Linearno programiranje

Uvodne napomene

Razvoj *metoda za rješavanje problema linearnog programiranja* mnogi svrstavaju među *najvažnija naučna dostignuća* sredine prošlog stoljeća. Danas je to standardni alat koji štedi milione dolara brojnim kompanijama u mnogim razvijenim zemljama svijeta. Prema *F. S. Hillieru*, "*Najveći dio svih naučnih izračunavanja na računarima je vezan za korištenje linearnog programiranja".*

Model linearnog programiranja se prvi put javlja u radovima *J. von Neuman-a* (1936) i *L. V. Kantorovič-a* (1939). Prvu efikasnu opću proceduru za rješavanje zadatka linearnog programiranja, poznatu pod nazivom ***simpleks metod***, razvio je *G. Dantzig* (1947). To je efikasan metod koji omogućuje "rutinsko" rješavanje ogromnih zadataka linearnog programiranja pomoću računara. Interesantno je da mada je dokazano da u *najgorem slučaju* simpleks metod ima vrijeme izvršavanja koje je *eksponencijalna funkcija* broja promjenljivih (što je *ravno katastrofi*), takvi "patološki" slučajevi za koje simpleks algoritam radi katastrofalno sporo *praktično se nikada ne javljaju u praksi* (za šta postoje i izvjesna teoretska objašnjenja). Štaviše, pri rješavanju *praktičnih problema* simpleks metod se pokazao kao *izuzetno efikasan*. Preciznije, u praksi, vrijeme izvršavanja simpleks algoritma je obično približno proporcionalno *trećem stepenu broja promjenljivih*, slično kao i kod većine *klasičnih algoritama linearne algebre* (*rješavanje sistema linearnih jednačina Gaussovim metodom eliminacije*, *nalaženje inverzne matrice*, itd.).

Pitanje da li postoji algoritam koji *čak i u najgorem slučaju* može naći rješenje problema linearnog programiranja u vremenu koje ne prelazi neku *polinomijalnu funkciju* broja promjenljivih bilo je otvoreno skoro četiri decenije. Potvrdan odgovor na ovo pitanje ponudio je tzv. ***elipsoidni algoritam*** (koji je neka vrsta sofisticirane *višedimenzionalne binarne pretrage*), koji se obično pripisuje *L. G. Khachiyan*-u (1979), mada je on u suštini vješta kompilacija ideja brojnih ruskih matematičara razvijenih u proteklom periodu. Međutim, za potrebe praktičnih problema, elipsoidni algoritam se pokazao *mnogo sporijim* od simpleks algoritma. Nekoliko godina kasnije, *N. Karmarkar* (1984) je razvio *novi efikasni algoritam* za rješavanje problema linearnog programiranja nazvan ***metod unutrašnje tačke*** (engl. interior-point approach). Slično elipsoidnom algoritmu, ovaj algoritam također garantira polinomijalno vrijeme izvršavanja čak i u najgorem slučaju, ali je *mnogo brži od njega*. Mada je za praktične probleme male do srednje veličine metod unutrašnje tačke *i dalje sporiji od simpleks algoritma*, danas se smatra da metod unutrašnje tačke ima *određene prednosti* (kako sa aspekta *brzine*, tako i sa aspekta *tačnosti*) u odnosu na simpleks metod pri rješavanju vrlo velikih (složenih) zadataka.

Modelom linearnog programiranja može se predstaviti jako veliki broj realnih problema, koji po svojoj prirodi i opisu mogu biti vrlo različiti. Pri tome se najveći broj primjena može definirati kao *opći problem dodjele* (*alokacije*) *ograničenih resursa na konkurentne aktivnosti na najbolji način*. Treba istaći i da su, istorijski gledano, ideje iz oblasti linearnog programiranja inspirirale *mnoge od centralnih koncepata optimizacione teorije*, kao što su ***konveksnost***, ***dualnost***, ***dekompozicija***, kao i razne njihove *generalizacije.*

Linearno programiranje je ubjedljivo najčešće korišteni model (zadatak) operacionih istraživanja. Osnovni razlozi za to su sljedeći:

- postoji veliki broj *različitih realnih problema* koji se mogu ***vjerno predstaviti*** modelom linearnog programiranja;
- postoje *vrlo efikasne metode* za rješavanje zadataka linearnog programiranja pogodne za *računasku implementaciju*;
- veliki broj algoritama za *druge tipove optimizacionih problema* radi tako što rješava probleme linearnog programiranja *kao potprobleme*.

Tipični zadaci, koji se mogu predstaviti modelom linearnog programiranja su ***problem optimalne proizvodnje*** i ***problem optimalne mješavine***. Stoga ćemo detaljnije razmotriti ova dva problema.

Prvo ćemo razmotriti *problem optimalne proizvodnje*. Tipični zadatak izrade *optimalnog plana proizvodnje* može se definirati na sljedeći način. Potrebno je proizvesti n *različitih vrsta proizvoda* P_j, $j = 1 \,..\, n$, pri čemu je za proizvodnju pojedinih dijelova tih proizvoda potrebno koristiti m *različitih mašina* M_i, $i = 1 \,..\, m$. Pri tome je poznato sljedeće:

- *efekti* c_j, $j = 1 .. n$ koji se mogu ostvariti *prodajom* svakog od proizvoda P_j respektivno;
- *raspoloživi kapaciteti* b_i, $i = 1 .. m$ svake od mašina M_i respektivno;
- *kapaciteti* (*tehnološki normativi*) $a_{i,j}$, $i = 1 .. m$, $j = 1 .. n$ potrebni za *proizvodnju jedne jedinice* bilo koje vrste proizvoda P_j na pojedinoj mašini M_i.

Cilj je naći *optimalni plan proizvodnje*, odnosno odrediti respektivno *broj jedinica* x_j, $j = 1 .. n$, svake vrste proizvoda P_j kojeg treba proizvesti da bi *efekti* ostvareni prodajom svih proizvedenih proizvoda bili *što je god moguće veći* (tj. *maksimalni*).

Razmotrimo kako se može zapisati matematski model ovog problema. Ako proizvedemo *za sada neodređen broj* x_j *jedinica* proizvoda P_j, efekat ostvaren *prodajom svih proizvedenih jedinica* proizvoda P_j će biti $c_j x_j$. Stoga će *ukupan efekat* ostvaren prodajom *svih jedinica svih vrsta proizvoda* biti:

$$Z(\mathbf{x}) = Z(x_1, x_2, ..., x_n) = c_1 x_1 + c_2 x_2 + ... + c_n x_n = \sum_{j=1}^{n} c_j x_j$$

Cilj je ostvariti *maksimalni efekat*, što znači da treba *maksimizirati funkciju* $Z(\mathbf{x})$. Pošto je u ovom slučaju *funkcija cilja linearna*, odnosno efekat (cijena) od prodaje *jedne jedinice proizvoda* je *konstantan* (ne zavisi od *broja proizvedenih jedinica*), i kako su pri tome svi efekti (cijene) *pozitivne veličine*, maksimalni efekat bi se dobio pri proizvodnji *beskonačnog broja* svih proizvoda P_j (matematički gledano, ovo je posljedica činjenice da linearna funkcija nije *ničim ograničena odozgo*). Međutim, iz postavke problema vidimo da je proizvodnja *ograničena kapacitetima mašina*. Proizvodnja *jedne jedinice* proizvoda P_j na mašini M_i zauzeće (potrošiće) dio ukupnog kapaciteta mašine M_i u iznosu $a_{i,j}$, tako da će proizvodnja x_j jedinica proizvoda P_j zauzeti (potrošiti) dio ukupnog kapaciteta mašine M_i u iznosu $a_{i,j} x_j$. Konačno, proizvodnja *svih jedinica svih vrsta proizvoda* zauzeće (potrošiće) dio ukupnog kapaciteta mašine M_i u iznosu

$$a_{i,1} x_1 + a_{i,2} x_2 + ... + a_{i,n} x_n = \sum_{j=1}^{n} a_{i,j} x_j$$

Kako je korištenje mašine M_i ograničeno njenim *kapacitetom* b_i, to mora vrijediti ograničenje

$$a_{i,1} x_1 + a_{i,2} x_2 + ... + a_{i,n} x_n = \sum_{j=1}^{n} a_{i,j} x_j \le b_i$$

Postoji ukupno m ovakvih ograničenja, po jedno za svaku od m mašina.

Pored ovih ograničenja, postoji i *prirodno ograničenje* na broj proizvednih jedinica, prema kojem se *ne može proizvesti negativan broj jednica proizvoda*. Stoga mora vrijediti

$$x_j \ge 0, j = 1 .. n.$$

Zadatak se, dakle, može napisati u obliku

$$\arg\max Z(\mathbf{x}) = c_1 x_1 + c_2 x_2 + ... + c_n x$$

p.o.

$$a_{1,1} x_1 + a_{1,2} x_2 + ... + a_{1,n} x_n \le b_1$$
$$a_{2,1} x_1 + a_{2,2} x_2 + ... + a_{2,n} x_n \le b_2$$
$$...$$
$$a_{m,1} x_1 + a_{m,2} x_2 + ... + a_{m,n} x_n \le b_m$$
$$x_1 \ge 0, x_2 \ge 0, ..., x_n \ge 0$$

ili, u kompaktnijem obliku koristeći skraćenu notaciju za sumaciju, kao

$$\arg\max Z(\mathbf{x}) = \sum_{j=1}^{n} c_j x_j$$

p.o.

$$\sum_{j=1}^{n} a_{i,j} x_j \le b_i, \; i = 1.. m$$
$$x_j \ge 0, \; j = 1 .. n$$

Uvođenjem tzv. ***tehnološke matrice*** **A** te ***vektora ograničenja*** **b** i ***vektora cijena*** **c** prema definicijama

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \dots & \dots & \dots & \dots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{pmatrix} \qquad \mathbf{c} = \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_n \end{pmatrix}$$

problem se može zapisati u sljedećem, *veoma kompaktnom matrično-vektorskom obliku*, koji se naročito često koristi za potrebe raznih teoretskih razmatranja i izvođenja:

$$\arg\max Z(\mathbf{x}) = \mathbf{c}^{\mathrm{T}}\,\mathbf{x}$$

p.o.

$$\mathbf{A}\,\mathbf{x} \le \mathbf{b}$$
$$\mathbf{x} \ge 0$$

Pri tome, kako su u gornjim relacijama (nejednakostima) sa obje strane *vektori* a ne *brojevi*, te nejednakosti treba shvatiti u smislu da su *sve komponente* vektora sa lijeve strane u naznačenoj relaciji sa *odgovarajućim komponentama* vektora sa desne strane.

Iz gore izloženog je vidljivo da se problem može predstaviti kao zadatak nalaženja *uvjetnog ekstrema*, pri čemu su kako *funkcija cilja*, tako i *sva ograničenja* opisani *linearnim funkcijama*. Pored toga, kao specifična ograničenja, javljaju se *ograničenja na nenegativnost svih promjenljivih*.

Prethodni problem se može posmatrati i kao problem ***optimalne raspodjele ograničenog resursa na pojedine aktivnosti***. Naime, *kapaciteti mašina* se mogu posmatrati i kao *raspoloživi resursi*, dok se, s druge strane, *proizvodnje proizvoda* mogu posmatrati kao *aktivnosti* na koje se "*troši*" raspoloživi resurs. Cilj je tada da se resursi *dodjele* (*alociraju*) aktivnostima na takav način da se dobije *maksimalni efekat*, odnosno *optimalna vrijednost funkcije cilja*.

- ***Primjer***: Neko preduzeće plasira na tržište dvije vrste mljevene kafe K_1 i K_2. Očekivana zarada je 3 novčane jedinice (skraćeno n.j.) po kilogramu za kafu K_1 (tj. 3 n.j./kg), a 2 n.j./kg za kafu K_2. Pogon za prženje kafe je na raspolaganju 150 sati sedmično, a pogon za mljevenje kafe 60 sati sedmično. Utrošeni sati za prženje i mljevenje po kilogramu proizvoda dati su u sljedećoj tabeli:

	K_1	K_2
Pržionica	0.5 h/kg	0.3 h/hg
Mlin	0.1 h/kg	0.2 h/kg

Formirati matematski model iz kojeg se može odrediti *koliko treba proizvesti* kafe K_1 a koliko kafe K_2 tako da ukupna zarada bude maksimalna.

Ako sa x_1 označimo količinu kafe K_1 a sa x_2 količinu kafe K_2 koju treba proizvesti (u kilogramima), matematički model datog problema se može zapisati u sljedećem obliku:

$$\arg\max Z(\mathbf{x}) = 3\,x_1 + 2\,x_2$$

p.o.

$$0.5\,x_1 + 0.3\,x_2 \le 150$$
$$0.1\,x_1 + 0.2\,x_2 \le 60$$

$$x_1 \ge 0,\ x_2 \ge 0$$

Razmotrimo sada *problem optimalne mješavine*. Tipični zadatak formiranja *optimalne mješavine raspoloživih sirovina*, koja bi sadržavala zahtijevane količine "korisnih" sastojaka, može se definirati na sljedeći način. Potrebno je napraviti *mješavinu* od *n različitih sirovina* S_j, $j = 1 \,..\, n$, pri čemu mješavina mora sadržavati *m različitih korisnih sastojaka* K_i, $i = 1 \,..\, m$. Pri tome su poznati sljedeći činioci:

- *cijene* c_j, $j = 1 \,..\, n$ raspoloživih sirovina S_j respektivno;
- *neophodne količine* b_i, $i = 1 \,..\, m$ svakog od korisnih sastojaka K_i respektivno;
- *količine* $a_{i,j}$, $i = 1 \,..\, m$, $j = 1 \,..\, n$ korisnih sastojaka K_i u sirovini S_j.

Cilj je naći *optimalnu mješavinu*, odnosno odrediti respektivno *količine* x_j, $j = 1 .. n$, svake od sirovina S_j pa da ukupna sredstva potrebna za nabavku sirovina budu *što je god moguće manja* (tj. *minimalna*).

Razmotrimo kako se može zapisati matematski model ovog problema. Ako upotrijebimo *za sada nepoznatu količinu* x_j sirovine S_j, *trošak* ostvaren *nabavkom ukupne količine* sirovine S_j će biti $c_j x_j$. Stoga će *ukupan trošak* ostvaren nabavkom *ukupne količine svih vrsta sirovina* biti:

$$Z(\mathbf{x}) = Z(x_1, x_2, ..., x_n) = c_1 x_1 + c_2 x_2 + ... + c_n x_n = \sum_{j=1}^{n} c_j x_j$$

Ovdje je cilj ostvariti *minimalan efekat*, što znači da treba *minimizirati funkciju* $Z(\mathbf{x})$. Pošto je i u ovom slučaju *funkcija cilja linearna*, odnosno trošak (cijena) nabavke *jedne jedinice sirovine* je *konstantan* (ne zavisi od *količine nabavljenih sirovina*) i *pozitivan*, minimalni trošak bi se dobio pri *nultoj nabavci sirovina* S_j (ako isključimo *negativne nabavke* kao *nerealne*). Međutim, iz postavke problema vidimo da je mješavina *ograničena minimalnim količinama korisnih sastojaka*. Nabavka količine x_j sirovine S_j će osigurati korisne sastojke K_i u iznosu $a_{i,j} x_j$, tako da će *sve nabavljene sirovine* sadržavati korisne sastojke K_i u ukupnoj količini

$$a_{i,1} x_1 + a_{i,2} x_2 + ... + a_{i,n} x_n = \sum_{j=1}^{n} a_{i,j} x_j$$

Ukupna količina korisnih sastojaka K_i je ograničena *najmanjom zahtijevanom količinom* b_i, tako da mora vrijediti ograničenje

$$a_{i,1} x_1 + a_{i,2} x_2 + ... + a_{i,n} x_n = \sum_{j=1}^{n} a_{i,j} x_j \geq b_i$$

Postoji ukupno m ovakvih ograničenja, po jedno za svaki od m korisnih sastojaka.

Pored ovih ograničenja, postoji i *prirodno ograničenje* na količine nabavljenih sirovina, prema kojem se *ne mogu nabaviti negativne količine sirovina*. Stoga mora vrijediti

$$x_j \geq 0, j = 1 .. n.$$

Zadatak se, dakle, može napisati u obliku

$$\arg\min Z(\mathbf{x}) = c_1 x_1 + c_2 x_2 + ... + c_n x_n$$

p.o.

$$a_{1,1} x_1 + a_{1,2} x_2 + ... + a_{1,n} x_n \geq b_1$$
$$a_{2,1} x_1 + a_{2,2} x_2 + ... + a_{2,n} x_n \geq b_2$$
$$...$$
$$a_{m,1} x_1 + a_{m,2} x_2 + ... + a_{m,n} x_n \geq b_m$$
$$x_1 \geq 0, x_2 \geq 0, ... x_n \geq 0$$

ili, u kompaktnijem obliku koristeći skraćenu notaciju za sumaciju, kao

$$\arg\min Z(\mathbf{x}) = \sum_{j=1}^{n} c_j x_j$$

p.o.

$$\sum_{j=1}^{n} a_{i,j} x_j \geq b_i, \; i = 1 .. m$$
$$x_j \geq 0, \; j = 1 .. n$$

Uz slične definicije kao u prethodnoj vrsti problema, ovaj problem se također može napisati veoma kompaktno u matrično-vektorskom obliku:

$$\arg\min Z(\mathbf{x}) = \mathbf{c}^T \mathbf{x}$$

p.o.

$$\mathbf{A}\,\mathbf{x} \geq \mathbf{b}$$
$$\mathbf{x} \geq 0$$

Dakle, i u ovom slučaju se problem može predstaviti kao zadatak nalaženja *uvjetnog ekstrema,* pri čemu su kako *funkcija cilja,* tako i *sva ograničenja* opisani *linearnim funkcijama*. Pored toga, kao specifična ograničenja, javljaju se *ograničenja na nenegativnost svih promjenljivih*. Ovako definirani zadaci čije su karakteristike *linearnost funkcije kriterija,* te *linearnost svih relacija kojima se opisuju ograničenja* nazivaju se ***zadaci linearnog programiranja***.

Česta, mada ne i obavezna karakteristika tipičnih problema linearnog programiranja je *nenegativnost svih promjenljivih*. Zaista, mada je većina stvarnih zadataka linearnog programiranja takva da se zahtijeva nenegativnost svih promjenljivih, teoretski se mogu razmatrati zadaci linearnog programiranja u kojima se takva ograničenja *ne javljaju*. Pored toga, takvi problemi ponekad imaju i *realnu osnovu*. Ipak, treba naglasiti da svi uobičajeni modeli linearnog programiranja uvijek *podrazumijevaju* da se zahtijeva i nenegativnost svih promjenljivih, a ukoliko to nije slučaj, ta činjenica se obavezno *posebno naglašava*.

- ***Primjer*** : Potrebno je obezbijediti vitaminsku terapiju koja će sadržavati četiri vrste vitamina V_1, V_2, V_3 i V_4. Na raspolaganju su dvije vrste vitaminskih sirupa S_1 i S_2 čije su cijene 40 n.j./g i 30 n.j./g respektivno. Vitaminski koktel mora sadržavati najmanje 0.2 g, 0.3 g, 3 g i 1.2 g vitamina V_1, V_2, V_3 i V_4 respektivno. Sljedeća tabela pokazuje sastav pojedinih vitamina u obje vrste vitaminskih sirupa:

	V_1	V_2	V_3	V_4
S_1	10 %	0 %	50 %	10 %
S_2	0 %	10 %	30 %	20 %

  Formirati matematski model iz kojeg se može odrediti *koliko treba nabaviti* sirupa S_1 a koliko sirupa S_2 tako da ukupni trošak bude minimalan.

Ako sa x_1 označimo količinu sirupa S_1 a sa x_2 količinu sirupa S_2 koju treba nabaviti (u gramima), matematički model datog problema se može zapisati u sljedećem obliku:

$$\arg\min Z(\mathbf{x}) = 40\,x_1 + 30\,x_2$$

p.o.

$$0.1\,x_1 \geq 0.2$$
$$0.1\,x_2 \geq 0.3$$
$$0.5\,x_1 + 0.3\,x_2 \geq 3$$
$$0.1\,x_1 + 0.2\,x_2 \geq 1.2$$
$$x_1 \geq 0,\ x_2 \geq 0$$

Vidimo da se opet radi o klasičnom problemu linearnog programiranja.

U prethodna dva primjera, sva ograničenja su bila istog tipa (tj. tipa "manje ili jednako" odnosno tipa "veće ili jednako". Sasvim je moguće imati i probleme u kojima se javljaju *različiti tipovi nejednakosti*, pa čak i ograničenja iskazana *jednakostima*, kao što je prikazano u sljedećem primjeru.

- ***Primjer*** : Planira se proizvodnja tri tipa detrdženta D_1, D_2 i D_3. Sa trgovačkom mrežom je dogovorena isporuka tačno 100 kg detrdženta bez obzira na tip. Za uvoz odgovarajućeg repromaterijala planirano su sredstva u iznosu od 110 \$. Po jednom kilogramu detrdženta, za proizvodnju detrdženata D_1, D_2 i D_3 treba nabaviti repromaterijala u vrijednosti 2 \$, 1.5 \$ odnosno 0.5 \$. Također je planirano da se za proizvodnju uposle radnici sa angažmanom od ukupno barem 120 radnih sati, pri čemu je za proizvodnju jednog kilograma detrdženata D_1, D_2 i D_3 potrebno uložiti respektivno 2 sata, 1 sat odnosno 1 sat. Prodajna cijena detrdženata D_1, D_2 i D_3 po kilogramu respektivno iznosi 10 KM, 5 KM odnosno 8 KM. Formirati matematski model iz kojeg se može odrediti *koliko treba proizvesti* svakog od tipova detrdženata da se pri tome ostvari maksimalna moguća zarada.

Slično kao u dosadašnjim primjerima, ako sa x_1, x_2 i x_3 označimo respektivno količinu detrdženata D_1, D_2 i D_3 koju treba proizvesti (u kilogramima), matematički model datog problema se može zapisati u sljedećem obliku:

$$\arg\max Z(\mathbf{x}) = 10\,x_1 + 5\,x_2 + 8\,x_3$$

p.o.

$$x_1 + x_2 + x_3 = 100$$
$$2\,x_1 + 1.5\,x_2 + 0.5\,x_3 \leq 110$$
$$2\,x_1 + x_2 + x_3 \geq 120$$

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

Vidimo da se u ovom modelu javljaju nejednakosti različitih tipova. Također, treba istaći da prisustvo ograničenja tipa jednakosti efektivno *umanjuje dimenzionalnost problema*. Zaista, prisustvo ograničenja $x_1 + x_2 + x_3 = 100$ dovodi do toga da tri promjenljive x_1, x_2 i x_3 *nisu nezavisne*, već se jedna od njih može posmatrati kao *zavisna*. Tako, ako recimo promjenljivu x_3 izrazimo preko promjenljivih x_1 i x_2 pomoću relacije $x_3 = 100 - x_1 - x_2$, možemo ovaj izraz za x_3 uvrstiti u funkciju cilja i ostale nejednakosti (ne smijemo zaboraviti i nejednakost $x_3 \geq 0$), čime problem dobija sljedeći oblik:

$$\arg\max Z(\mathbf{x}) = 2\,x_1 - 3\,x_2 + 800$$

p.o.

$$1.5\,x_1 + x_2 \leq 60$$
$$x_1 \geq 20$$
$$x_1 + x_2 \leq 100$$

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

Na ovaj način je dimenzionalnost problema smanjena sa tri na dvije (nezavisne) promjenljive. Ipak, treba primijetiti neke sitnije detalje. Prvo, u funkciji cilja *pojavio se i slobodni član*. Ovo ni na koji način ne mijenja način rješavanja problema. Naime, taj slobodni član možemo privremeno ignorirati (jer on ni od čega ne ovisi), a na kraju samo u optimalnom rješenju vrijednost funkcije cilja uvećamo za taj slobodni član. Drugo, interesantno je da su se u funkciji cilja pojavili i *negativni koeficijenti*. To se često dešava kao posljedica raznih *matematskih transformacija* nad polaznim modelom. Zbog toga, svi praktični metodi za rješavanje problema linearnog programiranja moraju ispravno raditi *bez obzira na znak pojedinih parametara koji se javljaju u modelu*, tim prije što se negativni koeficijenti u funkciji cilja i/ili u ograničenjima mogu pojaviti (mada ne tako često) već i u polaznim modelima *realnih problema*, kao što ilustrira sljedeći primjer.

- ***Primjer***: Teretni brod prebacuje dvije vrste tereta. Zbog prirode tereta, za svaku utovarenu tonu prve vrste tereta brodovlasnik treba platiti taksu od 500 KM, dok za svaku utovarenu tonu druge vrste tereta brodovlasnik dobija "kaparu" (predujam) od 200 KM. Gotovina kojim brodovlasnik raspolaže na početku utovara iznosi 3000 KM i brodovlasnik nema načina da u trenutku utovara nabavi veću količinu novca. Naravno, novac koji se dobije kao predujam zbog utovara druge vrste tereta može se koristiti za plaćanje takse za prvu vrstu tereta. Utovar prve vrste tereta traje pola sata po toni, a druge vrste tereta 15 minuta po toni. Ukupno vrijeme koje je na raspolaganju za utovar je najviše 12 sati, a u jednom trenutku može se utovarati samo jedna vrsta tereta. Zarada od prevoza prve vrste tereta iznosi 2000 KM po toni, a od drugog 100 KM po toni, koja se isplaćuje po obavljenom prevozu (u ovaj iznos nije uračunata ranije spomenuta "kapara"). Formirati matematski model iz kojeg se može odrediti koliko treba utovariti svake vrste tereta tako da se njegovim transportom ostvari maksimalna zarada.

Označimo sa x_1 i x_2 količine jedne odnosno druge vrste tereta koje treba prevesti (u tonama). Jasno je da tada ukupna zarada u KM koja se ostvaruje po osnovu prevoza iznosi $2000\,x_1 + 100\,x_2$. Međutim, to nisu sve komponente koje utiču na ukupan prihod. Prvo, kako po svakoj utovarenoj toni prve vrste tereta treba platiti 500 KM takse, to ukupan prihod u KM umanjuje za $500\,x_1$. S druge strane, za svaku utovarenu tonu drugog tereta dobija se 200 KM "kapare", što ukupan prihod u KM uvećava za $200\,x_2$. Slijedi da funkcija cilja koju treba maksimizirati glasi

$$Z(\mathbf{x}) = 2000\,x_1 + 100\,x_2 - 500\,x_1 + 200\,x_2 = (2000 - 500)\,x_1 + (100 + 200)\,x_2 = 1500\,x_1 + 300\,x_2$$

Razmotrimo sad ograničenja. Prvo ograničenje koje imamo je ograničenje na ukupno vrijeme utovara. Jasno je da ukupno vrijeme utovara za obje vrste tereta u satima iznosi $(1/2)\,x_1 + (1/4)\,x_2$, s obzirom da se

utovar obje vrste tereta ne može vršiti istovremeno. Kako je raspoloživo vrijeme za utovar ograničeno na 12 sati, to dovodi do ograničenja

$$(1/2)\,x_1 + (1/4)\,x_2 \le 12$$

Alternativno smo mogli sve vremenske jedinice umjesto u satima izraziti u minutama. U tom slučaju bi ukupno vrijeme utovara u minutama iznosilo $30\,x_1 + 15\,x_2$. Kako u 12 sati ima 720 minuta, to bi nas dovelo do ograničenja $30\,x_1 + 15\,x_2 \le 720$ koje je ekvivalentno prethodnom ograničenju.

Pored ograničenja na vrijeme utovara, imamo i ograničenje koje nameće činjenica da za utovar prve vrste robe treba plaćati posebnu taksu, a raspoloživa sredstva za plaćanje takse su ograničena. Kako se po svakoj toni plaća taksa od 500 KM, ukupna taksa koju treba platiti u KM iznosi $500\,x_1$. Ovaj iznos ne smije preći raspoloživa sredstva za plaćanje takse, koja se opet sastoje od početne gotovine od 3000 KM, te dodatnih sredstava koja se dobijaju na osnovu "kapare" za utovar druge vrste tereta, koja iznose $200\,x_2$ (u KM), tako da ukupna raspoloživa sredstva u KM iznose $3000 + 200\,x_2$. Ovo nas dovodi do ograničenja

$$500\,x_1 \le 3000 + 200\,x_2$$

koje se, nakon što se sve promjenljive prebace na lijevu stranu, može prikazati i u obliku:

$$500\,x_1 - 200\,x_2 \le 3000$$

Ovo ograničenje se može protumačiti i da ukupna "taksa" koju smijemo platiti ne smije preći 3000 KM, pri čemu se za prvu vrstu tereta plaća taksa od 500 KM po toni, dok se za drugu vrstu tereta "plaća negativna taksa" od –200 KM po toni (negativna znači da se ne radi o trošku nego o dobitku). Stoga se matematski model postavljenog zadatka može predstaviti u obliku

$$\arg\max Z(\mathbf{x}) = 3\,x_1 + 2\,x_2$$

p.o.

$$(1/2)\,x_1 + (1/4)\,x_2 \le 12$$
$$500\,x_1 - 200\,x_2 \le 3000$$

$$x_1 \ge 0,\ x_2 \ge 0$$

Ono što je interesantno u ovom modelu je pojava *negativnog koeficijenta* u jednom od ograničenja, iz čega slijedi da se i to može desiti. Isto tako, primijetimo da je kojim slučajem taksa koju treba platiti za utovar prve vrste tereta bila veća od naknade koja se ostvaruje po osnovu prevoza, tada bi i koeficijent uz x_1 u funkciji cilja također bio negativan (što pokazuje da se i to može desiti). Naravno, intuitivno je jasno da bi u optimalnom rješenju tada bilo $x_1 = 0$, odnosno prva vrsta tereta ne bi bila uopće isplativa za transport, jer se više plaća takse za prevoz tog tereta nego što se zarađuje njegovim transportom.

Osnovna terminologija i obilježavanje u linearnom programiranju

Sam naziv *model linearnog programiranja* se u literaturi često ravnopravno koristi sa terminima *zadatak linearnog programiranja* i *problem linearnog programiranja*, ili prosto samo *linearno programiranje*. Ovdje će se sa istim značenjem, u zavisnosti od konteksta, koristiti svi pomenuti nazivi. Pored toga, termin "problem" bez dodatka "linearnog programiranja" (ili nekog drugog dodatka koji praktično označava odgovarajući matematički model) se koristi kao *izvorni problem*, odnosno ono što *prethodi formiranju modela* (dok još i ne znamo kakav je model), na osnovu čega se model formira. Model je u tom smislu određena *idealizacija* (aproksimacija) *stvarnog problema*.

U modelu linearnog programiranja, za označavanje različitih elemenata (komponenti) modela tipično se koriste sljedeći *simboli* i *nazivi*:

- Z – funkcija *kriterija*, funkcija *cilja*, ukupna *mjera učinka*;
- x_j – *promjenljiva, nivo aktivnosti j* ($j = 1 \,..\, n$);
- c_j – *cijena, efekat, dobit, porast funkcije cilja* kao rezultat *porasta svake jedinice nivoa aktivnosti j* ($j = 1 \,..\, n$);
- b_i – *desna strana ograničenja*, količina resursa *i* koja je *na raspolaganju aktivnostima i* ($i = 1 \,..\, m$)
- $a_{i,j}$ – *količina resursa i* koju uzima (troši) *svaka jedinica aktivnosti j* ($i = 1 \,..\, m$, $j = 1 \,..\, n$).

Veličine c_j, b_i i $a_{i,j}$ (za $i = 1 .. m$ i za $j = 1 .. n$) predstavljaju *ulazne konstante* odnosno *parametre* modela. U terminologiji zadataka *donošenja odluka*, promjenljive x_1, x_2, ..., x_n se nazivaju *promjenljive odlučivanja*.

Svaka linearna nejednačina ili jednačina koju mora zadovoljiti rješenje naziva se ***ograničenje***. Skup svih ograničenja (tj. skup linearnih nejednačina i/ili jednačina koje mora zadovoljiti rješenje) naziva se ***skup ograničenja*** ili samo ***ograničenja***. Tačka (uređena n-torka promjenljivih x_i, $i = 1 .. n$) koja zadovoljava sva ograničenja naziva se ***dopustiva*** (***dozvoljena***) ***tačka***. Skup svih dopustivih tačaka je ***dopustivi*** (***dozvoljeni***) ***prostor*** ili ***dopustiva*** (***dozvoljena***) ***oblast***. *Broj* (*nezavisnih*) *promjenljivih* koje se javljaju u problemu naziva se ***dimenzionalnost*** problema.

Svako ograničenje koje je u nekoj tački ispunjeno *na jednakost* (tj. tako da su lijeva i desna strana ograničenja jednake) naziva se ***kritično ograničenje*** za tu tačku. Ukoliko za neku tačku postoji *više kritičnih ograničenja nego što iznosi dimenzionalnost problema*, takva tačka je ***degenerirana tačka***.

Može se desiti da je dopustivi prostor *prazan skup*, odnosno da su ograničenja međusobno *nesaglasna*. U tom slučaju, problem *nema rješenja* (u praksi, to obično znači da je proble, *loše postavljen*). Također, dopustivi prostor može biti *ograničen* ili *neograničen* skup. Pokazuje se da ako je dopustivi prostor ograničen skup, tada zadatak linearnog programiranja *sigurno ima konačno optimlalno rješenje*. Pri tome, optimalno rješenje može biti ili *jedinstveno* (tj. postiže se samo u jednoj tački), ili postoji *beskonačno mnogo tačaka* u kojima se postiže optimalno rješenje. Jasno je da u ovom drugom slučaju, funkcija cilja mora imati *istu vrijednost* u svakoj od tih tačaka.

Ako je dopustivi prostor *neograničen*, tada se zavisno od situacije može desiti da zadatak linearnog programiranja *ima konačno optimalno rješenje*, ali se isto tako može desiti da se optimalno rješenje nalazi *u beskonačnosti*, odnosno da funkcija cilja *nije ograničena* odozgo ili odozdo (ovisno tražimo li maksimum ili minimum) na dopustivoj oblasti. Ponekad se u takvim slučajevima također kaže da optimalno rješenje *ne postoji*, mada treba imati u vidu da je to drugi oblik "nepostojanja" u odnosu na situaciju kada su ograničenja nesaglasna.

Prema konvenciji, u zadacima linearnog programiranja (kao i u matematičkom programiranju općenito) se *svaka specifikacija* vrijednosti promjenljivih $(x_1, x_2, .., x_n)$ naziva *rješenje*, mada to *ne mora biti željeni* pa čak *ni dopustivi izbor*. Radi toga se definiraju različiti tipovi rješenja:

- ***dopustivo*** (***dozvoljeno***) ***rješenje*** je rješenje koje *zadovoljava sva ograničenja*;
- ***nedopustivo*** (***nedozvoljeno***) ***rješenje*** je rješenje koje *ne zadovoljava barem jedno ograničenje*, odnosno kod kojeg je *barem jedno ograničenje narušeno*;
- ***optimalno*** (***najbolje***) ***rješenje*** je ono dopustivo rješenje kod kojeg funkcija cilja ima *najbolju* (minimalnu ili maksimalnu) vrijednost.

Geometrijska interpretacija modela linearnog programiranja

Radi lakšeg razumijevanja zadatka (modela) linearnog programiranja, kao i boljeg razumijevanja osnovnih svojstava (osobenosti) ove vrste problema, često se razmatra njihova *geometrijska intepretacija*. Za tu svrhu, obično se analiziraju najjednostavniji, odnosno *dvodimenzionalni* zadaci linearnog programiranja. U načelu, grafičko predstavljanje je principijelno (mada uz znatne poteškoće) moguće i za *trodimenzionalne* zadatke, dok za zadatke veće dimenzionalnosti nikakva grafička interpretacija nije moguća.

Kao što je već rečeno, dimenzionalnost n zadatka linearnog programiranja određena je *brojem promjenljivih* x_j, $j = 1 .. n$. Međutim, *vrijednost funkcije cilja* Z *također predstavlja promjenljivu* (i to *zavisnu*), tako da se isti zadatak može posmatrati i u n+1-dimenzionalnom prostoru. U skladu s tim, zadatak linearnog programiranja koji zavisi od *dvije promjenljive* x_1 i x_2 moguće je ravnopravno posmatrati *kako u dvodimenzionalnom prostoru* (x_1, x_2), *tako i u trodimenzionalnom prostoru* (x_1, x_2, Z). Stoga, da bi se na što jasniji način predstavile osobenosti zadataka linearnog programiranja, u nastavku će se *dvodimenzionalni* zadatak linearnog programiranja predstaviti kako u *dvodimenzionalnom* prostoru (x_1, x_2), tako i u *trodimenzioalnom* prostoru (x_1, x_2, Z).

Razmotrimo prvo interpretaciju u dvodimenzionalnom prostoru (x_1, x_2). Kako je funkcija cilja Z funkcija *dvije nezavisne promjenljive*, nju je u dvodimenzionalnom prostoru moguće grafički predstaviti jedino *linijama njenih konstantnih vrijednosti* (tj. *nivo linijama*), i *pravcem najbržeg rasta funkcije* (*gradijentom*). Porast funkcije Z je predstavljen pomjeranjem nivo linije *u smjeru gradijenta*, dok se pad funkcije Z dešava pri pomjeranju nivo linije *u smjeru suprotnom od smjera gradijenta*.

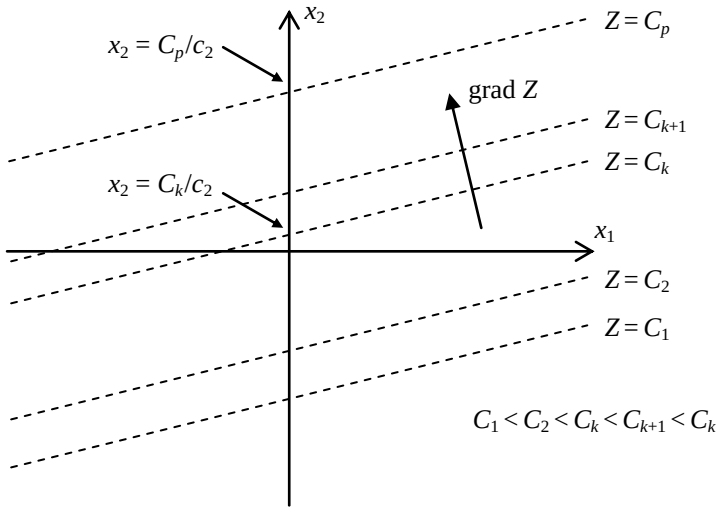
Nivo linije funkcije cilja (tj. linije *konstantnih vrijednosti funkcije cilja*) u dvodimenzionalnom koordinatnom sistemu date su jednačinama:

$$Z = c_1 x_1 + c_2 x_2 = C$$

gdje konstanta C predstavlja odgovarajuću vrijednost funkcije cilja. U dvodimenzionalnom prostoru (x_1, x_2) ova jednačina predstavlja *jednačinu pravca* koja se, u *eksplicitnom obliku*, može napisati kao

$$x_2 = k\, x_1 + l = -\frac{c_1}{c_2} x_1 + \frac{C}{c_2}$$

gdje je $k = -c_1/c_2$ *koeficijent smjera* ovog pravca, a $l = C/c_2$ je *dužina koju ovaj pravac odsjeca na x_2 osi*. Pored toga, vidljivo je da ovi pravci prolaze kroz tačke $(0, C/c_2)$ i $(C/c_1, 0)$. Pošto je koeficijent smjera za sve ove pravce *konstantan* (ne zavisi od veličine C), za razne vrijednosti konstante C će se dobiti *familija međusobno paralelnih pravaca* koji se nazivaju ***pravci funkcije cilja*** Z. Vrijednosti konstante C se mogu birati u opsegu od $-\infty$ do ∞, te se na taj način pravcima funkcije cilja može *prekriti čitav dvodimenzionalni prostor* (x_1, x_2), kako je prikazano na sljedećoj slici.



Pravac najbržeg porasta (ili *pada*) *funkcije cilja* možemo *intuitivno* odrediti *kao pravac koji je okomit na pravce funkcije cilja*, jer se tako ostvaruje *najbrže presjecanje*. Ovo je u potpunosti u skladu sa poznatim činjenicama iz matematske analize prema kojima je *najbrži porast neke funkcije* u smjeru njenog *gradijenta*, koji za ovu funkciju glasi

$$\text{grad } Z = \left(\frac{\partial Z}{\partial x_1}, \frac{\partial Z}{\partial x_2}\right)^{\mathrm{T}} = (c_1, c_2)^{\mathrm{T}}$$

Odavde vidimo da je ovaj vektor zaista *okomit* na pravce funkcije cilja, jer za njegov koeficijent smjera $k_{grad} = c_2/c_1$ i koeficijent smjera pravaca funkcije cilja k vrijedi $k\, k_{grad} = -1$. Međutim, to isto se može izvesti i na drugi način, koji se ne oslanja na naprednije elemente matematičke analize. Posmatrajmo dva pravca funkcije cilja, koji prolaze kroz dvije različite tačke $((x_1)_k, (x_2)_k)$ i $((x_1)_{k+1}, (x_2)_{k+1})$:

$$Z_k = C_k = c_1 (x_1)_k + c_2 (x_2)_k$$

$$Z_{k+1} = C_{k+1} = c_1 (x_1)_{k+1} + c_2 (x_2)_{k+1}$$

Promjena funkcije cilja pri prelasku sa jednog na drugi pravac iznosi

$$\Delta Z = Z_{k+1} - Z_k = c_1 ((x_1)_{k+1} - (x_1)_k) + c_2 ((x_2)_{k+1} - (x_2)_k)$$

ili, drugačije napisano,

$$\Delta Z = c_1 \Delta x_1 + c_2 \Delta x_2$$

gdje je $\Delta x_1 = (x_1)_{k+1} - (x_1)_k$ i $\Delta x_2 = (x_2)_{k+1} - (x_2)_k$. Neka smo prilikom prelaska sa tačke $((x_1)_k, (x_2)_k)$ na tačku $((x_1)_{k+1}, (x_2)_{k+1})$ napravili korak *određene fiksne dužine*, odnosno neka se ove tačke nalaze na *određenoj fiksnoj razdaljini* (recimo, 1). Bez obzira na fiksiranu *dužinu* koraka, mogu se dobiti *različite vrijednosti* promjene funkcije cilja ΔZ *u zavisnosti od toga u kojem se pravcu taj korak realizira*. Pošto je ukupni pomjeraj (po obje koordinate) po pretpostavci jednak 1, možemo pisati

$$\sqrt{\Delta x_1^{\,2} + \Delta x_2^{\,2}} = 1$$

odakle slijedi

$$\Delta x_2 = \sqrt{1 - \Delta x_1^{\,2}}$$

Uvrštavanjem u izraz za ΔZ, dobijamo

$$\Delta Z = c_1 \Delta x_1 + c_2 \sqrt{1 - \Delta x_1^{\,2}}$$

Da bismo odredili za koju vrijednost priraštaja Δx_1 će se dobiti *maksimalna promjena funkcije cilja* ΔZ, potrebno je naći *izvod* izraza za ΔZ po Δx_1 i *izjednačiti ga sa nulom*:

$$\frac{\Delta Z}{\Delta x_1} = c_1 - c_2 \frac{\Delta x_1}{\sqrt{1 - \Delta x_1^{\,2}}} = 0$$

Posljednju relaciju možemo napisati i kao

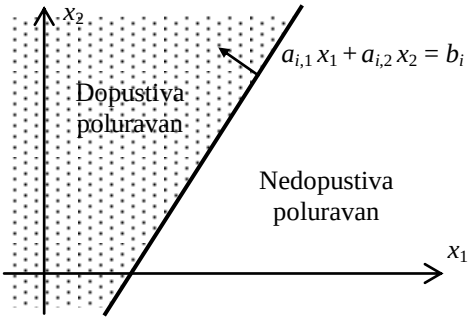
$$c_1 - c_2 \frac{\Delta x_1}{\Delta x_2} = 0$$

odakle neposredno slijedi

$$\Delta x_2 = \frac{c_2}{c_1} \Delta x_1$$

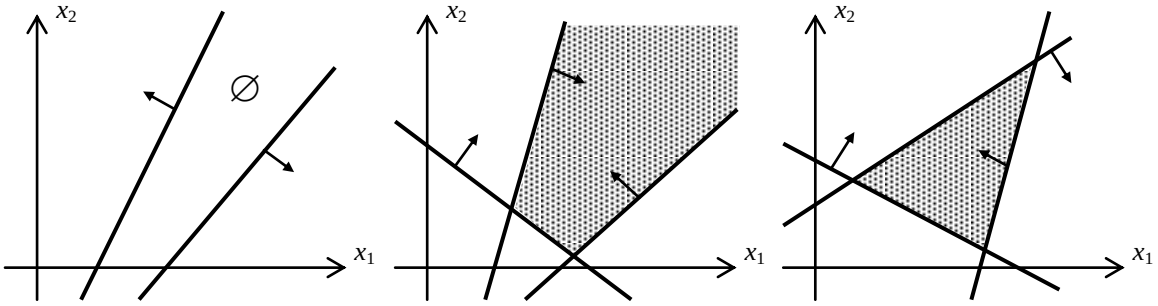
Odavde ponovo vidimo da je najveći prirast funkcije cilja u smjeru pravca koji ima koeficijent smjera c_2/c_1. Kako je koeficijent smjera pravaca funkcije cilja $k = -c_1/c_2$, *pravci najbržeg prirasta su okomiti na pravce funkcije cilja*, što je i trebalo pokazati.

Razmotrimo sad kakav je geometrijski smisao *ograničenja*. Poznato je da se linearne jednačine u dvodimenzionalnom prostoru grafički prikazuju kao *pravci*. Stoga se nejednačine mogu grafički prikazati kao *poluravni*. Svaka nejednačina dijeli prostor na dva dijela: na ***dopustivu poluravan***, odnosno skup tačaka koje *zadovoljavaju nejednačinu* (*dopustiva oblast u odnosu na tu nejednačinu*), te na ***nedopustivu poluravan***, odnosno skup tačaka koje *ne zadovoljavaju nejednačinu* (*nedopustiva oblast u odnosu na tu nejednačinu*). Konkretno, ako je ograničenje zadano u obliku nejednačine $a_{i,1} x_1 + a_{i,2} x_2 \le b_i$, onda jednačina $a_{i,1} x_1 + a_{i,2} x_2 = b_i$ predstavlja pravac u dvodimenzionalnom prostoru (x_1, x_2) koji dijeli taj prostor na dva dijela – dopustivu i nedopustivu poluravan, odnosno dopustivu i nedopustivu oblast u odnosu na tu nejednačinu.

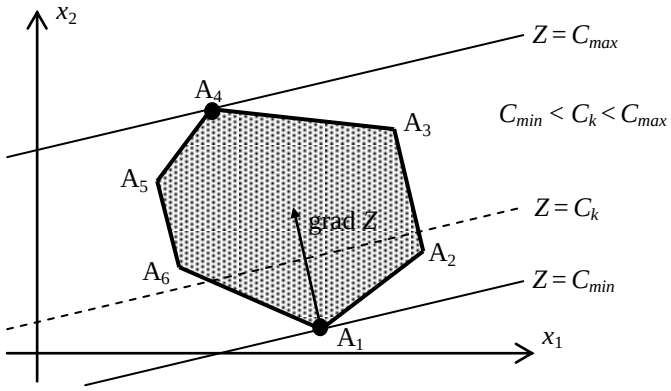


U zavisnosti od toga kako su zadana ograničenja, mogu se dobiti razni tipovi dopustivih oblasti, koje su zapravo *presjeci svih dopustivih poluravni*. Na prvom mjestu, dopustiva oblast može biti *prazan* ili

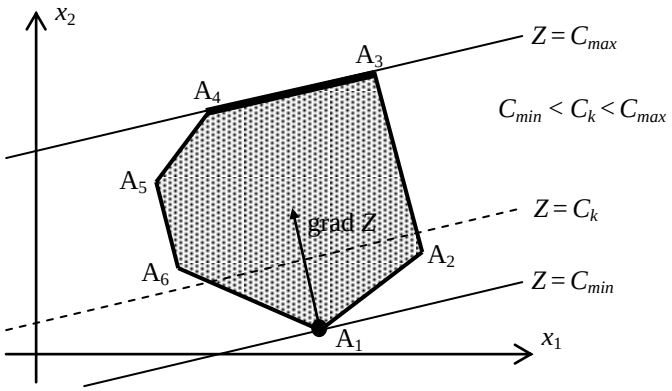
neprazan skup. Ako je dopustiva oblast neprazan skup, ona može biti kako *ograničena*, tako i *neograničena*. Specijalno, ako je dopustiva oblast ograničen neprazan skup, on je uvijek *poligon*, na osnovu poznate geometrijske činjenice da ukoliko je presjek konačno mnogo poluravni neprazan i ograničen, tada je on uvijek poligon. Geometrijska predstava raznih tipova dopustivih oblasti data je na sljedećoj slici.



Ako dopustiva oblast postoji i ako je *ograničena*, tada će promjenom konstante C od $-\infty$ do ∞ paraleni pravci funkcije cilja će za neko $C = C_{min}$ *prvi put* "*dodirnuti*" *dopustivu oblast* a za neko $C = C_{max}$ *posljednji put imati tačku u dopustivoj oblasti*. Ove veličine C_{min} i C_{max} prestavljaju *minimalne i maksimalne vrijednosti funkcije cilja* a odgovarajuće tačke na dopustivoj oblasti su *tačka minimuma* i *tačka maksimuma*. Na sljedećoj slici, to su tačke A_1 i A_4 respektivno.



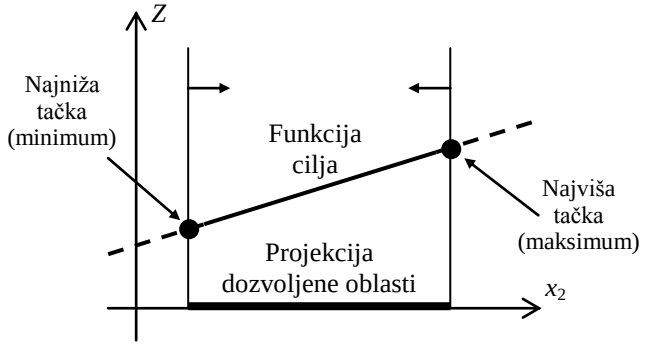
Iz provedenog razmatranja sasvim je jasno da se optimalno rješenje (bilo minimum, bilo maksimum) može nalaziti *isključivo u nekom od tjemena* poligona koji predstavlja dozvoljenu oblast. Jedini slučaj u kojem bi se moglo desiti *da neka tačka koja nije tjeme bude optimalna* je u slučaju ukoliko bi se neki od mnoštva paralelnih pravaca funkcije cilja *poklopili sa nekom od stranica* poligona koji predstavlja dozvoljenu oblast, kao na sljedećoj slici. Međutim, u tom slučaju će *svaka tačka te stranice biti optimalno rješenje*. Tako je, u primjeru sa sljedeće slike, *svaka tačka duži* A_3A_4 *maksimum funkcije cilja uz zadana ograničenja*.



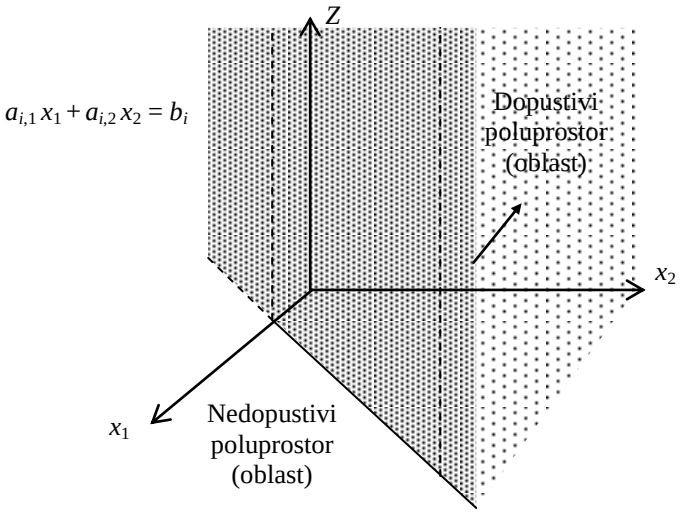
Može se pokazati da zaključak da se optimalno rješenje mora nalaziti *ili u nekom od vrhova dozvoljene oblasti* (u slučaju kada je rješenje jedinstveno) *ili duž jedne cijele stranice dozvoljene oblasti* (u slučaju kada

rješenje nije jedinstveno) vrijedi *ne samo za probleme linearnog programiranja koji imaju dvije nezavisne promjenljive*, već za *probleme linearnog programiranja općenito*.

Razmotrimo sada kako bi izgledala geometrijska interpretacija modela linearnog programiranja sa dvije nezavisne promjenljive *u trodimenzionalnom prostoru* (x_1, x_2, Z). U trodimenzionalnom prostoru, grafik svake linearne funkcije je *ravan*, pa je i grafik funkcije cilja, zadane u obliku $Z = c_1 x_1 + c_2 x_2$ također ravan. Ako se isključi trivijalan slučaj kada su $c_1 = 0$ i $c_2 = 0$, funkcija cilja je, geometrijski posmatrano, *beskonačna* "*kosa ravan*", čiji ekstrem (maksimum ili minimum) treba naći. Jasno je da se *najviša* (*maksimalna*) odnosno *najniža* (*minimalna*) tačka takve ravni *nalazi u beskonačnosti*, te da konačan ekstrem može postojati *samo uz ograničenja*, odnosno ukoliko nam je *kretanje u smjeru porasta* (odnosno *pada*) funkcije cilja *na neki način ograničeno*. Također je jasno da se u tom slučaju ekstrem mora nalaziti *na granici dozvoljene oblasti*. Osnovnu ideju ovoga ilustrira sljedeća slika, koja prikazuje *projekciju stvarne situacije* iz trodimenzionalnog prostora u ravan (x_2, Z).



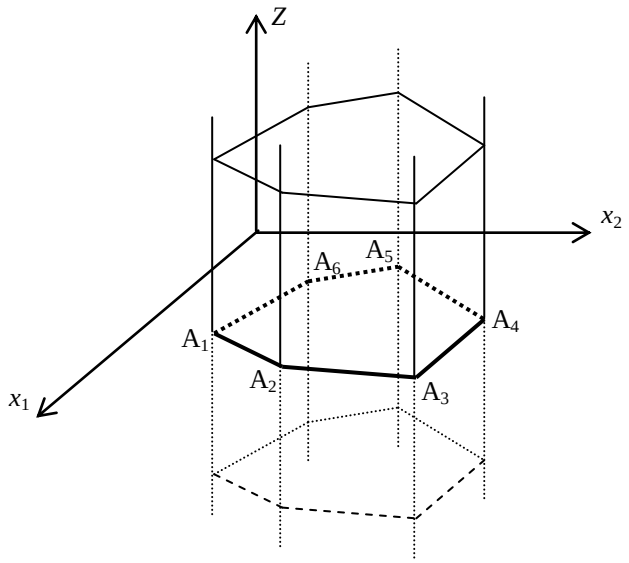
Da bismo mogli kompletirati trodimenzionalnu sliku, moramo razmotriti i kakav je *geometrijski smisao ograničenja u trodimenzionalnom prostoru*. Kako je poznato da se sve linearne jednačine u trodimenzionalnom prostoru grafički prikazuju kao *ravni*, nejednačine se mogu grafički prikazati kao *skupovi tačaka prostora koje se nalaze sa iste strane poluravni*, odnosno kao *poluprostori*. Stoga svaka nejednačina dijeli prostor na dva dijela: na ***dopustivi poluprostor***, odnosno skup tačaka koje *zadovoljavaju nejednačinu* (*dopustiva oblast u odnosu na tu nejednačinu*), te na ***nedopustivi poluprostor***, odnosno skup tačaka koje *ne zadovoljavaju nejednačinu* (*nedopustiva oblast u odnosu na tu nejednačinu*). Konkretno, ako je ograničenje zadano u obliku nejednačine $a_{i,1} x_1 + a_{i,2} x_2 \le b_i$, onda jednačina $a_{i,1} x_1 + a_{i,2} x_2 = b_i$ predstavlja ravan u trodimenzionalnom prostoru (x_1, x_2, Z) *paralelnu sa Z-osom*, odnosno *okomitu na ravan* (x_1, x_2). Ova ravan dijeli prostor na dva dijela – dopustivi i nedopustivi poluprostor, odnosno dopustivu i nedopustivu oblast u odnosu na tu nejednačinu. Dakle, ravan $a_{i,1} x_1 + a_{i,2} x_2 = b_i$, poput *vertikalnog zida*, razdvaja dopustivu od nedopustive oblasti, kao što je prikazano na sljedećoj slici.



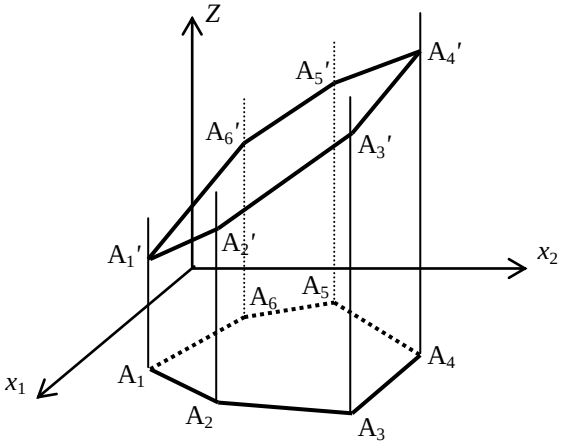
S obzirom da su ograničenja zadana skupom nejednačina, onda se dopustiva oblast nalazi *u presjeku* dopustivih poluprostora definiranih pojedinim ograničenjima. Pošto su svi skupovi opisani linearnim jednačinama i nejednačinama *konveksni skupovi* i pošto je *presjek konveksnih skupova također konveksan*

skup, slijedi da je dopustiva oblast u zadatku linearnog programiranja *uvijek konveksan skup*. Ovaj zaključak vrijedi *bez obzira na dimenzionalnost problema*.

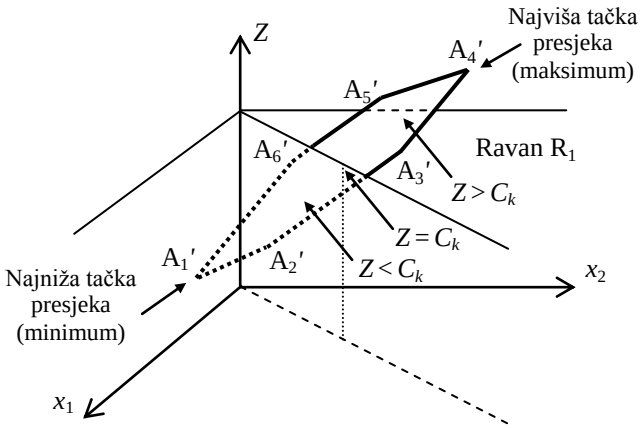
Ako je, u posmatranom dvodimenzionalnom problemu, dopustiva oblast *neprazan ograničen skup*, ona će u trodimenzionalnom prostoru imati oblik "c*ijevi*" (*okomite prizme*) *neograničene visine*, čije su stranice *paralelne sa Z-osom* i koja je *okomita na ravan* (x_1, x_2). Osnovica ove prizme je *konveksni poligon*. Primjer jedne ovakve "prizmatične" dopustive oblasti, čija je osnovica poligon $A_1A_2A_3A_4A_5A_6$, prikazan je na sljedećoj slici.



Razmotrimo sada na koji način se manifestira *najbolje* (*optimalno*) *rješenje* u ovakvoj interpretaciji. S obzirom da je funkcija cilja *beskonačna* "*kosa*" *ravan* u trodimenzionalnom prostoru (x_1, x_2, Z) dok je skup ograničenja *prizma neograničene visine* u istom prostoru, ravan funkcije cilja će *presjecati* prizmu ograničenja. Taj presjek ravni i prizme je, u općem slučaju također *nakošen* i ima svoju *najvišu* i *najnižu* (dopustivu) tačku u *vrhovima* (*tjemenima*) *presjeka*. Te tačke, najviša i najniža, predstavljaju respektivno *maksimum*, odnosno, *minimum* funkcije cilja *pri datim ograničenjima*. Na sljedećoj slici,.tačka A_4' predstavlja maksimum a tačka A_1' minimum fukcije cilja Z u dopustivoj oblasti koja je zadana poligonom $A_1A_2A_3A_4A_5A_6$.



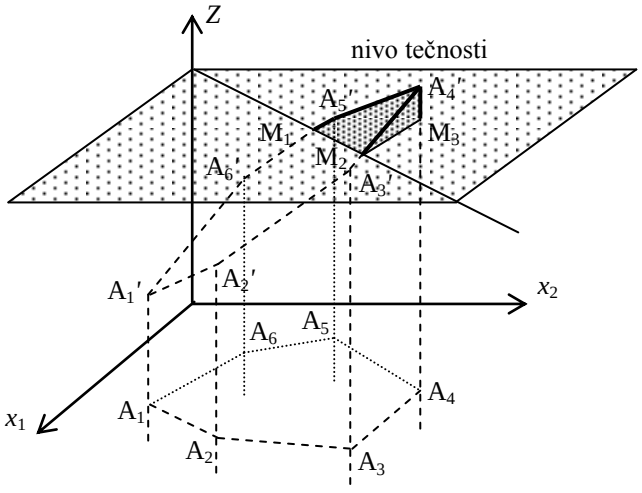
Površina presjeka, koju predstavlja poligon $A_1'A_2'A_3'A_4'A_5'A_6'$, je *nakošena* u odnosu na (x_1, x_2) ravan. Ako se taj poligon presječe sa ravni R_1 koja je *paralelna* sa (x_1, x_2) ravni i koja ima visinu C_k u odnosu na ravan (x_1, x_2) pri čemu je $Z(A_1) \le C_k \le Z(A_4)$, *linija presjeka* će također biti paralelna sa (x_1, x_2) ravni, tj. na toj će liniji *vrijednost funkcije cilja biti konstantna* i iznosiće $Z = C_k$. Pored toga, linija presjeka će *dijeliti poligon na dva dijela*, na dio na kojem sve tačke imaju visinu *manju* od C_k i na dio na kojem sve tačke imaju visinu *veću* od C_k. Ovo je ilustrirano na sljedećoj slici.



Bilo koja tačka na nakošenom poligonu koja se nalazi *ispod* linije $Z = C_k$, imaće vrijednost funkcije cilja *manju* od C_k, i obrnuto. Drugim rječima, linija $Z = C_{max}$ *ne smije presjecati nakošeni poligon u tački koja je takva da od nje ima ijedna viša tačka unutar dopustive oblasti*. Kako s druge strane tačka maksimuma *mora biti dopustiva*, odnosno *mora se nalaziti u dopustivoj oblasti*, oba uvjeta (nepresjecanje i dopustivost) se mogu zadovoljiti samo ako linija $Z = C_k$ *dodiruje* odnosno *tangira dopustivu oblast.* Tako se dobija tačka koja je *dopustiva*, a koja je takva da *nema nijedna tačka na većoj visini od nje u dopustivom prostoru*. Analogno zaključivanje se može sprovesti i za nalaženje najniže tačke (minimuma).

Iz provedenog razmatranja slijedi da se *ekstremna vrijednost funkcije cilja mora nalaziti u jednom od tjemena dopustivog prostora*, što smo već zaključili i razmatranjem provedenim u dvije dimenzije. U specijalnom slučaju, ekstremna vrijednost funkcije cilja se može nalaziti na jednoj od *stranica dopustive oblasti* (ako je prava $Z = C_k$ paralelna sa odgovarajućom stranicom dopustive oblasti), ali čak i u tom slučaju, ekstremna vrijednost funkcije cilja se također nalazi i u tjemenu (tačnije, u *dva tjemena*) dopustive oblasti.

Prethodno razmatranje se može jednostavno predstaviti kao uranjanje u tečnost "cijevi" čija je gornja strana koso odsječena. Kada se takva "cijev" okomito uranja u tečnost, granica umočenog i suhog dijela (omeđena tačkama M_1, M_2 i M_3 na sljedećoj slici) će biti *vodoravna*. *Posljednja tačka koja će se potopiti* je tačka, koja se nalazi *na vrhu presjeka*, odnosno tačka A_4' na slici. Upravo je to tražena tačka maksimuma.



- ***Primjer*** : Koristeći grafičku analizu, riješiti problem linearnog programiranja

$$\arg\max Z = 3x_1 + x_2$$

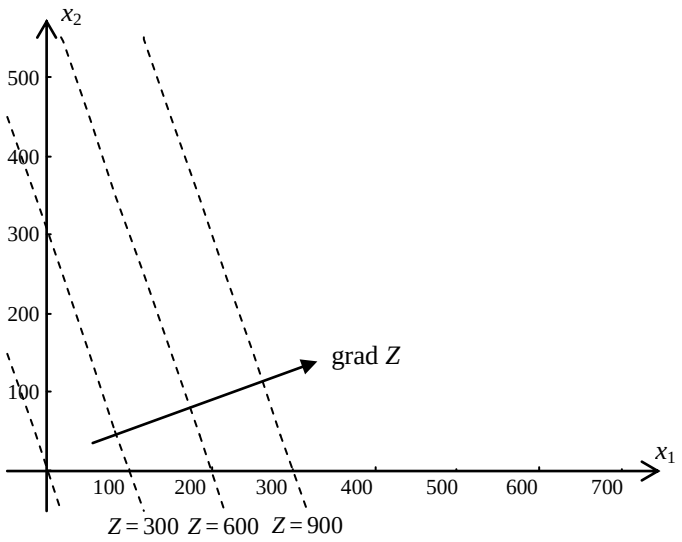
p.o.

$$0.5x_1 + 0.3x_2 \le 150$$
$$0.1x_1 + 0.2x_2 \le 60$$

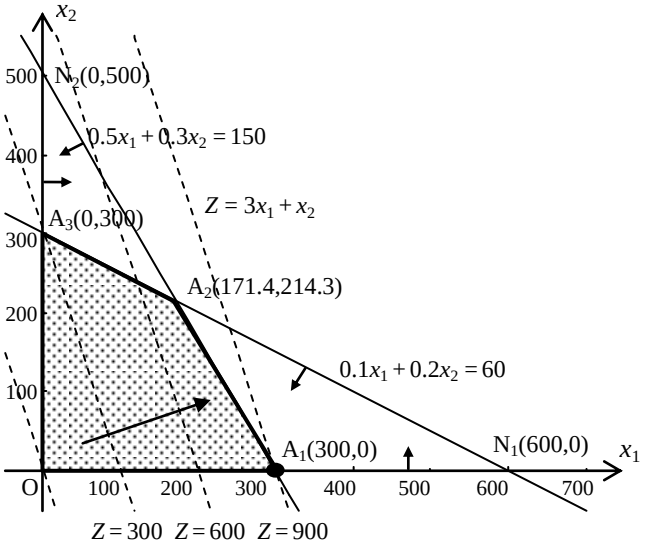
$$x_1 \ge 0,\ x_2 \ge 0$$

Postavljeni zadatak ćemo prvo grafički predstaviti u *dvodimenzionalnom* prostoru (x_1, x_2) a zatim u *trodimenzionalnom* prostoru (x_1, x_2, Z). U dvodimenzionalnom slučaju, nivo linije funkcije cilja, odnosno linije konstantnih vrijednosti funkcije cilja, date su jednačinama $3x_1 + x_2 = C$, pri čemu konstanta C predstavlja odgovarajuću vrijednost funkcije cilja. Ovo je jednačina pravca, koji se može napisati u eksplicitnom obliku kao $x_2 = C - 3x_1$, tako da je njegov koeficijent smjera $k = -3$, dok dužina odsječka koju taj pravac odsjeca na x_2 osi iznosi $l = C$.

Kako je koeficijent smjera *konstanta* (ne zavisi od veličine C), za razne vrijednosti konstante C dobija se familija međusobno paralelnih pravaca (pravaca funkcije cilja Z). Kako se vrijednosti kostante C mogu birati u proizvoljnom rasponu od $-\infty$ do $-\infty$, na taj način se pravcima funkcije cilja može prekriti čitav dvodimenzionalni postor (x_1, x_2). Za određeni interval vrijednosti konstante C, pravci funkcije cilja će presjecati dopustivu oblast ukoliko takva uopće postoji. Na sljedećoj slici je prikazano nekoliko karakterističnih pravaca funkcije cilja za vrijednosti C (odnosno Z) 0, 300, 600 i 900 respektivno, zajedno sa smjerom porasta funkcije cilja.



Kao što znamo, kod problema linearnog programiranja u dvodimenzionalnom prostoru, dopustiva oblast određena je presjekom *dopustivih poluravni*, čije su granice zadane *jednačinama pravaca*. Granice dopustivih poluravni, dopustiva oblast (prikazana osjenčeno) i pravci funkcije cilja ilustrirani su na sljedećoj slici:



U konkretnom primjeru, te granice su sljedeće:

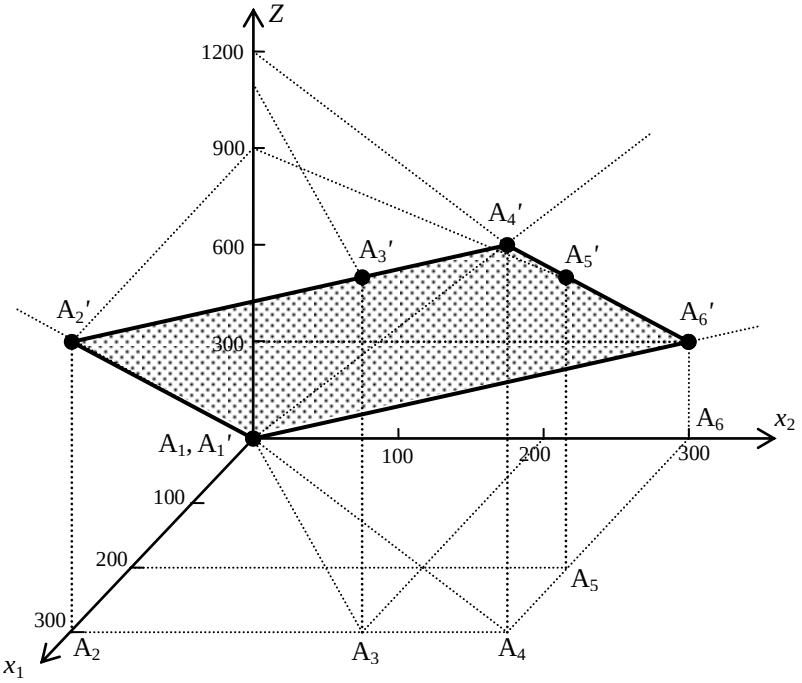
- Granica generirana ograničenjem $0.5\,x_1 + 0.3\,x_2 \le 150$ je pravac $0.5\,x_1 + 0.3\,x_2 = 150$ koji x_1-osu i x_2-osu presjeca respektivno u tačkama $x_1 = 150/0.5 = 300$ i $x_2 = 150/0.3 = 500$, odnosno koji prolazi kroz tačke $(0, 500)$ i $(300, 0)$;
- Granica generirana ograničenjem $0.1\,x_1 + 0.2\,x_2 \le 60$ je pravac $0.1\,x_1 + 0.2\,x_2 = 60$ koji x_1-osu i x_2-osu presjeca respektivno u tačkama $x_1 = 60/0.1 = 600$ i $x_2 = 60/0.2 = 300$, odnosno koji prolazi kroz tačke $(0, 300)$ i $(600, 0)$;
- Granica generirana ograničenjem $x_1 \ge 0$ je pravac $x_1 = 0$, tj x_2-osa;
- Granica generirana ograničenjem $x_2 \ge 0$ je pravac $x_2 = 0$, tj x_1-osa.

Četiri pravca koja predstavljaju granice dopustivih poluravni sijeku se (dvije po dvije) u ukupno šest tačaka, od kojih su neke *dopustive* (tj. *pripadaju* dopustivom prostoru), a neke su *nedopustive* (tj. *ne pripadaju* dopustivom prostoru), što se lako može provjeriti uvrštavanjem njihovih koordinata u nejednačine kojima se opisuju ograničenja. Koordinate presječnih tačaka dobijaju se rješavanjem sistema jednačina koje obrazuju jednačine pravaca koji se presjecaju. Te presječne tačke u konkretnom primjeru su sljedeće:

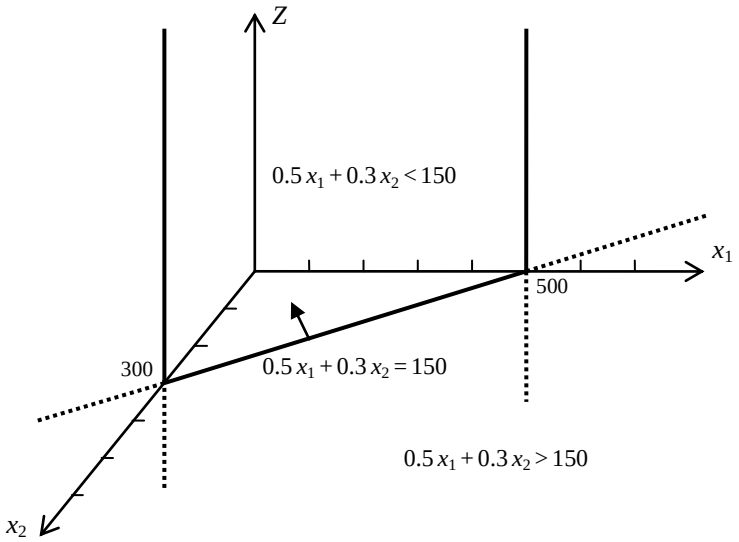
- Presjek graničnih pravaca $x_1 = 0$ i $x_2 = 0$ je *dopustiva* tačka $O(0, 0)$;
- Presjek graničnih pravaca $0.5\,x_1 + 0.3\,x_2 = 150$ i $x_2 = 0$ je *dopustiva* tačka $A_1(300, 0)$;
- Presjek graničnih pravaca $0.5\,x_1 + 0.3\,x_2 = 150$ i $0.1\,x_1 + 0.2\,x_2 = 60$ formira *dopustivu* tačku $A_2(1200/7, 1500/7)$ odnosno, približno zaokruženo na jednu decimalu, $A_2(171.4, 214.3)$;
- Presjek graničnih pravaca $0.1\,x_1 + 0.2\,x_2 = 60$ i $x_1 = 0$ je *dopustiva* tačka $A_3(0, 300)$;
- Presjek graničnih pravaca $0.1\,x_1 + 0.2\,x_2 = 60$ i $x_1 = 0$ je *nedopustiva* tačka $N_1(600, 0)$ (jer ne zadovoljava ograničenje $0.5\,x_1 + 0.3\,x_2 \le 150$);
- Presjek graničnih pravaca $0.5\,x_1 + 0.3\,x_2 = 150$ i $x_1 = 0$ je *nedopustiva* tačka $N_2(0, 500)$ (jer ne zadovoljava ograničenje $0.1\,x_1 + 0.2\,x_2 \le 60$);

Na prethodnoj slici, dopustivi prostor je predstavljen poligonom $OA_1A_2A_3$. Funkcija cilja je, kao što je to uobičajeno u dvodimenzionalnom prikazu, predstavljena *linijama konstantne vrijednosti*, odnosno *nivo linijama*. Te linije su pravci koji zadovoljavaju jednačine $3\,x_1 + x_2 = C_k$ za razne vrijednosti C_k (na slici su prikazani slučajevi $C_k = 0$, $C_k = 300$, $C_k = 600$ i $C_k = 900$). Pri tome, pravac $3\,x_1 + x_2 = 900$ *dodiruje* poligon $OA_1A_2A_3$ u tački A_3. U toj tački funkcija cilja ima svoju *najveću dopustivu vrijednost*. Stoga je ta tačka *optimalno rješenje* ($x_1 = 300$, $x_2 = 0$) i u njoj je optimalna vrijednost funkcije cilja $Z = 900$.

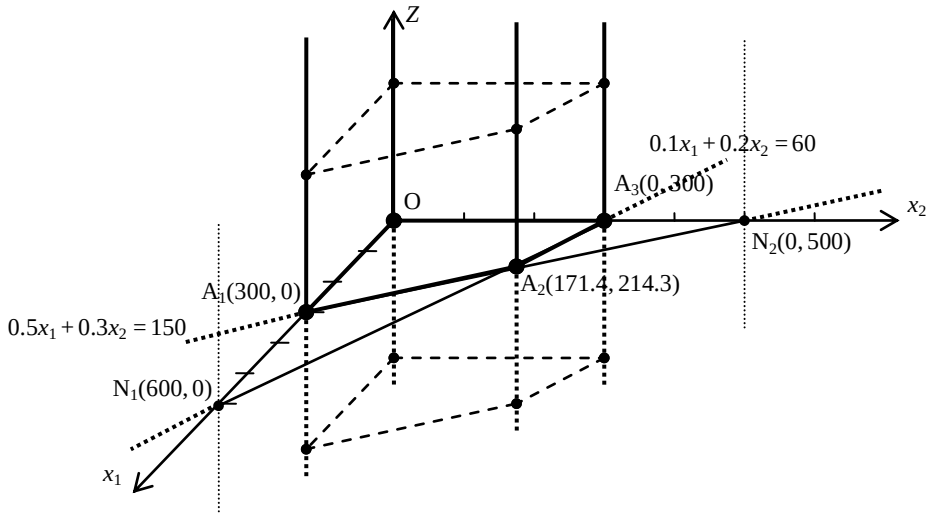
Razmotrimo sada reprezentaciju u *trodimenzionalnom* prostoru (x_1, x_2, Z). U tom prostoru, funkcija cilja $Z = 3\,x_1 + x_2$ je *ravan* koja prolazi kroz tačku $(0, 0, 0)$. Njen presjek sa ravni $x_1 = 0$ formira pravac $Z = x_2$, a presjek sa ravni $x_2 = 0$ pravac $Z = 3\,x_1$. Uzmemo li nekoliko nasumice odabranih tačaka u ravni (x_1, x_2), recimo $A_1(0, 0)$, $A_2(300, 0)$, $A_3(300, 200)$, $A_4(300, 300)$, $A_5(200, 300)$ i $A_6(0, 300)$, vrijednosti funkcije cilja u tim tačkama respektivno iznose 0, 900, 1100, 1200, 900 i 300. Slijedi da ravan funkcije cilja prolazi kroz tačke $A_1'(0, 0, 0)$, $A_2'(300, 0, 900)$, $A_3'(300, 200, 1100)$, $A_4'(300, 300, 1200)$, $A_5'(200, 300, 900)$ i $A_6'(0, 300, 300)$. Ovo je grafički prikazano na sljedećoj slici.



Već smo rekli da su granice dopustive oblasti u trodimenzionalnom prostoru (x_1, x_2, Z) zadane jednačinama ravni koje su paralelne sa Z–osom odnosno, koje su okomite na ravan (x_1, x_2). Tako je granica ograničenja $0.5\,x_1 + 0.3\,x_2 \le 150$ ravan $0.5\,x_1 + 0.3\,x_2 = 150$ paralelna sa Z-osom koja x_1-osu i x_2-osu presjeca u tačkama $x_1 = 150 / 0.5 = 300$ i $x_2 = 150 / 0.3 = 500$, pa je možemo grafički predstaviti kao na sljedećoj slici.

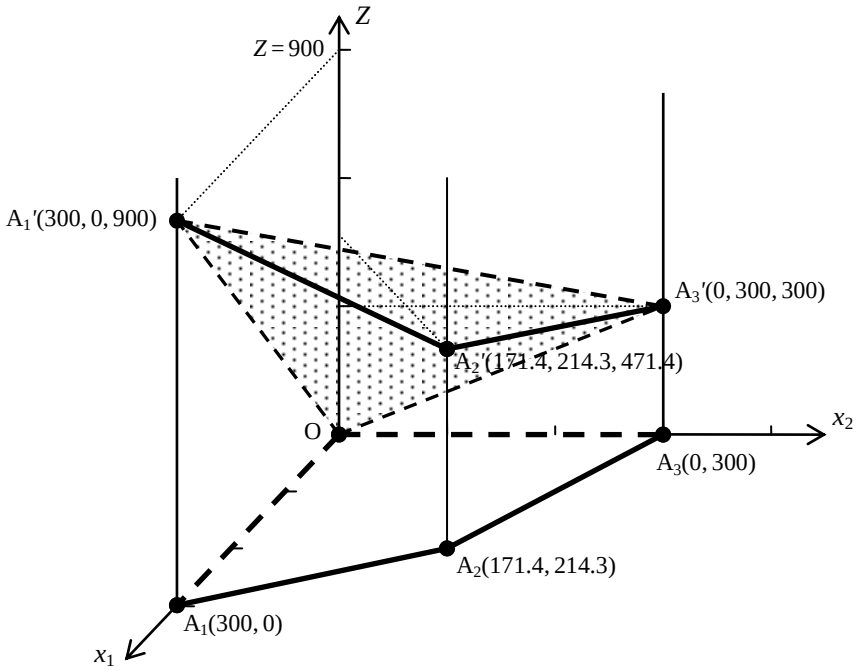


Granica ograničenja $0.1\,x_1 + 0.2\,x_2 \le 60$ je, također, ravan $0.1\,x_1 + 0.2\,x_2 = 60$ paralelna sa Z-osom koja x_1-osu i x_2-osu presjeca respektivno u tačkama $x_1 = 60/0.1 = 600$ i $x_2 = 60/0.2 = 300$. Konačno, granice ograničenja $x_1 \ge 0$ odnosno $x_2 \ge 0$ su ravni $x_1 = 0$ odnosno $x_2 = 0$. Ove četiri ravni imaju *šest mogućih presjeka* – presječnih pravaca. Od tih šest presječnih pravaca, u ovom konkretnom primjeru, *četiri pripadaju dopustivoj oblasti a dva ne pripadaju*. Ova ograničenja definiraju dopustivu oblast u obliku *beskonačno dugačke cijevi* (*prizme*) paralelne sa Z-osom i koja ima poligon $OA_1A_2A_3$ kao osnovicu, što je ilustrirano na sljedećoj slici.



Sada je potrebno naći *presjek ravni funkcije cilja* i "*cijevi*" *dopustivog prostora*. U prvoj dopustivoj tački $O(0, 0)$ vrijednost funkcije cilja je 0, što znači da ravan funkcije cilja prolazi kroz tačku $(0, 0, 0)$ trodimenzionalnog prostora (x_1, x_2, Z). Presjek ravni funkcije cilja i ravni $x_2 = 0$ je pravac $Z = 3\,x_1$ koji u dopustivi prostor ulazi u tački $(0, 0, 0)$ a završava u tački $A_1'(300, 0, 3x_1)$ odnosno $A_1'(300, 0, 900)$. Presjek ravni funkcije cilja i ravni $x_1 = 0$ je pravac $Z = x_2$ koji u dopustivi prostor ulazi u tački $(0, 0, 0)$ a završava u tački $A_3'(0, 300, x_2)$ odnosno $A_3'(0, 300, 300)$. Presjek ravni funkcije cilja i ravni $0.5\,x_1 + 0.3\,x_2 = 150$ je prava koja u dopustivi prostor ulazi u tački $A_1'(300, 0, 900)$ a završava u tački $A_2'(171.4, 214.3, 3\,x_1 + x_2)$ odnosno $A_2'(171.4, 214.3, 471{,}4)$. Konačno, presjek ravni funkcije cilja i ravni $0.1\,x_1 + 0.2\,x_2 = 60$ je prava koja u dopustivi prostor ulazi u tački $A_3'(0, 300, 300)$ a završava u tački $A_2'(171.4, 214.3, 471{,}4)$.

Iz provedene analize slijedi da trodimenzionalna ravan funkcije cilja $Z = 3x_1 + x_2$ presjeca "cijev" dopustivog prostora u tačkama O, A_1', A_2' i A_3', u kojima koordinate Z imaju vrijednosti 0, 900, 471.4 i 300 respektivno. Najviša dopustiva tačka je A_1' sa vrijednošću koordinate $Z = 900$, što predstavlja *maksimalnu* vrijednost funkcije cilja. Ovo je grafički predstavljeno na sljedećoj slici.



Ponovo dolazimo do istog zaključka da tačka A_1 predstavlja optimalno rješenje i da je $Z = 900$ optimalna vrijednost funkcije cilja.

Oblici modela linearnog programiranja

Bez obzira na široku rasprostranjenost modela linearnog programiranja, interesantno je da u literaturi *ne postoji usaglašenost* oko naziva pojedinih oblika metematičkog modela linearnog programiranja. Tako se, na primjer, model kod kojeg se traži *maksimum funkcije cilja* i kod kojeg su *sva ograničenja nejednačine tipa* "*manje ili jednako*" (ne računajući prirodna ograničenja na nenegativnost promjenljivih), u užbeniku "Operaciona istraživanja" *J. Petrića* naziva ***standardna forma***. Ista definicija se koristi i u knjizi "Operations Research – A Practical Introduction" *M. W. Cartera* i *C. C. Pricea*. S druge strane, ista stvar se u udžbeniku "Uvod u matematičko programiranje" *L. Neralića* naziva ***naša standardna forma*** uz obrazloženje da postoje i drugačije definicije standardne forme, dok se u udžbeniku "Operaciona istraživanja" *D. Cvetičanina* koristi naziv ***standardni problem maksimuma***. Međutim, ta šarolikost ne bi bila toliki problem da neki autori pod "standardnom formom" ne podrazumijevaju nešto drugo. Recimo, u knjizi "Matematički priručnik" *I. N. Bronsteina* i dr. koristi se termin standardna forma za ono što mnogi nazivaju ***prošireni model linearnog programiranja***. To je model u kojem se javljaju *kako ograničenja u vidu nejednakosti, tako i ograničenja u vidu jednakosti*. S druge strane, u knjizi "Introduction to Operations Research" *F. S. Hilliera* i *G. J. Liebermana* se pod standardnom formom naziva ***prošireni zadatak linearnog programiranja*** (model u kojem su sva ograničenja, osim onih na nenegativnost promjenljivih, izražena u formi jednakosti), što opet neki drugi autori nazivaju ***kanonskom formom***. Ponovo to ne bi bio problem da neki autori ne zovu kanonskom formom ono što smo ovdje na početku nazvali *standardnom formom*. Negdje se definira i ***normalni oblik zadatka linearnog programiranja***, kao prošireni zadatak linearnog programiranja u kojem se posebno uvode dodatne promjenljive sa ciljem da se ograničenja tipa nejednakosti svedu na jednakosti. Ova definicija se koristi recimo u knjizi "Optimization Theory with Applications" *D. A. Pierrea*. Recimo još da su termini "oblik" i "forma" zapravo sinonimi, a u literaturi se mogu susresti obje varijante. Sve u svemu, može se zaključiti da u raspoloživoj literaturi vlada *potpuno nesaglasje* oko terminologije koja se koristi za imenovanje pojedinih modela linearnog programiranja. Ovo odsustvo standardizavije u terminologiji je prilično čudno, s obzirom da tehnike linearnog programiranja predstavljaju dobro uhodanu metodologiju, koja je poznata već decenijama. U svakom slučaju, prije konsultiranja bilo kakve literature iz oblasti linearnog programiranja, potrebno prvo provjeriti *koje nazive autori koriste* za razne vrste modela linearnog programiranja, da kasnije ne bi bilo problema.

Ovdje će biti definirana četiri oblika modela linearnog programiranja: ***standardni oblik***, ***opći oblik***, ***normalni oblik*** i ***kanonski oblik***. Zbog navedenih neusaglašenosti oko naziva pojedinih oblika, ovdje će biti predloženi nazivi koji, po ocjeni autora, imaju najviše osnova s obzirom na prethodne definicije i usaglašenost sa definicijama u dostupnoj literaturi (mada ponovo treba istaći da su česti autori koji pod kanonskim oblikom smatraju ono što ćemo mi nazvati standardnim oblikom ili normalnim oblikom, a pod standardnim oblikom ono što ćemo mi nazvati normalnim oblikom ili kanonskim oblikom).

Razmotrimo najprije *standardni oblik* modela linearnog programiranja. Kako je zadatak linearnog programiranja najčešće definiran kao *opći problem dodjele* (*alokacije*) *ograničenih resursa na konkurentne aktivnosti na najbolji način*, takav problem se može najbolje matematički opisati kao zadatak nalaženja *maksimuma funkcije* na koju su postavljena ograničenja u obliku *nejednačina tipa* "*manje ili jednako*" i ograničenja na *nenegativnost promjenljivih*. Zbog toga se matematički model linearnog programiranja u standarnom obliku može iskazati kao

$$\arg\max Z = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

p.o.

$$a_{1,1} x_1 + a_{1,2} x_2 + \ldots + a_{1,n} x_n \le b_1$$
$$a_{2,1} x_1 + a_{2,2} x_2 + \ldots + a_{2,n} x_n \le b_2$$
$$\ldots$$
$$a_{m,1} x_1 + a_{m,2} x_2 + \ldots + a_{m,n} x_n \le b_m$$
$$x_1 \ge 0,\ x_2 \ge 0,\ \ldots\ x_n \ge 0$$

ili, u kompaktnijem obliku koristeći skraćenu notaciju za sumaciju, kao

$$\arg\max Z = \sum_{j=1}^{n} c_j x_j$$

p.o.

$$\sum_{j=1}^{n} a_{i,j} x_j \le b_i,\ i = 1..\ m$$
$$x_j \ge 0,\ \ j = 1\ ..\ n$$

gdje su $a_{i,j}$, b_j i c_i za $i = 1\ ..\ m,\ \ j = 1\ ..\ n$ zadani realni brojevi. Prvih m uvjeta (ograničenja) u ovom modelu nazivaju se ***funkcionalna ograničenja*** ili ***strukturna ograničenja***, a preostali uvjeti se nazivaju ***nenegativna ograničenja***, ***uvjeti nenegativnosti*** ili ***prirodna ograničenja*** čije je tipično tumačenje da korišteni resursi ne mogu biti negativni. Uvođenjem prikladnih matrica, prethodni model se može veoma kompaktno zapisati u matrično-vektorskom obliku kao

$$\arg\max Z = \mathbf{c}^{\mathrm{T}} \mathbf{x}$$

p.o.

$$\mathbf{A}\,\mathbf{x} \le \mathbf{b}$$
$$\mathbf{x} \ge 0$$

Bez obzira na veliku generalnost, prethodni model ipak nije sasvim općenit, jer se zadatak linearnog programiranja može definirati i u drugim oblicima:

- Umjesto *maksimizacije* može se zahtijevati *minimizacija*;
- Neka (ili sva) funkcionalna ograničenja mogu biti nejednakosti tipa "veće ili jednako", odnosno mogu imati formu $a_{i,1} x_1 + a_{i,2} x_2 + \ldots + a_{i,n} x_n \ge b_i$ za neke vrijednosti i;
- Neka (ili sva) funkcionalna ograničenja mogu biti zadana u vidu *jednačina*, odnosno mogu imati formu $a_{i,1} x_1 + a_{i,2} x_2 + \ldots + a_{i,n} x_n = b_i$ za neke vrijednosti i;
- Neke promjenljive ne moraju biti *nenegativne*, odnosno x_j ne mora biti nenegativno za neke vrijednosti j.

Bilo koji zadatak u kojem se javlja *bilo koja kombinacija* ovih oblika je također zadatak linearnog programiranja. Uzimajući u obzir naprijed navedene oblike u kojima se također može definirati model linearnog programiranja, može se postaviti *opći oblik modela linearnog programiranja* kao

$$\arg \left\{ \begin{matrix} \max \\ \min \end{matrix} \right\} Z = c_1 x_1 + c_2 x_2 + ... + c_n x_n$$

p.o.

$$a_{1,1} x_1 + a_{1,2} x_2 + ... + a_{1,n} x_n \lesseqgtr b_1$$
$$a_{2,1} x_1 + a_{2,2} x_2 + ... + a_{2,n} x_n \lesseqgtr b_2$$
$$...$$
$$a_{m,1} x_1 + a_{m,2} x_2 + ... + a_{m,n} x_n \lesseqgtr b_m$$
$$x_{j_1} \geq 0,\ x_{j_2} \geq 0,\ ...\ x_{j_k} \geq 0$$

gdje su $a_{i,j}$, b_j i c_i za $i = 1 .. m$, $j = 1 .. n$ zadani realni brojevi, $j_1, j_2, ..., j_k$ su neki indeksi iz opsega od 1 do n, dok znak $\lesseqgtr$ označava da se na tom mjestu može nalaziti bilo koja od oznaka "≤", "≥" ili "=".

Opći oblik problema linearnog programiranja može se lako svesti na standardni. Naime, ograničenja tipa "≥" lako se principijelno svode na ograničenja tipa "≤" *množenjem sa* –1, odnosno ograničenje poput

$$a_{i,1} x_1 + a_{i,2} x_2 + ... + a_{i,n} x_n \geq b_i$$

ekvivalentno je ograničenju

$$-a_{i,1} x_1 - a_{i,2} x_2 - ... - a_{i,n} x_n \leq -b_i$$

Također, ograničenje tipa jednakosti poput

$$a_{i,1} x_1 + a_{i,2} x_2 + ... + a_{i,n} x_n = b_i$$

može se zamijeniti sa dva ograničenja tipa nejednakosti

$$a_{i,1} x_1 + a_{i,2} x_2 + ... + a_{i,n} x_n \leq b_i$$
$$a_{i,1} x_1 + a_{i,2} x_2 + ... + a_{i,n} x_n \geq b_i$$

odnosno

$$a_{i,1} x_1 + a_{i,2} x_2 + ... + a_{i,n} x_n \leq b_i$$
$$-a_{i,1} x_1 - a_{i,2} x_2 - ... - a_{i,n} x_n \leq -b_i$$

Ipak, treba napomenuti da se ovakvim transformacijama dobijaju nejednakosti u kojima je *desna strana negativna*. Kako postoje postupci za rješavanje problema linearnog programiranja kojima *ta činjenica može praviti problem* (na prvom mjestu, tu spada *simpleks algoritam*), to svakako treba imati u vidu. Štaviše, neki autori čak smatraju da je nenegativnost koeficijenata sa desne strane nejednakosti nužna da bi se neki model uopće mogao smatrati standardnim modelom, tako da se prema tim autorima modeli koji sadrže nejednakosti čija je desna strana negativna ne smatraju standardnim.

Što se tiče situacija u kojima neke od promjenljivih *ne moraju biti nenegativne*, te situacije se također mogu *standardizirati* uvođenjem novih promjenljivih, o čemu će kasnije biti više riječi. Naime, svaka promjenljiva x_j koja *ne mora biti nenegativna* može se zamijeniti sa dvije promjenljive x_j' i x_j" koje *moraju biti nenegativne* pomoću smjene $x_j = x_j' - x_j''$.

Sada ćemo razmotriti *normalni oblik* modela linearnog programiranja. Naime, sve nejednačine koje se nalaze u funkcionalnim ograničenjima, mogu se *svesti na jednačine dodavanjem dodatnih izravnavajućih promjenljivih*, koje *moraju zadovoljavati uvjet nenegativnosti*. Recimo, ograničenje

$$a_{1,1} x_1 + a_{1,2} x_2 + ... + a_{1,n} x_n \leq b_1$$

može se transformirati u oblik

$$a_{1,1} x_1 + a_{1,2} x_2 + ... + a_{1,n} x_n + x_{n+1} = b_1$$

pri čemu dodatna promjenljiva x_{n+1} mora zadovoljavati uvjet $x_{n+1} \geq 0$. Slična transformacija se može izvršiti i u ostalim ograničenjima, pri čemu je u različita ograničenja potrebno uvoditi *različite izravnavajuće promjenljive*. Na taj način se dobija *model linearnog programiranja u normalnom obliku* koji se može prikazati kao

$$\arg\max Z = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

p.o.

$$\begin{aligned} &a_{1,1} x_1 + a_{1,2} x_2 + \ldots + a_{1,n} x_n + x_{n+1} &&= b_1 \\ &a_{2,1} x_1 + a_{2,2} x_2 + \ldots + a_{2,n} x_n \quad + x_{n+2} &&= b_2 \\ &\ldots \\ &a_{m,1} x_1 + a_{m,2} x_2 + \ldots + a_{m,n} x_n \quad + x_{n+m} &&= b_m \end{aligned}$$

$$x_1 \geq 0,\ x_2 \geq 0,\ \ldots\ x_n \geq 0,\ x_{n+1} \geq 0,\ x_{n+2} \geq 0,\ \ldots,\ x_{n+m} \geq 0$$

ili, u kompaktnijem obliku koristeći skraćenu notaciju za sumaciju, kao

$$\arg\max Z = \sum_{j=1}^{n} c_j x_j$$

p.o.

$$\sum_{j=1}^{n} a_{i,j} x_j + x_{n+i} = b_i,\ i = 1..m$$

$$x_j \geq 0,\ \ j = 1..n+m$$

gdje su $a_{i,j}$, b_i i c_j, $i = 1..m$, $j = 1..n$ zadani realni brojevi. Pored toga, često se koriste i dva još kompaktnija matrično-vektorska oblika zapisa ove vrste problema

$$\arg\max Z = \mathbf{c}^{\mathrm{T}} \mathbf{x}$$

p.o.

$$\mathbf{A}\,\mathbf{x} + \mathbf{x}_{\mathrm{D}} = \mathbf{b}$$
$$\mathbf{x} \geq 0,\ \mathbf{x}_{\mathrm{D}} \geq 0$$

i

$$\arg\max Z = \mathbf{c}^{\mathrm{T}} \mathbf{x}$$

p.o.

$$(\mathbf{A}\ \mathbf{I}) \begin{pmatrix} \mathbf{x} \\ \mathbf{x}_{\mathrm{D}} \end{pmatrix} = \mathbf{b}$$
$$\mathbf{x} \geq 0,\ \mathbf{x}_{\mathrm{D}} \geq 0$$

pri čemu je $\mathbf{x}_{\mathrm{D}} = (x_{n+1}, x_{n+2}, \ldots, x_{n+m})^{\mathrm{T}}$ *vektor dopunskih* odnosno *izravnavajućih promjenljivih*, a **I** je *jedinična matrica* formata $m \times m$.

U prethodno prikazanom modelu, ulogu izravnavajućih promjenljivih igraju promjenljive čiji su indeksi od n+1 do n+m. U načelu, nije neophodno da indeksacija bude takva da bi se model smatrao normalnim. Ono po čemu se svaka dopunska (izravnavajuća) promjenljiva ističe u odnosu na polazne promjenljive je to što svaka takva promjenljiva *figurira samo u jednom strukturnom ograničenju* i što je koeficijent koji uz nju stoji *uvijek jednak jedinici*. Stoga se svaki model linearnog programiranja u kojem postoji m strukturnih ograničenja od kojih su sva tipa jednakosti uvijek smatra normalnim ukoliko u njemu postoji m promjenljivih koje su takve da se svaka od njih javlja u jednom i samo jednom strukturnom ograničenju, sa pripadnim koeficijentom jednakim jedinici, bez obzira kakvi su njihovi indeksi.

Normalni model linearnog programiranja se često naziva i *prošireni zadatak linearnog programiranja*. On se najviše koristi za potrebe rješavanja zadatka linearnog programiranja primjenom *simpleks algoritma*. Međutim, kako simpleks algoritam *zahtijeva da svi koeficijenti* b_i, $i = 1..n$ *budu nenegativni*, neki autori smatraju da su gore prikazani modeli normalni *samo ako ispunjavaju dodatni uvjet da su svi koeficijenti* b_i, $i = 1..n$ *nenegativni*. Problemi koji se javljaju u slučaju da među koeficijentima b_i, $i = 1..n$ ima i negativnih (što se najčešće dešava pri vještačkoj transformaciji ograničenja tipa "veće ili jednako" u ograničenja tipa "manje ili jednako" množenjem sa –1) nisu posve jednostavni za tretman i o tome će kasnije biti posebno govora.

Kao što se može primijetiti, u normalnom obliku sva su ograničenja izražena jednačinama, osim uvjeta nenegativnosti promjenljivih. To daje povoda da se razmatra generalniji model u kojem su sva ograničenja

(osim ograničenja na nenegativnost promjenljivih) izražena u formi jednakosti, ali *u kojem sve promjenljive imaju ravnopravan tretman*, odnosno kod kojih se ne vrši diskriminacija promjenljivih na *polazne* (*osnovne*) i *dopunske* (*izravnavajuće*). Takav model naziva se *model linearnog programiranja u kanonskom obliku*, i on se može predstaviti kao

$$\arg\max Z = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

p.o.

$$\begin{aligned} &a_{1,1} x_1 + a_{1,2} x_2 + \ldots + a_{1,n} x_n = b_1 \\ &a_{2,1} x_1 + a_{2,2} x_2 + \ldots + a_{2,n} x_n = b_2 \\ &\ldots \\ &a_{m,1} x_1 + a_{m,2} x_2 + \ldots + a_{m,n} x_n = b_m \end{aligned}$$

$$x_1 \geq 0,\ x_2 \geq 0,\ \ldots\ x_n \geq 0$$

ili, u kompaktnijem obliku koristeći skraćenu notaciju za sumaciju, kao

$$\arg\max Z = \sum_{j=1}^{n} c_j x_j$$

p.o.

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

$$x_j \geq 0,\ \ j = 1\ ..\ n$$

odnosno, u kompaktnom matrično-vektorskom obliku kao

$$\arg\max Z = \mathbf{c}^{\mathrm{T}} \mathbf{x}$$

p.o.

$$\begin{aligned} &\mathbf{A}\,\mathbf{x} = \mathbf{b} \\ &\mathbf{x} \geq 0 \end{aligned}$$

Vidimo da je normalni oblik modela linearnog programiranja je zapravo specijalan slučaj kanonskog oblika. Tačnije, model u kanonskom obliku će biti ujedno i u normalnom obliku ukoliko se iz matrice **A** može izdvojiti m kolona koje će formirati jediničnu submatricu formata $m \times m$, gdje je m broj kolona matrice **A**.

Pristupi rješavanju problema linearnog programiranja

Kao što je već ranije rečeno, u zadacima linearnog programiranja se *svaka specifikacija* vrijednosti promjenljivih $(x_1, x_2, ..., x_n)$ naziva *rješenje*. Pri tome, cilj rješavanja zadataka linearnog programiranja je zapravo naći *optimalno rješenje*, tj. naći takve vrijednosti promjenljivih $(x_1, x_2, ..., x_n)$ da rješenje bude *dopustivo* i da pri tome funkcija cilja ima svoju *najbolju* (najveću ili najmanju vrijednost).

Postoji više metoda za rješavanje zadatka linearnog programiranja (inače, pod metodom se smatra svako smišljeno i plansko postupanje pri radu radi postignuća nekog uspjeha, istine ili saznanja, odnosno određeni put i način ispitivanja mišljenja i rada). Pri tome je zajednička karakteristika svih metoda za rješavanje netrivijalnih zadataka linearnog programiranja *nužnost korištenja računara* u procesu nalaženja najboljeg rješenja, s obzirom da količina neophodnih izračunavanja znatno prevazilazi količinu koja bi bila prihvatljiva za ručni rad.

Metodi koji se najčešće sreću u literaturi su sljedeći:

- ***Grafički metod***;
- ***Metod pretraživanja vrhova dopustivog prostora*** (presjeka ograničenja);
- ***Simpleks metod*** i njegovi bliski srodnici (***revidirani simpleks metod***, ***dualni simpleks metod***, ***metod aktivnog skupa***, i drugi);
- Metodi ***unutarnje tačke***.

Neki od ovih metoda se koriste uglavnom u *edukativne svrhe*, kao što su, na primjer, *grafički metod* i *metod pretraživanja vrhova presjeka dopustivog prostora*, dok su drugi metodi *profesionalno razvijeni postupci* za rješavanje vrlo velikih i složenih zadataka linearnog programiranja.

Najjednostavniji metod rješavanja je *grafički metod*. On je primjenljiv samo na vrlo jednostavne zadatke (najčešće *dvodimenzionalne* ili na one koji se lako *mogu svesti na dvodimenzionalne*) i uglavnom služi za jasnu ilustraciju prirode problema linearnog programinranja i lakšeg razumijevanja osobina njegovog rješenja. Bez obzira na to, ukoliko se radi ručno (tj. bez pomoći računara), dvodimenzionalni problemi se često *mogu brže riješiti grafičkim metodom nego ma kojim drugim metodom*, pogotovo ukoliko se u problemu javlja nešto veći broj ograničenja.

Metod *pretraživanja vrhova dopustivog prostora* (*presjeka* ograničenja) se dosta rijetko susreće u literaturi kao metod za rješavanje zadataka linearnog programiranja. Teoretski, ovaj metod se može primjenjivati za probleme *proizvoljne dimenzionalnosti*, ali složenost izračunavanja izuzetno brzo raste (*eksponencijalno*) sa porastom broja varijabli i ograničenja, tako da je ona praktično primjenljiva samo na probleme sa svega nekoliko promjenljivih i ograničenja. Međutim, i pored toga što je i ovaj metod primjenljiv samo na relativno jednostavne zadatke, on služi kao dobra ilustracija kako se jedan relativno jednostavan pristup mora odbaciti kao praktično neupotrebljiv kada se ustanovi da se trajanje izračunavanja rapidno uvećava *kombinatornom eksplozijom* sa porastom veličine problema. Pored toga, ovaj metod služi kao dobar *uvod u simpleks metod* i lakše razumijevanje *algebre simpleks metoda*.

Simpleks metod (odnosno *simpleks algoritam*) je najpoznatiji metod za rješavanje zadataka linearnog programiranja. On je *vrlo efikasan u praksi* i njegovom primjenom se može *garantirati* da će se pronaći *globalni optimum* u *konačno mnogo koraka* ukoliko se preduzmu *određene predostrožnosti* zbog mogućnosti *cikliranja*. Naime, kako će kasnije biti detaljnije pokazano, u izvjesnim rijetkim okolnostima, simpleks algoritam može upasti u *mrtvu petlju* (tj. *petlju bez izlaza*) i u tom slučaju algoritam se *nikada ne završava*. Srećom, postoje tehnike kojima se *takvi upadi u mrtvu petlju mogu izbjeći*. Mada je teoretski moguće konstruisati primjere za koje simpleks algoritam ima katastrofalno loše (tačnije eksponencijalno) vrijeme izvršavanja, takvi slučajevi se praktično nikada ne susreću u praksi. Zbog toga se kaže da se simpleks algoritam "vrlo rijetko loše ponaša". Postoje i brojne modifikacije simpleks algoritma, koje u pojedinim situacijama imaju nešto bolje perfomanse u odnosu na izvorni simpleks algoritam.

Metodi unutrašnje tačke spadaju u relativno nove pristupe. Nakon početnih *Khachyian*-ovih pokušaja (1979) sa elipsoidnim algoritmom, koji se nije pokazao upotrebljivim u praksi (bar ne za probleme linearnog programiranja), *Karmarkar*-ov metod unutrašnje tačke (1984) bio je prvi praktično upotrebljiv metod ovog tipa. Njegovo vrijeme izvršavanja je čak i u najgorem slučaju *polinomijalno*, dok su eksperimenti na praktičnim problemima pokazali da su njegove perfomanse *sasvim prihvatljive* u poređenju sa simpleks algoritmom, dok je za vrlo velike probleme čak i nešto bolji od njega. U godinama koje slijede, predloženi su i mnogi drugi metodi unutrašnje tačke. Poznat metod iz ove porodice metoda koji vrlo dobro radi u praksi je ***Mehrotra prediktor-korektor metod***, mada se o njemu (sa aspekta performansi) veoma malo zna teoretski. Trenutno mišljenje je da je za potrebe *rutinskih aplikacija* linearnog programiranja, *efikasnost dobrih implementacija metoda zasnovanih na simpleksu i metoda unutrašnje tačke veoma slična*. Treba još naglasiti i činjenicu da bez obzira na to što je složenost računarske implementacije metoda iz porodice simpleks metoda i metoda unutrašnje tačke otprilike podjednaka, svi metodi unutrašnje tačke su *izrazito nepogodni* za ručni rad, tj. rješavanje *bez pomoći računara* (zbog potrebe intenzivnog računanja sa necijelim brojevima), tako da klasični simpleks metod i dalje ostaje najpogodniji metod za ručno rješavanje problema linearnog programiranja.

Grafički metod

Grafički metod za rješavanje zadataka linearnog programiranja se zasniva na osobinama koje smo obradili prilikom razmatranja *geometrijske interpretacije* zadataka linearnog programiranja. Ovaj metod je praktično primjenljiv *samo na zadatke* sa *dvije promjenljive*, ili na zadatke koji se lako mogu *svesti na zadatke sa dvije promjenljive* (recimo, ukoliko imamo zadatak sa tri promjenljive u kojem se javlja jedno ograničenje tipa jednakosti, to ograničenje se može iskoristiti da jednu promjenljivu *izrazimo preko preostale dvije* i na taj način reduciramo zadatak na zadatak sa dvije promjenljive). Teoretski, grafički metod bi se (uz pomoć slika u prostoru) mogao primijeniti i na *opće zadatke sa tri promjenljive* (kao i na zadatke koji se mogu svesti na takve), ali nepreglednost trodimenzionalnih slika ovu teoretsku mogućnost čini teško praktično primjenljivom.

Rezimirajmo ukratko u čemu je bit grafičkog metoda. Neka je zadan *dvodimenzionalni* zadatak linearnog programiranja kod kojeg treba maksimizirati ili minimizirati funkciju cilja oblika $Z = c_1 x_1 + c_2 x_2$ pod ograničenjima koja su zadana skupom nejednakosti. Svako od ograničenja oblika $a_{i,1} x_1 + a_{i,2} x_2 \le b_i$ odnosno $a_{i,1} x_1 + a_{i,2} x_2 \ge b_i$ predstavlja *poluravan* čija je granica *pravac* određen jednačinom $a_{i,1} x_1 + a_{i,2} x_2 = b_i$. Taj pravac dijeli ravan (x_1, x_2) na *dopustivu* i *nedopustivu* oblast (odnosno *dopustivu* i *nedopustivu poluravan*) u odnosu na odgovarajuće ograničenje. Pri tome, taj pravac prolazi kroz tačke $(0, b_i/a_{i,2})$ i $(b_i/a_{i,1}, 0)$, osim u specijalnim slučajevima kada je $a_{i,1} = 0$ ili $a_{i,2} = 0$. Ukoliko je $a_{i,1} = 0$, pravac prolazi kroz tačku $(0, b_i / a_{i,2})$ i okomit je na x_2-osu. Ukoliko je $a_{i,2} = 0$, pravac prolazi kroz tačke $(b_i / a_{i,1}, 0)$ i okomit je na x_1-osu.

Da bismo odredili sa koje strane pravca $a_{i,1} x_1 + a_{i,2} x_2 = b_i$ se nalazi dopustiva poluravan, možemo testirati *proizvoljnu tačku*, recimo tačku $(0, 0)$. Kako je u ovoj tački $a_{i,1} x_1 + a_{i,2} x_2 = 0$, to ukoliko je b_i pozitivno, ova tačka *zadovoljava* nejednakost $a_{i,1} x_1 + a_{i,2} x_2 \le b_i$, pa je u slučaju takve nejednakosti dopustiva oblast *sa one strane pravca* $a_{i,1} x_1 + a_{i,2} x_2 = b_i$ *na kojoj se nalazi koordinatni početak*. S druge strane, ista tačka *ne zadovoljava* nejednakost $a_{i,1} x_1 + a_{i,2} x_2 \ge b_i$, pa je u slučaju takve nejednakosti dopustiva oblast *sa one strane pravca* $a_{i,1} x_1 + a_{i,2} x_2 = b_i$ *na kojoj se ne nalazi koordinatni početak*. Ukoliko je slučajno b_i negativno, vrijede *suprotni zaključci* u odnosu na maloprije izložene. U rijetkoj situaciji u kojoj je $b_i = 0$, pravac $a_{i,1} x_1 + a_{i,2} x_2 = b_i$ *prolazi tačno kroz koordinatni početak*, tako da njega ne možemo koristiti za testiranje, nego moramo upotrijebiti neku drugu tačku. Konačno, *dopustivu oblast* nalazimo kao *presjek svih dopustivih poluravni*.

Ukoliko dopustive poluravni *nemaju zajedničkih tačaka*, to znači da ograničenja nisu međusobno saglasna, odnosno razmatrani problem *nema rješenje*. To se dešava ukoliko su u problemu postavljeni međusobno protivrječni zahtjevi, što nije uvijek očigldno na prvi pogled.

Što se tiče funkcije cilja, ako funkciji cilja damo neku konstantnu vrijednost $Z = C$, dobijamo jednačinu pravca (nivo linije) $c_1 x_1 + c_2 x_2 = C$, koji prolazi kroz tačke $(0, C / c_2)$ i $(C / c_1, 0)$. U principu, dovoljno je nacrtati ovaj pravac za *jednu* nasumično odabranu vrijednost C, poželjno takvu da ovaj pravac (pravac funkcije cilja *presjeca* dopustivu oblast, mada nije obavezno). Pravci koji se dobijaju za sve druge vrijednosti C *paralelni su pravcu dobijenom za jednu fiksnu vrijednost* C. *Smjer najbržeg rasta* funkcije cilja određen je vektorom grad $Z = (c_1, c_2)^T$ i on je okomit na taj pravac. Na osnovu smjera ovog vektora možemo znati *na koju stranu raste funkcija cilja* bez potrebe da crtamo više pravaca za različite vrijednosti C. Sad, da bismo odredili optimalno rješenje, trebamo utvrditi *dokle možemo vršiti paralelno pomjeranje tog pravca* u smjeru porasta funkcije cilja (ukoliko tražimo maksimum), odnosno u suprotnom smjeru od smjera porasta funkcije cilja (ukoliko tražimo maksimum) a da taj pravac još uvijek ima dodirnih tačaka sa dopustivom oblašću. Očigledno, u oba slučaja, to će biti onda kada nivo linija tačno *dodiruje* (*tangira*) dopustivu oblast. Tačka dodira tada upravo predstavlja traženo optimalno rješenje. U slučaju da se nivo linija može neograničeno pomjerati u smjeru porasta funkcije cilja (odnosno suprotno od smjera porasta funkcije cilja ako tražimo minimum) a da pri tome nivo linija uvijek ima presjek sa dopustivom oblašću, tada se optimalno rješenje *nalazi u beskonačnosti*. Jasno, je da se takva situacija ne može desiti ukoliko je dopustiva oblast *ograničena*.

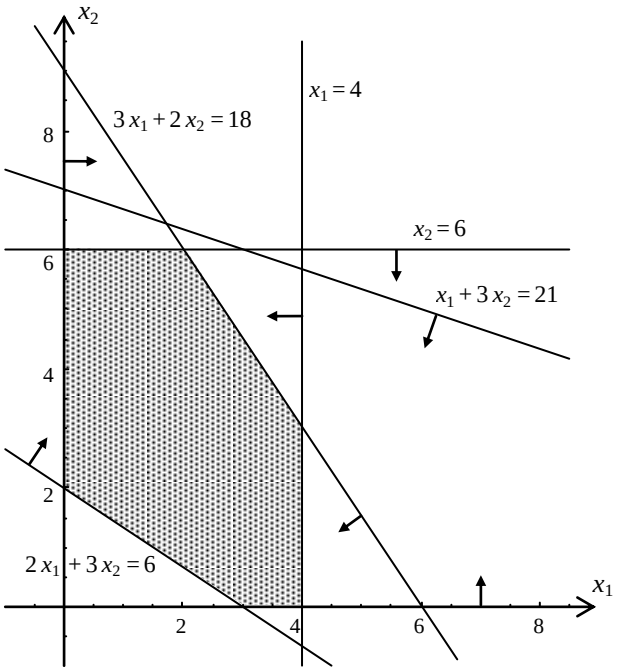
- ***Primjer*** : Grafičkim metodom riješiti problem linearnog programiranja

$$\arg\max Z = 3x_1 + 5x_2$$

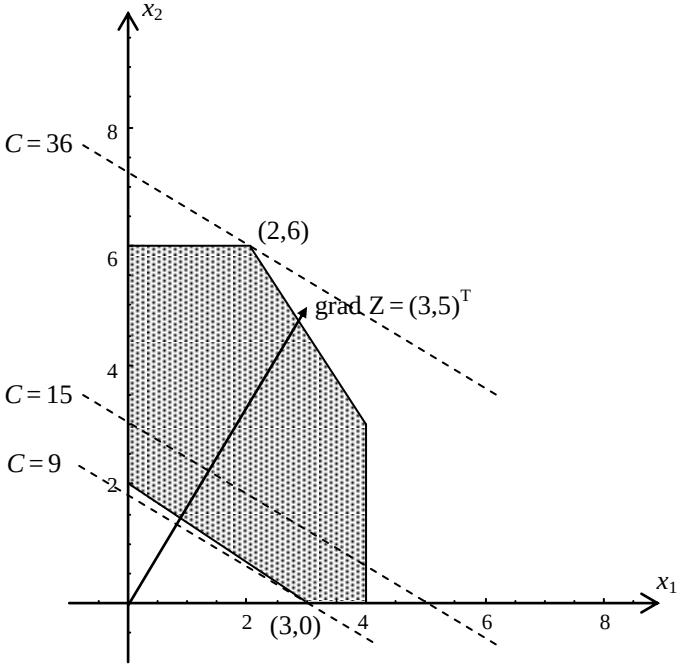
p. o.

$$x_1 \le 4$$
$$2x_2 \le 12$$
$$3x_1 + 2x_2 \le 18$$
$$x_1 + 3x_2 \le 21$$
$$2x_1 + 3x_2 \ge 6$$
$$x_1 \ge 0,\ x_2 \ge 0$$

Prvo treba odrediti dopustivu oblast, tj. odrediti sve vrijednosti (x_1, x_2) koje zadovoljavaju postavljena ograničenja. To će se dobiti tako što se nacrtaju pravci, koji predstavljaju granice dopustivih poluravni, pri čemu ćemo strelicama označiti poluravni koje su dopustive. To je urađeno na sljedećoj slici, na kojoj je dopustivo područje, kao presjek svih dopustivih poluravni, prikazano osjenčeno. Kako je dopustivi prostor *neprazan* i *ograničen*, to je siguran znak da ovaj zadatak linearnog programiranja ima *konačno optimalno rješenje*.



Sada je potrebno *odrediti tačku u dopustivom prostoru* koja *maksimizira vrijednost funkcije cilja*. Posmatraćemo familiju jednačina oblika $3x_1 + 5x_2 = C$ gdje je C ma kakva realna konstanta. Ova familija određuje *pravce funkcije cilja* duž kojih funkcija cilja ima konstantne vrijednosti $Z = C$. Svi ovi pravci su međusobno paralelni, imaju koeficijent smjera $k = -3/5$ i prolaze kroz tačke $(0, C/5)$ i $(C/3, 0)$. Zavisno od vrijednosti C, tako dobijene prave mogu, a i ne moraju imati dopustivih tačaka, tj. mogu a i ne moaju sjeći dopustivi prostor. Uzmimo sada za C neku proizvoljnu vrijednost, ali takvu da vrijednosti $C/5$ i $C/3$ budu "lijepi" brojevi, tako da se prava $3x_1 + 5x_2 = C$ može lijepo nacrtati. Neka je, recimo $C = 15$. Time dobijamo pravu koja prolazi kroz tačke $(0, 3)$ i $(5, 0)$. Ako nacrtamo ovu pravu, vidjećemo da ona *siječe* dopustivi prostor, kao što je prikazano na sljedećoj slici.



Jasno je da ova prava dijeli dopustivi prostor na dva dijela, na dio u kojem je za sve dopustive tačke vrijednost funkcije cilja *manja od* $Z = C = 15$ i na dio u kojem je za sve dopustive tačke vrijednost funkcije cilja *veća od* $Z = C = 15$. Između ostalog, iz toga slijedi da vrijednost funkcije cilja definitivno može biti veća od 15, a da rješenje bude dozvoljeno. Pitanje je samo *koliko može biti veća*. Smjer najbržeg rasta

funkcije cilja određen je vektorom grad $Z = (3, 5)^T$. Ovaj vektor je također prikazan na slici i vidljivo je da je on *okomit* na pravce funkcije cilja, kao što svakako znamo da mora biti. Da bismo dobili *najveću vrijednost funkcije cilja*, moramo utvrditi *dokle se nacrtana prava može paralelno pomjerati u smjeru najbržeg rasta funkcije cilja*, a da ona i dalje ima *makar jednu zajedničku tačku sa dopustivom oblasti*. Sa slike je vidljivo da će se to desiti kada prava funkcije cilja dotakne *vrh dopustive oblasti* koji leži na presjeku pravaca $x_2 = 6$ i $3x_1 + 2x_2 = 18$, a to je vrh sa koordinatama $(2, 6)$. Slijedi da je optimalno rješenje problema $x_1 = 2$ i $x_2 = 6$, pri čemu je optimalna vrijednost funkcije cilja $Z = 3x_1 + 5x_2 = 36$. Inače, prava funkcije cilja koja odgovara najvećoj mogućoj vrijednosti $C = C_{max}$ takvoj da prava funkcije cilja ima zajedničkih tačaka sa dopustivom oblašću naziva se ***gornja potporna prava*** (analogno se definira i ***donja potporna prava***). Slijedi da se problem maksimizacije (minimizacije) zapravo svodi na problem nalaženja gornje (donje) potporne prave.

Na ovom mjestu je bitno ukazati na *važnost preciznog crtanja*. Mada na prvi pogled izgleda da su pravci funkcije cilja u ovom primjeru *paralelni* jednom od pravaca koji čini granicu jednog od ograničenja (tačnije, pravcu $2x_1 + 3x_2 = 6$), oni *ipak to nisu*. Zaista, pravci funkcije cilja imaju koeficijent smjera $k = -3/5 = -0.6$, dok pravac $2x_1 + 3x_2 = 6$ ima koeficijent smjera $k = -2/3 \approx -0.67$, iz čega slijedi da pravci funkcije cilja imaju *neznatno blaži nagib* od nagiba pravca $2x_1 + 3x_2 = 6$, što se može uočiti na slici (s obzirom da je ona precizno nacrtana). Stoga, da je problem bio da se umjesto maksimuma nađe minimum, lako bismo pronašli da se optimalno rješenje nalazi u tački $(3, 0)$. Međutim, ako bismo slučajno *pogrešno nacrtali* pravac funkcije cilja tako da ima *neznatno veći nagib* od nagiba pravca $2x_1 + 3x_2 = 6$, tada bismo pogrešno zaključili da se optimalno rješenje nalazi u *sasvim drugoj tački*, tačnije u tački $(0, 2)$. Iz ovoga vidimo koliko je važno da imamo precizan crtež. Konačno, ukoliko bismo pogrešno nacrtali da je nagib pravaca funkcije cilja *isti* kao nagib pravca $2x_1 + 3x_2 = 6$, zaključili bismo (ponovo pogrešno) da je minimum svaka tačka duži koja spaja tačke $(3, 0)$ i $(0, 2)$.

Recimo još samo nekoliko riječi o mogućnosti grafičkog rješavanja *trodimenzionalnih* problema linearnog programiranja. Za takve tipove problema, ograničenja tipa nejednakosti umjesto dopustivih poluravni generiraju *dopustive* (trodimenzionalne) *poluprostore*, čiji će presjek, ukoliko je ograničen umjesto *konveksnog poligona*, kako je to slučaj kod dvodimenzionalnih problema, tvoriti neki *konveksan poliedar*. Dakle, dopustiva oblast kod takvih problema ima strukturu konveksnog poliedra. Mada je principijelno moguće nacrtati sliku koja grafički ilustrira dozvoljenu oblast za neki trodimenzionalni problem linearnog programiranja, ta slika će *veoma rijetko biti dovoljno pregledna da bi bila upotrebljiva*, osim možda za posve banalne probleme. Što se tiče funkcije cilja, kako je ona funkcija tri promjenljive, $Z = c_1x_1 + c_2x_2 + c_3x_3$, nju je u trodimezionalnom prostoru moguće predstaviti *jedino preko familija nivo površi* $Z = C$, odnosno $c_1x_1 + c_2x_2 + c_3x_3 = C$, gdje su C neke konstante. Kako su ovo jednačine *ravni*, možemo govoriti o ***nivo ravnima***, odnosno ***ravnima funkcije cilja***. Slično kao i u dvodimenzionalnom slučaju, sve su te nivo ravni *međusobno paralelne*. Smjer najbržeg rasta funkcije cilja određen je *gradijentom* grad $Z = (c_1, c_2, c_3)^T$ i on je *okomit* na ravni funkcije cilja. Sada bismo slično kao u dvodimenzionalnom slučaju principijelno mogli utvrditi *dokle možemo vršiti paralelno pomjeranje ravni funkcije cilja* u smjeru porasta funkcije cilja (ukoliko tražimo maksimum), odnosno u suprotnom smjeru od smjera porasta funkcije cilja (ukoliko tražimo maksimum) a da ta ravan još uvijek ima dodirnih tačaka sa poliedrom koji predstavlja dopustivu oblast. Jasno je da će to biti onda kada nivo ravan tačno *dodiruje* (*tangira*) dopustivu oblast, odnosno optimalno rješenje će se i u trodimenzionalnom slučaju obavezno nalaziti *u nekom od vrhova dopustive oblasti*. Ipak, gore opisani postupak je, zbog nepreglednosti odgovarajućih crteža, uglavnom nemoguće provesti u praksi.

Metod pretraživanja vrhova dopustivog prostora

Metod pretraživanja vrhova dopustivog prostora je *vrlo loš metod* za rješavanje problema linearnog programiranja, s obzirom da količina izračunavanja rapidno (eksponencijalno) raste sa porastom broja promjenljivih i ograničenja. Stoga je ovaj metod, čak i uz pomoć računara, praktično izvodljiv samo za posve male probleme. Ipak, ovaj metod je interesantan zbog činjenice da je, za razliku od grafičkog metoda, on *barem teoretski izvodljiv za probleme proizvoljne dimenzionalnosti*. Pored toga, on služi kao dobar uvod za razumijevanje *simpleks metoda*.

Radi razumijevanja metoda pretraživanja vrhova dopustivog prostora, rezimirajmo neka od *bitnih svojstava problema linearnog programiranja*. Ova svojstva su na osnovu grafičke interpretacije očigledna za slučaj dvodimenzionalnih i trodimenzionalnih problema, a uz pomoć nekih elementarnih činjenica iz linearne algebre, moguće je dokazati da iste osobine vrijede i za probleme *proizvoljne dimenzionalnosti*:

- Ako je dopustiva oblast nekog problema linearnog programiranja *neprazan skup*, odnosno ako ograničenja *nisu nesaglasna*, tada je ta dopustiva oblast *n-dimenzionalni konveksni poliedar* (*n*-dimenzionalni poliedar se naziva i ***politop***), pri čemu je *n* dimenzionalnost problema. Pri tome, dopustiva oblast posjeduje *barem jedan vrh*, dok je ukupan broj vrhova dozvoljene oblasti *konačan*.
- Ako problem linearnog programiranja ima optimalna rješenja, ona su uvijek *na granici dopustive oblasti*. Pri tome, ako je optimalno rješenje *jedinstveno*, tada se ono obavezno nalazi u *nekom od vrhova* dopustive oblasti. S druge strane, ako problem ima *više optimalnih rješenja*, tada su ona *sve tačke neke od stranica dopustive oblasti* koja sadrži *barem jedan vrh dozvoljene oblasti*, tako da se i u tom slučaju *barem jedno od optimalnih rješenja* nalazi *u nekom od vrhova* dopustive oblasti.
- Ako problem linearnog programiranja *ima optimalno rješenje*, a za neki vrh $\mathbf{x}^*$ dozvoljene oblasti se zna da je u njemu vrijednost funkcije cilja *bolja nego u njemu susjednim vrhovima*, tada $\mathbf{x}^*$ predstavlja *jedno optimalno rješenje* zadatka linearnog programiranja.
- Svaka *jedinstvena presječna tačka granica n različitih ograničenja* problema (koje, u geometrijskom smislu, predstavljaju *n-dimenzionalne hiperravni*), ukoliko takva postoji, *definira jednoznačno jedan vrh dopustive oblasti* (*n* je, kao i obično, dimenzionalnost problema). Pri tome, dopustiva oblast *nema drugih vrhova* osim onih koji se mogu dobiti presjecanjem granica *n* različitih ograničenja.
- Ako je dopustiva oblast *neprazna* i *ograničena*, tada problem linearnog programiranja *uvijek ima optimalno rješenje*. Ako je dopustiva oblast *neprazna i neograničena*, tada problem linearnog programiranja u kome se traži *maksimum* (ili *minimum*) ima (konačno) optimalno rješenje onda i samo onda ukoliko je funkcija cilja *ograničena odozgo* (ili *odozdo*) *na dopustivoj oblasti*, a to će biti onda i samo onda ukoliko je dozvoljena oblast *ograničena u smjeru gradijenta funkcije cilja*, odnosno *smjera najbržeg porasta funkcije cilja* (ili, za slučaj minimuma, *u smjeru suprotnom od gradijenta funkcije cilja*).

Primijetimo da se u gornjim svojstvima mnogo spominje pojam ***vrha*** (ili ***tjemena***) dopustive oblasti. Mada je ovaj pojam intuitivno jasan u dvije i tri dimenzije, intuicija postaje problematična u više dimenzija. Stoga je poželjno imati *preciznu definiciju* šta se podrazumijeva vrhom ili tjemenom nekog skupa. Za tačku $\mathbf{x} \in \Omega$ kažemo da je *vrh* (*tjeme*) od Ω ukoliko za svaki različit par tačaka $\mathbf{x}_1$ i $\mathbf{x}_2$ iz Ω i svako $\lambda \in (0,1)$ vrijedi da je $\mathbf{x}$ različito od $\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2$, odnosno, simbolički zapisano,

$$(\forall \mathbf{x}_1, \mathbf{x}_2 \in \Omega)\,(\forall \lambda \in (0,1))\;\mathbf{x}_1 \neq \mathbf{x}_2 \Rightarrow \mathbf{x} \neq \lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2$$

Šta ova definicija zapravo govori? Kako odranije znamo da skup tačaka oblika $\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2$ za $\lambda \in (0,1)$ predstavlja *skup tačaka duži čiji su krajevi* $\mathbf{x}_1$ *i* $\mathbf{x}_2$, slijedi da je tačka $\mathbf{x}$ vrh (tjeme) od Ω ako i samo ako *ona ne leži niti na jednoj duži čiji krajevi pripadaju skupu* Ω.

Prethodno navedene osobine podrazumijevaju da su prisutna ograničenja na nenegativnost svih promjenljivih. Mada većina navedenih osobina u većini slučajeva vrijedi i za općenitije slučajeve linearnog programiranja, bez ograničenja na nenegativnost promjenljivih mogu se pojaviti i neki *patološki slučajevi* koji se ne mogu desiti kada postoje ograničenja na nenegativnost. Najjednostavniji takav primjer je problem

$$\arg\max Z = x_1 + x_2$$

p.o.

$$x_1 + x_2 \leq 1$$

Dakle, $x_1 + x_2 \leq 1$ je jedino ograničenje (nema ograničenja na nenegativnost x_1 i x_2). Očigledno je optimalna vrijednost funkcije cilja $Z = 1$, a ona se postiže za ma koje x_1 i x_2 koji zadovoljavaju jednačinu $x_1 + x_2 = 1$, odnosno *na svim tačkama pravca* $x_1 + x_2 \leq 1$. S druge strane, dopustiva oblast je ovdje prosto *poluravan* $x_1 + x_2 \leq 1$. Takva oblast *nema niti jedan vrh*, tako da u ovom primjeru *ne možemo reći da se barem jedno optimalno rješenje nalazi u nekom od vrhova dopustive oblasti*. Ipak, ukoliko bismo pomoću smjena $x_1 = x_1' - x_1''$ i $x_2 = x_2' - x_2''$ uz $x_1' \geq 0$, $x_1'' \geq 0$, $x_2' \geq 0$ i $x_2' \geq 0$ problem sveli na standardni oblik

$$\arg\max Z = x_1' - x_1'' + x_2' - x_2''$$

p.o.

$$x_1' - x_1'' + x_2' - x_2'' \leq 1$$

$$x_1' \geq 0,\; x_1'' \geq 0,\; x_2' \geq 0,\; x_2' \geq 0$$

dobili bismo problem čija dopustiva oblast *ima tri vrha* i u *dva od njih* ($x_1' = 1$, $x_1'' = x_2' = x_2'' = 0$ i $x_2' = 1$, $x_1' = x_1'' = x_2'' = 0$) se zaista nalazi *optimalno rješenje* (koje odgovara optimalnim rješenjima $x_1 = 0$, $x_2 = 1$ odnosno $x_1 = 1$, $x_2 = 0$ polaznog problema).

Metod pretraživanja vrhova dopustivog prostora je u osnovi *vrlo jednostavan*. On se zasniva na tome da se prosto generiraju *svi vrhovi dopustivog prostora* (što je načelno moguće, jer ih ima *konačan broj*), a da se nakon toga izdvoji onaj vrh u kojem funkcija cilja ima *najbolju vrijednost*. Na osnovu prethodnih osobina, slijedi da je taj vrh ujedno i optimalno rješenje. Vrhovi se generiraju tako što se razmatraju *svi mogući presjeci* kombinacija od po *n* granica ograničenja. Oni presjeci koji su dopustivi, predstavljaju vršne tačke, dok se oni koji nisu dopustivi isključuju iz razmatranja.

- ***Primjer*** : Metodom pretraživanja vrhova dopustivog prostora, riješiti problem linearnog programiranja

$$\arg\min Z = 100\,x_1 + 200\,x_2 + 300\,x_3$$

p. o.

$$25\,x_1 + 20\,x_2 + 40\,x_3 \geq 1000$$
$$2\,x_1 + 4\,x_2 + 5\,x_3 \leq 500$$
$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0$$

U ovom primjeru, granice ograničenja su pravci $25\,x_1 + 20\,x_2 + 40\,x_3 = 1000$, $2\,x_1 + 4\,x_2 + 5\,x_3 = 500$, $x_1 = 0$, $x_2 = 0$ i $x_3 = 0$, a potencijalne vršne tačke su presjeci ma koje tri od tih pet granica. Moguće je ukupno napraviti 10 kombinacija tri granice iz skupa od ukupno pet granica. Rezultati analize svih tih kombinacija prikazani su u sljedećoj tabeli.

Kombinacije granica	Presječna tačka	Status	Vrijednost cilja
$25x_1+20x_2+40x_3=1000$ $2x_1+4x_2+5x_3=500$ $x_1=0$	$x_1=0$ $x_2=250$ $x_3=-100$	Nedopustiva	20000
$25x_1+20x_2+40x_3=1000$ $2x_1+4x_2+5x_3=500$ $x_2=0$	$x_1=-1000/3$ $x_2=0$ $x_3=700/3$	Nedopustiva	110000/3
$25x_1+20x_2+40x_3=1000$ $2x_1+4x_2+5x_3=500$ $x_3=0$	$x_1=-100$ $x_2=175$ $x_3=0$	Nedopustiva	25000
$25x_1+20x_2+40x_3=1000$ $x_1=0$ $x_2=0$	$x_1=0$ $x_2=0$ $x_3=25$	Dopustiva	7500
$25x_1+20x_2+40x_3=1000$ $x_1=0$ $x_3=0$	$x_1=0$ $x_2=50$ $x_3=0$	Dopustiva	10000
$25x_1+20x_2+40x_3=1000$ $x_2=0$ $x_3=0$	$x_1=40$ $x_2=0$ $x_3=0$	Dopustiva	4000
$2x_1+4x_2+5x_3=500$ $x_1=0$ $x_2=0$	$x_1=0$ $x_2=0$ $x_3=100$	Dopustiva	30000
$2x_1+4x_2+5x_3=500$ $x_1=0$ $x_3=0$	$x_1=0$ $x_2=125$ $x_3=0$	Dopustiva	25000
$2x_1+4x_2+5x_3=500$ $x_2=0$ $x_3=0$	$x_1=250$ $x_2=0$ $x_3=0$	Dopustiva	25000
$x_1=0$ $x_2=0$ $x_3=0$	$x_1=0$ $x_2=0$ $x_3=0$	Nedopustiva	0

Iz ove tablice vidimo da od ukupno 10 mogućih presječnih tačaka imamo 6 koje pripadaju dopustivom prostoru. Slijedi da naš dopustivi prostor ima ukupno 6 vršnih tačaka. Ukoliko smo sigurni da je dozvoljena oblast *ograničena*, optimalno rješenje se mora nalaziti u nekoj od njih. Međutim, lako je uvidjeti da je dozvoljena oblast zaista ograničena. Zaista. zbog prisustva na nenegativnost promjenljivih, vrijednosti svih promjenljivih su svakako *ograničene odozdo*. Pored toga, zbog prisustva ograničenja $2\,x_1 + 4\,x_2 + 5\,x_3 \leq 500$

i činjenice da niti jedna promjenljiva ne može biti nenegativna, slijedi također da *ni jedna od promjenljivih ne može neograničeno rasti* a da ovo ograničenje ostane zadovoljeno (preciznije, sigurno je da vrijedi $x_1 \le 500/2 = 250$, $x_2 \le 500/4 = 125$ i $x_3 \le 500/5 = 100$). Iz ovoga jasno slijedi ograničenost dopustivog prostora. Stoga se optimalno rješenje nalazi u onoj vršnoj tački za koju funkcija cilja ima *najmanju moguću vrijednost*, a to je tačka $x_1 = 40$, $x_2 = 0$, $x_3 = 0$ za koju se postiže optimalna vrijednost funkcije cilja $Z = 4000$.

U prethodnom primjeru *nije bilo ograničenja tipa jednakosti*. Ako takva ograničenja postoje, ona moraju biti uključena *u svaku kombinaciju koja se testira* (što efektivno smanjuje broj kombinacija). Zaista, kod ograničenja tipa jednakosti ne postoje *dopustivi i nedopustivi poluprostor sa jedne odnosno druge strane granice*, već je tada dopustiva *isključivo granica* (dakle, dopustivi skup za takvo ograničenje se poklapa sa graničnom hiperravni). Zbog toga takva ograničenja moraju biti uključena u svaku od kombinacija.

Mada je opisani metod konceptualno vrlo jednostavan, problem je što broj kombinacija koje treba razmotriti rapidno raste sa porastom problema. Zaista, neka je dat problem linearnog programiranja sa n promjenljivih i m ograničenja tipa nejednakosti, ne računajući ograničenja na nenegativnost promjenljivih. Zajedno sa ograničenjima na nenegativnost promjenljivih, to je *ukupno* $m + n$ *ograničenja*, odnosno toliko n-dimenzionalnih hiperravni koje predstavljaju granice tih ograničenja, tj. koje razdvajaju dopustivi od nedopustivog (n-dimenzionalnog) poluprostora. Kako ima ukupno $(m + n)! / (m! \, n!)$ načina da iz skupa od $m + n$ hiperravni izdvojimo njih n, toliko ima i tačaka koje trebamo testirati. Pri tome, za nalaženje svake od tačaka *treba riješiti sistem od n linearnih jednačina sa n nepoznatih*, što uz dobre metode rješavanja zahtijeva otprilike $2n^3/3$ elementarnih operacija (poput sabiranja i množenja). Slijedi da je ukupan broj elementarnih operacija koje treba izvršiti negdje oko $2n^3 (m + n)! / (3m! \, n!)$. Nažalost, ovo je ogroman broj već za relativno male vrijednosti m i n. Recimo, za $m = n = 30$ ovaj broj iznosi 212876246816750563200. Uz razumnu pretpostavku da klasični današnji računari mogu izvršiti oko *milijardu* elementarnih operacija u jednoj sekundi, za rješavanje problema veličine $m = n = 30$ ovim metodom današnjim računarima bi trebalo više od 67502 godina (superračunari bi ovo vrijeme možda mogli skratiti za jedan do dva reda veličine, ali vrijeme i dalje ostaje neprihvatljivo veliko)!

Iz svega opisanog slijedi da se metod pretraživanja vrhova dopustivog prostora čak i teoretski može primjenjivati *jedino u slučaju kada sa sigurnošću znamo da je dopustivi prostor ograničen*. U suprotnom, optimalno rješenje se umjesto u nekom od vrhova *može nalaziti u beskonačnosti*, a nema načina na koji bi metod pretraživanja vrhova dopustivog prostora to mogao saznati. Stoga se prirodno nameće pitanje *kako se može utvrditi da li je dopustivi prostor ograničen ili ne*. Nažalost, pokazuje se da u općem slučaju problem testiranja da li je dopustivi prostor ograničen ili ne nije nimalo jednostavan. Štaviše, pokazuje se da taj problem u osnovi nije ništa lakši nego problem rješavanja zadataka linearnog programiranja, jer se može pokazati da se problem testiranja da li je dopustivi prostor ograničen ili ne može svesti na problem jednog specijalno formiranog zadatka linearnog programiranja i testiranje da li njegovo optimalno rješenje zadovoljava neke uvjete ili ne. Srećom, u mnogo slučajeva je *moguće* jednostavno utvrditi da je dopustivi prostor ograničen. Na primjer, ukoliko postoje ograničenja na nenegativnost promjenljivih, dopustivi prostor će sigurno biti ograničen ukoliko među strukturnim ograničenjima postoji *barem jedno ograničenje tipa* "*manje ili jednako*" *u kojem su koeficijenti uz sve promjenljive kao i slobodni član pozitivni*. Taj smo kriterij koristili u prethodnom primjeru da pokažemo da je u njemu dopustivi prostor zaista konačan.

Simpleks algoritam

Simpleks algoritam je najpoznatiji i u praksi najviše korišteni postupak za rješavanje zadataka linearnog programiranja (inače, pod algoritmom se podrazumijeva svaka jasna i precizno definirana procedura koja za zadane početne podatke nakon konačno mnogo primijenjenih koraka daje rješenje problema ili odgovor da rješenje ne postoji). Ovaj algoritam, u svojoj osnovnoj formi, pripisuje se *George Dantzig*-u (1947). Slično metodu pretraživanja vrhova dopustivog prostora, simpleks algoritam se također zasniva na činjenici da *ako optimalno rješenje problema linearnog programiranja uopće postoji, ono se mora nalaziti u nekom od vrhova dopustivog prostora*, kojih ima *konačno mnogo*. Međutim, za razliku od metoda pretraživanja vrhova dopustivog prostora, simpleks algoritam omogućava da se optimalno rješenje nađe *bez potrebe* da se ispituju *svi vrhovi dopustivog prostora*.

Simpleks algoritam predstavlja *iterativnu proceduru* koja, *polazeći od jednog dopustivog vrha*, u svakoj iteraciji generira *novi vrh dopustivog prostora* u kojem funkcija cilja ima "*bolju*" vrijednost nego u tekućem vrhu. Postupak se zaustavlja kada se više ne može naći *niti jedan susjedni vrh* u kojem funkcija cilja ima bolju vrijednost nego u tekućem vrhu. Karakteristično je da simpleks uspješno algoritam rješava *svaki problem linearnog programiranja*, a također uspješno detektira situacije u kojima *dopustiva rješenja ne postoje* ili u kojima se optimalno rješenje nalazi *u beskonačnosti*.

Osnovna ideja simpleks algoritma je veoma jednostavna i zasniva se na sljedećoj iterativnoj strukturi:

1. Krenuti od *neke vršne tačke* dozvoljene oblasti.
2. Iz svake vršne tačke izlazi *konačno mnogo* (tačnije, *n*) *bridova* dopustive oblasti koji vode ka susjednim vršnim tačkama. Testirati *svaki od bridova koji izlaze iz tekuće vršne tačke* i ustanoviti da li se *funkcija cilja poboljšava ukoliko se krećemo duž njih*. Ukoliko ne postoji *niti jedan brid takav da se funkcija cilja poboljšava ukoliko se krećemo duž njega, tada je tekuća vršna tačka optmalno rješenje* i algoritam *završava sa radom*.
3. Izabrati *neki od bridova* takav da se funkcija cilja poboljšava ukoliko se krećemo duž njega. Pronaći *vršnu tačku u koju taj brid vodi* (jasno je da će u takvoj vršnoj tački funkcija imati bolju vrijednost nego u prethodnoj).
4. Proglasiti *novu vršnu tačku za tekuću* i vratiti se na korak 2.

Mada je opisana ideja veoma jednostavna, nije nimalo očigledno *kako konkretno realizirati* pojedine korake u prethodno opisanom postupku. Konkretno, simpleks algoritam, slično većini iterativnih postupaka za nalaženje optimalnih rješenja zadataka matematičkog programiranja, mora riješiti tri podzadatka:

- Pronaći *prvu dopustivu vršnu tačku* (tj. *neko dozvoljeno rješenje*, koje je ujedno vrh);
- Sa *tekuće* dopustive vršne tačke preći na *neku drugu* dopustivu vršnu tačku u kojoj funkcija cilja ima *bolju vrijednost*;
- Prepoznati *kada je dostignuta optimalna tačka*.

Pri rješavanju problema linearnog programiranja pomoću simpleks algoritma, polazi se od *standardne forme* problema linearnog programiranja

$$\arg\max Z = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

p.o.

$$\begin{aligned}
&a_{1,1} x_1 + a_{1,2} x_2 + \ldots + a_{1,n} x_n \le b_1 \\
&a_{2,1} x_1 + a_{2,2} x_2 + \ldots + a_{2,n} x_n \le b_2 \\
&\ldots \\
&a_{m,1} x_1 + a_{m,2} x_2 + \ldots + a_{m,n} x_n \le b_m
\end{aligned}$$

$$x_1 \ge 0,\ x_2 \ge 0,\ \ldots\ x_n \ge 0$$

gdje su $a_{i,j}$, b_j i c_i za $i = 1 \,..\, m$, $j = 1 \,..\, n$ zadani realni brojevi. Za početak ćemo pretpostaviti da su veličine b_j, $j = 1 \,..\, n$ *nenegativni brojevi*. Ukoliko to nije slučaj, simpleks algoritam se prilično komplicira, o čemu ćemo detaljno govoriti kasnije.

Trik na kojem se zasniva simpleks algoritam je *transformacija polaznog problema* u drugi ekvivalentni oblik koji je takav da se pojedini koraci koji su neophodni u gore opisanoj općoj proceduri izvode *na mnogo jednostavniji način* (inače, postoji metod poznat pod nazivom ***metod aktivnog skupa***, koji je "blizak rođak" simpleks metoda, u kojem se ta transformacija ne izvodi, nego se svi opisani koraci opće procedure izvode direktno nad polaznim oblikom problema u standardnoj formi). Taj ekvivalentni oblik nije ništa drugo nego tzv. *normalna forma* problema linearnog programiranja o kojoj smo govorili ranije. Ta forma se dobija kada se u ograničenja tipa nejednačina dodaju nove tzv. ***izravnavanjuće*** (ili prosto ***dodatne***) ***promjenljive***. Naime, uz upotrebu izravnavajućih promjenljivih, polazni problem se u normalnoj (proširenoj) formi može izraziti kao

$$\arg\max Z = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

p.o.

$$\begin{aligned}
&a_{1,1} x_1 + a_{1,2} x_2 + \ldots + a_{1,n} x_n + x_{n+1} &&= b_1 \\
&a_{2,1} x_1 + a_{2,2} x_2 + \ldots + a_{2,n} x_n \quad + x_{n+2} &&= b_2 \\
&\ldots \\
&a_{m,1} x_1 + a_{m,2} x_2 + \ldots + a_{m,n} x_n \quad + x_{n+m} &&= b_m
\end{aligned}$$

$$x_1 \ge 0,\ x_2 \ge 0,\ \ldots\ x_n \ge 0,\ x_{n+1} \ge 0,\ x_{n+2} \ge 0,\ \ldots,\ x_{n+m} \ge 0,$$

Postavlja se pitanje *koji je cilj prelaska na normalni oblik*. Na prvi pogled, na taj način smo samo povećali dimenzionalnost problema sa n na $n + m$, što više djeluje kao otežavajuća nego kao olakšavajuća okolnost. Međutim, takvim prelaskom neke geometrijske osobine dopustivog skupa u tako dobijenom

$(n+m)$-dimenzionalnom prostoru dobijaju znatno povoljnije forme nego u slučaju polaznog problema. Na prvom mjestu, dopustiva oblast više nije ma kakav višedimenzionalni poliedar, nego jedna njegova specijalna forma koja se naziva ***simpleks*** (odatle i potiče naziv algoritma). Inače, simpleks je neka vrsta višedimenzionalnog analogona trougla (jednodimenzionalni simpleks je *duž*, dvodimenzionalni simpleks je *klasični trougao*, trodimenzionalni simpleks je *trostrana piramida*, itd.). Geometrijske osobine simpleksa su mnogo povoljnije nego geometrijske osobine proizvoljnih višedimenzionalnih poliedara. Pored toga, vršne tačke dopustive oblasti problema zadanog u normalnoj formi imaju jednu veoma važnu osobinu, a to je da u njima *najviše m* (od ukupno $n+m$) *promjenljivih mogu biti različite od nule*, dok *sve ostale promjenljive moraju biti jednake nuli*. Preciznije, neka dopustiva tačka je vršna tačka *ako i samo ako su u njoj najviše m promjenljivih različite od nule*. Ovu osobinu je veoma lako dokazati ako pratimo ideje kako smo određivali vršne tačke u metodu pretraživanja vrhova dopustive oblasti.

Korisno je istaći da uvedene izravnavajuće promjenljive imaju i svoju *konkretnu interpretaciju*, odnosno nisu čisto vještačka matematska tvorevina. Naime, ukoliko razmotrimo neko konkretno (recimo *i*-to) strukturno ograničenje $a_{i,1}x_1 + a_{i,2}x_2 + ... + a_{i,n}x_n \le b_i$, desna strana b_i u tom ograničenju tipično predstavlja *raspoloživu količinu* nekog (*i*-tog) resursa, dok lijeva strana $a_{i,1}x_1 + a_{i,2}x_2 + ... + a_{i,n}x_n$ predstavlja *utrošenu količinu* istog resursa. Slijedi da izravnavajuća promjenljiva x_{n+i} predstavlja *preostalu* odnosno *neutrošenu količinu* (zalihe) istog resursa. Primijetimo još da promjenljiva x_{n+i} ima *istu jedinicu mjere* kao i odgovarajući parametar b_i.

Razmotrimo sada problem nalaženja *prve dopustive vršne tačke*. U slučaju kada je problem zadan u polaznom standardnom obliku, ovaj problem nije posve jednostavan, s obzirom da ponekad treba ispitati brojne kombinacije granica ograničenja prije nego što se nađe dozvoljeni presjek (koji je onda vršna tačka). Međutim, nakon transformacije polaznog zadatka u normalni oblik, problem nalaženja prve dopustive vršne tačke postaje *vrlo jednostavan*. Naime, primijetimo da se ograničenja koja se javljaju u normalnoj formi problema (ne računajući ograničenja na nenegativnost promjenljivih) mogu posmatrati kao *sistem od m jednačina sa* $n+m$ promjenljivih. Ukoliko matrica ovog sistema *ima puni rang m* (a ima, jer se u njoj nalazi jedinična submatrica formata $m \times m$), tada se iz ovog sistema *m* promjenljivih može izraziti preko preostalih $(n+m)-m = n$ promjenljivih, pri čemu se tih *n* promjenljivih mogu izabrati *proizvoljno*. Onih *m* promjenljivih za koje se odlučimo da ih *ne biramo proizvoljno* nego ih izražavamo preko ostalih *n* promjenljivih, nazivaju se ***bazne promjenljive***, a skup svih baznih promjenljivih nazivamo ***bazom***. Ako sad pažljivo razmotrimo sistem jednačina o kojem je riječ, tj. sistem

$$\begin{aligned}
&a_{1,1}x_1 + a_{1,2}x_2 + ... + a_{1,n}x_n + x_{n+1} &&= b_1\\
&a_{2,1}x_1 + a_{2,2}x_2 + ... + a_{2,n}x_n \quad + x_{n+2} &&= b_2\\
&...\\
&a_{m,1}x_1 + a_{m,2}x_2 + ... + a_{m,n}x_n \quad\quad + x_{n+m} &&= b_m
\end{aligned}$$

vidimo da jedno vrlo jednostavno rješenje možemo dobiti ukoliko uzmemo da su *prvih n promjenljivih jednake nuli*, tj. da je $x_j = 0$ za $j = 1..n$. Tada iz prethodnog sistema odmah slijedi da je $x_{n+i} = b_i$ za $i = 1..m$. Primijetimo da je, zbog činjenice da su svi koeficijenti b_i, $i = 1..m$ nenegativni, ovo rješenje *dopustivo* (sve promjenljive su nenegativne). Ono se naziva ***bazno dopustivo rješenje***. Na osnovu onoga što je ranije rečeno, ovo rješenje ujedno predstavlja i *vrh dopustivog prostora*, pošto je dopustivo i u njemu se nalazi najviše *m* nenultih promjenljivih. Na taj način smo našli jedno dopustivo rješenje. Dakle, *početno bazno dopustivo rješenje* problema linearnog programiranja zadanog u normalnom obliku dato je kao $\mathbf{x} = (0, 0, ..., 0, b_1, b_2, ..., b_m)^T$. Početna baza je $B = (x_{n+1}, x_{n+2}, ..., x_{n+m})$, dok je početna funkcije cilja $Z = 0$ (s obzirom da su sve promjenljive koje figuriraju u funkciji cilja jednake nuli).

Sada nije teško shvatiti razlog zbog kojeg je simpleks algoritmu problematično ukoliko je neki od koeficijenata b_i, $i = 1..m$ negativan. Zaista, u tom slučaju rješenje formirano na gore opisani način *ne bi bilo dopustivo*, s obzirom da tada izravnavajuća promjenljiva x_{n+i} koja odgovara negativnom koeficijentu b_i *ne bi ispunjavala uvjet nenegativnosti* $x_{n+i} \ge 0$.

Uočimo da u gornjem sistemu jednačina, promjenljive koje imaju ulogu baznih promjenljivih imaju sljedeća svojstva:

- u *ma kojem ograničenju*, može se nalaziti *samo jedna bazna promjenljiva sa nenultim koeficijentom*;
- koeficijent uz na koju baznu promjenljivu *u onom ograničenju u kojem se ona javlja* (sa nenultim koeficijentom) *uvijek je jednak jedinici*.

Isto tako, primijetimo da su u funkciji cilja nenulti koeficijenti *samo uz nebazne promjenljive*. Ključna ideja simpleks algoritma je da se na neki način obezbijedi da ova dva svojstva ostanu očuvana pri prelasku sa jedne na drugu vršnu tačku.

Razmotrimo sada problem nalaženja *nove vršne* (*bazne dopustive*) *tačke* u kojoj funkcija cilja ima "bolju" vrijednost. U simpleks algoritmu, sa bilo koje tekuće vršne tačke uvijek se vrši prelaz isključivo na *neku od njoj susjednih tačaka* (dvije vršne tačke su susjedne ukoliko su spojene *bridom dopustive oblasti*). Nije teško pokazati da su dvije bazne dopustive tačke susjedne ako i samo ako im se baze razlikuju za *samo jednu promjenljivu*. Potrebno je, dakle, *promijeniti početnu bazu* tako što će se *neka od nebaznih promjenljivih prevesti u baznu*, dok će pri tome *neka od baznih promjenljivih postati nebazna*. Pri tome, ono što time želimo postići je *da se vrijednost funkcije cilja poboljša* (odnosno *poveća*, pošto smo pretpostavili da nam je cilj *maksimizacija*). U nastavku ćemo razmotriti kako ovo izvesti.

Pretpostavimo da neku nebaznu promjenljivu, recimo x_q, želimo uvesti u bazu. Razmotrimo stoga šta će se desiti ukoliko promjenljiva x_q promijeni svoju vrijednost sa $x_q = 0$ na $x_q = t$ gdje je t neki pozitivan broj. Očigledno će se zbog te promjene funkcija cilja *promijeniti* za iznos $c_q x_q$, pri čemu je ta promjena *uvećanje* ukoliko je $c_q > 0$, a *umanjenje* ukoliko je $c_q < 0$ (ukoliko je $c_q = 0$, efektivno nema nikakve promjene). Naravno, da bi ograničenja ostala zadovoljena, sve bazne promjenljive moraju promijeniti svoje vrijednosti sa $x_{n+i} = b_i$ na $x_{n+i} = b_i - a_{i,\,q}\,t$ za sve $i = 1\,..\,m$. S obzirom da su svi koeficijenti b_i nenegativni, za dovoljno malo t moguće je postići da uvjeti nenegativnosti promjenljivih i dalje budu zadovoljeni (zapravo, za ovo je neophodno da svi koeficijenti b_i budu *pozitivni* a ne samo *nenegativni*; situaciju kada je neki od koeficijenata b_i jednak nuli ćemo kasnije posebno istražiti). Međutim, kako se u funkciji cilja *uopće ne pojavljuju bazne promjenljive*, to njihova promjena neće utjecati na vrijednost funkcije cilja, odnosno jedina promjena vrijednosti funkcije cilja potiče samo od promjene x_q (i ona iznosi $c_q x_q$).

Iz provedenog razmatranja slijedi da ima smisla u bazu uvoditi *samo one nebazne promjenljive uz koje u funkciji cilja stoji pozitivni koeficijent* $c_q > 0$ (traženje ovih pozitivnih koeficijenata geometrijski predstavlja traženje bridova duž kojih funkcija cilja raste). Ukoliko takvih nebaznih promjenljivih *nema*, funkciju cilja *nije moguće poboljšati*, i tekuće bazno rješenje je zapravo *optimalno* (u tom trenutku se algoritam *završava*). Ukoliko postoji *više nebaznih promjenljivih* uz koje u funkciji cilja stoji pozitivni koeficijent $c_q > 0$, tada se uvođenjem *bilo koje od njih* u bazu može postići poboljšanje funkcije cilja. Mada svaki mogući takav izbor prije ili kasnije vodi ka optimalnom rješenju, *nisu svi izbori jednako efikasni*. Pravila kojim se određuje *koju promjenljivu uvesti u bazu* nazivaju se ***pravila pivotiranja*** i diskusije o tome koje je najbolje pravilo pivotiranja *traju sve do danas* (*bez definitivnog odgovora*). Najčešće se koristi najjednostavnije tzv. ***Dantzig-ovo pravilo*** po kojem se prosto uzima da u bazu ulazi ona promjenljiva uz koju u funkciji cilja stoji *najveći pozitivni koeficijent*, mada se definitivno zna da ovo nije i najbolje pravilo (kasnije ćemo vidjeti i zašto). Dakle, po *Dantzig-ovom* pravilu u bazu uvodimo promjenljivu x_q za koju vrijedi

$$c_q = \max\,\{c_j \mid c_j > 0,\ j = 1\,..\,n\}$$

Sada je potrebno odrediti *promjenljivu koja napušta bazu*. Da bismo to obavili, pretpostavimo da smo promjenljivu $x_q = 0$ koja ulazi u bazu povećali na vrijednost $x_q = t$. Poželjno bi bilo da t bude što veće, da bismo što više povećali funkciju cilja. Međutim, kako pri tome stare bazne promjenljive moraju promijeniti svoju vrijednosti sa $x_{n+i} = b_i$ na $x_{n+i} = b_i - a_{i,\,q}\,t$ za sve $i = 1\,..\,m$, one će i dalje ostati nenegativne samo ukoliko t zadovoljava sistem nejednačina

$$b_i - a_{i,\,q}\,t \geq 0,\ i = 1\,..\,m$$

Ovdje mogu nastati dva slučaja. Ako je $a_{i,\,q} \leq 0$, tada t može imati ma kakvu pozitivnu vrijednost da nejednačina $b_i - a_{i,\,q}\,t \geq 0$ bude zadovoljena. Stoga, ukoliko vrijedi $a_{i,\,q} \leq 0$ *za sve vrijednosti* i od 1 do m, tada t (a samim tim i funkcija cilja) *mogu rasti bez ikakve granice*, tako da optimalno rješenje *ne postoji* (odnosno, *nalazi se u beskonačnosti*). Međutim, ukoliko je $a_{i,\,q} > 0$ makar za jednu vrijednost i, to nameće da mora biti $t \leq b_i/a_{i,\,q}$. Kako ovo mora biti zadovoljeno za *sve takve vrijednosti* i, slijedi da t smije biti najviše jednak *najmanjoj od vrijednosti* $b_i/a_{i,\,q}$ *za sve vrijednosti* i *za koje je* $a_{i,\,q} > 0$ (traženje najveće vrijednosti za t geometrijski odgovara traženju *koliko daleko* trebamo ići duž odabranog brida da pređemo u novu vršnu tačku). Odnosno, najveća dozvoljena vrijednost za t iznosi

$$t_{max} = \min\,\{b_i/a_{i,\,q} \mid a_{i,\,q} > 0,\ i = 1\,..\,m\}$$

Ukoliko je p vrijednost indeksa za koju se postiže ovaj minimum, tj. ukoliko je $b_p/a_{p,q} = t_{max}$, tada će pri povećanju vrijednosti promjenljive x_q sa 0 na t_{max} nova vrijednost promjenljive x_{n+p} biti

$$x_{n+p} = b_p - a_{p,\,q}\,t_{max} = b_p - a_{p,\,q}\,(b_p / a_{p,q}) = 0$$

Dakle, iz baze će ispasti *upravo promjenljiva* x_{n+p}.

Izborom promjenljive koja *ulazi u bazu* i promjenljive koja *izlazi iz baze* izabrali smo tzv. ***vodeću kolonu*** q i ***vodeću vrstu*** p. Na presjeku vodeće vrste i vodeće kolone nalazi se ***vodeći element*** (ili ***pivot***) $a_{p,q}$. Sada je potrebno *transformirati problem* u oblik koji će po strukturi biti *identičan polaznom*, samo *sa drugačijim izborom baznih i nebaznih promjenljivih*. Drugim riječima, trebamo postići da nova bazna promjenljiva x_q figurira *u samo jednoj jednačini* (sa nenultim koeficijentom) i da pri tome u toj jednačini koeficijent uz nju bude jednak jedinici. Konkretno, *vodeći element* (tj. element na presjeku p-te vrste i q-te kolone) treba postati jednak jedinici, a svi ostali koeficijenti u q-toj koloni trebaju postati jednaki nuli (kao posljedica te transformacije, u $(n+p)$-toj koloni, u kojoj je bila samo jedna jedinica a sve ostale nule, mogu se pojaviti i drugačiji koeficijenti). Pored toga, i funkciju cilja treba transformirati tako da u njoj nenulti koeficijenti budu samo uz (nove) nebazne promjenljive. Simbolički, to možemo iskazati ovako. Neka je polazni problem (u normalnom obliku) imao sljedeći oblik (u kojem su eksplicitno prikazani i koeficijenti koji su jednaki nuli odnosno jedinici):

$$\arg\max Z = c_1 x_1 + c_2 x_2 + \ldots + c_q x_q + \ldots + c_n x_n + 0\,x_{n+1} + 0\,x_{n+2} + \ldots + 0\,x_{n+p} + \ldots + 0\,x_{n+m}$$

p.o.

$$\begin{aligned}
&a_{1,1} x_1 + a_{1,2} x_2 + \ldots + a_{1,q} x_q + \ldots + a_{1,n} x_n + 1\,x_{n+1} + 0\,x_{n+2} + \ldots + 0\,x_{n+p} + \ldots + 0\,x_{n+m} = b_1 \\
&a_{2,1} x_1 + a_{2,2} x_2 + \ldots + a_{2,q} x_q + \ldots + a_{2,n} x_n + 0\,x_{n+1} + 1\,x_{n+2} + \ldots + 0\,x_{n+p} + \ldots + 0\,x_{n+m} = b_2 \\
&\ldots \\
&a_{p,1} x_1 + a_{p,2} x_2 + \ldots + a_{p,q} x_q + \ldots + a_{p,n} x_n + 0\,x_{n+1} + 0\,x_{n+2} + \ldots + 1\,x_{n+p} + \ldots + 0\,x_{n+m} = b_p \\
&\ldots \\
&a_{m,1} x_1 + a_{m,2} x_2 + \ldots + a_{m,q} x_q + \ldots + a_{m,n} x_n + 0\,x_{n+1} + 0\,x_{n+2} + \ldots + 0\,x_{n+p} + \ldots + 1\,x_{n+m} = b_m
\end{aligned}$$

$$x_1 \ge 0,\ x_2 \ge 0,\ \ldots\ x_{n+m} \ge 0,$$

Potrebno je ovaj problem transformirati da dobije sljedeći oblik (pri tome će se kao neophodan u funkciji cilja pojaviti i *slobodni član*, koji smo označili sa $-F'$):

$$\arg\max Z = c_1' x_1 + c_2' x_2 + \ldots + 0\,x_q + \ldots + c_n' x_n + 0\,x_{n+1} + 0\,x_{n+2} + \ldots + c_{n+p}' x_{n+p} + \ldots + 0\,x_{n+m} - F'$$

p.o.

$$\begin{aligned}
&a_{1,1}' x_1 + a_{1,2}' x_2 + \ldots + 0\,x_q + \ldots + a_{1,n}' x_n + 1\,x_{n+1} + 0\,x_{n+2} + \ldots + a_{1,\,p}' x_{n+p} + \ldots + 0\,x_{n+p} = b_1' \\
&a_{2,1}' x_1 + a_{2,2}' x_2 + \ldots + 0\,x_q + \ldots + a_{2,n}' x_n + 0\,x_{n+1} + 1\,x_{n+2} + \ldots + a_{2,\,p}' x_{n+p} + \ldots + 0\,x_{n+p} = b_2' \\
&\ldots \\
&a_{p,1}' x_1 + a_{p,2}' x_2 + \ldots + 1\,x_q + \ldots + a_{p,n}' x_n + 0\,x_{n+1} + 0\,x_{n+2} + \ldots + a_{p,\,p}' x_{n+p} + \ldots + 0\,x_{n+p} = b_p' \\
&\ldots \\
&a_{m,1}' x_1 + a_{m,2}' x_2 + \ldots + 0\,x_q + \ldots + a_{m,n}' x_n + 0\,x_{n+1} + 0\,x_{n+2} + \ldots + a_{m,\,p}' x_{n+p} + \ldots + 1\,x_{n+m} = b_m'
\end{aligned}$$

$$x_1 \ge 0,\ x_2 \ge 0,\ \ldots\ x_{n+m} \ge 0,$$

Ovu transformaciju nije teško postići. Za tu svrhu, iz vodeće (tj. p-te) vrste izrazimo novu baznu promjenljivu x_q preko *ostalih promjenljivih*:

$$x_q = (b_p - a_{p,1} x_1 - a_{p,2} x_2 - \ldots - a_{p,q-1} x_{q-1} - a_{p,q+1} x_{q+1} - \ldots - a_{p,n+m} x_{n+m}) / a_{p,\,q}$$

Ukoliko ovaj izraz za x_q *uvrstimo u sve preostale jednačine i funkciju cilja*, ona će nestati iz njih (tj. u njima će se pojaviti sa koeficijentom 0). Pored toga, p-tu jednačinu treba *podijeliti sa* $a_{p,\,q}$, da bi se u toj jednačini dobila jedinica uz x_q. Na taj način će se nova bazna promjenljiva x_q pojaviti sa koeficijentom 1 u vodećoj vrsti (ograničenju), dok će u svim drugim ograničenjima kao i u funkciji cilja ona imati koeficijent 0.

Pokažimo šta ovo efektivno znači. Dijeljenje p-te jednačine sa $a_{p,\,q}$ znači da između starih koeficijenata $a_{p,\,j}, j = 1 \ldots n+m$ i b_p te novih koeficijenata $a_{p,\,j}', j = 1 \ldots n+m$ i b_p' postoje veze

$$a_{p,\,j}' = a_{p,\,j} / a_{p,\,q}\ ,\ j = 1 \ldots n+m$$

$$b_p' = b_p / a_{p,\,q}$$

Da bismo vidjeli kako se *transformiraju ostali koeficijenti*, uvrstimo gornji izraz za x_q u neku jednačinu koja nije u p-toj vrsti (recimo i-tu, $i \ne p$), za koju možemo uzeti da glasi

$$a_{i,1}x_1 + a_{i,2}x_2 + \ldots + a_{i,n+m}x_{n+m} = b_i$$

Nakon uvrštavanja izraza za x_q, ova jednačina postaje

$$(a_{i,1} - a_{i,q}a_{p,1}/a_{p,q})x_1 + (a_{i,2} - a_{i,q}a_{p,2}/a_{p,q})x_2 + \ldots + (a_{i,n+m} - a_{i,q}a_{p,n+m}/a_{p,q})x_{n+m} = b_i - a_{i,q}b_p/a_{p,q}$$

Isto tako, funkcija cilja, koja je ranije imala oblik

$$c_1 x_1 + c_2 x_2 + \ldots + c_{n+m} x_{n+m}$$

nakon ove smjene dobija oblik

$$(c_1 - c_q a_{p,1}/a_{p,q})x_1 + (c_2 - c_q a_{p,2}/a_{p,q})x_2 + \ldots + (c_{n+m} - c_q a_{p,n+m}/a_{p,q})x_{n+m} + c_q b_p/a_{p,q}$$

Slijedi da između starih i novih koeficijenata za $i \neq p$ vrijede sljedeće veze (ukoliko usvojimo da je "stara" vrijednost za F bila $F = 0$):

$$a_{i,j}' = a_{i,j} - a_{i,q}a_{p,j}/a_{p,q},\ i = 1..p-1, p+1..m,\ j = 1..n+m$$

$$b_i' = b_i - a_{i,q}b_p/a_{p,q},\ i = 1..p-1, p+1..m$$

$$c_j' = c_j - c_q a_{p,j}/a_{p,q},\ j = 1..n+m$$

$$F' = F - c_q b_p/a_{p,q}$$

Treba napomenuti da se gore navedene formule koriste uglavnom samo kada se algoritam programira na računaru. Pri ručnom radu, neophodne transformacije se tipično izvode *elementarnim manipulacijama sa jednačinama koje čine ograničenja*, kao što su *množenje/dijeljenje neke jednačine sa nekim brojem*, odnosno *dodavanje na neku jednačinu druge jednačine prethodno pomnožene nekim pogodnim brojem* (sa ciljem da se *anuliraju neželjeni koeficijenti*).

Nakon obavljene transformacije, dobijamo problem koji je po formi identičan polaznom problemu, samo sa drugim skupom baznih i nebaznih promjenljivih (odnosno, izvršili smo prelazak u *drugu vršnu tačku*). Zbog toga na novodobijeni problem možemo primijeniti *analogan postupak*. Postupak se nastavlja sve dok ne dođemo do problema izraženog u obliku da su svi koeficijenti u transformiranoj funkciji cilja *negativni*. U tom slučaju, postignuto je optimalno rješenje, odnosno *tekuće bazno rješenje je ujedno i optimalno*.

- ***Primjer*** : Koristeći simpleks algoritam, riješiti problem linearnog programiranja

$$\arg\max Z = 3x_1 + x_2$$

p.o.

$$0.5x_1 + 0.3x_2 \leq 150$$
$$0.1x_1 + 0.2x_2 \leq 60$$

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

Prvo je potrebno ovaj problem svesti na normalni (prošireni) oblik, što se lako postiže uvođenjem dvije dopunske (izravnavajuće) promjenljive x_3 i x_4. Modificirani problem glasi:

$$\arg\max Z = 3x_1 + x_2$$

p.o.

$$0.5x_1 + 0.3x_2 + x_3 = 150$$
$$0.1x_1 + 0.2x_2 + x_4 = 60$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0$$

Za bazne promjenljive, kao što je opisano, uzimamo *dopunske promjenljive*, odnosno početna baza je $B = (x_3, x_4)$. Polazne vrijednosti *nebaznih* promjenljivih su, naravno, nule, odnosno imamo $x_1 = 0$ i $x_2 = 0$. Ovo odmah daje i polazne vrijednosti baznih promjenljivih $x_3 = 150$ i $x_4 = 60$. Dakle, početno bazno dopustivo rješenje je $\mathbf{x} = (0, 0, 150, 60)$, a početna vrijednost funkcije cilja je 0.

Sada je potrebno naći novu baznu dopustivu tački u kojoj funkcija cilja ima "bolju" vrijednost. Pošto su u funkciji cilja koeficijenti uz *obje nebazne promjenljive pozitivni*, bilo koju od njih da uvedemo u bazu, funkcija cilja će se poboljšati. Odlučimo se za onu sa *većim koeficijentom* u funkciji cilja, tj. za promjenljivu x_1. Dakle, u bazu će ući promjenljiva x_1.

Sljedeći korak je *izbor promjenljive koja napušta bazu*. Pretpostavimo da nebaznu promjenljivu želimo povećati sa vrijednosti $x_1 = 0$ na neku vrijednost $x_1 = t$. Kako je $x_2 = 0$, iz ograničenja odmah slijedi da mora biti $0.5\,t + x_3 = 150$ i $0.1\,t + x_4 = 60$, odakle slijedi $x_3 = 150 - 0.5\,t$ i $x_4 = 60 - 0.1\,t$. Uvjeti nenegativnosti $x_3 \geq 0$ i $x_4 \geq 0$ nameću da mora biti $t \leq 150/0.5 = 300$ i $t \leq 60/0.1 = 600$, tako da je

$$t_{max} = \min\{300, 600\} = 300$$

Poveća li se, dakle, promjenljiva x_1 na vrijednost $x_1 = t_{max} = 300$, promjenljiva x_3 će dostići minimalnu dozvoljenu vrijednost 0. Dakle, *iz baze ispada promjenljiva* x_3. Na ovaj način je određena *vodeća kolona* ($q = 1$) i *vodeća vrsta* ($p = 1$). Slijedi da je vodeći element $a_{1,1} = 0.5$.

Sada je potrebno izvršiti transformaciju problema u oblik *prilagođen novoj situaciji*. Ukoliko iz jednačine u vodećoj vrsti (prve jednačine) izrazimo novu baznu promjenljivu x_1 preko ostalih promjenljivih, dobijamo

$$x_1 = (150 - 0.3\,x_2 - x_3) / 0.5 = 300 - 0.6\,x_2 - 2\,x_3$$

Uvrštavanjem ove relacije u drugu jednačinu dobijamo

$$0.1\,(300 - 0.6\,x_2 - 2\,x_3) + 0.2\,x_2 + x_4 = 60$$

odnosno

$$0.14\,x_2 - 0.2\,x_3 + x_4 = 30$$

Slično, funkcija cilja postaje:

$$Z = 3\,x_1 + x_2 = 3\,(300 - 0.6\,x_2 - 2\,x_3) + x_2 = -0.8\,x_2 - 6\,x_3 + 900$$

Konačno, potrebno je još prvu jednačinu podijeliti sa vodećim elementom (da bismo dobili jedinicu uz x_1 u prvoj jednačini). Na taj način, transformirani problem glasi

$$\arg\max Z = -0.8\,x_2 - 6\,x_3 + 900$$

p.o.

$$\begin{aligned} x_1 + 0.6\,x_2 \quad &+ 2\,x_3 \quad\quad\; = 300 \\ 0.14\,x_2 &- 0.2\,x_3 + x_4 = 30 \end{aligned}$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0$$

Ovo je problem po formi identičan početnom, samo su sada bazne promjenljive x_1 i x_4. Prije nego što nastavimo dalje, pokažimo kako smo istu transformaciju mogli izvršiti i direktnim manipuliranjem sa jednačinama, bez izvlačenja promjenljive x_1 i uvrštavanja. Naime, prvo u polaznom problemu podijelimo prvu jednačinu sa vodećim elementom, da bismo dobili jedinicu uz x_1. Time problem dobija oblik

$$\arg\max Z = 3\,x_1 + x_2$$

p.o.

$$\begin{aligned} x_1 + 0.6\,x_2 &+ 2\,x_3 \quad\quad = 300 \\ 0.1\,x_1 + 0.2\,x_2 \quad\quad &+ x_4 = 60 \end{aligned}$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0$$

Sada nam "smeta" koeficijent 0.1 uz x_1 u drugoj jednačini (trebao bi da bude 0). Stoga ćemo *od druge jednačine oduzeti prvu jednačinu pomnoženu sa* 0.1. Time dobijamo:

$$\arg\max Z = 3x_1 + x_2$$

p.o.

$$\begin{aligned} x_1 + 0.6x_2 \quad + 2x_3 \quad\quad &= 300 \\ 0.14x_2 - 0.2x_3 + x_4 &= 30 \end{aligned}$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0$$

Ostaje nam još da transformiramo i funkciju cilja. Manje je očigledno da je sličnim trikom moguće transformirati i funkciju cilja. Međutim, ukoliko jednačinu u vodećem redu napišemo u obliku

$$x_1 + 0.6x_2 + 2x_3 - 300 = 0$$

postaje jasno da na funkciju cilja možemo dodati (ili od nje oduzeti) lijevu stranu ove jednačine *pomnoženu sa bilo čime*, jer je njena vrijednost (lijeve strane) jednaka nuli. Kako nam smeta nenulti koeficijent 3 uz novu baznu promjenljivu x_1 u funkciji cilja, oduzmimo sada od funkcije cilja prethodnu jednačinu pomnoženu sa 3. Ovim ćemo dobiti

$$\arg\max Z = -0.8x_2 - 6x_3 + 900$$

p.o.

$$\begin{aligned} x_1 + 0.6x_2 \quad + 2x_3 \quad\quad &= 300 \\ 0.14x_2 - 0.2x_3 + x_4 &= 30 \end{aligned}$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0$$

što smo i željeli postići.

Pokažimo konačno i kako se koriste formule za transformaciju koeficijenata (koje se obično koriste prilikom implementacije simpleks algoritma na računaru):

$$a_{1,1}' = a_{1,1} / a_{1,1} = 0.5 / 0.5 = 1$$

$$a_{1,2}' = a_{1,2} / a_{1,1} = 0.3 / 0.5 = 0.6$$

$$a_{1,3}' = a_{1,3} / a_{1,1} = 1 / 0.5 = 2$$

$$a_{1,4}' = a_{1,4} / a_{1,1} = 0 / 0.5 = 0$$

$$b_1' = b_1 / a_{1,1} = 150 / 0.5 = 300$$

$$a_{2,1}' = a_{2,1} - a_{2,1}\, a_{1,1} / a_{1,1} = 0.1 - 0.1 \cdot 0.5 / 0.5 = 0.1 - 0.1 = 0$$

$$a_{2,2}' = a_{2,2} - a_{2,1}\, a_{1,2} / a_{1,1} = 0.2 - 0.1 \cdot 0.3 / 0.5 = 0.2 - 0.06 = 0.14$$

$$a_{2,3}' = a_{2,3} - a_{2,1}\, a_{1,3} / a_{1,1} = 0 - 0.1 \cdot 1 / 0.5 = 0 - 0.2 = -0.2$$

$$a_{2,j}' = a_{2,4} - a_{2,1}\, a_{1,4} / a_{1,1} = 1 - 0.1 \cdot 0 / 0.5 = 1 - 0 = 1$$

$$b_2' = b_2 - a_{2,1} b_1 / a_{1,1} = 60 - 0.1 \cdot 150 / 0.5 = 60 - 30 = 30$$

$$c_1' = c_1 - c_1 a_{1,1} / a_{1,1} = 3 - 3 \cdot 0.5 / 0.5 = 3 - 3 = 0$$

$$c_2' = c_2 - c_1 a_{1,2} / a_{1,1} = 1 - 3 \cdot 0.3 / 0.5 = 1 - 1.8 = -0.8$$

$$c_3' = c_3 - c_1 a_{1,3} / a_{1,1} = 0 - 3 \cdot 1 / 0.5 = 0 - 6 = -6$$

$$c_4' = c_4 - c_1 a_{1,4} / a_{1,1} = 0 - 3 \cdot 0 / 0.5 = 0 - 0 = 0$$

$$F' = F - c_1 b_1 / a_{1,1} = 0 - 3 \cdot 150 / 0.5 = -900$$

Uglavnom, kako god, nakon provedenih transformacija, dobili smo oblik problema iz kojeg očitavamo novo bazno rješenje u kojem su bazne promjenljive x_1 i x_4, a nebazne x_2 i x_3. Kako su nebazne promjenljive jednake nuli, odnosno $x_2 = 0$ i $x_3 = 0$, iz novog oblika problema odmah očitavamo $x_1 = 300$ i $x_4 = 30$. Dakle, nova baza je $B = (x_1, x_4)$, dok je novo bazno dopustivo rješenje $\mathbf{x} = (300, 0, 0, 30)^T$. Novu vrijednost funkcije cilja možemo očitati uvrštavanjem x_1 i x_2 u polazni oblik funkcije cilja kao $Z = 3x_1 + x_2 = 3 \cdot 300 + 0 = 900$, ali možemo je dobiti još jednostavnije iz *transformiranog oblika* funkcije cilja $Z = -0.8x_2 - 6x_3 + 900$. Naime, kako u transformiranom obliku figuriraju *samo nebazne promjenljive* čija je vrijednost svakako

jednaka nuli, nova vrijednost funkcije cilja je prosto jednaka *slobodnom članu* u transformiranoj funkciji cilja, odnosno $Z = 900$.

Nakon obavljene transformacije, dobili smo zadatak koji *po svom obliku potpuno odgovara početnom zadatku* (samo uz druge bazne promjenljive), tako da na njega možemo primijeniti *isti postupak promjene baznog rješenja* u potrazi za eventualnim boljim baznim rješenjem. Međutim, kako su u novoj (tj. transformiranoj) funkciji cilja *svi koeficijenti uz promjenljive negativni*, odnosno *nijedan od koeficijenata tekuće funkcije cilja nije veći od nule*, dobijeno bazno rješenje je ujedno i *optimalno rješenje zadatka*.

Tabelarni oblik simpleks algoritma

Algebarski oblik simpleks algoritma je pogodan za razumijevanje postupka kojim se, mijenjajući po određenim pravilima bazna dopustiva rješenja, može pronaći optimalno rješenje. Međutim, algebarski oblik je *nepodesan za praktičnu primjenu*, s obzirom da se tokom postupka *bespotrebno piše gomila stvari* (recimo, bespotrebno se pišu oznake varijabli uz koeficijente u jednačinama, mada su nam za postupak potrebni *samo koeficijenti*, odnosno *parametri modela*). Kako je algebarskim postupkom ujedno određen *način mijenjanja parametara modela pri prelasku sa jednog baznog dopustivog rješenja u drugo*, to omogućava da se poblem linearnog programiranja *predstavi u formi tabele* i da se, koristeći prethodno izvedena *pravila izmjene parametara*, dođe do optimalnog rješenja *kroz niz tabela*, bez potrebe da se sve jednačine modela eksplicitno ispisuju. Na taj način, svakom dozvoljenom baznom rješenju odgovara po jedna tabela. Prelasci sa tabele na tabelu vrše se prostim manipuliranjem sa elementima tabele.

Za primjenu tabelarne forme simpleks algoritma, problem linearnog programiranja se, nakon svođenja na normalnu formu, prikazuje u vidu sljedeće tabele (tzv. ***simpleks tabele***), pri čemu je smisao pojedinih elemenata u tabeli jasan iz same postavke problema:

Baza	b_i	x_1	x_2	...	x_n	x_{n+1}	x_{n+2}	...	x_{n+m}
x_{n+1}	b_1	$a_{1,1}$	$a_{1,2}$	...	$a_{1,n}$	1	0	...	0
x_{n+2}	b_2	$a_{2,1}$	$a_{2,2}$	...	$a_{2,n}$	0	1	...	0
...	...	...	...	...	...	...	...	...	...
x_{n+m}	b_m	$a_{m,1}$	$a_{m,2}$	...	$a_{m,n}$	0	0	...	1
	0 (F)	c_1	c_2	...	c_n	0	0	...	0

Treba naglasiti da u literaturi *ne postoji ujednačenost* oko toga gdje stoje pojedini elementi u tabeli, ali to ništa ne mijenja sam postupak. Na primjer, neki autori stupac koji sadrži vrijednosti b_i stavljaju *na kraj*, a ne na početak tabele. Isto tako, mnogi autori koeficijente u posljednjoj vrsti (koji predstavljaju koeficijente funkcije cilja izražene preko nebaznih promjenljivih) čuvaju *negirane* (ovo je posljedica neznatno drugačije logike kojim ti autori izvode algebarsku formu simpleks algoritma). Osim jedne male sitnice kod izbora vodeće kolone (obrnut smisao nejednakosti u koraku 3. navedenom ispod), to apsolutno ni na kakav način ne mijenja tok postupka, tako da je u osnovi svejedno koristi li se jedna ili druga konvencija.

Rješavanje problema linearnog programiranja pomoću tabelarne verzije simpleks algoritma teče u sljedećim koracima, čiji smisao direktno slijedi iz algebarske verzije simpleks algoritma:

1. Formirati *početnu simpleks tabelu*.
2. Ukoliko su svi koeficijenti c_j u posljednjem redu tabele *negativni ili nule*, preći na korak 9.
3. Odabrati neku kolonu q u kojoj vrijedi da je $c_q > 0$ (recimo, kolonu u kojoj c_q ima *najveću vrijednost* ukoliko prihvatimo *Dantzigovo pravilo pivotiranja*) i proglasiti je za *vodeću kolonu.* Indeks q te kolone određuje *indeks promjenljive koja će ući u bazu*.
4. Ukoliko su svi elementi vodeće kolone *negativni ili nule*, funkcija cilja *nije ograničena odozgo na dopustivom prostoru*, odnosno optimalno rješenje je *u beskonačnosti*. Algoritam se tada prekida.
5. Za sve elemente $a_{i,\,q}$ vodeće kolone koji su *pozitivni* formirati odgovarajuće količnike $b_i/a_{i,\,q}$ (ovo se pri ručnom radu obično radi *desno od tabele*, kraj odgovarajućeg reda na koji se taj količnik odnosi) i pronaći red u kojem je taj količnik *najmanji*. Indeks p tog reda određuje *promjenljivu koja napušta bazu* (ne njen indeks; o kojoj se zaista promjenljivoj radi možemo vidjeti iz kolone tabele u kojoj se čuva informacija o tekućoj bazi). Taj red proglašavamo za *vodeći red*.
6. U koloni koja čuva informaciju o tekućoj bazi, *zamijenimo* promjenljivu koja napušta bazu promjenljivom koja ulazi u bazu.

7. Izvršimo transformaciju tabele tako da se na mjestu *vodećeg elementa* (u presjeku vodećeg reda i vodeće kolone) pojavi *jedinica*, a da svi ostali elementi u vodećoj koloni postanu jednaki *nuli*. Transformacija se može vršiti bilo po *izvedenim pravilima za transformaciju parametara modela* pri prelasku iz jednog dozvoljenog baznog rješenja u drugo (što se obično radi pri programiranju metoda na računaru), bilo vršenjem elementarnih operacija nad redovima tablice (što se obično radi pri ručnom radu). Legalne elementarne operacije su množenje ili dijeljenje čitavog reda nekim brojem (što se tipično izvodi nad vodećim redom), odnosno dodavanje ili oduzimanje na čitav red elemenata nekog drugog reda pomnoženog ili podijeljenog sa nekim brojem (što se tipično izvodi nad redovima koji nisu vodeći, dok je onaj drugi red koji se množi ili dijeli vodeći red).
8. Sa tako transformiranom tablicom vraćamo se na korak 2.
9. Optimalno rješenje je *nađeno*. Vrijednosti baznih promjenljivih očitavaju se iz odgovarajućih elemenata kolone koji predstavljaju vrijednosti b_i, dok ostale (nebazne) promjenljive imaju vrijednost jednaku nuli. Vrijednost koja se na kraju postupka dobije u posljednjem redu kolone u kojoj se inače nalaze vrijednosti b_i predstavlja *negiranu optimalnu vrijednost funkcije cilja*.

Ovim je završen opis tabelarne verzije simpleks algoritma. Ovdje treba ukazati na još jednu sitnicu. Ukoliko se na kraju postupka desi da za neku od nebaznih promjenljivih x_q imamo koeficijent $c_q = 0$, to znači da promjenljivu x_q možemo *uvesti u bazu* a da se vrijednost funkcije cilja *ne promijeni*. Efektivno, ovo znači da optimalno rješenje *nije jedinstveno*. Uvođenjem x_q u bazu i nastavljanjem postupka na uobičajeni način možemo preći sa jednog na drugo optimalno rješenje.

- ***Primjer*** : Riješiti problem iz prethodnog primjera koristeći tabelarnu formu simpleks algoritma.

Rješavanje započinjemo formiranjem početne simpleks tabele:

Baza	b_i	x_1	x_2	x_3	x_4
x_3	150	0.5	0.3	1	0
x_4	60	0.1	0.2	0	1
	0	3	1	0	0

Kako je najveća vrijednost u posljednjem redu 3, to ćemo kolonu koja odgovara x_1 proglasiti vodećom kolonom, što je u sljedećoj tabeli prikazano osjenčeno. Promjenljiva x_1 dakle ulazi u bazu. Potrebno je sada odrediti koja promjenljiva *napušta bazu*. Za tu svrhu formiramo količnike $150/0.5 = 300$ i $60/0.1 = 600$, što je prikazano sa desne strane sljedeće tabele. Najmanji količnik se postiže u redu koji odgovara promjenljivoj x_3. Stoga ona *napušta bazu*, a odgovarajući red proglašavamo za vodeći red (također prikazano osjenčeno).

Baza	b_i	x_1	x_2	x_3	x_4	
x_3	150	0.5	0.3	1	0	$150/0.5 = 300$
x_4	60	0.1	0.2	0	1	$60/0.1 = 600$
	0	3	1	0	0	

Sada je potrebno transformirati ovu tabelu tako da se u vodećoj koloni pojave sve same nule, osim na mjestu *vodećeg elementa*, tj. u presjeku vodećeg reda i vodeće kolone (što je u prethodnoj tabeli prikazano dvostruko osjenčeno), gdje treba da se pojavi jedinica. Za tu svrhu, prvo ćemo *vodeći red pomnožiti sa 2*, čime smo dobili jedinicu na mjestu vodećeg elementa. Nakon toga ćemo tako transformirani red *pomnožiti sa* 0.1 i *oduzeti ga* od drugog reda, čime dobijamo nulu na presjeku drugog reda i vodeće kolone. Konačno ćemo transformirani vodeći red *pomnožiti sa* –3 i *oduzeti ga* od posljednjeg reda, čime dobijamo nulu na presjeku posljednjeg reda i vodeće kolone. Na taj način se dobija sljedeća simpleks tabela:

Baza	b_i	x_1	x_2	x_3	x_4
x_1	300	1	0.6	2	0
x_4	30	0	0.14	–0.2	1
	–900	0	–0.8	–6	0

Sa ovako dobijenom simpleks tabelom vraćamo se nazad na početak. Međutim, kako su sada svi koeficijenti u posljednjem redu tabele *manji ili jednaki nuli*, postupak se završava, odnosno tekuće bazno dopustivo rješenje je optimalno. Iz tabele sad očitavamo da je baza optimalnog rješenja $B = (x_1, x_4)$. Također direktno očitavamo i da je $x_1 = 300$ i $x_4 = 30$, dok su nebazne promjenljive svakako jednake nuli ($x_2 = x_3 = 0$). Drugim riječima, optimalno rješenje je $\mathbf{x} = (300, 0, 0, 30)^T$. Također, iz tabele se direktno može očitati i optimalna vrijednost funkcije cilja $Z = 900$.

Prilikom transformacija simpleks tabela, veoma je poželjno rezultate koji nisu cijeli brojevi predstavljati u formi *razlomaka* (pogotovo pri ručnom radu), jer se na taj način izbjegavaju *nezgrapne manipulacije sa brojevima koji imaju mnogo decimala*, a također se izbjegava i *akumulacija grešaka zaokruživanja* koja može nastati ukoliko međurezultate ne vodimo sa dovoljnom tačnošću. Ovakav pristup ilustriran je u sljedećem primjeru.

- ***Primjer***: Za proizvodnju dva proizvoda P_1 i P_2 koriste se tri sirovine S_1, S_2 i S_3. U sljedećoj tabeli su prikazani utrošci svake od sirovina S_1, S_2 i S_3 po jednom kilogramu (*kg*) proizvoda P_1 odnosno jednom litru (*l*) proizvoda P_2, zatim raspoložive količine (zalihe) sirovina S_1, S_2 i S_3 u skladištu, te prodajne cijene proizvoda P_1 i P_2 po jedinici proizvedene količine (kilogramu odnosno litru).

	Proizvodi		
Sirovine	P_1	P_2	Zalihe
S_1	30 *l*/*kg*	16 *l*/*l*	22800 *l*
S_2	14 *kg*/*kg*	19 *kg*/*l*	14100 *kg*
S_3	11 *kom*/*kg*	26 *kom*/*l*	15900 *kom*
Cijene	800 \$/*kg*	1000 \$/*l*	

Pored toga, procjena je da se zbog ograničenih mogućnosti tržišta, u posmatranom periodu ne treba proizvoditi više od 550 litara proizvoda P_2.

U ovom primjeru namjerno je uzeto da su jedinice mjere količine proizvoda P_1 i P_2 te zaliha sirovina S_1, S_2 i S_3 *raznorodne*. Ovo je, naravno, sasvim legalno, ali moramo paziti da mjerne jedinice *moraju biti konzistentne* (npr. proizvod mjernih jedinica kojima se mjeri $a_{i,j}$ i x_j mora se slagati sa mjernom jedinicom kojom se mjeri b_i). Lako se vidi da matematski model ovog problema glasi:

$$\arg\max Z = 800\,x_1 + 1000\,x_2$$

p. o.

$$\begin{aligned} 30\,x_1 + 16\,x_2 &\le 22800 \\ 14\,x_1 + 19\,x_2 &\le 14100 \\ 11\,x_1 + 26\,x_2 &\le 15950 \\ x_2 &\le 550 \end{aligned}$$

$$x_1 \ge 0,\ x_2 \ge 0$$

Uvođenjem četiri dopunske promjenljive, dobijamo normalni model u obliku:

$$\arg\max Z = 800\,x_1 + 1000\,x_2$$

p. o.

$$\begin{aligned} 30\,x_1 + 16\,x_2 + x_3 &= 22800 \\ 14\,x_1 + 19\,x_2 + x_4 &= 14100 \\ 11\,x_1 + 26\,x_2 + x_5 &= 15950 \\ x_2 + x_6 &= 550 \end{aligned}$$

$$x_1 \ge 0,\ x_2 \ge 0,\ x_3 \ge 0,\ x_4 \ge 0,\ x_5 \ge 0,\ x_6 \ge 0$$

Sada možemo formirati početnu simpleks tabelu:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	
x_3	22800	30	16	1	0	0	0	22800/16 = 1425
x_4	14100	14	19	0	1	0	0	14100/19 ≈ 742
x_5	15950	11	26	0	0	1	0	15950/26 ≈ 613
x_6	550	0	1	0	0	0	1	550/1 = 550
	0	800	1000	0	0	0	0	

Primjenom opisanog postupka, lako ustanovljavamo da je vodeća kolona kolona koja odgovara promjenljivoj x_2 (koja će ući u bazu). Nakon toga, ustanovljavamo da je vodeći red onaj koji odgovara promjenljivoj x_6 (koja napušta bazu). Kako je vodeći element već jednak jedinici, sve što trebamo uraditi je postići da ostali elementi u vodećoj koloni budu nule. Za tu svrhu ćemo od prvog, drugog, trećeg i posljednjeg reda oduzeti vodeći red pomnožen sa 16, 19, 26 i 1000 respektivno. Time dobijamo sljedeću simpeks tabelu:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	
x_3	14000	30	0	1	0	0	–16	$14000/30 \approx 467$
x_4	3650	14	0	0	1	0	–19	$3650/14 \approx 261$
x_5	1650	11	0	0	0	1	–26	$1650/11 = 150$
x_2	550	0	1	0	0	0	1	
	–550000	800	0	0	0	0	–1000	

Promjenljiva koja sada ulazi u bazu je x_1, dok bazu napušta promjenljiva x_5. Primijetimo da u redu koji odgovara promjenljivoj x_2 nismo računali količnik, s obzirom da se u vodećoj koloni na tom mjestu nalazi *nula*. Nakon transformacije, dobija se sljedeća simpleks tabela:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	
x_3	9500	0	0	1	0	–30/11	604/11	$9500/(604/11) \approx 173$
x_4	1550	0	0	0	1	–14/11	155/11	$1550/(155/11) = 110$
x_1	150	1	0	0	0	1/11	–26/11	
x_2	550	0	1	0	0	0	1	$550/1 = 550$
	–670000	0	0	0	0	–800/11	9800/11	

U bazu sada ulazi x_6, a izlazi x_4. U redu koji odgovara promjenljivoj x_4 nismo računali količnik, jer je odgovarajući element vodeće kolone *negativan*. Transformacija daje sljedeću simpleks tabelu:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	
x_3	3460	0	0	1	–604/155	346/155	0	$3460/(346/155) = 1550$
x_6	110	0	0	0	11/155	–14/155	1	
x_1	410	1	0	0	26/155	–19/155	0	
x_2	440	0	1	0	–11/155	14/155	0	$440/(14/155) \approx 4871$
	–768000	0	0	0	–1960/31	240/31	0	

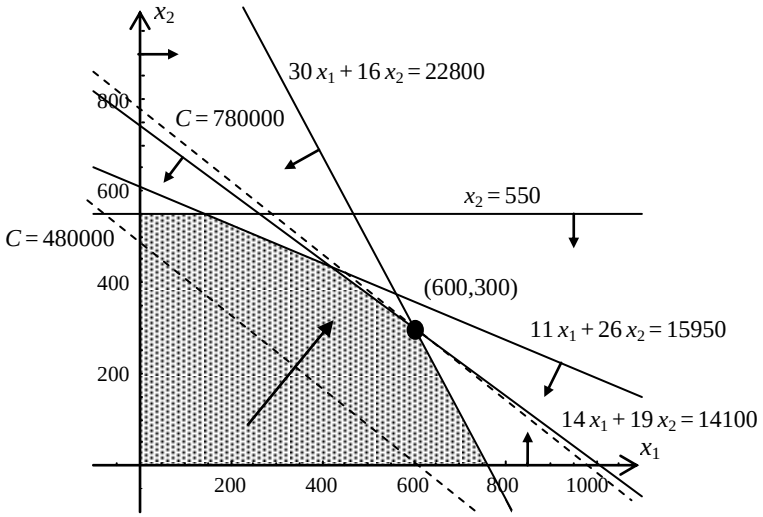
Nova bazna promjenljiva će biti x_5, a bazu napušta x_3. Nakon obavljenih transformacija, dolazimo do sljedeće simpleks tabele:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6
x_5	1550	0	0	155/346	–302/173	1	0
x_6	250	0	0	7/173	–15/173	0	1
x_1	600	1	0	19/346	–8/173	0	0
x_2	300	0	1	–7/173	15/173	0	0
	–780000	0	0	–600/173	–8600/173	0	0

Konačno smo dobili tabelu u kojoj su svi elementi u posljednjem redu negativni, tako da smo dobili optimalno rješenje. Dakle, baza optimalnog rješenja je $B = (x_1, x_2, x_5, x_6)$, odnosno samo optimalno rješenje glasi $\mathbf{x} = (600, 300, 0, 0, 1550, 250)^T$, dok je optimalna vrijednost funkcije cilja $Z = 780000$.

Prodiskutirajmo malo ovo rješenje. Uzimajući u obzir pojedine jedinice mjera, ovo rješenje možemo iskazati kao $x_1 = 600$ *kg*, $x_2 = 300$ *l*, $x_3 = 0$ *l*, $x_4 = 0$ *kg*, $x_5 = 1550$ *kom* i $x_6 = 250$ *l*. Dakle, slijedi da treba proizvoditi 600 *kg* proizvoda P_1 i 300 *l* proizvoda P_2, pri čemu će se ostvariti optimalna zarada od 780000 \$. Dalje, zbog $x_3 = x_4 = 0$, raspoložive zalihe sirovina S_1 i S_2 će biti u potpunosti iskorištene, odnosno na kraju proizvodnje *neće ostati nimalo tih sirovina*. Dakle, ograničenja na sirovine S_1 i S_2 u ovom primjeru su *kritična* (ispunjena su tačno do na jednakost), odnosno predstavljaju "*usko grlo*" zbog kojeg nije moguće dalje povećanje zarade (odavde slijedi da je zaradu moguće poboljšati jedino *ukoliko nabavimo dodatne količine sirovina* S_1 i S_2). S druge strane, zbog $x_5 = 1550$ *kom*, zalihe sirovine S_3 *neće biti u potpunosti iskorištene*, odnosno na kraju proizvodnje u skladištu će ostati neiskorištenih 1550 komada sirovine S_3, tako da ograničenje na količinu sirovine S_3 *nije* "*usko grlo*" u ovom problemu. Dakle, barem zasada, *nema potrebe* za nabavkom dodatnih količina sirovine S_3. Konačno, zbog $x_6 = 250$ *l*, vidimo da na tržištu ima prostora za još 250 litara proizvoda P_2, za koji smo se odlučili da ga ne proizvodimo. Dakle, ni ograničenje na maksimalnu potražnju tržišta za proizvodom P_2 nije "usko grlo" koje nam ograničava zaradu.

Ilustrativno je prikazati i rješenje istog problema grafičkim putem (što je moguće, jer je problem *dvodimenzionalan*). Na sljedećoj slici je prikazana dopustiva oblast, te dva pravca funkcije cilja $800\,x_1 + 1000\,x_2 = C$, prvi za nasumice odabranu vrijednost $C = 480000$, a drugi za vrijednost $C = 780000$ koji odgovara optimalnoj vrijednost funkcije cilja, s obzirom da takav pravac tangira dopustivu oblast. Treba obratiti pažnju da je prilično teško precizno nacrtati crtež koji odgovara ovom problemu, s obzirom da koeficijent smjera pravca $14\,x_1 + 19\,x_2 = 14000$, koji predstavlja granicu jednog od ograničenja, iznosi $k = -14/19 \approx -0.737$, te je stoga vrlo blizak koeficijentu smjera pravaca funkcije cilja $k_Z = -800/1000 = 0.8$. Slijedi da su pravci funkcije cilja *skoro paralelni* ovom ograničenju, tako da je prilično teško odrediti u kojoj tačno tački neki od pravaca funkcije cilja tangiraju dopuštenu oblast. Zbog toga je za primjenu grafičkog metoda od velike važnosti izračunati koeficijente smjera svih pravaca koji se javljaju u crtežu, da bi bilo jasno koji od njih ima veći, a koji manji nagib.



Još je interesantno pratiti *kojim je redom* simpleks algoritam posjećivao vršne tačke. Ukoliko iz razmatranih simpleks tabela posmatramo samo stanje promjenljivih x_1 i x_2, koje su postojale i u polaznom problemu, vidimo da niz simpleks tabela koji smo imali odgovara nizu prelazaka $(0, 0) \to (0, 550) \to (150, 550) \to (410, 440) \to (600, 300)$, kada je dostignut optimum. Međutim, isto tako treba primijetiti da ovaj prelaz *nije ujedno i najkraći niz prelaza* duž bridova dopustivog prostora kojim smo iz početnog dopustivog rješenja došli do optimalnog rješenja. Zaista, postoji kraći niz prelaza $(0, 0) \to (760, 0) \to (600, 300)$, kao što se vidi sa slike. O ovome će kasnije biti više govora.

Kostur za implementaciju simpleks algoritma

Na osnovu prethodno datog opisa simpleks algoritma, *nije teško sastaviti pseudokôd za implementaciju simpleks algoritma*. Ispod je prikazan jedan takav pseudokôd, koji *teoretski* radi za sve probleme linearnog programiranja u standardnom obliku kod kojih su svi koeficijenti b_i, $i = 1\,..\,m$ pozitivni (mada uglavnom radi i kada su neki od njih nule, o čemu će uskoro biti detaljnije govora). Ipak, ovaj kôd je *čisto školska verzija* i potrebne su mu neke dorade da bi bio *praktično upotrebljiv za veće probleme*, o čemu će također biti govora. Objašnjenja nekih *tehničkih detalja* u ovom pseudokôdu biće data odmah nakon njegovog prikaza.

```
for i = 1 .. m                          // inicijalizacija početne simpleks tabele
    BASE_i ← n + i
    a_{i,0} ← b_i
    for j = n + 1 .. n + m
        a_{i,j} ← 0
    a_{i,n+i} = 1
for j = 1 .. n
    a_{0,j} ← c_j
for j = n + 1 .. n + m
    a_{0,j} ← 0
a_{0,0} ← 0
```

```
optimal ← false                          // iteracije se vrše dok se ne dostigne optimum
while ¬ optimal
    cmax ← –1                            // pronalaženje promjenljive koja ulazi u bazu
    for j = 1 .. n + m
        if a0, j > 0 ∧ a0, j > cmax
            cmax ← a0, j
            q ← j
    if cmax = –1                         // ako su svi cj negativni, optimalno rješenje je nađeno
        optimal ← true
    else
        tmax ← ∞                         // pronalaženje promjenljive koja napušta bazu
        for i = 1 .. m
            if ai,q > 0
                if ai,0/ai,q < tmax
                    tmax ← ai,0/ai,q
                    p ← i
      if tmax = ∞
          return "Rješenje je neograničeno"
      BASEp ← q                          // promjena baze
      pivot ← ap,q
      for j = 0 .. m + n                 // prilagođavanje parametara modela novoj bazi
          ap,j ← ap,j / pivot
      for i = 0 .. m
          if i ≠ p
              factor ← ai,q
              for j = 0 .. n + m
                  ai, j ← ai, j – factor · ap, j
for j = 1 .. n + m                       // očitavanje optimalnog rješenja iz tabele
    xj ← 0
for i = 1 .. m
    xBASEi ← ai,0
Z ← a0,0
return x, Z
```

U ovom pseudokôdu je pretpostavljeno da se ulazni podaci nalaze u matrici **A** čiji su elementi $a_{i,j}$, $i = 1 \ldots m$, $j = 1 \ldots n$ te vektorima **b** i **c** čiji su elementi b_i, $i = 1 \ldots m$ odnosno c_j, $i = 1 \ldots n$ respektivno. Međutim, isto tako je pretpostavljeno da je matrica **A** dimenzionirana na dovoljnu veličinu da može da prihvati ne samo koeficijente izvornog modela, nego i koeficijente proširenog modela, pa čak i cijelu simpleks tabelu (uključujući dodatnu kolonu za koeficijente b_i odnosno dodatni red za koeficijente c_j). U jezicima koji su proizašli iz jezika *C* (*C++*, *Java*, *C#*, itd.) u kojima indeksi redova i kolona idu *od nule* a ne od jedinice, najprirodnije je koeficijente b_i odnosno c_j upravo čuvati u *nultoj koloni* odnosno *nultom redu* tablice, kao što je i urađeno u gore prikazanom kôdu (u slučaju da korišteni jezik ne podržava indeksiranje od nule, tada je najprirodnije koeficijente b_i odnosno c_j čuvati u $(n+m+1)$-voj koloni odnosno $(m+1)$-vom redu tablice, uz odgovarajuće trivijalne izmjene u prikazanom kôdu, koje uključuju i da će se pojedine petlje umjesto od 0 do $n+m$ odnosno od 0 do m kretati od 1 do $n+m+1$ odnosno od 1 do $m+1$). Inače, prikazana implementacija je direktno zasnovana na ranije datom opisu simpleks algoritma. Napomenimo još da vektor *BASE* čuva indekse promjenljivih koji se trenutno nalaze u bazi.

Bez obzira što gore prikazani kôd funkcionira, u njemu je potrebno napraviti neke izmjene da bi bio upotrebljiv i za veće probleme. Na prvom mjestu, očigledan nedostatak je u činjenici da se u prikazanoj implementaciji *potpuno nepotrebno čuvaju kolone koje odgovaraju baznim promjenljivim*, mada je tačno poznato kako njihovi elementi glase (u svakoj koloni je tačno jedna jedinica u tačno jednom redu, dok su ostali nule). Također se potpuno nepotrebno *gubi vrijeme na proračunavanje tih elemenata*. Ukoliko nekako izvedemo da se ti elementi ne čuvaju i ne proračunavaju, uštedićemo m^2 elemenata u memoriji, te vrijeme potrebno za proračunavanje m^2 elemenata po svakoj iteraciji, što za veće vrijednosti m može biti itekako značajno. Ovo nije preteško izvesti. Naime, u matrici nećemo čuvati *čitavu simpleks tabelu* nego samo kolone koje odgovaraju nebaznim promjenljivim, a imaćemo neki vektor koji govori *kojim promjenljivim odgovara koja kolona matrice u kojoj čuvamo elemente tablice koji se moraju preračunavati*. Nakon toga, pristup elementima simpleks tabele vršimo *indirektno*, prvo konsultirajući taj vektor da vidimo *u kojem redu matrice* koja čuva elemente (kompresovane) tablice taj element treba tražiti, pa tek tada vršimo pristup

odgovarajućem elementu. U svakom slučaju, ovo je pristup koji se *mora* uraditi ako želimo da naš program rješava iole veće probleme od problema iz školskih zbirki zadataka.

Sljedeća stvar je činjenica da mi još nismo riješili problem *negativnih desnih strana u ograničenjima tipa* "*manje ili jednako*" kao ni tretman ograničenja tipa "veće ili jednako" ili ograničenja tipa jednakosti (o čemu ćemo kasnije govoriti). Također ćemo uskoro vidjeti i da pojava tzv. *degeneriranih baznih rješenja* također može dovesti do problema. Sve ovo ozbiljni programi za rješavanje problema linearnog programiranja moraju također uzeti u obzir. Dalje, u većim problemima se tipično jako brzo akumuliraju greške zaokruživanja, koje mogu totalno degradirati rješenje. One se tipično mogu ublažiti prikladnim *skaliranjem ograničenja*, odnosno njihovim *množenjem* sa prikladno odabranim brojevima. Sve ovo dobar softver za rješavanje problema linearnog programiranja mora raditi *automatski*, bez potrebe za ručnom intervencijom od strane korisnika. Isto tako, u većim problemima tipično se dešava da je veliki broj koeficijenata u ograničenjima *jednak nuli*, odnosno da su matrice koeficijenata *rijetke*. Dobro napisan kôd za rješavanje problema linearnog programiranja mora naći način da *ne pamti bespotrebno ove koeficijente u memoriji*, niti *da vrši proračunavanje vrijednosti za koje se zna da će biti jednake nuli*, te da vrši *razne druge trikove kojima se može ostvariti memorijska i vremenska ušteda*. Pored toga, vidjećemo da se upotrebom *različitih pravila pivotiranja* često može smanjiti broj iteracija koji je potreban da se dođe do optimalnog rješenja. Također, nekada je u toku postupka poželjno *preći na neki od modifikacija simpleks metoda* (kao što su recimo *revidirani simpleks metod* ili *dualni simpleks metod*, o kojem će kasnije biti riječi), ukoliko se ustanovi da bi se *tako moglo prije doći do rješenja*. Bespotrebno je i govoriti da bi dobre implementacije programa za rješavanje problema linearnog programiranja sve ovo trebale prepoznavati i izvoditi potpuno automatski. Stoga ne treba da čudi da, bez obzira na činjenicu da gore prikazani kôd ima manje od pedesetak linija, dobre implementacije simpleks algoritma (koje uzimaju u obzir sve gore pobrojane činjenice i vrše razne trikove za memorijsku i vremensku optimizaciju) tipično imaju nekoliko hiljada, pa čak i desetina hiljada linija programskog kôda.

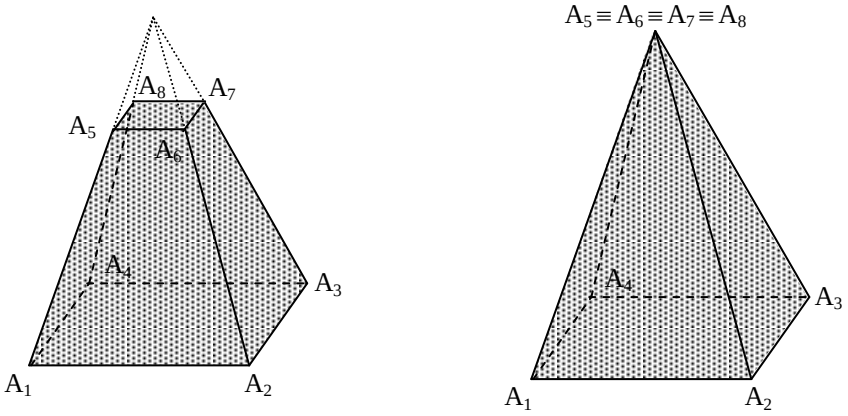
S obzirom da je dosta teško napisati *kvalitetnu* implementaciju kôda za rješavanje problema linearnog programiranja (mada je *naivna* implementacija gotovo trivijalna), sretna je okolnost da postoji *veliki broj takvih napisanih dobrih implementacija*, bilo u vidu samostalnih programa, bilo u vidu *biblioteka* za klasične programske jezike (*C*, *C++*, *Java*, itd.). Mnoge od takvih implementacija su dostupne *besplatno*, tj. kao *freeware* softver. Tu vrijedi istaći alate poput ***lp_solve*** (*http://groups.yahoo.com/group/lp_solve*), zatim ***CLP*** (*http://www.coin-or.org/Clp*) te ***GLPK*** (*http://www.gnu.org/software/glpk*). Neki programski jezici, kao što je npr. *Matlab* imaju ugrađene funkcije za rješavanje problema linearnog programiranja (konkretno, recimo funkcija "`linprog`" u *Matlab-u*). Konačno, ne treba zaboraviti i na *komercijalne softverske pakete* za rješavanje problema ne samo linearnog nego i općenito matematičkog programiranja (od kojih je ***LINDO*** jedan od najviše korištenih), koji *iako nisu besplatni*, imaju *besplatne demo verzije* koje su dovoljno moćne da mogu rješavati probleme manjih ili srednjih dimenzija. Sve ovo čini tehnike linearnog programiranja jednim od *danas najraspoloživijih tehnika uopće u svijetu primjena matematičkih metoda u nauci i tehnici*.

Problem degeneracije

Ranije smo rekli da u svakom baznom dopustivom rješenju *najviše m* promjenljivih mogu biti različite od nule, dok *barem n* promjenljivih imaju vrijednosti koje su jednake nuli. U svim dosada razmotrenim primjerima, imali smo situaciju da je *tačno m promjenljivih* različito od nule, i to su upravo *bazne promjenljive* u razmatranom baznom rješenju. Međutim, može se dogoditi da u nekom dopustivom baznom rješenju ima *manje od m* promjenljivih koje su različite od nule, odnosno *više od n* promjenljivih koje su jednake nuli. Pošto baznih promjenljivih ima ukupno *m*, to znači da u takvom baznom rješenju *barem jedna od baznih promjenljivih mora također biti jednaka nuli*. Za takva dopustiva bazna rješenja kažemo da su ***degenerirana***.

Degenerirana bazna rješenja mogu nastati na *dva načina*. Prvo, može se desiti da odmah *početno bazno dopustivo rješenje bude degenerirano*. Očito, to će se dogoditi ako i samo ako je *barem jedan od koeficijenata* b_i, $i = 1 \,..\, m$ *jednak nuli*. Drugo, degenerirana bazna rješenja mogu nastati u toku izvršavanja simpleks algoritma ukoliko se dogodi da pri određivanju t_{max} i promjenljive koja napušta bazu *više različitih količnika* $b_i / a_{i,\,q}$ za različite vrijednosti i ima istu vrijednost t_{max}. U tom slučaju, izbor promjenljive koja napušta bazu *nije jednoznačan*. Međutim, za koju god se promjenljivu (od više mogućnosti) odlučimo da napusti bazu, nakon promjene će postati nula ne samo ona, nego i sve ostale bazne promjenljive u redovima za koje je količnik $b_i / a_{i,\,q}$ imao istu vrijednost. Dakle, dobijamo novo dopustivo bazno rješenje u kojem će neke bazne promjenljive imati vrijednost nula, odnosno dobijamo *degenerirano dopustivo bazno rješenje*.

Pojava degeneracije *obično nije problem*, odnosno prethodno opisana opća struktura simpleks algoritma *najčešće uspješno rješava i probleme u kojima se javljaju degenerirana bazna rješenja*. Međutim, u izvjesnim situacijama, *degeneracija može dovesti do problema*, uključujući i situaciju da simpleks algoritam *upadne u ciklus*, odnosno *petlju bez izlaza* (što znači da se optimalno rješenje *nikada neće dostići*), ukoliko se ne preduzmu specijalne mjere za prevazilaženje takvih problema. Da bismo shvatili *zbog čega* degenerirana bazna rješenja mogu izazvati probleme, moramo prvo vidjeti tačno *kako takva rješenja nastaju* i *šta ona uopće znače*. Posmatrajmo stoga sljedeću sliku:



Lijevo je prikazana hipotetička dopustiva oblast nekog trodimenzionalnog problema oblika *krnje piramide*, koja očito ima 8 vršnih tačaka, tako da odgovarajući problem ima 8 dozvoljenih baznih rješenja. S druge strane, desno je prikazana također hipotetička dopustiva oblast, ali oblika *obične* a ne krnje *piramide* (četverostrane). Na prvi pogled, izgleda da ova dopustiva oblast ima 5 vršnih tačaka, te da također treba biti 5 dozvoljenih baznih rješenja. Međutim, slika sa desne strane se može posmatrati kao granični slučaj slike sa lijeve strane u kojoj se tjemena gornje osnovice krnje piramide *stežu u jednu tačku*, tako da se vršna tačka piramide koja se nalazi u njenom gornjem vrhu može shvatiti kao da je nastala *stapanjem četiri vršne tačke u jednu* (za takvu vršnu tačku *također kažemo da je degenerirana*). Simpleks algoritam upravo tako i vidi ovu vršnu tačku: *ne kao jednu, nego kao četiri vršne tačke* koje se nalaze na istom mjestu. Stoga, ovoj vršnoj tački odgovaraju četiri *različita* dozvoljena bazna rješenja, koja se razlikuju *samo po tome koje su promjenljive u bazi, a koje nisu* (vrijednosti svih promjenljivih, kao i vrijednosti funkcije cilja u ovim dozvoljenim baznim rješenjima su potpuno iste). Općenito, degenerirana bazna rješenja nastaju kad god dođe do stapanja *više različitih vršnih tačaka u jednu degeneriranu vršnu tačku*. Iz takvih degeneriranih tačaka *uvijek izlazi više bridova nego što je dimeznionalnost problema* (4 brida u gore prikazanom primjeru), dok iz "normalnih" vršnih tačaka uvijek izlazi *tačno onoliko bridova kolika je dimenzionalnost problema* (3 u gore prikazanom primjeru).

Sada možemo objasniti zbog čega degeneracija može praviti probleme u simpleks algoritmu. Glavni problem uzrokovan je činjenicom da se u degeneriranim baznim rješenjima često događa da je $t_{max} = 0$, tako da nakon provedene transformacije parametara modela, vrijednost funkcije cilja *ostane ista* umjesto da se poveća (s obzirom da porast funkcije cilja iznosi $c_q t_{max}$). Ovo zapravo znači da smo prešli *iz jedne vršne tačke u drugu vršnu tačku koja se nalazi na istom mjestu kao i prethodna* (geometrijski, $t_{max} = 0$ znači da su svi "bridovi" koji izlaze iz takve tačke dužine 0 i da oni mogu samo voditi u vršnu tačku koja je na istom mjestu kao i tekuća). S obzirom da se konačnost vremena izvršavanja simpleks algoritma oslanja na pretpostavku da se funkcija cilja *stalno povećava* iz iteracije u iteraciju, javlja se bojazan *da li simpleks algoritam zaista terminira* u slučaju da se pojave degenerirana bazna rješenja. U većini slučajeva, odgovor je potvrdan, s obzirom da najčešće nakon malo "lutanja" po vršnim tačkama koje su se sastale u jednu simpleks algoritam naiđe na neku u kojoj će t_{max} biti veći od nule (tj. da nađe tačku iz koje izlazi brid koji zaista vodi u neku različitu vršnu tačku), tako da će u sljedećoj iteraciji algoritam izaći iz degenerirane vršne tačke. Međutim, zna se dogoditi da algoritam *prosto upadne u ciklus* izmjenjujući jedno bazno rješenje sa drugim, misleći da prelazi iz jedne u drugu vršnu tačku, a da zapravo efektivno svo vrijeme "kruži" unutar jedne degenerirane vršne tačke u kojoj se sastalo mnogo različitih dozvoljenih (degeneriranih) baznih rješenja. Ovakve situacije su *veoma rijetke*, ali se ipak mogu desiti. Najpoznatiji takav primjer dao je *Beale* (1962), a on nastaje pri rješavanju problema

$$\arg\min Z = -3x_1 + 80x_2 - 2x_3 + 24x_4$$

p.o.

$$\begin{aligned}(1/4)x_1 - 8x_2 - x_3 + 9x_4 &\le 0\\ (1/2)x_1 - 12x_2 - (1/2)x_3 + 3x_4 &\le 0\\ x_3 &\le 1\end{aligned}$$

$$x_1 \ge 0,\ x_2 \ge 0,\ x_3 \ge 0,\ x_4 \ge 0$$

Transformacijom ovog problema na normalni oblik, dobijamo problem

$$\arg\max -Z = 3x_1 - 80x_2 + 2x_3 - 24x_4$$

p.o.

$$\begin{aligned}(1/4)x_1 - 8x_2 - x_3 + 9x_4 + x_5 &= 0\\ (1/2)x_1 - 12x_2 - (1/2)x_3 + 3x_4 + x_6 &= 0\\ x_3 + x_7 &= 1\end{aligned}$$

$$x_1 \ge 0,\ x_2 \ge 0,\ x_3 \ge 0,\ x_4 \ge 0,\ x_5 \ge 0,\ x_6 \ge 0,\ x_7 \ge 0$$

Ovom problemu odgovara sljedeća početna simpleks tabela:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_5	0	1/4	–8	–1	9	1	0	0	0/(1/4) = 0
x_6	0	1/2	–12	–1/2	3	0	1	0	0/(1/2) = 0
x_7	1	0	0	1	0	0	0	1	
	0	3	–80	2	–24	0	0	0	

Ovdje imamo početnu bazu (x_5, x_6, x_7) i početno bazno rješenje $\mathbf{x} = (0, 0, 0, 0, 0, 0, 1)^T$. Vidimo da je već početno bazno rješenje *degenerirano*. Probajmo primijeniti simpleks algoritam. Nemamo nikakve dileme oko toga da u bazu treba ući promjenljiva x_1 (prema *Dantzigovom pravilu pivotiranja*), ali oko izbora da li bazu treba napustiti x_5 ili x_6 javlja se dilema, jer se vrijednost $t_{max} = 0$ postiže u dva razlčita reda tabele. Odlučimo li se *nasumice* da bazu napusti x_5 dobijamo sljedeću simpleks tabelu:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_1	0	1	–32	–4	36	4	0	0	
x_6	0	0	4	3/2	–15	–2	1	0	0/4 = 0
x_7	1	0	0	1	0	0	0	1	
	0	0	16	14	–132	–12	0	0	

Primijetimo da se vrijednost funkcije cilja nije promijenila, nego je ostala 0. Ovdje sad nema nikakve dileme da u bazu treba ući promjenljiva x_2 a iz baze treba izaći promjenljiva x_6, tako da dolazimo do sljedeće simpleks tabele:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_1	0	1	0	8	–84	–12	8	0	0/8 = 0
x_2	0	0	1	3/8	–15/4	–1/2	1/4	0	0/(3/8) = 0
x_7	1	0	0	1	0	0	0	1	1/1 = 1
	0	0	0	8	–72	–4	–4	0	

Sada u bazu treba ući promjenljiva x_3, dok se po pitanju izlaska iz baze ponovo javlja dilema između promjenljivih x_1 i x_2. Odlučimo li (ponovo nasumice) da bazu napusti x_1 dobijamo sljedeću simpleks tabelu:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_3	0	1/8	0	1	–21/2	–3/2	1	0	
x_2	0	–3/64	1	0	3/16	1/16	–1/8	0	0/(3/16) = 0
x_7	1	–1/8	0	0	21/2	3/2	–1	1	1/(21/2) ≈ 0.095
	0	–1	0	0	12	8	–12	0	

Funkcija cilja i dalje stoji na početnoj vrijednosti 0. Sada u bazu treba ući promjenljiva x_4, a bazu (nedvosmisleno) treba napustiti promjenljiva x_2, čime dobijamo sljedeću simpleks tabelu:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_3	0	–5/2	56	1	0	2	–6	0	0/2 = 0
x_4	0	–1/4	16/3	0	1	1/3	–2/3	0	0/(1/3) = 0
x_7	1	5/2	–56	0	0	–2	6	1	
	0	2	–64	0	0	4	–4	0	

Još uvijek nije došlo ni do kakvog pomaka u funkciji cilja. U bazu sada ulazi x_5, a dilema se javlja između x_3 i x_4 po pitanju napuštanja baze. Odlučimo li se za x_3, dobijamo sljedeću simpleks tabelu:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_5	0	–5/4	28	1/2	0	1	–3	0	
x_4	0	1/6	–4	–1/6	1	0	1/3	0	0/(1/3) = 0
x_7	1	0	0	1	0	0	0	1	
	0	7	–176	–2	0	0	8	0	

U bazu sada ulazi promjenljiva x_6, dok bazu nedvosmisleno napušta x_4, čime se dobija sljedeća simpleks tabela:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_5	0	1/4	–8	–1	9	1	0	0
x_6	0	1/2	–12	–1/2	3	0	1	0
x_7	1	0	0	1	0	0	0	1
	0	3	–80	2	–24	0	0	0

Ako pažljivije pogledamo, vidjećemo *da je ovo ista simpleks tabela od koje smo krenuli*! Dakle, vrtili smo se u "krug", i nakon "lutanja" po skupini vršnih tačaka koje su se poklopile, nakon 6 iteracija vratili smo se u istu tačku iz koje smo krenuli. Jasno je, da ako ovako nastavimo dalje, *nećemo stići nikuda*. Prirodno se postavlja pitanje *ima li uopće izlaza iz ovog začaranog kruga*? Izlaza *mora biti* (jer i iz degeneriranih vršnih tačaka, pored gomile "fiktivnih" bridova dužine 0 koje spajaju međusobno "stopljene" vršne tačke, postoje bridovi koji vode ka stvarno različitim vršnim tačkama), samo što ih mi nismo uspjeli pronaći. Međutim, sasvim bi drugačija situacija bila da smo, u slučajevima kada smo imali dilemu koju promjenljivu izbaciti iz baze, pravili *drugačiji izbor* nego što smo to radili u prethodnom postupku. Recimo, da smo se u početnoj simpleks tabeli odlučili da bazu napusti ne x_5 nego x_6, dobili bismo sljedeću simpleks tabelu:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_5	0	0	–2	–3/4	15/2	1	–1/2	0	
x_1	0	1	–24	–1	6	0	2	0	
x_7	1	0	0	1	0	0	0	1	1/1 = 1
	0	0	–8	5	–42	0	–6	0	

Funkcija cilja je i dalje nula, i ovo bazno rješenje je također degenerirano, ali ovdje barem imamo da t_{max} *nije nula*, što znači da ćemo u sljedećoj iteraciji *napokon pobjeći iz degeneriranog vrha*. Zaista, nakon što u bazu uđe promjenljiva x_3 a iz nje izađe promjenljiva x_7, dolazimo do sljedeće simpleks tabele, u kojoj dopustivo bazno rješenje *više nije degenerirano*:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_5	3/4	0	–2	0	15/2	1	–1/2	3/4
x_1	1	1	–24	0	6	0	2	1
x_3	1	0	0	1	0	0	0	1
	–5	0	–8	0	–42	0	–6	–5

Pored toga, kako su u ovoj tabeli nema niti jedan pozitivan koeficijent c_j u posljednjem redu, to je nađeno rješenje *ujedno i optimalno*. Dakle, optimalno rješenje problema je $\mathbf{x} = (1, 0, 1, 0, 3/4, 0, 0)^T$, dok je optimalna vrijednost funkcije cilja $Z = 5$. Do istog optimalnog rješenja smo mogli doći i da smo u prvoj

simpleks tabeli izabrali da bazu ipak napušta x_5 a ne x_6, ali da smo nakon sljedeće dvije iteracije kada smo imali dilemu da li da bazu napusti x_1 ili x_2 izabrali x_2 umjesto x_1.

Kruženje kakvo je gore opisano *u praksi se jako rijetko dešava*. Mnogo češća pojava u slučaju degeneracije je pojava tzv. ***odugovlačenja*** ili ***zastoja*** (engl. *stalling*) koja nastaje kada se algoritam jako dugo zadržava u degeneriranim vršnim tačkama, odnosno provodi mnogo iteracija prije nego što uspije da napusti degenerirani vrh. Zaista, ukoliko imamo degenerirani vrh kojem odgovara degenerirano bazno rješenje u kojem je k promjenljivih jednako nuli (k je tzv. ***stepen degeneracije***), može se pokazati da tada može ukupno postojati do $(n+k)!/(n!\,k!)$ različitih degeneriranih baznih rješenja koje zapravo odgovaraju tom istom degeneriranom vrhu, što za veće vrijednosti n i k može biti veoma veliki broj (već u prethodnom primjeru za $n = 4$ i $k = 2$ imamo 15 degeneriranih baznih rješenja koje odgovaraju jednom te istom degeneriranom vrhu). Nažalost, problem sprečavanja zastoja do danas još nije adekvatno riješen. Naročito se često zastoji javljaju u slučajevima kada se neki *grafovski problemi* modeliraju i rješavaju kao problemi linearnog programiranja, jer se pokazuje da se u takvim problemima često javljaju degenerirana rješenja sa vrlo velikim stepenima degeneracije. Srećom, za rješavanje takvih problema uglavnom postoje *specijalizirani algoritmi* kod kojih se ne javljaju velike poteškoće sa degeneracijom.

Bez obzira na činjenicu da do danas ne postoje pouzdane strategije kojima se mogu izbjeći zastoji, *postoje brojne strategije* kojima se može izbjeći "kruženje", odnosno koje garantiraju da će simpleks algoritam uvijek terminirati (mada nemali broj programera prosto *ignorira* opasnost od kruženja, računajući da je vjerovatnoća takve pojave dovoljno mala da je gotovo izvjesno da do nje neće doći). Najpoznatija takva strategija je ***Blant-ovo pravilo*** po kojem do kruženja ne može doći ukoliko u slučaju dvosmislice *uvijek izbacujemo onu promjenljivu sa manjim indeksom* (*rednim brojem*), ali samo ukoliko pored toga odabir promjenljive koja ulazi u bazu ne vršimo po *Dantzigovom pravilu pivotiranja*, nego uvijek biramo da u bazu ulazi također promjenljiva sa najmanjim indeksom između svih onih koje teoretski smiju ući u bazu (tj. kod kojih je $c_j > 0$). Nažalost, Blant-ovo pravilo obično *osjetno povećava broj iteracija* koje su potrebne da bi se dostiglo optimalno rješenje. *Dantzig*, *Orden* i *Wolfe* (1955) su predložili ***leksikografsko pravilo*** koje također pouzdano sprečava kruženje. Ovo pravilo zahtijeva da se u slučaju da ima jednakih količnika $b_i/a_{i,q}$, računaju i odgovarajući količnici $a_{i,j}/a_{i,q}$ za razne vrijednosti j počev od 1 nadalje, i da se onda konačna odluka o tome koja promjenljiva napušta bazu donosi na osnovu *leksikografskog poređenja* vektora obrazovanih od takvih količnika. U detalje ovog pravila se nećemo upuštati, jer u praksi sasvim dobro radi i mnogo jednostavnije ***pravilo slučajnog izbora***, po kojem jednostavno u slučaju da imamo dileme koja promjenljiva treba napustiti bazu, izbor vršimo *nasumice* (pri čemu *ne smijemo vršiti uvijek isti izbor*, odnosno ne smijemo se uvijek odlučivati recimo za prvu promjenljivu po redu). Na taj način, može se desiti da napravimo pokoji "krug", ali ako izbor vršimo nasumice, prije ili kasnije ćemo "nabasati" na tačku iz koje postoje putevi koji vode van "začaranog kruga". Softverski se ovo rješenje vrlo jednostavno implementira ukoliko se uvjet $a_{i,0}/a_{i,q} < tmax$ u dijelu kôda koji računa $tmax$ zamijeni sa nešto složenijim uvjetom

$$a_{i,0}/a_{i,q} < tmax \vee (a_{i,0}/a_{i,q} = tmax \wedge RandomReal(0\,..\,1) > 0.5)$$

U ovom uvjetu, ukoliko u nekom redu naiđemo na količnik koji je već jednak ranijoj vrijednosti t_{max}, sa vjerovatnoćom od 50 % pamtimo taj red kao onaj koji odgovara vrijednosti t_{max}, dok u suprotnom zadržavamo postojeće mišljenje. Na taj način se uvodi element "slučajnosti" u odabir vodećeg reda.

Postoji još jedan problem vezan za degeneraciju, o kojem se u literaturi *veoma malo govori*, a na kojem ponekad "padaju" i *komercijalni programi* za rješavanje problema matematičkog programiranja (*R. Fletcher* navodi da je u svojoj praksi tri puta naišao upravo na takav slučaj pri rješavanju praktičnih problema). Ovaj problem se tipično javlja kod rješavanja *vrlo velikih problema*, a vezan je za *greške zaokruživanja*. Naime, već znamo da je za simpleks algoritam od velike važnosti računanje količnika $b_i/a_{i,q}$. Ako je $a_{i,q} = 0$, ovaj količnik se, kao što znamo, ne računa, uključujući i slučaj kada je $b_i = 0$ (što se dešava u degeneriranim rješenjima). Međutim, problem može nastati ukoliko se desi da usljed grešaka zaokruživanja, nakon izvjesnog broja iteracija neka vrijednost $a_{i,q}$ koja bi trebala biti jednaka nuli ne bude jednaka nuli, nego nekoj vrlo maloj pozitivnoj vrijednosti. Ako je pri tome $b_i > 0$, neće biti problema, jer će količnik $b_i/a_{i,q}$ tada biti vrlo veliki broj, tako da svakako neće biti kandidat za računanje t_{max}. Međutim, ukoliko je $b_i = 0$, a zbog grešaka u zaokruživanju b_i ne bude izračunat kao 0 nego također kao neki veoma mali broj, tada će se pri računanju količnika $b_i/a_{i,q}$ javiti *dijeljenje dva vrlo mala broja*, koje može kao rezultat dati praktično *ma makav broj*, što može ugroziti računanje t_{max} (ukoliko taj broj ispadne manji od *stvarne vrijednosti* koju t_{max} treba imati). Dogodi li se tako nešto, sljedeća iteracija će nas odvesti u tačku koja *uopće nije vršna tačka*, nakon čega će algoritam potpuno "poluditi" (najčešće će se dogoditi da će se pojaviti krah usljed dijeljenja nulom na mjestu gdje se tako nešto u normalnim okolnostima nikada ne bi smjelo dogoditi). *Fletcher* (1985) je našao način kako se i ovaj problem može prevazići, ali o tome ovdje ne možemo govoriti.

Efikasnost simpleks algoritma

Razmotrimo sada i pitanje *koliko je simpleks algoritam zaista efikasan*. Prvo razmotrimo šta se dešava *u toku jedne iteracije simpleks algoritma*. Kako je format simpleks tabele očigledno $(m+1)\times(m+n+1)$, to znači da klasični simpleks algoritam u svakoj iteraciji ažurira upravo toliko elemenata, odnosno otprilike $m(m+n)$ ukoliko zanemarimo jedinice koje svakako ne dolaze do izražaja pri većim vrijednostima m i n. To ujedno znači i da je broj aritmetičkih operacija po jednoj iteraciji otprilike proporcionalan sa $m(m+n)$. Međutim, kako smo ranije napomenuli, uz inteligentno kodiranje, m kolona simpleks tabele (koje odgovaraju baznim promjenljivim) se niti moraju čuvati u memoriji niti preračunavati, jer svakako *znamo kako će izgledati*. To broj aritmetičkih operacija smanjuje na nivo otprilike proporcionalan sa $m\,n$. Umjesto toga, *ozbiljnije softverske implementacije* simpleks algoritma tipično koriste tzv. ***revidirani simpleks metod***, kod kojeg je broj elemenata koje treba čuvati i preračunavati najviše oko m^2 (tačnije $m(m+1)$), a moguće i manje, ukoliko matrica problema koji se rješava *ima mnogo nula* (naime, za razliku od klasičnog simpleks metoda, revidirani simpleks metod može efikasno da *izbjegne razmatranje elemenata koji su nule*). To može da dovede do velike uštede u trajanju računanja u većim problemima, pogotovo ako je n mnogo veći od m ili ako matrica problema koji se rješava ima mnogo nula. Pored toga, revidirani simpleks metod *mnogo manje akumulira greške zaokruživanja*, te je i zbog toga pogodniji za veće probleme. Ovdje se nećemo upuštati u detalje revidiranog simpleks metoda. Ipak, treba napomenuti da je revidirani simpleks metod *uglavnom namijenjen za programiranje*, te je *znatno manje pogodan za ručni rad* od klasičnog simpleks metoda.

Računanje koliko vremenski troši jedna iteracija je lakši dio posla. Znatno teže je procijeniti *koliko zapravo iteracija* simpleks algoritmu treba da se dostigne optimalno rješenje, jer od toga suštinski zavisi trajanje izvršavanja simpleks algoritma (bilo običnog, bilo revidiranog). Razni empirijski testovi na velikom broju praktičnih problema pokazali su da broj iteracija gotovo nikada ne prelazi neki mali umnožak od m ili n, tako da su davane razne procjene da broj iteracija gotovo nikada ne premašuje iznos od, recimo, $3\max\{m,n\}$ ili $2(m+n)$, tako da je, u tom slučaju, ukupan broj operacija u simpleks algoritmu ograničen nekim manjim umnoškom od n^3 odnosno m^3. Dugo vremena za takvu empirijsku procjenu nije bilo nikakvih egzaktnih dokaza. Tek je *Smale* (1982) uspio dokazati da za *izvjesnu* klasu uvjetno rečeno "pravilnih" problema ova gornja granica zaista i vrijedi. Međutim, s obzirom na činjenicu da broj vrhova dopustive oblasti može *eksponencijalno rasti* sa porastom m odnosno n odavno je izazivao strah da bi u najgorem slučaju simpleks algoritam zaista mogao zahtijevati *eksponencijalno mnogo iteracija* (izraženo u funkciji od m i n). Da ovi strahovi nisu bezrazložni, pokazali su *Klee* i *Minty* (1971). Oni su naime pokazali da ukoliko dopustiva oblast ima oblik jedne vrste *blago deformirane n-dimenzionalne kocke* (tačnije, *kvadra*) poznate pod imenom ***Klee–Minty-jeva kocka***, postoje funkcije cilja koje će na takvoj dopustivoj oblasti tražiti *eksponencijalno mnogo iteracija*. Njihovu tvdnju je nešto kasnije iskoristio *Chvatal* da eksplicitno konstruira primjere takvih problema. Tako, poznati ***Klee–Minty–Chvatal-ov*** (***KMC***) ***primjer*** sa n promjenljivih i n ograničenja glasi

$$\arg\max Z = \sum_{j=1}^{n} 10^{j-1} x_j$$

p.o.

$$x_i + \sum_{j=i+1}^{n} 2\cdot 10^{j-i} x_j \le 100^{n-i},\ \ i = 1\,..\,n$$

$$x_j \ge 0,\ \ j = 1\,..\,n$$

Eksplicitno, recimo za $n = 5$, ovaj problem raspisano izgleda ovako (primijetimo usput da *realni problemi* gotovo nikada nemaju *ovakvo rasipanje veličine koeficijenata* kakvo se ovdje javlja):

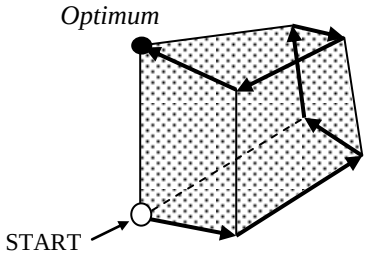
$$\arg\max Z = x_1 + 10\,x_2 + 100\,x_3 + 1000\,x_4 + 10000\,x_5$$

p.o.

$$\begin{aligned}
x_1 + 20\,x_2 + 200\,x_3 + 2000\,x_4 + 20000\,x_5 &\le 100000000\\
x_2 + 20\,x_3 + 200\,x_4 + 2000\,x_5 &\le 1000000\\
x_3 + 20\,x_4 + 200\,x_5 &\le 10000\\
x_4 + 20\,x_5 &\le 100\\
x_5 &\le 1
\end{aligned}$$

$$x_1 \ge 0,\ \ x_2 \ge 0,\ \ x_3 \ge 0,\ \ x_4 \ge 0,\ \ x_5 \ge 0$$

KMC primjer ima dopustivu oblast koja ima *tačno* 2^n *vršnih tačaka*, što samo po sebi nije nimalo neobično. Međutim, pođemo li od dopustivog baznog rješenja koje slijedi iz normalnog oblika problema u kojem početnu bazu čine dopunske promjenljive, simpleks algoritam uz Dantzigovo pravilo pivotiranja će obići *apsolutno sve vršne tačke* prije nego što stigne do tačke u kojoj se postiže optimum, odnosno tražiće *tačno* $2^n - 1$ *iteracija* (dakle, 31 iteraciju u gornjem primjeru za $n = 5$)! Sljedeća slika vizuelno ilustrira šta se ovdje dešava za $n = 3$.



Treba istaći da je KMC primjer eksplicitno konstruisan uz pretpostavku da će se koristiti *Dantzigovo pravilo pivotiranja*. Međutim, isto tako je jasno da Dantzigovo pravilo *nije nužno i najefikasnije moguće pravilo pivotiranja*. Zaista, vidjeli smo da porast funkcije cilja iz iteracije u iteraciju iznosi $c_q\, t_{max}$. Dantzigovo pravilo je birano tako da maksimizira c_q, ali to *neće nužno maksimizirati porast funkcije cilja*. Zaista, pri nekom drugom izboru promjenljive koja ulazi u bazu, moglo bi se dogoditi da vrijednost c_q bude *manja*, ali da se pri tom izboru promjenljive koja ulazi u bazu dobije *veća vrijednost* t_{max}, tako da ukupno produkt $c_q\, t_{max}$ bude također veći. Stoga je predloženo i pravilo pivotiranja poznato kao ***pravilo najvećeg prirasta funkcije cilja***, po kojem se kao kandidati za ulazak u bazu razmatraju *sve nebazne promjenljive* za koje je $c_j > 0$, za *svaki od potencijalnih kandidata* se računaju vrijednosti $t_{max}(j)$ (koje sad ovise od izbora j), te se na kraju kao konačni kandidat za ulazak u bazu uzima ona promjenljiva koja *maksimizira produkt* $c_j\, t_{max}$. Geometrijski, ovo znači da ispitujemo *sve susjede* tekuće vršne tačke u kojima je vrijednost funkcije cilja bolja u odnosu na tekuću, nakon čega prelazimo u *onu vršnu tačku u kojoj je funkcija cilja najbolja*.

- ***Primjer*** : Riješiti ponovo problem linearnog programiranja

$$\arg\max Z = 800\,x_1 + 1000\,x_2$$

p. o.

$$\begin{aligned} 30\,x_1 + 16\,x_2 &\le 22800 \\ 14\,x_1 + 19\,x_2 &\le 14100 \\ 11\,x_1 + 26\,x_2 &\le 15950 \\ x_2 &\le 550 \end{aligned}$$

$$x_1 \ge 0,\ x_2 \ge 0$$

koji smo već ranije razmatrali, ali ovaj put koristeći pravilo najvećeg prirasta funkcije cilja umjesto Dantzigovog pravila kao pravila pivotiranja.

Krećemo od iste početne simpleks tabele koju smo već imali (nakon svođenja problema na normalni oblik). Vidimo da imamo dva potencijalna kandidata za ulazak u bazu: x_1 i x_2. Prema Dantzigovom pravilu, odmah bi se odlučili za x_2, jer je $1000 > 800$. Međutim, po pravilu najvećeg prirasta funkcije cilja, potrebno je izračunati t_{max} za *obje opcije*. Pretpostavimo li da u bazu ulazi x_1, dobijamo $t_{max} = 760$, dok uz pretpostavku da u bazu ulazi x_2 dobijamo $t_{max} = 550$. Pomoćni računi za određivanje t_{max} u oba slučaja prikazani su desno od tabele.

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6		
x_3	22800	30	16	1	0	0	0	$22800/30 = 760$	$22800/16 = 1425$
x_4	14100	14	19	0	1	0	0	$14100/14 \approx 1007$	$14100/19 \approx 742$
x_5	15950	11	26	0	0	1	0	$15950/11 \approx 1381$	$15950/26 \approx 613$
x_6	550	0	1	0	0	0	1		$550/1 = 550$
	0	800	1000	0	0	0	0		

Kako je $800 \cdot 760 = 608000$, $1000 \cdot 550 = 550000$ i $608000 > 550000$, vidimo da po pravilu najvećeg prirasta funkcije cilja u bazu treba da uđe x_1 a ne x_2, tako da iz baze ispada x_3 (a ne x_6). Nakon prelaska u novo bazno rješenje, dobijamo sljedeću simpleks tabelu:

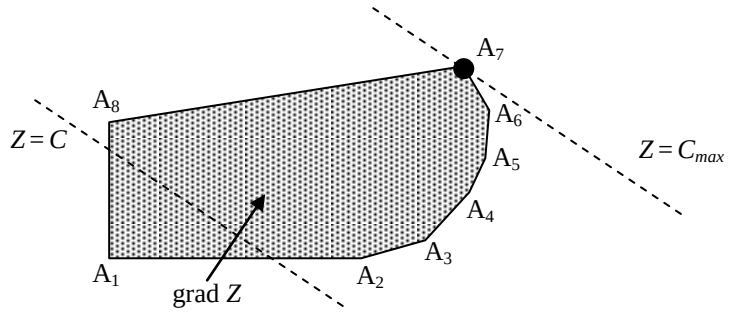
Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	
x_1	760	1	8/15	1/30	0	0	0	$760/(8/15) = 1425$
x_4	3460	0	173/15	–7/15	1	0	0	$3640/(173/15) = 300$
x_5	7590	0	302/15	–11/30	0	1	0	$7590/(302/15) \approx 377$
x_6	550	0	1	0	0	0	1	$550/1 = 550$
	–608000	0	1720/3	–80/3	0	0	0	

Sada je jedini kandidat za ulazak u bazu x_2, tako da ma koje pravilo pivotiranja daje da ona ulazi u bazu. Pri tome iz baze ispada promjenljiva x_4, nakon čega se dobija sljedeća simpleks tabela:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6
x_1	1550	0	0	155/346	–302/173	1	0
x_2	250	0	0	7/173	–15/173	0	1
x_5	600	1	0	19/346	–8/173	0	0
x_6	300	0	1	–7/173	15/173	0	0
	–780000	0	0	–600/173	–8600/173	0	0

Kako više nema pozitivnih elemenata, dostignuto je optimalno rješenje. Vidimo da smo došli do istog optimalnog rješenja kao i korištenjem Dantzigovog pravila pivotiranja, ali smo do njega došli nakon samo dvije iteracije, dok je korištenje Dantzigovog pravila zahtijevalo četiri iteracije.

Postavlja se pitanje *da li pravilo najvećeg prirasta funkcije cilja uvijek dovodi brže do optimalnog rješenja nego* Dantzigovo pravilo. Odgovor je odrečan. Mada je činjenicy da za većinu malih (školskih) problema ovaj pristup tipično brže vodi do rješenja, to ne mora biti uvijek slučaj. Zaista, pretpostavimo da imamo problem (sa dvije promjenljive) za koji dopustiva oblast i pravci funkcije cilja izgledaju kao na sljedećoj slici:



Ako krenemo iz tačke A_1, pravilo najvećeg prirasta funkcije cilja izabraće kao sljedeću vršnu tačku A_2, jer je u njoj funkcija cilja veća nego u tački A_8, ali će nakon toga put do optimalnog rješenja A_7 voditi kroz tačke A_3, A_4, A_5 i A_6. S druge strane, Dantzigovo pravilo će umjesto tačke A_2 izabrati tačku A_8 (jer je grad Z takav da je $c_2 > c_1$), tako da će se optimalno rješenje dostići već u sljedećoj iteraciji. Drugim riječima, *Dantzigovo pravilo je za ovaj slučaj efikasnije*. Problem sa pravilom najvećeg prirasta funkcije cilja je što ono djeluje "*pohlepno*", pokušavajući da ostvari *što bolji efekat u narednoj iteraciji, ne misleći pri tome šta će biti kasnije*. Ova "kratkovidnost" je karakteristična za gotovo sva predložena pravila pivotiranja i mnoga istraživanja se preduzimaju sa ciljem da se ta "kratkovidnost" pokuša prevazići. Generalno, danas se smatra da ušteda koja se u praksi ponekad dobija upotrebom pravila najvećeg prirasta funkcije cilja nisu dovoljna da opravdaju dodatna izračunavanja koja zahtijeva ovo pravilo, tako da se u praksi ipak mnogo češće koristi Dantzigovo pravilo.

Interesantno je da se, pomoću pravila najvećeg prirasta funkcije cilja, KMC primjer *rješava u jednoj iteraciji*. Međutim, i za ovo pravilo su nađeni primjeri za koje simpleks algoritam uz upotrebu ovog pravila zahtijeva eksponencijalno mnogo iteracija. Zapravo, *za svako do sada predloženo pravilo pivotiranja* su konstruisani primjeri čije rješavanje zahtijeva eksponencijalno mnogo iteracija uz upotrebu tog pravila. Nedavno su nađena neka *randomizirana* pravila pivotiranja (uz upotrebu slučajnih brojeva) za koja se ne

može desiti da broj iteracija bude eksponencijalna funkcija veličine problema, ali pitanje *da li postoji deterministično* (tj. bez upotrebe slučajnih brojeva) *pravilo pivotiranja koje ne vodi ka eksponencijalnom vremenu izvršavanja u najgorem slučaju još uvijek je otvoreno*. Ako je tzv. ***Hirsch-ova hipoteza*** tačna, odgovor na ovo pitanje trebao bi biti potvrdan, ali takvo pravilo još uvijek nije nađeno.

Vještačka početna baza i metod "velikog *M*"

Sve do sada, pretpostavljali smo da su svi koeficijenti b_i, $i = 1 .. m$ u standardnom modelu *nenegativni*, kao i da u problemu *nemamo ograničenja tipa jednakosti*. Vidjeli smo da, ukoliko to nije ispunjeno, *ne možemo lako formirati početno dozvoljeno bazno rješenje*. Neko bi mogao pomisliti da možemo *nasumice* probati izabrati m promjenljivih za bazne, izraziti ih preko preostalih promjenljivih te postaviti preostale promjenljive na nulu, *u nadi da ćemo tako dobiti dozvoljeno bazno rješenje*. Ponekada to i uspije, ali je u općem slučaju potrebno isprobati *neprihvatljivo mnogo kombinacija* prije nego što se zaista dođe do dozvoljenog baznog rješenja. Druga ideja mogla bi biti da započnemo izvršavanje simpleks algoritma počev od *nedozvoljenog baznog rješenja*, te da vršimo iteracije u nadi da ćemo u nekom trenutku "upasti" u dozvoljenu oblast, odnosno doći do nekog *dozvoljenog baznog rješenja*. Mada postoje i neki pristupi u kojima se zaista tako radi, osnovni je problem što kada smo izvan dozvoljene oblasti *nemamo nikakve vodilje koja bi nas upućivala koje promjenljive treba ubacivati u bazu a koje izbacivati da bismo došli do dozvoljenog baznog rješenja*. Stoga je potrebno razviti neki pouzdaniji pristup za rješavanje ovog problema.

Jedan od najuspješnijih koncepata za rješavanje ovog problema je tzv. koncept ***vještačkih promjenljivih*** i ***vještačke baze***. Primijetimo da ukoliko imamo ograničenje tipa "manje ili jednako" u kojem je desna strana *negativna*, množenjem sa –1 možemo ga svesti na ograničenje tipa "veće ili jednako" u kojem je desna strana pozitivna. Stoga, možemo bez umanjenja općenitosti pretpostaviti da naši problemi sa početnim dozvoljenim baznim rješenjem zapravo *potiču od ograničenja tipa* "*veće ili jednako*" (sa pozitivnom desnom stranom). Uzmimo stoga da imamo neko takvo ograničenje, poput

$$a_{i,1} x_1 + a_{i,2} x_2 + ... + a_{i,n} x_n \geq b_i$$

pri čemu je $b_i > 0$ (ako to nije slučaj, množenjem sa –1 dobijamo ograničenje tipa "manje ili jednako" sa pozitivnom desnom stranom, koje nam ne pravi probleme). Ovo ograničenje se također lako svodi na ograničenje tipa jednakosti uvođenjem izravnavajuće promjenljive, ali ovaj put ne njenim *dodavanjem* nego *oduzimanjem* sa lijeve strane (pri čemu ta izravnavajuća promjenljiva također mora zadovoljavati uvjet nenegativnosti). Dakle, ovo ograničenje možemo transformirati u oblik

$$a_{i,1} x_1 + a_{i,2} x_2 + ... + a_{i,n} x_n - x_{n+i} = b_i$$

Izravnavajuća promjenljiva x_{n+i} u ovom slučaju nema značenje "preostalih resursa", nego "viška u odnosu na zahtjeve". Međutim, problem je u tome što ovakvu izravnavajuću promjenljivu *ne možemo iskoristiti za formiranje baze*. Zaista, ukoliko bi, kao i inače, promjenljive x_j, $j = 1 .. n$ postavili na nulu a promjenljivu x_{n+i} proglasili za baznu, dobili bismo $x_{n+i} = -b_i < 0$, što *nije dozvoljeno*. Da bismo riješili ovaj problem, dodaćemo na lijevu stranu ove jednačine *još jednu nenegativnu promjenljivu* x^*_{n+i}. Drugim riječima, transformiraćemo ovu jednakost u oblik

$$a_{i,1} x_1 + a_{i,2} x_2 + ... + a_{i,n} x_n - x_{n+i} + x^*_{n+i} = b_i$$

Sada ćemo upravo tu promjenljivu x^*_{n+i} uzeti *da bude bazna promjenljiva*, tako da će u početnom baznom rješenju biti $x^*_{n+i} = b_i$. Ova promjenljiva naziva se ***vještačka promjenljiva***, s obzirom da ona *nije dio originalnog zadatka* i da je dodana samo da bismo *formalno dobili promjenljivu koja može biti bazna promjenljiva*. Pored toga, ova promjenljiva *nema nikakvo realno značenje*. Međutim, nije sve tako jednostavno. Ne možemo *tek tako dodavati promjenljive* u jednačine i nadati se *da će one ostati zadovoljene*. Zaista, ovako modificirana jednačina ima isti smisao kao i prethodna *jedino ako vrijedi* $x^*_{n+i} = 0$, a mi baš želimo da uzmemo da x^*_{n+i} nije nula. Zbog toga nam je potreban *neki mehanizam koji će obezbijediti da će u daljem toku postupka sve uvedene vještačke promjenljive biti oborene na nulu* (u svakom slučaju, prije dostizanja optimalnog rješenja).

Postoji više pristupa kojima se može osigurati obaranje vještačkih promjenljivih na nulu, a najpoznatiji je tzv. ***metod "velikog M"***, poznat i pod imenom ***metod kaznenih koeficijenata***. Po ovom metodu, funkcija cilja se modificira tako da se u nju ubace i vještačke promjenljive, kojima se da *vrlo nepovoljna cijena*, $-M$ za slučaj maksimizacije odnosno $+M$ za slučaj minimizacije, pri čemu se pretpostavlja da je M neki *vrlo veliki broj*. Na taj način, svaka nenulta vrijednost vještačke promjenljive djeluje *vrlo nepovoljno* na

funkciju cilja, te će se algoritam truditi da *što prije anulira takve promjenljive*, odnosno da ih *izbaci iz baze*. Drugim riječima, uvođenje vještačke promjenljive *narušava* odgovarajuće ograničenje, pa se izbacivanje vještačke promjenljive iz baze osigurava uvođenjem odgovarajućeg *nepovoljnog koeficijenta* (*–M* ili *+M*) u funkciji cilja odgovarajućeg znaka. Ovaj koeficijent naziva se i ***kazneni koeficijent***, jer on, *radikalno umanjujući* (*ili uvećavajući*) *funkciju cilja*, *kažnjava nedopustivost* baznih rješenja transformiranog zadatka u odnosu na originalni. Sve uvedene vještačke promjenljive *moraju se anulirati* prije nego što se algoritam završi. Desi li se slučajno da se simpleks algoritam završi a da se barem jedna vještačka promjenljiva nije anulirala, to je indikacija da su *početna ograničenja bila kontradiktorna*, odnosno da problem *nije dopustiv* (odnosno da *nema dozvoljenih rješenja*).

U slučaju da u polaznom problemu postoje ograničenja tipa jednakosti, jedna opcija je da se takvih ograničenja *oslobodimo*, što dolazi u obzir samo ako je takvih ograničenja malo. Univerzalniji pristup je da i ovdje koristimo pristup sa vještačkim promjenljivim. Naime, ograničenje tipa jednakosti

$$a_{i,1}\,x_1 + a_{i,2}\,x_2 + \ldots + a_{i,n}\,x_n = b_i$$

u kojem nemamo očigledan izbor za baznu promjenljivu, transformiraćemo u ograničenje

$$a_{i,1}\,x_1 + a_{i,2}\,x_2 + \ldots + a_{i,n}\,x_n + x^*_{n+i} = b_i$$

u kojem se javlja vještačka promjenljiva x^*_{n+i}, koju ćemo uzeti za baznu promjenljivu (primijetimo da u ovom slučaju *nismo uvodili dopunsku, nego samo vještačku promjenljivu*). Naravno, kako i u ovom slučaju vještačka promjenljiva x^*_{n+i} mora da se anulira prije dostizanja optimuma, *uvešćemo odgovarajući kazneni koeficijent* u funkciju cilja.

U svakom slučaju, nakon uvođenja vještačkih promjenljivih u sva ograničenja tipa "veće ili jednako" i ograničenja tipa jednakosti, funkcija cilja će imati oblik

$$Z = c_1\,x_1 + c_2\,x_2 + \ldots + c_n\,x_n - M\,x^*_{n+i_1} - M\,x^*_{n+i_2} - \ldots$$

ako je u pitanju maksimizacija, odnosno

$$Z = c_1\,x_1 + c_2\,x_2 + \ldots + c_n\,x_n + M\,x^*_{n+\,i_1} + M\,x^*_{n+\,i_2} + \ldots$$

ako je u pitanju minimizacija, pri čemu su i_1, i_2, itd. *indeksi redova u koje smo uveli vještačke promjenljive*. Nažalost, ovakav oblik funkcije cilja *nije direktno pogodan za primjenu simpleks algoritma*. Zaista, rekli smo da želimo da nam *upravo vještačke promjenljive budu u početnoj bazi*, a znamo da kod simpleks algoritma bazne promjenljive *nikada ne smiju figurirati u funkciji cilja*. Stoga je neophodno *transformirati funkciju cilja* da iz nje *nestanu vještačke promjenljive*. Ovo nije teško uraditi. Za tu svrhu, iz svakog ograničenja *izrazimo vještačku promjenljivu preko ostalih*. Ako je odgovarajuće polazno ograničenje bilo tipa "veće ili jednako", transformirano ograničenje ima oblik

$$a_{i,1}\,x_1 + a_{i,2}\,x_2 + \ldots + a_{i,n}\,x_n - x_{n+i} + x^*_{n+i} = b_i$$

odakle slijedi

$$x^*_{n+i} = b_i - a_{i,1}\,x_1 - a_{i,2}\,x_2 - \ldots - a_{i,n}\,x_n + x_{n+i}$$

Ako je polazno ograničenje bilo tipa jednakosti, dobijamo sličan izraz, samo bez izravnavajuće (dopunske) promjenljive. Stoga, za slučaj maksimizacije, funkciju cilja možemo transformirati na sljedeći način:

$$\begin{aligned}
Z &= c_1\,x_1 + c_2\,x_2 + \ldots + c_n\,x_n - M\,x^*_{n+i1} - M\,x^*_{n+i2} - \ldots = \\
&= c_1\,x_1 + c_2\,x_2 + \ldots + c_n\,x_n - M\,(b_{i1} - a_{i1,1}\,x_1 - a_{i1,2}\,x_2 - \ldots - a_{i\,1,n}\,x_n + x_{n+i1}) \\
&\quad - M\,(b_{i2} - a_{i2,1}\,x_1 - a_{i2,2}\,x_2 - \ldots - a_{i2,n}\,x_n + x_{n+i2}) - \ldots = \\
&= (c_1 + M\sum a_{i,1})\,x_1 + (c_2 + M\sum a_{i,2})\,x_2 + \ldots + (c_n + M\sum a_{i,n})\,x_n + \\
&\quad - M\,x_{n+i1} - M\,x_{n+i2} - \ldots - M\sum b_i
\end{aligned}$$

Ovdje se sumacija izvodi po svim vrijednostima indeksa $i = i_1, i_2, \ldots$ koji odgovaraju redovima u kojima ima vještačkih promjenljivih. Ako u nekom od takvih redova *nema izravnavajuća promjenljive* x_{n+i}, ona se ne pojavljuje ni u gornjem izrazu. Slično, ukoliko je problem bio minimizacija, funkciju cilja transformirano na sljedeći način:

$$
\begin{aligned}
Z &= c_1 x_1 + c_2 x_2 + \ldots + c_n x_n + M x^*_{n+i1} + M x^*_{n+i2} - \ldots = \\
&= c_1 x_1 + c_2 x_2 + \ldots + c_n x_n + M (b_{i1} - a_{i1,1} x_1 - a_{i1,2} x_2 - \ldots - a_{i\,1,n} x_n + x_{n+i1}) \\
&\quad + M (b_{i2} - a_{i2,1} x_1 - a_{i2,2} x_2 - \ldots - a_{i2,n} x_n + x_{n+i2}) + \ldots = \\
&= (c_1 - M \sum a_{i,1}) x_1 + (c_2 - M \sum a_{i,2}) x_2 + \ldots + (c_n - M \sum a_{i,n}) x_n - \\
&\quad + M x_{n+i1} + M x_{n+i2} + \ldots + M \sum b_i
\end{aligned}
$$

Ostaje još neriješeno pitanje *koliku vrijednost uzeti za parametar M*. Praktična ispitivanja pokazuju da je neophodno da *M* bude *barem za dva reda veličine* veći od svih ostalih parametara modela da bi se sa sigurnošću osigurala eliminacija vještačkih promjenljivih. Međutim, to može dovesti do računskih problema. Naime, kako koeficijenti funkcije cilja sada imaju oblik $kM + l$ gdje su k i l neke konstante, u takvim članovima je prvi sabirak tipično *mnogo veći* od drugog, a poznato je da je *sabiranje dva broja koji se međusobno znanto razlikuju po veličini problematična operacija sa aspekta tačnosti u računarskoj aritmetici* (zbog načina na koji se realni brojevi interno pamte u računarskoj memoriji). Stoga je bolja ideja koeficijente oblika $kM + l$ prosto pamtiti kao *par brojeva* k i l (recimo, u dva susjedna reda simpleks tabele) i onda poređenje koeficijenata takvog oblika izvoditi ovako. Neka recimo treba porediti $k' M + l'$ i $k'' M + l''$. S obzirom da je *M jako veliki broj*, ukoliko je $k' \neq k''$, rezultat poređenja prosto možemo utvrditi *poredeći samo* k' *i* k'', *ne obazirući se uopće na* l' *i* l''. Tek ukoliko je $k' = k''$, rezultat poređenja donosimo na osnovu poređenja l' i l''.

Konačno, nije na odmet napomenuti da nema nikakvog razloga zbog kojeg bi se vještačke promjenljive morale označavati *drugačije od ostalih*, kao što smo mi na primjer koristili zvjezdicu. Ovdje je to učinjeno *samo da bi se olakšalo izvođenje prethodnih relacija*, odnosno sa ciljem da se one *učine upadljivijim*.

- ***Primjer*** : Koristeći simpleks metod, riješiti problem linearnog programiranja

$$
\begin{aligned}
&\arg\min Z = 40 x_1 + 30 x_2 \\
&\text{p.o.} \\
&0.1 x_1 \qquad\qquad \geq 0.2 \\
&\qquad 0.1 x_2 \geq 0.3 \\
&0.5 x_1 + 0.3 x_2 \geq 3 \\
&0.1 x_1 + 0.2 x_2 \geq 1.2 \\
&x_1 \geq 0,\ x_2 \geq 0
\end{aligned}
$$

S obzirom da su u ovom problemu *sva ograničenja tipa* "*veće ili jednako*", u svako od ograničenja ćemo morati uvesti *i izravnavajuće i vještačke promjenljive*. Samim tim ćemo morati modificirati i funkciju cilja, uvođenjem *kaznenih koeficijenata* koji će obezbijediti eliminaciju vještačkih promjenljivih. Na ovaj način dobijamo sljedeći prošireni model, u kojem su x_3, x_5, x_7 i x_9 dopunske, a x_4, x_6, x_8 i x_{10} vještačke promjenljive:

$$
\begin{aligned}
&\arg\min Z = 40 x_1 + 30 x_2 + \mathrm{M}\, x_4 + \mathrm{M}\, x_6 + \mathrm{M}\, x_8 + \mathrm{M}\, x_{10} \\
&\text{p.o.} \\
&0.1 x_1 \qquad - x_3 + x_4 \qquad\qquad\qquad\qquad = 0.2 \\
&\qquad 0.1 x_2 \qquad - x_5 + x_6 \qquad\qquad\quad = 0.3 \\
&0.5 x_1 + 0.3 x_2 \qquad\qquad - x_7 + x_8 \qquad = 3 \\
&0.1 x_1 + 0.2 x_2 \qquad\qquad\qquad - x_9 + x_{10} = 1.2 \\
&x_j \geq 0,\ j = 1 \ldots 10
\end{aligned}
$$

Početnu bazu ćemo formirati od vještačkih promjenljivih, odnosno uzećemo $\mathrm{B} = (x_4, x_6, x_8, x_{10})$. Međutim, ovakav oblik zadatka *još nije spreman za primjenu simpleks algoritma* (nije u normalnom obliku), jer u funkciji cilja *postoje bazne promjenljive koje se množe nenultim koeficijentima*. Zato je potrebno ove promjenljive izraziti u funkciji od nebaznih promjenljivih u odgovarajućim ograničenjima i dobijene vrijednosti uvrstiti u funkciju cilja. Na taj način, možemo pisati:

$$
\begin{aligned}
x_4 &= 0.2 - 0.1 x_1 + x_3 \\
x_6 &= 0.3 - 0.1 x_2 + x_5 \\
x_8 &= 3 - 0.5 x_1 - 0.3 x_2 + x_7 \\
x_{10} &= 1.2 - 0.1 x_1 - 0.2 x_2 + x_9
\end{aligned}
$$

Kada se ove vrijednosti uvrste u funkciju cilja dobije se:

$$Z = 40\,x_1 + 30\,x_2 + \mathrm{M}\,(0.2 - 0.1\,x_1 + x_3) + \mathrm{M}\,(0.3 - 0.1\,x_2 + x_5) + \mathrm{M}\,(3 - 0.5\,x_1 - 0.3\,x_2 + x_7) + \\ + \mathrm{M}\,(1.2 - 0.1\,x_1 - 0.2\,x_2 + x_9) =$$

$$= (40 - 0.7\,\mathrm{M})\,x_1 + (30 - 0.6\,\mathrm{M})\,x_2 + \mathrm{M}\,x_3 + \mathrm{M}\,x_5 + \mathrm{M}\,x_7 + \mathrm{M}\,x_9 + 4.7\,\mathrm{M}$$

Do istog rezultata smo mogli doći i koristeći *gotovu formulu* za transformaciju funkcije cilja, jer je:

$$\sum a_{i,1} = a_{1,1} + a_{2,1} + a_{3,1} + a_{4,1} = 0.1 + 0 + 0.5 + 0.1 = 0.7$$
$$\sum a_{i,2} = a_{1,2} + a_{2,2} + a_{3,2} + a_{4,2} = 0 + 0\text{-}1 + 0.3 + 0.2 = 0.6$$
$$\sum b_i = b_1 + b_2 + b_3 + b_4 = 0.2 + 0.3 + 3 + 1.2 = 4.7$$

Konačno, s obzirom da se traži minimizacija, funkciju cilja ćemo pomnožiti sa –1 (da dobijemo problem maksimizacije), tako da dobijamo normalnu formu problema u obliku

$$\arg\max -Z = (0.7\,\mathrm{M} - 40)\,x_1 + (0.6\,\mathrm{M} - 30)\,x_2 - \mathrm{M}\,x_3 - \mathrm{M}\,x_5 - \mathrm{M}\,x_7 - \mathrm{M}\,x_9 - 4.7\,\mathrm{M}$$

p.o.

$$\begin{aligned}
0.1\,x_1 \quad &- x_3 + x_4 &&= 0.2\\
0.1\,x_2 \quad &- x_5 + x_6 &&= 0.3\\
0.5\,x_1 + 0.3\,x_2 \quad &- x_7 + x_8 &&= 3\\
0.1\,x_1 + 0.2\,x_2 \quad &- x_9 + x_{10} &&= 1.2
\end{aligned}$$

$$x_j \geq 0,\ j = 1\,..\,10$$

Možemo započeti sa formiranjem simpleks tabele. Početna simpleks tabela će izgledati ovako:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	
x_4	0.2	0.1	0	–1	1	0	0	0	0	0	0	0.2/0.1 = 2
x_6	0.3	0	0.1	0	0	–1	1	0	0	0	0	
x_8	3	0.5	0.3	0	0	0	0	–1	1	0	0	3/0.5 = 6
x_{10}	1.2	0.1	0.2	0	0	0	0	0	0	–1	1	1.2/0.1 = 12
M	4.7	0.7	0.6	–1	0	–1	0	–1	0	–1	0	
	0	–40	–30	0	0	0	0	0	0	0	0	

Vidimo da u bazu ulazi promjenljiva x_1, a iz nje izlazi promjenljiva x_4. Nakon transformacije, dobija se sljedeća simpleks tabela:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	
x_1	2	1	0	–10	10	0	0	0	0	0	0	
x_6	0.3	0	0.1	0	0	–1	1	0	0	0	0	
x_8	2	0	0.3	5	–5	0	0	–1	1	0	0	2/5 = 0.4
x_{10}	1	0	0.2	1	–1	0	0	0	0	–1	1	1/1 = 1
M	3.3	0	0.6	6	–7	–1	0	–1	0	–1	0	
	80	0	–30	–400	400	0	0	0	0	0	0	

Sada u bazu ulazi promjenljiva x_3, a izlazi promjenljiva x_8. Ovim se dobija sljedeća simpleks tabela:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	
x_1	6	1	0.6	0	0	0	0	–2	2	0	0	6/0.6 = 10
x_6	0.3	0	0.1	0	0	–1	1	0	0	0	0	0.3/0.1 = 3
x_3	0.4	0	0.06	1	–1	0	0	–0.2	0.2	0	0	0.4/0.06 ≈ 6.67
x_{10}	0.6	0	0.14	0	0	0	0	0.2	–0.2	–1	1	0.6/0.14 ≈ 4.29
M	0.9	0	0.24	0	–1	–1	0	0.2	–1.2	–1	0	
	240	0	–6	0	0	0	0	–80	80	0	0	

U sljedećoj iteraciji, u bazu ulazi promjenljiva x_2, a izlazi promjenljiva x_6, što nakon obavljenih transformacija daje sljedeću simpleks tabelu:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	
x_1	4.2	1	0	0	0	6	–6	–2	2	0	0	4.2/6 = 0.7
x_2	3	0	1	0	0	–10	10	0	0	0	0	
x_3	0.22	0	0	1	–1	0.6	–0.6	–0.2	0.2	0	0	0.22/0.6 ≈ 0.37
x_{10}	0.18	0	0	0	0	1.4	–1.4	0.2	–0.2	–1	1	0.18/1.4 ≈ 0.13
M	0.18	0	0	0	–1	1.4	–2.4	0.2	–1.2	–1	0	
	258	0	0	0	0	–60	60	–80	80	0	0	

U bazu sada ulazi promjenljiva x_5, a ispada promjenljiva x_{10} (čime su sve vještačke promjenljive ispale iz baze, odnosno anulirane su). Nakon toga se dobija sljedeća simpleks tabela:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
x_1	24/7	1	0	0	0	0	0	–20/7	20/7	30/7	–30/7
x_2	30/7	0	1	0	0	0	0	10/7	–10/7	–50/7	50/7
x_3	1/7	0	0	1	–1	0	0	–2/7	2/7	3/7	–3/7
x_5	9/70	0	0	0	0	1	–1	1/7	–1/7	–5/7	5/7
M	0	0	0	0	–1	0	–1	0	–1	0	–1
	1860/7	0	0	0	0	0	0	–500/7	500/7	–300/7	300/7

Pošto smo došli do tabele u kojoj su svi koeficijenti c_j negativni, algoritam se završava. Kako su pri tome sve vještačke promjenljive *anulirane*, dobijeno rješenje je *legalno* i *optimalno*. Dakle, baza optimalnog rješenja je $B = (x_1, x_2, x_3, x_5)$. Samo optimalno rješenje (ako u njega uključimo i vještačke promjenljive) glasi $\mathbf{x} = (24/7, 30/7, 1/7, 0, 9/70, 0, 0, 0, 0, 0)^T$, dok optimalna vrijednost funkcije cilja iznosi $Z = 1860/7$ (podatak iz tabele *nismo negirali*, jer se tražio minimum).

Treba napomenuti da nakon što vještačka promjenljiva *jedanput napusti bazu, ne postoji nikakav mehanizam kojim bi ona mogla ponovo ući u bazu*. Ovo omogućava da nakon što eliminiramo odgovarajuću vještačku promjenljivu iz baze, više ne vodimo i ne preračunavamo odgovarajuću kolonu u tabeli. Na taj način, možemo *osjetno skratiti postupak* (mada se u praksi kolone koje odgovaraju vještačkim promjenljivim često *zadržavaju*, jer se na kraju postupka iz njih mogu ponekad očitati neke informacije *korisne za postoptimalnu analizu*). Po skraćenom postupku, tok algoritma bi tekao ovako. Krenimo od početne simpleks tabele:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	
x_4	0.2	0.1	0	–1	1	0	0	0	0	0	0	0.2/0.1 = 2
x_6	0.3	0	0.1	0	0	–1	1	0	0	0	0	
x_8	3	0.5	0.3	0	0	0	0	–1	1	0	0	3/0.5 = 6
x_{10}	1.2	0.1	0.2	0	0	0	0	0	0	–1	1	1.2/0.1 = 12
M	4.7	0.7	0.6	–1	0	–1	0	–1	0	–1	0	
	0	–40	–30	0	0	0	0	0	0	0	0	

U bazu ulazi promjenljiva x_1, dok vještačka promjenljiva x_4 iz nje izlazi, što odmah omogućava da eliminiramo odgovarajuću kolonu. Nakon transformacije, dobija se sljedeća simpleks tabela:

Baza	b_i	x_1	x_2	x_3	x_5	x_6	x_7	x_8	x_9	x_{10}	
x_1	2	1	0	–10	0	0	0	0	0	0	
x_6	0.3	0	0.1	0	–1	1	0	0	0	0	
x_8	2	0	0.3	5	0	0	–1	1	0	0	2/5 = 0.4
x_{10}	1	0	0.2	1	0	0	0	0	–1	1	1/1 = 1
M	3.3	0	0.6	6	–1	0	–1	0	–1	0	
	80	0	–30	–400	0	0	0	0	0	0	

U bazu sada ulazi promjenljiva x_3, a iz nje izlazi vještačka promjenljiva x_8, što ponovo omogućava eliminaciju jedne kolone. Ovim se dobija se sljedeća simpleks tabela:

Baza	b_i	x_1	x_2	x_3	x_5	x_6	x_7	x_9	x_{10}	
x_1	6	1	0.6	0	0	0	–2	0	0	$6/0.6 = 10$
x_6	0.3	0	0.1	0	–1	1	0	0	0	$0.3/0.1 = 3$
x_3	0.4	0	0.06	1	0	0	–0.2	0	0	$0.4/0.06 \approx 6.67$
x_{10}	0.6	0	0.14	0	0	0	0.2	–1	1	$0.6/0.14 \approx 4.29$
M	0.9	0	0.24	0	–1	0	0.2	–1	0	
	240	0	–6	0	0	0	–80	0	0	

Nakon toga, u bazu ulazi promjenljiva x_2, a izlazi vještačka promjenljiva x_6, koju ćemo također eliminirati. Ovo, nakon obavljenih transformacija, daje sljedeću simpleks tabelu:

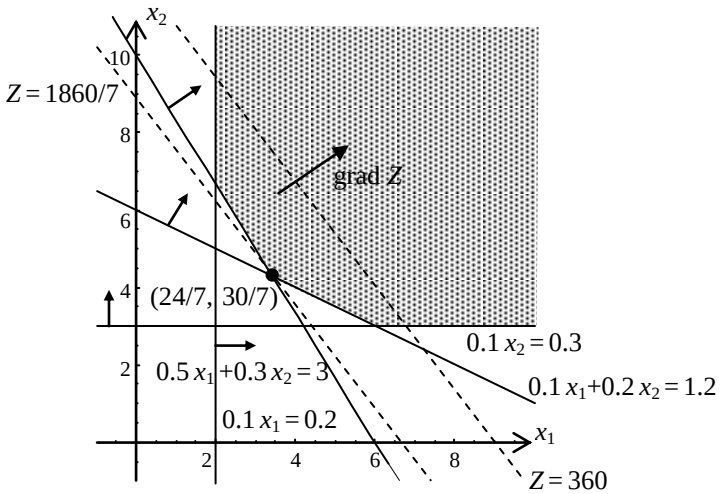
Baza	b_i	x_1	x_2	x_3	x_5	x_7	x_9	x_{10}	
x_1	4.2	1	0	0	6	–2	0	0	$4.2/6 = 0.7$
x_2	3	0	1	0	–10	0	0	0	
x_3	0.22	0	0	1	0.6	–0.2	0	0	$0.22/0.6 \approx 0.37$
x_{10}	0.18	0	0	0	1.4	0.2	–1	1	$0.18/1.4 \approx 0.13$
M	0.18	0	0	0	1.4	0.2	–1	0	
	258	0	0	0	–60	–80	0	0	

U sljedećoj iteraciji, u bazu ulazi promjenljiva x_5, a iz nje ispada posljednja preostala vještačka promjenljiva x_{10}. Nakon što sve vještačke promjenljive ispadnu iz baze, možemo iz tablice ukloniti i red koji sadrži koeficijente koji se množe sa M (oni će u svakom slučaju do tog trenutka postati jednaki nuli). Ovim se dobija sljedeća simpleks tabela:

Baza	b_i	x_1	x_2	x_3	x_5	x_7	x_9
x_1	24/7	1	0	0	0	–20/7	30/7
x_2	30/7	0	1	0	0	10/7	–50/7
x_3	1/7	0	0	1	0	–2/7	3/7
x_5	9/70	0	0	0	1	1/7	–5/7
	1860/7	0	0	0	0	–500/7	–300/7

Kako među koeficijentima c_j nema pozitivnih, ovim je optimalno rješenje nađeno.

Pogledajmo na kraju još i grafičku interpretaciju ovog problema i njegovog rješenja, što je uvijek poučno uraditi kad god je problem dvodimenzionalan:



- ***Primjer***: Potrebno je napraviti obrok od četiri vrste hrane H_1, H_2, H_3 i H_4, koje po kilogramu sadrže 250, 150, 400 i 200 vitaminskh jedinica respektivno. Obrok mora sadržavati najmanje 300 vitaminskih jedinica po kilogramu, ne smije sadržavati više od 30 % hrane H_4, niti više od 50 % hrane H_2 i H_3 zajedno. Cijene svake od ovih hrana po kilogramu su 32, 56, 50 i 60 novčane jedinice respektivno. Potrebno je odrediti procentualni udio svake od hrana u obroku, pa da cijena obroka po kilogramu bude minimalna.

Označimo sa x_1, x_2, x_3 i x_4 procentualne udjele hrana H_1, H_2, H_3 i H_4 u obroku. Kako se obrok sastoji samo od ove četiri vrste hrane i ničega drugog, zbir procentualnih udjela svih ovih hrana zajedno mora biti 100 %, što nameće ograničenje $x_1 + x_2 + x_3 + x_4 = 1$. Ostala ograničenja su očigledna iz same postavke zadatka, tako da matematski oblik problema glasi

$$\arg\min Z = 32\,x_1 + 56\,x_2 + 50\,x_3 + 60\,x_4$$

p.o.

$$\begin{aligned} x_1 + \;\; x_2 + \;\; x_3 + \;\; x_4 &= 1 \\ 250\,x_1 + 150\,x_2 + 400\,x_3 + 200\,x_4 &\geq 300 \\ x_4 &\leq 0.3 \\ x_2 + \;\; x_3 &\leq 0.5 \end{aligned}$$

$$x_1 \geq 0,\;\; x_2 \geq 0,\;\; x_3 \geq 0,\;\; x_4 \geq 0$$

Ograničenja $x_1 + x_2 + x_3 + x_4 = 1$ tipa jednakosti bi se u principu mogli osloboditi uvođenjem smjene $x_4 = 1 - x_1 - x_2 - x_3$ (i tako smanjiti dimenzionalnost problema), ali ovdje ćemo pokazati *kako sa takvim ograničenjem možemo direktno raditi* uvođenjem odgovarajuće vještačke promjenljive. Prvo ćemo uvesti dopunske promjenljive x_5, x_6 i x_7 u sva ograničenja osim prvog (u ograničenju tipa "veće li jednako" odgovarajuća dopunska promjenljiva ide sa znakom "–"), a zatim ćemo u prva dva ograničenja koja nisu tipa "manje ili jednako" uvesti i odgovarajuće vještačke promjenljive x_8 odnosno x_9 (naravno, izravnavajuće i vještačke promjenljive *nismo morali ovako numerirati*). Pri tome, moramo uvesti i odgovarajuće *kaznene koeficijente* u funkciji cilja uz vještačke promjenljive, da bismo obezbijedili njihovu eliminaciju. Na taj način, dobijamo proširenu model:

$$\arg\min Z = 32\,x_1 + 56\,x_2 + 50\,x_3 + 60\,x_4 + \mathrm{M}\,x_8 + \mathrm{M}\,x_9$$

p.o.

$$\begin{aligned} x_1 \;\; + x_2 \;\; + x_3 \;\; + \;\; x_4 \qquad\qquad + x_8 \;\; &= 1 \\ 250\,x_1 + 150\,x_2 + 400\,x_3 + 200\,x_4 - x_5 \qquad + x_9 &= 300 \\ x_4 \;\; + x_6 \qquad\quad &= 0.3 \\ x_2 \;\; + x_3 \qquad + x_7 \;\; &= 0.5 \end{aligned}$$

$$x_j \geq 0,\;\; j = 1 \,..\, 9$$

Početnu bazu će ovdje činiti dvije vještačke promjenljive x_8 i x_9, te dopunske promjenljive x_6 i x_7, odnosno uzećemo $\mathrm{B} = (x_8, x_9, x_6, x_7)$. Kao i u prethodnom primjeru, ovakav oblik zadatka nije spreman za direktnu primjenu simpleks algoritma, jer u funkciji cilja postoje bazne promjenljive koje se množe nenultim koeficijentima. Izrazimo ove promjenljive u funkciji od nebaznih promjenljivih:

$$x_8 = 1 - x_1 - x_2 - x_3 - x_4$$

$$x_9 = 300 - 250\,x_1 - 150\,x_2 - 400\,x_3 - 200\,x_4 + x_5$$

Uvrštavanjem ovih vrijednosti u funkciju cilja dobijamo:

$$\begin{aligned} Z &= 32\,x_1 + 56\,x_2 + 50\,x_3 + 60\,x_4 + \\ &\quad + \mathrm{M}\,(1 - x_1 - x_2 - x_3 - x_4) + \mathrm{M}\,(300 - 250\,x_1 - 150\,x_2 - 400\,x_3 - 200\,x_4 + x_5) = \\ &= (32 - 251\,\mathrm{M})\,x_1 + (56 - 151\,\mathrm{M})\,x_2 + (50 - 401\,\mathrm{M})\,x_3 + (60 - 201\mathrm{M})\,x_4 + \mathrm{M}\,x_5 + 301\,\mathrm{M} \end{aligned}$$

Do istog rezultata smo mogli doći i koristeći *gotovu formulu* za transformaciju funkcije cilja, jer je:

$$\begin{aligned} \sum a_{i,1} &= a_{1,1} + a_{2,1} = 1 + 250 = 251 \\ \sum a_{i,2} &= a_{1,2} + a_{2,2} = 1 + 150 = 151 \\ \sum a_{i,3} &= a_{1,3} + a_{2,3} = 1 + 400 = 401 \\ \sum a_{i,4} &= a_{1,4} + a_{2,4} = 1 + 200 = 201 \\ \sum b_i &= b_1 + b_2 = 1 + 300 = 301 \end{aligned}$$

S obzirom da se traži minimizacija, funkciju cilja još treba pomnožiti sa –1, tako da se dobija normalna forma problema u obliku

$$\arg\max -Z = (251\,\mathrm{M}-32)x_1 + (151\,\mathrm{M}-56)x_2 + (401\,\mathrm{M}-50)x_3 + (201\mathrm{M}-60)x_4 - \mathrm{M}\,x_5 - 301\,\mathrm{M}$$

p.o.

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 + x_8 &= 1 \\ 250x_1 + 150x_2 + 400x_3 + 200x_4 - x_5 + x_9 &= 300 \\ x_4 + x_6 &= 0.3 \\ x_2 + x_3 + x_7 &= 0.5 \end{aligned}$$

$x_j \geq 0,\ j = 1 \,..\, 9$

Sada je sve spremno za formiranje prve simpleks tabele, koja će izgledati ovako:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	
x_8	1	1	1	1	1	0	0	0	1	0	1/1 = 1
x_9	300	250	150	400	200	–1	0	0	0	1	300/400 = 0.75
x_6	0.3	0	0	0	1	0	1	0	0	0	
x_7	0.5	0	1	1	0	0	0	1	0	0	0.5/1 = 0.5
M	301	251	151	401	201	–1	0	0	0	0	
	0	–32	–56	–50	–60	0	0	0	0	0	

U sljedeću bazu ulazi promjenljiva x_3, a iz nje izlazi promjenljiva x_7. Nakon obavljenih transformacija, dolazimo do sljedeće simpleks tabele.

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	
x_8	0.5	1	0	0	1	0	0	–1	1	0	0.5/1 = 0.5
x_9	100	250	–250	0	200	–1	0	–400	0	1	100/250 = 0.4
x_6	0.3	0	0	0	1	0	1	0	0	0	
x_3	0.5	0	1	1	0	0	0	1	0	0	
M	100.5	251	–250	0	201	–1	0	–401	0	0	
	25	–32	–6	0	–60	0	0	50	0	0	

Sada će u bazu ući promjenljiva x_1, a iz nje će ispasti promjenljiva x_9. Kako je ona ujedno vještačka promjenljiva, ona više nikada neće ući nazad u bazu, te ćemo pripadnu kolonu *u potpunosti izbaciti iz tabele*. Time se, nakon obavljenih preračunavanja, dolazi do sljedeće tabele:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	
x_8	0.1	0	1	0	0.2	0.004	0	0.6	1	0.1/1 = 0.1
x_1	0.4	1	–1	0	0.8	–0.004	0	–1.6	0	
x_6	0.3	0	0	0	1	0	1	0	0	
x_3	0.5	0	1	1	0	0	0	1	0	0.5/1 = 0.5
M	0.1	0	1	0	0.2	0.004	0	0.6	0	
	37.8	0	–38	0	–34.4	–0.128	0	–1.2	0	

U bazu sada ulazi promjenljiva x_2, a iz nje ispada promjenljiva x_8. Kako je ona također vještačka promjenljiva, ni ona više nikada neće ući nazad u bazu, te ćemo pripadnu kolonu također *u potpunosti izbaciti iz tabele* (mada ćemo kasnije vidjeti da bi se *upravo iz ove kolone* po završetku algoritma mogle očitati i *neke korisne informacije*, ali one nam *za sada ne trebaju*). Ovim su ujedno iz problema *eliminirane sve vještačke promjenljive*, te iz tabele možemo izbaciti i red koji se množi sa koeficijentom M. Tako dolazimo do sljedeće simpleks tabele:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_2	0.1	0	1	0	0.2	0.004	0	0.6	0.1/0.6 ≈ 0.167
x_1	0.5	1	0	0	1	0	0	–1	
x_6	0.3	0	0	0	1	0	1	0	
x_3	0.4	0	0	1	–0.2	–0.004	0	0.4	0.4/0.4 = 1
	41.6	0	0	0	–26.8	0.024	0	21.6	

Konačno smo *ušli u dopustivi prostor* (tj. našli smo *jedno dozvoljeno bazno rješenje*), jer smo eliminirali sve vještačke promjenljive. Međutim, ovo dopustivo rješenje nije i optimalno, jer u posljednjem redu tabele ima pozitivnih koeficijenata c_j. Jasno je sada da u bazu treba uvesti promjenljivu x_7. Iz baze pri tome ispada promjenljiva x_2, tako da se nakon obavljenih preračunavanja dolazi do sljedeće simpleks tabele:

Baza	b_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_7	1/6	0	5/3	0	1/3	1/150	0	1
x_1	2/3	1	5/3	0	4/3	1/150	0	0
x_6	0.3	0	0	0	1	0	1	0
x_3	1/3	0	–2/3	1	–1/3	–1/150	0	0
	38	0	–36	0	–34	–0.12	0	0

Ovim smo konačno stigli do optimalnog rješenja. Odavde direktno možemo očitati optimalno rješenje $x_1 = 2/3 \approx 66.7$ %, $x_2 = 0$, $x_3 = 1/3 \approx 33.3$ %, $x_4 = 0$, $x_5 = 0$, $x_6 = 0.3 = 30$ % i $x_7 = 1/6 \approx 16.7$ %, dok je optimalna vrijednost funkcije cilja $Z = 38$. Dakle, optimalan obrok sastoji se od 2/3 dijela (ili oko 66.7 %) hrane H_1 i 1/3 dijela (ili oko 33.3 %) hrane H_2, dok hrane H_2 i H_4 *uopće ne treba koristiti*. Optimalna cijena obroka je 38 novčanih jedinica po kilogramu. Također, vidimo da će obrok sadržavati tačno onoliko vitaminskih jedinica koliko je minimalno propisano (zbog $x_5 = 0$), dok će udio hrane H_4 odnosno zbirni udio hrana H_2 i H_3 biti za 30 % odnosno oko 16.7 % manji od maksimalno dozvoljene propisane granice.