. Vučić, *Fizika II, Elektromagnetika i optika*, Naučna knjiga, 1967.

[17] V. Ivanović, *Pravila za proračun potrebnog broja transportnih sredstava*, Vojnoizdavački glasnik, sveska 1-3, 1940, 1–10.

[18] L.V. Kantorovich *Matematicheskie metodi v organizacii i planirovanii proizvodstva*, Izd. LGU, 1939.

[19] N. Karmarkar,*A new polynomial-time algorithm for linear programming*, Combinatorica, 4, 1984, 373-395.

[20] 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.

[21] L.G. Khachian, *A Polinomial Algorithm in Linear Programming*, Doklady Akademii Nauk SSSR, Vol. 244, No, 5, 1979, pp. 1093-1096.

[22] V. Klee, G.L. Minty, *How good is the simplex method?*, O. Shisha, ed., Inequalities III, Academic Press, New York, 1972, pp. 159–175.

[23] V. Klee, P. Kleinschmidtm, *The d-step conjecture and its relatives*, Math. Operations Research 12, 1987, pp. 718–755.

[24] 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.

[25] V. Kovačević-Vujčić, and M.D. Ašić, *Stabilization of interior-point methods for linear programming,* Computational Optimization and Applications, **14** (1999), 1–16.

[26] K. Miettinen, *Nonlinear Multiobjective Optimization*, Kluwer Academic Publishers, Boston 1999.

[27] 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.

[28] Lj. Milić Z. Dobrosavljević, *Uvod u Digitalnu Obradu Signala*, Elektrotehnički fakultet, Univerzitet u Beogradu, 1999.

[29] *Microsoft Developer Network (13)*, Microsoft Corporation Inc., 1998.

[30] G.V. Milovanović *Numerička analiza I deo*, Naučna knjiga, Beograd 1991.

[31] S. Narula, L. Kirilov, V. Vassilev, Reference direction approach for solving multiple objective nonlinear programming problems, IEEE Transactions on Systems, Man, and Cybernetics, 24(5), 804-806, 1994.

[32] E. Nering, A. Tucker, *Linear Programs and Related Problems*, Academic Press, New York, 1993.

[33] *www.netlib.org*

[34] M.D. Petković, P.S. Stanimirović, N.V. Stojković, *Two modifications of revised simplex metod*, Matematicki Vesnik, 54 (2002), pp. 163–169.

[35] M.D. Petković, P.S. Stanimirović, *Partitioning method for two-variable rational and polynomial matrices*, Mathematica Balcanica vol. 19, 2005, pp. 185–194.

[36] 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.

[37] M.D. Petković, P.S. Stanimirović, *Interpolation algorithm of Leverrier-Faddev type for polynomial matrices*, Numerical Algorithms, prihvaćeno za štampu.

[38] M. Sakarovitch, *Linear programming,* Springer-Verlag, New York, 1983.

[39] W. Stadler (Ed.), *Multicriteria Optimization in Engineering and in the Sciences*, Plenum Press, New York, 1988.

[40] P.S. Stanimirović, N.V. Stojković, M.D. Petković, *Modification and implementation of two phases simplex method*, biće objavljeno.

[41] P.S. Stanimirović, N.V. Stojković, M.D. Petković, *Several Modifications of Simplex Method*, Filomat, 2003.

[42] 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.

[43] P.S. Stanimirović, N.V. Stojković, M.D. Petković, *Run-time transformations in implementation of linear multi-objective optimization*, PRIM, Budva 2004.

[44] R. Stanojević, *Linearno Programiranje*, Institut za ekonomiku industrije, Beograd 1966.

[45] N.V. Stojković, *Primal-dual i simpleks metodi za rešavanje problema linearnog programiranja*, Doktorska disertacija, PMF Niš, 2001.

[46] J. Strayer, *Linear Programming and Its Applications*, Springer-Verlag 1989

[47] 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.

[48] 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.

[49] R.J. Vanderbei, N.J. Kasdin, D.N. Spergel, *Rectangular-Mask Coronagraphs for High-Contrast Imaging*, arXiv:astro-ph/0401644 v1 30 Jan 2004.

[50] S. Vukadinović, S. Cvejić *Matematičko programiranje,* Univerzitet u Prištini, Priština, 1996.

[51] G.Wang, Y.Wei and S. Qiao, *Generalized Inverses: Theory and Computations*, Science Press, Beijing, 2004.

[52] 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.

[53] S. Wolfram, *The Mathematica Book, 4th ed.*, Wolfram Media/Cambridge University Press, 1999.

[54] Y. Zang, *User's guide to LIPSOL*, Department of Mathematics and Statistics, University of Maryland, Baltimore County, Baltimore, Md., July 1995.