Degeneracija problema linearnog programiranja

Zadravec, Veronika

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Veronika Zadravec

DEGENERACIJA PROBLEMA LINEARNOG PROGRAMIRANJA

ZAVRŠNI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Veronika Zadravec

Matični broj: 43159/14–R

Studij: Poslovni sustavi

DEGENERACIJA PROBLEMA LINEARNOG PROGRAMIRANJA

ZAVRŠNI RAD

Mentor/Mentorica:

Dr. sc. Perši Nenad

Varaždin, rujan 2019.

Veronika Zadravec

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristila drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Tema ovog rada je degeneracija problema kod linearnog programiranja. U ovom radu prvo je objašnjeno što su operacijska istraživanja i linearno programiranje te primjena danas. Rad se fokusira na linearno programiranje. Kroz rad navedeni su i objašnjeni opći, standardni i kanonski problem kod linearnog programiranja, te dualni oblik. U nastavku objašnjena je simpleks metoda koja je jedna od metoda rješavanja problema linearnog programiranja, te grafička metoda i problem transporta. Isto tako objašnjeno je u radu što je degeneracija i kruženje kod simpleks metode i kod transportnog problema te na primjeru prikazano kako riješiti takve zadatke. Na kraju je programsko rješenje za 2 primjera degeneracije kod simpleks algoritma.

Ključne riječi: simpleks, linearno programiranje, minimum, maksimum, degeneracija, opći problem, standardni problem, kanonski, dual, transportni

Sadržaj

1.Uvod

Tema ovog završnog rada je degeneracija problema kod linearnog programiranja. Svrha ovog rada je pobliže objasniti što je degeneracija te kako se rješava simpleks metodom. Kod svakog problema rješavanog pomoću simpleks metode računamo maksimalni profit ili minimalni utrošak resursa. Kod stvarnih primjera proizvodnje u nekom poduzeću simpleks metoda predstavlja način kako izračunati koje proizvode, u kojoj količini i na kojim radnim mjestima proizvoditi u određenim uvjetima.

Najprije su opisana operacijska istraživanja i linearno programiranje. Pobliže je objašnjena važnost linearnog programiranja na primjeni danas. Kroz rad objašnjeni su opći, kanonski i standardni oblik problema te dual istih. Problemi su prikazani u matričnom i vektorskom zapisu. Potom je objašnjena simpleks metoda koja je jedna od najpoznatijih metoda za rješavanje problema linearnog programiranja. Navedene su razlike između problema za maksimum i minimum dok je cilj simpleks metode optimizacija funkcije, tj. pronalaženje najboljeg odnosa inputa i outputa kako bi se ostvario maksimalni profit ili minimalni troškovi. Uz simpleks metodu postoje i grafička metoda koja se koristi kada imamo problem optimizacije s dvije varijable, te problem transporta kod kojeg tražimo najisplativiji transport nekog tereta. Rješenje koje odabiremo naziva se optimalno rješenje kod kojeg su svi uvjeti zadovoljeni. Moguće je jedno ili više optimalnih rješenja.

Potom je objašnjen problem degeneracije, kada nastaje i kako se manifestira, te koji su problemi kod pojave istog. Kada dođe do degeneracije to znači da se rješenja svake iduće iteracije u simpleks tablici neće razlikovati od prethodnog, tj. ne dolazi do poboljšanja programa i porasta funkcije. Radi toga može doći do kruženja rješenja koja su izvan dometa optimalnog rješenja. Zato je potrebno obratiti pozornost na metode rješavanja degeneriranih problema.

Na kraju je na aplikaciji prikazano rješenje problema simpleks metodom kod kojeg je došlo do degeneracije i izveden je kratki zaključak ovog rada.

2.Metode i tehnike rada

U ovom radu korištena je metoda istraživanja i analize za prikupljanje podataka. Najviše korištene su informacije i podaci iz stručne literature o ovom znanstvenom području. Uz knjige, korišteno je i nekoliko internetskih izvora za izradu teoretskog dijela završnog rada.

Za izradu programskog rješenja završnog rada korišteni su sljedeći alati i biblioteke:

- Programski jezik Python
- Python 3.7
- Jupyter Notebook
- Scipy

3.Operacijska istraživanja i linearno programiranje

Operacijska istraživanja nemaju jedinstvenu definiciju, ali taj pojam je moguće opisati. „Hrvatsko društvo za operacijska istraživanja (HDOI) izdalo je 1994. Godine brošuru [56], u kojoj se kaže „da se operacijska istraživanja bave matematičkim modeliranjem realnih procesa u svrhu donošenja optimalnih odluka““. (Lukač, Neralić, 2012, str 1). Operacijska istraživanja koriste se kako bi omogućila i pripomogla kod donošenja odluka unutar nekog sustava kako bi taj sustav bio što efikasniji. Koriste matematičke modele za optimizaciju tih sustava. Rješavanje problema kod operacijskih istraživanja provodi se u 4 faze po Lukač i Neralić (2012.) :

1. **Prikupljanje podataka za formulaciju realnog problema koji treba riješiti.** U prvoj fazi prikupljaju se podaci od strane formiranog tima koji dobro poznaje zadani problem. Veliki je naglasak na ciljeve koji se žele ostvariti. Isto tako veliku ulogu kod formuliranja problema imaju pretpostavke i uvjeti unutar kojih je zadani problem. Kod prikupljanja podataka isto tako pozornost se obraća i na buduće parametre po kojima će se poslije zadani problem rješavati. Zato je bitno da tim koji prikuplja podatke je upoznat s područjem problema kako bi za vrijeme prikupljanja podataka imali na umu ciljeve poduzeća te zadane parametre koji će se kasnije koristiti kod rješavanja problema.
2. **Formulacija odgovarajućeg matematičkog modela.** U drugoj fazi formulira se matematički model koji opisuje stvarni model što je točnije moguće. Taj matematički model ima realno značenje i zadanu funkciju cilja koja je određena ograničenjima. Ovdje dolazimo do pojma linearnog programiranja u slučaju da su za matematički model ispunjene pretpostavke linearnosti pri čemu je funkcija cilja linearna, a ograničenja su linearne jednadžbe ili nejednadžbe a da pritom varijable imaju uvjet nenegativnosti.
3. **Rješavanje modela, odnosno problema matematičkog programiranja.** U trećoj fazi pronalazi se optimalna metoda za rješavanje modela na računalu uz korištenje odgovarajuće programske podrške. Ako je naš matematički model problem linearnog programiranja koristimo se simpleks metodom i odgovarajućim programom. U slučaju da ne postoji odgovarajuće programsko rješenja, izrađuje se.
4. **Implementacija dobivenog rješenja.** U posljednjoj fazi dobiveno rješenje se implementira za korištenje donošenja odluka u daljnjim postupcima unutar zadanog sustava. Dobiveno matematičko rješenje ne mora nužno biti realizirano, ali može uvelike doprinijeti kod donošenja odluka. Ako dobiveno

rješenje nije zadovoljavajuće, dolazi do potrebe za poboljšanjem matematičkog modela i ponavljanja jedne ili više faza dok se ne dobije prihvatljivo rješenje.

Dakle linearno programiranje ili linearna optimizacija je najbitnija metoda za rješavanje problema unutar operacijskih istraživanja. Služi za pronalaženje optimalnog rješenja kod nekog problema kako bi se maksimizirao profit ili minimalizirali troškovi. Kod svakog problema optimizacije rješenje se pronalazi unutar zadanih ograničenja i uvjeta tog sustava. Ako rješavamo linearni problem za maksimum, cilj je pronaći rješenje kod kojeg je profit, tj. output maksimiziran. Kod linearnog problema za minimum tražimo optimalno rješenje kod kojeg su troškovi, tj. inputi minimizirani.

Po autorima Kalpić, D. i Mornar V. linearno programiranje objašnjeno je kao:

Samo linearno programiranje (LP) je specijalan slučaj matematičkog programiranja, definiran kao: min/max $f(\underline{x})$ uz $g_i(\underline{x}) \leq 0$; i=1,...,m

$$h_j(\underline{x}) = 0 \text{ ; } j=1,...,k$$

gdje je $\underline{x}$ vektor od n komponenti x_1, x_2, ... x_n, a g_1,..., g_n, i h_1,... h_k, jesu funkcije definirane u n-dimenzionalnom euklidskom prostoru. Kod linearnog programiranja za zadanu realnu linearnu funkciju od n strukturnih varijabli traži se ekstrem (minimum ili maksimum), uz uvjet da bude zadovoljeno m+k linearnih ograničenja postavljenih na strukturne varijable, a formuliranih u obliku linearnih jednadžbi i nejednadžbi. (Kalpić D., Mornar V., 1996, str 5.)

3.1. Primjena linearnog programiranja

Po Fergusonu i Sargentu (1964.) (kao što citira Loomba) linearno programiranje koje je relativno nova metoda već je demonstrirala svoju vrijednost kao pomagalo donošenja odluka u poslovanju, industriji i vladi. Kao neke primjere vrsta problema koje je moguće riješiti linearnim programiranjem navode određivanje rasporeda unutar postrojenja poduzeća, distribuciju robe i optimalne mješavine proizvoda te raspodjelu rada i drugih resursa. „Ukratko, linearno programiranje je metoda određivanja optimalnog proizvoda međuovisnih aktivnosti s obzirom na raspoložive resurse.“(N. Paul Loomba, 1964, str 1).

Danas linearno programiranje uvelike olakšava donošenje odluka kod problema proizvodnje proizvoda unutar nekog poduzeća s ograničenim resursima. Na primjer zamislimo da neko poduzeće X proizvodi 3 proizvoda koja ćemo nazvati A, B i C. Unutar proizvodnog pogona tog poduzeća postoje 3 radne stanice koje ćemo nazvati stanica 1, stanica 2 i stanica 3. Isto tako sva 3 proizvoda A, B i C zahtijevaju određeno vrijeme obrade

po radnoj stanici od kojih svaka ima vremensko ograničenje. Kao što vidimo iz tablice proizvodnja svakog proizvoda razlikuje se po potrebnom vremenu obrade po radnoj stanici i neto prihodu po jedinici proizvoda za poduzeće. Kod ovakvih problema odluke se mogu donijeti korištenjem simpleks metode linearnog programiranja. Pomoću simpleks metode za ovakav problem nizom matematičkih operacija dolazimo do optimalnog rješenja iz kojeg vidimo koji proizvod je najisplativiji za poduzeće X s obzirom na utrošak resursa, u ovom primjeru vremensko ograničenje po radnoj stanici, i/ili donosi najveći profit poduzeću po jedinici proizvoda. Ova tablica je prikazana samo kao vizualno pomagalo za predodžbu problema, a kasnije u radu objašnjeni su načini rješavanja sličnih problema koristeći simpleks metodu linearnog programiranja. Ovakav jednostavni problem možemo prevesti u problem raspodjele rada unutar poduzeća, problem alokacije resursa ili transporta i mnoge druge.

Tablica 1. Primjer proizvodnje poduzeća X

Radna stanica	Proizvod			Vremensko ograničenje po radnoj stanici
	A	B	C	
Stanica 1	5	5	1	15
Stanica 2	4	2	7	20
Stanica 3	1	4	1	24
Net prihod po jedinici proizvoda	5	7	10	

Kod linearnog programiranja simpleks metodom svaki problem je standardni ili opći problem za maksimum ili minimum. Pošto se ne može računati sa standardnim ili općim problemom, moguće je svaki pretvoriti u kanonski koji se kasnije koristi za rješavanje problema simpleks algoritmom.

3.1.1. Standardni, opći i kanonski problem linearnog programiranja

S matematičkog gledišta problemi linearnog programiranja svode se na standardni, opći i kanonski problem. Isto tako sva tri problema mogu biti problem za maksimum gdje je cilj maksimalni profit ili problem za minimum gdje je pozornost usmjerena na minimalizaciju troškova. Uzmimo za primjer tablicu iz prethodnog poglavlja. Elementi u sredini tablice su tehnički koeficijenti koje označavamo s a_{ij}. Strukturne varijable, tj. proizvode označavamo s x_j. Sama funkcija cilja ima oznaku Z. Cijena j-tog proizvoda ili komponente je označena s c_j. Kapaciteti ili resursi označeni su s $b_{i,}$ dok s n označujemo broj proizvoda a s m broj ograničenja. Isto tako imamo i uvjete nenegativnosti koje možemo označiti kao $x \geq 0$. Iz toga slijedi da funkcija cilja prema Babiću (2010.) glasi:

$$Opt. \sum_{j=1}^{n} c_j x_j$$

A skup ograničenja na strukturne varijable :

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

Isto tako zadani problem možemo prikazati u vektorskom i matričnom obliku zapisa. Po Babiću (2010) problem matematičkog programiranja može se definirati na sljedeći način:

$$Max(Min)\{f(x_1, x_2, \dots, x_n) | X \in S\}$$

Što znači da se zapravo radi o određivanju maksimuma ili minimuma neke funkcije koja ima n varijabli $f(x_1, x_2, \dots, x_n)$, gdje X predstavlja vektor iz euklidskog vektorskog prostora R^n kojemu su te varijable komponente, tj.

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Vektor X pripada nekom skupu S koji smo definirali ograničenjima zadanim u početnom problemu, a on predstavlja skup mogućih rješenja problema, te zato za skup S vrijedi da je $S \subseteq R^n$. Prema Periću(2017.) skup dopustivih rješenja je dobiven presjekom hiperravnina, tj. presjekom skupova svih točaka svih poluprostora (x_1, x_2, …, x_n). Problem je problem linearnog programiranja ako je $f(x_1, x_2, \dots, x_n)$ linearna funkcija i ako su ograničenja skupa S linearna. Prema Periću(2017.) ako neku točku dužine koja spaja točke x_1 i x_2 izrazimo kao $x = \lambda x_1 + (1 - \lambda) x_2$ uz uvjet $0 \leq \lambda \leq 1$ onda za tu linearnu kombinaciju vrijedi da je ta kombinacija konveksna kombinacija dviju točaka. Skup dopustivih rješenja S u linearnom programu ima osobine konveksnog skupa točaka. Na konveksnom skupu S postoje ekstremne i neekstremne točke. Ako je broj ekstremnih točaka (u ekstremnoj točci P u nekom skupu S vrijednost funkcije poprima ekstremnu vrijednost.) skupa konačan onda taj skup nazivamo konveksnim poliedrom. Prema Neraliću (2016) neka točka T skupa S je ekstremna točka tog skupa, ako za svaku dužinu kojoj pripada točka T u skupu S vrijedi da je točka T ujedno i krajnja točka te dužine. U ekstremnim točkama skupa nalazi se optimalno rješenje funkcije. Standardni problem maksimuma kod kojeg su sva ograničenja na skup S tipa ≤ možemo zapisati kao:

$$Max \sum_{j=1}^{n} c_j x_j$$

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

$$x_j \geq 0, j = 1,2, \dots, n$$

Što nam govori da standardni problem maksimuma linearnog programiranja ima n varijabli i m ograničenja koja su sva tipa ≤. Za matrični prikaz problema prema Babiću (2010) potrebno je uvođenje sljedećih matrica i vektora:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, C = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}, A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Što možemo zapisati kao:

$$MaxC^T X$$

uz uvjete:

$$AX \leq B$$

$$X \geq 0$$

Kompletni problem proizvodnje za prethodno navedeni primjer prema Babiću (2010) ima oblik:

$$Z = 5x_1 + 7x_2 + 10x_3 \rightarrow max$$

$$5x_1 + 5x_2 + 1x_3 \leq 15$$

$$4x_1 + 2x_2 + 7x_3 \leq 20$$

$$1x_1 + 4x_2 + 1x_3 \leq 24$$

$$x_1, x_2, x_3 \geq 0$$

Ovako zadani problem naziva se standardni problem maksimuma linearnog programiranja. Kao što je već prije spomenuto, simpleks metoda računa s kanonskim oblikom problema, zato je potrebno standardni problem transformirati u kanonski. Kanonski problem se od standardnog razlikuje po tome da kod uvjeta nejednadžbe zamijenimo odgovarajućim jednadžbama, osim uvjeta nenegativnosti. Kako bismo standardni problem pretvorili u kanonski prema Martiću(1979) potrebno je nejednadžbu $AX \leq B$ zamijeniti s jednadžbom $AX + IU = B$ uz dodatni uvjet $U \geq 0$. Varijable vektora U koje su dodane zovu se oslabljene varijable.

Uz standardni i kanonski problem postoji i opći problem kod linearnog programiranja. Kao što je prije spomenuto, kod standardnog problema svi uvjeti su nejednadžbe koje kako bismo preveli u kanonski svodimo na jednadžbe. Kod općeg problema uvjeti ne moraju nužno biti nejednadžbe, već mogu biti i jednadžbe. Isto tako standardni problem za maksimum može kao uvjete imati nejednadžbe tipa veće ili jednako (≥), a isto tako i nejednadžbe tipa manje ili jednako (≤). Analogno vrijedi i za opći problem za minimum. Prema Martiću (1979) ako s A^i označimo i-ti redak – vektor od A i s M i N skupove $M = \{1,2,\cdots,m\}, N = \{1,2,\cdots,n\}$. Neka vrijedi za skup S⊆M i C(S)=M\S i T⊆N i C(T)=N\T onda matrični zapis općeg problema glasi:

$$Max\ C^T X$$

uz uvjete:

$$x_j \geq 0, j \in T$$
$$A^i X \leq b_i, i \in S$$
$$A^i X = b_i, i \in C(S)$$

Kako bi bilo moguće riješiti opći problem, potrebno ga je prvo prevesti u kanonski oblik zapisa. Matričnom obliku zapisa $MaxC^T x$ dodaje se manjak i artificijelne varijable. Na nejednadžbu $A^i x \leq b_i$ potrebno je dodati manjak *U* kako bi bilo moguće istu preoblikovati u jednadžbu uz uvjet da je $x, U \geq 0$. Iako je $A^i x = b_i$ već u obliku jednadžbe, na nju se dodaje artificijelna varijabla *W* uz uvjet da je $x, W \geq 0$. Pa matrični oblik zapisa izgleda:

$$MaxC^T x$$
$$A^i x + U = b_i$$
$$A^i x + W = b_i$$
$$X, U, W \geq 0$$

3.1.2. Dual kod problema linearnog programiranja

Kod linearnog programiranja zadani problem nazivamo originalnim problemom ili primalom, no isto tako postoji i dualni problem ili dual. Dual primala je novi problem linearnog programiranja koji se dobiva iz primala. Dual se formira iz podataka iz primala tako da svaka varijabla u primalu postaje ograničenje duala, dok svako ograničenje primala postaje varijabla u dualu. Dual je zapravo inverzna funkcija primala što znači da ako je problem primala standardi problem za maksimum onda je njegov dual standardni problem za minimum. Uzmimo za primjer primala prethodni zapis standardnog problema za maksimum i njegov dual.

Tablica 2: Usporedba primala i duala

Primal	Dual
$Max \sum_{j=1}^{n} c_j x_j$	$Min \sum_{i=1}^{m} y_i b_i$
$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, i = 1,2,\dots,m$	$\sum_{i=1}^{m} y_i a_{ij} \geq c_j, j = 1,2,\dots,n$
$x_j \geq 0, j = 1,2,\dots,n$	$y_i \geq 0, i = 1,2,\dots,m$
$Max\ C^T X$	$Min\ Y^T B$
$AX \leq B$	$Y^T A \geq C^T$
$X \geq 0$	$Y \geq 0$

Kao što je vidljivo iz tablice, po Babiću (2010.) ograničenja primala *m* u dualu postaju varijable *m*, a varijable *n* iz primala u dualu postaju ograničenja *n*. Dual je inverzna funkcija

primala pa je zato dual duala ponovno primal, tj. originalni problem. Kod pretvorbe primala u dual dodaje se novi vektor $Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$. Kako bi bilo moguće iz primala dobili dual potrebno je samo desnu stranu ograničenja originalnog problema pomnožiti s novim vektorom varijabli *Y.* Naime ograničenja u dualu dobiju se tako da se redom množe varijable počevši od prve iz primala s y_i. Svaka varijabla iz primala transformira se u odgovarajuće ograničenje u dualu. Desna strana ograničenja u primalu isto tako postaju određeni koeficijenti duala, redom od prvog uz x_i do zadnjeg.

Tablica 3: Zapis primala i duala

Primal	Dual
$Z = 5x_1 + 7x_2 + 10x_3 \rightarrow max$	$Z^d = 15y_1 + 20y_2 + 24y_3 \rightarrow min$
$5x_1 + 5x_2 + 1x_3 \leq 15$	$5x_1 + 4x_2 + 1x_3 \geq 5$
$4x_1 + 2x_2 + 7x_3 \leq 20$	$5x_1 + 2x_2 + 4x_3 \geq 7$
$1x_1 + 4x_2 + 1x_3 \leq 24$	$1x_1 + 7x_2 + 1x_3 \geq 10$
$x_1, x_2, x_3 \geq 0$	$y_1, y_2, y_3 \geq 0$

Analogno tome ako bismo radili dual duala primala, dobili bismo početni primal. Isto tako dual nije potrebno posebno rješavati jer se njegovo rješenje dobije iz rješenja primala kod korištenja simpleks algoritma.

3.2. Simpleks algoritam

„Simpleks metoda je jedna od najpoznatijih metoda za rješavanje problema linearnog programiranja. Autor simpleks metode je G. Dantzig, koji veliki dio zasluga za temeljne ideje sam pripisuje J. Von Neumanu.“(Babić, 2010, str 121). Simpleks metoda otkrivena je između kasnih 1940-ih i ranih 1950-ih godina. Simpleks metoda sastoji se od više koraka koji se ponavljaju kako bi se došlo do optimalnog rješenja zadanog problema ako ono postoji, te je ona zato iterativna metoda. Kod algoritma simpleks metode postoje 4 glavna koraka a to su:

1. **Formuliranje nekog početnog mogućeg rješenja.**
2. **Testiranje rješenja kako bi se odredilo da li je ono i optimalno.**
3. **Ako početno rješenje nije optimalno, zadaju se upute kako doći do boljeg rješenja.**
4. **Nakon konačno mnogo koraka utvrđuje se da li rješenje postoji i ono je optimalno ili uopće ne postoji.**

Prema Martiću (1979) kod simpleks metode potrebno je riješiti sustav jednadžbi $A_1x_1 + A_2x_2 + \cdots + A_nx_n = B$, gdje su vektori A_1, A_2,…, A_n stupci matrice A. Problem je u tome da se

vektor B izražava kao kombinacija vektora A_j, (j=1, 2, …, n). U početku nije poznato kako B izraziti u terminima vektora A_j, pa se uvode jedinični vektori $I_1, I_2, \ldots, I_m$ koji čine bazu prostora R^m. Za B pišemo:

$$B = I_1 b_1 + I_2 b_2 + \cdots + I_m b_m$$

Potrebno je jedinične vektore zamijeniti vektorima A_j i ta se operacija nastavlja sve dok se ne zamijene svi jedinični vektori. Ako to nije moguće tada problem nema rješenje ili se vektor B može izraziti s manje od m vektora A_j (ako ta pretpostavka vrijedi, dolazi do degeneracije). Simpleks algoritam samo može raditi s kanonskim problemom koji se dobije iz svakog standardnog problema. Rješavanjem standardnog problema koji je preveden u kanonski i pronalaženjem optimalnog rješenja za kanonski problem ujedno pronalazimo i optimalno rješenje za početni standardni problem. Dopunske varijable, tj. jedinični vektori ulaze u funkciju cilja s koeficijentom nula jer ne smiju utjecati na vrijednost funkcije, tj. ne doprinose nikakav prihod i koeficijenti koji se dodaju tim varijablama u početnoj tablici su 1. Kada je standardni problem pretvoren u kanonski postavlja se početna simpleks tablica čiji je cilj pronalaženje bazičnog mogućeg rješenja.

Tablica 4: Početna tablica za problem maksimuma

$c_j \rightarrow$			c_1	c_2	…	c_s	…	c_n	0	0	…	0	…	0
c_s	Bazično rješenje		Strukturalne varijable						Dopunske varijable					
↓	Varijabla	Količina	x_1	x_2	...	x_s	...	x_n	u_1	u_2	...	u_r	...	u_m
0	u_1	a_{10}	a_{11}	a_{12}	...	a_{1s}	...	a_{1n}	1	0	...	0	...	0
0	u_2	a_{20}	a_{21}	a_{22}	...	a_{2s}	...	a_{2n}	0	1	...	0	...	0
⋮	⋮	⋮	⋮	⋮		⋮		⋮	⋮	⋮		⋮		⋮
0	u_r	a_{r0}	a_{r1}	a_{r2}	...	a_{rs}	...	a_{rn}	0	0	...	1	...	0
⋮	⋮	⋮	⋮	⋮		⋮		⋮	⋮	⋮		⋮		⋮
0	u_m	a_{m0}	a_{m1}	a_{m2}	...	a_{ms}	...	a_{mn}	0	0	...	0	...	1
	Z_j-c_j	0	- c_1	- c_2	…	- c_s	…	- c_n	0	0	…	0	…	0

(Prema: Dobrenić, 1978.)

Prema Dobreniću (1978.) simpleks tablica u prvom redu zaglavlja sadržava koeficijente varijabli c_j iz jednadžbe cilja a u prvom stupcu c_s koeficijente iz jednadžbe cilja varijabli koje se nalaze u početnom bazičnom rješenju. U drugom stupcu su varijable iz bazičnog rješenja i njihove vrijednosti u trećem stupcu. U drugom stupcu početne simpleks tablice nalaze se dopunske varijable čije se vrijednosti koje su jednake ograničenjima b_i nalaze u trećem stupcu. Vrijednost trećeg stupca na početku je 0 jer je vrijednost strukturnih varijabli na početku jednaka 0. Isto tako za lakše razumijevanje daljnjih koraka umjesto oznaka b_i koristit će se oznake $a_{i0.}$ Središnji dio početne tablice je matrica input-output koeficijenata a_{ij} strukturnih varijabli, a desni dio tablice je jedinična matrica dopunskih varijabli. Zadnji red u

tablici Z_j-c_j je red kriterija optimalnosti. Relativni troškovi u tom redu predstavljaju povećanje ili umanjenje za *Z*.

Na prvu tablicu dodana su još dva stupca K i R. Stupac K je stupac kontrole gdje provjeravamo točnost izračuna nakon svake transformacije tablice. Stupac R je stupac u koji se upisuju kvocijenti količina u bazičnom rješenju i pozitivnih koeficijenata a_{ij}. U stupcu R tražimo najmanji od izračunatih kvocijenata. Postavljanjem početnog bazičnog rješenja završena je prva faza simpleks algoritma. Prelazi se na drugu fazu jer prvo rješenje nije prihvatljivo pošto se sastoji samo od dopunskih varijabli. U drugoj fazi poboljšava se simpleks algoritmom rješenje i pokušava se doći do optimalnog rješenja. U drugoj fazi ponavljaju se iteracije koje se sastoje od tri koraka prema Dobreniću (1978.):

1. Određivanje vodećeg stupca
2. Određivanje vodećeg retka
3. Transformacija koeficijenata tabele
    a. Transformacija koeficijenata vodećeg reda
    b. Transformacija koeficijenata ostalih redova

Dobrenić (1978.) govori da u prvom koraku prvo se testira da li vrijedi $(Z_j - c_j) \geq 0, \forall j, j = 1,2,\dots,n$, tj. da li su sve vrijednosti u redu Z_j-c_j veće ili jednake nuli. Ako ta pretpostavka vrijedi, dobiveno rješenje je optimalno i problem se ne rješava dalje. Ako ne vrijedi zadana pretpostavka, to znači da postoje vrijednosti $c_j' < 0$ što znači da se program može dalje optimizirati. Prvo se određuje vodeći stupac. Za vodeći stupac uzima se stupac $c_j' < 0$ koji po apsolutnoj vrijednosti ima najveću vrijednost, $\max\{|c_j'|\}$ i taj vektor se uvodi u bazu. U navedenom primjeru najveću negativnu vrijednost poprima stupac x_2 pa je to vektor koji se odabire za vodeći stupac i ulazi u bazu.

U drugom koraku određuje se vodeći red, tj. određuje se vektor koji izlazi iz baze. Prema Dobreniću (1978.) prvo u vodećem stupcu provjeravamo da li za sve input-output koeficijente vrijedi da su manji ili jednaki nuli. Ako da, to znači da ne postoji konačni maksimum. Ako ne vrijedi pretpostavka $a_{is} \leq 0$ onda se traži $min\left\{\frac{a_{i0}}{a_{is}}\right\}, \forall\, a_{is} > 0$. Dakle računamo koeficijente koje zapisujemo u stupac *R* a računaju se kao omjer količine bazičnog rješenja i strukturne varijable. Ako je minimalni koeficijent od izračunatih $\frac{a_{r0}}{a_{rs}}$ tada je vodeći red *r*, a broj a_{rs} nazivamo ključni broj ili pivot. Iz baze izlazi vektor za koji je taj koeficijent minimalan, a ako minimalan koeficijent nije jedinstven onda dolazi do degeneracije.

Sada dolazi do prve iteracije, tj. nove simpleks tablice. Nova simpleks tablica popunjava se po trećem koraku, transformacija koeficijenata tabele a_{ij} u koeficijente a'_{ij}. Prema Dobreniću (1978.) prvo transformiramo vodeći stupac tako da postaje jedinični stupac tako da pivot poprimi vrijednost 1 a svi ostali u stupcu 0. Potom transformiramo koeficijente vodećeg reda

tako da svaki element vodećeg reda podijelimo s pivotom, tj. za $i = r, a'_{ij} = \frac{a_{ij}}{a_{rs}}, (j = 0,1,2, \dots, n+m)$.

Kada smo transformirali vodeći red i stupac prelazimo na transformaciju koeficijenata ostalih redova. Dobrenić (1978.) govori da koeficijente ostalih redova transformiramo po pravilu za $i \neq r, a'_{ij} = a_{ij} - a'_{rj} * a_{is},\ (j = 0,1,2, \dots, n+m)$. Dakle nove vrijednosti računaju se tako da se od stare vrijednosti oduzme umnožak nove vrijednosti iz vodećeg retka u istom stupcu i stare vrijednosti elementa u vodećem stupcu i istom retku. Isto pravilo vrijedi i za red *Z-c*. Radi lakšeg popunjavanja simpleks tablice vrijede pravila ako u vodećem stupcu u prethodnoj tablici postoji vrijednost 0, vrijednosti tog reda se prepisuje te analogno vrijedi i za vodeći red. Nakon prve transformacije prije nego se nastavi s računanjem sada je moguće pomoću neobaveznog stupca K, tj. kontrole provjeriti da li je u prethodnim koracima sve dobro izračunato. Ako je vrijednost u stupcu K koja je izračunata po formuli jednaka vrijednosti koja se dobije zbrajanjem elemenata u redu onda to znači da je prethodni korak transformacije elemenata točan. Nakon svake iteracije provjerava se da li je rješenje optimalno, tj. dali vrijedi $Z_j - c_j \geq 0$. Na isti način rješava se i problem za minimum samo je razlika u tome da u bazu ulazi vektor za koji vrijedi da je $(z_j - c_j) > 0$,a optimalno rješenje je dobiveno u iteraciji za koju vrijedi $(z_j - c_j) \leq 0$, tj. u zadnjem redu u tablici ne postoje pozitivna rješenja. Iz rješenja primala pomoću simpleks algoritma moguće je odrediti i rješenje duala istog problema. Rješenje dualne funkcije optimalno je i isto je kao i rješenje funkcije primala, dakle $Z^d = Z$, a vrijednosti varijabli čitaju se ispod dopunskih varijabli u zadnjem redu tablice.

3.3. Druge metode rješavanja problema linearnog programiranja

Osim simpleks metode postoje i druge metode rješavanja problema linearnog programiranja. Postoji grafička metoda koja se koristi kod problema linearnog programiranja ako problem ima samo dvije varijable i transportni problem kada je potrebno prevesti istovrsni teret do odredišta iz ishodišta uz najmanje moguće troškove.

3.3.1. Grafička metoda

Kao što je već spomenuto, grafička metoda se koristi kada problem linearnog programiranja koji se rješava ima samo dvije varijable. Grafička metoda pobliže je objašnjena na sljedećem primjeru:

$$Z = 2x_1 + 4x_2 \rightarrow max$$

$$3x_1 + 6x_2 \leq 21$$
$$10x_1 + 5x_2 \leq 35$$
$$x_1, x_2 \geq 0$$

Skup točaka koje zadovoljavaju zadane nejednadžbe nazivamo skup mogućih rješenja. Iz uvjeta nenegativnosti $x_1, x_2 \geq 0$ očito je da je rješenje u prvom kvadrantu koordinatnog sustava uključujući ishodište i pozitivne dijelove koordinatnih osi. Skup rješenja, tj. ostale nejednadžbe nalaze se između navedenih osi. Uzimamo pravac po pravac, određujemo nultočke ili bilo koje dvije točke koje leže na tom pravcu i nacrtamo ga spajajući točke.

Za prvu nejednadžbu određene su dvije točke:

$$A\left(0, \frac{7}{2}\right), B(7,0)$$

Analogno kao s prvim pravcem određuju se dvije točke i na drugom pravcu:

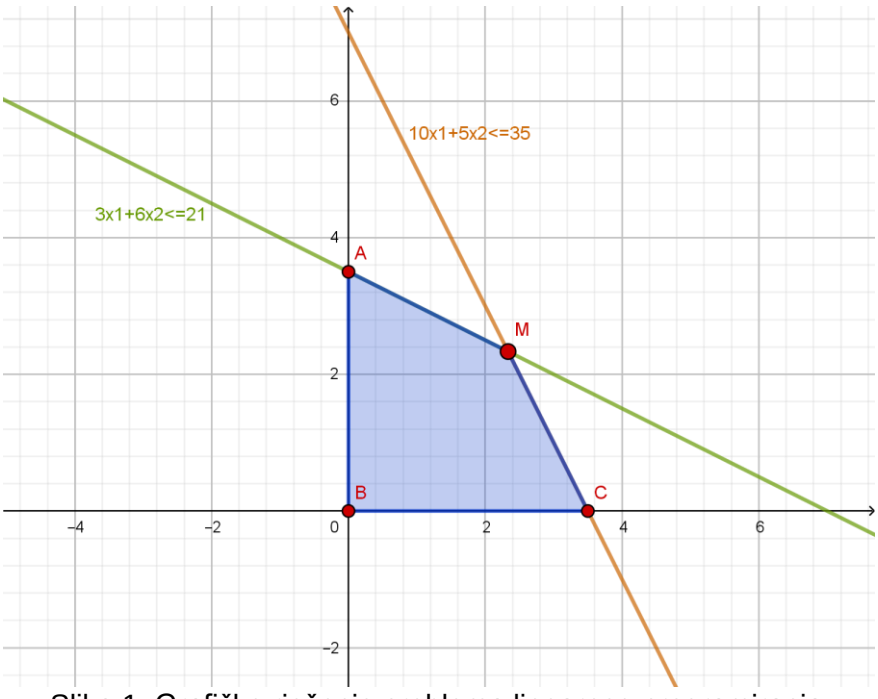
$$C(0,7), D\left(\frac{7}{2,0}\right)$$

Potom izjednačavamo obadva pravca kako bismo odredili točku u kojoj se ta dva pravca sijeku:

$$M\left(\frac{7}{3}, \frac{7}{3}\right)$$

Pravci se sijeku u točci M s koordinatama $M(\frac{7}{3}, \frac{7}{3})$. Vrijednosti točke M, tj. koordinate uvrštavaju se u funkciju cilja kako bi se dobilo optimalno rješenje.

$$Z = 1$$



Slika 1: Grafičko rješenje problema linearnog programiranja

Iz grafa vidljivo je sjecište točaka, tj optimalno rješenje, a područje mogućih rješenja je zajedničko područje ispod pravaca pobojano plavom bojom.

3.3.2. Problem transporta

„**Problem transporta** (transportation problem) jedan je od prvih problema formuliranih u terminima linearnog programiranja.“(Lukač, Neralić, 2012, str, 31). „Klasični transportni problem javlja se kao problem linearnog programiranja, gdje je potrebno prevesti homogeni (istovrsni) teret iz skupa ishodišta u skup odredišta uz najniže moguće troškove.“ (Babić, 2010, str, 229). Kako bi bilo moguće formulirati problem transporta i navesti njegova svojstva pretpostavimo da postoji *m* ishodišta, tj. centara ponude robe koju je potrebno prevesti u *n* odredišta, tj. centara potražnje. Ishodišta *m* imaju određenu količinu ponude robe koju označavamo s a_i gdje je $i = 1,2,3, \dots, m$. Isto tako odredišta *n* imaju određenu količinu potražnje robe koju označavamo s b_j gdje je $j = 1,2,3, \dots n$. S x_{ij} označavamo količinu tereta koji se prevozi iz i-tog ishodišta do j-tog odredišta uz pretpostavku da je ukupna količina ponude jednaka ukopnoj količini potražnje i da vrijedi da je $a_i, b_i > 0$. Ako pretpostavka ne vrijedi onda je to otvoreni transportni problem koji kako bi ga bilo moguće pretvoriti u zatvoreni uvodimo fiktivno ishodište ili odredište, ovisno o problemu, koje ima potrebnu količinu potražnje odnosno ponude. Zbog toga takav se model transporta ujedno naziva zatvorenim modelom jer prema Babiću (2010) vrijedi :

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$$

Isto tako mora vrijediti pretpostavka da je roba koja se transportira homogena te da se prevozi direktno od odredišta do ishodišta bez pretovara. S c_{ij} označava se konstantni trošak transporta po jedinici tereta iz i-tog ishodišta u j-to odredište koji je linearan te za pošiljku mora vrijediti $c_{ij} * x_{ij}$. Troškovi transporta c_{ij} mogu biti pozitivni, negativni ili jednaki nuli (u slučaju kada se uvode fiktivna ishodišta ili odredišta). Sam problem transporta sastoji se u tome da se odrede nepoznate količine robe x_{ij} koja se prevozi od ishodišta do odredišta uz uvjet da se preveze sva roba iz ishodišta i da se zadovolje sve potrebe odredišta uz minimalne ukupne troškove transporta. Babić(2010) navodi da formulacija transportnog problema glasi:

$$Min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

s ograničenjima:

$$\sum_{j=1}^{n} x_{ij} = a_i, i = 1,2, \dots, m$$

$$\sum_{i=1}^{m} x_{ij} = b_j, j = 1,2, \dots, n$$

$$x_{ij} \geq 0, \forall\, i, j$$

Transportni problem obično se zapisuje u obliku tablice.

Tablica 5: Početna tablica za transportni problem

Ishodišta	Odredišta			a_i
	O_1	O_2	O_3	
I_1				
I_2				
I_3				
b_j				

(Prema Dobrenić, 1978.)

Transportni problem rješava se u dva koraka. U prvom koraku određuje se početno bazično rješenje dok se u drugom koraku to rješenje poboljšava. Za određivanje početnog bazičnog rješenja postoji više metoda:

1. Metoda sjeverozapadnog kuta
2. Metoda minimalnih troškova
3. Vogelova metoda
4. Kötzigova metoda

Isto tako za poboljšavanje i testiranje optimalnosti rješenja postoji više metoda:

1. Metoda skakanja s kamena na kamen
2. Modificirana metoda distribucije (MODI)

Metoda sjeverozapadnog kuta ili dijagonalna metoda daje početno bazično nedegenerirano rješenje. Prema Lukač i Neralić (2012) kod metode sjeverozapadnog kuta tablica se počinje puniti od prvog gornjeg lijevog (sjeverozapadnog) kuta u kojem se nalazi polje (1,1) i kreće se prema desnom donjem (jugoistočnom) kutu u kojem se nalazi polje (m,n). U prvo moguće polje unosi se maksimalna količina tereta ovisno o ponudi prvog ishodišta ili potražnji prvog odredišta $x_{11} = min\{a_1, b_1\}$. Ako vrijedi $x_{11} = a_1$ time je iscrpljena ponuda ishodišta 1 ili ako vrijedi $x_{11} = b_1$ zadovoljena je potražnja odredišta 1. Ako

je $a_1 > b_1$, $x_{11} = b_1$ te $x_{i1} = 0$ nastavljamo se kretati po retku 1 i uzimamo prvo iduće slobodno polje $x_{12} = \min\{a_1 - b_1, b_2\}$ i tako sve dok se ne zadovolje uvjeti, tj. ne iscrpi se u potpunosti ponuda ishodišta 1. Ako pak vrijedi da je $b_1 > a_1$, $x_{11} = a_1$ te $x_{1j} = 0$ nastavljamo se kretati po stupcu 1 i uzimamo prvo iduće slobodno polje $x_{21} = \min\{b_1 - a_1, a_2\}$ i tako sve dok se ne zadovolje potrebe, tj. potražnja odredišta 1. Kada je prvi korak završen, u drugom se opet kretnja nastavlja na isti način. Pokušavamo zadovoljiti potražnju odredišta ili iskoristiti ponudu ishodišta. U slučaju da vrijedi $a_1 = b_1$ dolazi do degeneracije početnog bazičnog rješenja i na kraju nije zadovoljeno $m + n - 1$ ograničenja. Ako kroz zadatak nije došlo do degeneracije, na kraju je zadovoljeno $m + n - 1$ ograničenja, tj. pozitivnih vrijednosti x_{ij} i postavljeno je bazično početno rješenje. No najveći problem kod metode sjeverozapadnog kuta je taj da ona ne vodi računa o troškovima transporta te to početno bazično rješenje može biti jako daleko od optimalnog rješenja problema.

Metoda minimalnih troškova uzima u obzir troškove transporta i početno bazično rješenje bliže je optimalnom rješenju nego kod metode sjeverozapadnog kuta. Lukač i Neralić (2012) navode da se u tablici prvo traži polje s najmanjim troškovima i u njega se upisuje maksimalni mogući teret s obzirom na potrebe odredišta i kapacitete ishodišta. To polje nalazi se u stupcu *s* i retku *r* pa za to polje vrijedi $x_{rs} = \min\{a_r, b_s\}$. Potom kao i u metodi sjeverozapadnog kuta ispušta se redak ili stupac, ovisno o tome da li su zadovoljene potrebe odredišta ili je iscrpljena ponuda ishodišta. Isti postupak ponavlja se na preostalim poljima locirajući najjeftinija i na njih se postavlja najveći mogući teret sve dok nisu zadovoljene sve potrebe, tj. sav teret nije alociran. Ako u postupku dođe do $a_r = b_s$ to znači da je opet došli do degeneracije i u tom slučaju ispušta se ili redak r ili stupac s. Kada je sav teret alociran dobije se početno bazično rješenje koje je u odnosu na metodu sjeverozapadnog kuta bliže optimalnom rješenju.

Prema Lukač i Neralić (2012) te Babić (2010) Vogelova metoda isto tako kao i metoda minimalnih troškova vodi računa ne samo o alokaciji tereta po poljima već i o troškovima transporta. Vogelova metoda isto tako teret raspoređuje po poljima gdje su troškovi jeftiniji pa će bazično početno rješenje dobiveno ovom metodom biti bliže optimalnom od rješenja dobivenog metodom sjeverozapadnog kuta. Kod Vogelove metode potrebno je za svaki red i stupac matrice prvo izračunati pripadajuću razliku, tj. kazne za nekorištenje najjeftinije rute. Ta razlika je razlika između dva najmanja troška u tom stupcu ili retku i računa se tako da se prvo u promatranom stupcu ili retku poredaju troškovi po veličini od najmanjeg do najvećeg. Potom se za svaki stupac i redak oduzima najmanji trošak od drugog po veličini i dobivena vrijednost je tražena razlika za taj stupac ili redak. Ta razlika znači da ako se teret ne

transportira po najjeftinijoj ruti onda će se troškovi transporta povećati za tu razliku. Ta razlika može biti jednaka nuli ako je u jednom stupcu ili retku najmanji trošak isti na dva ili više polja. Od dobivenih razlika traži se najveća i potom se u tom stupcu ili retku odabire polje koje ima najmanji jedinični trošak i na njega se stavlja najveći mogući teret. Znači za polje koje se nalazi u retku *r* i stupcu *s* vrijedi $x_{rs} = \min\{a_r, b_s\}$. Potom se taj stupac ili redak ispušta i nastavlja se s postupkom ispočetka. Ako je najveća razlika ista u više redaka i/ili stupaca tada se može odabrati bilo koji od njih. Vogelova metoda računanja razlika ponavlja se sve dok ne preostane samo jedno polje za popuniti i na njega se onda stavlja odgovarajući teret s obzirom na ograničenja tog stupca ili reda. Kada je sav teret alociran postavljeno je početno bazično rješenje koje je u većini slučajeva jako slično optimalnom rješenju pa se zato i Vogelova metoda smatra najboljom metodom za određivanje početnog bazičnog rješenja.

Babić (2010) navodi još i Kötzigovu metodu koja je metoda koja se primjenjuje samo u posebnom slučaju transportnog problema kada vrijedi $m = 2$, tj. postoje dva ishodišta ili $n = 2$, tj. postoje dva odredišta. U slučaju da postoje dva ishodišta onda se za tu tablicu troškova računaju dva retka r_j, k_j gdje vrijedi $r_j = c_{1j} - c_{2j}$. Dakle u red r_j upisuje se razlika troškova za prvo i drugo ishodište. U red k_j upisuju se brojevi od 1 nadalje tako da se najmanjoj razlici u redu r_j pridoda broj 1 u redu k_j i tako redom. Teret se upisuje u polja tako da u prvo polje iznad najmanjeg broja u redu k_j dodamo najveći moguću teret a potom u polje ispod. Kada su zadovoljene sve potrebe tog stupca isti se ispušta i postupak se nastavljamo na idući po redu ovisno o brojevima u redu k_j. Analogno ovome postupku isto se radimo i u slučaju da postoje dva odredišta, dodaju se dva stupca.

Metoda skakanja s kamena na kamen i MODI metoda predstavljaju vrstu simpleks procedure gdje je potrebno izračunati sve razlike z_{ij}-c_{ij}. Potrebno je odrediti razlike samo za prazna polja u tablici jer je za bazične varijable ta razlika jednaka nuli. Optimalno rješenje se dobiva kada su sve razlike manje ili jednake nuli kao i kod simpleks algoritma, jer je transportni problem problem za minimum. Po Babiću (2010) prvo se postavlja bazično rješenje problema transporta nekom od prethodno navedenih metoda. Kada je početno bazično rješenje postavljeno potrebno je izračunati diferencije za preostale nebazične vektore (prazna polja) unutar tablice. Kod metode skakanja s kamena na kamen za svako prazno polje traži se zatvoreni put kojim se „skačući s kamena na kamen“ vraća na to polje tako da:

a. Prvo se traži polje u istom retku na kojeg se može skočiti na kamen u istom stupcu jer bilo koja dva uzastopna polja moraju biti u istom stupcu ili retku

b. Potom se kretnja nastavlja unutar tog stupca na kamen koji je u istom retku s kojeg je ponovno moguće skočiti na kamen u istom stupcu jer tri uzastopna polja ne smiju biti u istom stupcu ili retku
c. Kretnja se tako nastavlja sve dok se ne stigne na kamen koji je u istom stupcu ili retku kao i početno prazno polje za koje se računa diferencija

Nakon što je određen zatvoreni put za to polje potrebno je zbrojiti jedinične troškove po putu tako da prvo zauzeto polje na putu ima pozitivan trošak, drugo negativan i tako do zadnjeg polja. Isti postupak ponavlja se sve dok sve diferencije nisu $z_{ij} - c_{ij} \leq 0$ što znači da je rješenje optimalno. Isto tako provjerava se da li je rješenje degenerirano, tj. da li je broj alokacija jednak m+n-1.

Razlika između metode skakanja s kamena na kamen i MODI metode je ta da MODI metoda koristi i dualni problem problema transporta. Isto kao i za prethodnu metodu, kod MODI metode potrebno je početno bazično rješenje. Prema Babić (2010) u prvom koraku za svaki red i stupac računaju se dualne varijable u_i i v_j pri čemu one nemaju uvjet nenegativnosti. Kod MODI metode diferencije z_{ij}-c_{ij} računaju se po formuli $z_{ij} - c_{ij} = (u_i + v_j) - c_{ij}$. Kao što je već spomenuto za bazične varijable vrijedi $z_{ij} - c_{ij} = 0$ pa to povlači $0 = (u_i + v_j) - c_{ij} \rightarrow c_{ij} = (u_{ij} + v_{ij})$. Kako bi bilo moguće izračunati sve dualne varijable koje su potrebne uzima se da je $u_1 = 0$. Kada su izračunate sve dualne varijable, pomoću njih izračunaju se svi relativni troškovi za nezauzeta polja. Nakon toga postupak je isti kao i kod metode skakanja s kamena na kamen. Prvo se odabire polje s pozitivnim relativnim troškom za koje se traži zatvoreni put preko zauzetih polja i tako se seli teret po poljima. Postupak se ponavlja sve dok se ne dobije tablica koja gdje su sve diferencije $z_{ij} - c_{ij} \leq 0$ i time se dobiva optimalno rješenje. Isto iz MODI metode, tj. vrijednosti na marginama tablice moguće je ujedno odrediti optimalno rješenje dualnog problema po formuli:

$$T^* = \sum_{i=1}^{m} a_i u_i + \sum_{j=1}^{n} b_j u_j$$

4.Degeneracija i kruženje

Degeneracija je uobičajena pojava kod rješavanja problema linearnog programiranja neovisno o metodi koja se koristi. „O degeneraciji u linearnom programiranju govori se kada bazično rješenje ima manje od *m* varijabli različitih od nula. Grafički je degeneracija izražena u slučajevima kada neka restrikcija dotiče područje rješenja u nekom uglu, ali pri tome ne pridonosi određivanju vrijednosti rješenja.“(Barković, 2001., str 29.) Prema Tulsian i Tulsian (2012) degeneracija problema linearnog programiranja se događa kada početno bazično rješenje ima manje varijabla različitih od nule od broja neovisnih ograničenja. „Prilikom računanja simpleks-metodom može se desiti da se za izbor pivot stupca i pivot reda javi više jednako dobrih kandidata. U tom slučaju dolazi do degeneracije“ (Barković, 2001, str 29.) Kod transportnog problema degeneracija je relativno česta pojava no za razliku od simpleks metode kod transportnog problema ne dolazi do kruženja. Kod transportnog problema dolazi do degeneracije kada je broj x_{ij} manji od m+n-1. „Drugim riječima, ako degenerirano bazično rješenje postoji, tada je bar jedna parcijalna suma od a_i jednaka nekoj parcijalnoj sumi od b_j.“(Martić, 1979, str 193). Degeneracija je pobliže objašnjena u sljedećim potpoglavljima.

4.1. Degeneracija i kruženje kod simpleks metode

Degeneracija kod simpleks algoritma može se pojaviti na dva načina. Prema Martiću (1979.) bazično moguće rješenje $\{x_1,x_2,\ldots,x_m\}$ za neki problem linearnog programiranja je degenerirano, ako za bar jedan x_i vrijedi $x_i = 0$ za i=1,2,...,m. Što znači da je samo početno bazično rješenje već degenerirano jer ima manje od m pozitivnih vrijednosti bazčnih varijabli. Drugi slučaj pojave degeneracije kod problema nastaje kada se za dva ili više indeksa i dobije ista vrijednost za minimalni omjer θ_0 što znači da nije moguće jednoznačno odrediti vektor koji izlazi iz baze. Za primjer uzmimo da je u simpleks tabeli T izračunato:

$$\theta_0 = \frac{a_{10}}{a_{1s}} = \frac{a_{20}}{a_{2s}}$$

Nakon transformacije ako se za vodeći red r odabere r=1 u novoj tabeli dobivamo novo rješenje koje je degenerirano:

$$a'_{20} = a_{20} - \frac{a_{10}}{a_{1s}} * a_{2s} = 0$$

U novoj tablici pojava te nule znači da postoji mogućnost da za novu tabelu vrijedi $\theta_0 = 0$. Kod takvog slučaja novo rješenje neće se razlikovati od prethodnog programa po vrijednosti funkcije te nova iteracija ne poboljšava program. Isto tako ta situacija može se ponoviti u više iteracija pogotovo ako bazično rješenje sadržava više od jedne nule. U tom slučaju dolazi do problema kruženja rješenja koje je izvan domašaja optimuma.

Kod prvog koraka kada biramo vodeći red, tj. vektor koji izlazi iz rješenja. Kada tražimo $min\left\{\frac{a_{i0}}{a_{is}}\right\}, \forall\, a_{is} > 0$, ako su dobivene dvije i/ili više jednakih vrijednosti dolazi do degeneracije jer su u tom slučaju dva i/ili više vektora jednako dobri kandidati za izlazak iz baze. Ako bismo u ovom koraku odabrali bilo koji od ta dva vektora postoji mogućnost komplikacije problema, tj. može doći do kruženja. Bez obzira na pojavu degeneracije u problemu linearnog programiranja svejedno ga je moguće riješiti simpleks metodom. Kako bi se izbjegla mogućnost kruženja u ovom primjeru koristi se metoda koja je po Martiću(1979) rezultirala iz Charnesove procedure perturbacije. U idućem koraku umjesto jednakih omjera $\frac{a_{10}}{a_{1s}}$ i $\frac{a_{20}}{a_{2s}}$, uspoređuju se omjeri $\frac{a_{i1}}{a_{is}}, i = 1{,}2.$ Dakle, kao vektor koji izlazi iz baze odabire se onaj za koji je kvocijent koeficijenta idućeg stupca i vodećeg stupca manji.

Kako bi se izbjeglo kruženje kod početnog bazičnog degeneriranog rješenja najlakši način rješavanja problema je koristeći Blandovo pravilo koje glasi:

- Od svih vektora koji su pogodni kandidati za ulazak u bazu odabire se onaj s najmanjim indeksom
- Od svih vektora koji su pogodni kandidati za izlazak iz baze odabire se onaj s najmanjim indeksom

Prema Vadnalu (1972.) linearni program kojemu treba odrediti vrijednosti varijabla x_1, …, x_n formuliran kao:

$$x_1 \geq 0, \cdots, x_n \geq 0$$
$$a_{11}x_1 + \cdots + a_{1n}x_n \geq b_1$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$a_{m1}x_1 + \cdots + a_{mn}x_n \geq b_m$$

s funkcijom cilja koja ima minimum:

$$f(x_1, \cdots x_n) = c_1x_1 + \cdots + c_nx_n$$

gdje s P^r označavamo jedinični vektor koji izbacujemo iz baze. Kada ne vrijedi da je vektor P^r jednoznačan i dolazi do degeneracije potrebno je promijeniti linearni program. Linearni program s uvjetnom vektorskom jednadžbom:

$$x_1P^1 + \cdots + x_nP^n = P^0$$

I prethodno navedenom funkcijom cilja promijenimo tako da umjesto vektora P^0 uzimamo vektor P(Ɛ):

$$P(\varepsilon) = P^0 + \varepsilon P^1 + {}_{\varepsilon}^{2}P^2 + \cdots {}_{\varepsilon}^{n}P^n$$

gdje je Ɛ neki mali pozitivni broj koji konvergira prema 0 te se samim time vektor P(Ɛ) malo razlikuje od vektora P^0 i konvergira prema njemu. Iz toga slijedi novi linearni program s nepromijenjenom funkcijom cilja i jednadžbom:

$$x_1P^1 + \cdots + x_nP^n = P(\varepsilon)$$

Za bazično nedegenerirano moguće rješenje gdje je prvih n komponenta pozitivno, a ostale su jednake 0 oblika:

$$x = \{x_1, x_2, \cdots, x_n, 0, \cdots, 0\}$$

vrijedi jednadžba:

$$x_1P^1 + x_2P^2 + \cdots + x_mP^m = P^0$$

kojoj s lijeve i desne strane pribrojimo izraz:

$$\varepsilon P^1 + \varepsilon^2P^2 + \cdots + \varepsilon^nP^n$$

kako bismo dobili jednadžbu:

$$x_1P^1 + x_2P^2 + \cdots + x_mP^m + \varepsilon P^1 + \varepsilon^2P^2 + \cdots + \varepsilon^nP^n = P(\varepsilon)$$

Nakon toga potrebno je sve vektore P^j zamijeniti bazičnim vektorima po formulaciji:

$$P^j = a_{1j}P^1 + a_{2j}P^2 + \cdots + a_{mj}P^m, \qquad j = 1,2, \cdots, n$$

te time dobivamo jednadžbu:

$$x_1P^1 + x_2P^2 + \cdots + x_mP^m + \varepsilon(a_{11}P^1 + a_{21}P^2 + \cdots + a_{m1}P^m)\varepsilon^2(a_{12}P^1 + a_{22}P^2 + \cdots + a_{m2}P^m) + \varepsilon^n(a_{1n}P^1 + a_{2n}P^2 + \cdots + a_{mn}P^m) = P(\varepsilon)$$

Kad preuredimo prethodnu jednadžbu dobijemo sljedeće:

$$(x_1 + \varepsilon a_{11} + \varepsilon^2a_{12} + \cdots + \varepsilon^na_{1n})P^1 + (x_2 + \varepsilon a_{21} + \varepsilon^2a_{22} + \cdots + \varepsilon^na_{2n})P^2 + (x_m + \varepsilon a_{m1} + \varepsilon^2a_{m2} + \cdots + \varepsilon^na_{mn})P^m = P(\varepsilon)$$

U sljedećem koraku uvođenjem supstitucije:

$$x_i + \varepsilon a_{i1} + \varepsilon^2a_{i2} + \cdots + \varepsilon^na_{in} = y_i, \qquad i = (1,2, \cdots, m)$$

Dobijemo jednadžbu:

$$y_1P^1 + y_2P^2 + \cdots + y_mP^m = P(\varepsilon)$$

Iz koje slijedi da tako promijenjeni program ima moguće rješenje koje je bazično i nedegenerirano:

$$y = \{y_1, y_2, \cdots, y_m, 0, \cdots, 0\}$$

jer svaki pozitivni element x_i iz prvobitnog programa ima odgovarajući pozitivni element y_i u promijenjenom programu. Promijenjeni linearni program uvijek bude nedegeneriran jer je uvijek moguće odabrati Ɛ koji je proizvoljni maleni relativni broj koji teži prema 0 te zbog toga su promijenjeni i prvobitni program povezani. Ovako promijenjeni program koristi se za odabir vektora koji se odstranjuje iz baze.

Nadalje Vadnal (1972) navodi kako je potrebno uvesti u bazu vektor P^k po formuli:

$$P^k = a_{1k}P^1 + \cdots + a_{rk}P^r + \cdots + a_{mk}P^m$$

pomnožiti s pozitivnim brojem θ_1 i oduzeti od jednadžbe promijenjenog programa kako bismo dobili jednadžbu iz koje možemo odrediti vektor koji izlazi iz baze koja glasi:

$$(y_1 - \theta a_{1k})P^1 + \cdots + (y_r - \theta a_{rk})P^r + \cdots + (y_m - \theta a_{mk})P^m + \theta P^k = P(\varepsilon)$$

Sada određujemo pozitivni broj θ tako da jedna od diferencija $y_i - \theta a_{ik} = 0, za\ a_{ik} > 0$. U tom slučaju vrijedi da $\theta = \frac{y_i}{a_{ik}}$, tj. θ određuje najmanji od omjera.

Kako bi bilo moguće dokazati da vrijedi da je samo jedan omjer najmanji potrebno je dokazati da su svi različiti. Raspisom proizvoljnog razlomka dokazuje se prethodna tvrdnja jer dolazi do uzastopnih potencija broja Ɛ.

$$\frac{y_i}{a_{ik}} = \frac{x_i + \varepsilon a_{i1} + \varepsilon^2 a_{i2} + \cdots + \varepsilon^n a_{in}}{a_{ik}}$$

$$\frac{y_i}{a_{ik}} = \frac{x_i}{a_{ik}} + \varepsilon \frac{a_{i1}}{a_{ik}} + \varepsilon^2 \frac{a_{i2}}{a_{ik}} + \cdots \varepsilon^k + \cdots + \varepsilon^n \frac{a_{in}}{a_{ik}}$$

Iako je moguće riješiti degenerirani problem u linearnom programiranju on svejedno predstavlja problem kruženja kod simpleks algoritma. Iako u praksi kruženje nije zabilježeno, svejedno postoji mogućnost istog. Kruženje se događa kada se iz iteracije u iteraciju funkcija cilja ne mijenja jer je porast funkcije cilja jednak nuli i nakon nekoliko ciklusa mogući je povratak na početno bazično rješenje.

4.2. Degeneracija kod transportnog problema

Degeneracija je zapravo česta pojava kod transportnog problema. Do degeneracije može doći kod početnog rasporeda tereta, tj. postavljanja početnog bazičnog rješenja ili kod bilo koje iteracije. Transportni problem je degeneriran kada kod početnog bazičnog rješenja u tablici imamo manje od m+n-1 pozitivnih bazičnih varijabli. Rješavanje takvog transportnog problema otežano je jer kod metode skakanja s kamena na kamen ne možemo konstruirati zatvoreni put jer nam nedostaje polja (kamenja) dok kod MODI metode dobivamo sustav s m+n varijabli, a broj jednadžbi je manji od m+n-1. Kod transportnog problema prema Babić (2010) svaka bazična varijabla zapravo je jednaka razlici parcijalnih suma $\sum' a_i$ i $\sum' b_j$ te je zato razlog pojave degeneracije slučaj u kojem je neka od parcijalnih suma a_i jednaka nekoj od parcijalnih suma b_j. Kada su te dvije sume jednake, njihova razlika je jednaka 0 i time dolazi do degeneriranog rješenja. Kako bi bilo moguće riješiti takav problem potrebno je izbjeći degeneraciju tako da se čine male izmjene kod ponude a_i i potražnje b_j.Te izmjene se čine tako da se na ponudu i potražnju dodaje neki mali pozitivni broj Ɛ>0. Radi lakšeg razumijevanja degeneraciju transportnog problema prikazat ćemo na primjeru.

Primjer 3: Kamioni prevoze kakao iz tvornice s tri ishodišta I_1, I_2 i I_3 u četiri druge tvornice na daljnju obradu. Prvo ishodište ima kapacitet od 30 kilograma, drugo 20 i treće 50 kilograma. Prva tvornica dnevno troši 10 kilograma, druga 20, treća 30 i četvrta 40 kilograma. Cijene prijevoza zadane su tablicom:

Tablica 6: Početna tablica za transportni problem

Ishodišta	Odredišta				a_i
	O_1	O_2	O_3	O_4	
I_1	10	11	13	17	30
	x_{11}	x_{12}	x_{13}	x_{14}	
I_2	12	22	22	15	20
	X_{21}	X_{22}	X_{23}	X_{24}	
I_3	18	27	24	9	50
	X_{31}	X_{32}	X_{33}	X_{34}	
b_j	10	20	30	40	100/100

Prvi korak u rješavanju zadatka je određivanje funkcije cilja primala i duala i ograničenja nad funkcijama.

$$Z = c_{11} * x_{11} + c_{12} * x_{12} + c_{13} * x_{13} + c_{14} * x_{14} + c_{21} * x_{21} + c_{22} * x_{22} + c_{23} * x_{23} + c_{24} * x_{24} + c_{31} * x_{31} + c_{32} * x_{32} + c_{33} * x_{33} + c_{34} * x_{34} \rightarrow min$$

$$Z = 10x_{11} + 11x_{12} + 13x_{13} + 17x_{14} + 12x_{21} + 22x_{22} + 22x_{23} + 15x_{24} + 18x_{31} + 27x_{32} + 24x_{33} + 9x_{34} \rightarrow min$$

$$x_{11} + x_{12} + x_{13} + x_{14} = 30 \qquad x_{11} + x_{21} + x_{31} = 10$$
$$x_{21} + x_{22} + x_{23} + x_{24} = 20 \qquad x_{12} + x_{22} + x_{32} = 20$$
$$x_{31} + x_{32} + x_{33} + x_{34} = 50 \qquad x_{13} + x_{23} + x_{33} = 30$$
$$x_{14} + x_{24} + x_{34} = 40$$

$$Z^d = 30u_1 + 20u_2 + 50u_3 + 10v_1 + 20v_2 + 30v_3 + 40v_4 \rightarrow max$$

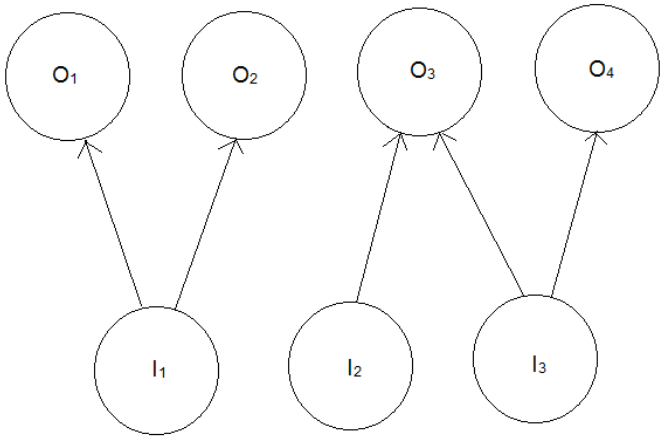
$$u_1 + v_1 \leq 10 \qquad u_2 + v_1 \leq 11 \qquad u_3 + v_1 \leq 13 \qquad u_4 + v_1 \leq 17$$
$$u_1 + v_2 \leq 12 \qquad u_2 + v_2 \leq 22 \qquad u_3 + v_2 \leq 22 \qquad u_4 + v_2 \leq 15$$
$$u_1 + v_3 \leq 18 \qquad u_2 + v_3 \leq 27 \qquad u_3 + v_3 \leq 24 \qquad u_4 + v_3 \leq 9$$

U sjedećem koraku postavljamo tablicu početnog bazičnog rješenja koristeći jednu od prethodno navedenih metoda za alokaciju početnog tereta. U ovom primjeru korištena je metoda sjeverozapadnog kuta.

Tablica 7: Raspored tereta metodom sjeverozapadnog kuta

Ishodišta	Odredišta				a_i
	O_1	O_2	O_3	O_4	
I_1	10	11	13	17	30
	10	**20**			
I_2	12	22	22	15	20
			20		
I_3	18	27	24	9	50
			10	**40**	
b_j	10	20	30	40	100/100

Iz gornje tablice vidljivo je da je početno bazično rješenje degenerirano jer je broj pozitivnih bazičnih varijabli u tablici manji od m+n-1. m+n-1=6 dok je broj pozitivnih bazičnih varijabli 5. Kako bismo riješili problem degeneracije prvo moramo locirati prazno polje s najmanjim troškom, u ovom primjeru to je polje x_{21} čiji je jedinični trošak 12. Prvo provjeravamo za to polje da li je moguće odrediti zatvoreni put. Iz primjera vidljivo je da je nemoguće odrediti zatvoreni put za polje x_{21} pa je ono pogodan kandidat za alokaciju fiktivnog tereta. Na polje alociramo fiktivni teret Ɛ te smo time dobili dovoljan broj pozitivnih bazičnih varijabli i problem više nije degeneriran. Ovu degeneraciju možemo prikazati i na grafički način iz kojeg se vidi da odredišta i ishodišta nisu povezana u jedan zajednički sustav već su razdvojena na dva.



Slika 2: Grafički prikaz degeneracije transportnog problema (Prema: Dobrenić 1978.)

Kako bismo riješili degeneraciju fiktivnim teretom spajamo ishodište I_2 s odredištem O_1 tako da se dodaje mali pozitivni broj Ɛ>0, tj. fiktivni teret na polje x_{21}. U sljedećem koraku potrebno je popuniti prazna polja, tj. izračunati relativne troškove putem zatvorenog puta. Za prvo prazno polje x_{13} zatvoreni put je preko polja x_{23}, x_{21}, x_{11}. Teret računamo tako da prvo

polje dobiva negativni predznak, drugo pozitivni, treće opet negativni i tako do zadnjeg polja. Dakle izračun tereta za polje x_{13} glasi $c_{13^*} = -c_{13} + c_{23} - c_{21} + c_{11} = -13 + 22 - 12 + 10 = 7$. Isti postupak ponavljamo za ostatak praznih polja u tablici.

$$c_{14^*} = -c_{14} + c_{34} - c_{33} + c_{23} - c_{21} + c_{11} = -17 + 9 - 24 + 22 - 12 + 10 = -12$$
$$c_{22^*} = -c_{22} + c_{21} - c_{11} + c_{12} = -22 + 12 - 10 + 11 = -9$$
$$c_{24^*} = -c_{24} + c_{43} - c_{33} + c_{23} = -15 + 9 - 24 + 22 = -8$$
$$c_{31^*} = -c_{31} + c_{21} - c_{23} + c_{33} = -18 + 12 - 22 + 24 = -4$$
$$c_{32^*} = -c_{32} + c_{12} - c_{11} + c_{21} - c_{23} + c_{33} = -27 + 11 - 10 + 12 - 22 + 24 = -12$$

Na kraju popunjena tablica izgleda ovako:

Tablica 8: Popunjena početna tablica transportnog problema

Ishodišta	Odredišta				a_i
	O_1	O_2	O_3	O_4	
I_1	10	11	13	17	30
	10	**20**	7	-12	
I_2	12	22	22	15	20
	ε	-9	**20**	-8	
I_3	18	27	24	9	50
	-4	-12	**10**	**40**	
b_j	10	20	30	40	100/100

Ukupni troškovi za početnu tablicu iznose :

$$Z = c_{11} * x_{11} + c_{12} * x_{12} + c_{23} * x_{23} + c_{33} * x_{33} + c_{34} * x_{34}$$
$$= 10 * 10 + 20 * 11 + 20 * 22 + 10 * 24 + 40 * 9 = 1360$$

Iz tablice vidljivo je da rješenje nije optimalno jer se u polju x_{13} nalazi pozitivni relativni trošak koji iznosi 7. Sada je potrebno na to polje staviti određenu količinu tereta. Teret selimo po zatvorenom putu koji je za polje x_{13} preko polja x_{23}, x_{21} i x_{11}. Teret selimo tako da ga dodamo na polje x_{13}, oduzmemo s polja x_{23}, dodamo na polje x_{21}, i ponovno oduzmemo s polja x_{11}. Prvo na zatvorenom putu na poljima s kojih oduzimamo teret tražimo najmanji teret, u ovom slučaju to je na polju x_{11} i iznosi 10. Nakon prenošenja tereta tablica izgleda ovako:

Tablica 9: Prva iteracija transportnog problema

Ishodišta	Odredišta				a_i
	O_1	O_2	O_3	O_4	
I_1	10	11	13	17	30
		20	**10**		
I_2	12	22	22	15	20
	10+Ɛ		**10**		
I_3	18	27	24	9	50
			10	**40**	
b_j	10	20	30	40	100/100

Iz tablice je vidljivo da je novi raspored nedegeneriran jer je broj pozitivnih bazičnih varijabli jednak m+n-1, tj. 6. Sada je potrebno ponovno izračunati relativne troškove za prazna polja putem zatvorenog puta. Nakon izračuna popunjena tablica prve iteracije izgleda ovako:

Tablica 10: Popunjena tablica prve iteracije transportnog problema

Ishodišta	Odredišta				a_i
	O_1	O_2	O_3	O_4	
I_1	10	11	13	17	30
	-7	**20**	**10**	-19	
I_2	12	22	22	15	20
	10+Ɛ	-2	**10**	-8	
I_3	18	27	24	9	50
	-4	-5	**10**	**40**	
b_j	10	20	30	40	100/100

Iz tablice je vidljivo da je druga iteracija i optimalno rješenje transportnog problema jer su svi relativni troškovi negativnog predznaka, a ujedno je rješenje nedegenerirano jer je broj bazičnih varijabli 6 što znači da vrijedi pravilo da je broj bazičnih varijabli jednak m+n-1. Na polju x_{21} imamo vrijednost 10+Ɛ no kao što smo na početku spomenuli Ɛ je toliko maleni broj da je zanemariv pa ne računamo s njime u funkciji cilja. Ukupni troškovi za drugu iteraciju iznose:

$$Z = c_{12} * x_{12} + c_{13} * x_{13} + c_{21} * x_{21} + c_{23} * x_{23} + c_{33} * x_{33} + c_{34} * x_{34}$$
$$= 20 * 11 + 10 * 13 + 10 * 12 + 10 * 22 + 10 * 24 + 40 * 9 = 1290$$

5.Rješavanje degeneriranog problema linearnog programiranja

Rješavanje degeneriranog problema linearnog programiranja koji rješavamo simpleks algoritmom pobliže je objašnjena pomoću programa izrađenog koristeći Jupyter Notebook-a. Jupyter Notebook je neprofitna „open-source“ web-stranica s koja je nastala iz Python projekta 2014. godine,a omogućuje stvaranje i razmjenu dokumenata koji sadrže kodove, jednadžbe, vizualizacije i/ili tekst. Uz Jupyter korištena je python biblioteka koja se naziva SciPy koja koristi Numpy za matematičke funkcije. Korištena je biblioteka SciPy jer dolazi s modulima za razne zadatke, uključujući probleme linearne optimizacije. Konkretno korišten je scipy.optimize koji među ostalim sadržava funkciju linprog (koja minimizira linearnu funkciju ovisno o graničenjima i uvjetima jednakosti i/ili nejednakosti) i minimize (koja minimizira skalarne funkcije za jednu ili više varijabli).

5.1. Primjer rješenja simpleks algoritmom

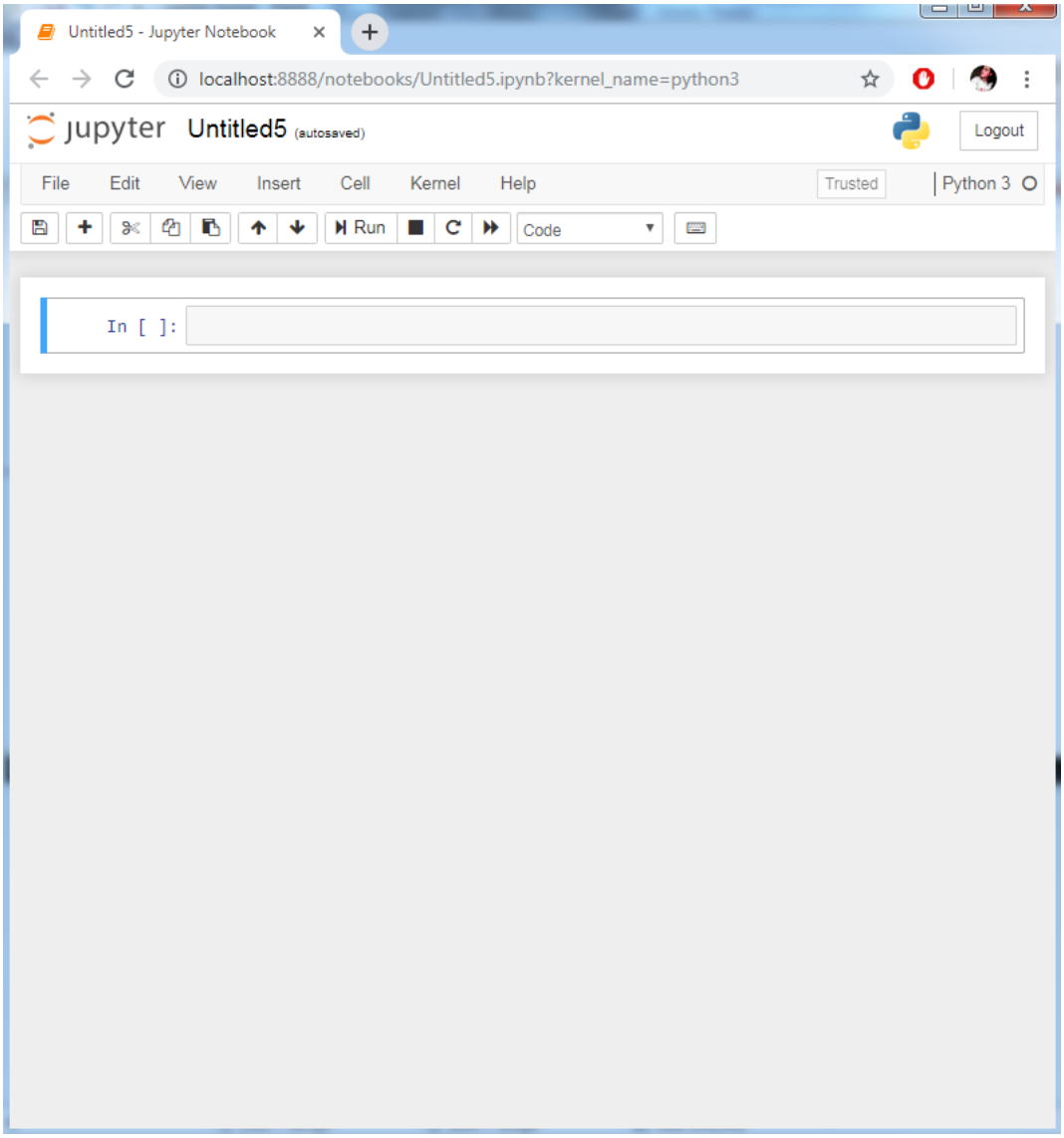
Primjer 1:linearna funkcija za max

$$Z = 5x_1 + 3x_2 + 6x_3 \rightarrow max$$
$$4x_1 + 6x_2 + 2x_3 \leq 40$$
$$2x_1 + 2x_2 + x_3 \leq 20$$
$$x_1 + x_2 + 2x_3 \leq 60$$
$$x_1, x_2, x_3 \geq 0$$

Kada bismo krenuli ovaj problem rješavati na papiru prvo bismo trebali taj isti pretvoriti u kanonski oblik te popuniti početnu simpleks tablicu. Potom odabrati vodeći stupac i red, izbaciti iz baze i ubaciti odgovarajuće vektore te transformirati sve elemente po pravilima i postupcima navedenim u prethodnim poglavljima. Ručnim računanjem došli bismo eventualno do rješenja ovog problema no ono bi oduzelo puno više vremena i ostavlja prostora za eventualne pogreške kod transformacije elemenata, zato je bolji izbor koristiti programska rješenja za takve zadatke. Na internetu postoji već mnogo gotovih simpleks kalkulatora od kojih se jedan nalazi na web stranici: https://cbom.atozmath.com/CBOM/Simplex.aspx?q=sm koji će se koristiti za usporedbu rješenja dobivenog programom vlastite izrade.

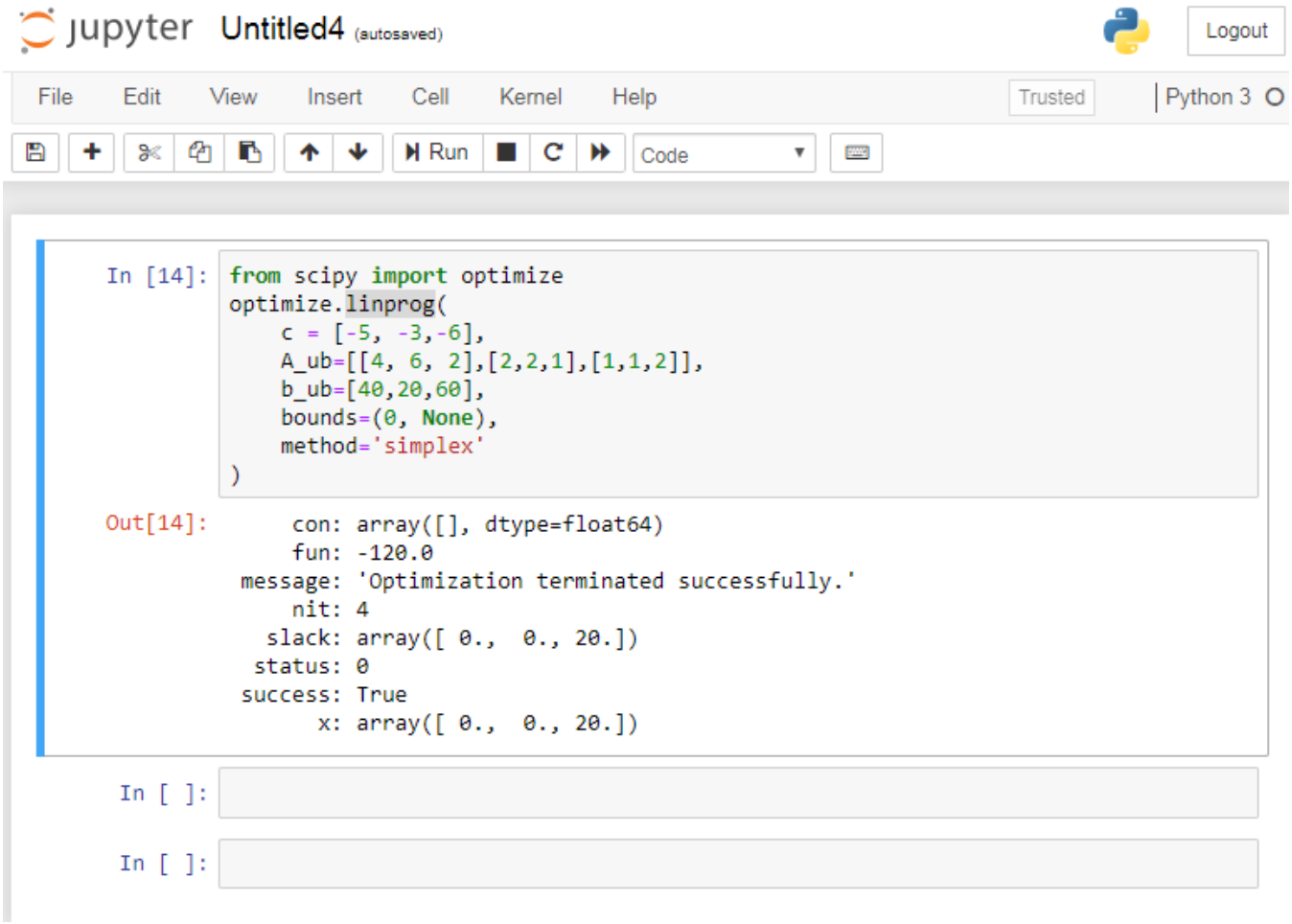
Kako bismo uopće mogli koristiti Jupyter Notebook prvo ga je potrebno instalirati. U cmd prozoru potrebno je upisati komandu „pip install jupyterlab“. Kada je instalacija dovršena,

možemo ga otvoriti pomoću komande „jupyter notebook“ na što nam se otvara prozor u web pregledniku kao što je prikazan na sljedećoj slici.



Slika 3. Jupyter Notebook u aplikaciji Google Chrome

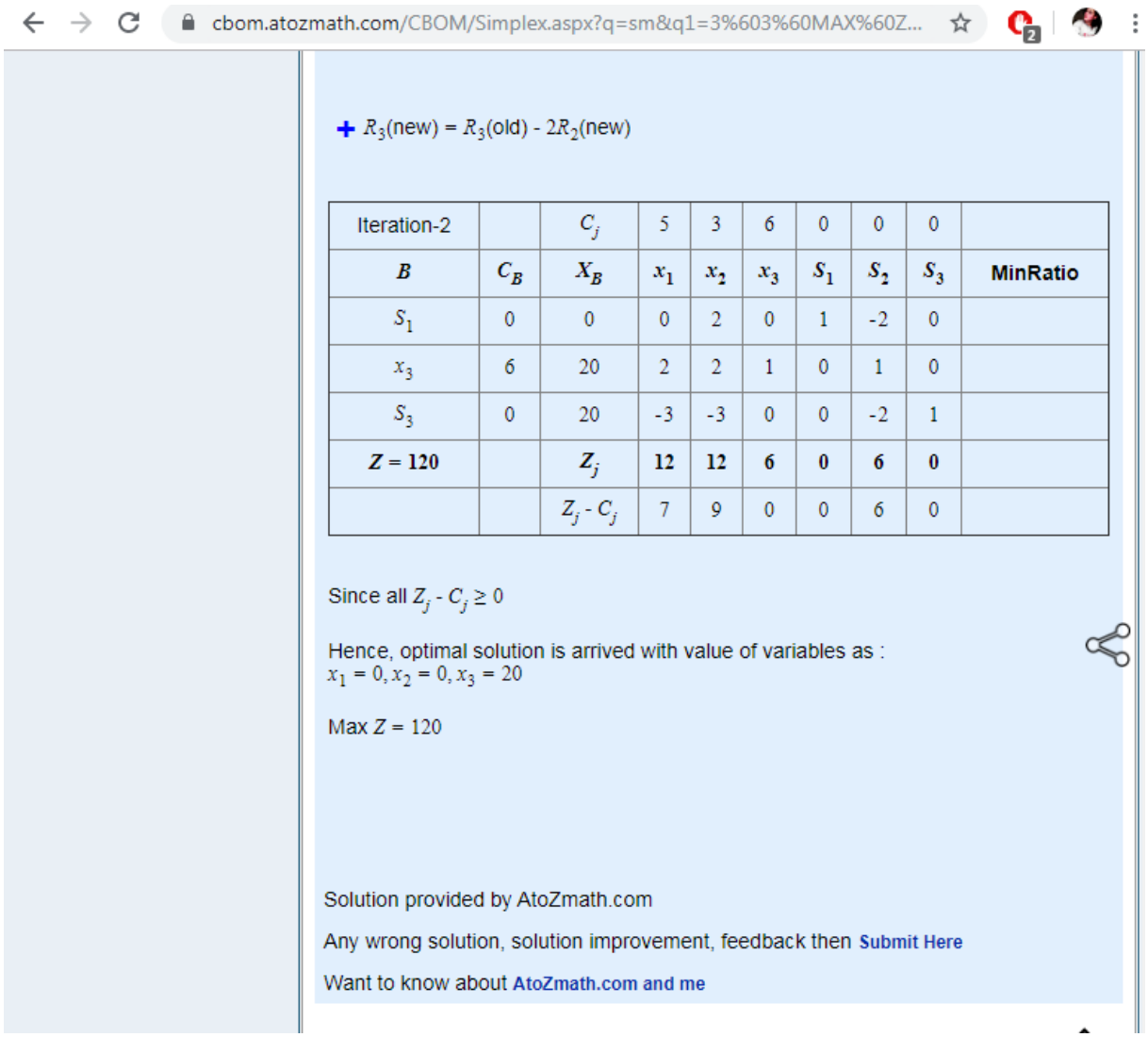
Kao što je ranije navedeno, sve funkcije koje pozivamo su funkcije za minimizaciju programa dok je naš primjer funkcija maksimizacije pa je zato potrebno u naš kalkulator unijeti vrijednosti funkcije *Z* s negativnim predznakom kako bismo kao rješenje dobili –f(x). Znači dobiveno rješenje funkcije potrebno je samo pomnožiti s -1.



Slika 4. Rješenje Primjera 1

Na prethodnoj slici vidimo da su u listu c upisane vrijednosti funkcije s negativnim predznakom kako bismo mogli dobiti maksimizaciju problema. Lista listi A_ub sadrži matricu A dok lista b_ub sadrži u sebi vrijednosti matrice B. Bounds=(0, None) su ograničenja što znači $x_1,x_2,x_3 \geq 0$ i na kraju metoda koju koristimo je simplex. Iz prethodne slike vidimo da je rješenje našeg programa 120, što znači da je Z=120, za $x_1=0$, $x_2=0$ i $x_3=20$. $Z = 5x_1 + 3x_2 + 6x_3 \rightarrow 120 = 5 * 0 + 3 * 0 + 6 * 20$.

Ako prethodni zadatak upišemo u simpleks kalkulator dostupan na internetu radi provjere rezultata dobit ćemo isto rješenje što je prikazano na slici 5.



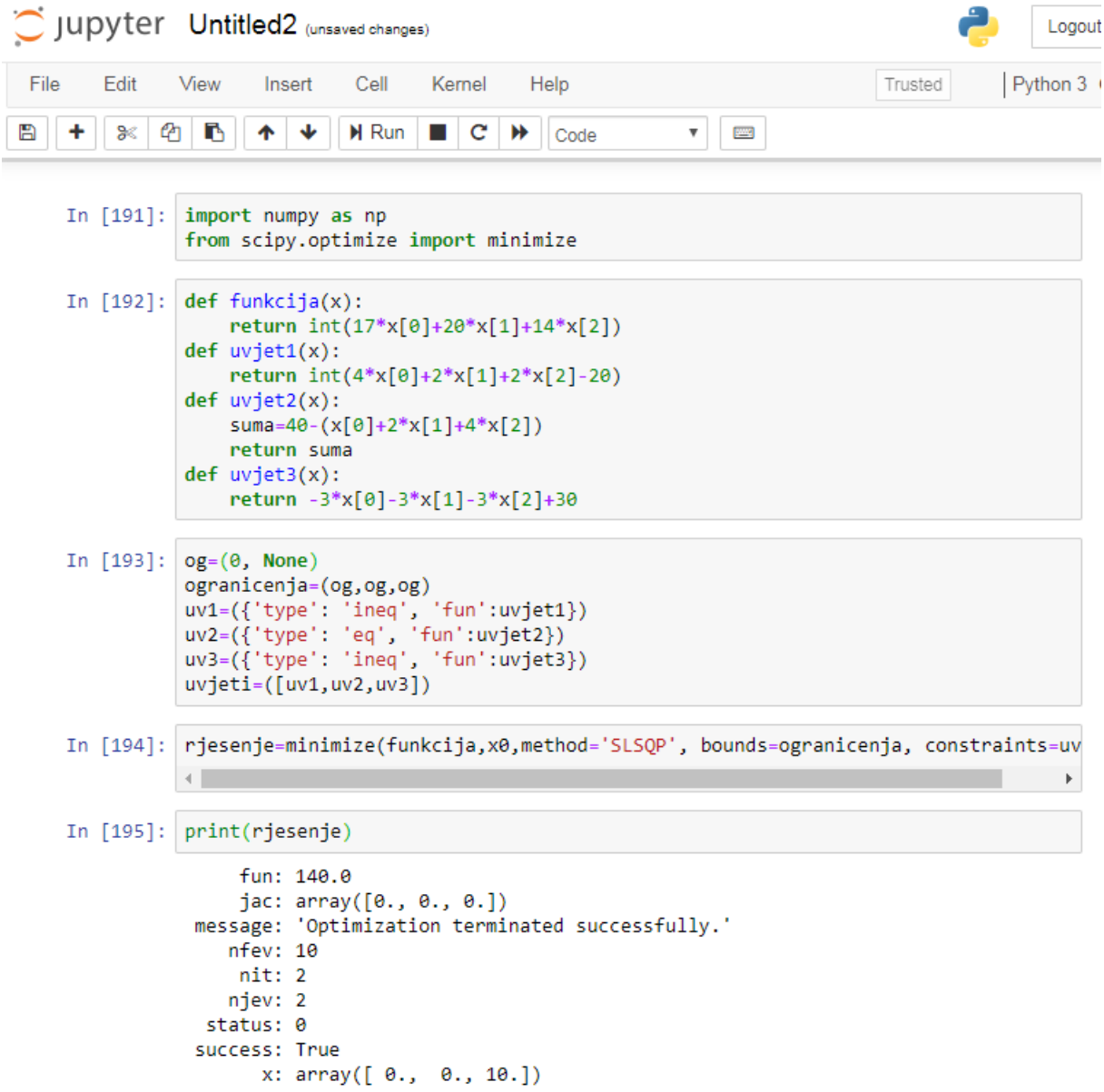
Iteration-2		C_j	5	3	6	0	0	0	
B	C_B	X_B	x_1	x_2	x_3	S_1	S_2	S_3	**MinRatio**
S_1	0	0	0	2	0	1	-2	0	
x_3	6	20	2	2	1	0	1	0	
S_3	0	20	-3	-3	0	0	-2	1	
$Z = 120$		Z_j	**12**	**12**	**6**	**0**	**6**	**0**	
		$Z_j - C_j$	7	9	0	0	6	0	

Slika 5. Rješenje Primjera 1 s online kalkulatorom

Primjer 2:linearna funkcija za min

$$Z = 17x_1 + 20x_2 + 14x_3 \rightarrow min$$
$$4x_1 + 2x_2 + 2x_3 \geq 20$$
$$1x_1 + 2x_2 + 4x_3 = 40$$
$$3x_1 + 3x_2 + 3x_3 \leq 30$$
$$x_1, x_2, x_3, \geq 0$$

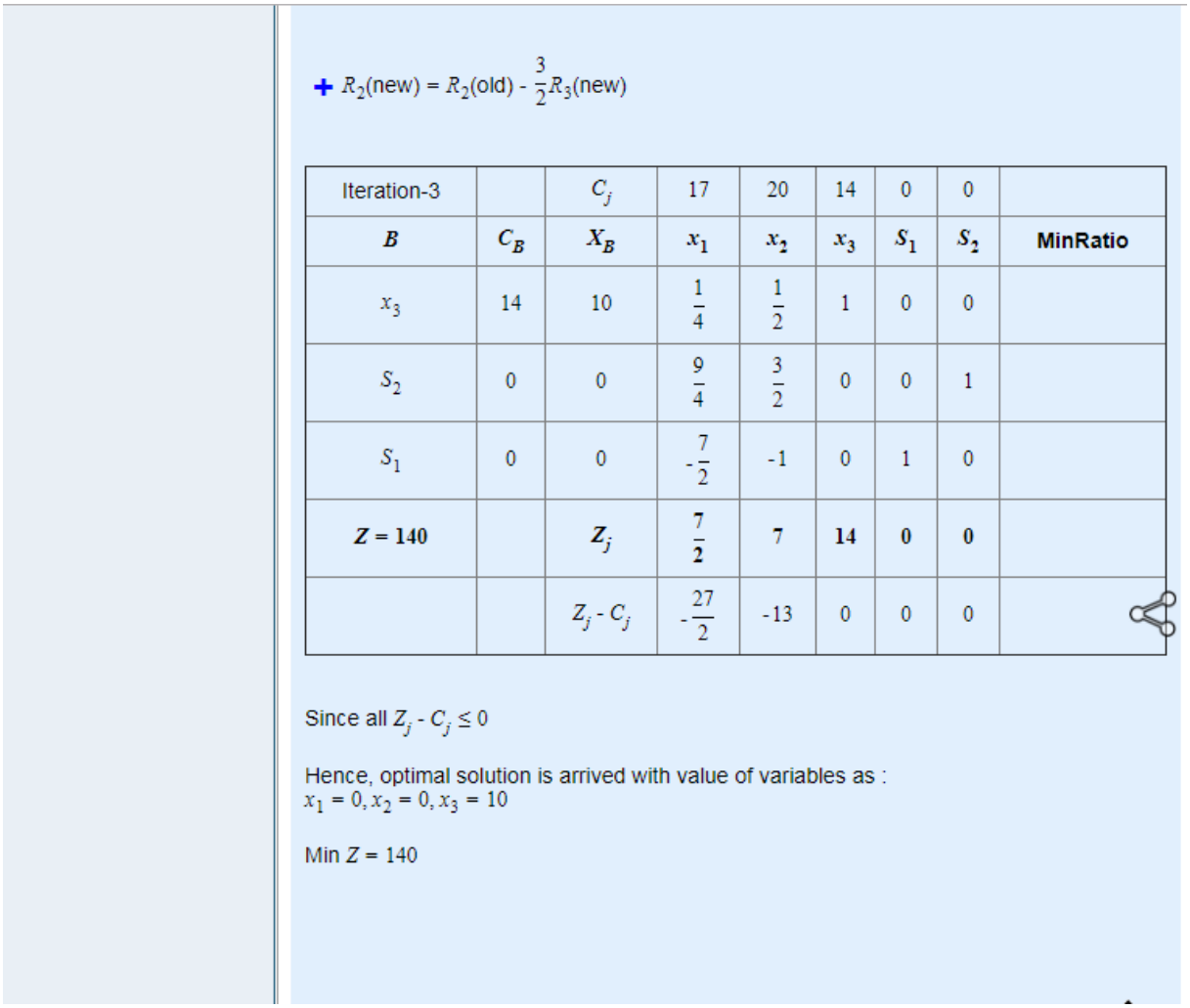
Za rješavanje ovog primjera pozvana je funkcija minimize. Kod i rješenje vidljivi su na sljedećoj slici.



Slika 6. Rješenje Primjera 2

Na prethodnoj slici vidimo prvo je definirana funkcija koja se naziva funkcija koja vraća vrijednost funkcije *Z.* Potom su uvjeti definirani kao funkcije s nazivom uvjet1, uvjet2 i uvjet3. Uvjet1 $4x_1 + 2x_2 + 2x_3 \geq 20$ zapisujemo tako da 20 prebacimo na lijevu stranu nejednadžbe čime dobiva negativan predznak i kasnije u programu uvjetu dodajemo tip nejednakosti. Uvjet2 $1x_1 + 2x_2 + 4x_3 = 40$ koji je jednadžba zapisujemo tako da sve nepoznanice na lijevoj strani prebacimo na desnu stranu jednadžbe i kasnije tom uvjetu dodajemo tip jednakosti. Uvjet3 $3x_1 + 3x_2 + 3x_3 \leq 30$ je tipa manje ili jednako (≤) dok program svaku nejednakost interpretira kao veće ili jednako (≥) što znači da je potrebno nejednakost pomnožiti s (-1) kako bismo okrenuli znak nejednakosti. Kada smo pomnožili nejednadžbu potom zapisujemo ovaj uvjet na isti način kao i uvjet1. Ponovno postavljamo ograničenja, tj. $x_1,x_2,x_3 \geq 0$ zapisujemo kao og=(0, None). Kada smo postavili ograničenja na sve varijable i za svako od

ograničenja odredili tip, pozivamo funkciju minimize. Ako prethodni zadatak upišemo u simpleks kalkulator dostupan na internetu radi provjere rezultata dobit ćemo isto rješenje što je prikazano na slici 7.



$+ R_2(\text{new}) = R_2(\text{old}) - \frac{3}{2}R_3(\text{new})$

Iteration-3		C_j	17	20	14	0	0	
B	C_B	X_B	x_1	x_2	x_3	S_1	S_2	MinRatio
x_3	14	10	$\frac{1}{4}$	$\frac{1}{2}$	1	0	0	
S_2	0	0	$\frac{9}{4}$	$\frac{3}{2}$	0	0	1	
S_1	0	0	$-\frac{7}{2}$	-1	0	1	0	
$Z = 140$		Z_j	$\frac{7}{2}$	7	14	0	0	
		$Z_j - C_j$	$-\frac{27}{2}$	-13	0	0	0	

Since all $Z_j - C_j \leq 0$

Hence, optimal solution is arrived with value of variables as :
$x_1 = 0, x_2 = 0, x_3 = 10$

Min $Z = 140$

Slika 7. Rješenje Primjera 2 s online kalkulatorom

6.Zaključak

U ovom radu obrađena je tema degeneracije problema u linearnom programiranju. Kako bi uopće bilo moguće objasniti što je degeneracija na početku rada približeni je pojam operacijskih istraživanja i samog linearnog programiranja kao matematička metoda. Pobliže je objašnjena primjena linearnog programiranja danas. Danas imamo više vrsta problema linearnog programiranja, opći, standardni i kanonski kojim računamo u simpleks algoritmu. Isto tako jedna od najpoznatijih metoda je simpleks metoda koja koristi simpleks algoritam no nije i jedina metoda. Uz simpleks metodu u ovom radu obrađeno je grafičko rješavanje problema linearnog programiranja koje koristimo kada imamo problem sa samo dvije varijable i transportni problem. Kod svake metode objašnjeni su postupci i koraci rješavanja kako bi bilo lakše razumjeti dolazak do rješenja i kako bi uopće bilo moguće objasniti što je i kako nastaje degeneracija. Prvo je objašnjeno kako prepoznati degeneraciju i kako ona nastaje kod problema linearnog programiranja. Sama degeneracija opisana je i na primjeru simpleks metode te transportnog problema. Prvo je prikazan primjer degeneriranog transportnog problema i taj isti je riješen. Potom su prikazana dva primjera degeneriranog problema linearnog programiranja kod simplex metode. Prvi primjer je standardni problem maksimizacije funkcije dok je drugi opći problem minimizacije funkcije. Obadva primjera su degenerirana te kao što je u prethodnim poglavljima opisano, ona su bez problema riješena. Sama degeneracija ne predstavlja nemogućnost rješavanja problema, i uz mnoga istraživanja danas postoji mnogo pravila i metoda kako izbjeći i/ili riješiti degenerirani problem. Degeneracija predstavlja prijetnju kod rješavanja problema u smislu da produžuje broj koraka rješavanja i u ekstremnim slučajevima može dovesti do kruženja, no svakako moguće je i takve probleme riješiti u konačnom broju koraka.

7.Popis literature

[1] Babić, Z. (2010). Linearno programiranje. Split: Ekonomski fakultet.

[2] Barković, D. (2001). Operacijska istraživanja. Osijek: Sveučilište Josipa Jurja Strossmayera.

[3] BestT, M. J. (1985). Linear programming : active set analysis and computer programs. Englewood Cliffs: Prentice-Hall.

[4] Dobrenić, S. (1978). Operativno istraživanje. Varaždin: FOI.

[5] Kalpić, D., Mornar, V. (1996). Operacijska istraživanja. Zagreb: Zeus-DRIP.

[6] Kreko, B. (1966). Linearno programiranje. Beograd: Savremena administracija.

[7] Loomba, P. (1964). Linear programming : an introductory analysis. New york: McGraw-Hill Book Company.

[8] Lukač, Z, Neralić L.. (2012). Operacijska istraživanja. Zagreb: Element.

[9] Martić, Lj. (1979). Matematičke metode za ekonomske analize II. Zagreb: Narodne novine.

[10] Neralić, L. (2016). O linearnom programiranju I. Preuzeto 07.09.2019. s https://www.mioc.hr/wp/wp-content/uploads/2016/03/O-Linearnom-programiranju-I-2.pdf

[11] Perić, T. (2017). Linearni model proizvodnje. Preuzeto 07.09.2019. s http://www.efzg.unizg.hr/UserDocsImages/MAT/tperic/2_OI_2017.pdf

[12] Tulsian, P. C., Tulsian B. (2012). Operations Research (Theory and Practice). New Delhi: S. Chand & Company Ltd.

[13] Vadnal, A. (1972). Linearno programiranje : Teorija i upotreba u privredi. Zagreb: Informator.

8.Popis slika

9.Popis tablica

10.Prilozi

Python kod za primjer 1:

```
from scipy import optimize
optimize.linprog(
    c = [-5, -3,-6],
    A_ub=[[4, 6, 2],[2,2,1],[1,1,2]],
    b_ub=[40,20,60],
    bounds=(0, None),
    method='simplex'
)
```

Python kod za primjer 2:

```
import numpy as np
from scipy.optimize import minimize

def funkcija(x):
    return int(17*x[0]+20*x[1]+14*x[2])
def uvjet1(x):
    return int(4*x[0]+2*x[1]+2*x[2]-20)
def uvjet2(x):
    suma=40-(x[0]+2*x[1]+4*x[2])
    return suma
def uvjet3(x):
    return -3*x[0]-3*x[1]-3*x[2]+30

og=(0, None)
ogranicenja=(og,og,og)
uv1=({'type': 'ineq', 'fun':uvjet1})
uv2=({'type': 'eq', 'fun':uvjet2})
uv3=({'type': 'ineq', 'fun':uvjet3})
uvjeti=([uv1,uv2,uv3])

rjesenje=minimize(funkcija,x0,method='SLSQP', bounds=ogranicenja,
constraints=uvjeti)

print(rjesenje)
```