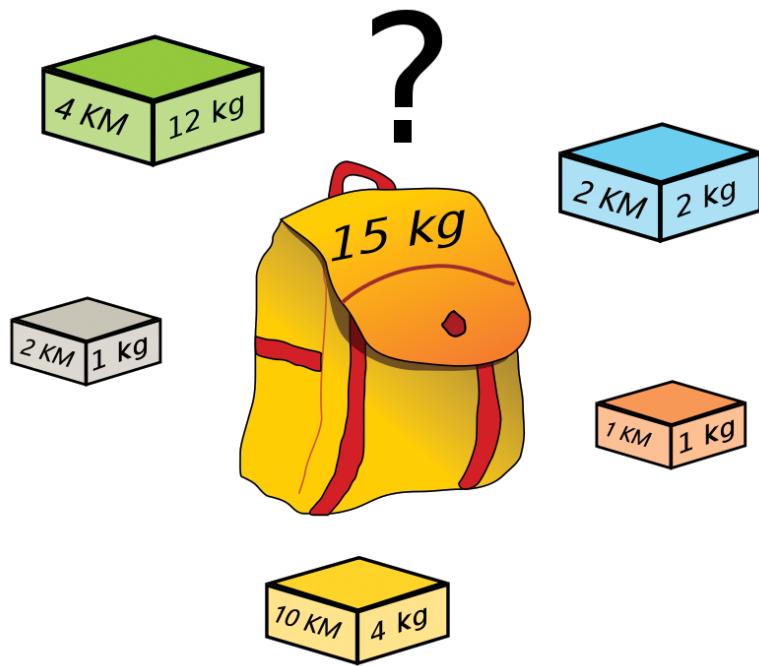


Uvod u operaciona istraživanja

Marko Đukanović i Dragan Matić



Univerzitet u Banjoj Luci
Prirodno-matematički fakultet
2022.

Marko Đukanović, Dragan Matić
Uvod u operaciona istraživanja

Izdavavač
Prirodno-matematički fakultet
Univerzitet u Banjoj Luci

Recenzenti
doc. dr Milana Grbić,
Prirodno-matematički fakultet, Univerzitet u Banjoj Luci
doc. dr Aleksandar Kartelj
Matematički fakultet, Univerzitet u Beogradu

Slika na naslovnoj strani
Which boxes to choose to maximize the amount of money while still
fullfilling the 15 kg constraint? (autor slike: Dake)

BIBLIOTEKA

Predgovor

Ovaj udžbenik je namijenjen studentima koji slušaju uvodni kurs iz Operacionih istraživanja. Po svojoj strukturi i sadržaju udžbenik najviše prati nastavni plan i program za Operaciona istraživanja, za studente treće godine informatičkog smjera Prirodno-matematičkog fakulteta Univerziteta u Banjoj Luci.

Prepostavljamo da čitaoci ovog udžbenika u dovoljnoj mjeri vladaju solidnim znanjem iz matematike i programiranja, koje uključuje osnovne linearne algebre (manipulacija vektorima i matricama), teorije grafova, kombinatorike, kalkulusa, teorije algoritama i struktura podataka. Za implementaciju programa linearног programiranja, podrazumijeva se posjedovanje osnovnih programerskih vještina, koje se stiču na uvodnim kursevima iz programiranja, te poznавanje rada u programskim jezicima C/C++ i/ili Python.

Cilj ovog udžbenika je da čitaoca upozna kako sa lijepom matematičkom teorijom koja je u osnovi operacionih istraživanja, tako i sa praktičkim problemima, koji se rješavaju primjenom odgovarajućih tehnika. U udžbeniku je prezentovan veliki broj primjera realnih problema koji potiču iz prakse, a na različitim mjestima su objasnijene različite faze rješavanja tih problema, počev od identifikovanja samog problema, njegove formalne matematičke formulacije, analize metoda za rješavanje, pa do implementacije rješenja u odgovarajućem rješavaču ili pak pomoću samostalno dizajniranog algoritma.

Uz pomoć znanja stečenog iz ovog udžbenika, polaznik će biti sposoban da sam identificuje i formuliše probleme koji spadaju u oblast ope-

racionih istraživanja (sa naglaskom na probleme linearног i cjelobrojnog programiranja), analizira različite metode za rješavanje datih problema, te, primjenjujući odgovarajuće računarske alate, implementira algoritme i rješava probleme.

Unutar svakog poglavlja prikazan je i značajan broj primjera koji mogu pomoći lakšem razumijevanju gradiva. Neki primjeri su urađeni kompletno, od početka do kraja, dok neki drugi sadrže prikaze koraka koji su ključni za razumijevanje datog koncepta. Treba pomenuti da u modelovanju problema metodama linearног programiranja, često postoji više podjednako dobrih i efikasnih rješenja. Stoga se od čitaoca ne očekuje da predložena rješenja usvoji kao jedina moguća, već da analizom ponuđenih rješenja i povezivanjem raznih koncepata i tehnika sam dođe do svojih rješenja, koja su podjednako ispravna i efikasna.

Na kraju svakog poglavlja dat je veći broj pitanja i zadataka. Cilj ovih zadataka nije da čitalac samo reprodukuje stečeno znanje, već da sam stekne sposobnost da modeluje i implementira algoritme kojima rješava različite probleme, procjenjuje kvalitet rješenja, pronalazi rješenja za razne probleme, koji, na prvi pogled, nisu u bliskoj vezi sa gradivom iznesenim u udžbeniku.

Mnogi koncepti u operacionim istraživanjima se međusobno prepliću i nadopunjaju. Npr. teorija dualnosti se koristi u dekompozicionim metodama, simpleks metoda se može kombinovati sa drugim metodama za rješavanje problema linearног i cjelobrojnog linearног programiranja, a egzaktne i heurističke metode za rješavanje problema često se međusobno kombinuju i objedinjuju u tzv. hibridne metode. Stoga je preporuka da se ovaj udžbenik ne čita isključivo redom, već da se, čak i mnogo češće nego što je to u tekstu sugerisano, sadržaj više poglavlja izučava u isto vrijeme, vraća na prethodna poglavlja ili prelazi na neka naredna.

Na kraju, želimo da se zahvalimo recenzentima, doc. dr Milani Grbić i doc. dr Aleksandru Kartelju, koji su svojim sugestijama i komentarima poboljšali kvalitet ovog udžbenika. Takođe, zahvaljujemo se na podršci i kolegama sa Katedre za računarske i informatičke nauke Prirodno-matematičkog fakulteta Univerziteta u Banjoj Luci.

Notacija

Uvedimo neke notacijske napomene i konvencije koje smo koristili u knjizi.

- Smatraćemo da je vektor uvijek dat u kolonskoj notaciji, tj. $x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$, dok je njegov odgovarajući transponovani vektor $x^T = (x_1, \dots, x_n) \in \mathbb{R}^{1 \times n}$.
- Što se tiče skalara i vektora, nećemo koristiti drugačije oznake da bismo ih razlikovali. Jednostavno, biće jasno iz konteksta da li je riječ o jednom ili drugom. Dakle, nula vektor ćemo takođe označavati sa $x = (0, \dots, 0)^T := 0$, dok je $x = (1, \dots, 1)^T := 1$.
- Ako je $x \in \mathbb{R}^n$, notacija $x \geq 0$ podrazumijeva da je vrijednost svake od koordinatna vektora x veća ili jednaka 0, tj. $x_i \geq 0$, za sve $i = 1, \dots, n$.

Sadržaj

1 Predmet izučavanja Operacionih istraživanja	1
2 Principi modelovanja u Operacionim istraživanjima	7
2.1 Definisanje problema i sakupljanje relevantnih podataka	8
2.2 Proces formulisanja matematičkog modela	10
2.3 Izvođenje rješenja iz modela	14
2.4 Testiranje modela	16
2.5 Pripremanje modela za primjenu	17
2.6 Implementacija	19
3 Osnove linearног programiranja	21
3.1 Grafički metod	24
3.2 Standardna forma linearног programiranja	26
3.3 Teorija linearног programiranja	33
3.4 Neke tehnike u modelovanju problema linearног programiranja	42
3.5 Zadaci	47
4 Simpleks metoda	53
4.1 Konstrukcija simpleks metode	54
4.2 Tabelarni oblik simpleks metoda	57
4.3 Primjena simpleks metoda	59
4.4 Dvofazni simpleks metod	65
4.5 = geometrijskoj interpretaciji simpleks metoda	70

4.6	Zadaci	75
5	Dualnost i dopustivost	77
5.1	Teoreme dualnosti	79
5.2	Farkašova lema	87
5.3	Furije–Mockinov metod eliminacije	89
5.4	Zadaci	93
6	Cjelobrojno linearno programiranje	97
6.1	Modelovanje nekih problema cjelobrojnog linearног programiranja	98
6.2	Tehnike transformacija nekih nelinearnih izraza u ILP-u	107
6.2.1	Prepoznavanje binarnih promjenljivih	108
6.2.2	Logička ograničenja	108
6.2.3	Alternativna ograničenja	109
6.2.4	Uslovna ograničenja	110
6.2.5	K-alternative	110
6.2.6	Složene alternative	111
6.2.7	Fiksne cijene u funkciji cilja	112
6.2.8	Po-dijelovima-linearna funkcija	112
6.2.9	Eliminacija proizvoda promjenljivih	114
6.3	Kompleksnost rješavanja cjelobrojnog programiranja	118
6.4	Zadaci	120
7	Algoritamske tehnike za rješavanje Cjelobrojnog linearног programiranja	127
7.1	Metoda grananja i ograničavanja (B&B)	128
7.1.1	Bazni B&B za rješavanje problema ILP-a	129
7.2	Implicitna enumeracija	133
7.3	Metoda odsjecajućih ravnih	136
7.4	B&C algoritam za rješavanje ILP-a	145
7.5	Lagranžova relaksacija	148
7.6	Metod generisanja kolona	152
7.7	Zadaci	161

8 Dekompozicione metode	167
8.1 Benderova dekompozicija	167
8.2 Dantzig–Wolfe dekompozicija	185
8.3 Zadaci	201
9 Metode za pronalaženje dopustivih rješenja	203
9.1 Pohlepni algoritmi	204
9.1.1 Algoritmi za pronalaženje MST	205
9.1.2 Dajkstrin algoritam	207
9.1.3 Razlomljeni problem jednodimenzionalnog ruksaka	209
9.1.4 Pohlepni pristup za rješavanje LCS problema	210
9.2 Aproksimativni algoritmi	213
9.2.1 Aproksimativni algoritmi za problem minimalnog pokrivanja čvorova	214
9.2.2 Aproksimativni algoritmi za problem trgovačkog putnika	217
9.2.3 Aproksimativni algoritmi za problem binarnog pakovanja	221
9.2.4 Aproksimativni algoritmi za problem pokrivanja skupa	223
9.3 Heurističke metode	226
9.4 Metaheurističke metode	227
9.4.1 Klasifikacija metaheuristika	228
9.4.2 Metoda lokalne pretrage	230
9.4.3 Genetski algoritmi	231
9.4.4 Metoda promjenljivih okolina	241
9.5 Zadaci	244
10 Optimizacioni rješavači	249
10.1 Optimizacioni rješavač CPLEX	249
10.2 PuLP alat	256
10.3 Primjena alata PuLP na rješavanje jednog problema lokacija sa ograničenim kapacitetima	259
10.4 Primjena CPLEX rješavača na problem najdužeg zajedničkog podniza bez ponavljanja	261

10.5 Zadaci	267
A Metod unutrašnje tačke	271
B Makefile za kompajliranje CPLEX programa	281

Glava 1

Predmet izučavanja Operacionih istraživanja

Operaciona istraživanja su oblast koja pripada matematičkim disciplinama. Ona predstavlja osnovnu disciplinu u menadžmentu, a naziv joj potiče iz njene osnovne uloge, a to je istraživanje operacija u organizacionim sistemima (logistika, problem raspoređivanja, dodjeljivanja zadataka itd.) gdje je svrha optimizacija (troškova, vremena, ljudskih resursa, opreme itd.). Operaciona istraživanja u osnovi predstavljaju analitički metod rješavanja problema i donošenja odluka koje se primjenjuju u organizaciji raznih sistema i procesa. Metodologija ovih istraživanja najčešće obuhvata razbijanje problema na bazne komponente, te potom njihovo rješavanje u definisanim koracima uz pomoć matematičkog aparata.

Koncept operacionih istraživanja potiče iz vremena Drugog svjetskog rata, gdje su vojni planeri razvijali strategiju postavljanja vojske i vojne opreme sa ciljem što veće mobilnosti i efikasnosti. Od tog vremena, pa sve do danas, tehnike u operacionim istraživanjima se primjenjuju u rješavanju velikog broja problema u poslovnom svijetu, industriji, društvenim pitanjima i mnogim drugim aspektima savremenog društva.

Proces u operacionom istraživanju se ugrubo može predstaviti sljedećim koracima:

-
- Opisivanje problema koji rješavamo.
 - Konstrukcija modela problema koji izražava taj problem preko promjenljivih i relacija između njih.
 - Korištenje modela za generisanje rješenja problema.
 - Testiranje svakog od rješenja i analiziranje njihovih kvaliteta.
 - Implementacija odabranog rješenja datog problema.

Detaljima ovih koraka ćemo se baviti u narednoj sekciji.

Glavne teme izučavanja operacionih istraživanja su:

- *Optimizacija*: uključuje poređenja i navođenje pretrage prema potencijalno boljim opcijama u okviru zadanih resursa.
- *Simulacija*: podrazumijeva izgradnju modela problema iz koga se dobijaju rješenja, koja se nakon toga provjeravaju prije nego što se implementiraju u realnoj situaciji.
- *Vjerovatnoća i statistika*: podrazumijeva (*i*) upotrebu matematičkih algoritama i podataka u smislu otkrivanja korisnih informacija koje služe za dobijanje pouzdanih predviđanja u realnom svijetu; (*ii*) testiranje kvaliteta i upotrebljivosti rješenja.

Operaciona istraživanja imaju nezamjenjivu ulogu u nizu obasti kao što su problemi raspoređivanja, urbanog planiranja, optimizacije na mrežama i inženjeringu, rutiranja paketa, upravljanja rizicima, upravljanja lancem snabdjevanja i mnogim drugim.

Operaciona istraživanja rješavaju probleme iz prakse (nazovimo ih poslovni problemi), koji ljudima štede novac i vrijeme. Ovi problemi mogu biti raznoliki i po pravilu ne izgledaju naročito povezani jedni sa drugim. Gledajući šire, osnov im je uvijek isti, donošenje niza odluka da bi se došlo do cilja na što efikasniji način. Jedna od najmoćnijih tehnika koje se mogu koristiti u operacionim istraživanjima je *matematičko programiranje*. Treba imati na umu da riječ „programiranje“ nema standardno značenje na koje smo i

navikli – imperativno, vec ono u ovom kontekstu označava programiranje optimizacionih modela, koje je u osnovi deklarativno.

Put od učenja o problemu kojeg klijent predstavi do pronađaska rješenja može biti jako izazovan. Ovaj proces rješavanja se dijeli u četiri osnovna nivoa apstrakcije:

1. poslovni problemi;
2. generički problemi;
3. paradigma modelovanja;
4. algoritamska rješenja.

Pod *poslovnim problemima* se ne podrazumijevaju samo problemi iz domena ekonomije, što samo ime možda nameće, već svi problemi koji imaju primjenu u stvarnom svijetu. Pojam „poslovni“ više odgovara tome da problem nema svoju preciznu matematičku formulaciju, već je neformalno definisan kroz jezički (deskriptivan) opis. Najčešće, ovi problemi proizilaze iz problema u ekonomiji ili industriji i kao takvi se i opisuju. Npr. u neformalnom opisu problema raspoređivanja medicinskog osoblja u (tri) smjene, navode se uslovi poput toga da svaki radnik mora da odradi 40 sati sedmično ili da niti jedan radnik ne smije da radi više od dvije noćne smjene u toku sedmice. Nakon opisnog, neformalnog definisanja, naredni korak je formalno definisanje problema, odnosno, „spuštanje“ (opisa) problema na niži nivo apstrakcije. U praksi se ovaj proces odvija iterativno i povezuje se sa terminom *generički problemi*. Npr. problem pravljenja rasporeda časova, koji je poslovni problem, se može prevesti u generički OR problem bojenja grafa.

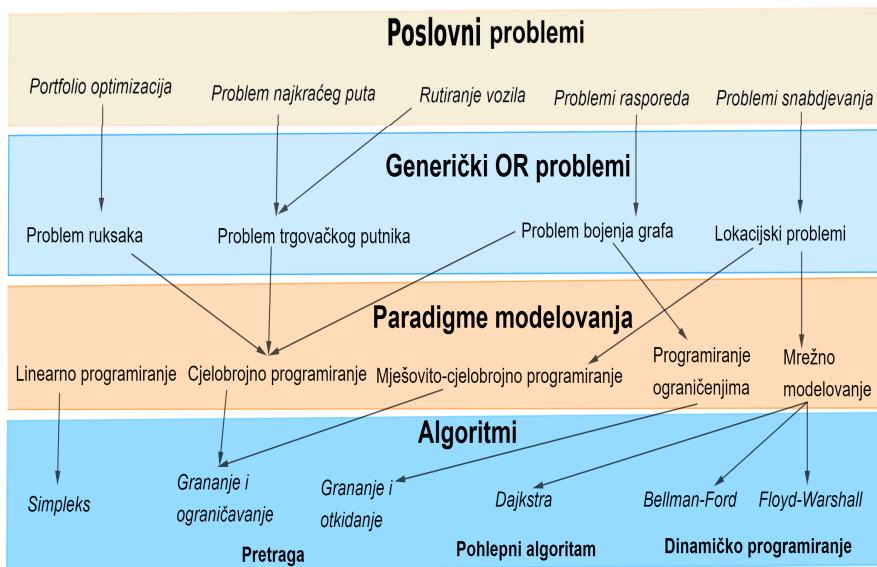
Na početku razvoja operacionih istraživanja kao naučne discipline nije bilo strogog matematičkog formalizma. Matematičari su prepoznali da se većina poslovnih problema može preslikati u generičke familije problema nižeg nivoa. Na kraju se završilo sa nekoliko opšteprihvaćenih generičkih klasa na koje se preslikava gotovo svaki poslovni problem. U praksi, najveći dio vremena se provodi u prevođenju poslovnog problema u jedan od tipova generičkih problema. Nakon toga, rješavanje problema je manje-više

standardna procedura. Generički problemi operacionih istraživanja su dovoljno koncizni da se mogu predstaviti matematičkom notacijom. Međutim, u svrhu boljeg razumijevanja, u praksi se koristi jezik modelovanja koji je na višem nivou apstrakcije. Npr. problemi raspoređivanja se obično opisuju upotrebom „stvarnih“ termina, kao što su resursi, aktivnosti, prioriteti, uslovi raspoređivanja i sl.

Paradigma modelovanja. Kod generičkih problema koji su definisani jezikom višeg nivoa, notacija je najčešće vezana za problem koji se razmatra. Npr. resursi i aktivnosti nisu nešto što se može definisati u poznatom problemu trgovačkog putnika. U osnovi, svaki od ovih generičkih problema se može opisati nekom od paradigmi modelovanja. Paradigma modelovanja predstavlja skup pravila koja omogućuju da predstavimo probleme višeg nivoa, koristeći strukture nižeg nivoa, kao što su, na primjer, matrice. Kada se primjenjuje ova paradigma, problem se izražava upotrebom matematičke notacije ili algebarskog jezika modelovanja (eng. *Algebraic Modelling Language* (AML)), koji konvertuje matematičku notaciju u matrice koje su proslijedene sljedećem nivou (*algoritamska rješenja*), koji generiše rješenja. Najpoznatije paradigme modelovanja su sljedeće: *linearno*, *cjelobrojno* i *mješovito-cjelobrojno programiranje*. Postoje i druge paradigme poput modelovanja ograničenjima (eng. *constraint programming*) i mrežnog modelovanja (eng. *network models*).

Algoritamska rješenja. Algoritam je konačan niz instrukcija čijim izvršavanjem možemo riješiti problem ili njegov dio. Neki algoritmi su opšti, kao, na primjer, sortiranje ili pretraživanje, dok su drugi vezani uz specifične probleme. Algoritmi pretrage su izuzetno važni u rješavanju problema operacionih istraživanja. Među njima se izdvajaju „Branch & X“-tip algoritama koji rješavaju Cjelobrojne, Mješovito-cjelobrojne probleme kao i probleme zasnovane na modelovanju ograničenjima. Među ove algoritme spadaju Branch & Bound (za rješavanje problema cjelobrojnog programiranja), Branch & Cut, Branch & Price (tj. metod generisanja kolona), itd.

Dinamičko programiranje je još jedna značajna programska paradigma koja se primjenjuje za rješavanje problema u oblasti operacionih istraživanja. Ova familija algoritama rješava probleme tako što eksploratiše optimalnu pod-strukturu problema. Drugim riječima, polazni problem razbijemo na



Slika 1.1: Četiri nivoa apstrakcije pri rješavanu problema operacionih istraživanja.

manje podprobleme, koje rješavamo rekurzivno. U rješavanju glavnog problema, koristimo pronađena optimalna rješenja podproblema, izbjegavajući ponovno rješavanje podproblema koji su već jednom bili rješavani. Ova programska paradigma ima značajnu ulogu u rješavanju grafovskih problema. U praksi, problemi iz oblasti operacionih istraživanja mogu biti prilično teški za egzaktno rješavanje, što podrazumijeva da je za dobijanje optimalnog rješenja potrebno jako puno vremenskih i memoriskih resursa. Za potrebe dobijanja brzog (ne obavezno optimalnog) rješenja koriste se *pohlepni algoritmi*. Oni obično ne mogu da garantuju optimalnost, ali su po svojoj prirodi jako brzi u nalaženju dovoljno dobrog rješenja. Jedan od najpoznatijih pohlepnih algoritama je *Dajkstrin* algoritam za problem pronašlaska najkraćeg

puta koji, pri određenim uslovima, nalazi i optimalno rješenje.

U nastavku ćemo detaljno objasniti svaki od nivoa apstrakcije u operacionim istraživanjima, najvažnije paradigme modelovanja i najznačajnije algoritme za rješavanje navedenih problema. Odnosi između sva četiri nivoa apstrakcije su dati na Slici 1.1. Na kraju knjige će biti riječi o opštem rješavaču, koji je u savremenoj parksi jedan od standardnih alata za rješavanje problema iz oblasti operacionih istraživanja.

Glava 2

Principi modelovanja u Operacionim istraživanjima

Uobičajeno je izraz Operaciona istraživanja zapisivati akronimom OR (eng. *operations research*). U ovoj glavi ćemo objasniti glavne faze tipičnog OR-a, koje se realizuje u komercijalne svrhe. Faze OR-a možemo sumirati u sljedeće (preklapajuće) faze:

- Definisanje problema i sakupljanje relevantnih podataka.
- Formulisanje matematičkog modela datog problema.
- Razvoj računarske procedure koja će pronaći rješenja datog problema na osnovu modela.
- Testiranje modela i, po potrebi, rad na njegovom poboljšanju.
- Pripremu za primjenu modela, kako je definisano od strane korisnika koji naručuje/finansira sistem baziran na OR.
- Implementaciju i upotrebu rješenja.

U nastavku ćemo opisati svaku od ovih faza.

2.1 Definisanje problema i sakupljanje relevantnih podataka

Većina praktičnih problema, sa kojima se susrećemo su u početku opisani na neprecizan način. Stoga, prvi zadatak podrazumijeva dobro definisanje cilja/namjene (eng. *objectives*) samog zadatka koji treba riješiti, uslova pod kojim je rješenje problema validno, vremenskih ograničenja za donošenje odluka i moguće alternativne scenarije. Definisanje problema na dovoljno dobar način je značajan korak, jer uveliko utiče na relevantnost zaključaka istraživanja. Izvlačenje odgovora ili dobijanje rezultata iz pogrešne postavke polaznog problema gotovo sigurno neće dovesti do ispravnog rješenja.

Zbog toga je važno da osoba (ili više osoba) koja modeluje problem, ne radi samostalno, već blisko sarađuje sa poslodavcem. Osoba koja je zadužena za formulisanje problema, savjetuje se sa poslodavcem za koga se razvija sistem, poslije čega se izrađuje detaljna tehnička analiza u vezi problema i izlažu preporuke za rješenje poslodavcu, koji je ključan u donošenju odluka.

Izvještaj koji se predaje poslodavcu, najčešće sadrži osnovno rješenje, ali i brojne alternative koje odgovaraju različitim uslovima ili vrijednostima parametara problema (npr. balansiranje između cijene i dobiti, stabilnosti poslovanja i dobiti/profita, itd.). Na poslodavcu je da vrednuje prijedloge, uzimajući u obzir niz faktora, nakon čega se donosi odluka zasnovanu na najboljoj procjeni.

OR istraživanje podrazumijevano traži ona rješenja koja su optimalna, ali se optimalnost može odnositi samo za određenu komponentu procesa za koji se sistem razvija. Prema tome, ciljevi koji su formulirani bi trebali da traže optimalna rješenja čitavog procesa koji se optimizuje. U praksi, ovakav zahtjev nije trivijalan, te se problem svodi na razmatranje dijela inicijalno definisanog problema (ili njegove pojednostavljene varijante). Drugim riječima, analiza bi mogla postati pretjerano glomazna ako bi zadani ciljevi bili previše uopšteni. Umjesto toga, ciljevi bi trebalo da budu specifični, koliko god je to moguće, ali da se pri tome obuhvate glavni ciljevi donosilaca odluka.

Jedan od mogućih pristupa zaobilaženja problema pod-optimizacije je

korištenje dugoročne maksimizacije nekog cilja, npr. profita. Riječ „dugoročno“ ukazuje da cilj pruža fleksibilnost u razmatranju aktivnosti koje ne donose momentalnu dobit, ali omogućuju krajnji profit. Ovakav cilj je dovoljno specifičan, a ipak i dovoljno širok da obuhvati osnovni cilj organizacija, dolazak do (zadovoljavajuće) zarade. Zapravo, svi drugi legitimni ciljevi ovakvih organizacija mogu se podvesti u prethodno pomenuti. Međutim, u praksi organizacije rijetko koriste ovaj pristup. Velik broj organizacija priлагodjava zadovoljavajući profit sa drugim ciljevima, umjesto fokusiranja na maksimizovanje dugoročne zarade. Neki od takvih ciljeva su: izgradnja stabilnog profita, povećanje udjela na tržištu, zadržavanje stabilnosti cijena, povećanje uticaja kompanije, povećanje ponude proizvoda, itd. Dalje, treba napomenuti i dodatna razmatranja, poput društvenih odgovornosti, koje nisu direktno motivisane profitom.

Postoji nekoliko strana kojih se direktno tiču poslovi firme: (1) vlasnici (dioničari, itd.), koji žele profit; (2) zaposlenici koji žele stalno zaposlenje, stabilan posao, te razumne plate; (3) kupci koji žele proizvod zadovoljavajućeg kvaliteta po razumnoj cijeni; (4) dobavljači, koji žele integritet i razumnu prodajnu cijenu proizvoda; i (5) vlada, koja želi plaćanje poreza i razmjenu dobara. Sve ove strane na određeni način utiču na poslovanje firme, ali i obrnuto. Stoga, dok je glavna odgovornost uprave profit (što, na kraju krajeva, koristi svim stranama), napomenimo da se treba sagledati i širi društveni uticaj ovakvog procesa.

Osobe zadužene za modelovanje obično provode dosta vremena priključujući relevantne podatke o problemu. Potrebno je prikupiti mnogo podataka kako bi se došlo do razumijevanja problema i dolaska do potrebnih ulaznih podataka matematičkog modela, koji se konstruiše u narednoj fazi. Mnogi podaci često nisu dostupni, bilo zbog toga što informacije nikada nisu sačuvane ili su zastarjele ili su pak sačuvane u neočekivanim formatima. Potrebno je komunicirati sa poslodavcem, kako bi pronašli sve podatke značajne za projekat. Međutim, dio podataka mogu biti prilično „neprecizan“, tj. to su grube procjene zasnovane na iskustvenim pretpostavkama. Tada je neophodno izdvojiti značajnu količinu vremena za poboljšanje preciznosti podataka, da bi se proces nastavio sa najboljim mogućim resursima i smanjila mogućnost pravljjenja grešaka.

Primjer 2.1. OR studija je urađena za policijsku upravu jednog grada za razvoj računarskog sistema za optimalno raspoređivanje i postavljanje patrolnih službenika. U procjeni ove studije, identifikovani su sljedeći glavni ciljevi:

1. Održavati visok nivo sigurnosti građana.
2. Održavati visok nivo zadovoljstva u policiji.
3. Minimizovati troškove operacija.

Da bi se postigao prvi cilj, policijska uprava i gradska vlada su zajednički radili na uspostavljanju želenog nivoa zaštite. Matematički model je razvijen na način da se postizanje potrebnog nivoa zaštite realizuje pomoću funkcije cilja. Da bi se postigao drugi cilj, u samom modelu je uključen i uslov o ravnomjerno raspoređenim radnim satima među policajcima. Treći cilj je ugrađen u model usvajanjem dugoročnog cilja smanjenja broja policajaca potrebnih za postizanje prva dva cilja.

2.2 Proces formulisanja matematičkog modela

Nakon što se definiše problem odlučivanja, sljedeća faza je formulisanje tog problema u obliku koji je odgovarajući za dalju analizu. Uobičajeni OR pristup se sastoji od konstrukcije matematičkog modela koji na najbolji način odgovara realnom problemu. Prije nego što objasnimo opšti proces konstruisanja takvog modela, recimo nešto o modelima uopšte, a posebno o matematičkim modelima.

U izvornom značenju, model je oblik (može biti i prikaz, tip ili ponasanje) koji se uzima za primjer, koji se oponaša ili na koji se neko ugleda. Model se može definisati i kao uzorak po kojem se izrađuje neki proizvod. U svakodnevnom životu se veoma često susrećemo sa različitim modelima. Tipični primjeri uključuju model aviona, ljudskog lica (portret), zemaljske kugle (globus) itd. Slično tome, modeli igraju važnu ulogu u nauci i poslovanju, što uključuje modele atoma, modele genetskih struktura, matematičke jednačine koje opisuju fizičke zakone kretanja, grafikoni, itd. Takvi

modeli su od neprocjenjive važnosti za apstraktno poimanje suštine predmeta istraživanja, ukazivanja na međusobnu interakciju pojava i olakšavanja kompleksnih analiza.

Matematički model je približni opis nekih pojava ili objekata u stvarnom svijetu, uz pomoć matematičke simbolike (funkcija, jednačina itd.). Npr. drugi Njutnov zakon se matematički modeluje poznatom formulom $F = ma$. Matematički model poslovnih problema se sastoji od sistema jednačina i matematičkih izraza koji opisuju osnov problema. Npr. postoji n mjerljivih odluka koje treba odrediti, a koje su predstavljene pomoću nepoznatih promjenljivih x_1, \dots, x_n . Odgovarajuća mjera učinka (profita) je izražena preko matematičkih funkcija nepoznatih koje odgovaraju odlukama ($3x_1 + 5x_2 + x_3$). Ova funkcija se naziva *ciljna* (objektivna) funkcija, ili *funkcija cilja*. Ograničenja vrijednosti promjenljivih se takođe izražavaju matematičkim izrazima, (npr. $x_1 + x_2 + x_3 \leq 1$). Ovakvi matematički izrazi za ograničenja se tako i nazivaju – *ograničenja* (eng. *constraints*). Konstante u ograničenjima, te u ciljnoj funkciji, se nazivaju *parametri* modela.

Prema tome, rješavanje matematičkog modela podrazumijeva izbor vrijednosti promjenljivih odlučivanja, tako da se maksimizuje/minimizuje vrijednost funkcije cilja u odnosu na određena ograničenja. Takvi modeli, sa manjim varijacijama, karakterišu i modele koji su prisutni u OR oblasti. Određivanje odgovarajućih vrijednosti, koje se dodjeljuju parametrima modela, je kritični dio procesa konstrukcije modela.

Za razliku od problema iz udžbenika, za koje su ove vrijednosti unaprijed poznate, određivanje parametara u stvarnim situacijama najprije zahtijeva prikupljanje relevantnih podataka. Kao što je objašnjeno u prethodnoj sekciji, prikupljanje tačnih podataka je veoma izazovan proces. Dodijeljena vrijednost parametru često predstavlja samo grubu procjenu stvarne situacije. Zbog nepouzdanosti vrijednosti parametara, važno je analizirati kako se rješenje, koje je izvedeno iz modela, mijenja pri malim promjenama vrijednosti parametara. Problemi iz prakse obično nemaju samo jedan „ispravan” model. Često dva ili više različito formulisana modela mogu da odgovaraju jednom problemu. Ovo je stvar razvoja modela koji je iterativni proces koji proizlazi iz rezultata testiranja i potrebe za konstrukcijom boljeg modela.

Postoji i napisano pravilo da nijedan matematički model koji opisuje

realnu situaciju nije potpuno tačan, pa je cilj konstruisati onaj model koji, što je moguće preciznije, opisuje tu realnu situaciju.

U nastavku ćemo vidjeti brojne primjere matematičkih modela. Jedna posebno važna vrsta modela koju proučavamo u sljedećih nekoliko poglavlja je model linearнog programiranja, gdje su i ciljna funkcija i ograničenja zadati linearnim funkcijama. Matematički modeli imaju još jednu prednost u odnosu na jezički opis problema – problem se opisuje mnogo sažetije. Na taj način, cjelokupna struktura problema je razumljivija, a otkrivanje važnih uzročno-posljedičnih veza u samom problemu je dosta lakše. Takođe, matematički model čini most za primjenu moćnih matematičkih tehniki i računara u analizi i rješavanju problema. Zapravo, računarski programi za rješavanje raznih tipova modela su u današnje vrijeme široko dostupni (na primjer, poznati rješavači kao što su CPLEX, Gurobi, Lindo itd.).

Postoje i „zamke”, na koje treba paziti, kada koristimo matematičke modele. Ako želimo da, na kraju, formirani model bude upotrebljiv za rješavanje problema, često su neophodne aproksimacije i pojednostavljanja pretpostavki. Stoga se mora paziti da model ostane valjan prikaz problema nakon primjene svih pojednostavljenja. To znači da treba da postoji značajna korelacija između onoga što model predviđa i onoga što bi se zapravo i dobija primjenom modela u stvarnoj situaciji. Da bi se utvrdilo da li je ovaj zahtjev zadovoljen, potrebno je izvršiti značajna ispitivanja i posljedične izmjene u inicijalnom modelu. O ovome ćemo govoriti u narednim sekcijama. U osnovi, provjera valjanosti modela zapravo se provodi tokom svih faza izrade modela.

Preporučljiv pristup u razvoju modela je da se započne sa pojednostavljenom verzijom, a evolucionim načinom ići prema složenijim modelima, koji će bolje odražavati stvarni problem. Ovaj proces obogaćivanja modela se nastavlja dok god je model rješiv. Kompromis, kojeg se pridržavamo, je odnos između preciznosti modela i mogućnosti njegovog rješavanja. Ključni korak u razvoju modela je konstrukcija funkcije cilja. On podrazumijeva razvijanje kvantitativne mjere uspješnosti u odnosu na svaki krajnji cilj, koji je identifikovan tokom definisanja problema. Ako postoji više ciljeva, njihove se mjere obično transformišu i kombinuju u jednu složenu mjeru – ali postoje i pristupi koji nezavisno analiziraju funkcije cija, tzv. visekrite-

rijumska optimizacija. Ukupna mjera može biti nešto opipljivo (npr. dobit), što odgovara višem cilju organizacije. Nakon što se razvije cijelokupna mjera, ciljna funkcija se izražava kao matematička funkcija čiji su argumenti promjenljive odlučivanja.

Primjer 2.2. Uzmimo jednog proizvođača negaziranih pića. On pravi dvije vrste soka: multivitaminski gusti i multivitaminski bistri sok, od sastojaka A i B , te vode. Da bi se napravilo $100l$ multivitamininskog gustog soka potrebno je $3l$ sastojka A i $8l$ sastojka B , a za bistri sok je potrebno $6l$ sastojka A i $4l$ sastojka B . Sto litara gustog soka nosi zaradu od 100KM , a $100l$ bistrog zaradu od 125KM . Proizvođač u skladištu ima na raspolaganju $30l$ sastojka A i $44l$ sastojka B . Napravljene sokove sipa u burad. Za gusti sok ima bure u koje stane $500l$, a za bistri sok bure kapaciteta $400l$. Cilj proizvođača sokova je odrediti količinu soka kojeg treba napraviti da bi se maksimizovala profit, uzimajući u obzir data ograničenja. Drugim riječima, treba konstruisati plan proizvodnje u cilju maksimizacije profita.

Model za ovaj problem bi se mogao konstruisati na sljedeći način:

- Promjenljive odlučivanja: x_1 i x_2 koje označavaju koliko stotina litara gustog soka i koliko litara bistrog soka proizvođač treba da napravi, respektivno.
- Funkcija cilja: $f(x_1, x_2) = c_1x_1 + c_2x_2$, gdje su $c_1 = 100$ i $c_2 = 125$ koeficijenti. Funkciju f treba maksimizovati.
- Ograničenja: $0 \leq x_1 \leq 5$, $0 \leq x_2 \leq 4$, $3x_1 + 6x_2 \leq 30$ i $8x_1 + 4x_2 \leq 44$.

Prva dva ograničenja se odnose na kapacitet buradi za smještanje gustog, odnosnog bistrog soka. Preostala dva ograničenja su vezana za količinu raspoloživih resursa, odnosno sastojaka A i B , potrebnih za proizvodnju sokova.

U narednim poglavljima knjige ćemo detaljno razmatrati postupke kako se dolazi do rješenja ovakvih modela.

2.3 Izvođenje rješenja iz modela

Nakon formulacije matematičkog modela za definisani problem, sljedeća faza u OR procesu je razvoj postupka (algoritma) za izvođenje rješenja na osnovu konstruisanog modela. Nekada je to relativno jednostavan korak, u kojem se na računaru izvršava neki od standardnih algoritama u okviru odgovarajućih računarskih paketa/rješavača matematičkog modela. Sa druge strane, nekada su matematički modeli toliko složeni da se ne mogu riješiti direktnom primjenom standardnih algoritama, već se podrazumijeva razvoj i primjena drugih, sofisticiranih tehnika. Nakon dobijanja rezultata, potrebno ih je analizirati. Postupak analize rezultata ćemo, takođe, objasniti u nastavku udžbenika.

Uobičajeni zadatak u OR oblasti je potraga za optimalnim (najboljim) rješenjem i razvoj odgovarajućih postupaka koji služe za pronađenje optimalnih rješenja zadatog modela. Treba imati na umu da su ta rješenja optimalna samo za dati model. Kako model ne mora da u potpunosti odgovara stvarnom problemu, ne postoji garancija da će se pronađeno optimalno rješenje modela pokazati i najboljim mogućim rješenjem za stvarni problem koji se razmatra. Međutim, ako je model dobro formulisan i testiran, rezultujuće rješenje bi trebalo da bude prihvatljivo. U praksi se često pokaže da je „dovoljno dobro“ rješenje čak prihvatljivije nego trošenje nerazumno mnogo vremena za traženje optimalnog.

Najčešće se prvo postavi lista zahtjeva, koje dati model treba da ispunii, a rješenje, koje zadovoljava te uslove (zahtjeve), najčešće se uzima kao prihvatljivo. U procesu donošenja odluka, pokušava se što više koristiti naučni pristup, sa primarnim ciljem pronalaska globalnog optimuma za posmatrani model. Međutim, uslov da se u razumnom vremenu dođe do modela koji generiše rješenje zadovoljavajućeg kvaliteta, najčešće ima prednost u odnosu na pronalazak optimuma u nerealno dugom vremenskom roku. Osoba koja rukovodi procesom treba da razmotri troškove procesa razvoja modela, a zatim pokuša da maksimizuje korist u oba smjera. Povremeno se koriste približni (heuristički) pristupi rješavanju problema. Ovakvi pristupi su intuitivno dizajnirani postupci kojima se ne garantuje dobijanje optimalnog rješenja. Posljednjih nekoliko decenija postignut je velik napredak u razvoju

efiksnih heurističkih postupaka (uključujući tzv. metaheuristike o kojima će više biti riječi u Glavi 9), a njihova upotreba u ovom polju iz dana u dan sve više dobija na značaju.

Kao što je već nekoliko puta pomenuto, i optimalno rješenje za konstruisani model može biti daleko od idealnog za stvarni problem, pa su potrebne dodatne analize. Stoga je analiza „postoptimalnosti“ (analiza provedena nakon pronalaska optimalnog rješenja) vrlo važan dio OR studija. Ova analiza se ponekad naziva i *šta-ako analiza* (eng. what-if analysis), jer uključuje postavljanje pitanja o tome što bi bilo sa optimalnim rješenjem ako se uvedu različite pretpostavke, što implicira dodavanje novih ograničenja u postojeći model. Ova pitanja često postavlja poslodavac koji donosi konačne odluke u vezi procesa dodavanja uslova.

Pojava softvera koji rade sa proračunskim tabelama (prvenstveno Excel-a) odigralo je centralnu ulogu u provođenju analize postoptimalnosti. Jedna od velikih prednosti ovakvih softvera je jednostavnost u korišćenju, gdje se lako dobija analiza šta se događa sa optimalnim rješenjem kada se naprave promjene na modelu. Ovakva eksperimentisanja sa promjenama u modelu mogu biti od velike pomoći u razumijevanju ponašanja samog modela i uvjerenja u njegovu valjanost. Analiza postoptimalnosti dijelom uključuje provođenje analize osjetljivosti, radi utvrđivanja koji parametri modela su najkritičniji („osjetljivi parametri“) u određivanju rješenja. Uobičajena definicija osjetljivog parametra je sljedeća:

Osjetljivi parametri matematičkog modela su oni parametri čija (čak i mala) promjena direktno utiče na promjenu optimalnog rješenja.

Prepoznavanjem osjetljivih parametara pronalazimo one parametre čija se vrijednost mora odrediti sa posebnom pažnjom, kako bi se izbjeglo наруšavanje rezultata modela. Da napomenemo, vrijednost koja se dodjeljuje parametru obično predstavlja samo procjenu neke veličine (npr. jedinična dobit ili vrijeme potrebno da se neka aktivnost izvrši) čija će vrijednost postati preciznija tek nakon što se rješenje primijeni na (stvarni) problem. Stoga, nakon što se identifikuju osjetljivi parametri, posebna pažnja se posvećuje bližoj procjeni svakog parametra, te rasponu njihovih mogućih vrijednosti. U nekim slučajevima, određeni parametri modela predstavljaju interesne odluke (npr. dodjele resursa). Ako je to slučaj, često postoji odre-

đena fleksibilnost u vrijednostima koje se dodjeljuju ovakvim parametrima u smislu da se neke vrijednosti povećavaju smanjenjem drugih. Postoptimalna analiza uključuje i istraživanje takvih kompromisa. Analiza postoptimalnosti takođe uključuje i dobijanje niza rješenja koji aktivira niz poboljšanja u modelu i konvergenciju prema idealnom modelu. Postupak poboljšanja modela se nastavlja sve dok poboljšanja u sljedećim rješenjima ne postanu zanemariva da bi se nastavilo sa ovakvim iterativnim procesom. Alternativna rješenja (rješenja koja su optimalna za jedan od nekoliko približnih modela) se takođe mogu predstaviti poslodavcu za konačan odabir.

2.4 Testiranje modela

Razvoj velikih matematičkih modela je analogan razvoju velikih softvera. U praksi se često dešava da se, po završetku izrade prve verzije softvera, javi i određen broj grešaka (bagova). Zbog toga se pristupa raznim procedurama testiranja proizvoda, kako bi se pokušalo pronaći i ispraviti što više grešaka. Na kraju, nakon niza iteracija poboljšanja programa, programeri mogu da zaključe da trenutni program sada generalno daje valjane rezultate sa velikom sigurnošću. Može se desiti da neke manje greške ostanu skrivene u programu, te se možda nikada i neće ukloniti. Slično tome, prva verzija matematičkog modela problema neizbjježno sadrži određene nedostatke. Neki relevantni faktori ili međusobni odnosi i zavisnosti između elemenata modela često ne budu ispravno ugrađeni u model, a neki parametri ne budu pravilno procijenjeni. Iako su nepoželjne, ove pojave ne treba shvatati kao fatalne greške, već kao očekivanu pojavu. One mogu biti uzrokovane problemima u komunikaciji, izazovima u razumijevanja svih aspekata problema, te poteškoćama pri prikupljanju pouzdanih podataka. Stoga, prije nego što upotrijebimo model, on se mora temeljno testirati kako bismo mogli identifikovati i ispraviti što više njegovih nedostataka. Ovaj postupak ispitivanja i poboljšanja modela radi povećanja njegove valjanosti se naziva *validacija modela*.

Budući da osobe koje razvijaju model mogu provesti mjesecce na izradi svih njegovih detalja, može se desiti i da dođe do odstupanja od suštine

modela. Stoga, nakon što su detalji uključeni u početnu verziju modela, dobar način za početak provjere valjanosti je ponovni pogled na cijelokupni model kako bi se provjerilo da li model sadrži očigledne greške i propuste. U cilju što objektivnijeg pristupa, poželjno je da grupa koja radi ovaj pregled uključi barem jednog pojedinca koji nije učestvovao u konstrukciji modela. Preispitivanje definicije problema i poređenje sa modelom mogu pomoći u otkrivanju nekih grešaka. Ponekad se dodatni uvid u valjanost modela dobija i mijenjanjem vrijednosti parametara i/ili promjenljivih odlučivanja, kao i provjerom da li se izlaz iz modela ponaša na očekivan način. To se često može otkriti kada se parametrima ili promjenljivima dodjeljuju ekstremne vrijednosti u blizini njihovih maksimuma ili minimuma (takozvani „korner slučajevi“ – eng. *corner cases*).

Sistematski pristup u ispitivanju modela podrazumijeva upotrebu retrospektivnog testa. Ovaj test uključuje upotrebu istorijskih podataka koji prate konstrukciju modela, a zatim procjenjuje koliko bi dobro model i rezultujuće rješenje funkcionalisali, ako bi ono bilo korišteno u praksi. Ovaj pristup može da ukaže na dijelove modela koji sadrži određene nedostatke, što bi dalje zahtijevalo dodatne izmjene i poboljšanja. Sa druge strane, nedostatak retrospektivnog testa je taj što koristi iste podatke kao i oni koji su korišteni za formulaciju modela. Važno je još dokumentovati postupke testiranja valjanosti modela. Ovo može povećati povjerenje budućih klijenata u model. Takođe, ako se u budućem vremenu pojave pitanja u vezi sa modelom, ova dokumentacija će biti korisna prilikom otkrivanja potencijalnih problema.

2.5 Pripremanje modela za primjenu

Ako želimo da model bude upotrijebljen više puta u praksi, sljedeći korak je instaliranje dobro dokumentovanog sistema za primjenu modela po propisima poslodavca. Ovaj sistem uključuje implementaciju samog modela, postupak dobijanja rješenja (uključujući analizu postoptimalnosti), operativne postupke za izvođenje, itd. Svaki put pri korištenju sistema, baze podataka i informacioni sistem za upravljanje treba da omoguće unos podataka u model, te je zbog toga potrebno obezbijediti i odgovarajući interfejs.

Nakon što se na model primijeni postupak rješavanja, dodatni programi mogu automatski pokrenuti implementaciju rezultata kako bi ih korisnici mogli odmah primijeniti u realnoj situaciji. U drugim slučajevima, može se instalirati i interaktivni računarski sistem za podršku odlučivanju, koji pomaže poslodavcu da koristi podatke i modele kako bi, po potrebi, podržao (ali ne i zamijenio) donošenje odluka. Dodatna programska rješenja mogu generisati menadžerske izvještaje, koji pomažu pri tumačenju izlaznih podataka modela, kao i implikacije na primjenu u realnom okruženju. U velikim OR studijama potrebno je nekoliko mjeseci (a nekada čak i duže) za razvoj, testiranje i instaliranje ovakvog računarskog sistema. Pri tome, treba uzeti u obzir da ove aktivnosti uključuju i razvoj i izvođenje postupka održavanja sistema tokom njegove buduće upotrebe. Kako se uslovi obično mijenjaju tokom vremena, potrebno je obezbijediti i mogućnost za izmjene sistema, uključujući i izmjene samog modela, u skladu sa tim uslovima.

Sljedeći primjer nam opisuje sistem u IBM-u za kontrolu usluga i zaliha rezervi da bi se dobio osjećaj o kompleksnosti ovakvih sistema zasnovao na OR studiju.

Primjer 2.3. Sistem nazvan *Optimizer* pruža optimalnu kontrolu usluga i zaliha rezervnih dijelova u cijeloj IBM-ovoj mreži za distribuciju dijelova u SAD-u. Ona uključuje dva središnja automatizovana skladišta, desetine terenskih distribucijskih centara i stanica za dijelove, te nekoliko hiljada vanjskih (povezanih) lokacija. Vrijednost inventara dijelova koji se održava u ovoj mreži se procjenjuje u milijardama dolara. *Optimizer* se sastoji od četiri glavna modula. Modul sistema predviđanja sadrži nekoliko programa za procjenu stope kvara pojedinih vrsta dijelova. Modul sistema za dostavljanje podataka sastoji se od otprilike 100 programa koji obrađuju preko 15 GB podataka kako bi se osigurali ulazni podaci za model. Sistem odlučivanja potom rješava model kako bi se optimizovala kontrola zaliha. Rješavanje ovog modela može da traje i po sedmicu dana. Četvrti modul uključuje šest programa koji integriraju *Optimizer* u IBM-ov sistem upravljanja zalihami dijelova.

2.6 Implementacija

Nakon što se razvije sistem za primjenu modela, posljednja faza podrazumijeva upotrebu ovog sistema na način kako je to propisao poslodavac, kao naručilac ovakvog sistema. Ova faza je ključna jer se ovdje „ubiru plodovi rada” sa obje strane. Stoga je važno da osobe zadužene za izradu sistema učestvuju u realizaciji ove faze, kako bi se osiguralo da se rješenja modela precizno prevedu u operativni postupak, ali i da bi se otkrili i ispravili svi nedostaci dobijenih rješenja. Uspjeh faze implementacije uveliko zavisi i od podrške poslodavca i uprave. Više je vjerovatno da će osobe uključene u razvoj sistema dobiti podršku ako su dobro informisli poslodavca i podsticli njegovo aktivno učestvovanje tokom razvoja rješenja; dobra komunikacija pomaže da istraživanje postigne ono što je poslodavac i želio. Faza implementacije uključuje nekoliko koraka. Prvo, operativnom rukovodstvu se daje opis novog sistema koji treba da bude usvojen za korištenje. Dalje, obje strane, i poslodavac i razvojni tim, dijele odgovornost za razvijanje koraka potrebnih za pokretanje sistema. Operativno rukovodstvo zatim nadgleda osoblje koje je uključeno u proces implementacije, te mu daje detaljna uputstva za korištenje sistema. Time počinje proces primjene sistema u realnim uslovima. Nakon uspješnog uvođenja sistema, očekuje se njegovo aktivno korištenje u narednom periodu. Imajući to na umu, određene osobe iz razvojnog tima treba da nadgledaju iskustva korisnika sistema radi utvrđivanja koje izmjene u budućnosti bi trebalo uraditi u sklopu sistema u cilju njegovog lakšeg korištenja. Tokom vremenskog perioda u kojem se koristi novi sistem, važno je i dalje dobijati povratne informacije o funkcionisanju sistema, te provjeravati da li su pretpostavke o modelu i dalje zadovoljene. Kada se pojave značajna odstupanja od originalnih pretpostavki modela, model treba ponovno pregledati i eventualno izvršiti korekcije. Nakon završetka projekta, potrebno je dovoljno jasno i precizno dokumentovati metodologiju izrade cjelokupnog sistema, kako bi se čitav posao mogao ponoviti. Repliciranje bi trebalo biti dio profesionalnog i etičkog kodeksa projektovanja (informacionih) sistema.

Primjer 2.4. Vratimo se na realni IBM projekat pomenut u Primjeru 2.3.

Tri su se glavna faktora pokazala posebno važnim za uspješan završetak projekta. Prvi je uključivanje korisničkog tima i (operativnog) rukovodstva koji su savjetovali razvojni tim tokom trajanja projekta. U vrijeme faze implementacije, operativno rukovodstvo imalo je snažan uticaj i naglašavana je važnost instaliranja *Optimizer-a* u funkcionalna područja poslovanja IBM-a. Drugi faktor uspjeha je dobijanje povratne informacije većeg broja korisnika sistema. To je pomoglo u identifikovanju problematičnih tačaka u sistemu, koje su ispravljene prije potpune implementacije. Treći faktor je taj da je novi sistem uvođen postepeno, fazu po fazu, uz pažljivo testiranje u svakoj fazi, kako bi se uklonile veće greške, prije nego što je sistem pušten za upotrebu na globalnom nivou.

Ostatak ove knjige je fokusiran na matematičke modele i odgovarajuće metode za njihovo rješavanje. Ovim poglavljem smo objasnili, a kroz neke primjere i ilustrovali, da konstrukcija modela složenih problema ne sadrži samo ova dva koraka, već su oni dio cijelokupnog OR procesa. Treba imati na umu da su sve faze vrlo važne za provođenje uspješnog OR projekta. Predlažemo da se nakon izučavanja algoritama za rješavanja modela, čitalac vrati na ovo poglavlje u cilju što jasnije slike razvoja potpunog OR projekta zasnovanog na kompleksnim procesima u praksi. OR zahtijeva značajnu količinu domišljatosti, saradnje i komunikacije, kreativnosti i inovativnosti, tako da u praksi i ne postoji jedan standardni postupak kojeg bi se timovi trebali pridržavati pri razvijaju sistema zasnovanih na operacionim istraživanjima.

Glava 3

Osnove linearнog programiranja

Razvoj linearнog programiranja poчиње sredinom dvadesetog vijeka. Sovjetski matematičar i ekonomista Leonid Kantorovič je tokom Drugog svjetskog rata formulisao problem ekvivalentan problemu linearнog programiranja u cilju boljeg upravljanja resursima i smanjenja vojnih troškova. Neposredno nakon Drugog svjetskog rata, američki matematičar Džordž Dencig razvija čuveni simpleks algoritam, čime se otvara prostor za efikasno rješavanje problema linearнog programiranja. U periodu koji je uslijedio, pa sve do danas, linearно programiranje se koristi kao standardan alat za rješavanje raznih problema u industriji i ekonomiji uopšte, što je dovelo do milionskih ušteda ogromnom broju kompanija širom svijeta. Stoga se razvoj linearнog programiranja smatra jednim od najvažnijih naučnih dostignuća XX vijeka. Napisane su stotine udžbenika o linearном programiranju, kao i hiljade naučnih radova, koji ovu paradigmu primjenjuju na probleme iz raznih naučnih i praktičnih domena.

Linearno programiranje koristi matematički model za opisivanje problema. Pridjev „linearno“ podrazumijeva da su matematičke funkcije korištene u modelu linearne funkcije. Riječ „programiranje“ se ne odnosi na računarsko programiranje, već je to sinonim za planiranje i optimizaciju. Dakle,

linearno programiranje uključuje planiranje aktivnosti u cilju postizanja optimalnog rezultata, tj. rezultata koji najbolje postiže navedeni cilj (prema matematičkom modelu) među svim mogućim alternativama. Za rješavanje problema linearног programiranja razvijeno je dosta efikasnih metoda. Iako jedan od nastarijih, *simpleks metod* se i danas koristi za rješavanje problema velikih dimenzija, te je zaslužan za ogroman uticaj linearног programiranja u raznim oblastima nauke i privrede.

Primjer 3.1. U ovom primjeru opisujemo jedan industrijski projekat naziva WYNDOR-GLASS CO projekat, ilustrujući kako OR može da, upotrebom linearног programiranja, riješi ozbiljne probleme u industriji.

Firma WYNDOR-GLASS proizvodi visokokvalitetne staklene proizvode, uključujući prozore i staklena vrata. Postoje tri pogona. Aluminijski okviri i okovi izrađuju se u pogonu 1, drveni okviri izraђuju se u pogonu 2, a pogon 3 proizvodi staklo i sastavlja proizvode. Zbog pada zarade, rukovodstvo firme odlučilo je da izmijeni liniju proizvoda koje nudi. Neprofitabilni proizvodi se ukidaju, oslobađajući proizvodne kapacitete za lansiranje dva nova proizvoda sa velikim prodajnim potencijalom:

- Proizvod 1: Staklena vrata od $2m$ sa aluminijskim okvirom;
- Proizvod 2: Dvostruka drvena vrata sa okvirom visine $1.5m$.

Uslovi i zahtjevi proizvodnje su sljedeći:

- Proizvod 1 zahtijeva korištenje proizvodnih kapaciteta u pogonima 1 i 3, ali ne koristi postrojenje 2;
- Proizvod 2 za proizvodnju koristi pogone 2 i 3.

Rukovodstvo kompanije je došlo do zaključka da bi kompanija mogla prodati sve proizvode koje je u mogućnosti da proizvede u svojim pogonima. Međutim, budući da oba proizvoda koriste proizvodni kapacitet u postrojenju 3, nije očigledno jasno koja bi kombinacija proizvodnje dva proizvoda bila najisplativija.

Nakon razgovora sa upravom, utvrđeni su glavni ciljevi istraživanja. Definicija (idealizovanog) problema koji treba da se riješi je sljedeća:

-
- Utvrditi stope proizvodnje ova dva proizvoda kako bi se maksimizovala ukupna dobit, koja je podložna ograničenjima u proizvodnim kapacitetima koji su dostupni u sva tri pogona.
 - Svaki proizvod se proizvodi u serijama od po 20 komada, tako da se stopa proizvodnje definiše kao broj serija proizvedenih na nivou jedne sedmice.
 - Dopuštena je bilo koja kombinacija stope proizvodnje koja zadovoljava ograničenja, uključujući i slučaj da se ne proizvede niti jedan od (dva) proizvoda kao i slučaj da se proizvede maksimalan broj jednog (od dva) proizvoda na osnovu utvrđenih kapaciteta.

U cilju razvijanja modela, neophodno je prikupiti sljedeće informacije:

- (i) Broj radnih sati u sedmici dostupan u svakom pogonu za proizvodnju proizvoda.
- (ii) Broj radnih sati koji je neophodno utrošiti za proizvodnju jedne serije proizvoda u svakom pogonu.
- (iii) Dobit po proizvedenoj seriji svakog novog proizvoda. Odlučeno je da je dobit svake dodatne proizvedene serije približno konstantna, bez ozbira na ukupan broj proizvedenih serija.

Budući da značajniji troškovi za pokretanje proizvodnje i promovisanje novih proizvoda nisu postojali, ukupna dobit je aproksimirana dobiti po proizvedenoj seriji koja je pomnožena sa brojem serija proizvodnje.

Dalje, trebalo je procijeniti konstante (parametre) u modelu; ovo je ostvareno u komunikaciji između uprave firme i tima zaduženog za rješavanje problema. Pomoć stručnog osoblja iz raznih odjela firme je odigrala ključnu ulogu u dobijanju razumnih procjena za konstante. Osoblje u proizvodnom odjelu je obezbijedilo informacije o podacima za stavku (i). Za procjenu slučaja (ii), ključna je bila analiza proizvodnih inženjera koji su uključeni u dizajniranje proizvodnih procesa novih proizvoda. Analizirajući podatke o resursima i uključivanjem marketinškog odjela firme, data je procjena za slučaj (iii). Tabela 3.1 pokazuje sve ove procjene.

		Vrijeme proizvodnje serije (h)		
		Proizvod		
Pogon		1	2	Vrijeme za proizvodnju po sedm. (h)
1		1	0	4
2		0	2	12
3		3	2	18
Profit po seriji		3000	5000	

Tabela 3.1: Procjene konstanti modela.

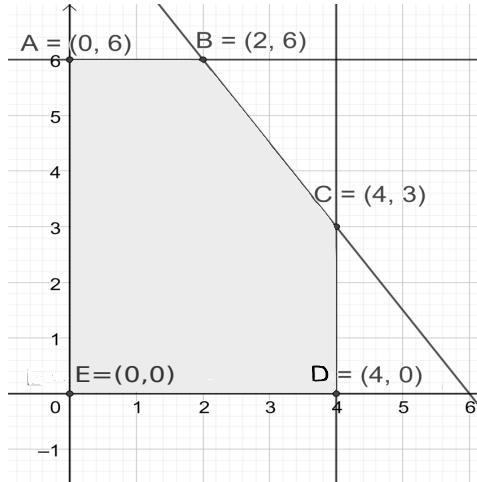
U nastavku konstruišemo model problema. Uvedimo promjenljivu x_1 za broj serija proizvoda 1 proizведенog na sedmičnom nivou, te x_2 broj serija proizvoda 2 koji je proizведен na sedmičnom nivou. Funkcija cilja je jednaka $f(x_1, x_2) = 3000x_1 + 5000x_2$, dok se rješenja traže pod ograničenjima

- $x_1, x_2 \geq 0$
- $x_1 \leq 4, 2x_2 \leq 12$
- $3x_1 + 2x_2 \leq 18$

Dakle, na osnovu realnog problema, formirali smo odgovarajući matematički model (idealizovanog problema) koji sadrži linearnu funkciju dvije promjenljive i nekoliko ograničenja, koja su zapisana preko linearnih nejednačina. Cilj rješavanja ovog matematičkog modela je pronalaženje onih vrijednosti promjenljivih x_1 i x_2 koji se uklapaju u ograničenja i za koje posmatrana funkcija cilja ima maksimalnu vrijednost.

3.1 Grafički metod

Problem iz prethodne sekcije je malih dimenzija jer sadrži samo dvije promjenljive odlučivanja. Samim tim, moguća rješenja mogu biti predstavljena u dvije dimenzije. U ovom slučaju možemo koristiti *grafički metod* za rješavanje problema. Postupak uključuje rad u dvodimenzionalnom Dekartovom koordinatnom sistemu sa osama x_1 i x_2 . Prvi korak je identifikovati vrijednosti (x_1, x_2) koje zadovoljavaju data ograničenja, odnosno, treba



Slika 3.1: Dopustiv region.

odrediti skup *dopustivih rješenja*. U narednoj sekciji ovog poglavlja ćemo preciznije definisati navedeni termin, ali je, u ovom trenutku, pod ovim skupom dovoljno smatrati ona rješenja koja zadovoljavaju sva ograničenja modela problema. Predstavimo grafički svako od ograničenja, odakle dobijamo oblast koja uključuje svih 5 ograničenja, vidjeti Sliku 3.1. Rezultujući region rješenja (x_1, x_2) je *dopustivi region* ovog problema.

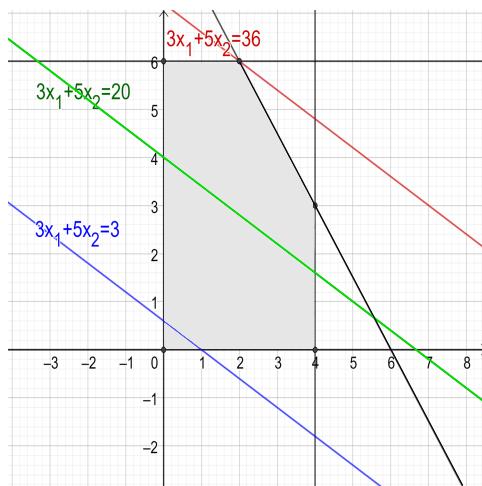
Posljednji korak se sastoji u odabiru tačke u ovom regionu koja maksimizuje vrijednost (ciljne) funkcije $f = 3x_1 + 5x_2$.

Pogledajmo sada niz (implicitnih) funkcija $3x_1 + 5x_2 = c$, za $c \in \{3, 20, 36\}$, kao na Slici 3.2. Jasno se vidi da prava $f = 36$ prolazi kroz tačku (vrh) regiona $(2, 6)$, koja je svakako dopustiva. Takođe, vidimo da niti jedna druga funkcija $f = c$, $c > 36$ ne presijeca dopustiv region, odakle zaključujemo da je tačka $(2, 6)$ optimalno rješenje problema. Dakle, funkcija cilja ima maksimalnu vrijednost 36 i dostiže se u tački $(x_1, x_2) = (2, 6)$.

Ovaj se postupak često naziva *grafičkom metodom za linearno programiranje*. Ona se može koristiti za rješavanje bilo kojeg problema linearног programiranja sa dvije promjenljive. Uz dodatni angažman, ovu metodu je

moguće proširiti na problem sa tri promjenljive, što je krajnji domet grafičke metode. Za uopšten slučaj se koriste naprednije metode, o kojima će biti riječi u narednim sekcijama.

3.2 Standardna forma linearног programiranja



Slika 3.2: Dodavanje $f = c$ i nalazak optimuma transliranjem po dopustivom regionu.

U ovoj sekciji govorimo o standardnoj formi modela linearног programiranja. Uvedimo neke ključne pojmove koji se obično koriste u modelovanju i rješavanju problema linearног programiranja.

Podimo od pojmove *resursi* i *aktivnosti*. Prepostavimo da u našem razmatranju imamo m resursa i n aktivnosti. Neki standardni resursi su vrijeme, novac, oprema, mašine, radnici itd. Aktivnosti uključuju ulaganje novca u određene projekte, dodjelu posla nekoj mašini ili radniku, dostavljanje robe iz jednog u drugo mjesto, itd. Najčešći tip primjene linearног programiranja uključuje povezivanje resursa i aktivnosti, odnosno dodjelu resursa određenim aktivnostima. Dostupna količina svakog resursa je naj-

češće ograničena, pa aktivnosti trebaju biti pažljivo raspoređene u odnosu na odgovarajuće resurse. Utvrđivanje ove raspodjеле uključuje odabir vrsta i nivoa aktivnosti za koje se postiže najbolja moguća vrijednost funkcije cilja. Funkcija cilja može da predstavlja mjeru dobiti (profita), mjeru vremena realizacije aktivnosti, mjeru broja proizvedenih proizvoda, itd.

U nastavku navodimo notaciju koja se najčešće koriste u formiranju opšteg oblika modela linearног programiranja (LP):

- f : funkcija cilja (dobiti)
- x_j : količina aktivnosti j ($j = 1, \dots, n$);
- c_j : povećanje u profitu ako se količna aktivnosti j poveća za jedan;
- b_i : ukupna količina resursa i ($i = 1, \dots, m$) koja je dozvoljena za pridruživanje aktivnostima;
- a_{ij} : količina resursa i koja se uzima za aktivnost j .

Prema ovoј notaciji, matematički model problema linearног programiranja ima sljedećу postavku:

$$\max c_1x_1 + \dots + c_nx_n \tag{3.1}$$

t.d.

$$a_{i1}x_1 + a_{i2}x_2 + \dots + a_{in}x_n \leq b_i, \quad i = 1, \dots, m \tag{3.2}$$

$$x_i \geq 0, \quad i = 1, \dots, n. \tag{3.3}$$

Postavka (3.1)–(3.3) se još naziva i *standardna forma* problema linearног programiranja. Funkcija (3.1) se naziva funkcija cilja koju je potrebno maksimizovati pod uslovima (3.2), koji se nazivaju *funkcionalna ograničenja*. Ograničenja (3.3) se nazivaju *nenegativna ograničenja*.

Kompaktiniji oblik standardne forme LP-a je dat sa:

$$\max c^T x \quad (3.4)$$

t.d.

$$Ax \leq b \quad (3.5)$$

$$x \geq 0, \quad (3.6)$$

gdje je matrica $A \in \mathbb{R}^{m \times n}$, vektori $x, c \in \mathbb{R}^n$ i vektor $b \in \mathbb{R}^m$.

Treba imati na umu da se ograničenja (3.5) mogu zapisati i na sljedeći način:

- $Ax \geq b;$
- $Ax = b;$
- neke od promjenljivih $x_i \leq 0$.

Bilo koji problem koji koristi neke od prethodna tri ograničenja predstavlja i dalje problem linearog programiranja. Dodavajući tzv. *izravnavaajuće promjenljive* (eng. slack variables) $s \in \mathbb{R}^m$ u linearni program sa ograničenjima $Ax \leq b$, dobijamo linearni program sa ograničenjima $Ax + s = b, s \geq 0$. Isto tako, linearni program sa ograničenja $Ax \geq b$ se može transformisati u ograničenja sa jednakostima dodavanjem izravnavajućih promjenljivih sa $Ax - s = b, s \geq 0$.

Forma linearnog programa sa ograničenjima $Ax = b$ se naziva *kanonska forma* linearog programiranja.

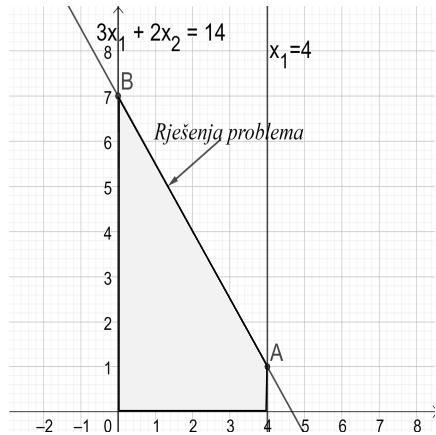
Takođe, ako u linearnom programu postoji promjenljiva koja ne zadovoljava uslov nenegativnosti, ona se može zamijeniti razlikom dve nove negativne promenljive. Kanonski oblik linearog programiranja je pogodan za efikasno izvođenje Gausovih transformacija nad matricom problema, što se, na primjer, koristi u glavnim iteracijama simpleks metoda.

U nastavku koristimo sljedeću terminologiju linearog programiranja (LP):

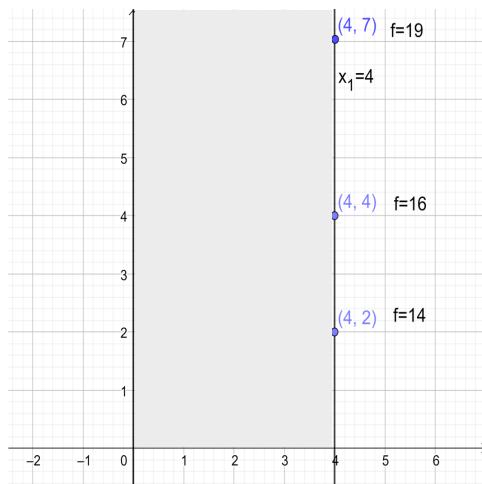
-
- *Rješenje* u standardnom matematičkom značenju najčešće označava rezultat izračunavanja, odnosno krajnje rješenje posmatranog problema. Međutim, u rješavanju problema LP-a to nije slučaj. Bilo koja specifikacija vrijednosti promjenljivih (x_1, \dots, x_n) se smatra rješenjem, bez obzira da li je riječ o željenom krajnjem rješenju ili čak o nekom rješenju koje i ne zadovoljava sva ograničenja. U zavisnosti od toga, rješenja možemo podijeliti na:
 - *dopustiva rješenja* (eng. *feasible solution*) – podrazumijevaju ona rješenja koja zadovoljavaju sva ograničenja u modelu;
 - *nedopustiva rješenja* (eng. *infeasible solution*) – ona rješenja koja ne zadovoljavaju barem jedno od ograničenja u modelu.
 - *Dopustivi region*: označava skup svih rješenja koja su dopustiva. Ovo je prostor koji pretražujemo u svrhu pronađaska najboljeg rješenja modela. Takođe, može se desiti da problem nema niti jedno dopustivo rješenje. Za takav problem se kaže da je *nedopustiv* (eng. *infeasible*).
 - *Optimalno rješenje* je ono rješenje iz dopustivog regiona za koje je funkcija cilja najveća (pod pretpostavkom da maksimizujemo funkciju cilja), odnosno najmanja (pod pretpostavkom da minimizujemo funkciju cilja). Treba napomenuti da optimalno rješenje ne mora biti jedinstveno. Npr. na Slici 3.3 je prikazan dopustiv region sa ograničenjima $x_1, x_2 \geq 0$, $x_1 \leq 4$ i $3x_1 + 2x_2 \leq 14$, dok je funkcija cilja, koja se maksimizuje, data sa $f = 3x_1 + 2x_2$.

Za razliku od nedopustivog problema, može se desiti i sasvim druga situacija, da je problem *neograničen* (eng. *unbounded*). U tom slučaju vrijednosti neke od promjenljivih se mogu povećavati, da se ne naruši dopustivost, a tako da funkcija cilja (u problemu maksimizacije) teži ka beskonačno, tj. nije ograničena odozgo nekom vrijednošću.

Primjer neograničenog problema možemo vidjeti na Slici 3.4, gdje je potrebno maksimizovati funkciju $f = 3x_1 + x_2$, pod uslovima $0 \leq x_1 \leq 4$, te $x_2 \geq 0$. Kao što se može vidjeti, kako vrijednost promjenljive x_2 teži ka beskonačnosti, tako i vrijednost funkcije f teži ka beskonačnosti.



Slika 3.3: LP sa više optimalnih rješenja.



Slika 3.4: Primjer LP-a koji je neograničen.

Treba imati na umu da (optimalno) rješenje LP-a ne mora da bude cjelobrojno. Budući da svaka promjenljiva odlučivanja predstavlja nivo odgovarajuće aktivnosti, pretpostavlja se da se aktivnosti mogu izvoditi i na

razlomljenim nivoima, odnosno, aktivnost se ne dijeli samo na diskretne dijelove.

Ako su domeni svih promjenljivih LP-a cjelobrojne vrijednosti, riječ je o modelu *cjelobrojnog programiranja*. Ako pored cjelobrojnih, postoje i neke promjenljive koje uzimaju neprekidne vrijednosti, riječ je o modelima *mješovitog–cjelobrojnog linearнog programiranja*. Veliki broj realnih problema koji dolaze iz prakse su upravo problemi mješovitog–cjelobrojnog linearнog programiranja.

U naredna dva primjera opisani su modeli linearнog programiranja za dva poznata problema.

Primjer 3.2. *Problem optimalne dijete.* Prepostavimo da su nam na raspolaganju namirnice N_1, \dots, N_n . Jedinična cijena namirnice N_j je $c_j > 0$, $j = 1, \dots, n$. U namirnicama su prisutni nutritivni elementi e_i , $i = 1, \dots, m$ pri čemu namirnica N_j sadrži $a_{ij} \geq 0$ nutritivnog elementa e_i , $i = 1, \dots, n$, $j = 1, \dots, m$. Svaka osoba u toku dana treba da unese barem b_i jedinica nutritivnog elementa e_i . Potrebno je modelovati sljedeći problem:

Koliko je koje namirnice potrebno kupiti za (dnevnu) prehranu da bi se zadovoljila dnevna potreba za svim nutritivnim elementima, a da se pri tome minimizuje cijena prehrane?

Rješenje. Uvedimo realne promjenljive $x_j \geq 0$ za količinu hrane namirnice N_j , $j = 1, \dots, n$, koja će se koristiti za prehranu. Funkcija cilja treba da odgovara minimizaciji cijene prehrane:

$$f(x) = c_1x_1 + \dots + c_nx_n.$$

Ograničenja problema su vezana za zahtjev da se zadovolji dnevna potreba za unosom svakog nutritivnog elementa. Odavde, slijedi da za i -ti element treba da bude zadovoljeno

$$a_{i,1}x_1 + \dots + a_{i,n}x_n \geq b_i$$

za svako $i = 1, \dots, m$. Dodajući još uslove nenegativnosti, $x_i \geq 0$, formulisali smo linearni model za ovaj problem.

Primjer 3.3. *Problem najboljeg pravca (problem linearne regresije).* Zadani su podaci $(x_i, y_i), i = 1, \dots, n$. Potrebno je odrediti pravu sa jednačinom $y = kx + l, k, l \in \mathbb{R}$ koja, u smislu l_1 norme, najbolje aproksimira date podatke. Napomenimo da umjesto l_1 često stoje i neke druge norme, poput euklidske, l_2 norme. Drugim riječima, potrebno je minimizovati funkciju cilja

$$f(k, l) = \sum_{i=1}^n |kx_i + l - y_i|.$$

Rješenje. Označimo sa $z_i = |kx_i + l - y_i| = \max\{kx_i + l - y_i, y_i - kx_i - l\}$, $i = 1, \dots, n$.

Primijetimo da absolutna vrijednost ne odgovara direktno problemu LP-a, bilo da je prisutna u funkciji cilja ili ograničenjima, jer ona ne predstavlja linearu funkciju (već je linearna po dijelovima). Međutim, polazni problem možemo da preformulišemo na sljedeći problem minimizacije linearog programiranja:

$$f(x) = c^T w = z_1 + z_2 + \dots + z_n + 0 \cdot k + 0 \cdot l, \text{ pod uslovima}$$

- $-z_i \leq kx_i + l - y_i$
- $kx_i + l - y_i \leq z_i, i = 1, \dots, n$
- $k, l \in \mathbb{R}$.

Na kraju ove sekcije, navednimo nekoliko osnovnih implicitnih pretpostavki o problemima linearnog programiranja.

1. *Proporcionalnost.* Doprinos svake aktivnosti ciljnoj funkciji f proporcionalan je njenom koeficijentu doprinsosa, tj. $c_i x_i$. Doprinos svake aktivnosti svakom od ograničenja u modelu proporcionalan je njenom koeficijentu doprinsosa, tj. $a_i x_i$.
2. *Aditivnost.* Svaka funkcija u modelu je zbir doprinsosa pojedinih aktivnosti, npr. $f(x) = \sum_i a_i x_i$.
3. *Djeljivost.* Promjenljive odlučivanja se mogu podijeliti na vrijednosti koje nisu cijele, uzimajući u obzir razložene vrijednosti, dok god su ograničenja zadovoljena.

-
4. *Pouzdanost.* Prepostavljamo da su vrijednosti parametara u modelu poznate sa velikom dozom pouzdanosti ili se barem tako tretiraju. Dobijeno optimalno rješenje je optimalno za razmatrani problem. Ako su vrijednosti parametara pogrešne, tada rezultujuće rješenje po pravilu ima malu (upotrebnu) vrijednost.

3.3 Teorija linearog programiranja

U ovom poglavlju ćemo formalno izvesti opšte uslove pod kojima se problem linearog programiranja može riješiti. Prije nego što krenemo sa teorema vezanim sa tim, kao i egzistenciju optimalnog rješenja u LP-u, navedimo nekoliko osnovnih definicija i pratećih teorema.

Definicija 3.1. Neka je $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$. Skup $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$ se naziva poliedar u \mathbb{R}^n .

Definicija 3.2. Za skup $S \in \mathbb{R}^n$ kažemo da je konveksan akko za sve $x, y \in S$ i $\lambda \in [0, 1]$ vrijedi $\lambda x + (1 - \lambda)y \in S$

Definicija 3.3. Za funkciju $f : \mathbb{R}^n \mapsto \mathbb{R}$ kažemo da je konveksna akko

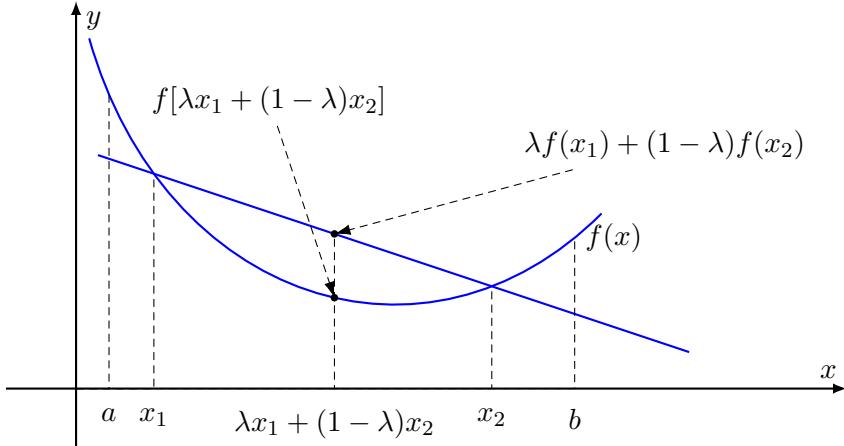
$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y),$$

za svaki $x, y \in \mathbb{R}^n$ i svaki $\lambda \in [0, 1]$.

Primijetimo da je svaka funkcija oblika $f(x_1, \dots, x_n) = \sum_{i=1}^n c_i x_i = c^T x$ za neke $c_i \in \mathbb{R}$, konveksna, što se lako provjerava na osnovu definicije konveksnosti. Dakle, LP u svom opštem obliku ima ciljnu funkciju koja je konveksna. Geometrijska interpretacija konveksnosti funkcije je prikazana na Slici 3.5.

Definicija 3.4. Neka su dati vektori $z^{(i)} \in \mathbb{R}^n$, te $\lambda_i \in [0, 1]$ $i = 1, \dots, m$ tako da je $\sum_{i=1}^m \lambda_i = 1$. Vektor $x = \sum_{i=1}^m \lambda_i z^{(i)}$ se naziva konveksna kombinacija vektora $z^{(i)}$, $i = 1, \dots, n$.

Konveksne funkcije imaju lijepa svojstva sa stanovišta teorije i u kombinaciji sa konveksnim skupovima čine osnov teorije optimalnosti. Inače, vrijedi sljedeća teorema.



Slika 3.5: Konveksna funkcija: geometrijska interpretacija.

Teorema 3.1. Neka je $f : S \mapsto \mathbb{R}$ konveksna funkcija definisana na konveksnom skupu $S \subseteq \mathbb{R}^n$. Ako funkcija f u tački $x^* \in S$ postiže lokalni minimum, onda ta tačka predstavlja i tačku globalnog minimuma.

Definicija 3.5. Neka je $a \in \mathbb{R}^n$ i $b \in \mathbb{R}$. Skup $\{x \in \mathbb{R}^n \mid a^T x = b\}$ se naziva hiperravan u \mathbb{R}^n čiji je vektor normale a , dok se $\{x \in \mathbb{R}^n \mid a^T x \geq b\}$ naziva poluprostor.

Pokažimo sada da je poliedar svakog LP-a konveksan skup. To je važna karakteristika, jer se uslovima problema LP-a formira upravo poliedar kao skup dopustivih rješenja.

Propozicija 3.1. Vrijede sljedeće tvrdnje:

- (i) Poluprostor u \mathbb{R}^n je konveksan skup.
- (ii) Presjek konačno mnogo konveksnih skupova je konveksan skup.

Dokaz. (i) Neka je dat poluprostor $P = \{x \in \mathbb{R}^n \mid a^T x \geq b\}$. Uzmimo $x, y \in P$, te neko $\lambda \in [0, 1]$. Treba da pokažemo da konveksna kombinacija ova dva vektora pripada P . Iz $x \in P$, slijedi $a^T x \geq b$. Takođe, kako je $y \in P$,

onda je $a^T y \geq b$. Sada imamo sljedeće:

$$a^T(\lambda x) \geq \lambda b \quad (3.7)$$

$$a^T((1 - \lambda)y) \geq (1 - \lambda)b \quad (3.8)$$

Sabirajući posljednje dvije nejednakosti, dobijamo

$$a^T(\lambda x + (1 - \lambda)y) \geq b, \quad (3.9)$$

odakle slijedi da $\lambda x + (1 - \lambda)y \in P$, čime je tvrdnja dokazana.

(ii) Neka su S_1, \dots, S_n konveksni skupovi. Ako $x, y \in \cap_{i=1}^n S_i$, onda slijedi da $x, y \in S_i$ za svaki $i = 1, \dots, n$. Zbog konveksnosti svakog od skupova, $\lambda x + (1 - \lambda)y \in S_i$, za svaki $i = 1, \dots, n$. Prema tome, vrijedi $\lambda x + (1 - \lambda)y \in \cap_{i=1}^n S_i$, odakle slijedi da je $\cap_{i=1}^n S_i$ konveksan. \square

Iz prethodne propozicije vrijedi sljedeća posljedica.

Posljedica 3.1. *Poliedar P nekog problema LP-a je konveksan skup.*

Dokaz. Definišimo $S_i := \{(x_1, \dots, x_n) \in \mathbb{R}^n \mid C_i^T x \geq b_i\}$, gdje je $C_i = (a_{i,1}, \dots, a_{i,n})$, $i = 1, \dots, m$. Kako je $\cap_{i=1}^m S_i$ poliedar koji odgovara problemu LP-a, na osnovu prethodne propozicije (ii), slijedi tvrdnja. \square

Kako je poliedar konveksan skup, Teorema 3.1 nam garantuje da, uz pretpostavku da postoji lokalni minimum (maksimum), onda postoji i globalni minimum (maksimum), odnosno optimalno rješenje zadanog problema LP-a.

Nadalje, formalno definišimo pojmove ekstremne tačke (eng. extreme point), vrha (eng. vertex) i baznog dopustivog rješenja koji igraju bitnu ulogu u teoriji linearog programiranja.

Definicija 3.6. *Neka je P poliedar u \mathbb{R}^n . Vektor $x \in P$ se naziva ekstremna tačka poliedra P ako ne postoje vektori $y, z \in P$, $y \neq x, z \neq x$ i skalar $\lambda \in [0, 1]$ takvi da je $x = \lambda y + (1 - \lambda)z$.*

Dakle, ekstremna tačka poliedra se ne može napisati kao netrivijalna konveksna kombinacija tačaka iz tog poliedra.

Neformalno rečeno, ako je x ekstremna tačka poliedra, tada ona ne leži između neke dvije tačke y i z poliedra P .

Definicija 3.7. Neka je dat poliedar $P \subseteq \mathbb{R}^n$. Tačku $x \in P$ nazivamo vrh (poliedra) ako postoji linearna funkcija $c^T x$ koja je strogo minimizovana u toj tački, tj. za sve $y \in P, y \neq x$, vrijedi $c^T x < c^T y$.

Geometrijski posmatrano, vrh poliedra P je ona tačka poliedra kroz koju možemo provući poluravan (hiperravan) sa svojstvom da se sve tačke iz P (izuzev vrha) nalaze sa iste strane te poluravnji.

Uvedimo sada nekoliko novih pojmove koji će nam pomoći da definišemo bazno dopustivo rješenje.

Definicija 3.8. Neka je poliedar P zadan na sljedeći način

- $a_i^T x \geq b_i, i \in C_1$
- $a_i^T x = b_i, i \in C_2$
- $a_i^T x \leq b_i, i \in C_3$.

Ako za neko $x^* \in P$ i neki $i_0 \in C_1 \cup C_1 \cup C_2$ vrijedi $a_{i_0}^T x^* = b_{i_0}$, onda je indeks (ograničenje) i_0 aktivno u x^* .

Definicija 3.9. Neka je zadan poliedar iz Definicije 3.8. Neka je $I = \{i \in C_1 \cup C_2 \cup C_3 \mid a_i^T x^* = b_i\}$ skup indeksa aktivnih u $x^* \in \mathbb{R}^n$. Ako je skup $\{a_i \mid i \in I\}$ linearno nezavisno, onda se x^* naziva bazno rješenje.

Definicija 3.10. Bazno dopustivo rješenje x^* je bazno rješenje koje zadovoljava uslove linearog programa.

Pokažimo sada da vrhovi, ekstremne tačke poliedra i bazna dopustiva rješenja predstavljaju ekvivalentne pojmove kada je riječ o linearom programiranju.

Teorema 3.2. Neka je P poliedar i neka je $x^* \in P$. Sljedeće tri tvrdnje su ekvivalentne:

-
1. x^* je vrh poliedra;
 2. x^* je ekstremna tačka poliedra;
 3. x^* je bazno dopustivo rješenje (BDR).

Dokaz. Pokažimo da vrijedi $1 \Rightarrow 2, 2 \Rightarrow 3$, pa onda $3 \Rightarrow 1$, čime ćemo pokazati ekvivalenciju sve tri tvrdnje.

$1 \Rightarrow 2$. Neka je x^* vrh poliedra. Iz Definicije 3.7 slijedi da postoji $c \in \mathbb{R}^n$ tako da $c^T x^* < c^T x$, za sve $x \in P, x \neq x^*$. Prepostavimo da x^* nije ekstremna tačka poliedra P . To bi značilo da postoje $y, z \in P$, koji su različiti od x^* , takvi da se x^* može predstaviti kao njihova (netrivijalna) konveksna kombinacija, tj. $x^* = \lambda y + (1 - \lambda)z$, za neko $\lambda \in (0, 1)$. Na osnovu prepostavke je $c^T x^* < c^T y$. Sada je

$$\begin{aligned} c^T x^* &= c^T(\lambda y + (1 - \lambda)z) = c^T(\lambda y) + c^T((1 - \lambda)z) > \\ &> \lambda c^T y + (1 - \lambda)c^T z = c^T y. \end{aligned} \quad (3.10)$$

što je nemoguće, odakle slijedi tvrdnja.

$2 \Rightarrow 3$. Ovu implikaciju ćemo dokazati kontrapozicijom. Prepostavimo da tačka x^* nije BDR i pokažimo da iz toga slijedi da ona nije ni ekstremna tačka. Bez smanjenja opštosti, zapišimo poliedar P na sljedeći način.

$$a_i^T x \geq b_i, i \in C_1 \quad (3.11)$$

$$a_i^T x = b_i, i \in C_2. \quad (3.12)$$

Ako x^* nije BDR, to znači da je broj uslova koji su aktivni u x^* manji od n . Ako sa $I \subseteq C_1 \cup C_2$ označimo skup aktivnih indeksa u x^* , onda je $\{a_i \mid i \in I\} \subseteq \mathbb{R}^n$ pravi potprostor od \mathbb{R}^n . Prema tome, postoji vektor $d \in \mathbb{R}^n \setminus \{0\}$ za koji vrijedi $a_i^T d = 0$, za sve $i \in I$. Definišimo vektore

$$y = x^* + \epsilon d, z = x^* - \epsilon d,$$

za neko $\epsilon > 0$, pa pokažimo da $y, z \in P$.

Ako je $i \in I$, onda je

$$a_i^T y = a_i^T(x^* + \epsilon d) = a_i^T x^* + \epsilon a_i^T d = a_i^T x^* = b_i.$$

Preostalo je da pokažimo da za $i \notin I$ vrijedi $a_i^T y \geq b_i$, odakle bi slijedilo da $y \in P$.

Dakle, $a_i^T y = a_i^T(x^* + \epsilon d) = a_i^T x^* + \epsilon a_i^T d$. Ako je $a_i^T d \geq 0$, slijedi da je $a_i^T y > a_i^T x^* \geq b_i$, odakle slijedi da $y \in P$. Ako je $a_i^T d < 0$, onda odaberimo takve ϵ za koje će vrijediti $a_i^T x^* + \epsilon a_i^T d > b_i$. Interval vrijednosti za ϵ dobijemo iz sljedećeg računa:

$$a_i^T x^* + \epsilon a_i^T d > b_i \Rightarrow \epsilon < \frac{b_i - a_i^T x^*}{a_i^T d}$$

Za svaki $\epsilon \in \left(0, \frac{b_i - a_i^T x^*}{a_i^T d}\right)$ vrijedi $a_i^T y > b_i$, odakle slijedi $y \in P$. Slično pokažemo i da vrijedi $z \in P$. Prema tome, lako je vidjeti da vrijedi $x^* = \frac{1}{2}y + \frac{1}{2}z$. Dakle, x^* se dobija kao netrivijalna konveksna kombinacija, što je u suprotnosti sa definicijom ekstremne tačke.

$3 \Rightarrow 1$. Neka je x^* BDR. Treba da pokažemo da je x^* takođe vrh poliedra P . Označimo sa I skup aktivnih indeksa u x^* i definišimo $c = \sum_{i \in I} a_i$. Tada imamo $c^T x^* = \sum_{i \in I} a_i^T x^* = \sum_{i \in I} b_i$. Za proizvoljno $x \in P$ dobijamo

$$c^T x = \sum_{i \in I} a_i^T x \geq c^T x^* \quad (3.13)$$

Iz toga vidimo da je x^* rješenje LP problema minimizacije. Jednakost u (3.13) vrijedi ako i samo ako je $a_i^T x = b_i$ za sve $i \in I$. Kako je x^* BDR, broj linearne nezavisnih vektora $a_i, i \in I$ koji su aktivni u x^* je jednak n , pa sistem $a_i^T x = b_i, i \in I$ ima jedinstveno rješenje koje je upravo x^* . Prema tome, pokazano je da postoji vektor $c \in \mathbb{R}^n$ za koji vrijedi $c^T x > c^T x^*$ za sve $x \neq x^*, x \in P$, odake, na osnovu definicije vrha poliedra, slijedi tvrdnja. \square

U nastavku ćemo dati potrebne uslove za postojanje BDR-a, tj. ekstremne tačke poliedra. Takođe, daćemo i postupak za konstrukciju BDR-a, te ćemo uvesti pojam degenerativnog rješenja.

Teorema 3.3. *Neka je $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ i $\text{rang}(A) = m$, $m < n$. Vektor x je bazno rješenje akko je $Ax = b$ i ako postoje indeksi p_1, p_2, \dots, p_m takvi da vrijedi:*

-
1. kolone A_{p_1}, \dots, A_{p_m} su linearne nezavisne
 2. ako $i \notin \{p_1, \dots, p_m\}$, onda $x_i = 0$.

Iz prethodne teoreme možemo zaključiti sljedeće. Da bismo konstruisali bazno rješenje, potrebno je izvršiti sljedeće korake:

1. Odaberemo m linearne nezavisne kolone A_{p_1}, \dots, A_{p_m} matrice A .
2. Riješimo sistem $Bx_B = b$, za $x_B = [x_{p_1}, \dots, x_{p_m}]$ i $B = [A_{p_1} \dots, A_{p_m}]$
3. Dodijelimo

$$x = (x_1, \dots, x_n) := \begin{cases} 0, & j \notin \{p_i \mid i = 1, \dots, m\} \\ (x_B)_i, & j \in \{p_i \mid i = 1, \dots, m\} \end{cases},$$

čime dobijamo bazno rješenje.

Promjenljive x_{p_1}, \dots, x_{p_m} se nazivaju *bazne promjenljive*, dok vektore A_{p_1}, \dots, A_{p_m} nazivamo *baznim vektorima*. Indekse p_1, \dots, p_m nazivamo *bazni indeksi*, a matricu B matricom baze.

Definišimo sada pojam *degenerativnog* rješenja i degenerativnog baznog rješenja.

Definicija 3.11. Neka je dat poliedar $P = \{x \in \mathbb{R}^n \mid a_i^T x \geq b_i, x_i \geq 0, i = 1, \dots, m\}$. Za bazno rješenje $x^* \in \mathbb{R}^n$ kažemo da je *degenerativno* ako je više od n ograničenja aktivno u x^* .

Definicija 3.12. Neka je $P = \{x \in \mathbb{R}^n \mid Ax = b, x_i \geq 0, i = 1, \dots, m\}$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ i neka je x bazno rješenje. Vektor x nazivamo *degenerativno bazno rješenje* ako je više od $n - m$ komponenti u vektoru x jednako 0.

U sljedećem primjeru ćemo pokazati kako se pronalazi bazno rješenje, te kako izgleda degenerativno bazno rješenje.

Primjer 3.4. Zadajmo poliedar $P = \{(x_1, x_2, x_3)^T \mid x_2 - x_1 = 0, x_1 + x_2 + x_3 = 2, x_1, x_2, x_3 \geq 0\}$. Pronaći bazna dopustiva rješenja i odrediti jesu li ona degenerativna.

Rješenje. Lako je vidjeti da je $A = \begin{pmatrix} -1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ te $b = [0 \ 2]^T$. Prateći postupak konstrukcije baznog rješenja, pronađimo dvije linearne nezavisne kolone matrice A . Npr. možemo uzeti prve dvije kolone, pa je matrica $B = \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}$. Riješimo sistem jednačina (stavka 2), odakle je rješenje $x_B = [1 \ 1]^T$, a krajnje rješenje je $x^* = [1 \ 1 \ 0]^T$. Kako je x^* aktivno u sva tri uslova $x_2 - x_1 = 0$, $x_1 + x_2 + x_3 = 2$ i $x_3 = 0$, rješenje je nedegenerativno bazno dopustivo rješenje. Takođe, možemo odabrati drugu i treću kolonu matrice za bazne kolone (kao i prvu i treću), odakle bismo dobili rješenje $x^* = [0 \ 0 \ 2]^T$, aktivno u četiri uslova, pa smo dobili degenerativno bazno dopustivo rješenje.

U nastavku govorimo o egzistenciji optimalnog rješenja problema LP-a. Definišimo prvo pojam pravca u poliedru.

Definicija 3.13. *Poliedar P sadrži pravac akko postoji $x \in P$ i $d \in \mathbb{R}^n \setminus \{0\}$ tako da je $x + \lambda d \in P$, za sve $\lambda \in \mathbb{R}$.*

Primjer pravca koji postoji u poliedru je dat na Slici 3.6.

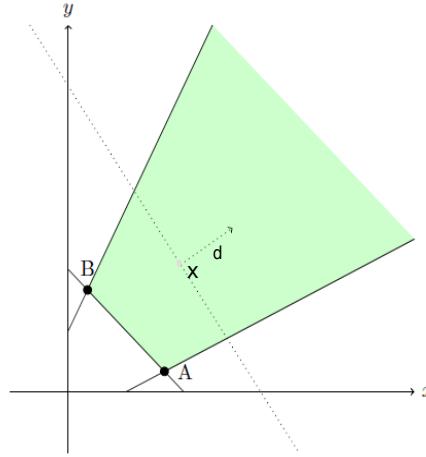
Sljedeću teoremu, koja daje vezu između ekstremnih tačaka poliedra i dopustivih pravaca, navodimo bez dokaza.

Teorema 3.4. *Neka je dat poliedar $P = \{x \in \mathbb{R}^n \mid a_i^T x \geq b_i, i = 1, \dots, m\} \neq \emptyset$. Tada su sljedeće tri tvrdnje ekvivalentne:*

- *Poliedar P ima barem jednu ekstremnu tačku.*
- *Poliedar P ne sadrži pravac.*
- *Postoji n linearne nezavisne vektora medju kolonama matrice A.*

Sljedeća teorema nam daje dovoljan uslov za postojanje optimalnog rješenja koje je uvijek jedna od ekstremnih tačka poliedra (dopustivog regionala) problema linearнog programiranja.

Teorema 3.5. *Neka je zadan problem LP-a sa funkcijom cilja $c^T x$ koju je potrebno minimizovati na poliedru P . Pretpostavimo da za poliedar P*



Slika 3.6: Pravac u poliedru

postoji barem jedna ekstremna tačka i da za dati problem postoji optimalno rješenje. Tada postoji i optimalno rješenje koje je ekstremna tačka poliedra P .

Dokaz. Sa $O^* \neq \emptyset$ označimo skup svih optimalnih rješenja nad poliedrom $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$, te neka je $c^T x^* = v$, za sve $x^* \in O^*$. Vidimo da je $O^* = \{x \in \mathbb{R}^n \mid Ax \geq b, c^T x = v\}$ takođe poliedar i da važi $O^* \subset P$. Iz Teoreme 3.4 slijedi da poliedar P ne sadrži pravac. Iz toga zaključujemo da ni O^* nema pravac, odakle slijedi da O^* ima barem jednu ekstremnu tačku (stavka 1 Teoreme 3.4). Označimo tu ekstremnu tačku sa q^* . Pokažimo da je to ujedno i ekstremna tačka od P . Ako bismo pretpostavili da q^* nije ekstremna tačka poliedra P , po definiciji, slijedi da postoji $y, z \in P$ tako da $q^* = \lambda y + (1 - \lambda)z$, $y \neq q^*, z \neq q^*, \lambda \in [0, 1]$. Kako je $q^* \in O^*$, slijedi

$$v = c^T q^* = c^T(\lambda y + (1 - \lambda)z) = \lambda \underbrace{c^T y}_{\geq v} + (1 - \lambda) \underbrace{c^T z}_{\geq v} \geq \lambda v + (1 - \lambda)v = v,$$

odakle imamo $c^T y = v$ i $c^T z = v$, dakle $y, z \in O^*$. Prema tome, q^* nije ekstremna tačka od O^* . Dolazimo do kontradikcije, odakle zaključujemo da q^* mora biti ekstremna tačka i od P . \square

Navedimo sada i važnu posljedicu prethodne teoreme.

Posljedica 3.2. *Ako rješavamo problem minimizacije LP-a na poliedru P koji ima barem jednu ekstremnu tačku, tada vrijedi tačno jedna od sljedeće dvije tvrdnje:*

1. *Funkcija cilja ima vrijednost $-\infty$ (funkcija cilja je neograničena).*
2. *Postoji ekstremna tačka u kojoj funkcija cilja dostiže optimalno rješenje.*

Prethodna tvrđenja nam daju bazu za konstrukciju simpleks metoda koji je, kao što smo već pomenuli, jedna od najpoznatijih i najefikasnijih generalnih metoda za rješavanje LP problema. Kako ćemo vidjeti u narednom poglavljtu, simpleks metoda je zasnovana na iterativnom posjećivanju vrhova poliedra u cilju pronalaska optimalnog rješenja.

3.4 Neke tehnike u modelovanju problema linear-nog programiranja

Apsolutna vrijednost u funkciji cilja. Pretpostavimo da imamo model zadat sa

$$\begin{aligned} & \min \sum_{j \in J} |x_j| \\ & \text{t.d.} \\ & \sum_{j \in J} a_{ij} x_j \leq b_i, \forall i \in I \\ & x_j \in \mathbb{R}, \forall j \in J. \end{aligned} \tag{3.14}$$

Kao što znamo, apsolutna vrijednost nije linearna funkcija, ali se ovaj model može transformisati u model linearnog programiranja na sljedeći način: svaku promjenljivu ćemo zapisati kao razliku dvije pozitivne promjenljive, tj. $x_j = x_j^+ - x_j^-$, a apsolutnu vrijednost kao zbir dvije pozitivne promjenljive, tj. $|x_j| = x_j^+ + x_j^-$. Prema tome, prethodni model se transformiše u ekvivalentan (u smislu optimalnog rješenja) model

$$\begin{aligned} \min & \sum_{j \in J} (x_j^+ + x_j^-) \\ \text{t.d.} & \\ \sum_{j \in J} a_{ij} (x_i^+ - x_i^-) & \leq b_i, \forall i \in I \\ x_i^+, x_i^- & \geq 0, \forall j \in J. \end{aligned} \tag{3.15}$$

Može se pokazati da za optimalno rješenje prethodnog modela vrijedi $x^+ \cdot x^- = 0$. Pretpostavimo da je $x_j^*, j \in J$ optimalno rješenje LP-a (3.14). Rješenje za LP (3.15) dato u obliku $(x_j^*)^* = x_j^*$ ako je $x_j^* \geq 0$, a inače $(x_j^*)^* = 0$ i slično $(x_j^-)^* = -x_j^*$ ako je $x_j^* \leq 0$, a inače $(x_j^-)^* = 0$ je dopustivo za (3.15). Lako se provjeri da su vrijednosti ova dva rješenja u odnosu na odgovarajuće funkcije cilja jednake. Pokažimo da je rješenje $((x^+)^*, (x^-)^*)$ ujedno i optimalno rješenje modela (3.15). Neka je rješenje (x^+, x^-) optimalano za model (3.15) za koje postoji neko j tako da $x_j^+, x_j^- > 0$. Bez smanjenja opštosti, pretpostavimo da je $x_j^+ \geq x_j^-$. Rješenje (y^+, y^-) za koje je $y_i^+ = x_i^+$ i $(y_i^- = x_i^-)$ za sve $i \neq j$ i $y_j^+ = x_j^+ - x_j^- \geq 0$ i $(y_j^-)^* = 0$ je dopustivo, i daće istu ili manju vrijednost funkcije cilja nego rješenje prije transformacije. Prema tome, ukoliko postoje, optimalna rješenja LP-a (3.15) su među onim rješenjima za koje vrijedi da barem jedna od promjenljivih x_j^+ ili x_j^- , $j \in J$ jednaka 0. Sva ostala rješenja su podoptimalna za model (3.15) i sigurno mogu da budu izuzeta iz domena rješenja (3.15) bez gubitka optimalnosti. Prema tome, za svako dopustivo rješenja x za (3.14) postoji dopustivo rješenje (x^+, x^-) za (3.15) i obrnuto. Pri tome, vrijednosti funkcija cilja ovakvih rješenja su jednake. Prema tome, ova dva modela su ekvivalenta. Može se pokazati da transformacija vrijedi

i opštiji slučaj, tj. kada je funkcija cilja data sa $f(x) = \sum_{j \in J} c_j |x_j|$, $c_j \geq 0, j \in J$.

Min-max problem. Neka je dat sljedeći model:

$$\min_x \max_{k \in K} \sum_{j \in J} c_{kj} x_j$$

t.d.

$$\begin{aligned} \sum_{j \in J} a_{ij} x_j &\leq b_i \quad \forall i \in I \\ x_j &\geq 0 \quad \forall j \in J. \end{aligned}$$

Ovaj model se transformiše u model LP-a uvođenjem promjenljive

$$z = \max_{k \in K} \sum_{j \in J} c_{kj} x_j,$$

koja označava maksimalnu vrijednost promjenljive z . Da bi se, nakon uvedene smjene, ograničila vrijednost promjenljive z , u početni model potrebno je uključiti sljedeća ograničenja:

$$\sum_{j \in J} c_{kj} x_j \leq z \quad \forall k \in K. \tag{3.16}$$

Kada se z minimizuje, ova ograničenja osiguravaju da je z veće ili jednako $\sum_{j \in J} c_{kj} x_j$, za sve k . Optimalna vrijednost z neće biti veća od maksimuma svih $\sum_{j \in J} c_{kj} x_j$, jer z minimizujemo. Prema tome, optimalna vrijednost promjenljive z će težiti da bude što manja moguća i jednaka maksimalnoj vrijednosti $\sum_{j \in J} c_{kj} x_j$ nad skupom K . Dakle, početni model se transformiše u:

$$\min z$$

t.d.

$$\sum_{j \in J} a_{ij} x_j \leq b_i \quad \forall i \in I$$

$$\sum_{j \in J} a_{kj} x_j \leq z \quad \forall k \in K$$

$$x \geq 0.$$

Razlomljena funkcija cilja. Neka je dat sljedeći model:

$$\min \frac{\sum_{j \in J} c_j x_j + \alpha}{\sum_{j \in J} d_j x_j + \beta}$$

t.d.

$$\sum_{j \in J} a_{ij} x_j \leq b_i, \forall i \in I$$

$$x_j \geq 0, \forall j \in J.$$

Ovdje je funkcija cilja data kao količnik dvije linearne funkcije. Ovakvi modeli su česta pojava u modelovanju finansijskog planiranja. Objasnimo postupak transformacije frakcione funkcije cilja u linearnu. Pretpostavimo da je $\sum_{j \in J} d_j x_j + \beta \neq 0$ za sve tačke iz dopustivog regiona. Glavni trik u transformaciji je uvođenje nove promjenljive y_j i parametra t , tako da je $y_j = t x_j$. U nastavku, pretpostavljamo da je imenilac u funkciji cilja pozitivan. Slično je izvođenje i sa negativnim imenocem, s tim da imamo znak „-“ ispred, što će promijeniti znak nejednakosti u ograničnjima krajnjeg modela. Dakle, uvedemo $t = \frac{1}{\sum_{j \in J} d_j x_j + \beta}$. Time dobijamo model

$$\min \sum_{j \in J} c_j x_j t + \alpha t$$

t.d.

$$\sum_{j \in J} a_{ij} x_j \leq b_i, \forall i \in I$$

$$\sum_{j \in J} d_j x_j t + \beta t = 1$$

$$t > 0$$

$$x \geq 0.$$

Množeći prvo ograničenje sa $t > 0$ te uvodeći smjenu $y_j = t x_j$, dobijamo

(ekvivalentan) model:

$$\begin{aligned} & \min \sum_{j \in J} c_j y_j + \alpha t \\ & \text{t.d.} \\ & \sum_{j \in J} a_{ij} y_j \leq b_i t, \forall i \in I \\ & \sum_{j \in J} d_j y_j + \beta t = 1 \\ & t > 0 \\ & y \geq 0. \end{aligned}$$

Da bi LP model bio validan, potrebno je staviti $t \geq 0$ i na kraju provjeriti optimalnu vrijednost promjenljive t , odnosno, da li je $t > 0$.

Domenske granice u ograničenjima. Neka je dat model

$$\begin{aligned} & \min_{j \in J} c_j x_j \\ & \text{t.d.} \\ & l_i \leq \sum_{j \in J} a_{ij} x_j \leq u_i, \forall i \in I \\ & x_j \geq 0, \forall j \in J. \end{aligned}$$

Direktnom transformacijom dobijamo sljedeći LP (konjunkcija ograničenja):

$$\begin{aligned} & \min_{j \in J} c_j x_j \\ & \text{t.d.} \\ & l_i \leq \sum_{j \in J} a_{ij} x_j, \forall i \in I \\ & \sum_{j \in J} a_{ij} x_j \leq u_i, \forall i \in I \\ & x_j \geq 0, \forall j \in J. \end{aligned}$$

Kako se $\sum_{j \in J} a_{ij}x_j$ javlja u dva ograničenja (što utiče na performanse rješavača), prethodni model se dalje transformiše u (efikasniji) LP model:

$$s_i + \sum_{j \in J} a_{ij}x_j = u_i, \forall i \in I$$

$$0 \leq s_i \leq u_i - l_i, \forall i \in I.$$

Dakle, ako je $s_i = 0$, onda je $\sum_{j \in J} a_{ij}x_j = u_i$, dok ako je $s_i = u_i - l_i$, imamo da je $\sum_{j \in J} a_{ij}x_j = l_i$, $i \in I$.

Konačno, dobijamo ekvivalentan LP model

$$\min_{j \in J} c_j x_j$$

t.d.

$$s_i + \sum_{j \in J} a_{ij}x_j = u_i, \forall i \in I$$

$$0 \leq s_i \leq u_i - l_i, \forall i \in I$$

$$x_j \geq 0, \forall j \in J.$$

3.5 Zadaci

1. Kompanija „Zdrav ljubimac“ proizvodi dvije vrste hrane za pse: Gricko i NjamNjam. Svako pakovanje Gricko hrane sadrži 2 kg žitarica i 3 kg mesa; svako pakovanje NjamNjam-a sadrži 3 kg žitarica i 1,5 kg mesa. Kompanija vjeruje da može prodati onoliko hrane za pse koliko može napraviti. Gricko se prodaje za 2,80 KM po paketu, a NjamNjam za 2,00 KM po paketu. Proizvodnja kompanije je ograničena na nekoliko načina. Prvo, kompanija može kupiti samo do 400.000 kg žitarica svakog mjeseca za 0,20 KM po kg. Dalje, kompanija može kupiti samo do 300.000 kg mesa mjesečno za 0,50 KM po kg. Osim toga, za proizvodnju hrane Gricko potrebna je posebna mašina, a proizvodni kapacitet ove maštine je 90.000 paketa mjesečno. Varijabilni troškovi miješanja i pakovanja hrane za pse iznose 0,25 KM po pakovanju za

hranu Gricko i 0,20 KM po paketu za hranu NjamNjam. Prepostavimo da ste vođa odjela hrane za pse ove kompanije. Na koji način biste upravljali odjelom proizvodnje kako bi maksimizovali profit?

2. Kompanija proizvodi velike industrijske ključeve za cijevi u jednoj od svojih fabrika. Procjena potražnje ovog proizvoda za sljedećih 6 mjeseci data je tabelom

Januar	370
Februar	430
Mart	380
April	450
Maj	520
Jun	440

Sa trenutnom radnom snagom, firma vjeruje da može proizvesti približno 420 ključeva za cijevi mjesečno po cijeni od 40 KM po ključu u standardnom režimu proizvodnje. Dodatnih 80 ključeva mjesečno može se izraditi prekovremenim radom po cijeni od 45 KM po ključu. Ključevi se mogu napraviti unaprijed i držati u zalihamama za kasniju isporuku po cijeni od 3 KM mjesečno po ključu. Mjesečna potražnja za ključevima mora biti zadovoljena svakog mjeseca. Pred kraj decembra (početak januara) kompanija ima 10 ključeva u inventaru. Ona želi planirati svoju proizvodnju, uključujući prekovremeni rad i zalihe za sljedećih 6 mjeseci, kako bi povećala profit. Pod pretpostavkom da je prihod od ovih ključeva stalan, uprava proizvodnje će povećati profit minimizacijom ukupnih troškova nastalih u proizvodnji i isporuci ključeva. Modelovati ovaj problem.

3. (Varijacija prethodnog problema) Kompanija proizvodi prenosne računare. Predviđanja prodaje (u hiljadama) za naredna četiri mjeseca prikazana su u sljedećoj tabeli.

Januar	30
Februar	15
Mart	15
April	25

Proizvodni kapacitet kompanije je 30000 računara mjesečno po cijeni od 250 KM po računaru. Moguće je proizvesti i više proizvoda ali po uvećanoj cijeni od 300 KM. U startu je na lageru 2000 računara. Troškovi skladištenja su 25 KM po svakom računaru koji ostane na zalihamu na kraju mjeseca. Pretpostavljamo da su skladišni kapaciteti neograničeni. Sa proizvodnjom se počinje 1. januara. Kako isplanirati proizvodnju za sljedeća četiri mjeseca tako da se zadovolje svi zahtjevi, a troškovi svedu na minimum?

- Teretni avion ima tri odjeljka za skladištenje tereta: prednji, srednji i zadnji. Ovi odjeljci imaju sljedeća ograničenja po težini i po prostoru:

Odjeljak	Težinski kapacitet	Prostorni kapacitet
Prednji	10	6800
Srednji	16	8700
Zadnji	8	5300

Osim toga, da bi se održala ravnoteža aviona, teret se mora ravnomjerno rasporediti u odgovarajuće pregrade. Sljedeća četiri utovara su dostupna za isporuku na sljedeći let:

Teret	Težina	Zapremina	Profit
C_1	18	480	310
C_2	15	650	380
C_3	23	580	350
C_4	12	390	285

Bilo koji dio ovih tereta se može prihvati za transport. Cilj je utvrditi

koju količinu svakog tereta C_1, C_2, C_3 i C_4 treba prihvati i kako ga rasporediti po odjeljcima, da bi ukupna dobit na letu bila maksimalna.

Uputstvo. Definisati promjenljive odlučivanja $x_{i,j}$ koje predstavljaju broj (tona) tereta i (za $C_i, i = 1, \dots, 4$, respektivno) koji se skladišti u odjeljak $j, j = 1, 2, 3$ (prednji, srednji, zadnji).

5. Kompanija proizvodi četiri proizvoda (1, 2, 3, 4) na dvije mašine (X i Y). Vrijeme (u minutama) za obradu jednog proizvoda na svakoj mašini je dato tabelom:

Proizvod	Mašina X	Mašina Y
1	10	27
2	12	19
3	13	33
4	8	23

Profiti po jedinici proizvoda za proizvod $i = 1, 2, 3, 4$, su 10, 12, 17 i 8 KM, respektivno. Za proizvodnju proizvoda 1, obje mašine X i Y se koriste (u početku proizvodnje obje mašine treba da budu pokrenute), ali ostali proizvodi mogu biti proizvedeni na bilo kojoj mašini.

Fabrika je mala i proizvodni prostor je vrlo ograničen. Proizvodnja na nivou sedmice se skladišti na $50m^2$ podne površine, pri čemu svaki od proizvoda (1, 2, 3, i 4) zauzima 0.1, 0.15, 0.5 i 0.05 (kvadratnih metara) prostora, respektivno.

Zahtjevi kupaca su specifični. Količina proizvoda 3 je povezana sa količinom proizvoda 2. Preciznije, tokom sedmice trebalo bi se proizvesti približno dvostruko više proizvoda 2 nego proizvoda 3. Mašina X je van pogona (zbog održavanja/kvara) 5% vremena, a mašina Y ukupno 7% vremena. Pretpostaviti da radna sedmica ima ukupno 35 sati. Formulisati model proizvodnje ovih proizvoda preko linearног programa, u cilju maksimizovanja profita.

Uputstvo. Promjenljive odlučivanja se vezuju za količinu proizvoda koji se proizvode na svakoj od mašina. Dakle, promjenljive x_i označava

broj proizvoda i koji se proizvode na sedmičnom nivou na mašini X , dok je y_j označava broj proizvoda j koji se proizvode na sedmičnom nivou na mašini Y , $i \in \{1, 2, 3, 4\}$, $j \in \{2, 3, 4\}$. Promjenljivu y_1 ne definишemo jer se proizvod 1 izvršava na obje mašine.

6. Grafičkom metodom riješiti sljedeći problem

$$\begin{aligned} & \max x + 2y \\ & \text{t.d.} \\ & x \leq 10 \\ & 2x + y \geq 0 \\ & x, y \geq 0. \end{aligned}$$

7. Grafičkom metodom riješiti sljedeći problem

$$\begin{aligned} & \max 8x + y \\ & \text{t.d.} \\ & x + y \leq 40 \\ & 2x + y \leq 60 \\ & x, y \geq 0. \end{aligned}$$

8. Neka je dat poliedar $P = \{(x_1, x_2, x_3, x_4)^T \in \mathbb{R}^4 \mid x_2 - x_1 = 0, x_4 - x_3 = 0, x_1 + x_2 + x_3 + x_4 = 2, x_1, x_2, x_3, x_4 \geq 0\}$. Pronaći bazna dopustiva rješenja i ispitati da li su ona degenerativna.

Glava 4

Simpleks metoda

Simpleks metod je jedan od najpoznatijih i najefikasnijih algoritama za rješavanje problema linearнog programiranja. Ideja za pronalaženje optimalnog rješenja problema LP-a simpleks metodom je zasnovana na obilasku vrhova dopustivog regionalnog problema. Vrhovi se uskcesivno obilaze tako da se, prelaskom u novi vrh, vrijednost funkcije cilja smanjuje (ako je dat zadatak minimizacije). Simpleks metod je razvio George Dantzig 1947. godine, a s obzirom na njegovu važnost u rješavanju problema linearнog programiranja, spada među najznačajnije matematičke algoritme XX vijeka.

Neka je dat problem LP-a u kanonskom obliku

$$\begin{aligned} & \min c^T x \\ & Ax = b \\ & x \geq 0, \end{aligned} \tag{4.1}$$

gdje su $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, $x \in \mathbb{R}^n$ uz pretpostavku da je poliedar $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$ datog problema LP-a neprazan. U ovom poglavlju, osim ako drugačije nije naznačeno, strogo ćemo poštovati ovu notaciju.

Definicija 4.1. Za pravac $d \in \mathbb{R}^n$ kažemo da je dopustiv pravac u vektoru $x \in P$ akko postoji skalar $\lambda > 0$ tako da $x + \lambda d \in P$.

4.1 Konstrukcija simpleks metode

Neka je $x = [x_1, \dots, x_n]^T$ bazno dopustivo rješenje problema (4.1), gdje je B matrica baze, x_B bazni dio vektora x , pri čemu su na mjestima nebaznih indeksa iz skupa N postavljene nule. Kao što je rečeno, ideja simpleks metode je „šetanje“ po vrhovima poliedra, dok ne nađemo optimalno rješenje. Krenimo pravcem d iz tačke x pri čemu trebamo obratiti pažnju na to da je zadovoljeno sljedeće:

- *uslov optimalnosti*, tj. $c^T(x + \lambda d) \leq c^T x, \lambda > 0$, što znači da idemo u pravcu smanjenja vrijednosti funkcije cilja;
- *uslov dopustivosti*, potrebno je da pravac d dopustiv u tački x , tj. $(x + \lambda d) \in P$, za neko $\lambda > 0$. Dakle, uvijek se krećemo samo tačkama dopustivog regiona.

Birajmo neku nebaznu koordinatu u x , koja će služiti kao pravac pri pokušaju nalaska boljeg dopustivog rješenja problema. Neka je to $x_j, j \in N$. Označimo vektor $d = (d_B, d_N)^T$, gdje je d_B bazni dio, a d_N nebazni dio. Koordinate nebaznog dijela d_N su jednake 0, osim na koordinati j , koja je postavljena na vrijednost 1. Bazni dio vektora d izvešćemo iz uslova dopustivosti i pretpostavke da „šetamo“ pravcem koji se poklapa sa granicom dopustivog regiona, tj. poliedra P : $Ax + \lambda Ad = b$, odakle je $Ad = 0$. Tada vrijedi

$$Ad = \sum_{i=1}^n A_i d_i = \sum_{i=1}^m A_{p_i} d_i + A_j = Bd_B + A_j = 0$$

za one koordinate p_i koje pripadaju indeksima koordinata baznog dijela, odakle je $d_B = -B^{-1} \cdot A_j$. Za ovako odabran d imamo zadovoljen uslov dopustivosti za ograničenje $A(x + \lambda d) = b$. Posmatrajmo sada ograničenje nenegativnosti, tj. da li je zadovoljeno $x + \lambda d \geq 0$:

- Ako je x nedegenerativno, onda postoji tačno $n - m$ komponenti koje su jednake 0, pa je za nebazni dio komponenti $x_i + \lambda d_i = 0$, za $i \in N \setminus \{j\}$ i $x_j + \lambda d_j = 0$, pa imamo ispunjen uslov nenegativnosti ($x + \lambda d \in P$). Za bazne komponente imamo $x_{p_i} \cdot d_{p_i} > 0$ ako je $d_{p_i} > 0$.

U suprotnom, možemo odabrati malu vrijednost $\lambda > 0$ (pogledati definiciju dopustivog pravca $x + \lambda d$) tako da $x_{p_i} \cdot d_{p_i} > 0$, a time i $x_{p_i} + \lambda d_{p_i} > 0$, te je uslov nenegativnosti zadovoljen.

- Ako je x degenerativno bazno dopustivo rješenje, slijedi da je više od $n - m$ komponenti od x jednako 0. Kao i u prvom slučaju, za nebazni slučaj pokazujemo uslov nenegativnosti. Tada za bazne komponente vektora $x + \lambda d$ postoji bazni indeks i za koji je $x_{p_i} = 0$, pa ako bi bilo $d_{p_i} < 0$, onda ne bismo mogli naći $\lambda > 0$, tako da $x_{p_i} + \lambda d_{p_i} > 0$, čime je narušena dopustivost. Pojavljivanje ovakvog slučaja se može zaobići primjenom određenih pravila izbora nove bazne koordinate umjesto koordinate j , što će biti i pokazano u nastavku ove sekcije.

Da bi se pravilno izabrala nebazna promjenljiva x_j , koristimo uslov optimalnosti $c^T(x + \lambda d) < c^T x$, odakle slijedi $c^T d < 0$. Dakle,

$$c^T d = c_B^T d_B + c_j = c^T (-B^{-1} A_j) + c_j < 0.$$

Prema tome, biramo (nebazni) indeks j tako da je zadovoljeno $\bar{c}_j = c_j - c_B^T B^{-1} A_j < 0$. Vektor $\bar{c} = [\bar{c}_1, \dots, \bar{c}_n]^T$ se naziva *vektor doprinosa*, dok je vrijednost \bar{c}_j *doprinos nebazne promjenljive* x_j .

Iz sljedeće teoreme dobijamo uslove optimalnosti rješenja.

Teorema 4.1. *Neka je x bazno dopustivo rješenje i neka je B bazna matrica, a \bar{c} vektor doprinosa. Tada vrijedi sljedeće*

- *Ako je $\bar{c} > 0$, onda je x optimalno rješenje.*
- *Ako je x optimalno negenerativno rješenje, onda je $\bar{c} \geq 0$.*

Ostalo nam je još da odredimo dužinu pravca d , tj. vrijednost $\lambda > 0$. Ideja je da se krene nekim smjerom dok god se ne dođe do drugog vrha, tj. treba odrediti dužinu vektora d koji je jednak $\lambda^* = \max\{\lambda \in \mathbb{R} \mid x + \lambda d \geq 0\}$. Razlikujemo sljedeće slučajeve:

- Ako je $d > 0$, onda je $x + \lambda d \geq 0$, za sve $\lambda > 0$, pa je $\lambda^* = \infty$, tj. ciljna funkcija je neograničena.

-
- Ako postoji indeks i , takav da je $d_i < 0$, onda za dovoljno malu vrijednost $\lambda > 0$, takvo da $x_i + \lambda d_i \geq 0$, imamo $\lambda \leq -\frac{x_i}{d_i}$. Prema tome

$$\lambda^* = \min_{\{i \in [m] | d_{p_i} < 0\}} -\frac{x_{p_i}}{d_{p_i}}$$

Za indeks l za koji se postiže ovaj minimum, vrijedi $x_{p_l} + \lambda^* d_{p_l} = 0$.

Može se pokazati sljedeća teorema, koja slijedi iz konstrukcije simpleks metode.

Teorema 4.2. *Kolone $A_{p_i}, i \neq l, i = 1, \dots, m$ i A_j su linearno nezavisne, pa je matrica $[A_{p_1}, \dots, A_{p_{l-1}}, A_j, A_{p_{l+1}}, \dots, A_{p_m}]$ regularna. Vektor $x + \lambda^* d$ predstavlja bazno dopustivo rješenje problema (4.1).*

Navedimo sada osnovne korake jedne iteracije simpleks metoda:

1. Inicijalizacija: za bazne indekse $p_1, \dots, p_m \in [n]$, riješiti sistem i dobiti bazno dopustivo rješenje x .
2. Izračunati doprinos za sve $\bar{c}_j = c_j - c_B^T B^{-1} A_j$ $j \in N$. Ako za sve j , $\bar{c}_j \geq 0$, optimum je pronađen, pa algoritam prestaje sa radom. Inače, odabratи neki j td. $\bar{c}_j < 0$.
3. Izračunati $d = -B^{-1} A_j = -u$. Ako je svaki $d_i \geq 0$, funkcija cilja $f = -\infty$, te se završava izvršenje algoritma.
4. Ako je za barem neki i , $d_i < 0$, tada računamo

$$\lambda^* = \min_{\{i \in [m] | d_{p_i} < 0\}} -\frac{x_{p_i}}{d_{p_i}} = \min_{\{i \in [m] | u_{p_i} < 0\}} \frac{x_{p_i}}{u_{p_i}}$$

5. Neka je $\lambda^* = \frac{x_{p_l}}{u_{p_l}}$. Nova baza se formira tako što na mjesto kolone A_{p_l} stavimo kolonu A_j . Novo bazno rješenje $y = x + \lambda^* d$ je dato sa $y_j = \lambda^*$, $y_{p_l} = x_{p_l} + \lambda^* d_{p_l}$, za indeks l koji je bazni indeks, te $y_i = 0$, za $i \in N \setminus \{j\}$.

Naredna iteracija metode koristi novu bazu i bazno rješenje y i izvodi (po potrebi) korake 2–5.

4.2 Tabelarni oblik simpleks metoda

Simpleks algoritam podrazumijeva stalnu zamjenu jedne kolone baze B nekim nebaznim vektorom u iterativnoj šemi, u cilju smanjenja trenutne vrijednosti ciljne funkcije, dok god je to moguće. U praktičnom računanju, za simpleks metodu se zbog kompaktnosti zapisa, umjesto transformisanja standardnog oblika LP-a, obično koristi tabelarni prikaz. Generički tabelarni prikaz simpleks metode se može vidjeti u Tabeli 4.1. Ovakva tabela se naziva simpleks tabela.

			$x_{p_1} = \bar{b}_1$
$B^{-1}A_1$	\dots	$B^{-1}A_n$	\vdots
\bar{c}_1	\dots	\bar{c}_n	$x_{p_m} = \bar{b}_m$
			$-c_B^T x_B$

Tabela 4.1: Simpleks tabela

Na osnovu generičke Tabele 4.1 i koraka simpleks metode opisanih u prethodnoj sekciji, izvršavaju se sljedeći koraci:

- Konstruisati inicijalnu simpleks tabelu sa početnom bazom B i odgovarajuće bazno dopustivo rješenje x .
- Posmatrati koeficijente doprinosa \bar{c}_j (posljednja vrsta tabele). Ako su svi $\bar{c}_j \geq 0$, rješenje se ne može dalje popraviti, pa je nađeno optimalno rješenje. U protivnom, odabratи neki indeks j za koji je $\bar{c}_j < 0$.
- Posmatrati vektor (kolonu) $u = B^{-1}A_j$. Ako su sve koordinate vektora u negativne, rješenje LP-a je $-\infty$ (funkcija cilja neograničena), algoritam završava sa radom.
- Za svaki $u_i > 0$, računati x_{p_i}/u_i , i neka je ovaj količnik najmanji za indeks l . Kolona A_j se uključuje u bazu, dok se kolona A_{p_l} isključuje iz baze.

-
- Svakom redu simpleks tabele dodati l -ti red (pivot red) kojeg množimo odgovarajućim skalarom da bi pivot element (tj. element u presjeku l -te vrste i j -te kolone) postao 1, a svi ostali elementi u pivot koloni 0.

U ovom postupku može se desiti situacija da postoje dva ili više redova koji su kandidati za uključivanje u bazu. Ako se odabere nepovoljan kandidat, postupak može da proizvede degenerativno BDR ili čak da dovede do ciklusa. Da bi se to spriječilo, Robert Bland (1977. godine) je uveo pravilo za pivotiranje danas poznato pod nazivom *Blandovo pravilo* koje glasi:

1. Među svim nebaznim promjenljivima koje su pogodne za uključivanje u bazu odaberemo onu najmanjeg indeksa.
2. Među svim baznim promjenljivima koje su pogodne za isključivanje iz baze odaberemo onu sa najmanjim indeksom.

Može se pokazati da se poštovanjem ovih pravila simpleks metoda sigurno završava u konačno mnogo koraka. Treba napomenuti da Blandovo pravilo nije jedino pravilo za izbjegavanje ciklusa. Postoji i leksikografsko pravilo pivotiranja koje, zbog konciznosti, ne objašnjavamo ovdje.

Obratimo pažnju da se problem minimizacije vrlo lako prevodi u problem maksimizacije sljedećom transformacijom

$$\min_{x \in P} f(x) = -\max_{x \in P} (-f(x)).$$

Prema tome, ako problem maksimizacije rješavamo simpleks metodom, koraci metoda ostaju isti. Jedina izmjena je u koraku 2, gdje se mijenja znak nejednakosti, jer se mijenja znak koeficijenata u funkciji cilja. Dakle, posmatraju se indeksi j za koje je $\bar{c}_j > 0$.

4.3 Primjena simpleks metoda

Transformišimo standardni oblik problema LP-a

$$\max c^T x$$

t.d.

$$Ax \leq b$$

$$x \geq 0$$

na kanonski oblik pogodan simpleks metodi ($Ax = b$) dodavanjem „izjednacavajućih“ promjenljivih s na sljedeći način:

$$\max c^T x \tag{4.2}$$

$$Ax + s = b \tag{4.3}$$

$$x \geq 0, s \geq 0. \tag{4.4}$$

$$(4.5)$$

što u matričnom obliku odgovara zapisu

$$\max c^T x \tag{4.6}$$

$$(A|I) \cdot \begin{pmatrix} x \\ s \end{pmatrix} = b \tag{4.7}$$

$$(x, s) \geq 0. \tag{4.8}$$

$$(4.9)$$

Primjer 4.1. Neka je dat problem LP-a:

$$f = \max x_1 + x_2$$

t.d.

$$x_1 + 3x_2 \leq 9$$

$$2x_1 + x_2 \leq 8$$

$$x_1, x_2 \geq 0.$$

Rješenje. Pretvorimo model u kanonski oblik dodavajući izjednačavajuće promjenljive $s = (s_1, s_2) \geq 0$, pa dobijamo

$$\begin{array}{rcll} x_1 & + 3x_2 & + s_1 & = 9 \\ 2x_1 & + x_2 & + s_2 & = 8 \\ x_1 & + x_2 & & = f - 0 \end{array}$$

Ovaj problem odgovara simpleks tabeli

x_1	x_2	s_1	s_2	
1	3	1	0	9
2	1	0	1	8
1	1	0	0	0

Tabela 4.2: Inicijalna simpleks tabela.

Na početku konstruišimo bazu uzimajući vektore s_1 i s_2 . U tom slučaju, bazno dopustivo rješenje je $x = (0, 0, 9, 8)$, za koje je $f = 0$. Sada ažiriramo simpleks tabelu izvodeći pivotiranje. Izaberimo indeks j tako da je $\bar{c}_j > 0$. Neka je $j = 1$, odnosno, u razmatranje ćemo uzeti promjenljivu x_1 , čijim doprinosom želimo da uvećamo funkciju cilja, jer rješavamo problem maksimizacije. Izaberimo pivot vrstu tako što posmatramo odnos $\bar{b}_i/\bar{a}_{1,j}$, za sve $\bar{a}_{1,j} > 0$, $B^{-1}A_1 = \bar{a}_{1,j}$. Tražimo onaj indeks koji minimizuje datu vrijednost. Imamo dvije mogućnosti, $9/1$ i $8/2$, pa dobijamo da je $i = 2$ traženi indeks. Prema tome, pivot je element $\bar{a}_{1,2} = 2$. Sada izvršimo elementarne operacije pivotiranja oko ovog elementa. Primjenom Gausove metode eliminacije, elemente u koloni 1 simpleks tabele, osim elementa druge vrste, svedemo na nule, dok element u drugoj vrsti prve kolone postavimo na 1. Vrsti 1 se dodaje druga vrsta pomnožena sa $-\frac{1}{2}$. Druga vrsta se dijeli sa $\frac{1}{2}$. Trećoj vrsti se dodaje druga vrsta pomnožena sa $-\frac{1}{2}$. Primjenjujući ove transformacije, dobijamo simpleks tabelu:

0	$\frac{5}{2}$	1	$-\frac{1}{2}$	5
1	$\frac{1}{2}$	0	$\frac{1}{2}$	4
0	$\frac{1}{2}$	0	$-\frac{1}{2}$	-4

Tabela 4.3: Simpleks tabela nakon prvog pivotiranja

Bazne promjenljive sada lako prepoznajemo iz Tabele 4.3, jer odgovarajući vektori kolone zajedno formiraju jediničnu matricu. Prema tome, nebazne promjenljive su x_2 i s_2 , koje dobijaju vrijednost 0, dok ostale lako izračunavamo, odakle je $x = (4, 0, 5, 0)$. Ponavljamo prethodne korake nad novom simpleks tabelom. Potrebno je pronaći novi pivot element. Lako se vidi da je jedino kolona 2 odgovarajuća. Pošto je $5/(5/2) < 4/(1/2)$, zaključujemo da pivot element pronalazimo u prvoj vrsti. Dakle, pivot je $\bar{a}_{1,2} = \frac{5}{2}$. Postupkom Gausove eliminacije oko prve vrste, dobija se simpleks Tabela 4.4.

0	1	$\frac{2}{5}$	$-\frac{1}{5}$	2
1	0	$-\frac{1}{5}$	$\frac{3}{5}$	3
0	0	$-\frac{1}{5}$	$-\frac{2}{5}$	-5

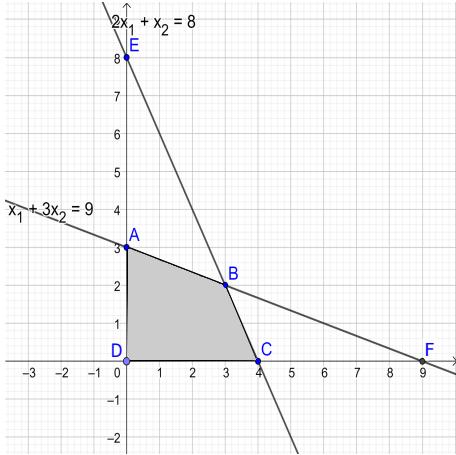
Tabela 4.4: Simpleks tabela: korak 2.

Kako su sve vrijednosti doprinosa manje (ili jednake) 0, zaključujemo da smo pronašli optimum koji je jednak $x = (3, 2, 0, 0)$, a optimalna vrijednost je $f = 5$.

Analizirajmo još neke aspekte u ovom primjeru. Na Slici 4.1, tačke A, B, C , i D su bazna dopustva rješenja. Tačke E i F su takođe bazna rješenja, ali nisu dopustiva. Simpleks algoritam prvo kreće od tačke D prema tački C , te posljednjom iteracijom završava u tački B (optimum).

Iz posljednje simpleks tabele (Tabela 4.4) se može uočiti i sljedeće:

- Jedinična matrica $m \times m$ ugrađena u gornju lijevu podmatricu simpleks tabele (blok sa elementima $\bar{a}_{i,j}$) sadrži kolone koje čine vektore baznih promjenljivih.



Slika 4.1: Region pretrage (bazna rješenja označena).

- U odgovarajućim kolonama u posljednjoj vrsti (ciljna vrsta) se nalaze nule.

Prema tome, vektori x_i , koji su bazni (kolone baze B), su napisani preko nebaznih vektora $x_i \in N$. Kako je $x_i = 0$, za sve $i \in N$, vrijednosti baznih vektora x_i , tj. kolona baze B , se trivijano izračunavaju.

Primjer 4.2. Navedimo i primjer jednog linearog programa koji je neograničen, te ga riješimo simpleks metodom.

Neka je dat problem LP-a:

$$\begin{aligned}
 & \max -x + 3y \\
 & \text{t.d.} \\
 & -x + y \leq 3 \\
 & x - 2y \leq 2 \\
 & x, y \geq 0.
 \end{aligned}$$

Prevedimo problem na kanonski oblik, dodavanjem izjednačavajućih promjenljivih.

$$\begin{aligned} & \max -x + 3y \\ & \text{t.d.} \\ & -x + y + s_1 = 3 \\ & x - 2y + s_2 = 2 \\ & (x, s) \geq 0. \end{aligned}$$

Ovom programu odgovara simpleks tabela:

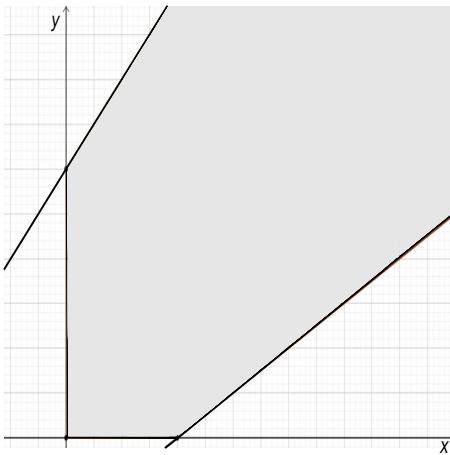
$$\begin{array}{cccc|c} -1 & 1 & 1 & 0 & 3 \\ 1 & -2 & 0 & 1 & 2 \\ \hline -1 & 3 & 0 & 0 & 0 \end{array}$$

Kako rješavamo problem minimizacije, tražićemo pivota u koloni 2. Jedinji kandidat za pivota je element $\bar{a}_{1,2} = 1$. Pivotiranjem oko tog elementa dobijamo novu simpleks tabelu

$$\begin{array}{cccc|c} -1 & 1 & 1 & 0 & 3 \\ -1 & 0 & 2 & 1 & 8 \\ \hline 2 & 0 & -3 & 0 & -9 \end{array}$$

Kandidata za pivota tražimo u prvoj koloni. Međutim, kako su svi elementi prve kolone negativni, to znači da se vrijednost promjenljive x može povećavati proizvoljno, odakle slijedi da nije ograničena. Grafička reprezentacija dopustivog, neograničenog regiona, prikazana je na Slici 4.2.

Primjer 4.3. U nastavku dajemo primjer kada pivotiranje u simpleks metodi generiše degenerativno rješenje.



Slika 4.2: Neograničen region.

Posmatrajmo sljedeći linearni program:

$$\max 2x + 3y$$

t.d.

$$-x + y \leq 3$$

$$x - 2y \leq 2$$

$$x + 6y \leq 18$$

$$x, y \geq 0.$$

Slično kao i u prethodnom primjeru, dodavanjem izjednačavajućih promjenljivih, dobijamo simpleks tabelu

-1	1	1	0	0	3
1	-2	0	1	0	2
1	6	0	0	1	18
2	3	0	0	0	0

Prepostavimo smo za baznu promjenljivu izabrali y , tj. pivotiramo oko nekog od elemenata u koloni 2 (zanemarimo ovdje pomenuto Blandovo pravilo

odabira pivota). Za pivota odaberemo $\bar{a}_{1,2} = 1$ element, te nakon pivotiranja dobijamo novu simpleks tabelu:

$$\begin{array}{ccccc|c} -1 & 1 & 1 & 0 & 0 & 3 \\ -1 & 0 & 2 & 1 & 0 & 8 \\ 7 & 0 & -6 & 0 & 1 & 0 \\ \hline 5 & 0 & -3 & 0 & 0 & -9 \end{array}$$

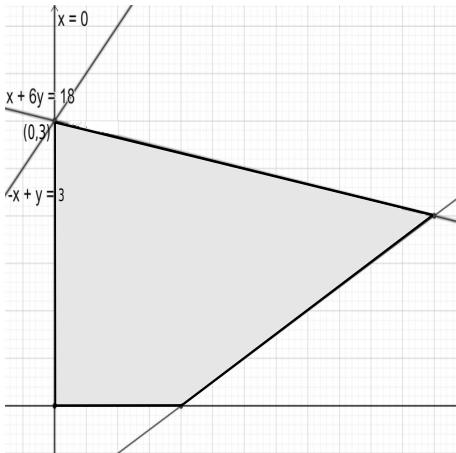
Ovdje počinju problemi. Pivotiranje bi trebalo da ide oko nekog elementa u prvoj koloni. Kako tu imamo samo jedan element strogovo veći od 0, a to je $\bar{a}_{3,1} = 7$, on je ujedno i pivot. Izvršavajući operaciju pivotiranja, promjenljive \bar{b} neće promijeniti svoje vrijednosti, tj. dobijamo sljedeću tabelu:

$$\begin{array}{ccccc|c} 0 & 1 & \frac{1}{7} & 0 & \frac{1}{7} & 3 \\ 0 & 0 & \frac{8}{7} & 1 & \frac{1}{7} & 8 \\ 1 & 0 & -\frac{6}{7} & 0 & \frac{1}{7} & 0 \\ \hline 0 & 0 & \frac{9}{7} & 0 & -\frac{5}{7} & -9 \end{array}$$

Geometrijski gledano, razlog za ovakvu situaciju su tri prave $x = 0$, $-x + y = 3$ i $x + 6y = 18$ koje se susreću u tački $(0, 3)$ (Slika 4.3). U prošlom koraku, kada su x i s_1 bile nebazne promjenljive, tačka $(0, 3)$ je posmatrana kao presjek $x = 0$ i $-x + y = 3$. Nakon sljedećeg koraka s_1 i s_3 postaju nebazne, pa se pomjeramo u tački presjeka pravih $-x + y = 3$ i $x + 6y = 18$, što je opet ista tačka (prema tome, nema ni poboljšanja vrijednosti funkcije cilja). Ovo pivotiranje se naziva *degenerativno pivotiranje*. U ovom primjeru, napišimo vrh $(0, 3)$ nakon sljedeće iteracije pivotiranja. U komplikovanim primjerima, kada se mnoga ograničenja mogu sresti u jednom vrhu, ostanak u istom vrhu u narednim simpleks koracima je velika opasnost. Ovaj slučaj se može izbjegći poštovanjem Blandovog pravila. Primijetimo da koristeći Blandovo pravilo u prvom koraku simpleks metode – uzimajući za pivota elemente iz kolone 1, a ne kolone 2 – degenerativno pivotiranje se ne bi pojavilo.

4.4 Dvofazni simpleks metod

Postoje dva pitanja koja se nameću u primjeni simpleks algoritma:



Slika 4.3: Tačka presjeka $(0, 3)$ gdje se javlja degenerativno pivotiranje.

- Da li se uvijek (kao i koliko teško) može naći bazno dopustivo rješenje za inicijalni korak simpleks algoritma?
- Da li se simpleks algoritam uvijek prekida pronalaskom optimalnog rješenja ili je funkcija cilja neograničena?

Ako postoji jedinična podmatrica I_m u gornjoj lijevoj blok matrici simpleks tabele, onda je jasno da je odgovor na prvo pitanje pozitivan. U primjeru 4.1 su linearnom programu, koji sadrži ograničenja $Ax \leq b, x \geq 0$ pridodate izjednačavajuće promjenljive, kako bi se uslovi nejednakosti u ograničenjima transformirali u potrebne jednakosti. Na taj način je formirana i jedinična matrica, pa je i bazu jednostavno odrediti.

Međutim, šta ako je linearan program direktno dat u kanonskom obliku, tj. ograničenjima oblika $Ax = b, x \geq 0$? Bez smanjenja opštosti, pretpostavimo da je $b \geq 0$. Ako ne postoji očigledno bazno dopustivo rješenje ovog problema, onda uvodimo *vještačke* promjenljive (eng. auxiliary variables)

$s = (s_1, \dots, s_m)$. Nakon toga, rješavamo sljedeći problem LP-a:

$$\begin{aligned} & \min_{x,s} \sum_{i=1}^m s_i \\ & \text{t.d.} \\ & Ax + s = b \\ & x, s \geq 0. \end{aligned} \tag{4.10}$$

Postoje dva moguća slučaja:

1. Ako početni problem LP-a sa uslovima koji su jednakosti (4.1) ima optimalno rješenje, onda rješenje LP-a (4.10) ima (optimalno) rješenje $s = 0$.
2. Ako (4.1) nema dopustivo rješenje, onda je barem jedan $s_i > 0$.

Prema tome, kada prvo riješimo problem LP-a (4.10) simpleks metodom, možemo da zaključimo:

- Ako se desio slučaj 2, odustajemo od daljeg rješavanja problema.
- Ako se desio slučaj 1, optimalno rješenje problema (4.10) sa $s_i = 0$ nam daje početno bazno dopustivo rješenje problema (4.1) u prvom simpleks koraku, a, prema tome, i bazne vektore.

Ovaj korak nas vodi ka *dvofaznoj simpleks metodi*. U nastavku je prikazan primjer kako se problem linearнog programiranja rješava primjenom ove metode.

Primjer 4.4. Primjenom dvofaznog simpleks metoda riješiti sljedeći problem LP-a:

$$\min x_1 + x_2 - x_3 - x_4$$

t.d.

$$x_1 + 2x_2 + x_3 + x_4 = 7$$

$$2x_1 - x_2 - x_3 - 3x_4 = -1$$

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

Dodajmo vještačke promjenljive na početni problem, uz prethodno množenje ograničenja 2 sa (-1), odakle dobijamo:

$$\min x_1 + x_2 - x_3 - x_4$$

t.d.

$$x_1 + 2x_2 + x_3 + x_4 + s_1 = 7$$

$$-2x_1 + x_2 + x_3 + 3x_4 + s_2 = 1$$

$$x_1, x_2, x_3, x_4, s_1, s_2 \geq 0.$$

Prva faza metoda je rješavanje problema

$$\min s_1 + s_2$$

t.d.

$$x_1 + 2x_2 + x_3 + x_4 + s_1 = 7$$

$$-2x_1 + x_2 + x_3 + 3x_4 + s_2 = 1$$

$$x_1, x_2, x_3, x_4, s_1, s_2 \geq 0.$$

Formirajmo početnu simpleks tabelu.

1	2	1	1	1	0	7
-2	1	1	3	0	1	1
0	0	0	0	1	1	0

Množeći prvu i drugu vrstu prethodne tabele sa (-1) , te sabiranjem sa trećom vrstom, dobijamo tabelu

$$\begin{array}{ccccccc|c} 1 & 2 & 1 & 1 & 1 & 0 & 7 \\ -2 & 1 & 1 & 3 & 0 & 1 & 1 \\ \hline 1 & -3 & -2 & -4 & 0 & 0 & -8 \end{array}$$

Pošto rješavamo problem minimizacije, u prethodnoj tabeli potražimo promjenljivu sa negativnim koeficijentom doprinosa. Prva takva promjenljiva je u koloni 2, u kojoj i tražimo pivota. Kako je $\frac{7}{2} > \frac{1}{1}$, slijedi da je pivot $\bar{a}_{2,2} = 1$. Izvršimo transformacije, tako što postavimo sve nule u datoj koloni. Dobijamo novu tabelu:

$$\begin{array}{ccccccc|c} 5 & 0 & -1 & -5 & 1 & -2 & 5 \\ -2 & 1 & 1 & 3 & 0 & 1 & 1 \\ \hline -5 & 0 & 1 & 5 & 0 & 3 & -5 \end{array}$$

Jedini vektor sa negativnim doprinosom u prethodnoj simpleks tabeli je onaj koji odgovara prvoj koloni, pa u njoj i tražimo pivota. Kako se jedino u prvoj vrsti nalazi pozitivan element, koji odgovara vrijednosti pravca $\frac{5}{5}$, jasno je da je pivot $\bar{a}_{1,1} = 5$. Pivotiranjem oko ovog elementa, dobijamo tabelu

$$\begin{array}{ccccccc|c} 1 & 0 & -\frac{1}{5} & -1 & \frac{1}{5} & -\frac{2}{5} & 1 \\ 0 & 1 & \frac{3}{5} & 1 & \frac{3}{5} & \frac{1}{5} & 3 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}$$

Kako su koeficijenti doprinosa svi veći ili jednaki 0, našli smo minimum. Prema tome, bazno dopustivo rješenje je jednako $(x, s) = (1, 3, 0, 0, 0, 0)$.

Sada nastavljamo dalje sa algoritmom, odnosno sa drugom fazom metoda. Posljednja simpleks tabela prve faze dvofaznog simpleks metoda se lako transformiše za početni problem:

- posljednja vrsta dobija koeficijente originalne funkcije cilja razmatranog LP-a;
- kolone koje odgovaraju promjenljivima s izbrišemo iz simpleks tabele;

-
- ostale elemente tabele ne mijenjamo.

Iz ovih koraka, dobijamo simpleks tabelu

$$\begin{array}{cccc|c} 1 & 0 & -\frac{1}{5} & -1 & 1 \\ 0 & 1 & \frac{3}{5} & 1 & 3 \\ \hline 1 & 1 & -1 & -1 & 0 \end{array}$$

Načinimo sada elementarne transformacije na ovoj tabeli; kandidata za pivota nađimo u koloni 3. Jedini element sa pozitivnom vrijednošću je $\bar{a}_{2,3} = \frac{3}{5}$, koji je onda i traženi pivot. Vršeći elementarne transformacije oko pivota, dobijamo tabelu

$$\begin{array}{cccc|c} 1 & \frac{1}{3} & 0 & -\frac{4}{3} & 2 \\ 0 & \frac{5}{3} & 1 & \frac{5}{3} & 5 \\ \hline 1 & \frac{8}{3} & 0 & \frac{2}{3} & 3 \end{array}$$

Pošto su svi doprinosi svi veći od 0, našli smo optimalnu simpleks tabelu. Optimalna rješenja su $x_1 = 2, x_3 = 5$, pa je optimum jednak -3.

4.5 = geometrijskoj interpretaciji simpleks metoda

Sljedeći primjer demonstrira odgovarajuće geometrijske korake („šetanje“) po dopustivom regionu, u skladu sa iteracijama simpleks metode. Neka je dat problem:

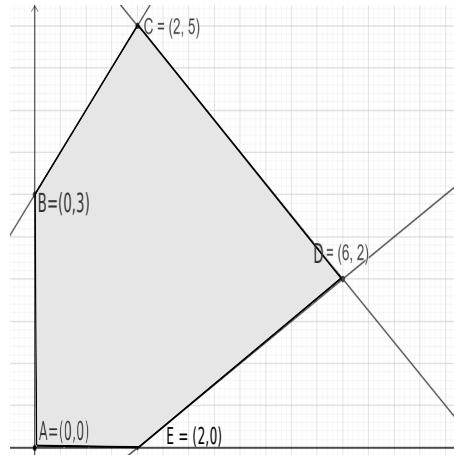
$$\begin{aligned} & \max 2x + 3y \\ & \text{t.d.} \\ & -x + y \leq 3 \\ & x - 2y \leq 2 \\ & 3x + 4y \leq 26 \\ & x, y \geq 0. \end{aligned}$$

Dodajmo izjednačavajuće promjenljive u početni LP, odakle dobijamo

$$\begin{aligned} & \max 2x + 3y \\ \text{t.d.} \quad & -x + y + s_1 = 3 \\ & x - 2y + s_2 = 2 \\ & 3x + 4y + s_3 = 26 \\ & x, y, s_1, s_2, s_3 \geq 0. \end{aligned}$$

Inicijalna simpleks tabela je data sa

	x	y	s_1	s_2	s_3	
s_1	-1	1	1	0	0	3
s_2	1	-2	0	1	0	2
s_3	3	4	0	0	1	26
$-z$	2	3	0	0	0	0



Slika 4.4: Korak 1

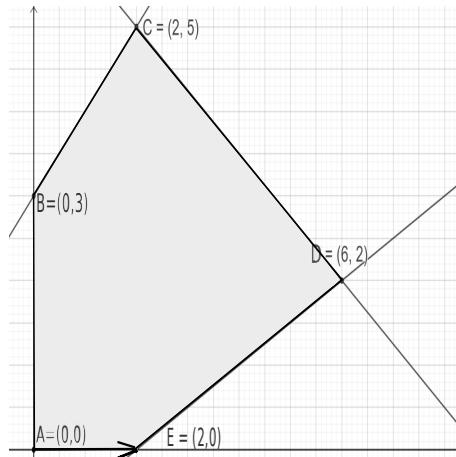
Kao što je već pomenuto, izjednačavajuće promjenljive čine pogodne bazne promjenljive za početak simpleks metode, jer one formiraju jediničnu podmatricu u gornjoj blok matrici simpleks tabele. Prema tome, bazno rješenje je $(x, y, s_1, s_2, s_3) = (0, 0, 3, 2, 26)$, koje je ujedno i dopustivo. Na Slici 4.4 je prikazan dopustivi region problema. Nakon prvog koraka, nalazimo se u tački $A = (0, 0)$, koja je dopustiva. Trenutna najbolja vrijednost je jednaka 0. Izvršimo sada promjenu baznih vektora na osnovu koraka simpleks metoda. Pošto maksimizujemo, tražimo one koeficijente dopirnosa koji su veći od 0. Biramo prvu kolonu, koja odgovara promjenljivoj x . Pronadimo pivot element: kandidati su s_2 i s_3 . Kako je $\frac{26}{3} > \frac{2}{1}$, onda je pivot $\bar{a}_{2,1} = 1$. Pivotiramo oko ovog elementa i dobijemo tabelu:

	s_2	y	s_1	s_2	s_3	
s_1	0	-1	1	1	0	5
x	1	-2	0	1	0	2
s_3	0	10	0	-3	1	20
$-z$	0	7	0	-2	0	-4

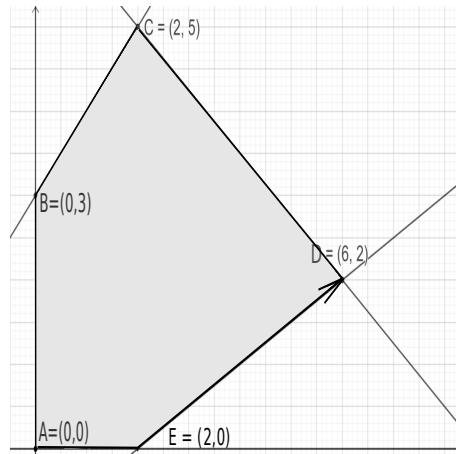
Ova tabela vizuelno odgovara Slici 4.5. Prelazi se u bazno rješenje $(x, y, s_1, s_2, s_3) = (2, 0, 5, 0, 20)$, odnosno do tačke $E = (2, 0)$.

Dalje, narednog pivota nalazimo u drugoj koloni, koja odgovara promjenljivoj y . Jedini kandidat za pivota je element $\bar{a}_{3,2} = 10$, pa nakon pivotiranja dobijamo tabelu

	s_2	s_3	s_1	s_2	s_3	
s_1	0	0	1	$\frac{7}{10}$	$\frac{1}{10}$	7
x	1	0	0	$\frac{2}{5}$	$\frac{1}{5}$	6
y	0	1	0	$-\frac{3}{10}$	$\frac{1}{10}$	2
$-z$	0	0	0	$\frac{1}{10}$	$-\frac{7}{10}$	-18



Slika 4.5: Korak 2



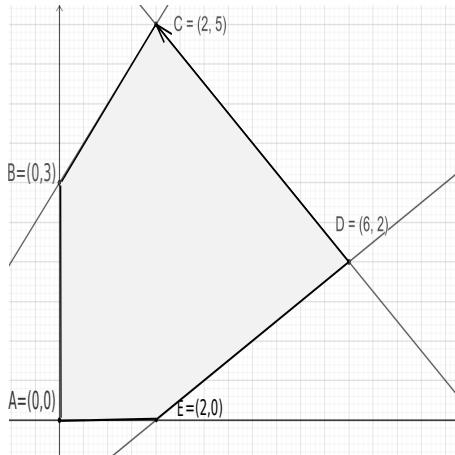
Slika 4.6: Korak 3

Ova tabela odgovara kretanju od tačke $E = (2, 0)$ do tačke $D = (6, 2)$, što je i prikazano na Slici 4.6. Zaključujemo da je jedina koordinata koja odgovara pozitivnom doprinosu kolona 4 simpleks tabele. Pivotiranje vršimo

oko elementa $\bar{a}_{1,4} = \frac{7}{10}$ i dobijamo simpleks tabelu

	s_2	s_3	s_1	s_2	s_3	
s_2	0	0	$\frac{10}{7}$	1	$\frac{1}{7}$	10
x	1	0	$\frac{4}{7}$	0	$\frac{1}{7}$	2
y	0	1	$\frac{3}{7}$	0	$\frac{1}{7}$	5
$-z$	0	0	$-\frac{1}{10}$	0	$-\frac{5}{7}$	-19

Kako ne postoje koordinate čiji su doprinosi pozitivni, našli smo optimum, koji se dostiže u tački $C = (2, 5)$. Prethodna simpleks tabela odgovara Slici 4.7, dok sve iteracije u simpleks metodi odgovaraju putanji $A \rightarrow E \rightarrow D \rightarrow C$.



Slika 4.7: Korak 4

U slučaju da smo, u prvom koraku simpleks metode, umjesto promjenljive x u bazu uključili promjenljivu y , do optimuma bismo došli pomjeranjem u tačku B , a potom u tačku C , koja je i optimum. U ovom slučaju bi bilo potrebno generisati samo dvije simpleks tabele (a ne tri, kao što smo imali u ovom primjeru).

4.6 Zadaci

1. Uz pomoć Simpleks metoda, riješiti sljedeći problem LP-a:

$$\max 3x + 2y$$

t.d.

$$2x + y \leq 18$$

$$2x + 3y \leq 42$$

$$3x + y \leq 24$$

$$x, y \geq 0.$$

Uporedite dobijeno rješenje sa rješenjem koje se dobije koristeći grafičku metodu.

2. Koristeći simpleks metodu, riješiti sljedeći problem LP-a:

$$\max 2x + 3y + z$$

t.d.

$$3x + 5y \leq 5$$

$$2x + y - z \leq 13$$

$$z \leq 4$$

$$x, y, z \geq 0.$$

3. Uz pomoć dvofaznog simpleks metoda, riješiti sljedeći problem LP-a:

$$\min x_1 + x_2 + x_3$$

t.d.

$$x_1 + 2x_2 + 3x_3 = 3$$

$$-x_1 + 2x_2 + 6x_3 = 2$$

$$-4x_2 - 9x_3 = -5$$

$$3x_3 + x_4 = 1$$

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

-
4. Uz pomoć simpleks metoda, riješiti sljedeći problem LP-a:

$$\begin{aligned} & \min -x_1 - x_2 - x_3 \\ & \text{t.d.} \\ & 2x_1 - x_2 + 2x_3 + x_4 = 4 \\ & 2x_1 - 3x_2 + x_3 + x_5 = -5 \\ & -x_1 + x_2 - 2x_3 + x_6 = -1 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0. \end{aligned}$$

5. Uz pomoć simpleks metoda, riješiti sljedeći problem LP-a:

$$\begin{aligned} & \max 2x_1 + x_2 \\ & \text{t.d.} \\ & 4x_1 + 3x_2 \leq 12 \\ & 4x_1 + x_2 \leq 8 \\ & 4x_1 + 2x_2 \leq 8 \\ & x_1, x_2 \geq 0. \end{aligned}$$

Da li se u nekom koraku simpleks metode javlja degenerativno bazno rješenje? Ako je to slučaj, diskutovati zašto je ono degenerativno.

6. Iako je simpleks metod u prosječnom slučaju efikasan metod za rješavanje LP problema, u najgorem slučaju, vrijeme izvršenja eksponentijalno zavisi od broja vrhova dopustivog skupa. Koristeći pouzdane internet izvore, pronađi i analiziraj primjer LP problema sa D promjenljivih i D ograničenja, kod koga simpleks metod obilazi svih 2^D vrhova politopa. Jedan od primjera takvog politopa je tzv. Kli-Mintijeva kocka (eng. Klee-Minty cube).

Glava 5

Dualnost i dopustivost

U teoriji linearog programiranja, ali i teoriji optimizacije uopšte, pod dualnošću podrazumijevamo da se neki problem može posmatrati s dva različita aspekta: kao primalni i kao dualni problem. Preciznije, teorija dualnosti povezuje dva problema linearog programiranja – jedan od njih je problem linearog programiranja koji maksimizuje funkciju cilja, dok je drugi linearan program koji minimizuje (ne obavezno istu) funkciju cilja. Na osnovu istih ulaznih podataka, svaki problem LP-a se može navesti u drugom, ekvivalentnom obliku. Ponekad rješavanje dualnog problema LP-a može da bude puno lakše nego rješavanje početnog problema. Dualnost implicira da svaki problem linearog programiranja možemo analizirati na dva potpuno različita načina, ali sa ekvivalentnim (optimalnim) rješenjima.

U ovom poglavlju ćemo najprije objasniti kako se od početnog, primarnog problema formira dualni problem. Nakon toga, biće navedene Teoreme o slaboj i jakoj dualnosti, koje daju vezu između rješenja i ograničenja primarnog i dualnog problema. U nastavku poglavlja su objašnjene još neke važne tehnike koje se koriste za rješavanje problema linearog programiranja: Farkašova lema i Furije-Mockinova eliminacija. Dodatno, metod unutrašnje tačke će biti dat u Dodatku A.

Prije nego što ćemo uvedemo pojam dualnog problema, objasnimo motivaciju koja stoji iza nastanka teorije dualnosti.

Primjer 5.1. Posmatrajmo jedan LP i probajmo naći što bolju gornju procjenu za optimalnu vrijednost funkcije cilja na osnovu ograničenja u datom modelu.

$$\begin{array}{lllll} \max f((x_1, x_2)) = & x_1 & + & x_2 & \\ \text{t.d.} & 4x_1 & + & 3x_2 & \leq 9 \\ & x_1 & + & x_2 & \leq 8 \\ & x \geq 0 & & & \end{array}$$

Iz uslova je jasno da su gornje granice date sa $f(x_1, x_2) = x_1 + x_2 \leq 4x_1 + 3x_2 \leq 9$, kao i $f(x_1, x_2) \leq x_1 + x_2 \leq 8$. Gornje granice se mogu i poboljšati na osnovu sljedećeg niza nejednakosti

$$f(x_1, x_2) \leq \frac{1}{4}(4x_1 + 3x_2) + \frac{1}{4}(x_1 + x_2) = \frac{5}{4}x_1 + x_2 \leq \frac{9}{4} + 2 = \frac{17}{4},$$

čime dobijamo bolju gornju granicu vrijednosti funkcije f .

Izvedimo sada ovaku konstrukciju u opštijem obliku. Iskoristimo ograničenja data u modelu, i iskombinujmo ih sa koeficijentima funkcije cilja na sljedeći način. Neka su $y_1, y_2 \geq 0$ koeficijenti sa kojima množimo ograničenja u početnom modelu. Tada imamo

$$\begin{aligned} y_1 \cdot (4x_1 + 3x_2) &\leq 9y_1 \\ y_2 \cdot (x_1 + x_2) &\leq 8y_2 \end{aligned}$$

Sabirajući ove dvije nejednakosti i grupišući sabirke dobijamo

$$x_1(4y_1 + y_2) + x_2(3y_1 + y_2) \leq 9y_1 + 8y_2 \quad (5.1)$$

Posmatrajući funkciju cilja i koeficijente uz promjenljive u (5.1), jasno je da treba da vrijedi $4y_1 + y_2 \geq 1$ i $3y_1 + y_2 \geq 1$. Da bi ocjena gornje granice izraza sa lijeve strane nejednakosti (5.1) bila što bolja, potrebno je minimizovati izraz $9y_1 + 8y_2$, prema prethodno pomenuta (dva) uslova. Prema tome, početnom problemu, koji označavamo sa (P) , odgovara dualni

problem (D)

$$\begin{aligned} \min \quad & 9y_1 + 8y_2 \\ \text{t.d.} \quad & 4y_1 + y_2 \geq 1 \\ & 3y_1 + y_2 \geq 1 \\ & y = (y_1, y_2) \geq 0. \end{aligned}$$

U opštem slučaju, za problem (P)

$$\begin{aligned} \max \quad & c^T x \\ \text{t.d.} \quad & Ax \leq b \\ & x \geq 0. \end{aligned}$$

odgovorarajući dualni problem (D) je dat sa

$$\begin{aligned} \min \quad & b^T y \\ \text{t.d.} \quad & A^T y \geq c \\ & y \geq 0. \end{aligned}$$

5.1 Teoreme dualnosti

U ovom odjeljku ćemo navesti najvažnije teoreme koje omogućavaju primjenu teorije dualnosti na rješavanje problema linearнog programiranja.

Teorema 5.1 (Teorema slabe dualnosti). *Ako je x dopustivo rješenje za problem (P) i y dopustivo rješenje za problem (D), onda je*

$$c^T x \leq b^T y.$$

Dokaz. Kako je $A^T y \geq c$, transponovanjem obje strane dobijemo $y^T A \geq c^T$. Množenjem obje strane posljednje nejednakosti sa $x \geq 0$, dobijamo

$$y^T A x \geq c^T x.$$

Kako je $Ax \leq b$ i $y \geq 0$, tada dobijamo $y^T b = b^T y \geq c^T x$

□

Iz Teoreme slabe dualnosti slijedi sljedeća važna posljedica.

Posljedica 5.1. *Ako je y dopustivo rješenje problema (D) , onda bilo koje dopustivo rješenje x problema (P) je ograničeno odozgo sa $b^T y$. Prema tome, ako je problem (D) dopustiv, onda je (P) ograničen. Slično se može pokazati da vrijedi i obrnuto, kada problemi (P) i (D) zamijene uloge.*

Pokažimo da važi i naredno tvrđenje, koje ćemo koristiti u dokazu Teoreme jake dualnosti.

Propozicija 5.1. *Ako je x^* dopustivo rješenje za (P) , y^* dopustivo rješenje za (D) i ako vrijedi $c^T x^* = b^T y^*$, onda je x^* optimalno rješenje za problem (P) , a y^* optimalno rješenje za problem (D) .*

Dokaz. Neka je x bilo koje dopustivo rješenje problema (P) . Iz teoreme o slaboj dualnosti imamo $c^T x \leq b^T y^* = c^T x^*$, odakle slijedi da je x^* upravo optimalno rješenje problema (P) . Slično vrijedi i za sva dopustiva rješenja problema (D) : $b^T y^* = c^T x^* \geq b^T y$, odakle imamo da je y^* optimalno rješenje problema (D) . \square

Teorema 5.2 (Teorema jake dualnosti). *Ako problem (P) ima optimalno rješenje x^* , onda i problem (D) ima optimalno rješenje y^* i pri tome je $c^T x^* = b^T y^*$.*

Dokaz. Napišimo ograničenja problema (P) kao $Ax+s=b$, $x, s \geq 0$. Primijenimo simpleks metod, pa posmatrajmo posljednju simpleks tabelu, odnosno tabelu iz koje se dobija rješenje x^* .

$\overbrace{x \text{ promjenljive}}$		$\overbrace{s \text{ promjenljive}}$	
$c_1^* c_2^* \dots c_n^*$		$-y_1^* - y_2^* \dots - y_m^*$	$-f^*$

gdje je

1. $c_i^* \leq 0, i = 1, \dots, n$
2. $-y_i^* \leq 0, i = 1, \dots, m$

-
3. $c^T x - f^* = (c^*)^T x - (y^*)^T s$ (iz koeficijenata doprinosa simpleks metode)

Prema tome,

$$\begin{aligned} c^T x &= f^* + (c^*)^T x - (y^*)^T s \\ &= f^* + (c^*)^T x - (y^*)^T (b - Ax) \\ &= f^* - (y^*)^T b + ((c^*)^T + (y^*)^T A)x \end{aligned}$$

koje vrijedi za sve x . Specijalno, za $x = 0$, dobijamo

$$f^* = (y^*)^T b. \quad (5.2)$$

Dalje, važi i $c^T x = ((c^*)^T + (y^*)^T A)x$, odnosno $c = A^T y^* + c^*$.

Kako je $c^* \leq 0$, imamo $A^T y^* \geq c$. Kako je $-y^* \leq 0$, slijedi da je y^* dopustivo rješenje za (D) . Iz (5.2) imamo da je vrijednost funkcije cilja u y^* upravo $f^* = c^T x^*$. Iz Propozicije 5.1 slijedi da je y^* optimalno rješenje problema (D) .

□

Primijetimo da koeficijenti y^* posljednje vrste završne simpleks tabele odgovaraju „izjednačavajućim” promjenljivima i one nam daju rješenje problema (D) .

Primjer 5.2. Moguće je da problem (P) i njegov dual (D) nemaju dopustivo rješenje. Pogledajmo sljedeći primjer:

$$\begin{array}{ll} \max & 2x_1 - x_2 \\ \text{t.d.} & x_1 - x_2 \leq 1 \\ & -x_1 + x_2 \leq -2 \\ & x_1, x_2 \geq 0. \end{array}$$

Rješenje. Množenjem drugog ograničenja sa -1 , dobijamo $x_1 - x_2 \geq 2$, što u kombinaciji sa prvim ograničenjem jasno pokazuje da problem (P) nema

dopustivih rješenja. Dualni problem (D) ovog LP-a je dat sa

$$\begin{aligned} \min \quad & x_1 - 2x_2 \\ \text{t.d.} \quad & y_1 - y_2 \geq 2 \\ & -y_1 + y_2 \geq -1 \\ & y_1, y_2 \geq 0. \end{aligned}$$

Množeći drugo ograničenje sa -1 , dobijamo $y_1 - y_2 \leq 1$, što u kombinaciji sa prvim ograničenjem jasno pokazuje da problem (D) nema dopustivo rješenje.

Analizirajmo sada i koncept *komplementarnosti* (eng. *complementary slackness*), koji nam omogućava da uspostavimo još neke važne odnose između promjenljivih i ograničenja primarnog i dualnog problema.

Teorema 5.3 (Teorema komplementarnosti). *Neka su x i y dopustiva rješenja problema (P) i (D), redom. Vektori x i y su optimalna rješenja ova dva problema akko vrijedi*

$$(Ax - b)_i y_i = (a_{i1}x_1 + \dots + a_{in}x_n - b_i)y_i = 0, i = 1, \dots, m \quad (5.3)$$

i

$$(A^T y - c)_j x_j = (a_{1j}y_1 + \dots + a_{mj}y_m - c_j)x_j = 0, j = 1, \dots, n \quad (5.4)$$

Uslovi (5.4) i (5.3) se nazivaju *uslovi komplementarnosti*.

Dokaz. Iz teoreme o slaboj dualnosti imamo:

$$c^T x \leq (A^T y)^T x = y^T A x \leq b^T y. \quad (5.5)$$

Dalje, iz teoreme jake dualnosti imamo

$$\begin{aligned} x \text{ i } y \text{ su oba optimalna} &\iff c^T x = b^T y \iff c^T x = y^T A x = b^T y \\ &\iff (y^T A - c^T)x = 0 \wedge y^T(Ax - b) = 0 \\ &\iff \sum_i (A^T y - c)_i x_i = 0 \wedge \sum_j (Ax - b)_j y_j = 0 \end{aligned}$$

Kako je $A^T y \geq c$, $\sum_i (A^T y - c)_i x_i$ je suma nenegativnih komponenti, odakle slijedi da $(A^T y - c)_i x_i = 0$ za sve i . Takođe, slično vrijedi i za drugu sumu, jer $Ax - b \leq 0$, pa je svaki $(Ax - b)_j y_j$ negativan, odakle slijedi da je $(Ax - b)_j y_j = 0$ za sve j . \square

Komentar. Nekada je lakše riješiti jedan od problema, (P) ili (D) . Npr. lakše je riješiti dualni problem sa dvije promjenljive i 4 ograničenja (grafički metod), nego primalni (P) problem sa 4 promjenljive i 2 ograničenja.

Primjer 5.3. Posmatrajmo primarni (P) i dualni (D) problem, gdje su

$$A = \begin{pmatrix} 1 & 4 & 0 \\ 3 & -1 & 1 \end{pmatrix}, b = \begin{pmatrix} 1 \\ 3 \end{pmatrix}, c = \begin{pmatrix} 4 \\ 1 \\ 3 \end{pmatrix}$$

Posmatrajmo jedno dopustivo rješenje problema (P) , $x = (0, \frac{1}{4}, \frac{13}{4})$. Da li je ono optimalno? Ako bi bilo optimalno, prema Teoremi komplementarnosti, vrijedilo bi sljedeće:

$$x_2 \geq 0 \Rightarrow (A^T y)_2 = c_2, \text{ tj. } 4 \cdot y_1 - y_2 = 1$$

$$x_3 \geq 0 \Rightarrow (A^T y)_3 = c_3, \text{ tj. } 0 \cdot y_1 + y_2 = 3$$

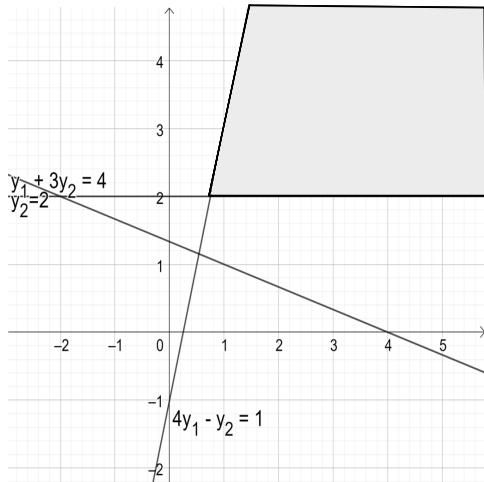
što nam daje $y = (1, 3)$. Preostalo ograničenje $y_1 + 3y_2 \geq 4$ takođe vrijedi za vektor y , odakle imamo da je y dopustivo rješenje odgovarajućeg problema (D) . Prema tome, zaključujemo da su x i y dopustiva rješenja koja zadovoljavaju svojstvo komplementarnosti, pa su to i optimalna rješenja.

Optimalno rješenje je u presjeku pravih $y_2 = 3$ i $4y_1 - y_2 = 1$, odnosno $y = (1, 3)$. Provjerimo, pomoću Teoreme komplementarnosti, potencijalne tačke za optimalno rješenje problema (P) . Imamo

$$y_1 > 0 \Rightarrow (Ax - b)_1 y_1 = 0 \Rightarrow x_1 + 4x_2 = 4$$

$$y_2 > 0 \Rightarrow (Ax - b)_2 y_2 = 0 \Rightarrow 3x_1 - x_2 + x_3 = 3$$

i kako je $y_1 + 3y_2 > 4$, onda $(A^T y)_1 > c_1$ i prema tome $x_1 = 0$, odakle $x = (0, \frac{1}{4}, \frac{13}{4})$.



Slika 5.1: Dualni problem: dopustiv region

Primjer 5.4. Posmatrajmo problem LP-a

$$\begin{aligned}
 \max \quad & 10x_1 + 10x_2 + 20x_3 + 20x_4 \\
 \text{t.d.} \quad & 12x_1 + 8x_2 + 6x_3 + 4x_4 \leq 210 \\
 & 3x_1 + 6x_2 + 12x_3 + 24x_4 \leq 210 \\
 & x_1, \dots, x_4 \geq 0.
 \end{aligned}$$

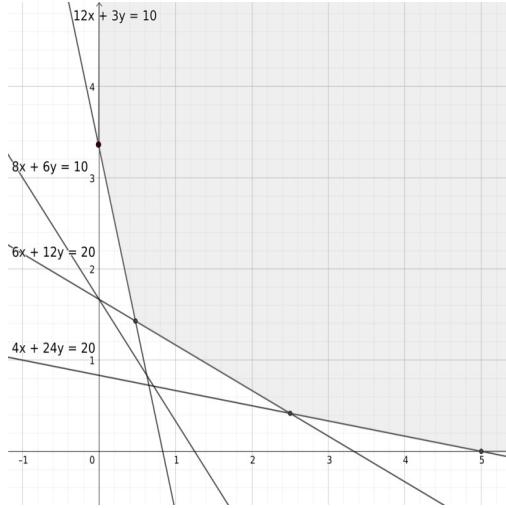
Ovaj problem ima 4 promjenljive i 2 ograničenja. Prema tome, ima smisla razmatrati dualni problem (D) datog primala (P), jer u tom slučaju dobijamo linearni program od samo dvije promjenljive koji se može lako riješiti grafičkom metodom. Odgovarajući problem (D) je dat sa:

$$\begin{aligned}
\min \quad & 210y_1 + 210y_2 \\
s.t. \quad & \\
& 12y_1 + 3y_2 \geq 10 \\
& 8y_1 + 6y_2 \geq 10 \\
& 6y_1 + 12y_2 \geq 20 \\
& 4y_1 + 24y_2 \geq 20 \\
& y_1, y_2 \geq 0.
\end{aligned}$$

Dopustiv region dualnog problema je dat na Slici 5.2. Dalje, može se pokazati da je optimum duala u presjeku prave $12y_1 + 3y_2 = 10$ i prave $6y_1 + 12y_2 = 20$. Na osnovu Slike 5.2 vidimo da prava koja odgovara drugom i četvrtom ograničenju LP-a mimoilazi tačku optima, jer vrijeđi stroga nejednakost nakon uvrštavanja tačke optima u ograničenja. Dakle, iz uslova komplementarnosti slijedi da za optimum x primala (P) vrijedi $x_2 = 0$ i $x_4 = 0$. Kako je $y_1, y_2 > 0$, onda vrijedi

$$\begin{aligned}
10x_1 + 20x_3 &= 210 \\
3x_1 + 12x_3 &= 210
\end{aligned}$$

odakle dobijamo rješenje $x_1 = 10, x_3 = 15$, pa je optimalno rješenje $x^* = (10, 0, 15, 0)$.



Slika 5.2: Dopustiv region duala.

Dualnost se najlakše opisuje kao odnos između problema maksimizacije i problema minimizacije, bez obzira koji od ta dva problema je primal, (P) a koji dual (D). Kompletna lista mogućih odnosa između ograničenja jednog problema i promjenljivih u drugom problemu je data sljedećom tabelom.

(P)	(D)
ograničenja \leq	promjenljive ≥ 0
ograničenja $=$	promjenljive (neograničene)
ograničenja \geq	promjenljive ≤ 0
promjenljive ≥ 0	ograničenja \geq
promjenljive (neograničene)	ograničenja $=$
promjenljive ≤ 0	ograničenja \leq

Tabela 5.1: Odnos promjenljivih i ograničenja problema (P) i odgovarajućeg duala (D).

Napomena. Simpleks metod ne garantuje rješavanje proizvoljnog problema LP u polinomijalnom broju koraka u odnosu na veličinu ulaza (pogledati

zadatak 6, u prethodnoj glavi). Međutim, rješavanje problema LP-a je moguće u polinomijalnom vremenu, uz pomoć metoda *unutrašnje tačke* (eng. interior point method), konstruisan od strane N. Karmakar-a sredinom 80ih godina prošlog vijeka. Radi kompleksnosti samog algoritma i njegovog izvođenja, mi ga ne prikazujemo u glavnem dijelu knjige, već u Apendiksu A gdje čitalac kojeg interesuje ovaj metod može da vidi i prouči ideju i glavne korake konstrukcije ovog značajnog algoritma.

5.2 Farkašova lema

Još jedna primjena teorije dualnosti LP-a se ogleda u dokazivanju Farkašove teoreme alternative. U suštini, njena geometrijska interpretacija kaže sljedeće: svaki vektor pripada ili datom zatvorenom konveksnom konusu (specijalan slučaj je ograničeni konveksni skup), ili postoji hiperravan koja razdvaja vektor od konusa; druge mogućnosti ne postoje.

Teorema 5.4. *Neka je $A \in \mathbb{R}^{m \times n}$ matrica i $c \in \mathbb{R}^n$. Onda jedan i samo jedan od sljedeća dva sistema ima rješenje:*

$$1. \quad Ax \geq 0 \text{ i } c^T x < 0;$$

$$2. \quad A^T y = c \text{ i } y \geq 0.$$

Dokaz. Pogledajmo prvi sistem. Uočimo sličnosti sa LP-om:

$$\begin{aligned} & \min c^T x \\ & \text{t.d.} \\ & Ax \geq 0. \end{aligned}$$

Ako pogledamo desnu stranu ograničenja, vidimo da je vektor $x = 0$ dopustivo rješenje problema, tako da problem ima barem jedno dopustivo rješenje. Zaključujemo da dati LP ili ima optimalno rješenje ili je neograničen.

Posmatrajmo njegov dual dat sa

$$\begin{aligned} & \max 0 \\ & \text{t.d.} \\ & A^T y = c \\ & y \geq 0. \end{aligned}$$

Primijetimo da, ako postoji barem jedno dopustivo rješenje za dati dual, optimalno rješenje je 0. Ako ne postoji niti jedno dopustivo rješenje, problem je nedopustiv.

Iz odnosa definisanog primala i njegovog duala zaključujemo:

- primal ima optimalno rješenje akko dual ima optimalno rješenje;
- primal je neograničen akko je njegov dual nedopustiv.

Odgovorimo sada na pitanje kada primal ima optimalno rješenje. Ono što vidimo je da ako bi x bio dopustiv tako da je $c^T x < 0$, to bi onda i bilo koje kx , za proizvoljno $k > 0$ bilo takođe dopustivo. Iz toga bi slijedilo da vrijednost funkcije cilja nije ograničena odozdo, tj. teži ka $-\infty$. Dakle, u tom slučaju je problem neograničen. Zaključujemo da, kad god primal ima optimalno rješenje, ta vrijednost mora biti 0 i dostiže se za $x = 0$. Inače, problem je neograničen. Dalje, ako primal ima optimalno rješenje, to je ekvivalentno sa tim da je sistem iz slučaja 1 ove teoreme nedopustiv (nema rješenja). Ako sistem iz slučaja 1 ima rješenje, primal je neograničen (vrijedi i obrnut smjer). Ako pogledamo dualni problem, dual ima optimalno rješenje akko sistem iz slučaja 2 ima rješenje. Slično, dual nije dopustiv akko je sistem iz slučaja 2 nedopustiv (nema rješenje). Iz niza prethodnih ekvivalencija, zaključujemo da vrijedi:

- Sistem iz slučaja 1 nema rješenje akko sistem iz slučaja 2 ima rješenje.
- Sistem iz slučaja 1 ima rješenje akko sistem iz slučaja 2 nema rješenje.

□

Primjer 5.5. Ovim primjerom analizirajmo geometrijsku interpretaciju Farkašove leme. Neka je dat sistem:

$$\begin{aligned}x_1 + 4x_2 &< 0 : c^T x \\2x_1 + 3x_2 &\geq 0 \\3x_1 + 2x_2 &\geq 0 \\4x_1 + x_2 &\geq 0 \\-x_1 + 3x_2 &\geq 0 : Ax \geq 0.\end{aligned}$$

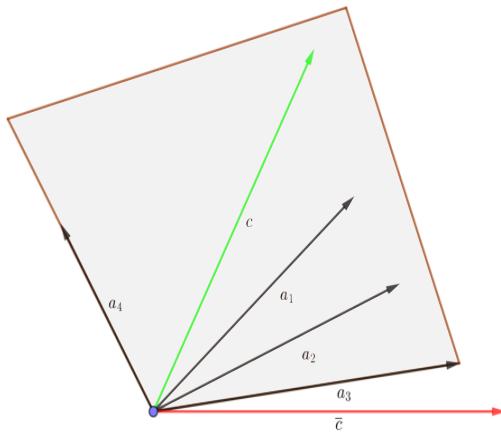
Pogledajmo kako izgleda sistem iz slučaja 2. Taj sistem je dat sa:

$$\begin{aligned}2y_1 + 3y_2 + 4y_3 - y_4 &= 1 \\3y_1 + 2y_2 + y_3 + 3y_4 &= 4 \\y_1, y_2, y_3, y_4 &\geq 0.\end{aligned}$$

Iz geometrijske interpretacije slijedi da se vektor $c = (1, 4)^T$ može predstaviti kao linearna kombinacija vektora vrsta matrice A , kao što vidiemo na Slici 5.3. Vektor $c = (1, 4)^T$ se nalazi unutar linearogn pokrivača $S^A = \text{span}\{a_1, \dots, a_n\} = \{\sum_{i=1}^n \lambda_i a_i \mid \lambda_i \in \mathbb{R}\}$, što implicira da početni sistem nema rješenje. Kada bismo posmatrali $c = (5, 0)^T$, zaključili bi da se on ne može izraziti kao linearna kombinacija vektora vrste matrice A , jer je van prostora S^A . Prema tome, rješenje početnog problema za ovu konfiguraciju bi postojalo.

5.3 Furije–Mockinov metod eliminacije

Jedno od važnih pitanja vezano za problem LP-a je sljedeće. *Neka su date matrica $A \in \mathbb{R}^{m \times n}$ i vektor $b \in \mathbb{R}^m$, da li je skup $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ neprazan?* Dakle, da li postoji barem jedno dopustivo rješenje linearogn programa?



Slika 5.3: Geometrijska interpretacija Farkašove leme.

Problem dopustivosti može da služi i da odgovorimo na sljedeće pitanje: *Da li je optimalna vrijednost funkcije cilja problema LP-a veća od neke unaprijed zadate vrijednosti k ?* Na ovo pitanje možemo odgovoriti pomoću *Furiye-Mockinovog* metoda eliminacije. Pored osnovne primjene, ovaj algoritam može poslužiti za dokazivanje Farkašove leme. Jedna od praktičnih primjena ove metode je u teoriji informacija, konkretno u informaciono-teorijskim dokazima ostvarivosti koji se opisuju uslovima pod kojima je zagarantovano postojanje šeme kodiranja sa dobrim performansama. Ovi uslovi se često opisuju linearnim sistemom nejednakosti, koji su upravo ulazne vrijednosti *Furiye-Mockinove* eliminacije.

Metoda *Furiye-Mockinove* eliminacije strukturalno drugačija od simpleks metode. Ideja algoritma je zasnovana na redukciji problema sa n promjenljivih na ekvivalentan problem koji sadrži $n - 1$ promjenljivih. Ekviva-

lentnost se posmatra u odnosu na svojstvo da početni problem ima rješenje akko ga ima i redukovani problem. Eliminacija promjenljivih se vrši iterativno, odnosno, u svakom koraku se redukuje broj promjenljivih za jedan. Na kraju se početni sistem svodi na sistem sa jednom promjenljivom, za koji se lako može provjeriti da li sadrži dopustivo rješenje.

Ponovimo, glavni korak ovog metoda je redukcija problema na ekvivalentan problem sa manjim brojem promjenljivih. Pretpostavimo da su ograničenja data sa promjenljivima x_1, \dots, x_n i da želimo da eliminišemo promjenljivu x_n . Za svako od ograničenja

$$a_{i,1}x_1 + \cdots + a_{i,n}x_n \leq b_i \quad (5.6)$$

vrijedi jedna od sljedećih alternativa

- $x_n \geq \frac{b_i - a_{i,1}x_1 - \cdots - a_{i,n-1}x_{n-1}}{a_{i,n}} = L_i$ ili
- $x_n \leq \frac{b_i - a_{i,1}x_1 - \cdots - a_{i,n-1}x_{n-1}}{a_{i,n}} = U_i$

$i = 1, \dots, m$, zavisno od toga da li je $a_{i,n} > 0$ ili $a_{i,n} < 0$. U slučaju da je $a_{i,n} = 0$, ostavljamo ograničenje (5.6) nepromijenjeno, jer x_n ne učestvuje u datom ograničenju. Na ovaj način ćemo dobiti najviše m gornjih i donjih granica za x_n koje se mogu izraziti preko $n - 1$ promjenljivih, tj.

$$x_n \geq L_1, \dots, x_n \geq L_k \text{ te } x_n \leq U_1, \dots, x_n \leq U_l,$$

za neke $k, l \in \mathbb{N}$, $k + l \leq m$. Moguće je izabrati x_n koje zadovoljava gornja ograničenja akko

$$\max\{L_1, \dots, L_k\} \leq x_n \leq \min\{U_1, \dots, U_l\}, \quad (5.7)$$

jer bi tada postojalo nešto između najveće donje granice i najmanje gornje granice za vrijednost promjenljive x_n . Međutim, kako su sve donje i gornje granice izražene preko nepoznatih promjenljivih x_1, \dots, x_{n-1} , ne može se

ustanoviti koje od njih su veće, a koje manje. Kompromis postižemo dodavanjem sljedećih nejednakosti

$$\begin{array}{llll} L_1 \leq U_1 & L_1 \leq U_2 & \cdots & L_1 \leq U_l \\ L_2 \leq U_1 & L_2 \leq U_2 & \cdots & L_2 \leq U_l \\ \vdots & \vdots & \ddots & \vdots \\ L_k \leq U_1 & L_k \leq U_2 & \cdots & L_k \leq U_l \end{array}$$

Ako postoji neka vrijednost x_n koja zadovoljava sva ograničenja (5.7), onda je za sve i, j , $L_i \leq x_n \leq U_j$. Obrnuto, ako svih $k \cdot l$ nejednakosti vrijede, onda vrijedi i (5.7), pa se može izabratи неки x_n između max i min gornje granice.

Prema tome, početni problem je redukovani na sistem od $n - 1$ promjenljivih sa dodatnih $k \cdot l$ nejednakosti izraženih gore, uz originalna ograničenja koja ne uključuju x_n . Ovaj proces nastavimo iterativno, dok ne dobijemo samo jednu promjenjivu sa dodatnim ograničenjima i potom (trivijalno) riješimo sistem.

Primjer 5.6. Pretpostavimo da nam je dat sistem nejednačina:

$$\begin{aligned} x - y &\leq 1 \\ -x + 2y &\geq 1 \\ 3x - 5y &\geq 1 \\ x, y &\geq 0. \end{aligned}$$

Eliminišimo promjenljivu y iz sistema. Kako je $x - y \geq 1 \Rightarrow y \leq x - 1$, $-x + 2y \geq 1 \Rightarrow y \geq \frac{x+1}{2}$, te $3x - 5y \geq 1 \Rightarrow y \leq \frac{3x-1}{5}$, uparivanjem gornjih i donjih granica dobijamo

$$\left\{ \begin{array}{l} \frac{x+1}{2} \leq x - 1 \\ 0 \leq x - 1 \\ \frac{x+1}{2} \leq \frac{3x-1}{5} \\ 0 \leq \frac{3x-1}{5} \\ x \geq 0, y \geq 0. \end{array} \right.$$

Ove nejednakosti se mogu pojednostaviti u donje i gornje granice za x , na sljedeći način:

$$\left\{ \begin{array}{l} x \geq 3 \\ x \geq 1 \\ x \geq 7 \\ x \geq \frac{1}{3} \\ x \geq 0. \end{array} \right.$$

Sve ove nejednakosti su donje granice. Vidimo da vrijednost $x = 7$ zadovoljava sve navedene nejednakosti. Tada, donje i gornje granice za y postaju uslovi: $y \leq 6$, $y \geq 4$, $y \leq 4$ te $y \geq 0$. Prema tome, možemo uzeti vrijednost $y = 4$, čime smo pronašli i dopustivo rješenje polaznog sistema.

5.4 Zadaci

1. Napišite odgovarajući dual linearog programa:

$$\begin{aligned} & \max x_1 - x_2 + 3x_3 \\ & \text{t.d.} \\ & x_1 + x_2 + x_3 \leq 10 \\ & 2x_1 - x_2 - x_3 \leq 2 \\ & 2x_1 - 2x_2 - 3x_3 \leq 6 \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

-
2. Napišite odgovarajući dual linearog programa:

$$\min 3x_1 - 2x_2 + 4x_3$$

t.d.

$$3x_1 + 5x_2 + 4x_3 \geq 7$$

$$6x_1 + x_2 + 3x_3 \geq 4$$

$$7x_1 - 2x_2 - x_3 \leq 10$$

$$x_1 - 2x_2 + 5x_3 \geq 3$$

$$4x_1 + 7x_2 - 2x_3 \geq 2$$

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

3. Napišite dual sljedećeg linearog programa:

$$\min x_1 - 3x_2 - 2x_3$$

t.d.

$$3x_1 - x_2 + 2x_3 \leq 7$$

$$2x_1 - 4x_2 \geq 12$$

$$-4x_1 + 3x_2 + 8x_3 = 10$$

$$x_1, x_2 \geq 0$$

$$x_3 \in \mathbb{R}.$$

4. Pokazati da je dual dualnog problema ponovo primalni problem (P).
5. Vratite se na Primjer 5.4. Vrijednost 210 u drugom ograničenju primalnog problema (P) zamijenite sa 420 i diskutujte optimalna rješenja (P) i (D) na sličan način kako je urađeno u samom primjeru.
6. Ako je primalni problem (P) neograničen (ima neograničenu funkciju cilja), njegov dual nije dopustiv. Dokazati.
7. Pokazati da vrijede odnosi dati u Tabeli 5.1.
8. Istražiti verziju simpleks metoda zvanu *Dualni simpleks metod*.

-
9. Koristeći Farkaš-ovu lemu, pokazati sljedeći rezultat.

Tačno jedan od sljedećih sistema ima rješenje:

- (a) $Ax > 0$;
- (b) $y^T A = 0, y \geq 0, y \neq 0$.

10. Koristeći Farkaš-ovu lemu, pokazati sljedeći rezultat. Tačno jedan od sljedećih sistema ima rješenje:

- (a) $Ax \leq b$;
- (b) $y^T A = 0, y^T b < 0, y \geq 0$.

11. Koristeći Furije-Mockin-ov metod eliminacije, pokazati da postoji rješenje za sljedeći sistem nejednačina:

$$\begin{aligned} x - 5y + 2z &\geq 7 \\ 3x - 2y - 6z &\geq -12 \\ -2x + 5y - 4z &\geq -10 \\ -3x + 6y - 3z &\geq -9 \\ -10y + z &\geq -15 \\ x, y, z &\geq 0. \end{aligned}$$

12. Pomoću Furije-Mockin-ovog metoda eliminacije, pokazati da je sistem

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \\ x_1 + x_2 &\leq -2 \end{aligned}$$

nedopustiv.

13. Neka su x^0 i y^0 dopustive tačke za polazni (kanonski) LP i njegov dual. Dokazati da su te dvije tačke optimalne ako i samo ako važi

$$\langle Ax^0 - b, y^0 \rangle = \langle A^T y^0 - c, x^0 \rangle = 0$$

14. Formirati dualan LP za problem

$$\min -x_1 + 2x_2 + 3x_3$$

t.d.

$$x_1 - x_2 + 2x_3 = 1$$

$$2x_1 + x_2 \leq 3$$

$$x \geq 0$$

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

pa riješiti oba problema.

15. Koristeći teoriju dualnosti, riješiti problem linearog programiranja

$$\min x_1 + 2x_2 + \cdots + nx_n$$

t.d.

$$x_1 \geq 1$$

$$x_1 + x_2 \geq 2$$

⋮

$$x_1 + x_2 + \cdots + x_n \geq n$$

$$x_1 \geq 0, x_2 \geq 0, \dots, x_n \geq 0.$$

(5.8)

Glava 6

Cjelobrojno linearno programiranje

U mnogim realnim problemima, promjenljive odlučivanja ne dobijaju neprekidne, već isključivo diskretne vrijednosti. Npr. kod problema dodjele poslova određenim mašinama, zahtijevaju se promjenljive koje su diskretnog tipa, kojima se definiše koji posao će biti dodijeljen kojoj mašini. U slučaju da su u nekom modelu sve promjenljive odlučivanja cjelobrojne, tada kažemo da je riječ o problemu *cjelobrojnog programiranja* (eng. *integer programming*). U ovoj glavi ćemo analizirati koncepte *cjelobrojnog linearног programiranja* (ILP), gdje se, pored integralnosti (cjelobrojnosti) promjenljivih, zahtijeva da funkcija cilja i sva ograničenja u modelu budu linearni izrazi po promjenljivima. Prema tome, ILP ima opšti oblik:

$$\begin{aligned}
 & \max c^T x \\
 & \text{t.d.} \\
 & Ax \leq b \\
 & x \geq 0 \\
 & x_i \in \mathbb{Z}, \forall i.
 \end{aligned} \tag{6.1}$$

Problemi cjelobrojnog linearног programiranja se veoma često sreću i u teoriji i u praksi, te je, samim tim, važno poznavati i tehnike rješavanje

ovih problema. Sa druge strane, pokazuje se da je rješavanje ovih problema u velikom broju slučajeva značajno teže nego rješavanje klasičnih problema linearнog programiranja. U narednom odjeljku ćemo modelovati nekoliko poznatih optimizationih problema pomoću koncepta cjelobrojnog linearнog programiranja. U Odjeljku 6.2 ćemo prikazati neke korisne tehnike transformacije nekih nelinearnih izraza u IP kako bi se dobio ekvivalentan ILP model. U Odjeljku 6.3 ćemo pokazati da je rješavanje ILP-a NP težak problem. Tehnike rješavanja ILP-a ćemo detaljnije razmatrati u Glavi 7.

6.1 Modelovanje nekih problema cjelobrojnog linearнog programiranja

Primjer 6.1. *Cjelobrojni problem jednodimenzionalnog ruksaka.* Neka je dato n proizvoda i neka je težina i -tog proizvoda jednaka w_i , dok je vrijednost i -tog proizvoda jednaka c_i , $i = 1, \dots, n$. Koje od proizvoda treba da stavimo u ruksak, čija je maksimalna težina (kapacitet) $C > 0$, tako da vrijednost proizvoda u ruksaku bude maksimalna, poštujući kapacitet ruksaka?

Rješenje. Definišimo promjenljive x_i , koje dobijaju vrijednost 1 ako i -ti proizvod uključujemo u ruksak, odnosno, vrijednost 0, ako i -ti proizvod nije uključen u ruksak, za $i = 1, \dots, n$. Ograničenje kapaciteta ruksaka se modeluje sa $\sum_{i=1}^n w_i x_i \leq C$. Funkcija cilja maksimizuje ukupnu vrijednost odabralih proizvoda i modeluje se uz pomoć linearne kombinacije $f(x) = \sum_{i=1}^n c_i x_i$.

Primjer 6.2. *Jedan problem rasporeda radnika u dežurstvu.* Uprava bolnice želi da napravi raspored dežurstva po sedmičnim noćnim smjenama za svoje medicinske sestre. Potrebno je da broj medicinskih sestara u noćnoj smjeni za j -ti dan bude barem (cjo broj) d_j , $j = 1, \dots, 7$. Svaka medicinska sestra radi pet dana zaredom u noćnoj smjeni.

Pronaći minimalan broj medicinskih sestara koje bolnica treba da zaposli, da bi se takve smjene mogle organizovati.

Rješenje. Označimo sa x_i broj medicinskih sestara koje svoju noćnu smjenu započinju na i -ti dan. Potrebno je minimizovati funkciju $f = \sum_{i=1}^7 x_i$

pod uslovima da su ispunjeni zahtjevi za brojem medicinskih sestara, što je modelovano sa:

- $x_1 + x_2 + x_3 + x_4 + x_5 \geq d_5$
- $x_2 + x_3 + x_4 + x_5 + x_6 \geq d_6$
- $x_3 + x_4 + x_5 + x_6 + x_7 \geq d_7$
- $x_4 + x_5 + x_6 + x_7 + x_1 \geq d_1$
- $x_5 + x_6 + x_7 + x_1 + x_2 \geq d_2$
- $x_6 + x_7 + x_1 + x_2 + x_3 \geq d_3$
- $x_7 + x_1 + x_2 + x_3 + x_4 \geq d_4$

uz nenegativne uslove

- $x_i \geq 0, x_i \in \mathbb{Z}, j = 1, \dots, 7.$

Primijetimo da su u ovom problemu promjenljive cjelobrojne. Primijetimo da sve promjenljive figurišu u tačno pet ograničenja, što odgovara uslovu da svaka od sestara radi pet dana zaredom u noćnoj smjeni.

Primjer 6.3. *Problem oglašavanja.* Kompanija želi da se oglašava u medijima. Postoji nekoliko načina oglašavanja: televizijsko, novinsko i radio oglašavanje. Cijena svakog medija sa pokrivenošću publike navedena je u Tabeli 6.1.

	Televizija	Novine	Radio
Cijena po oglašavanju	2000	600	300
Broj gledalaca/slušalaca/čitalaca	10000	4000	1800

Tabela 6.1: Cijene reklama sa brojem gledalaca/slušalaca/čitalaca.

Novine ograničavaju broj oglasa za kompanije na deset (po sedmici). Štaviše, kako bi se uravnotežilo oglašavanje između sve tri vrste medija,

na radiju se ne smije pojaviti više od polovine ukupnog broja oglasa za kompaniju. Najmanje 10% od svih oglasa bi se trebalo pojaviti na televiziji. Sedmični budžet za oglašavanje date kompanije iznosi 18200KM.

Koliko oglasa treba zakupiti za svaku od tri vrste medija sa ciljem povećanja ukupnog broja gledalaca?

Rješenje. Označimo sa x_1 broj oglasa na televiziji, sa x_2 broj oglasa u novinama i x_3 broj oglasa na radiju. Potrebno je maksimizovati uspješnost oglašavanja, tj. funkciju $f(x) = 10000x_1 + 4000x_2 + 18000x_3$. Uslovi pod kojima tražimo rješenja su modelovani na sljedeći način:

- $2000x_1 + 600x_2 + 300x_3 \leq 18200$ (uslov za cijenu);
- $x_2 \leq 10$ (limit na broj reklama u novinama);
- $x_1 + x_2 \geq x_3$ (uslov za maksimalno polovinu oglašavanja na radiju);
- $x_1 \geq 0.1 \cdot (x_1 + x_2 + x_3)$ (uslov za broj reklama na televiziji);
- $x_i \geq 0, x_i \in \mathbb{Z}, i = 1, 2, 3$.

Primjer 6.4. *Problem pokrivanja.* Telefonska kompanija želi da instalira antene na neka mjesta kako bi pokrila šest oblasti. Postoji pet mogućih mjesta za instaliranje antena. Nakon urađenih simulacija, za svaku oblast utvrđen je intenzitet signala koji antena šalje, kad je postavljena na određenom mjestu. Tabela 6.2 sadrži primjer jedne instance problema sa nivoima intenziteta signala.

	oblast 1	oblast 2	oblast 3	oblast 4	oblast 5	oblast 6
Mjesto A	10	20	16	25	0	10
Mjesto B	0	12	18	23	11	6
Mjesto C	21	8	5	6	23	19
Mjesto D	16	15	15	8	14	18
Mjesto E	21	13	13	17	18	22

Tabela 6.2: Intenziteti signala.

Prijemnici prepoznaju samo signale čija je jačina najmanje $d > 0$. Nadalje, ne može da se desi da više od jednog signala dostiže jačinu signala d u istoj oblasti, inače bi to prouzrokovalo smetnje u prijemu.

Potrebno je odrediti gdje treba postaviti antene kako bi se signalom pokrio maksimalan broj oblasti.

Rješenje. Posmatrajmo ovaj problem malo opštije i definišimo:

- I : skup mjesto;
- J : skup oblasti;
- σ_{ij} : indikator nivoa signala antene smještene u mjestu $i \in I$ u oblasti $j \in J$;
- d : parametar koji označava minimalnu jačinu signala;
- N : parametar koji označava maksimalni broj signala iznad granice koju prijemnik u jednoj oblasti može da prepozna (u našoj instanci imamo da je $N = 1$);
- x_i : binarna promjenljiva koja uzima vrijednost 1 ako je antena smještena u mjestu $i \in I$, 0 inače;
- z_j : binarna promjenljiva koja uzima vrijednost 1 ako je oblast $j \in J$ pokrivena signalom, 0 inače;
- M_j : dovoljno veliki parametar, npr. za svako mjesto $j \in J$, $M_j = |\{i \in I \mid \sigma_{ij} \geq d\}|$. Primjetimo da M_j označava gornje ograničenje za broj mjesto iz kojih se dovoljno jak signal može poslati predajniku u oblasti $j \in J$.

Formulišimo sada (cjelobrojni) linearni program ovog problema. Funkcija cilja je data sa:

$$f(x) = \min \sum_{j \in J} z_j$$

tako da

$$\sum_{\{i \in I | \sigma_{ij} \geq d\}} x_i \geq z_j, \forall j \in J \quad (6.2)$$

$$\sum_{\{i \in I | \sigma_{ij} \geq d\}} x_i \leq N + M_j(1 - z_j), \forall j \in J \quad (6.3)$$

$$x_i, z_j \in \{0, 1\}, \forall i \in I, \forall j \in J \quad (6.4)$$

Ograničenje (6.2) zahtijeva da, ukoliko je oblast pokrivena signalom, tj. ako je $z_j = 1$, tada mora postojati neko mjesto $i \in I$ koji ima antenu jačine barem d koja pokriva tu oblast. U slučaju da je $z_j = 0$, ovo ograničenje je redundantno.

Ograničenje (6.3) ograničava maksimalan broj signala koje prijemnici u oblasti $j \in J$ mogu obraditi. Preciznije, ako je oblast $j \in J$ pokrivena, tj. $z_j = 1$, onda predajnikom u oblasti $j \in J$ ne može da bude prihvачeno više od N (dovoljno jakih) signala. U slučaju da je $z_j = 0$, ovo ograničenje postaje redundantno, jer je ova oblast isključena iz rješenja, zbog čega i ukupan broj signala koji se šalje predajniku nije relevantan. Treba napomenuti da je ovo takozvano *big-M* ograničenje, koje ćemo analizirati u narednim sekcijama.

Ograničenjima (6.4) definisana je binarna priroda promjenljivih.

Primjer 6.5. *Jedan lokacijski problem.* Lokacijski problemi se često postavljaju na sljedeći način: pretpostavimo da postoji n postrojenja (prodavnica) i m kupaca. Dva pitanja koja treba da budu obuhvaćena rješenjem su:

1. koja od n postrojenja otvoriti i
2. koje postrojenje koristiti za opsluživanje pojedinih kupaca, kako bi se zadovoljila fiksna potražnja svakog od kupaca uz minimalni trošak otvaranja postrojenja.

Prije nego što modelujemo ovaj problem, uvedimo sljedeću notaciju.

- f_i : označava (fiksni) trošak otvaranja postrojenja i , za $i = 1, \dots, n$;

-
- c_{ij} : označava troškove zadovoljenja potražnje kupca j od strane postrojenja i (cijena dostave, cijena proizvoda, itd.), $i = 1, \dots, n$, $j = 1, \dots, m$;
 - d_j : označava ukupnu potražnju kupca j , $j = 1, \dots, m$;
 - u_i : označava maksimalnu količinu proizvoda (kapacitet zadovoljenja) koji se mogu nabaviti u postrojenju i , tj. u_i je kapacitet postrojenja i .

Definišimo sljedeće promjenljive odlučivanja:

- $x_i \in \{0, 1\}$, gdje je $x_i = 1$ ako je postrojenje i otvoreno, a inače 0.
- $y_{ij} \in \mathbb{R}^+$ označava dio ukupne potražnje d_j kupca j koja je ispunjena u prodavnici i .

Rješenje. Funkcija cilja problema je data sa:

$$\min f = \sum_{i=1}^n f_i x_i + \sum_{i,j} c_{ij} y_{ij} d_j, \quad (6.5)$$

pod uslovima

$$\sum_i y_{ij} = 1, \forall j \in \{1, \dots, m\} \quad (6.6)$$

$$\sum_j d_j y_{ij} \leq u_i x_i, \forall i \in \{1, \dots, n\}. \quad (6.7)$$

Funkcija cilja (6.5) minimizuje ukupnu cijenu zadovojenja (dijela) potreba kupaca u postrojenjima i cijene otvaranja postrojenja.

Ograničenje (6.6) garantuje da su potrebe svakog kupca zadovoljene. Ograničenje (6.7) ograničava da, ukoliko je postrojenje i otvoreno, ono može da ponudi kupcima onoliko koliko mu njegov kapacitet dozvoljava.

Ovaj model pripada modelu *mješovitnog linearнog programiranja*, gdje u istom modelu postoje i cjelobrojne i neprekidne promjenljive.

Sljedeći primjer spada u modele *binarnog linearнog programiranja*.

Primjer 6.6. *Problem postrojenja sa neograničenim kapacitetom.* Čest slučaj je varijacija prethodnog problema lokacija gdje je $u_i = +\infty, i = 1 \dots, m$. U ovom slučaju optimalna strategija se sastoji u zadovoljenju kompletne (neparcijalne) potražnje kupca j iz najbližeg otvorenog objekta. Zbog toga, neprekidne promjenljive y_{ij} možemo zamijeniti binarnim promjenljivima $z_{ij}, i = 1, \dots, n, j = 1, \dots, m$ pri čemu promjenljiva dobija vrijednost 1 ako je kupac j snabdjeven od strane postrojenja i . Prema tome, model izgleda ovako:

$$\begin{aligned} & \min \sum_{i=1}^n f_i x_i + \sum_{i,j} c_{ij} z_{ij} d_j \\ & \text{t.d.} \\ & \sum_i z_{ij} = 1, \forall j \in \{1, \dots, m\} \\ & \sum_j d_j z_{ij} \leq M x_i, \forall i \in \{1, \dots, n\} \\ & x_i, z_{ij} \in \{0, 1\} \text{ za } i = 1, \dots, n, j = 1, \dots, m. \end{aligned}$$

Vrijednost $M > 0$ je konstanta koja uzima neku veliku vrijednost (npr. zbir količine potražnji svih kupaca). Međutim, u praksi se često izbjegava upotreba M -konstante, te se tako ovo ograničenje mijenja sa $z_{ij} \leq x_i$ za $i = 1, \dots, n, j = 1, \dots, m$.

Primjer 6.7. *Problem rasporeda* (eng. scheduling problems). Čitava klasa problema koja se naziva sekvenciranje, raspoređivanje i usmjeravanje u osnovi su predstavljeni modelima cjelobrojnog programiranja. U ove probleme, na primjer, spadaju problemi rasporeda časova ili rasporeda ispita na fakultetima. Kod ovih problema postoje ograničenja u pogledu broja i kapaciteta učionica dostupnih u bilo kom trenutku, dostupnosti predavača na fakultetu u određeno vrijeme i preferencijama studenata ili nastavnika za određene rasporede. Kriterijumi za određivanje funkcije cilja mogu da

budu različiti. Kod rasporeda polaganja ispita, može se minimizovati broj potrebnih termina za održavanje ispita, uz osnovni uslov da dva ispita moraju biti u različitim terminima ako postoji barem jedan student koji planira da polaže oba ispita. Kod jednog problema rasporeda časova, raspoređivanje studenata u učionice se može vršiti na način da se minimizuje broj učenika koji ne mogu pohađati nastavu iz predmeta koji je njihov prvi izbor, u odnosu na ostale izbore. U tom slučaju, jasno je da imamo promjenljivu $X_{i,j,n}$ za i -tog učenika koji pohađa j -to predavanje tokom n -tog vremenskog perioda (recimo, vrijeme je diskretizovano po 45 min/čas) – koja je jednaka 1 ako se student uklapa u takav raspored, inače 0.

Posmatrajmo, ipak, malo specifičniji problem u odnosu na najopštiji, koji je dosta težak i za modelovanje, a kamoli za rješavanje. Razmotrimo raspoređivanje osoblja zaduženog za upravljanje letom aviona. Avionska kompanija treba da rasporedi svoje osoblje na rutama koje pokrivaju letove. Treba paziti, na primjer, da jedna posada treba da upravlja letom iz Beograda u Zagreb (u 10:00), a zatim letom iz Zagreba u Ciriš (u 14:00). Dakle, jedna posada bi trebalo da je uključena u rutu na kojoj upravlja različitim letovima. Prema tome, raspoređivanje posade na rutu j se modeluje uz pomoć binarne promjenljive

$$x_j = \begin{cases} 1, & \text{ako je barem jedna posada pridružena ruti } j \\ 0, & \text{inače.} \end{cases}$$

Dalje, definišimo

$$a_{ij} = \begin{cases} 1, & \text{ako je let } i \text{ pridružen ruti } j \\ 0, & \text{inače.} \end{cases}$$

Neka je c_j cijena zaduživanja neke posade ruti j . Ovdje a_{ij} definišu prihvatljive kombinacije letova sa rutama, uzimajući u obzir karakteristike kao što su redoslijed krakova za uspostavljanje veza između letova sa uključenim vremenom održavanja letova (iskrcavanje, istovar kofera, utovar novih kofera, ukrcavanje itd.) na istoj ruti leta posade.

Model ovakvog problema je dat sa

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

t.d.

$$\sum_{j=1}^n a_{ij} x_j \geq 1, \forall i \in \{1, \dots, n\} \quad (6.8)$$

$$x_i \in \{0, 1\}, \forall i \in \{1, \dots, n\} \quad (6.9)$$

$$a_{ij} \in \{0, 1\}, \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\}$$

Ograničenje (6.8) nam govori da, za svaki let i , mora postojati barem jedna ruta j za koju je barem jedna posada zadužena. Svakako, više letova može da bude pridruženo jednoj ruti, pa je otuda suma na lijevoj strani veća ili jednaka 1.

Primjer 6.8. *Problem trgovackog putnika* (eng. Traveling Salesman Problem – TSP). Neka je dato n gradova. Krenuvši od svog mjesta, putnik želi da posjeti ostalih $n - 1$ gradova pod minimalnim troškovima i da se na kraju vrati u početni grad. U suštini, ovo je problem grafovske prirode, gdje je ulaz (bez smanjenja opštosti) kompletan težinski graf, a potrebno je naći Hamiltonovu konturu minimalne težine u datom grafu. Pod težinom konture se podrazumijeva zbir težina njenih grana.

Rješenje. Označimo težinu grane od grada i do grada j sa c_{ij} . Definišimo (binarne) promjenljive

$$x_{ij} = \begin{cases} 1, & \text{ako putnik u svojoj ruti iz grada } i \text{ posjeti grad } j \\ 0, & \text{inače.} \end{cases}$$

Potrebno je nametnuti sljedeća ograničenja – svaki grad se treba naći u ruti, što možemo postići sa

$$\sum_{i=1}^n x_{ij} = 1, \forall j = 1, \dots, n,$$

što znači da se iz tačno jednog grada dolazi u svaki grad j , dok sličnim jednakostima nametnemo da se iz grada j odlazi u tačno jedan drugi grad sa:

$$\sum_{j=1}^n x_{ij} = 1, \forall i = 1, \dots, n$$

uz uslove $x_{ij} \geq 0$. Primijetimo sljedeće. Ovakvi uslovi ne sprečavaju pojedu rješenja koja predstavljaju uniju disjunktnih podruta, što nije dopustivo rješenje za TSP. Ako pretpostavimo da je rješenje dato u obliku podruta R_1, \dots, R_k , gdje je $R_i = \{v_{i_1}, \dots, v_{i_{p_i}}\}$, onda se pojava podruta može izbjegći dodavanjem dodatnih ograničenja u početni model problema. Tako problem suksesivno rješavamo dodavanjem ograničenja, sve dok ne dobijemo rješenje koje je u stvari jedna ruta koja obuhvata sve gradove. Recimo, eliminacija pojave podrute R_2 (u odnosu na podrutu R_1) se vrši dodavanjem ograničenja

$$\sum_{l=1}^{p_1} \sum_{t=1}^{p_2} x_{1_l, 2_t} \geq 1.$$

U najgorem slučaju, potrebno je dodati $2^n - 1$ ovakvih ograničenja prije nego se dobije dopustivo rješenje. Generalno ograničenje koje služi za eliminaciju podruta u modelu TSP-a se može dodati sa:

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 2, \forall S \subset [n]. \quad (6.10)$$

6.2 Tehnike transformacija nekih nelinearnih izraza u ILP-u

U prethodnom odjeljku je objašnjeno modelovanje nekih problema pomoću cjelobrojnog linearнog programiranja. Kao što se može vidjeti, modeli nekih problema pravolinijski slijede iz definicije samog problema, dok je za modelovanje nekih drugih, kompleksnijih problema, potrebno uključiti veći broj dodatnih promjenljivih i ograničenja. U ovom odjeljku ćemo prikazati

neke standardne transformacije i tehnike, kojima različite (nelinearne) izrave transformišemo i modelujemo ih pomoću cjelobrojnih promjenljivih i linearnih ograničenja.

6.2.1 Prepoznavanje binarnih promjenljivih

Pretpostavimo da je potrebno uključiti sljedeće aktivnosti: (1) izgraditi novo postrojenje ili (2) pokrenuti reklamnu kampanju ili (3) razviti novi proizvod. Jasno je da su u pitanju logičke odluke, odnosno odluke tipa „da ili ne”, pa ih, prema tome, modelujemo uvođenjem binarnih promjenljivih $x_i \in \{0, 1\}$, $i = 1, 2, 3$. Često se nameće uslov da se najviše jedna od ovih odluka izvrši, što se modeluje izrazom $\sum_{i=1}^3 x_i \leq 1$. Ovo ograničenje se obično naziva *ograničenjem sa višestrukim izborom* (eng. *multiple choice constraint*), jer se njime ograničava izbor odluka na najviše jednu od, u ovom slučaju, tri dostupne alternative. Ukoliko se zahtijeva da se tačno jedna od ovih odluka izvrši, onda se znak nejednakosti u posljednjem izrazu mijenja znakom jednakosti, odnosno, ograničenje postaje $\sum_{i=1}^3 x_i = 1$.

6.2.2 Logička ograničenja

Jedno od najjednostavnijih logičkih pitanja u matematičkom programiranju je da li dati izbor promjenljivih odlučivanja zadovoljava ograničenje

$$f(x_1, \dots, x_n) \leq b \quad (6.11)$$

ili ograničenje vrijedi u generalnom slučaju, tj. invariantno je (vrijedi u svakom slučaju).

Da bismo modelovali ograničenja koja odgovaraju prethodnoj alternativi, uvedimo binarnu promjenljivu y na sljedeći način:

$$y = \begin{cases} 1, & \text{ako je poznato da je ograničenje uvijek zadovoljeno,} \\ 0, & \text{inače} \end{cases}$$

te napišimo

$$f(x) - My \leq b, \quad (6.12)$$

gdje je M dovoljno velika konstanta, takva da je ograničenje zadovoljeno za sve x , ako je $y = 1$. Kad god je $y = 0$ i ako je uslov (6.11) zadovoljen, onda je i ograničenje (6.12) zadovoljeno. U praksi je vrijednost konstante M često određena karakteristikama samog problemom. Ideja ovog pristupa je modelovanje logičkog „ili“, kako će biti pokazano u narednoj sekciji.

6.2.3 Alternativna ograničenja

Razmotrimo sljedeću situaciju sa alternativnim ograničenjima:

$$\begin{aligned} f_1(x) &\leq b_1 \vee \\ f_2(x) &\leq b_2, \end{aligned} \tag{6.13}$$

odnosno, u modelu se traži je barem jedno od alternativnih ograničenja zadovoljeno. Ovakvo ograničenje se može modelovati kombinovanjem logičkih ograničenja i ograničenja sa višestrukim izborom. Dakle, imamo

$$\begin{aligned} f_1(x) - M_1 y_1 &\leq b_1 \\ f_2(x) - M_2 y_2 &\leq b_2 \\ y_1 + y_2 &\leq 1 \\ y_1, y_2 &\in \{0, 1\}. \end{aligned}$$

Konstante M_1 i M_2 se biraju tako da oba ograničenja iz (6.13) budu trivijalno zadovoljena, te time postanu invarijante kada je $y = 1$.

Ograničenje $y_1 + y_2 \leq 1$ implicira da barem jedna od y -promjenljivih mora da bude 0, odnosno, barem jedno alternativno ograničenje mora da bude zadovoljeno.

Pojednostavimo ovu formulaciju koristeći ograničenje $y_1 + y_2 = 1$ umjesto ograničenja nejednakosti, što povlači da je tačno jedna promjenljiva jednaka 1, a druga 0. Uvrštavanjem $y_2 = 1 - y_1$, dobijamo

$$\begin{aligned} f_1(x) - M_1 y_1 &\leq b_1 \\ f_2(x) - M_2(1 - y_1) &\leq b_2 \\ y_1 &\in \{0, 1\}. \end{aligned}$$

6.2.4 Uslovna ograničenja

Pretpostavimo da želimo da modelujemo sljedeću situaciju, sa ograničenjima

$$f(x) > b_1 \Rightarrow f(x) \leq b_2$$

Kako je implikacija $p \Rightarrow q$ ekvivalentna sa $\neg p \vee q$, prethodno ograničenje je ekvivalentno sa

$$f(x) \leq b_1 \vee f(x) \leq b_2$$

čime smo problem sveli na problem sa alternativnim ograničenjima.

6.2.5 K-alternative

Pokušajmo da modelujemo situaciju zadatu sa m ograničenja

$$f_j(x) \leq b_j, j = 1, \dots, m,$$

od kojih barem k ograničenja treba da bude zadovoljeno.

Modelovanje ovakvog scenarija vršimo na način da prvo odredimo konstante, čijim uključivanjem u izraz dobijamo ograničenja koja su uvijek ispunjena. Dodijelimo tim konstantama oznaće $M_1, \dots, M_m > 0$, redom. U tom slučaju, opšti problem može biti definisan na sljedeći način

$$f_j(x) + (1 - y_j)M_j \leq b_j, j = 1, \dots, m \quad (6.14)$$

$$\sum_{i=1}^m y_i \geq k \quad (6.15)$$

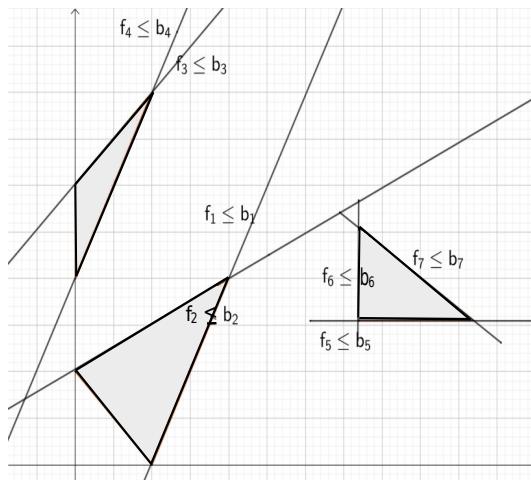
$$y_j \in \{0, 1\}, j = 1, \dots, m. \quad (6.16)$$

Primijetimo da ako je $y_j = 1$, tada je j -to ograničenje zadovoljeno (za dato x). Dakle, barem k ograničenja treba da budu zadovoljena da bi x bilo dopustivo rješenje.

6.2.6 Složene alternative

Pretpostavimo da se dopustiv region problema LP sastoji od tri disjunktna regionala kao na Slici 6.1. Ovakav slučaj se može modelovati uz pomoć tzv. konstante veliko $M > 0$ (eng. *big M*), te ograničenjem sa višestrukim izborom, odakle dobijamo:

$$\begin{array}{l} f_1(x) - M_1y_1 \leq b_1 \\ f_2(x) - M_2y_1 \leq b_2 \text{ (region 1)} \\ \hline f_3(x) - M_3y_2 \leq b_3 \\ f_4(x) - M_4y_2 \leq b_4 \text{ (region 2)} \\ \hline f_5(x) - M_5y_3 \leq b_5 \\ f_6(x) - M_6y_3 \leq b_6 \\ f_7(x) - M_7y_3 \leq b_7 \text{ (region 3)} \\ \hline y_1 + y_2 + y_3 \leq 2 \\ x = (x_1, x_2) \geq 0 \\ y_1, y_2, y_3 \in \{0, 1\}. \end{array}$$



Slika 6.1: Dopustiv region sa više disjunktnih regionala.

Uslov $y_1 + y_2 + y_3 \leq 2$ nam upravo ograničava da rješenje (x_1, x_2) pripada barem jednom od (disjunktnih) dopustivih regionala. Npr. ako je $y_3 = 0$, onda je svako od ograničenja

$$f_5(x) \leq b_5, \quad f_6(x) \leq b_6 \quad \text{i} \quad f_7(x) \leq b_7$$

ispunjeno.

6.2.7 Fiksne cijene u funkciji cilja

Kod nekih problema, funkcija cilja problema minimizacije može da sadrži i neke fiksne troškove, kao što su fiksni troškovi ulaganja ili fiksni troškovi postavljanja opreme. Npr. trošak proizvodnje x jedinica određenog proizvoda može da se sastoji od fiksnih troškova postavljanja opreme i promjenjivih troškova koji se tiču proizvedene robe na opremi (mašinama). Pretpostavimo da oprema ima kapacitet proizvodnje od B jedinica (nekog proizvoda). Neka je y binarna promjenljiva koja ukazuje na to kada nastaje fiksni trošak, pri čemu je $y = 1$ ako je $x > 0$, dok je $x = 0$ ako je $y = 0$. Tada se doprinos cijeni proizvodnje od x jedinica proizvoda može predstaviti sa

$$yB + cx$$

pod uslovima:

$$x \leq By \tag{6.17}$$

$$x \geq 0 \tag{6.18}$$

$$y \in \{0, 1\}. \tag{6.19}$$

Primijetimo da je ovaj model pripada *Mješovitom-cjelobrojnom programiranju*, gdje postoje obje vrste promjenljivih, neprekidne i cjelobrojne. O ovoj paradigmici će biti više riječi u narednim sekcijama.

6.2.8 Po-dijelovima-linearna funkcija

Po-dijelovima-linearna funkcija sama po sebi nije linearna. Međutim, kada joj se domen podijeli na segmente (intervale), ona postaje linearna

na svakom od tih segmenata. Napomenimo da se svaka neprekidna relana funkcija može aproksimirati skupom po-dijelovima-linearnih funkcija.

Radi jednostavnosti, posmatrajmo primjer zadat u ravni. Pretpostavimo da je funkcija definisana preko tri linearne funkcije na sljedeći način.

$$f(x) = \begin{cases} c_1x, & \text{za } x \in I_1 = [\delta_1^1, \delta_2^1] \\ c_2x, & \text{za } x \in I_2 = [\delta_2^1 = \delta_1^2, \delta_2^2] \\ c_3x, & \text{za } x \in I_3 = [\delta_2^2 = \delta_1^3, \delta_2^3]. \end{cases}$$

gdje su I_1, I_2 i I_3 zatvoreni intervali u \mathbb{R} , koji se nadovezuju jedan na drugi.

Da bi se modelovala ovakva funkciju cilja, promjenljivu x izrazimo preko tri nove promjenljive

$$x = \beta_1 + \beta_2 + \beta_3$$

pod uslovima

$$\begin{aligned} 0 \leq \beta_1 &\leq \delta_2^1 - \delta_1^1 = b_1 \\ 0 \leq \beta_2 &\leq \delta_2^2 - \delta_1^2 = b_2 \\ 0 \leq \beta_3 &\leq \delta_2^3 - \delta_1^3 = b_3, \end{aligned} \tag{6.20}$$

odakle slijedi da je funkcija cilja jednaka

$$C = c_1\beta_1 + c_2\beta_2 + c_3\beta_3.$$

Primjetimo da se pojavljuju problemi na graničnim uslovima intervala, da-kle u tačkama $I_1 \cap I_2 = \{\delta_2^1\}$ i $I_2 \cap I_3 = \{\delta_2^2\}$. Trebalo bi da vrijedi da $\beta_1 = \delta_2^1 - \delta_1^1$ kad god je $\beta_2 > 0$ kao i $\beta_2 = \delta_2^2 - \delta_1^2$ kad god je $\beta_3 > 0$. S obzirom da su ovo uslovna ograničenja, ona se modeluju uvođenjem dodatnih binarnih promjenljivih

$$\omega_1 = \begin{cases} 1, & \text{ako } \beta_1 \text{ dostiže svoju gornju granicu} \\ 0, & \text{inače,} \end{cases}$$

te

$$\omega_2 = \begin{cases} 1, & \text{ako } \beta_2 \text{ dostiže svoju gornju granicu} \\ 0, & \text{inače,} \end{cases}$$

pa se ograničenja (6.20) mogu napisati u obliku

$$\begin{aligned} b_1\omega_1 &\leq \beta_1 \leq b_1 \\ b_2\omega_2 &\leq \beta_2 \leq \omega_1 b_2 \\ 0 &\leq \beta_3\omega_2 \leq b_3 \\ \omega_1, \omega_2 &\geq 0. \end{aligned} \tag{6.21}$$

Primijetimo da, ako $\omega_1 = 0$, onda je $\omega_2 = 0$ kako bi se održala dopustivost za ograničenja propisana od strane β_2 , pri čemu se uslovi u (6.21) prevode u

$$0 \leq \beta_1 \leq b_1, \beta_2 = 0, \beta_3 = 0.$$

Ako je $\omega_1 = 1, \omega_2 = 0$, onda se uslovi u (6.21) transformišu u

$$\beta_1 = b_1, 0 \leq \beta_2 \leq b_2, \beta_3 = 0.$$

Ako je $\omega_1 = \omega_2 = 1$, onda se uslovi u (6.21) transformišu u

$$\beta_1 = b_1, \beta_2 = b_2, 0 \leq \beta_3 \leq b_3.$$

Prema tome, postoje tri dopustive kombinacije parametara ω_1 i ω_2 :

- $\omega_1 = 0, \omega_2 = 0$ za koje je $x \in I_1$ jer $\beta_2 = \beta_3 = 0$;
- $\omega_1 = 1, \omega_2 = 0$ za koje je $x \in I_2$, jer je $\beta_1 = b_1, \beta_3 = 0$;
- $\omega_1 = 1, \omega_2 = 1$ za koje je $x \in I_3$, jer je $\beta_1 = b_1, \beta_2 = b_2$.

6.2.9 Eliminacija proizvoda promjenljivih

Jasno je da proizvod dvije ili više promjenljivih narušavaju linearnost modela, te je korisno poznavati mehanizam kako se takvi proizvodi mogu izbjegći, odnosno transformisati u odgovarajuće linearizovane relacije. U ovom odjeljku ćemo prezentovati kako se proizvod dvije promjenljive može zamjeniti jednom promjenljivom sa dodatnim ograničenjima. Taj metod se, po potrebi, može proširiti i na proizvod više od dvije promjenljive. Razlikujemo nekoliko slučajeva:

-
- x_1 i x_2 su binarne promjenljive. Uvedemo smjenu $y = x_1x_2$, uz sljedeće uslove za promjenljivu y :

$$\begin{aligned}y &\leq x_1 \\y &\leq x_2 \\y &\geq x_1 + x_2 - 1 \\y &\in \{0, 1\}.\end{aligned}$$

Ova tehnika se može generalizovati na slučaj kada imamo proizvod n binarnih promjenljivih. Definišimo promjenljivu $y = x_1 \cdot \dots \cdot x_n$, $x_i \in \{0, 1\}, i = 1, \dots, n$. Uključivanjem sljedećih ograničenja

$$\begin{aligned}y &\leq x_1 \\&\vdots \\y &\leq x_n \\y &\geq \sum_{i=1}^n x_i - (n-1) \\y &\in \{0, 1\}.\end{aligned}$$

postiže se tražena transformacija.

- x_1 je binarna, x_2 je realna (pozitivna) promjenljiva, $0 \leq x_2 \leq u$. Opet uvodimo istu smjenu $y = x_1x_2$, gdje je y neprekidna promjenljiva. Dodamo sljedeća ograničenja, kako bismo osigurali da y dobije vrijednost proizvoda:

$$\begin{aligned}y &\leq ux_1 \\y &\leq x_2 \\y &\geq x_2 - u(1-x_1) \\y &\geq 0.\end{aligned}$$

- x_1, x_2 su obje neprekidne promjenljive, $l_1 \leq x_1 \leq u_1$, $l_2 \leq x_2 \leq u_2$. Uvedemo promjenljive $y_1 = \frac{1}{2}(x_1 + x_2)$ i $y_2 = \frac{1}{2}(x_1 - x_2)$. Tada se

izraz x_1x_2 može predstaviti u separabilnoj formi $y_1^2 - y_2^2$ koja se zatim može aproksimirati po-dijelovima-linearnim funkcijama. Granice za vrijednosti novih promjenljivih su

$$\frac{1}{2}(l_1 + l_2) \leq y_1 \leq \frac{1}{2}(u_1 + u_2)$$

i

$$\frac{1}{2}(l_1 - u_2) \leq y_2 \leq \frac{1}{2}(u_1 - l_2)$$

Aproksimacija nelinearnih funkcija. Kako smo pomenuli u prethodnoj sekciјi, ako je funkcija cilja realna, neprekidna i nelinearna, ona se može aproksimirati uz pomoć niza po-dijelovima-linearnih funkcija. Na tu aproksimaciju ćemo potom primjeniti transformaciju ograničenja koja je već prezentovana u prethodnoj sekciјi, a tiče se po-dijelovima-linearnim funkcijama.

U sljedećem primjeru primijenimo nekoliko navedenih transformacija ograničenja.

Primjer 6.9. Prepostavimo da je stanovništvo koncentrisano u I okruga unutar grada i da u okrugu $i \in [I]$ stanuje p_i ljudi. Preliminarna analiza (premjeri zemljjišta, političke i društvene prilike itd.) dala je potencijalne lokacije za izgradnju vatrogasnih domova na J lokacija. Neka $d_{ij} \geq 0$ predstavlja udaljenost središta okruga (dijela grada) i do lokacije $j \in J$.

Zadatak: *Potrebno je odabrati lokacije gdje će se podići vatrogasni domovi za svaki od okruga. O funkciji cilja i dodatnim ograničenjima ćemo diskutovati u toku modelovanja samog problema.*

Definišimo binarne promjenljive

$$y_j = \begin{cases} 1, & \text{ako je lokacija } j \text{ odabrana za izgradnju doma} \\ 0, & \text{inače} \end{cases}$$

i

$$x_{ij} = \begin{cases} 1, & \text{ako je lokacija za vatrogasni dom } j \text{ nadležna za okrug } i \\ 0, & \text{inače.} \end{cases}$$

Svaki okrug treba da bude pridružen tačno jednom vatrogasnog domu, pa se to ograničenje modeluje sa

$$\sum_{j \in J} x_{ij} = 1, \forall i \in I \quad (6.22)$$

Takođe, ako lokacija za vatrogasnog doma nije odabrana, onda nijedan okrug nije pridružen toj lokaciji. Odnosno, ako je $y_j = 0$, onda $\sum_{i \in I} x_{ij} = 0$. Ovo je uslovno ograničenje, koje se modeluje sa

$$\sum_{i \in I} x_{ij} \leq y_j |I|. \quad (6.23)$$

Udaljenost okruga i do nadležnog vatrogasnog doma je jednaka $d_i = \sum_{j \in J} d_{ij} x_{ij}$. Dalje, veličina populacije (broj ljudi) koja će biti pod nadzorom od strane doma na lokaciji j je jednaka

$$s_j = \sum_{i \in I} p_i x_{ij}. \quad (6.24)$$

Pretpostavimo da je jedan okrug posebno osjetljiv na požar i da lokacije 1 i 2 ili lokacije 3 i 4 treba da budu iskorištene da zaštite taj okrug. Ovakva situacija se modeluje sa ograničenjima

$$y_1 + y_2 \geq 2 \text{ ili } y_3 + y_4 \geq 2$$

što je ekvivalentno sa

$$\begin{aligned} y_1 + y_2 &\geq 2z \\ y_3 + y_4 &\geq 2(1 - z) \\ z &\in \{0, 1\}. \end{aligned} \quad (6.25)$$

Pretpostavimo da cijena izgradnje vatrogasnog doma na mjestu j koje može voditi brigu o s_j ljudi košta $f_j(s_j)$. Pretpostavimo takođe da je budžet kojim raspolažemo jednak B . Prema tome, vrijedi ograničenje

$$\sum_{j \in J} y_j f_j(s_j) \leq B. \quad (6.26)$$

Funkcija cilja koja bi se mogla optimizovati u ovoj varijanti problema je minimizacija udaljenosti okruga koja je najudaljenija od svog nadležnog vatrogasnog doma (označeno sa d_i)

$$\min D$$

gdje je $D = \max d_i$. Ova funkcija cilja se može napisati u ekvivalentnom obliku kao:

$$\begin{aligned} & \min D \\ & \text{t.d.} \\ & D \geq d_i, \forall i \in I \\ & (6.22) - (6.26) \\ & D \in \mathbb{R}^+. \end{aligned}$$

Kako bismo završili modelovanje, ostalo je još da se svaka funkcija $f_j(s_j)$ zamjeni aproksimacijom cjelobrojnog programiranje. Detalje ostavljamo čitaocu kao zadatak. Ako bi funkcija $f_j(s_j)$ sadržala fiksne troškove, tada se ne bi trebale uvoditi nove promjenljive fiksnih troškova, jer u tu svrhu već služi ranije uvedena promjenljiva y_j .

6.3 Kompleksnost rješavanja cjelobrojnog programiranja

Na početku ove glave smo pomenuli da je rješavanje problema cjelobrojnog programiranja NP-težak problem. Napomenimo da to nije slučaj sa problemom linearног programiranja, za koji smo u Glavi 5 pokazali da je rješiv u polinomijalnom vremenu. U nastavku ove sekcije dajemo dokaz da je rješivost ILP-a zaista NP-težak problem. Ideja dokaza je da problem 3-SAT polinomijalno svedemo na ILP. Podsjetimo se definicije 3-SAT problema.

Neka su dati literali x_1, \dots, x_n . Svaki literal može da bude pozitivan (x_1) ili negativan ($\neg x_1$). Kažemo da je formula napisana u *konjunktivnoj normalnoj formi* (CNF) akko je napisana kao konjunkcija klauzula ili literala. Pod klauzulom podrazumijevamo formulu koja je sastavljen od literala

povezanih disjunkcijama. Npr. formula $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2) \wedge (x_1 \vee x_3)$ je napisana u CNF. Ona se sastoji od dvije klauzule sastavljena od po dva literalna. CNF formula $\phi(x)$ je zadovoljiva akko postoji dodjela \bar{x} literalima ($\bar{x}_i \in \{\text{TRUE}, \text{FALSE}\}$) tako da je $\phi(\bar{x})$ tačna, prema zakonima Bulove algebre.

Kažemo da je konjunktivna formula $\phi(x)$ napisana u 3–CNF akko se u svakoj klauzuli nalazi najviše 3 različita literala. Napomenimo da se, u smislu rješivosti, proizvoljna klauzula može redukovati u 3–CNF. Dakle, svaka klauzula $l_1 \vee l_2 \vee \dots \vee l_n$ se može pretvoriti u 3–CNF na sljedeći način:

$$(l_1 \vee l_2 \vee x_2) \wedge (\neg x_2 \vee l_3 \vee x_3) \wedge (\neg x_3 \vee l_4 \vee x_4) \wedge \dots \wedge (\neg x_{n-3} \vee l_{n-3} \vee x_{n-2}) \wedge (\neg x_{n-2} \vee l_{n-1} \vee l_n),$$

gdje su x_2, \dots, x_{n-2} proizvoljne bulove promjenjive (literali).

Treba imati na umu da ove formule nisu logički ekvivalentne, ali su ekvivalentne u smislu zadovoljenja. Dakle, ako je jedna od formula zadovoljiva za neku dodjelu logičkih vrijednosti literalima, onda je to i druga, i obrnuto. Poznato je da je problem 3–SAT, gdje je ulaz 3–CNF formula, u opštem slučaju NP-kompletan problem.

Teorema 6.1. *Problem ILP-a je NP-težak.*

Dokaz. Izvedimo polinomnu redukciju instance 3–SAT problema na instancu problema ILP-a. Neka je dat 3–SAT izraz sa literalima x_1, \dots, x_n . Konstruјimo ILP instancu sa promjenljivima z_1, \dots, z_n . Svaka klauzula $\phi_i(x)$ se pretvara u nejednakost (ograničenje) C_i na sljedeći način:

- $z_i = x_i$, ako se literal x_i pojavljuje u klauzuli ϕ_i ;
- $z_i = 1 - x_i$ ako se $\neg x_i$ pojavljuje u klauzuli ϕ_i ;
- znak disjunkcije se mijenja znakom sabiranja „+“;
- vrijednost konstruisanog izraza treba da bude ≥ 1 .

Primjera radi, ako je data instanca problema 3-SAT sa $\phi(x) = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee \neg x_4 \vee x_5)$, transformišemo je (u polinomnom vremenu) na instancu ILP-a

$$\begin{aligned} & \min f \\ C_1 : & z_1 + (1 - z_2) + (1 - z_3) \geq 1 \\ C_2 : & z_2 + (1 - z_4) + z_5 \geq 1 \\ C_3 : & z_i \in \{0, 1\}, i = 1, \dots, 5. \end{aligned}$$

Dakle, iz gornje redukcije, provjera dopustivosti ILP instance odgovara provjeri zadovoljivosti 3-CNF izraza ϕ . Prema tome, problem rješivosti ILP-a je NP-težak, s obzirom da je 3-SAT problem NP-kompletan. \square

Može se pokazati i sljedeća teorema.

Teorema 6.2. *Neka je $k \in \mathbb{Z}$. Problem odlučivanja da li postoji cijelobrojno dopustivo rješenje ILP-a vrijednosti manje od k je NP-kompletan.*

6.4 Zadaci

1. Preduzeće za prikupljanje reciklažnog otpada raspolaže sa dvije deponije, koje su označene sa A i B. Dnevni troškovi održavanja deponije A su 300KM, a deponije B 45KM. U deponiji A se dnevno prikupi 250 flaša i 100 limenki, dok se u deponiji B prikupi 160 flaša i 350 limenki. Da bi se isplatio rad deponija, mjesечно treba prikupiti barem 5200 flaša i 4000 limenki. Koliko bi dana mjesечно trebale raditi obje deponije da budu isplative i da troškovi budu minimalni?
2. Fabrika proizvodi artikle A i B te ih prodaje po cijeni 30 KM, odnosno 50 KM, po komadu. Za proizvodnju fabrika koristi samo jednu mašinu, pri čemu je za proizvodnju artikla A potrebno 19 minuta, a proizvoda B 26 minuta. Tokom dana aparat može da radi najviše 8 sati. Odredi koliko treba proizvesti proizvoda A, a koliko B, da bi zarada bila maksimalna.

-
3. Aleksandar se spremi na planinarenje. Na raspolaganju ima samo jedan ruksak i pet namirnica, po jedan komad svake od njih. Ruksak je kapaciteta 2.5 l i ne smije biti teži od 2 kg . U tabeli su prikazani podaci o namirnicama.

Namirnica	Masa (g)	Zapremina (L)	Kalorijska vrijednost (kcal)
Čokolada	300	0,5	1500
Kokos	500	1,4	1300
Mlijeko	1000	1	660
Pasulj u konzervi	400	0,8	650
Keks	500	0,9	1800

Kako treba napuniti ruksak da ukupna kalorijska vrijednost ponesenih namirnica bude maksimalna?

4. Riješiti prethodni zadatak ako Aleksandar na raspolaganju ima dovoljno veliki broj svih pojedinačnih namirnica.
5. Preduzeće razmatra izvođenje 5 trogodišnjih projekata, ali u svakoj godini može uložiti najviše 25 000 KM. Procjena troškova, u hiljadama maraka, za svaki projekt u tri godine, kao i očekivana dobit od projekta (čista dobit – troškovi su oduzeti) dati su tabelom:

Projekat	1. godina	2. godina	3. godina	Dobit
1	5	1	8	20
2	4	7	10	40
3	3	9	2	20
4	7	4	1	15
5	8	6	10	30

Odrediti koje od projekata treba realizovati da se maksimizuje ukupna dobit.

6. Zadan je skup S koji se sastoji od n brojeva. Koristeći tehniku cjelobrojnog linearнog programiranja, podijeliti skup S na dva podskupa,

tako da se zbirovi elemenata u prvom i drugom podskupu što manje razlikuju.

7. Riješiti prethodni zadatak uz dodatni uslov da nijedan podskup ne smije da sadrži više od $\frac{2n}{3}$ elemenata.
8. U ovom zadatku je potrebno modelovati *problem totalnog pokrivanja*. Sa D_c označimo maksimalnu duljinu koja se može pokriti instalacijom c . Pod instalacijom podrazumijevamo skladište, antene i sl. Pod „pokrivanjem“ podrazumijevamo ispunjavanje zahtjeva klijenata, koji žive u nekom mjestu. Sa c_i označimo cijenu postavljanja instalacije na mjesto i . Sa d_{ij} označimo udaljenost između mjesta i i j . Zadatak ovog problema je da se odredi koja mjesta da se odaberu za instalacije, tako da su sva (ostala) mjesta pokrivena, ali da pri tome minimizujemo troškove instalacija. Sa jednog odabranog mjesta c se pokrivaju sva susjedna mjesta koja su u okviru radijusa D_c .
9. U ovom zadatku se modeluje *problem maksimalnog pokrivanja* (eng. *maximum coverage problem*). Neka je na ulazu data kolekcija skupova $S = \{S_1, \dots, S_m\}$ te broj $k \in \mathbb{N}$. Potrebno je naći podskup $S' \subseteq S$, tako da je $|S'| \leq k$, a broj pokrivenih elemenata $|\bigcup_{s' \in S'} s'|$ je maksimizovan.

Uputstvo. Sa $x_j \in \{0, 1\}$ označimo promjenljivu koja dobija vrijednost 1 ako je skup S_i odabran u rješenje S' , inače 0. Dodatno, sa $y_j \in \{0, 1\}$ definijemo promjenljivu koja dobija vrijednost 1 ako je element e_j pokriven rješenjem.
10. Potrebno je modelovati *P-center problem čvorova* (eng. *vertex p-center problem*). Formulacija problema je sljedeća. Ulagani podaci su predstavljeni neusmjerenim grafom $G = (I, J, E)$, gdje su objekti potražnje predstavljeni vrhovima $i \in I$, a moguće lokacije (servisnih) objekata su date drugim skupom vrhova $j \in J$, Grane $e_{i,j} \in E$ postoje samo između čvorova $i \in I$ i čvorova $j \in J$. Dalje, parovima čvorova su dodijeljene pozitivne težine $d_{i,j} \geq 0$, što predstavlja udaljenost između čvorova i i j . Treba imati na umu da je moguće imati i udaljenost 0

između čvora potražnje i moguće lokacije objekta. Za svaki čvor $i \in I$ pridružena je težina zahtjeva $h_i > 0$, koja predstavlja količinu zahtjeva koje taj čvor može da ispuni.

Zadatak je postaviti p objekata kako bi se smanjila maksimalna udaljenost između bilo kojeg čvora potražnje i njegovog servisnog objekta.

11. Pretpostavimo da se selimo u novi stan i da imamo samo dva kofera u koje možemo spakovati svoje stvari. U prvi kofer može da se spakuje 22 kg, a drugi 28 kg. Težina i vrijednosti stvari su dati sa tabelom

Stavka	A	B	C	D	E	F	G	H
Težina	10	9	15	3	11	6	3	4
Vrijednost	5	2	7	6	1	6	8	6

Konstruišite model koji maksimizuje vrijednost predmeta koji mogu biti ubaćeni u ova dva kofera.

12. Višedimenzionalni više-direkcioniji problem particonisanja brojeva (eng. *multidimensional multi-way number partitioning problem*) se definiše na sljedeći način. Neka je dat skup vektora S . Vektori mogu da budu iz proizvoljnog prostora \mathbb{R}^m , $m \in \mathbb{N}$. Zadatak je podijeliti skup S na $p \geq 1$ particija tako da su sume vrijednosti elemenata po koordinatama po particijama što ujednačenije.
13. U ovom zadatku se razmatra *Problem maksimalnog nezavisnog skupa* (eng. *independent set problem*). Dat je graf $G = (V, E)$. Potrebno je naći podskup $V' \subseteq V$ maksimalne kardinalnosti takav da V' ne sadrži ni jedan par susjednih čvorova.
14. Problem maksimalne klike (eng. *maximum clique problem*) se definiše na sljedeći način. Dat graf $G = (V, E)$. Potrebno je naći podskup $V' \subseteq V$ maksimalne kardinalnosti tako da je indukovani podgraf $G[V']$ grafa G kompletan graf.

-
15. Proizvođač igračaka planira proizvodnju novih igračaka. Troškovi pripreme proizvodnih pogona i jedinični profit za svaku igračku dati su sljedećom tabelom:

Igračka	Troškovi pripreme	Profit
1	45000	12
2	76000	16

Kompanija ima dvije fabrike koje mogu proizvoditi ove igračke. Kako bi se izbjeglo udvostručenje troškova pripreme za proizvodnju, samo jedna fabrika se može koristiti u datom momentu.

Stope proizvodnje svake igračke date su tabelom (u jedinicama/satu):

	Igračka 1	Igračka 2
Fabrika 1	52	38
Fabrika 2	42	23

Fabrika 1 i 2, imaju 480 i 720 sati radnog vremena za proizvodnju ovih igračaka, respektivno. Proizvođač želi znati koju će od novih igračaka proizvesti, u kojoj tvornici i koliko svake (ako uopšte) treba proizvesti kako bi se ukupna zarada maksimizovala.

16. Hrana se proizvodi rafiniranjem sirovih ulja i njihovim miješanjem. Postoje dvije kategorije sirovog ulja:

- vegansko ulje:
 - (a) VEG1
 - (b) VEG2
- obično ulje:
 - (a) Ulje1
 - (b) Ulje2
 - (c) Ulje3

Cijene za kupovinu svakog ulja date su u nastavku (u KM/toni)

VEG1	VEG2	Ulje1	Ulje2	Ulje3
115	128	132	109	114

Finalni proizvod prodaje se po 180 KM po toni. Biljna ulja i nebiljna ulja zahtijevaju različite proizvodne linije za rafiniranje. Nije moguće rafinisati više od 210 t biljnih ulja i više od 260 t nebiljnih ulja. Pretpostavimo da u procesu rafiniranja nema gubitka težine, a troškovi prerade mogu se zanemariti.

Postoji tehničko ograničenje koje se odnosi na tvrdoču konačnog proizvoda. U jedinicama u kojima se mjeri tvrdoča, dopušteno je da ona ima vrijednosti između 3.5 i 6.2. Pretpostavlja se da se tvrdoča mijenja linearno sa miješanjem vrsta ulja. Tvrdoča sirovog ulja je data sljedećom tabelom

VEG1	VEG2	Ulje1	Ulje2	Ulje3
8.8	6.2	1.9	4.3	5.1

Potrebno je odrediti koja ulja kupiti i kako ih miješati tako da kompanija maksimizuje svoj profit.

U osnovni model uključiti i sljedeća ograničenja:

- Hrana nikad ne smije biti pravljena miješanjem više od 3 vrste ulja.
- Ako je neko ulje uključeno u upotrebu, koristi se barem 30 t tog ulja.
- Ako su ulja VEG1 ili VEG2 korištena, onda i Ulje2 mora biti korišteno.

Glava 7

Algoritamske tehnike za rješavanje Cjelobrojnog linearног programiranja

U ovom poglavlju će biti analizirane različite tehnike kojima se rješavaju problemi cjelobrojnog linearног programiranja. Za razliku od problema LP-a, koji se efikasno rješava simpleks metodom, za rješavanje problema cjelobrojnog linearног programiranja ne postoji generalno efikasna metoda. Drugim riječima, efikasnost metoda, kojima se rješava ovaj problem, u velikoj mjeri zavisi od karakteristika samog problema.

Tehnike za rješavanje su uglavnom zasnovane na tzv. relaksaciji ILP-a u odgovarajući LP, koji se od polaznog problema (ili nekog od podproblema generisanog u toku rješavanja) dobija zanemarivanjem uslova o cjelobrojnosti promjenljivih. Iako se dobijena LP relaksacija može efikasno riješiti uz pomoć simpleks metoda, to ne znači da se iz rješenja relaksacije može lako doći do rješenja polaznog ILP-a. Npr. ako rješenje LP relaksacije označimo sa x^* , zaokruživanjem svake koordinate vektora x^* na njenu najbližu cjelobrojnu vrijednost, dobijamo cjelobrojni vektor \bar{x} . U ovom slučaju, uslov cjelobrojnosti jeste zadovoljen, ali ne postoji nikakva garancija da će, u opštem slučaju, ovakvo rješenje izvedeno iz LP relaksacije problema (6.1), biti

i dopustivo za početni ILP. Pored toga, skaliranjem desnih strana ograničenja ILP-a, kao i koeficijenata funkcije cilja na odgovarajući način, moguće je konstruisati problem, čije je optimalno cjelobrojno rješenje proizvoljno udaljeno od zaokruženog rješenja LP relaksacije.

Razmatranje opisano u prethodnom paragrafu ukazuje na to da se, uz relaksaciju početnog ILP-a u LP, moraju koristiti suptilnije tehnike za određivanje optimalnog rješenja ILP-a.

Metode za egzaktno rješavanje ILP-a se mogu klasifikovati na sljedeći način:

1. *Enumerativne tehnike* – dinamičko programiranje, metoda granjanja i ograničavanja (eng. *branch-and-bound* – B&B), metoda implicitne enumeracije i druge;
2. *Tehnike odsjecajućih ravnih*.

7.1 Metoda grananja i ograničavanja (B&B)

Metoda grananja i ograničavanja je zasnovana na poznatom principu „zavadi pa vladaj“ (eng. divide and conquer) gdje se problem, rekurzivno, rastavlja na manje dijelove. Kasnije se (optimalna rješenja) manjih problema kombinuju na odgovarajući način za dobijanje optimalnog rješenja početnog problema. U B&B metodi važan dodatak je upotreba strategije da se problem nikada ne rastavlja na manje podprobleme ako se može procijeniti da će rješenje podproblema biti suboptimalno (ograničavanje). Drugim riječima, algoritam „ne ulazi“ u one grane, za koje se unaprijed može pokazati da sadrže samo rješenja koja su lošija od trenutno najboljeg dopustivog rješenja. Očigledno, na taj način dolazi do uštete u broju operacija koje je potrebno izvršiti u toku izvođenja algoritma, a, samim tim, dolazi i do uštete memorije i vremena izvršenja. U rješavanju ILP-a, dopustiv skup se dijeli na manje skupove, dodavanjem specijalnih ograničenja u model. Rekursivni postupak se izvršava sve dok se ne dobiju podproblemi, čije relaksacije obezbjeđuju cjelobrojno rješenje ili se zaključi da podproblemi nisu dopustivi. Ovakvi pristupi se razlikuju u načinu podjele dopustivog

skupa i zbog toga u literaturi postoji nekoliko različitih B&B procedura.

7.1.1 Bazni B&B za rješavanje problema ILP-a

U rješavanju problema ILP-a uz pomoć B&B procedure, sljedeće jednostavne činjenice mogu biti korisne:

- U LP relaksaciji (problema maksimizacije), optimalna vrijednost relaksacije će biti gornja granica (eng. upper bound, skraćeno UB) optimuma posmatranog ILP-a na odgovarajućem podregionu.
- Bilo koja cjelobrojna tačka LP relaksacije na nekom podskupu dopustivog skupa uvijek je donja granica (eng. lower bound, skraćeno LB) optimalne vrijednosti početnog ILP-a.

Na osnovu ove dvije činjenice, uvedimo sada sistematičnu podjelu dopustivog regiona na niz podregiona:

1. U korjenom čvoru B&B stabla rješavamo LP relaksaciju početnog problema (npr. simpleks metodom) i dobijamo rješenje $x^* = (x_1^*, \dots, x_n^*)$. Ako je rješenje cjelobrojno, prekidamo dalje izvršavanje i stavimo $LB = f(x^*)$, koje je optimalno.
2. Za sve koordinate i za koje je $x_i^* \notin \mathbb{Z}$, dopustiv region se može podijeliti na dva (disjunktna) podskupa uvođenjem ograničenja: $z_i \geq \lfloor x_i^* \rfloor + 1$ ili $z_i \leq \lfloor x_i^* \rfloor$ i to je upravo mjesto podjele dopustivog regiona. Ako ima više takvih koordinata, biramo onu koordinatu kojoj je decimalni dio najveći. Recimo da je to koordinata k .
3. Sada dijelimo region problema na dva podregiona (lijevi i desni potomak čvor u B&B drvetu), dodavajući ograničenja $x_k \geq \lfloor x_k^* \rfloor + 1$ i $x_k \leq \lfloor x_k^* \rfloor$, redom.
4. Rješavamo relaksaciju oba (pod)problema i razmatramo sljedeće slučajeve.

-
- Ako je neko od rješenja cjelobrojno, popravljamo LB (ako smo dobili bolje dopustivo rješenje) i odgovarajući podproblem (nje-gov region) se dalje ne dijeli.
 - Ako rješenje x nije cjelobrojno i pri tome je vrijednost optimuma manja od trenutne LB , taj podproblem se dalje više ne dijeli (*bound* procedura).
 - Dalje, ako je podproblem nedopustiv, problem se ne dijeli (i ne razmatra više).
 - Inače, idemo na korak 2 dijeljeći podregion na nove (manje) podregione nekog od podproblema za koje to nije urađeno.

Primijetimo da ovom metodom formiramo drvo enumeracije koje nam garantuje nalazak optimalnog rješenja početnog ILP-a. Takođe, primijetimo da ovo drvo u generalnom slučaju raste eksponencijalno u odnosu na veličinu ulaznog problema.

Ovo je bila okvirna shema B&B procedure. Ono što je ostalo nerazjašnjeno je koji od neriješenih podproblema prvo rješavati. Kažemo da se čvorovi čiji je odgovarajući problem podijeljen na podregione naziva *neaktivan*, a inače *aktivan*. Generalni B&B metod za rješavanje problema ILP-a je dat Algoritmom 1. Neki od kriterijuma za izbor čvora (podproblema) koji se rješava su sljedeći:

- Izabratи one aktivne čvorove koji su najdublje u drvetu, da bi se što prije stiglo do listova B&B stabla i potencijalnih dopustivih rješenja.
- Birati aktivne čvorove nivo po nivo. Dakle, sve aktivne čvorove (tj. odgovarajuće podprobleme) jednog nivoa rješavati, pa onda preći na čvorove narednog nivoa.
- Koristiti heurističku funkciju pri odlučivanju koji aktivni čvor (tj. odgovarajući podproblem) prvo rješavati.

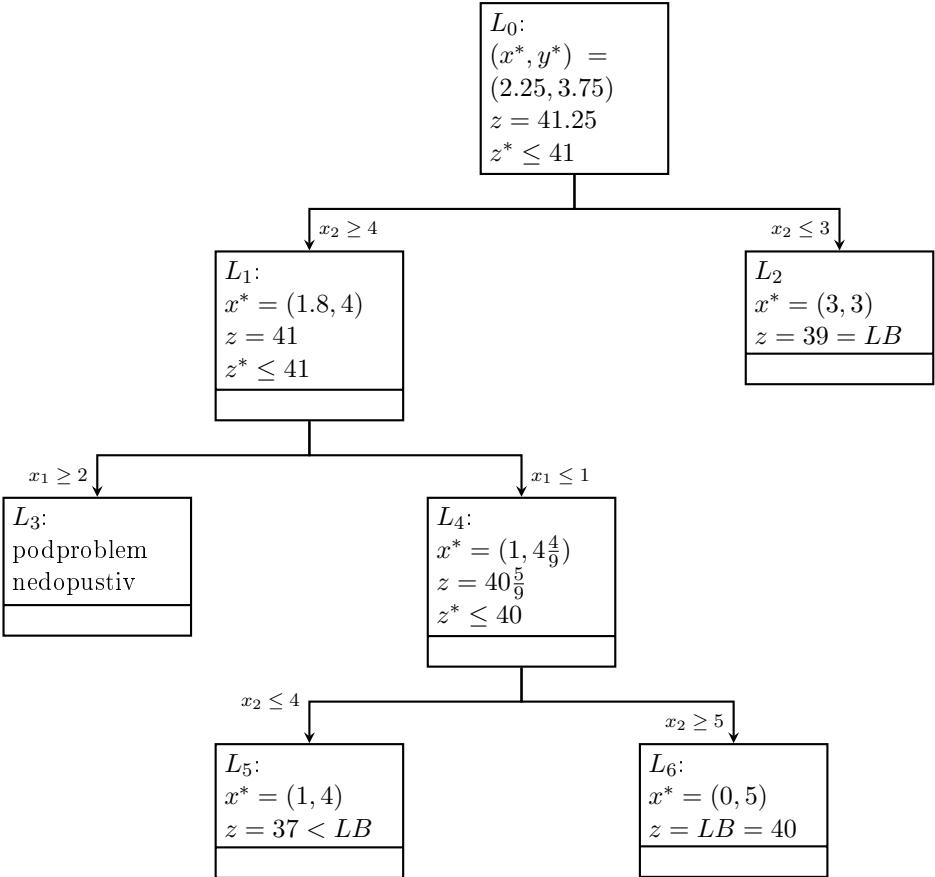
Primjer 7.1. Na Slici 7.1 je prikazano B&B drvo sljedećeg ILP-a:

Algoritam 1 Generalni B&B za rješavanje ILP-a.

```
1:  $LB \leftarrow -\infty$ 
2: Početni problem je aktivan (korjeni čvor)
3: while postoji neki aktivni čvor do
4:    $v_j \leftarrow$  Odabrati neki od aktivnih čvorova; // strategija
5:   Označiti  $v_j$  kao neaktivan;
6:    $x^j \leftarrow$  Riješiti relaksaciju podproblema koji odgovara čvoru  $v_j$ 
7:    $z_j^* \leftarrow f(x^j)$ ;
8:   if  $z_j^* \leq LB$  then
9:     Čvor  $j$  nije relevantan za dalje razmarađanje;
10:    end if
11:    if  $z_j^* > LB$  then
12:      if  $x^j$  je dopustivo then
13:         $LB \leftarrow z_j^*$ ;
14:         $x^* \leftarrow x^j$ ;
15:      else
16:         $v_j', v_j'', \dots, v_j^{(')k} \leftarrow$  Generisati podprobleme (čvorove) problema
           koji odgovara problemu čvora  $v_j$  (branch procedura);
17:        Označi aktivnim nove čvorove;
18:      end if
19:    end if
20:  end while
```

$$\begin{aligned} & \max 5x_1 + 8x_2 \\ & x_1 + x_2 \leq 6 \\ & 5x_1 + 9x_2 \leq 45 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

Napomene. Da bi se granice poboljšale, treba imati na umu sljedeće činjenice:



Slika 7.1: Primjer B&B metoda za rješavanje problema ILP-a.

- Ako su koeficijenti ciljne funkcije cijelobrojni, onda $z^* \leq \lfloor z^* \rfloor$;
- Dodavanjem važećih ograničenja u sam model, što će biti objašnjeno u Poglavlju 7.3.

7.2 Implicitna enumeracija

Posmatrajmo sada specijalnu B&B proceduru koja se može primijeniti na rješavanje problema binarnog cjelobrojnog programiranja, gdje diskretne promjenljive modeluju logičke odluke (da ili ne). Ovaj algoritam ne zahtjeva da se posmatraju rješenja LP-relaksacija, već se, umjesto toga, radi sa enumeracijom. U totalnoj enumeraciji bismo izlistali sve moguće kombinacije vrijednosti koje promjenljive mogu da dobiju, pa bi se izabrala ona koja je dopustiva i daje optimalnu vrijednost za ciljnu funkciju. Ovaj pristup se može koristiti na problemima malih dimenzija, gdje postoji svega nekoliko binarnih promjenljivih. Primjetimo da je u problemu sa n binarnih promjenljivih, ukupan broj kombinacija za vrijednosti promjenljivih jednak 2^n , odnosno raste eksponencijalno u odnosu na veličinu ulaza.

Prisjetimo se da je, u standardnoj B&B proceduri, podjela na manje probleme izvedena dodavanjem dodatnih ograničenja u početni model, a potom su izbačena ograničenja za cjelobrojnost. U implicitnoj enumeraciji se primjenjuje suprotna taktika, zadržavaju se 0-1 restrikcije za promjenljive, ali se u modelu ignoriraju linearna ograničenja. Ideja implicitne enumeracije je uključiti B&B proces tako što se fiksiraju neke od promjenljivih na 0 ili 1. Promjenljive koje su još neodređene se nazivaju *slobodne promjenljive*. B&B kreće od slučaja da ne postoje promjenljive koje nisu slobodne. U svakom koraku fiksiramo jednu od promjenljivih i rješavamo problem bez razmatranja ograničenja sa nejednakostima. Potom, provjeravamo da li je to rješenje dopustivo (zadovoljava ograničenja sa nejednakostima). U narednim podjelama prethodne (ne-slobodne) promjenljive ostaju i dalje fiksirane.

Pretpostavimo da je u pitanju problem maksimizacije. Proces implicitne enumeracije se sastoji od sljedećih koraka.

Fiksiramo $i = 1$ i $LB = -\infty$.

1. U početnom problemu biramo $x_j = 1$ za sve j (kažemo da je svaka promjenljiva *slobodna*) za koje je $c_j > 0$, a inače 0. Ako je dobijeno rješenje dopustivo, ažuriramo vrijednost za LB , algoritam se prekida.
2. Fiksiramo sada $x_i = 0$ za jedan podproblem, te $x_i = 1$, za drugi podproblem koji treba da se riješe.

-
3. Provjerimo za oba podproblema da li je rješenje (sa fiksnim dijelom) koje maksimizuje rješenje podproblema (kao u koraku 1) dopustivo. U slučaju da jeste, odgovarajući podproblem se više ne dijeli; po mogućnosti ažuriramo LB , ukoliko je nađeno novo najbolje rješenje. Ukoliko fiksiranje dijela rješenja vodi narušavanju nekog od ograničenja bez obzira na vrijednosti ostalih promjenljivih, podproblem je nedopustiv i više se ne dijeli.
4. U suprotnom, ažuriramo $i = i + 1$ i idemo na korak 3 sa podproblemima koji mogu dalje da se dijele, dok god je $i \leq n$. Inače, izlazimo iz petlje sa optimalnim rješenjem (LB).

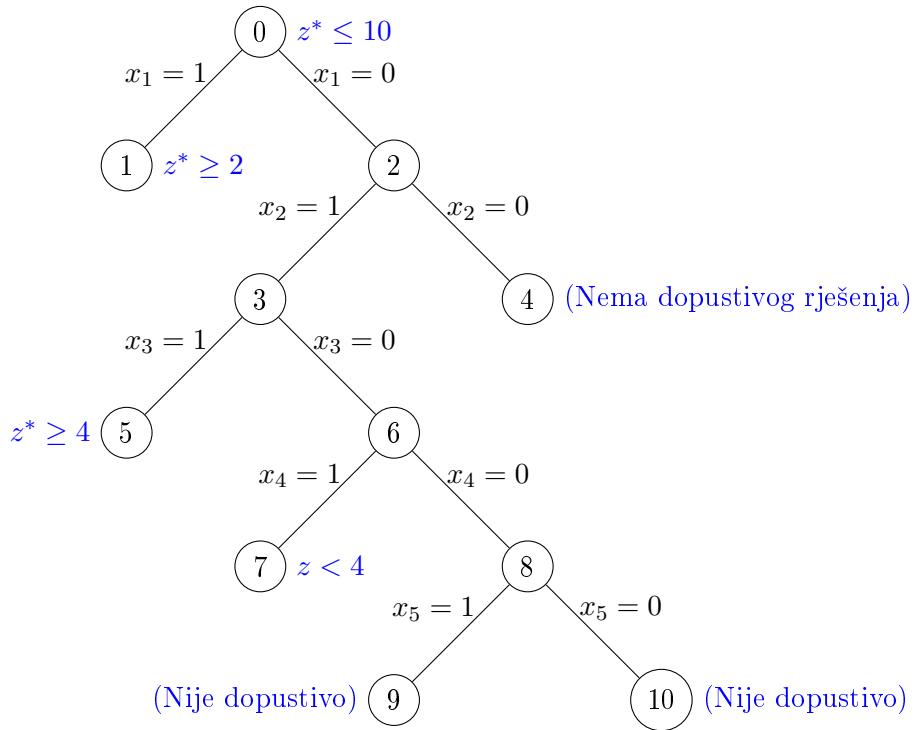
Primijetimo da i u ovom slučaju dobijamo drvo odlučivanja (binarno drvo) gdje su na granama koje spajaju čvor na dubini i sa čvorom na dubini $i + 1$ ugrađene odluke o tome da li je promjenljiva x_i fiksirana na 0 ili 1.

Primjer 7.2. Na Slici 7.2 je dato drvo odlučivanja za problem

$$\begin{aligned} & \max -8x_1 - 2x_2 - 4x_3 - 7x_4 - 5x_5 + 10 \\ & \text{t.d.} \\ & -3x_1 - 3x_2 + x_3 + 2x_4 + 3x_5 \leq -2 \\ & -5x_1 - 3x_2 - 2x_3 - x_4 + x_5 \leq -4 \\ & x_i \in \{0, 1\}, \text{ za } i = 1, \dots, 5. \end{aligned}$$

Prodiskutujmo ovo rješenje po čvorovima dobijenog stabla:

- U čvoru 0 imamo rješenje $x_i = 0$, za sve i , koje nije dopustivo, ali tada za optimalno rješenje vrijedi $z^* = f(x^*) \leq 10$.
- Dalje, granamo ovaj problem na dva (korak 2), jedan gdje je $x_i = 1$ a drugi $x_1 = 0$, odakle dobijamo dva podproblema (odgovaraju čvorovima 1 i 2). U čvoru 2 imamo podproblem za koji je $x_1 = 1$. Jasno je da kada uvrstimo $(1, 0, \dots, 0)$ u početni problem, da je ono dopustivo. Prema tome, optimalna vrijednost problema neće biti manji od 2,



Slika 7.2: Drvo odlučivanja za rješavanje problema binarnog programiranja.

odakle zaključujemo da je za ovaj podproblem $z^* = f(x^*) \geq 2$. Dakle, imamo $LB=2$.

- Problem u čvoru 2 je isti kao problem u čvoru 1, pa nastavljamo dalje sa dijeljenjem problema fiksirajući slobodne promjenljive.
- Sada fiksiramo promjenjivu x_2 (uz već fiksiranu odluku $x_1 = 0$). Za podproblem u čvoru 4, provjerimo da ne postoji dopuna do dopustivog rješenja za ovako fiksirane vrijednosti (narušeno ograničenje 1), čime se prekida dalje granjenje po ovoj grani. Za podproblem u čvoru 3, imamo sada podjelu na dva podproblema fiksiranjem promjenljive x_3 , tj. dobijamo dva nova čvora 5 i 6. Za problem u čvoru 5, lako se vidi da za ovakve fiksirane promjenljive imamo dopustivo rješenje za koje je vrijednost funkcije cilja jednaka 4, pa zaključujemo da je $z^* = f(x^*) \leq 4$, odakle imamo $LB = 4$. Problem u čvoru 6 dijelimo dalje (po promjenljivoj x_4), odakle dobijamo dva nova podproblema, koji odgovaraju čvorovima 7 i 8. Lako se zaključi da je $z^* < LB$ za podproblem u čvoru 7. Za čvor 8, imamo novo grananje (po promjenljivoj x_5). Kako su ovdje sve promjenljive fiksirane, primijetimo da niti jedna od ovih tačaka nije dopustiva.
- Prema tome, optimalno rješenje početnog problema je $LB = 4$ i dostiže se u tački $x = (0, 1, 1, 0, 0)$.

Totalna enumeracija bi generisala 32 čvora za gornji problem. Implicitna enumeracija generiše svega 11 čvorova.

7.3 Metoda odsjecajućih ravnih

Metoda odsjecajućih ravnih rješava problem cjelobrojnog programiranja mijenjanjem rješenja njegove relaksacije, sve dok se ne dobije model čije je optimalno rješenje cjelobrojno. Metoda nije bazirana na principu dijeljenja dopustivog regiona problema na podregione, kao što je to slučaj sa B&B metodama, već radi sa jednim linearnim programom, kojeg poboljšava iterativnim dodavanjem novih ograničenja. Nova ograničenja sucesiv-

no smanjuju dopustiv region, sve dok se ne pronađe cjelobrojno optimalno rješenje. Istoriski gledano, to je prvi algoritam razvijen za probleme cjelobrojnog programiranja, za koji je formalno dokazano da konvergira ka optimumu u konačnom broju koraka. Iako se algoritam generalno smatra neefikasnim, važan je sa metodološkog aspekta, jer je njegov razvoj kasnije doveo do konstrukcije drugih, efikasnijih algoritama.

Definicija 7.1. *Validna nejednakost za ILP je svako ograničenje koje ne eliminiše niti jedno dopustivo cjelobrojno rješenje problema.*

Drugi naziv za *validnu nejednakost* je *odsjecajuća ravan*.

Primjer 7.3. Za sljedeći problem ILP-a

$$\begin{aligned} \max z &= 3x_1 + 4x_2 \\ \text{t.d.} \\ 5x_1 + 8x_2 &\leq 24 \\ x_1, x_2 &\geq 0, x_1, x_2 \in \mathbb{Z}, \end{aligned}$$

jedna njegova validna nejednakost bi bila $2z - x_1 = (6x_1 + 8x_2) - x_1 = 5x_1 + 8x_2 \leq 24$, dakle $2z - x_1 \leq 24$, a kako sigurno vrijedi $x_1 \leq 4$, dobijamo $2z \leq 24 + 5$. Iz prethodnog imamo $z \leq 14.5$, što zbog cjelobrojnosti funkcije cilja daje $z \leq 14$.

Definicija 7.2. *Konveksni omotač za problem (6.1) je najmanji (dopustivi) poliedar nekog LP-a koji sadrži sve dopustive cjelobrojne tačke tog problema.*

Ako riješimo LP problem čiji je dopustivi skup konveksni omotač cjelobrojnih tačaka, onda je to rješenje optimalno za odgovarajući ILP. Međutim problem nalaska ovakvog konveksnog omotača je takođe NP-težak, pa prema tome nije očekivano da se takav omotač može naći u opštem slučaju. Nalazak korisnih ograničenja (ne svih) koja učestvuju u izgradnji konveksnog omotača je takođe veoma težak zadatak, mada je ovo ponekad i moguće. Prema tome, zadatak se svodi na nalazak korisnih odsjecajućih ravni, nezavisno od problema pronalaska konveksnog omotača, što se najčešće i radi u praksi.

Primjer 7.4. Posmatrajmo *problem pakovanja skupa* dat na sljedeći način. Neka je data kolekcija D rombova (u \mathbb{R}^2).

Zadatak je izabrati maksimalan broj rombova tako da se nikoja dva od njih ne presijecaju. Za rombove kažemo da se presijecaju ako imaju barem jednu zajedničku tačku u presjeku. Neka je O skup svih parova rombova koji se presijecaju. Sljedeći ILP odgovara ovom problemu:

$$\begin{aligned} \max & \sum_{d \in D} x_d \\ \text{t.d.} & \\ x_d + x_{d'} & \leq 1, \forall (d, d') \in O \\ x_d & \in \{0, 1\}, \forall d \in D \end{aligned}$$

gdje je x_d binarna promjenljiva, koja dobija vrijednost 1 ako je romb d izabran u riješenju, inače 0. Kao što vidimo, broj binarnih promjenljivih u praksi može biti veoma velik, što otežava efikasnu primjenu neke od B&B tehnika.

Pokušajmo dodati odsjecajuće ravni u prethodni ILP. Za svaku tačku c , enumеришмо sve one (moguće) rombove iz D koji sadrže ovu tačku. Označimo takav skup sa $D(c)$. S obzirom na uslove zadatka, za svaki skup $D(c)$, najviše jedan romb može da bude odabran. Sada, u prethodni model dodamo sljedeća ograničenja

$$\sum_{d \in D(c)} x_d \leq 1, \forall c. \quad (7.1)$$

Ovim postupkom smo formirali odsjecajuće ravni, koje će ubrzati rješavanje posmatranog problema za red veličine.

Razmotrimo sada *Gomorijeve odsjecajuće ravni*, kao generalnu metodu za dodavanje odsjecajućih ravni u rješavanju ILP-a. Ideja se sastoji u generisanju ravni iz nekog od ograničenja dobijenim optimalnom tabelom simpleks metode, koja predstavlja LP relaksaciju problema. Neka simpleks metod generiše sljedeći skup jednakosti u formi:

$$x_i + \sum_j \bar{a}_{i,j} s_j = \bar{b}_i, i = 1, \dots, m$$

gdje su x_i bazne promjenjive, s_j nebazne promjenjive, dok su vrijednosti \bar{a}_{ij} elementi gornje lijeve podmatrice optimalne simpleks tabele (4.1) koji odgovaraju baznim vektorima.

Napišimo prethodnu jednakost u drugačijem obliku tako da se na lijevoj strani nalaze cjelobrojni dijelovi, dok su na desnoj decimalni dijelovi, tj.

$$x_i + \sum_j \lfloor \bar{a}_{i,j} \rfloor s_j - \lfloor \bar{b}_i \rfloor = \bar{b}_i - \lfloor \bar{b}_i \rfloor - \sum_j (\bar{a}_{i,j} - \lfloor \bar{a}_{i,j} \rfloor) s_j.$$

Ako je x dopustivo cjelobrojno rješenje, onda je desna strana manja od 1, a kako je lijeva strana cjelobrojna u tom slučaju, dobijamo da je desna strana manja ili jednaka 0, pa imamo

$$\bar{b}_i - \lfloor \bar{b}_i \rfloor - \sum_j (\bar{a}_{i,j} - \lfloor \bar{a}_{i,j} \rfloor) s_j \leq 0 \quad (7.2)$$

za bilo koje (bazno) dopustivo rješenje x koje je cjelobrojno. Dalje, kako su vrijednosti nebaznih promjenjivih jednake 0 u bilo kom baznom dopustivom rješenju, zaključujemo

$$\bar{b}_i - \lfloor \bar{b}_i \rfloor - \sum_{ij} (\bar{a}_{i,j} - \lfloor \bar{a}_{i,j} \rfloor) s_j = \bar{b}_i - \lfloor \bar{b}_i \rfloor \geq 0.$$

Prema tome, pokazali smo da odsjecajuće ravni (7.2) ne eliminišu bazna dopustiva rješenja, pa ispunjavaju sve uslove za validnu nejednakost. Uvodeći dodatnu promjenjivu y_i za nejednakosti u (7.2), sljedeća (validna) ograničenja dodajemo u ILP:

$$\bar{b}_i - \lfloor \bar{b}_i \rfloor = \sum_j (\bar{a}_{i,j} - \lfloor \bar{a}_{i,j} \rfloor) s_j - y_i \quad (7.3)$$

Primjer 7.5. Neka je dat sljedeći problem ILP-a

$$\max 3x_1 + 4x_2$$

t.d.

$$\frac{2}{5}x_1 + x_2 \leq 3$$

$$\frac{2}{5}x_1 - \frac{2}{5}x_2 \leq 1$$

$$x_1, x_2 \geq 0,$$

$$x_1, x_2 \in \mathbb{Z}.$$

Riješimo ga uz pomoć Gomorijevih odsjecajućih ravnih.

Rješenje. Inicijalna simpleks tabela (LP relaksacije) je data sa

$\frac{2}{5}$	1	1	0	3
$\frac{2}{5}$	$-\frac{2}{5}$	0	1	1
-3	-4	0	0	0

Uključimo drugu promjenljivu (odg. kolona 2) u bazu, pa imamo

$\frac{2}{5}$	1	1	0	3
$\frac{14}{25}$	0	$\frac{2}{5}$	1	$\frac{11}{5}$
$-\frac{7}{5}$	0	4	0	12

Dalje, biramo novi pivot u prethodnoj simpleks tabeli, a to je element $\bar{a}_{2,1} = \frac{14}{25}$, pa transformacijama dobijamo simpleks tabelu

0	1	$\frac{10}{14}$	$-\frac{10}{14}$	$\frac{20}{14}$
1	0	$\frac{10}{14}$	$\frac{25}{14}$	$\frac{55}{14}$
0	0	5	$\frac{5}{2}$	$\frac{35}{2}$

Kako u posljednjoj vrsti nemamo negativnih vrijednosti, zaključujemo da je ovo optimalna simpleks tabela. Na osnovu (7.3), za prvi red simpleks tabele i odgovarajuće ograničenje generišemo odsjecajuću ravan:

$$\frac{5}{7}s_1 + \frac{2}{7}s_2 - y_1 = \frac{6}{14} = \frac{3}{7}$$

pa je dodajemo u (završnu) simpleks tabelu:

$$\begin{array}{ccccccc|c} 0 & 1 & \frac{10}{14} & -\frac{10}{14} & 0 & | & \frac{20}{14} \\ 1 & 0 & \frac{10}{14} & \frac{25}{14} & 0 & | & \frac{55}{14} \\ 0 & 0 & \frac{5}{7} & \frac{2}{7} & -1 & | & \frac{3}{7} \\ \hline 0 & 0 & 5 & \frac{5}{2} & 0 & | & \frac{35}{2} \end{array}$$

Formirajmo sada jediničnu podmatricu u gornjoj lijevoj matrici \bar{A} , imajući u vidu treću kolonu (i odgovarajuću promjenljivu s_1), a pivotirajući oko elementa $\bar{a}_{3,3}$. Dobijamo

$$\begin{array}{ccccccc|c} 0 & 1 & 0 & -1 & 1 & | & 1 \\ 1 & 0 & 0 & \frac{3}{2} & 1 & | & \frac{7}{2} \\ 0 & 0 & 1 & \frac{2}{5} & -\frac{7}{5} & | & \frac{3}{5} \\ \hline 0 & 0 & 0 & \frac{1}{2} & 7 & | & \frac{29}{2}. \end{array}$$

Kao što vidimo, rješenje problema je $(\frac{7}{2}, 1, 0, 0, \frac{3}{5})$, što i dalje nije cijelobrojno rješenje. Dalje, dodajemo novu odsjecajuću ravan, koja se izvodi iz drugog ograničenja (drugi red simpleks tabele).

$$\frac{1}{2}s_2 - y_2 = \frac{1}{2}.$$

Dodajmo ovo ograničenje u posljednju simpleks tabelu

$$\begin{array}{ccccccc|c} 0 & 1 & 0 & -1 & 1 & 0 & | & 1 \\ 1 & 0 & 0 & \frac{3}{2} & 1 & 0 & | & \frac{7}{2} \\ 0 & 0 & 1 & \frac{2}{5} & -\frac{7}{5} & 0 & | & \frac{3}{5} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & -1 & | & \frac{1}{2} \\ \hline 0 & 0 & 0 & \frac{1}{2} & 7 & 0 & | & \frac{29}{2} \end{array}$$

Sada za promjenljivu x_4 (četvrta kolona), pivotiramo oko elementa $\bar{a}_{4,4}$, pa dobijamo simpleks tabelu

0	1	0	0	1	-2	2
1	0	0	0	1	3	2
0	0	1	0	$-\frac{7}{5}$	$\frac{4}{5}$	$\frac{1}{5}$
0	0	0	1	0	-2	1
0	0	0	0	7	1	14

Iz tabele zaključujemo da je bazno rješenje problema $x = (2, 2, \frac{1}{5}, 1)$, a optimalno $x^* = (2, 2)$, odakle dobijamo i optimalno rješenje početnog problema.

Primjer 7.6. U ovom primjeru ćemo prikazati ILP problem u kojem su svi koeficijenti u ograničenjima cijelobrojni.

$$\max 3x_1 + 4x_2$$

t.d.

$$3x_1 - x_2 \leq 12$$

$$3x_1 + 11x_2 \leq 66$$

$$x_1, x_2 \geq 0$$

$$x_1, x_2 \in \mathbb{Z}$$

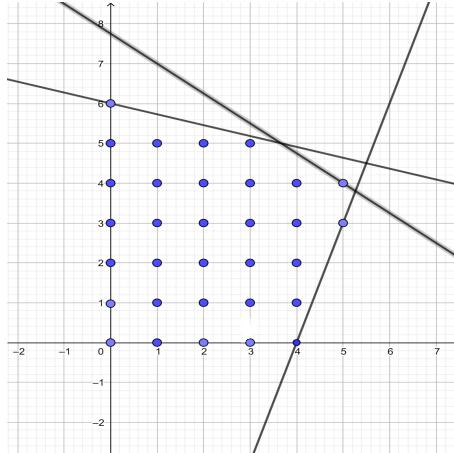
Rješenje. Na Slici 7.3 dat je skup tačaka koje su dopustive za problem. Funkcija cilja dostiže optimum u tački $x^* = (5, 4)$.

Riješimo sada ovaj problem uz pomoć Gomorijevih odsjecačućih ravnih. Početna simpleks tabela je data sa

3	-1	1	0	12
3	11	0	1	66
-3	-4	0	0	0

Nakon dvije iteracije pivotiranja, dobijamo simpleks tabelu

1	0	$\frac{11}{36}$	$\frac{1}{36}$	$\frac{11}{2}$
0	1	$-\frac{1}{12}$	$\frac{1}{12}$	$\frac{9}{2}$
0	0	$\frac{7}{12}$	$\frac{5}{12}$	$\frac{69}{2}$



Slika 7.3: Dopustiv skup tačaka problema.

Optimalno rješenje je $x = (\frac{11}{2}, \frac{9}{2}, 0, 0)$ koje nije cijelobrojno. Dodajmo Go-morijevu odsjecajuću ravan generisanu iz ograničenja prve vrste simpleks tabele (za prvu baznu komponentu $x_1 = \frac{11}{2}$):

$$\frac{11}{36}x_2 + \frac{1}{36}s_1 - y_1 = \frac{1}{2},$$

pa dobijamo tabelu

1	0	$\frac{11}{36}$	$\frac{1}{36}$	0	$\mid \frac{11}{2}$
0	1	$-\frac{1}{12}$	$\frac{1}{12}$	0	$\mid \frac{9}{2}$
0	0	$\frac{11}{36}$	$\frac{1}{36}$	-1	$\mid \frac{1}{2}$
0	0	$\frac{7}{12}$	$\frac{5}{12}$	0	$\mid \frac{69}{2}$

Kako bazno dopustivo rješenje nije očigledno, primijenimo dvofazni simpleks metod da bi izabrali bazu. Dobijamo tabelu

1	0	0	0	1	$\mid 5$
0	1	0	$\frac{1}{11}$	$-\frac{3}{11}$	$\mid \frac{51}{11}$
0	0	1	$\frac{1}{11}$	$-\frac{36}{11}$	$\mid \frac{18}{11}$
0	0	0	$\frac{4}{11}$	$\frac{21}{11}$	$\mid \frac{369}{11}$

Kako su svi koeficijenti u posljednjoj vrsti veći od 0, slijedi da je optimalno bazno rješenje jednakoj $x = (5, \frac{51}{11}, \frac{18}{11}, 0, 0)$, koje opet nije cjelobrojno. Prema tome, dodajmo sljedeću Gomorijevu odsjecajuću ravan za bazno rješenje $x_2 = \frac{51}{11}$ (posmatrajući vrstu 2 posljednje simpleks tabele):

$$\frac{1}{11}s_2 + \frac{8}{11}y_1 - y_2 = \frac{7}{11},$$

odakle imamo tabelu

1	0	0	0	1	0	5
0	1	0	$\frac{1}{11}$	$-\frac{3}{11}$	0	$\frac{51}{11}$
0	0	1	$\frac{1}{11}$	$-\frac{36}{11}$	0	$\frac{18}{11}$
0	0	0	$\frac{1}{11}$	$\frac{8}{11}$	-1	$\frac{7}{11}$
0	0	0	$\frac{4}{11}$	$\frac{21}{11}$	0	$\frac{369}{11}$

I ovdje nije očigledno šta su bazne promjenljive, pa i ovdje primijenimo dvofazni simpleks metod. Nakon toga, dobijamo

1	0	0	$-\frac{1}{8}$	0	$\frac{11}{8}$	$\frac{33}{8}$
0	1	0	$\frac{1}{8}$	0	$-\frac{3}{8}$	$\frac{39}{8}$
0	0	1	$\frac{1}{2}$	0	$-\frac{9}{2}$	$\frac{9}{2}$
0	0	0	$\frac{1}{8}$	1	$-\frac{11}{8}$	$\frac{7}{8}$
0	0	0	$\frac{1}{8}$	0	$\frac{21}{8}$	$\frac{255}{8}$

Odgovaraće optimalno bazno dopustivo rješenje i dalje nije cjelobrojno – niti jedna komponenta nije cjelobrojna.

Prema tome, dodajemo novu Gomorijevu odsjecajuću ravan za vrstu 2 prethodne simpleks tabele, koja je oblika

$$\frac{1}{8}s_2 + \frac{5}{8}y_2 - y_3 = \frac{7}{8},$$

te dobijamo tabelu

1	0	0	$-\frac{1}{8}$	0	$\frac{11}{8}$	0	$\left \frac{33}{8} \right.$
0	1	0	$\frac{1}{8}$	0	$-\frac{3}{8}$	0	$\left \frac{39}{8} \right.$
0	0	1	$\frac{1}{2}$	0	$-\frac{9}{2}$	0	$\left \frac{9}{2} \right.$
0	0	0	$\frac{1}{8}$	1	$-\frac{11}{8}$	0	$\left \frac{7}{8} \right.$
0	0	0	$\frac{1}{8}$	0	$\frac{5}{8}$	-1	$\left \frac{7}{8} \right.$
0	0	0	$\frac{1}{8}$	0	$\frac{21}{8}$	0	$\left \frac{255}{8} \right.$

I ovdje nije očigledno koju promjenljivu treba dodati u bazne promjenljive (bazu), pa primijenimo dvofazni simpleks metod, odakle dobijamo

1	0	0	0	1	0	0	$\left 5 \right.$
0	1	0	0	$-\frac{1}{2}$	0	$\frac{1}{2}$	$\left 4 \right.$
0	0	1	0	$-\frac{7}{2}$	0	$\frac{1}{2}$	$\left 1 \right.$
0	0	0	1	$\frac{5}{2}$	0	$-\frac{11}{2}$	$\left 7 \right.$
0	0	0	0	$-\frac{1}{2}$	1	$-\frac{1}{2}$	$\left 0 \right.$
0	0	0	0	1	0	2	$\left 31 \right.$

gdje je lako uočiti da je optimalno bazno dopustivo rješenje $x = (5, 4, 1, 7, 0, 0, 0)$, tj. rješenje početnog problema je $x^* = (5, 4)$.

7.4 B&C algoritam za rješavanje ILP-a

U ovom odjeljku analiziramo metodu granjanja i odsijecanja (eng. branch and cut, skraćeno B&C). U odnosu na B&B metodu, koja je opisana u Odjeljku 7.1, B&C dodaje i metod odsjecačujućih ravni u rješavanju podproblema koji odgovaraju čvorovima drveta. Dakle, ako uzimamo trenutno aktivni čvor i rješavamo odgovarajući podproblem, dijelimo ga na nekoliko podproblema (podjelom dopustivog regiona na više disjunktnih – dodavanjem ograničenja) ili ga rješavamo rekurzivnim dodavanjem (Gomorijevih) odsjecačujućih ravni. Ilustrujmo ovu ideju na primjeru rješavanja jednog ILP-a.

Primjer 7.7. Riješimo sljedeći ILP

$$\max 6x_1 + 5x_2$$

t.d.

$$3x_1 + x_2 \leq 11$$

$$-x_2 + 2x_2 \leq 5$$

$$x_1, x_2 \geq 0$$

$$x_1, x_2 \in \mathbb{Z}$$

Rješenje. B&C procedura prvo rješava LP relaksaciju početnog problema (simpleks metodom), odakle dobijamo rješenje $x = (\frac{17}{7}, \frac{26}{7})$ sa optimalnom vrijednošću $33\frac{1}{7}$. Nakon toga treba da se donese odluka, da li se problem rješava dodavanjem odsjecajućih ravni ili podjelom dopustivog regiona na manje (disjunktne) podregione. Podijelimo region po koordinati x_1 , odakle ćemo dobiti dva podproblema

$$\max 6x_1 + 5x_2$$

t.d.

$$3x_1 + x_2 \leq 11$$

$$-x_2 + 2x_2 \leq 5 \quad (P1)$$

$$x_1 \geq 3$$

$$x_2 \geq 0$$

$$x_1, x_2 \in \mathbb{Z}$$

$$\begin{aligned}
& \max 6x_1 + 5x_2 \\
& \text{t.d.} \\
& 3x_1 + x_2 \leq 11 \\
& -x_2 + 2x_2 \leq 5 && (P2) \\
& x_1 \leq 2 \\
& x_2 \geq 0 \\
& x_1, x_2 \in \mathbb{Z}
\end{aligned}$$

Rješavanjem relaksacije problema $(P1)$ dobijamo rješenje $x^* = (3, 2)$, koje je cjelobrojno, pa prema tome i dopustivo u početnom problemu. Vrijednost funkcije cilja je 28, što je i donja granica (LB) početnog problema. Relaksacija problema $(P2)$ nam daje rješenje $x^* = (2, \frac{7}{2})$, čija je optimalna vrijednost 29.5. Prema tome, rješenje nije cjelobrojno. Primjenjujući metod Gomorijevih odsjecajućih ravnih na problem $(P2)$, konstruišemo odsjecajuću ravan $2x_1 + x_2 \leq 7$ koju dodajemo u problem $(P2)$, pa dobijamo problem

$$\begin{aligned}
& \max 6x_1 + 5x_2 \\
& \text{t.d.} \\
& 3x_1 + x_2 \leq 11 \\
& -x_2 + 2x_2 \leq 5 \\
& 2x_1 + x_2 \leq 7 && (P3) \\
& x_1 \leq 2 \\
& x_2 \geq 0 \\
& x_1, x_2 \in \mathbb{Z}
\end{aligned}$$

Rješavajući LP relaksaciju problema $(P3)$, dobijamo rješenje $x^* = (\frac{9}{5}, \frac{17}{5})$, a optimalna vrijednost funkcije cilja je 27.8. Kako je $LB \geq 27.8$, niti jedno cjelobrojno rješenje problema $(P3)$ ne može biti bolje od trenutno najbolje donje granice ($LB = 28$), pa zaključujemo da je optimalno rješenje početnog problema dato sa $x^* = (3, 2)$.

7.5 Lagranžova relaksacija

U rješavanju problema optimizacije, pronalazak optimalnog rješenja je osnovni cilj. Međutim, za mnoge probleme nije realno očekivati da se, u razumnom vremenu, može doći do optimalnog rješenja za instance velikih dimenzija. Umjesto toga, najbolje što se može učiniti je da se ponudi dobra granica za optimalnu vrijednost. Ako je ta granica blizu optimuma, u mnogo slučajeva ona može da zamijeni optimum, čime štedimo resurse kojima raspolažemo, a dobijamo rješenje koje je prihvatljivo.

Ideja *Lagranžove relaksacije* se sastoji u tome da u modelu ILP-a koji ima ograničenja koja problem čine teškim (ova procjena najčešće dolazi uz praktičnog iskustva), posmatramo kao dio ciljne funkcije koji se penalizuje nekim parametrom. Ovo ograničenje potom brišemo iz skupa ograničenja modela, a zatim rješavamo ovakav (parametrizovan) problem. Fiksirajući parametar penalizacije, rješenje takvog problema predstavlja donju granicu (za problem minimizacije) optimalnog rješenja inicijalnog problema. Dobijena granica je često dovoljno blizu vrijednosti optimuma, što ovu metodu čini upotrebljivom u rješavanju praktičnih problema.

Primjer 7.8. Posmatrajmo problem *najkraćeg puta sa ograničenjem* (eng. Constrained shortest path problem). Neka je dat usmjeren neciklični graf $G = (V, E)$. Za svaku granu $(i, j) \in E$ dodijelimo cijenu puta $c_{i,j}$ i vrijeme putovanja $t_{i,j}$. Pretpostavimo da je čvor $i = 1$ početni (izvorni) čvor, a čvor $n = |V|$ završni čvor (terminalni čvor). Zadatak je da se nađe najjeftiniji put od izvornog do terminalnog čvora, ali tako da se čitav put pređe u vremenskom periodu T .

Ovaj problem se modeluje pomoću sljedećeg ILP-a.

$$z^* = \min \sum_{(i,j) \in E} c_{i,j} x_{i,j} \quad (7.4)$$

$$\text{t.d.} \quad (7.5)$$

$$\sum_i x_{i,j} - \sum_i x_{j,i} = \begin{cases} 1, & \text{ako } i = 1 \\ -1, & \text{ako } i = n \\ 0, & \text{inače} \end{cases}, \forall j \in V \quad (7.6)$$

$$\sum_{(i,j) \in E} t_{i,j} x_{i,j} \leq T, \forall j \in V \quad (7.7)$$

$$x_{i,j} \in \{0, 1\}, \forall (i, j) \in E. \quad (7.8)$$

Binarna promjenljiva $x_{i,j}$ dobija vrijednost 1 ako je na putu odabrana grana (i, j) , a 0 inače. Prvo ograničenje u modelu osigurava da je dopustivo rješenje upravo put od izvornog do terminalnog čvora. Drugo ograničenje nam modeluje uslov da vrijeme prelaska puta treba da bude u granicama dopustivog. Ova ograničenja spadaju u „teška”, jer se njihovim dodavanjem u model, njegovo rješavanje značajno otežava (vrijeme potrebno za rješavanje se drastično povećava i/ili kompleksnost problema se teorijski povećava – npr. sa P na NP). Kao što znamo, pronalazak najkraćeg puta u usmjerenim grafovima, sa unaprijed definisanim izvornim čvorom, je polinomijalno rješiv (na primjer, uz pomoć Dajkstrinog algoritma). Međutim, dodavanjem ograničenja u vezi vremena, taj problem postaje NP-težak. Stoga, ideja je relaksirati ovaj problem uz prebacivanje ograničenja za vrijeme u funkciju cilja, koja se zatim kažnjava penalom. Dakle, Lagranžova relaksacija problema je data sa

$$\begin{aligned} L(\lambda) &= \min \sum_{(i,j) \in E} c_{i,j} x_{i,j} + \lambda \left(\sum_{(i,j) \in E} t_{i,j} x_{i,j} - T \right) \\ &= \min \sum_{(i,j) \in E} (c_{i,j} + t_{i,j}) x_{i,j} - \lambda T \end{aligned}$$

t.d.

$$\sum_i x_{i,j} - \sum_i x_{j,i} = \begin{cases} 1, & \text{ako } i = 1 \\ -1, & \text{ako } i = n \\ 0, & \text{inače,} \end{cases} \quad \forall j \in V$$
$$x_{i,j} \in \{0, 1\}, \forall (i, j) \in E.$$

Primijetimo da je $L(\lambda) \leq z(\lambda) \leq z^*$, za sve $\lambda > 0$, gdje je $z(\lambda)$ problem koji ima funkciju cilja $L(\lambda)$ uz uključeno ograničenje o vremenu u skup ograničenja modela. Dakle, za $\lambda > 0$, $L(\lambda)$ nam daje donju granicu za optimalno rješenje početnog problema.

Problem $L^* = \max\{L(\lambda) \mid \lambda > 0\}$ se naziva *problem Lagranžovog množioca*. Može se pokazati da vrijedi

$$L(\lambda) \leq L^* \leq z^*.$$

U praksi se pokazalo da je Lagranžova relaksacija dobra tehnika za dobijanje granica optimalne vrijednosti problema.

Primjer 7.9. Primijenimo ovu tehniku na *Problem trgovačkog putnika* sa ulaznim težinskim grafom $G = (V, E)$, gdje je $V = \{1, \dots, n\}$. Razmotrimo problem sa grafovske strane i pokušajmo da izvedemo njegovu relaksaciju.

Relaksacija: definijemo sa T_1 skup svih ruta u G koje nakon brisanja grana koje pokrivaju čvor 1 formiraju (pokrivajuće) stablo. Dakle, prije samog rješavanja problema, enumerišimo sve ovakve rute u grafu G .

Shodno tome, konstruišemo sljedeći model

$$z^* = \min \sum_{(i,j) \in E} c_{i,j} x_{i,j}$$

t.d.

$$\sum_{\{i \mid (i,j) \in E\}} x_{i,j} = 2, \forall i \in \{1, \dots, n\}$$
$$x \in T_1,$$

gdje $x_{i,j} = 1$ ako grana (i, j) pripada ruti, a 0 inače. Ako je $x = (i, j) \in T_1$, to znači da postoji neka ruta u G u kojoj su čvorovi i i j susjedni, gdje se nakon brisanja grana čvora 1, formira (pokrivajuće) stablo grafa G .

Lagranžova relaksacija za ovaj problem može se zadati sa

$$L(\lambda) = \min \sum_{(i,j) \in E} c_{i,j}^\lambda x_{i,j} - 2 \sum_i \lambda_i \\ x \in T_1$$

gdje je $\lambda = (\lambda_1, \dots, \lambda_n)$ i $c_{i,j}^\lambda = c_{i,j} + \lambda_i + \lambda_j$.

U praksi se pokazalo da su rješenja dobijena Lagranžovom relaksacijom blizu optimalnim rješenju problema. Optimalno pokrivajuće stablo Lagranžovog problema $L(\lambda^*)$ za optimalno λ^* će nam, u praksi, često dati rješenje sa nekoliko visećih čvorova.

Konstruišimo još jednu Lagranžovu relaksaciju za ovaj problem, uvodeći tzv. *ograničenja pokrivanja*. Neka je dat skup $S \subseteq V$. Ograničenje pokrivanja u odnosu na skup S osigurava postojanje najviše $|S| - 1$ grana u svakoj ruti kojima oba kraja pripadaju čvorovima iz S . Ovom smo ograničili pojavljivanje ciklusa među čvorovima u konačnom rješenju. Prethodni scenario modelujemo sljedećim modelom

$$z^* = \min \sum_{(i,j) \in E} c_{i,j} x_{i,j} \\ \text{t.d.} \\ \sum_{\{i|(i,j) \in E\}} x_{i,j} = 2, \forall i \in \{1, \dots, n\} \\ \sum_{(i,j) \in E \wedge i,j \in S} x_{i,j} \leq |S| - 1, \forall S \subseteq V \\ \sum_{(i,j) \in E} x_{i,j} = n \\ x_{i,j} \in \{0, 1\}, (i, j) \in E.$$

Lagranžova relaksacija prethodnog ILP-a ima oblik

$$L(\lambda) = \min \sum_{(i,j) \in E} c_{i,j}^\lambda x_{i,j} - 2 \sum_i \lambda_i$$

t.d.

$$\sum_{(i,j) \in E} x_{i,j} \leq |S| - 1, \forall S \in V$$

$$\sum_{(i,j) \in E} x_{i,j} = n$$

gdje je $\lambda = (\lambda_1, \dots, \lambda_n)$ i $c_{i,j}^\lambda = c_{i,j} + \lambda_i + \lambda_j$.

7.6 Metod generisanja kolona

U praksi se često nailazi na situaciju da je u LP-u (ili ILP-u) broj promjenljivih značajno (eksponencijalno) veći u odnosu na broj ograničenja. Kod takvih ekstremno velikih modela, matrica modela je izuzetno velikih dimenzija. Sa druge strane, veliki dio matrice (tj. njene kolone) nam obično nikada i neće biti relevantan za rješavanje. Da bi se riješila specifična instanca problema, ono što nam je potrebno su

- Ograničenja koja se vezuju uz optimalnost.
- Promjenljive koje su bazne u optimalnom rješenju.

Dakle, ono što je potrebno da znamo su ograničenja, čije su odgovaraajuće dualne vrijednosti pozitivne, kao i promjenljive koje imaju pozitivne vrijednosti koeficijenata doprinosa za optimalno rješenje (tj. vektor \bar{c} iz simpleks tabele). Ako bismo ove promjenljive i ograničenja imali na raspolaganju od samog početka, problem bi bio riješen efikasno. Međutim, do ove informacije nije jednostavno doći. U simpleks metodi, vektor doprinosa je potrebno računati u svakoj iteraciji, što nije efikasno ako je matrica A velikih dimenzija. Takođe, vrijeme potrebno za generisanje simpleks tabele bi bilo ogromno. Oba ova problema su rješiva uz pomoć *metode generisanja kolona*. Ideja ovog metoda se sastoji u sljedećem.

-
- Izaberimo podskup S kolona matrice A u cilju određivanja potencijalno dobrog inicijalnog baznog dopustivog rješenja.
 - Riješimo restrikovani LP na podskupu kolona S .
 - Procijenimo ostale kolone i dodajmo u S one sa negativnim cijenama doprinosa.
 - Iterativno pozivamo perthodna dva koraka dok god postoje kolone sa negativnim doprinosom koje nisu u S .

Tehnički zapisano, krećemo od *restrikovanog master problema* (RMP)

$$\begin{aligned} & \min c^T x \\ & \text{t.d} \\ & \sum_{i \in I} A_i x_i \leq b \\ & x \geq 0. \end{aligned} \tag{RMP}$$

gdje I inicijalizujemo kao skup kolona koje čine inicijalno bazno dopustivo rješenje za RMP. Po mogućnosti, dodajemo i neke druge kolone za koje smatramo da su „bitne” u kontekstu problema. Riješimo RMP, te izračunamo odgovarajuće optimalno dualno rješenje. Sada je potrebno naći promjenljivu x_j i odgovarajuću kolonu A_j za koju je koeficijent doprinosa negativan, tj. $\bar{c}_j = c_j - c_B^T B^{-1} A_j < 0$. Ovo se može uraditi rješavanjem tzv. podproblema generisanog kolonama (eng. *column generating subproblem – CGSP*):

$$\min_{a \in C} \bar{c}_j = \min_{a \in C} c_a - c_B^T B^{-1} a,$$

gdje je C skup kolona nad nekim globalnim skupom. Za LP problem možemo staviti $C = \{1, \dots, n\}$.

Ako je rješenje problema kolona $a \in C$ za koju je $\bar{c}_a < 0$, dodamo odgovarajuću vektor kolonu A_a u skup S , te riješimo RMP na novom skupu kolona. Ako ne postoji takva kolona, onda je rješenje RMP takođe optimalno

i za početni problem. Proces dodavanja kolona se nastavlja sve dok postoji kolona $a \in C$ sa negativnom cijenom doprinosa.

Postoje više različitih pristupa koje se odnose na ažuriranje skupa S tokom iteracija. Najjednostavniji pristup je zadržavanje svake kolone koja je do sada izgenerisana u skup S i dodavanje nove u postojeći skup. Naprednija strategija je zasnovana na izbacivanju onih kolona iz skupa S koje su nebazne u svakoj iteraciji.

Bazna šema za generisanje kolona. U osnovi, šema se sastoji od dva koraka:

1. Za ulazne kolone prvo izaberemo one sa najvećim negativnim koeficijentom doprinosa, među promjenljivima koje odgovaraju skupu I .
2. Jednom kada su svi koeficijenti nenegativni (tj. kada smo našli optimalnu vrijednost za RMP), tražimo nove promjenljive (kolone) van skupa I (rješavajući CGSP), koje će biti ubaćene u I .

Ako u koraku 1 za skup I uzmemu bazne kolone, u svakom koraku pri rješavanju CGSP dobijamo kolonu koja ulazi kao bazna i zamjenjuje kolonu iz I . To znači da rješavanje RMP traje samo jednu iteraciju (simpleks metoda), jer uvijek imamo m kolona i novu ulaznu kolonu A_a . Dakle, ovaj proces ne uzima mnogo memorije i rješavanje RMP je efikasno. Težište ovog pristupa je prebačeno na korak 2 i rješavanje CGSP. Ako ne postoji efikasan način rješavanja ovog problema, onda je i efikasna primjena CG okvira teško ostvariva.

Iterativni proces rješavanja RMP problema se može prekinuti i ranije, bez dostizanja optimalnog rješenja početnog problema uslijed velikog broja iteracija. U ovom slučaju, moguće je da postoji kolona van RMP problema koja će eventualno ući u konstrukciju optimalne baze. Takođe, ako RMP nije riješen do optimalnosti u nekoj iteraciji, ali CGSP jeste, i dalje se garantuje rješavanje početnog (LP) problema do optimalnosti kada se (master) iteracije prekinu. Prema tome, rješavanje RMP problema u svakoj iteraciji nije toliko bitno dok god rješavamo CGSP do optimalnosti u svakom koraku. Obrnuto ne vrijedi. Ako bismo iskoristili neku heuristiku za rješavanje CGSP, onda se optimalnost početnog LP-a ne može garantovati. Razlog je

vrlo jednostavan, jer heuristika treba da nađe bilo koju kolonu sa negativnim koeficijentom doprinosa koja će biti ubačena u RMP, čime se proces uspješno nastavlja. Sa druge strane, ako heuristika ne uspije da pronađe takve kolone, to ne znači da takva kolona ne postoji.

U nastavku primjenimo *metode generisanja kolona* na poznatom *Cutting stock* problemu.

Primjer 7.10. Fabrika papira proizvodi rolne papira fiksne širine W . Kupci naručuju različit broj rolni različite širine $w \in \mathbb{R}^m$. Vektorom $b \in \mathbb{R}^m$ se definiše zahtjev koji se odnosi za broj naručenih odgovarajućih rolni. Zadatak je pronaći optimalan način rezanja velike rolne da bi se zahtjevi ispunili, a smanjila količina otpada nastala nakon rezanja.

Rješenje. Na Slici 7.4 je dato rješenje jedne instance ovog problema. U ovom problemu, kolone predstavljaju dopustive obrasce (kombinacije rezanja). Jedan dio obrasca sa slike bi bio 1820×2 (rolne) u prvoj koloni. Primjetimo da broj različitih obrazaca raste eksponencijalno u odnosu na broj narudžbi. Ako bismo izvršili enumeraciju svih mogućih obrazaca, trebali bismo riješiti sljedeći ILP model

$$\begin{aligned} & \min \sum_i x_i \\ & \sum_{ij} a_{i,j} x_j \geq b_i, \forall i \\ & x_j \geq 0, x_j \in \mathbb{Z} \end{aligned} \tag{CSILP}$$

Parametar $a_{i,j}$ predstavlja broj koliko se puta u narudžbi i (određena širina – visina – sjećenja rolne) pojavljuje obrazac j (odgovara vrsti na Slici 7.4). Promjenljivom x_j se određuje širina j -tog sjećenja rolne.

Primjera radi, na Slici 7.4 indeks 1 odgovara obrascu (dužine) 1820, indeks 2 obrascu (dužine) 1380, indeks 3 obrascu 2150, indeks 4 obrascu 1930, itd. Nadalje, prvo sjećenje sa slike odgovara sljedećim vrijednostima: $a_{1,1} = 3, a_{1,2} = a_{1,3} = \dots = 0$, i $x_1 = 2$. Ovo znači da u prvom sjećenju visine 2, obrazac 1 (tj. dužina 1820) se pojavljuje tri puta, dok se ostali obrasci ne pojavljuju ovdje. Drugo sjećenje (narudžba) odgovara sljedećim

	0	1120	2240	3360	4480	5600 mm
2x	1820		1820		1820	
3x	1380		2150		1930	
12x	1380		2150		2050	
7x	1380		2100		2100	
12x	2200		1820		1560	
8x	2200		1520		1880	

Slika 7.4: Rješenje jedne instance.

vrijednostima: $a_{1,1} = 0, a_{1,2} = 1, a_{1,3} = 1, a_{1,4} = 0 = a_{1,5} = \dots = 0$, dok je $x_2 = 3$. Ovo znači da se u drugom sjećenju visine 3, obrazac 1820 ne pojavljuje, obrazac 1380 pojavljuje jednom, kao i obrasci 2150 i 1930, dok se ostali obrasci ne pojavljuju.

Ideja kojom se vodimo pri konstrukciji algoritma je da se obrasci ne formiraju na početku procedure, već da se generišu prema potrebi (po kolonama), koristeći *metodu generisanja kolona*. Kolona a odgovara dopustivom obrascu akko vrijedi

$$\sum_{i=1}^m a_i w_i \leq W,$$

gdje vektor a sadrži samo nenegativne (cjelobrojne) vrijednosti.

Primijetimo da su, na osnovu prethodnog, razmatrajući relaksaciju problema *CSILP*, vrijednosti koeficijenata doprinosa jednaki $\bar{c} = 1 - \sum_i a_{j,i} y_i$.

Dakle, CG podproblem izgleda ovako

$$\begin{aligned} \max \quad & \sum_{i=1}^m y_i a_i \\ \text{t.d.} \quad & \begin{aligned} \sum_i w_i a_i &\leq W \\ a_i &\geq 0 \\ a_i &\in \mathbb{Z}, \end{aligned} \end{aligned} \tag{CGP}$$

gdje su y_i vrijednosti optimuma duala.

Ovaj problem je poznat pod nazivom *cjelobrojni problem jednodimenzionalnog ruksaka*. Iako je NP-težak, on se u praksi relativno efikasno rješava uz pomoć dinamičkog programiranja.

Za ovaj problem je lako naći inicijalno bazno dopustivo rješenje. Za i -tu baznu kolonu uzimamo i -ti jedinični vektor, tj. obrazac dobijen rezanjem jedne rolne dužine w_i . Ovakav skup kolona zaista formira inicijalnu dopustivu bazu. Svakako, malo je vjerovatno da će ove kolone biti korištene u optimalnom rješenju, ali to nije ni očekivano od inicijalnog rješenja. Algoritam za rješavanje ovog problema je sljedeći.

1. Konstruišimo inicijalno bazno dopustivo rješenje (jedinični vektori) i kolone ubacimo u skup S .
2. Riješimo relaksaciju restrikovanog problema (*CSILP*) i vratimo dužno optimalno rješenje.
3. Riješimo (*CGP*).
4. Ako je vrijednost optimalnog rješenja negativna, ubaciti novu kolonu a (koja je rješenje podproblema) u skup S . Inače, prekidamo proces jer je optimalno rješenje nađeno.
5. Riješiti relaksaciju (*CSILP*) na novom skupu kolona S .

Primjetimo da je dodavanje nove kolone u S odgovara dodavanju novog obrasca u rješenje.

Primjer 7.11. Riješimo sada jednu instancu ovog problema pomocu CG okvira. Recimo da imamo $W = 15$, te tri narudžbe za sjećenje:

-
1. stavka: $b_1 = 80$ rolni dužine 4;
 2. stavka: $b_2 = 50$ rolni dužine 6;
 3. stavka: $b_3 = 100$ rolni dužine 7.

Definišimo sa $A_j = \begin{pmatrix} a_4 \\ a_6 \\ a_7 \end{pmatrix}$ patern j , koji označava koliko kojih ronli smo ugradili u veliku rolnu na jediničnu širinu. Enumerišimo sve paterne i otpad koji se generiše njime:

$$A_1 = \begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix}, A_2 = \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix}, A_3 = \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix}, A_4 = \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix}, A_5 = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}, A_6 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix},$$

sa veličinom otpada

$waste_1 = 3, waste_2 = 3, waste_3 = 1, waste_4 = 0, waste_5 = 1, waste_6 = 2$, redom. ILP model koji odgovara gore ulaznim podacima je sljedeći:

$$\min x_1 + x_2 + x_3 + x_4 + x_5 + x_6$$

s.t.

$$\begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix} x_1 + \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix} x_2 + \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix} x_3 + \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix} x_4 + \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} x_5 + \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} x_6 = \begin{pmatrix} 80 \\ 50 \\ 100 \end{pmatrix}$$

$$x_i \geq 0, x_i \in \mathbb{Z}, i = 1, \dots, 6.$$

Uzmimo prve tri promjenljive x_1, x_2 i x_3 koji će generisati početni skup kolona I . Dakle, imamo sljedeći restrikovani MP:

$$\min x_1 + x_2 + x_3$$

s.t.

$$\begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix} x_1 + \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix} x_2 + \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix} x_3 = \begin{pmatrix} 80 \\ 50 \\ 100 \end{pmatrix}$$

$$x_i \geq 0, x_i \in \mathbb{Z}, i = 1, 2, 3.$$

Da bismo problem posmatrali sa jednostavnije strane, izbacimo uslove integralnosti (cjelobrojnosti promjenjivih). Tada ovaj problem možemo da rije-

šimo uz pomoć simpleks metoda, gdje je $B^{-1} = \begin{pmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{pmatrix}$, dok završna

(optimalna) simpleks tabela ima oblik

$$\begin{array}{cccc|c} x_1 & \frac{1}{3} & 0 & 0 & \frac{80}{3} \\ x_2 & 0 & \frac{1}{2} & 0 & 25 \\ x_3 & 0 & 0 & \frac{1}{2} & 50 \\ \hline z & \frac{1}{3} & \frac{1}{2} & \frac{1}{2} & \frac{305}{3} \end{array}$$

Sljedeći korak je naći kolonu čiji je koeficijent doprinosa negativan i ubaciti je potom u restrikovani MP u cilju poboljšanja trenutnog rješenja. Dakle, treba da riješimo CGSP

$$\min_{i=1,\dots,n} \bar{c}_j = \min_{i=1,\dots,n} 1 - y^T N \quad (7.9)$$

gdje su N vektori (kolone) koji odgovaraju nebaznim promjenljivim. Ovaj problem je ekvivalentan rješavanju problema

$$\max_{j=1,\dots,n} y^T A_j, \quad (7.10)$$

gdje su A_j paterni, $j = 1, \dots, 6$. Ako je rješenje veće od 0, optimalno rješenje nađeno rješavanjem prethodnog RMP je optimalno i za početni problem. Inače, biramo promjenljivu (i odgovarajuću kolonu) sa minimalnim negativnim doprinosom koju potom ubacujemo u RMP. Kako je svaki patern dat u opštem slučaju kao $A_j = \begin{pmatrix} a_4 \\ a_6 \\ a_7 \end{pmatrix}$, CG podproblem se svodi na sljedeći optimizacioni problem

$$\max y_1 a_4 + y_2 a_6 + y_3 a_7 \quad (7.11)$$

uz zadovoljenje uslova o maksimalnoj širini

$$4a_4 + 6a_6 + 7a_7 \leq 15 \quad (7.12)$$

$$a_4, a_6, a_7 \geq 0, a_4, a_6, a_7 \in \mathbb{Z}. \quad (7.13)$$

Kako smo to i ranije pomenuli, ovaj problem je klasični problem ruksaka koji se u praksi rješava veoma efikasno. Dakle, ako je rješenje problema ruksaka manje ili jednako od 1, trenutno najbolje rješenje je optimalno rješenje. U protivnom, optimalno rješenje $(a_4^*, a_6^*, a_7^*)^T$ se uključuje u RMP kao nova kolona. Na osnovu prethodne simpleks tabele imamo $y_1 = \frac{1}{3}, y_2 = y_3 = \frac{1}{2}$, pa treba da riješimo sljedeći CG problem

$$\begin{aligned} & \max \frac{1}{3}a_4 + \frac{1}{2}a_6 + \frac{1}{2}a_7 \\ & 4a_4 + 6a_6 + 7a_7 \leq 15 \\ & a_4, a_6, a_7 \geq 0, a_4, a_6, a_7 \in \mathbb{Z}. \end{aligned}$$

Pomoću algoritma dinamičkog programiranja, riješimo ovaj problem i dobijamo rješenje $(a_4^*, a_6^*, a_7^*)^T = (2, 0, 1)^T = A_4$, optimalne vrijednosti $\frac{7}{6}$.

Konvertujemo vektor $\begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix}$ na bazu koja je generisana vektorima baze x_1 , x_2 i x_3 . Iz toga slijedi da se vektor $B^{-1}A_4 = \begin{pmatrix} \frac{2}{3} \\ 0 \\ \frac{1}{3} \end{pmatrix}$ uključuje u novu simpleks tabelu (uz redukovano cijenu doprinosa od $-(1 - \frac{7}{6}) = \frac{1}{6}$):

x_1	$\frac{1}{3}$	0	0	$\frac{2}{3}$	$\frac{80}{3}$
x_2	0	$\frac{1}{2}$	0	0	25
x_3	0	0	$\frac{1}{2}$	$\frac{1}{3}$	50
z	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{305}{3}$

U novoj koloni tražimo pivot element (kao i u simpleks metodi). Lako je vidjeti da je pivot element $\frac{2}{3}$, pa primjenjujući elementarne transformacije po vrstama oko pivota, dobijamo simpleks tabelu

x_4	$\frac{1}{2}$	0	0	1	40
x_2	0	$\frac{1}{2}$	0	0	25
x_3	$-\frac{1}{4}$	0	$\frac{1}{2}$	0	30
z	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{2}$	0	95

Dalje, generišimo novi CG podproblem čitajući vrijednost dualnih promjenjivih iz prethodne simpleks tabele $y_1 = \frac{1}{4}$, $y_2 = \frac{1}{2}$, $y_3 = \frac{1}{2}$, a dobijamo novi podproblem

$$\frac{1}{4}a_4 + \frac{1}{2}a_6 + \frac{1}{2}a_7$$

uz uslove nenegativnosti i kapaciteta i integralnosti (cjelobrojnosti). Rješavajući problem dinamičkim programiranjem, dobijamo optimalno rješenje $A_5 = (2, 1, 0)^T$. Vrijednost optimuma za CG podproblem je jednaka 1, odašte slijedi da je optimum početnog problema pronađen sa prethodnim rješenjem (ne postoji ulazna kolona koje se dodaju u RMP). Rješenje je jednako 95 i dato je sa $x_1^* = 0, x_2^* = 25, x_3^* = 30, x_4^* = 40$. Dakle, koristimo samo 3 različita obrasca za sječenje.

7.7 Zadaci

- Pomoću metode grananja i ograničavanja (B&B) riješiti sljedeći problem ILP-a

$$\begin{aligned}
 & \max 100x_1 + 150x_2 \\
 & \text{t.d.} \\
 & 8000x_1 + 4000x_2 \leq 40000 \\
 & 15x_1 + 30x_2 \leq 200 \\
 & x_1, x_2 \geq 0 \\
 & x_1, x_2 \in \mathbb{Z}.
 \end{aligned}$$

- Pomoću metode grananja i ograničavanja (B&B) riješiti sljedeći pro-

blem ILP-a

$$\max 5x_1 + 4x_2$$

t.d.

$$3x_1 + 4x_2 \leq 10$$

$$x_1, x_2 \geq 0$$

$$x_1, x_2 \in \mathbb{Z}.$$

Nakon primjene B&B metode, riješiti problem grafički dijeljeći region na podregione i uoprediti dobijena rješenja.

3. Pomoću metode grananja i ograničavanja (B&B) riješiti sljedeći problem ILP-a

$$\max 5x_1 + 6x_2 + 4x_3$$

t.d.

$$5x_1 + 3x_2 + 6x_3 \leq 20$$

$$x_1 + 3x_2 \leq 12$$

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

$$x_2 \in \mathbb{Z}.$$

4. Pomoću metode implicitne enumeracije riješiti sljedeći ILP

$$\max 1000x_1 + 700x_2 + 800x_3$$

t.d.

$$5000x_1 + 6000x_2 + 4000x_3 \leq 10000$$

$$x_1, x_2, x_3 \in \{0, 1\}.$$

5. Pomoću metode implicitne enumeracije riješiti sljedeći ILP

$$\begin{aligned} & \max 20x_1 + 30x_2 + 10x_3 + 40x_4 \\ & \text{t.d.} \\ & 2x_1 + 4x_2 + 3x_3 + 7x_4 \leq 10 \\ & 10x_1 + 7x_2 + 20x_3 + 15x_4 \leq 40 \\ & x_1 + 10x_2 + x_3 \leq 10 \\ & x_1, x_2, x_3, x_4 \in \{0, 1\}. \end{aligned}$$

6. Metodom Gomorijevih odsjecajućih ravni, riješiti sljedeći ILP

$$\begin{aligned} & \min -x_1 - x_2 \\ & \text{t.d.} \\ & 2.1x_1 + 5.2x_2 \leq 20 \\ & 4.5x_1 + 3.5x_2 \leq 17 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

7. Metodom Gomorijevih odsjecajućih ravni, riješiti sljedeći ILP

$$\begin{aligned} & \max x_1 + 4x_2 \\ & \text{t.d.} \\ & 2x_1 + 4x_2 \leq 7 \\ & 5x_1 + 3x_2 \leq 15 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

8. Uz pomoć algoritma B&C riješiti sljedeći problem ILP-a

$$\begin{aligned} & \max x_1 + 2x_2 \\ \text{t.d.} \\ & -2x_1 + 2x_2 \leq 5 \\ & 6x_1 + 4x_2 \leq 25 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

9. Za sljedeći problem

$$\begin{aligned} & \min c^T x \\ \text{t.d.} \\ & Ax = b \\ & Dx \geq q \\ & x \geq 0 \end{aligned}$$

napisati odgovarajuću Lagranžovu relaksaciju.

10. Neka je dat problem ILP-a

$$\begin{aligned} & \max x_2 \\ \text{t.d.} \\ & -2x_1 + 2x_2 \leq 1 \\ & 2x_1 + 2x_2 \leq 7 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

Definisati Lagranžovu relaksaciju za ovaj problem (npr. uzeti prvo ograničenje za „teško”). Riješiti početni problem sa jednom od tehniku za rješavanje ILP-a. Potom, riješiti Lagranžovu relaksaciju problema za $\lambda = 0.25$. Šta možemo zaključiti na osnovu rješenja Lagranžove relaksacije i početnog problema?

-
11. Pomoću CG okvira (generisanje kolona), riješiti jedni instancu *Cutting stock* problema gdje je $W = 20$, te postoje tri narudžbe za sječenje:
1. stavka: $b_1 = 60$ rolni dužine 7; 2. stavka: $b_2 = 40$ rolni dužine 8; 3. stavka: $b_3 = 70$ rolni dužine 10.

Glava 8

Dekompozicione metode

Dekompozicione metode u matematičkom programiranju su tehnike pomoću kojih se polazni problem razbija u više manjih problema, te se, rješavanjem tih manjih problema, na kraju dolazi i do rješenja polaznog.

Široka primjena metoda linearog, cjelobrojnog i mješovitog cjelobrojnog linearog programiranja (MILP) u modelovanju i rješavanju velikog broja različitih problema iz teorije i prakse, doveli su do porasta interesa za razvoj i primjenu dekompozicionih metoda. Čak se i tradicionalne metode za rješavanje problema cjelobrojnog programiranja, kao što su B&B ili od-sjecajuće ravni, takođe mogu smatrati dekompozicionim metodama, jer one dijele polazni problem na LP relaksaciju i cjelobrojna ograničenja.

Treba napomenuti da i savremeni MILP rješavači, o čemu će biti govora kasnije, takođe, u svojim implementacijama koriste i dekompozicione algoritme, koji, u nekim slučajevima, lakše i brže dovode do rješenja. U ovom poglavlju razmatraćemo dvije standardne dekompozicione metode: Benderovu i Dantzig-Wolfe dekompoziciju.

8.1 Benderova dekompozicija

Benderova metoda je uvedena šezdesetih godina dvadesetog vijeka, kao algoritam za rješavanje MILP programa. Osnovna ideja Benderove dekom-

pozicije primijenjenog na MILP program je podjela skupa promjenljivih na dva podskupa, od kojih se jedan skup promjenljivih (skup cjelobrojnih promjenljivih) smatra „komplikovanim”, odnosno skupom promjenljivih čijim se fiksiranjem dolazi do značajnog pojednostavljenja polaznog problema. Prvi problem, koji se naziva i master problem, rješava se nad prvim skupom promjenljivih, dok se vrijednosti promjenljivih iz drugog skupa, za dato rješenje prvog podproblema, određuju rješavanjem drugog podproblema. Uspješnu primjenu Benderove dekompozicije, između ostalog, obezbjeđuje i teorija dualnosti, odnosno teoreme o slaboj i jakoj dualnosti.

Posmatrajmo sada sljedeću formulaciju MILP programa

$$\begin{aligned} \min \quad & f^T x + g^T y \\ \text{t.d.} \quad & \\ Ax + By \geqslant & c \\ x \geqslant & 0, \quad y \in \mathbb{Z} \end{aligned} \tag{8.1}$$

gdje su x i y vektori neprekidnih i cjelobrojnih promjenljivih, respektivno, dimenzija p i q . f, g, c su vektori, a A i B matrice odgovarajućih dimenzija.

Pretpostavimo da raspolažemo jednom početnom vrijednošću vektora y . Za tu vrijednost (označimo je sa \bar{y}) dobijamo sljedeći podproblem

$$\begin{aligned} \min \quad & f^T x + g^T \bar{y} \\ \text{t.d.} \quad & \\ Ax \geqslant & c - B\bar{y} \\ x \geqslant & 0 \end{aligned} \tag{8.2}$$

Posmatrajmo dualni problem podproblema (8.2), ignorišući sabirak $g^T \bar{y}$, koji je konstantan.

$$\begin{aligned} \max \quad & u^T(c - B\bar{y}) \\ \text{t.d.} \quad & \\ u^T A \leqslant & f \\ u \geqslant & 0 \end{aligned} \tag{8.3}$$

Sada uključujemo teoriju dualnosti i razmatramo sljedeće mogućnosti, koje mogu da se javi.

1. Ako je optimalno rješenje problema (8.2) z^* , a optimalno rješenje duala (8.3) u^* , na osnovu teoreme o jakoj dualnosti imamo

$$z^* - g^T \bar{y} = (u^*)^T (c - B\bar{y}) \quad (8.4)$$

Dalje, uočimo da oblast dopustivih rješenja dualnog problema (8.3) ne zavisi od vrijednosti \bar{y} . Stoga, u^* je uvijek dopustivo rješenje problema (8.3), bez obzira na vrijednost \bar{y} . Odavde, na osnovu teoreme o slaboj dualnosti dobijamo

$$z \geq g^T y + (u^*)^T (c - By) \quad (8.5)$$

Posljednja nejednakost određuje jednu donju granicu funkcije cilja polaznog problema. Ovo je ključni element koji se koristi u Benderovoj dekompoziciji, pomoću kog se određuje jedna odsjecajuća ravan za master problem, koji će biti definisan kasnije. Treba napomenuti da odsjecajuća ravan (8.5) ukazuje na činjenicu da, ako bi se za y uzela vrijednost \bar{y} , funkcija cilja bi bila veća ili jednaka vrijednosti \bar{z} , koja se dobije u podproblemu, s obzirom da važi $z \geq g^T y + (u^*)^T (c - B\bar{y})$.

Ovo takođe daje i donju granicu za druge vrijednosti y koje bismo mogli probati u narednim iteracijama.

2. Posmatrajmo sada slučaj kada je dualni podproblem neograničen. U toj situaciji važi nejednakost

$$(u^*)^T (c - By) > 0 \quad (8.6)$$

gdje je $(u^*)^T$ vektor koji definiše pravac mogućih rješenja duž kog funkcija neograničeno raste. Ovakav pravac se naziva i *ekstremni zrak*.

U ovom slučaju, cilj je eliminisati ovaj zrak iz daljeg razmatranja (budućih rješenja). Da bismo to uradili, potrebno je osigurati da izraz $(u^*)^T (c - By)$ bude manji ili jednak od nule u svim narednim koracima. Tako, master problemu dodajemo novu odsjecajuću ravan, definisanu sa

$$(u^*)^T (c - By) \leq 0 \quad (8.7)$$

Sada možemo definisati master problem koji nastaje dodavanjem svih odsjecajućih ravni generisanih u prethodnim koracima.

$$\begin{aligned} & \min z \\ & \text{t.d.} \\ & z \geq g^T y + (u^k)^T (c - By), \quad k \in K \\ & (u^k)^T (c - By) \leq, \quad k \in L \\ & y \in \mathbb{Z}, \end{aligned} \tag{8.8}$$

gdje je K skup ekstremnih tačaka, a L skup ekstremnih zrakova podproblema. Rješavanjem master problema (koji je obično puno lakši od početnog problema) dobijamo novu vrijednost za y , koju koristimo u sljedećoj iteraciji.

Treba napomenuti da je podproblem (8.2) restrikcija originalnog problema (8.1), pa je njegova optimalna vrijednost z^* jedna gornja granica za funkciju cilja polaznog problema. Takođe, master problem (8.8) je relaksacija polaznog problema, jer se pri rješavanju master problema ne posmatraju sva ograničenja. Stoga, optimalna vrijednost master problema je donja granica za funkciju cilja originalnog problema.

Postupak primjene Benderove dekompozicije na problem minimizacije prikazan je Algoritmom 2.

Algoritam 2 Benderova dekompozicija

- 1: Postavi $\bar{y} \in \mathbb{Z}$ na inicijalnu vrijednost
- 2: $LB \leftarrow -\infty; UB \leftarrow +\infty$
- 3: **while** $UB - LB > \epsilon$ **do**
- 4: Riješi dualni problem (Benderov podproblem)

$$\begin{aligned} \max \quad & u^T(c - B\bar{y}) \\ \text{t.d.} \quad & u^T A \leq f \\ & u \geq 0. \end{aligned}$$

- 5: **if** podproblem nema dopustivo rješenje **then**
- 6: KRAJ // nema rješenja
- 7: **end if**
- 8: **if** podproblem ima optimalno rješenje **then**
- 9: $UB \leftarrow \min\{UB, g^T y + (u^k)^T(c - By)\}$
- 10: Generiši Benderovu odsjecajuću ravan $z \geq g^T y + (u^k)^T(c - By)$ i dodaj je u master problem.
- 11: **end if**
- 12: **if** podproblem nije ograničen **then**
- 13: Generiši Benderovu dopustivu ravan $(u^k)^T(c - By) \leq 0$ i dodaj je u master problem.
- 14: **end if**
- 15: Riješi Benderov master problem:

$$\begin{aligned} \min \quad & z \\ \text{t.d.} \quad & \\ & g^T y + (u^k)^T(c - By), \quad k \in K \\ & (u^k)^T(c - By), \quad k \in L \\ & y \in \mathbb{Z} \end{aligned}$$

i odredi novu vrijednost \bar{y} i vrijednost rješenja z^* .

- 16: $LB \leftarrow \max\{LB, z^*\}$
 - 17: **end while**
-

U primjeru kojim ilustrujemo upotrebu Benderove dekompozicije posmatramo jedan jednostavan problem maksimizacije dobiti na osnovu ulaganja po različitim kriterijumima. Iako je Algoritmom 2 opisan postupak rješavanja problema minimizacije, algoritam se vrlo lako može preformulirati tako da se može primijeniti i na problem maksimizacije. Jasno je da će u tom slučaju, dualni problem, prikazan u liniji 4 algoritma tada postati problem minimizacije, te će se, rješavanjem dualnog podproblema dobiti donje granice.

Primjer 8.1. Prepostavimo da raspolažemo sumom od 1000KM, koju možemo iskoristiti za ostvarivanje dobiti prema dva različita kriterijuma:

- Oročenje novca sa fiksnom kamatom u iznosi od 4.5%. Dodatno ograničenje je da se može oročiti samo cjelobrojan iznos novca.
- Ulaganje u neke od 10 vrsta dionica ($f_1, f_2, f_3, \dots, f_{10}$), sa stopama rasta 1%, 2%, ..., 10%, respektivno. Za svaku vrstu dionica, može se investirati maksimalno 100KM.

Postavlja se pitanje kako treba uložiti početnu količinu novca, tako da se ostvari maksimalna dobit.

Napomena. Prije nego što krenemo sa rješavanjem problema, naglasimo da ovako formulisan problem predstavlja pojednostavljeni, i ne previše realističnu verziju problema portfelja u ekonomiji, koji pripada oblasti optimizacije protfelja (eng. portfolio optimization). Inače, realni problem ulaganja novca pripada dvo-kriterijumskoj optimizaciji, gdje se pored maksimizacije profita, minimizuje i rizik ulaganja.

Rješenje. S obzirom na postavku zadatka, jasno je da bi optimalno rješenje trebalo da podrazumijeva da se uloži u dionice f_5, \dots, f_{10} sa maksimalnim mogućim ulaganjem (po 100KM), dok se ostatak novca oročava sa kamatom od 4.5%.

Sada ćemo problem formalno definisati i riješiti ga metodom Benderove dekompozicije.

Neka su promjenljive definisane sa:

x_1, x_2, \dots, x_{10} – iznos uložen u dionice $f_1, f_2, f_3, \dots, f_{10}$, respektivno

y – iznos koji je oročen sa fiksnom kamatom u iznosi od 4.5%. Sada se problem može formulisati na sljedeći način

$$\begin{aligned}
 & \max \quad 1.01x_1 + 1.02x_2 + \dots + 1.10x_{10} + 1.045y \\
 & \text{t.d.} \\
 & x_1 + x_2 + \dots + x_{10} + y \leq 1000 \\
 & x_1 \leq 100 \\
 & \vdots \\
 & x_{10} \leq 100 \\
 & y \in \mathbb{Z} \\
 & x_1, x_2, \dots, x_{10}, y \geq 0
 \end{aligned} \tag{8.9}$$

Zapišimo sada model u obliku koji odgovara postavci metoda Benderove dekompozicije.

$$\begin{aligned}
 & \max \quad f^T x + g^T y \\
 & \text{t.d.} \\
 & Ax + By \leq c \\
 & x \geq 0, \quad y \in \mathbb{Z}^+
 \end{aligned} \tag{8.10}$$

gdje je

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{10} \end{pmatrix}, \quad f = \begin{pmatrix} 1.01 \\ 1.02 \\ \vdots \\ 1.10 \end{pmatrix}$$

Primijetimo da su u našem primjeru g i y jednodimenzionalni vektori, koje ćemo u daljem zapisu identifikovati sa njihovom (jedinom) koordinatom, odnosno $y = (y)$ i $g = (1.045)$. Takođe je i

$$A = \begin{pmatrix} 1 & \dots & 1 \\ 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \text{i} \quad c = \begin{pmatrix} 1000 \\ 100 \\ \vdots \\ 100 \end{pmatrix}$$

Primijenimo sada Benderovu dokompoziciju. Najprije treba odrediti početnu vrijednost za \bar{y} , za koju, u ovom koraku, možemo uzeti bilo koju nene-gativnu vrijednost. Neka je, na primjer, $\bar{y} = 1500$. Dalje, neka su $LB = -\infty$ i $UB = +\infty$ i uzmimo da je $\epsilon = 0.1$.

Nakon inicijalnih postavki, prvi korak je rješavanje dualnog problema (Benderovog podproblema). S obzirom da je početni problem problem maksimizacije, dualni problem će sada biti problem minimizacije. Obilježimo promjenljive dualnog problema vektorom u , koji ima 11 koordinata (polazni problem ima 11 oraničenja).

Računamo funkciju cilja dualnog problema

$$u^T(c - B\bar{y}) = (u_1 \dots u_{11}) \cdot \left(\begin{pmatrix} 1000 \\ 100 \\ \vdots \\ 100 \end{pmatrix} - \begin{pmatrix} 1500 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \right) = \\ = -500u_1 + 100u_2 + \dots + 100u_{11}$$

dok su ograničenja dualnog problema

$$u_1 + u_2 \geq 1.01$$

$$u_1 + u_3 \geq 1.02$$

\vdots

$$u_1 + u_{11} \geq 1.1$$

$$u_1 \geq 1.045$$

$$u \geq 0$$

Lako se vidi da rješenje dualnog problema nije ograničeno (sa donje strane), za $u_1 = +\infty$, $u_2 = u_3 = \dots = u_{11} = 0$. Odnosno,

$$u = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

je ekstremni zrak.

Prateći tok algoritma, određujemo odsjecajuću ravan

$$u^T(c - By) \geq 0$$

odnosno

$$1000 - y \geq 0$$

Sada rješavamo master problem

$$\begin{aligned} & \max z \\ & \text{t.d.} \\ & 1000 - y \geq 0 \\ & y \in \mathbb{Z}^+ \end{aligned}$$

Vidimo da, bez obzira na vrijednost y , vrijednost funkcije cilja nije ograničena. Drugim riječima, ako uzmemo bilo koju dopustivu vrijednost za y (recimo $\bar{y} = 1000$) dobijamo da je $\bar{z} = +\infty$. Sada možemo zaključiti da u prvoj iteraciji algoritma nismo uspjeli da smanjimo gornju granicu, te nam ostaje $UB = +\infty$.

Prelazimo na sljedeću iteraciju. Formulišimo novi dualni problem.

Računamo novu funkciju cilja dualnog problema sa vrijednošću $\bar{y} = 1000$

$$u^T(c - B\bar{y}) = (u_1 \dots u_{11}) \cdot \left(\begin{pmatrix} 1000 \\ 100 \\ \vdots \\ 100 \end{pmatrix} - \begin{pmatrix} 1000 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \right) = 100u_2 + \dots + 100u_{11}$$

Imajući u vidu ograničenja dualnog problema, zaključujemo da za vrijednost vektora

$$u = \begin{pmatrix} 1.1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

funkcija cilja ima minimalnu vrijednost jednaku 0.

Sada ćemo moći da popravimo donju granicu, odnosno

$$LB = \max\{LB, g^T y + (u)^T(c - By)\} = \max\{-\infty, 1.045 * 1000 + 0\} = 1.045.$$

Dalje, dodajemo odsjecajuću ravan kao ograničenje master problema:

$$z \leq g^T y + (u)^T(c - By),$$

odnosno

$$\begin{aligned} z &\leq 1.045y + \begin{pmatrix} 1.1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}^T \left(\begin{pmatrix} 1000 \\ 100 \\ \vdots \\ 100 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} y \right) \\ z &\leq 1.045y + \begin{pmatrix} 1.1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}^T \begin{pmatrix} 1000 - y \\ 100 \\ \vdots \\ 100 \end{pmatrix} \end{aligned}$$

Odnosno

$$z \leq 1100 - 0.055y$$

Sada novodobijenu odsjecajuću ravan dodajemo ograničenjima master problema i rješavamo master problem

$$\begin{aligned} \max z \\ 1000 - y \geq 0 \\ z \leq 1100 - 0.055y \\ y \in \mathbb{Z}^+ \end{aligned}$$

Nije teško odrediti da je optimalna vrijednost funkcije cilja master problema $\bar{z} = 1100$, koja se dobija za vrijednost $\bar{y} = 0$. Sada možemo da ažuriramo gornju granicu koristeći vrijednost funkcije cilja, odnosno

$$UB = \min\{UB, \bar{z}\} = 1100$$

Pošto je razlika između gornje i donje granice još uvijek veća od ϵ ($UB - LB = 1100 - 1045 = 55$), prelazimo na sljedeću iteraciju.

Uzimamo $\bar{y} = 0$ i ponovo rješavamo dualni problem.

Funkcija cilja dualnog problema sada izgleda

$$\begin{aligned} u^T(c - B\bar{y}) &= (u_1 \dots u_{11}) \cdot \left(\begin{pmatrix} 1000 \\ 100 \\ \vdots \\ 100 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \right) = \\ &= 1000u_1 + 100u_2 + \dots + 100u_{11}. \end{aligned}$$

S obzirom na ograničenja u dualnom problemu, funkcija ima minimalnu vrijednost 1055 za vektor

$$u = \begin{pmatrix} 0 \\ 1.01 \\ 1.02 \\ \vdots \\ 1.1 \end{pmatrix}.$$

Pošto dual ima optimalno rješenje, računamo novu donju granicu, odnosno,

$$LB = \max\{LB, g^T y + (u)^T(c - By)\} = \max\{1.045, 0 + 1055\} = 1055$$

Dobijamo novu odsjecajuću ravan koju ćemo pridodati za rješavanje master problema

$$z \leq 1.045y + \begin{pmatrix} 0 \\ 1.01 \\ \vdots \\ 1.1 \end{pmatrix}^T \left(\begin{pmatrix} 1000 \\ 100 \\ \vdots \\ 100 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} y \right).$$

Odnosno

$$z \leq 1.045y + 1055$$

Sada novi master problem izgleda

$$\begin{aligned} & \max z \\ & 1000 - y \geq 0 \\ & z \leq 1100 - 0.055y \\ & z \leq 1.045y + 1055 \\ & y \in \mathbb{Z}^+ \end{aligned}$$

Rješavanjem ovog problema cjelobrojnog programiranja (na primjer, pomoću nekog od računarskih programa) dobija se optimalno rješenje za cjelobrojnu vrijednost $\bar{y} = 41$, koje iznosi $\bar{z} = 1097.745$

Sada možemo da ažuriramo i gornju granicu, odnosno

$$UB = \min\{1100, \bar{z}\} = \min\{1100, 1097.745\} = 1097.745$$

Računamo razliku između gornje i donje granice

$$UB - LB = 1097.745 - 1055 = 42.745$$

i pošto je razlika još uvijek veća od ϵ , nastavljamo dalje sa iterativnim postupkom.

Uzmimo $\bar{y} = 41$ i ponovo formirajmo i riješimo dualni problem. Funkcija cilja dualnog problema sada izgleda

$$\min 959u_1 + 100u_2 + \dots + 100u_{11},$$

a rješavanjem problema uz data ograničenja dobijamo optimalno rješenje 1013.59, za vrijednosti vektora

$$u = \begin{pmatrix} 1.01 \\ 0 \\ 0.01 \\ 0.02 \\ \vdots \\ 0.09 \end{pmatrix}$$

Ažuriranjem donje granice dobija se

$$LB = \max\{1055, 1.045 \cdot 41 + 1013.59\} = \max\{1055, 1056.435\} = 1056.435$$

Dalje, dodajemo novu odsjecajuću ravan

$$z \leq 1.045y + \begin{pmatrix} 1.01 \\ 0 \\ 0.01 \\ 0.02 \\ \vdots \\ 0.09 \end{pmatrix}^T \left(\begin{pmatrix} 1000 \\ 100 \\ \vdots \\ 100 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} y \right).$$

Odnosno

$$z \leq 1.045y + 1010 - 1.01y + 100 \cdot 0.45$$

t.j.

$$z \leq 0.035y + 1055$$

Ovo ograničenje dodajemo master problemu, koji sada izgleda

$$\begin{aligned} & \max z \\ & 1000 - y \geq 0 \\ & z \leq 1100 - 0.055y \\ & z \leq 1.045y + 1055 \\ & z \leq 0.035y + 1055 \\ & y \in \mathbb{Z}^+ \end{aligned}$$

Rješavanjem ovog master problema dobijamo optimalno rješenje za cje-lobrojnu vrijednost $\bar{y} = 500$, koje iznosi $\bar{z} = 1072.5$.

Ažuriramo i gornju granicu

$$UB = \min\{1097.745, 1072.5\} = 1072.5.$$

Računamo razliku između gornje i donje granice $1072.5 - 1056.435$, što je još uvijek veće od ϵ , pa prelazimo na narednu iteraciju.

Uzimamo $\bar{y} = 500$ i formiramo novi dualni problem sa funkcijom cilja

$$\min 500u_1 + 100u_2 + \dots + 100u_{11},$$

čija optimalna vrijednost iznosi 540, za vrijednost vektora u

$$u = \begin{pmatrix} 1.05 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.01 \\ 0.02 \\ \vdots \\ 0.05 \end{pmatrix}.$$

Ažuriramo donju granicu,

$$LB = \max\{1056.435, 1.045 \cdot 500 + 540\} = 1062.5$$

Dalje, master problemu pridružujemo odsjecajuću ravan

$$z \leq 1.045y + \begin{pmatrix} 1.05 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.01 \\ 0.02 \\ \vdots \\ 0.05 \end{pmatrix}^T \left(\begin{pmatrix} 1000 \\ 100 \\ \vdots \\ 100 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} y \right).$$

Odnosno

$$z \leq 1.045y + 1050 - 1.05y + 100 \cdot 0.15$$

t.j.

$$z \leq -0.005y + 1065$$

Kada i ovo ograničenje dodamo master problemu, dobijamo novi master problem

$$\begin{aligned} & \max z \\ & \text{t.d.} \\ & 1000 - y \geq 0 \\ & z \leq 1100 - 0.055y \\ & z \leq 1.045y + 1055 \\ & z \leq 0.035y + 1055 \\ & z \leq -0.005y + 1065 \\ & y \in \mathbb{Z}^+ \end{aligned}$$

čije se optimalno rješenje iznosi $\bar{z} = 1063.75$, a dobija se za cijelobrojnu vrijednost $\bar{y} = 250$.

Ažuriranjem gornje granice dobijamo

$$UB = 1063.75,$$

pa je razlika između gornje i donje granice jednaka $1063.75 - 1062.5 = 1.25$, što je još uvjek veće od ϵ .

Ponovo formiramo dualni problem uzimajući $\bar{y} = 250$.

Nova funkcija cilja sada izgleda

$$\min 750u_1 + 100u_2 + \dots + 100u_{11},$$

a optimalna vrijednost funkcije cilja je 800.5 za vrijednost vektora u

$$u = \begin{pmatrix} 1.03 \\ 0 \\ 0 \\ 0 \\ 0.01 \\ 0.02 \\ 0.03 \\ 0.04 \\ 0.05 \\ 1.06 \\ 1.07 \end{pmatrix}.$$

Ispitujemo da li se ažurira donja granica:

$$LB = \max\{1062.5, 1.045 \cdot 250 + 800.5\} = \max\{1062.5, 1061.75\} = 1062.5.$$

Dalje, određujemo odsjecajuću ravan

$$z \leq 1.045y + \begin{pmatrix} 1.03 \\ 0 \\ 0 \\ 0 \\ 0.01 \\ 0.02 \\ 0.03 \\ 0.04 \\ 0.05 \\ 0.06 \\ 0.07 \end{pmatrix}^T \left(\begin{pmatrix} 1000 \\ 100 \\ \vdots \\ 100 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} y \right).$$

Odnosno

$$z \leq 1.045y + 1030 - 1.03y + 28,$$

t.j.

$$z \leq 0.015y + 1058$$

i dodajemo je master problemu, koji izgleda

$$\begin{aligned} & \max z \\ & \text{t.d.} \\ & 1000 - y \geq 0 \\ & z \leq 1100 - 0.055y \\ & z \leq 1.045y + 1055 \\ & z \leq 0.035y + 1055 \\ & z \leq -0.005y + 1065 \\ & z \leq 0.015y + 1058 \\ & y \in \mathbb{Z}^+ \end{aligned}$$

Rješavanjem master problema dobijamo novu vrijednost $\bar{y} = 350$, a gornja granica se ažurira na

$$UB = 1063.25.$$

Ponavljamajući postupak još jednom, dobijamo redom:

- vrijednost funkcije duala 697,
- vektor

$$u = \begin{pmatrix} 1.04 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.01 \\ 0.02 \\ 0.03 \\ 0.04 \\ 0.05 \\ 1.06 \end{pmatrix}$$

- $LB = 1062.75$,
- odsjecajuću ravan: $z \leq 0.005y + 1061$.

Rješavajući još jednom master problem, koji sada izgleda

$$\begin{aligned} & \max z \\ \text{s.t.} \\ & 1000 - y \geq 0 \\ & z \leq 1100 - 0.055y \\ & z \leq 1.045y + 1055 \\ & z \leq 0.035y + 1055 \\ & z \leq -0.005y + 1065 \\ & z \leq 0.015y + 1058 \\ & z \leq 0.005y + 1061 \\ & y \in \mathbb{Z}^+ \end{aligned}$$

dobijamo da je nova vrijednost $\bar{y} = 400$, a gornja granica $UB = 1063$.

Formirajući i rješavajući (pokazaće se, po posljednji put), dualni problem, dobija se optimalno rješenje 645, pa donja granica iznosi 1063.

Vidimo da smo dobili da je donja granica jednaka gornjoj i možemo obustaviti iterativni postupak.

Sada znamo da se optimalno rješenje polaznog problema (8.10), odnosno (8.9), dobija za vrijednost $\bar{y} = 400$, te se on svodi na problem

$$\begin{aligned} \max \quad & f^T x \\ \text{s.t.} \quad & Ax \leq c - B\bar{y} \\ & x \geq 0 \end{aligned} \tag{8.11}$$

čije je rješenje predstavljeno vektorom

$$x = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 100 \\ 100 \\ 100 \\ 100 \\ 100 \\ 100 \end{pmatrix}$$

8.2 Dantzig–Wolfe dekompozicija

U ovom odjeljku ćemo objasniti još jednu korisnu dekompozicionu tehniku za rješavanje problema linearнog programiranja – *Dantzig–Wolfe* (DW) dekompoziciju. Ova tehnika je originalno dizajnirana za rješavanje problema linearнog programiranja velikih dimenzija. Osnovna ideja DW dekompozicije je da se na podskup ograničenja polaznog problema primjeni *Teorema Minkovskog*, odnosno *Teorema o reprezentaciji za konveksne poliedre*, te da se polazni problem razbije na više manjih podproblema.

Teorema 8.1. Neka je $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ ograničeni poliedar i neka je $\{x^{(1)}, \dots, x^{(p)}\}$ skup ekstremnih tačaka poliedra P . Tada za sve $x \in P$ vrijedi

$$x = \sum_{i=1}^p \lambda_i x^{(i)},$$

gdje je $\lambda_i \geq 0, i = 1, \dots, p, \sum_{i=1}^p \lambda_i = 1$.

Podimo od kanonske formulacije problema linearog programiranja

$$\begin{aligned} & \min c^T x \\ & \text{t.d.} \\ & Ax = b \\ & x \in X \end{aligned} \tag{8.12}$$

gdje je skup X poliedar. Iako za DW dekompoziciju nije neophodno, u daljem razmatranju, radi lakše analize DW dekompozicije, smatraćemo da je X ograničen poliedar, odnosno politop.

Dalje, navešćemo i sljedeću pretpostavku. Ako bismo iz skupa ograničenja izbacili ograničenja $Ax = b$, tada bi rješavanje problema bilo jednostavno. Drugim riječima, skup X je opisan nekim „jednostavnim” ograničenjima, dok ograničenja $Ax = b$ čine problem značajno težim, pa ćemo ih, zbog toga, zvati „komplikovanim ograničenjima”.

Posmatrajmo sada konfiguraciju za primjenu Teoreme o reprezentaciji za konveksne poliedre na skup X .

Neka su tačke $x^{(1)}, x^{(2)}, \dots, x^{(t)}$ vrhovi politopa X . Prema Teoremi o reprezentaciji za konveksne poliedre, za svaku tačku $x \in X$ vrijedi da se ona može predstaviti kao konveksna kombinacija (ekstremnih) tačaka $x^{(1)}, x^{(2)}, \dots, x^{(t)}$.

Neka je $x \in X$ proizvoljna tačka iz skupa X . Tada postoji $\lambda_1, \lambda_2, \dots, \lambda_t$,

takvi da važi:

$$\begin{aligned} x &= \sum_{j=1}^t \lambda_j x^{(j)} \\ \sum_{j=1}^t \lambda_j &= 1 \\ \lambda_j &\geq 0, \quad j = 1, \dots, t \end{aligned} \tag{8.13}$$

Ponovimo da Teorema o reprezentaciji za konveksne poliedre ima svoju formulaciju i u slučaju da poliedar nije ograničen, ali zbog pojednostavljenja daljeg objašnjenja DW dekompozicije, ovaj slučaj nećemo razmatrati.

Uzimajući u obzir prikazanu reprezentaciju proizvoljne tačke $x \in X$, početni problem (8.12) sada možemo preformulisati na

$$\begin{aligned} \min \quad & c^T \cdot \sum_{j=1}^t \lambda_j x^{(j)} \\ \text{t.d.} \quad & A \cdot \sum_{j=1}^t \lambda_j x^{(j)} = b \\ & \sum_{j=1}^t \lambda_j = 1 \\ & \lambda_j \geq 0, \quad j = 1, \dots, t \end{aligned} \tag{8.14}$$

odnosno

$$\begin{aligned}
 & \min \sum_{j=1}^t (c^T x^{(j)}) \lambda_j \\
 & \text{t.d.} \\
 & \sum_{j=1}^t (Ax^{(j)}) \lambda_j = b \\
 & \sum_{j=1}^t \lambda_j = 1 \\
 & \lambda_j \geq 0, \quad j = 1, \dots, t
 \end{aligned} \tag{8.15}$$

Vidimo da se početni problem (8.12) sada može predstaviti ekvivalentnom reprezentacijom (8.15), u kojoj se, umjesto polaznih promjenljivih (vektora x), sada posmatraju promjenljive λ_j , $j = 1, \dots, t$, uz pretpostavku da su vrhovi $x^{(1)}, x^{(2)}, \dots, x^{(t)}$ konveksnog politopa X poznati.

Skup promjenljivih λ_j , $j = 1, \dots, t$ ćemo zvati λ promjenljive. Variranjem vrijednosti λ promjenljivih po dopustivom skupu iz formulacije (8.15), generišu se tačke iz dopustivog skupa polazne formulacije (8.12). Sa druge strane, svaka tačka x iz formulacije (8.12) ima svoju reprezentaciju preko odgovarajućih vrijednosti λ promjenljivih u dopustivom skupu formulacije (8.15). S obzirom na to da i funkcije cilja odgovaraju jedna drugoj, zaključujemo da je i optimalno rješenje polaznog problema (8.12) jednak optimalnom rješenju reformulisanog problema (8.15).

Samom reformulacijom polaznog problema (8.12) u DW formulaciju (8.15) nismo mnogo postigli, jer broj vrhova politopa, a time i broj λ promjenljivih može biti jako veliki.

Međutim, pravi razlog za DW dekompoziciju leži u činjenici da za rješavanje problema (8.15) ne moramo eksplicitno da koristimo sve vrhove politopa X . Umjesto toga, krenućemo od jednog manjeg podskupa, a samim tim i podskupom skupa λ promjenljivih. Nakon toga, primijenićemo tehniku „generisanja kolona“ (eng. *column generation*), kako bismo u razmatranje uključili nove vrhove politopa X , koji će pomoći poboljšanju rješenja tzv.

ograničenog master problema (eng. *Restricted master problema* – RMP).

Označimo sada problem formulisan sa (8.15) kao *Master problem* (MP). Na osnovu master problema formiramo RMP sa manjim brojem λ promjenljivih.

$$\begin{aligned} \min \quad & \sum_{j \in I} (c^T x^{(j)}) \lambda_j \\ \text{t.d.} \quad & \sum_{j \in I} (Ax^{(j)}) \lambda_j = b \\ & \sum_{j \in I} \lambda_j = 1 \\ & \lambda_j \geq 0, \quad j \in I \end{aligned} \tag{8.16}$$

gdje je I skup indeksa onih vrhova politopa koji su uključeni u RMP.

Označimo sa \bar{z} optimalnu vrijednost funkcije cilja RMP, a sa z^* optimalnu vrijednost master problema.

Primijetimo da je \bar{z} jedna gornja granica za master problem, jer je svako dopustivo rješenje RMP takođe i dopustivo za master problem. Ova konstatacija će nam biti od koristi u razmatranju koje slijedi.

Da bismo provjerili da li je optimalno rješenje RMP ujedno i optimalno rješenje master problema, potrebno je da provjerimo da li su koeficijenti doprinosi svih nebaznih promjenjivih nenegativni, tj. postoji li neka nebazna promjenljiva koja može doprinijeti dodatnom smanjenju vrijednosti funkcije cilja. Na osnovu karakteristika (optimalne) simpleks tabele i tvrđenja koja važe na osnovu teorije dualnosti, ove vrijednosti su ekvivalentne optimalnim vrijednostima dualnih promjenjivih (duala problema RMP).

Označimo vektorom y optimalne vrijednosti dualnih promjenljivih koje se odnose na ograničenja $\sum_{j \in I} (Ax^{(j)}) \lambda_j = b$ i skalarom α optimalnu vrijednost dualne promjenljive koja se odnosi na ograničenje $\sum_{j \in I} \lambda_j = 1$.

Da bi se odredili koeficijenti doprinosu za λ_j , odredimo šta se dobija u odgovarajućoj koloni. U funkciji cilja, imamo $c^T x^{(j)}$, u ograničenjima $Ax^{(j)}$ i još jednu jedinicu (na osnovu ograničenja $\sum_{j \in I} \lambda_j = 1$). Tako, kolona na

koju se fokusiramo da bismo izračunali doprinose izgleda ovako

$$\begin{pmatrix} c^T x^{(j)} \\ Ax^{(j)} \\ 1 \end{pmatrix}$$

Sada rješavamo podproblem generisanja kolona (skraćeno CG podproblem), čiji je cilj da se pronađe najmanji koeficijent doprinosa. Ukoliko je koeficijent manji od nule, dobija se nova kolona koja treba da se doda u podproblem RMP. U slučaju da je nenegativan, tada optimalno rješenje RMP predstavlja ujedno i optimalno rješenje MP.

CG podproblem se definiše kao

$$\min_{j=1,\dots,t} \{c^T x^{(j)} - y^T A x^{(j)} - \alpha \cdot 1\} = \min_{x \in X} \{c^T x - y^T A x - \alpha\} \quad (8.17)$$

jer je rješenje problema na desnoj strani jednakosti upravo u nekom od vrhova politopa X .

Prisjetimo se sada da smo za jednu od polaznih pretpostavki imali da je skup X opisan kao skup sa „jednostavnim” ograničenjima, što znači da je problem na desnoj strani jednakosti (8.17) jednostavan za rješavanje.

Reformulišimo CG podproblem sada kao problem maksimizacije, gdje ćemo za funkciju cilja uzeti negativnu vrijednost funkcije cilja minimizacije, tj.

$$\max_{j=1,\dots,t} \{y^T A x^{(j)} - c^T x^{(j)} + \alpha\} \quad (8.18)$$

Neka je \hat{z} optimalna vrijednost funkcije cilja ovog problema maksimizacije.

Posmatrajmo sada jedno dopustivo rješenje x polaznog problema. Za to rješenje važi $x \in X$ i $Ax = b$, pa imamo da vrijedi

$$y^T A x - c^T x + \alpha \leq \hat{z} \Leftrightarrow c^T x \geq y^T A x + \alpha - \hat{z} \Leftrightarrow c^T x \geq y^T b + \alpha - \hat{z}.$$

Primijetimo sada da je $y^T b + \alpha$ zapravo optimalna vrijednost funkcije duala od RMP, koja je dalje, jednaka optimalnoj vrijednosti \bar{z} od RMP-a.

$$c^T x \geq \bar{z} - \hat{z}$$

Sada zaključujemo da je razlika između optimalne vrijednosti RMP-a i problema maksimizacije CG problema donja granica za originalni LP. Pošto ovo važi za proizvoljno rješenje x , zaključujemo da važi

$$z^* \geq \bar{z} - \hat{z}.$$

Ovim smo dobili da je razlika $\bar{z} - \hat{z}$ donja granica za optimalnu vrijednost funkcije cilja polaznog problema. Dalje, prisjetimo se da je \bar{z} gornja granica za z^* , odnosno važi

$$\bar{z} \geq z^* \geq \bar{z} - \hat{z}. \quad (8.19)$$

Izraz (8.19) se može koristiti za određivanje koliko se optimalno rješenje polaznog problema razlikuje od rješenja koja se dobijaju u iterativnom procesu DW dekompozicije.

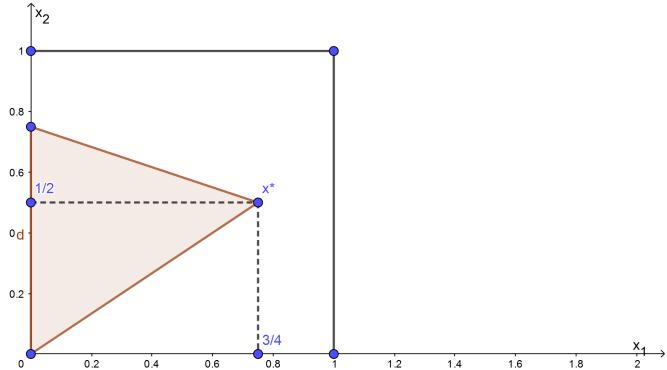
U slučaju kada je \hat{z} jednako nuli, iz (8.19) se vidi da je vrijednost \bar{z} zapravo i optimalno rješenje polaznog problema.

Primjer 8.2. Posmatrajmo problem linearog programiranja

$$\begin{aligned} & \max x_1 \\ & \text{t.d.} \\ & x_1 + 3x_2 \leq \frac{9}{4} \\ & 2x_1 - 3x_2 \leq 0 \\ & x \in X = \{(x_{1,2}) : 0 \leq x_{1,2} \leq 1\}. \end{aligned} \quad (8.20)$$

Vidimo da u ovom problemu imamo dva „komplikovana“ ograničenja ($x_1 + 3x_2 \leq \frac{9}{4}$ i $2x_1 - 3x_2 \leq 0$). Dalje, vidimo da je skup X definisan kao jedinični kvadrat, što u našem slučaju to znači da je riječ o „jednostavnom“ ograničenju.

Prije nego što na ovom problemu ilustrujemo upotrebu DW dekompozicije, riješimo problem na neki od (u ovom slučaju) bržih načina. Ako problem predstavimo grafički (Slika 8.1), vidimo da je dopustivi skup osjenčeni



Slika 8.1: Grafička interpretacija problema. Optimalno rješenje je označeno sa x^* .

trougao, a da se maksimalna vrijednost funkcije cilja dostiže u vrhu $(\frac{3}{4}, \frac{1}{2})$ tog trougla, odnosno,

$$z^* = \frac{3}{4}, \quad \text{za } x^* = \begin{pmatrix} \frac{3}{4} \\ \frac{1}{2} \end{pmatrix}. \quad (8.21)$$

Formirajmo sada polaznu konfiguraciju za primjenu DW dekompozicije. Skup X je politop koji ima četiri tjemena:

$$x^{(1)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad x^{(2)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad x^{(3)} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \text{ i } x^{(4)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Zapišimo i polazni problem u kompaktnijem obliku, odnosno

$$A = \begin{pmatrix} 1 & 3 \\ 2 & -3 \end{pmatrix} \quad b = \begin{pmatrix} \frac{9}{4} \\ 0 \end{pmatrix} \quad c = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

pa polazni problem možemo napisati kao

$$\begin{aligned} & \max c^T x \\ & \text{t.d.} \\ & Ax \leq b \\ & x \in X \end{aligned} \tag{8.22}$$

Na osnovu teoreme o reprezentaciji, svako rješenje $x \in X$ možemo zapisati kao

$$\begin{aligned} x &= \sum_{j=1}^4 \lambda_j x^{(j)} \\ \sum_{j=1}^4 \lambda_j &= 1 \\ \lambda_1, \lambda_2, \lambda_3, \lambda_4 &\geq 0 \end{aligned}$$

Preformulacijom problema (8.22) dobijamo

$$\begin{aligned} & \max \sum_{j=1}^4 (c^T x^{(j)}) \lambda_j \\ & \text{t.d.} \\ & \sum_{j=1}^4 (Ax^{(j)}) \lambda_j \leq b \\ & \sum_{j=1}^4 \lambda_j = 1 \\ & \lambda_j \geq 0, \quad j = 1, \dots, 4 \end{aligned} \tag{8.23}$$

Odnosno, nakon sređivanja

$$\max \lambda_2 + \lambda_4$$

t.d.

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix} \lambda_2 + \begin{pmatrix} 3 \\ -3 \end{pmatrix} \lambda_3 + \begin{pmatrix} 4 \\ -1 \end{pmatrix} \lambda_4 \leq \begin{pmatrix} \frac{9}{4} \\ 0 \end{pmatrix} \quad (8.24)$$

$$\sum_{j=1}^4 \lambda_j = 1$$

$$\lambda_j \geq 0, \quad j = 1, \dots, 4$$

Da bismo u ograničenjima izbjegli znak \leq u prva dva ograničenja, dodajmo modelu dvije izjednačavajuće promjenljive s_1 i s_2 , te model zapišimo kao

$$\max \lambda_2 + \lambda_4$$

t.d.

$$\lambda_2 + 3\lambda_3 + 4\lambda_4 + s_1 = \frac{9}{4}$$

$$2\lambda_2 - 3\lambda_3 - \lambda_4 + s_2 = 0 \quad (8.25)$$

$$\sum_{j=1}^4 \lambda_j = 1$$

$$\lambda_j \geq 0, \quad j = 1, \dots, 4$$

$$s_1, s_2 \geq 0$$

Sada se može inicijalnizovati RMP, biranjem dopustive baze.

Izabraćemo promjenljive λ_1, s_1 i s_2 kao inicijalne bazne promjenljive. Pošto promjenljive λ_2, λ_3 i λ_4 ne egzistiraju u podproblemu, RMP izgleda

ovako

$$\begin{aligned}
 & \max 0 \\
 & \text{t.d.} \\
 & s_1 = \frac{9}{4} \\
 & s_2 = 0 \\
 & \lambda_1 = 1 \\
 & \lambda_1, s_1, s_2 \geq 0
 \end{aligned} \tag{8.26}$$

Posljednji problem se trivijalno rješava. Funkcija cilja ima vrijednost 0, a vrijednost promjenljivih se direktno očitavaju iz ograničenja. Ispitajmo sada optimalnu vrijednost dualnih promjenjivih, dakle y i α uz pomoć simpleks tabele problema (8.26). Ona izgleda ovako

	s_1	s_2	λ_1	
s_1	1	0	0	$\frac{9}{4}$
s_2	0	1	0	0
λ_1	0	0	1	1
z	0	0	0	0

Posmatrajmo sada dualni problem problema (8.23). Označimo sa y_1 , y_2 i α promjenljive dualnog problema. Kako imamo optimalnu simpleks tabelu, vrijednost optimuma dualnih promjenjivih jednaka je odgovarajućim koeficijentima doprinosa (posljednji red tabele), pa je $y_1 = 0 = y_2 = \alpha$.

Iteracija 1. Konstruišemo CG problem sa

$$\max_{x \in X} (1, 0)^T x + 0 - (0, 0) \begin{pmatrix} 1 & 3 \\ 2 & -3 \end{pmatrix} x = \max_{x \in X} x_1. \tag{8.27}$$

Optimalna vrijednost problema (8.27) je jednaka $\hat{z} = 1$, i dostiže se, recimo, u ekstremnoj tački $x^{(2)} = (1, 0)^T$. Kako je optimalna vrijednost pozitivna, ulazna kolona za prethodni podproblem postoji, te je treba da odaberemo.

\vdots	B^{-1}	$B^{-1}b$
BV	$(y_1, y_2, \alpha) = c_B^T B^{-1}$	$\bar{z} = c_B^T B^{-1}b$

Tabela 8.1: Forma tabele za praćenje vrijednosti u iterativnom procesu DW dekompozicije

Ova kolona (ako posmatramo problem (8.25)) će biti data sa

$$\begin{pmatrix} c^T x^{(2)} \\ Ax^{(2)} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 2 \\ 1 \end{pmatrix} \quad (8.28)$$

Kada pogledamo prethodne vrijednosti kolone i master problem (8.25), jasno je da ona odgovara koeficijentima promjenljive λ_2 . Dakle, prethodna kolona je data u odnosu na λ_2 , ali je sada prebacimo na trenutnu bazu datu sa s_1, s_2 i λ_1 , gdje je $B^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. Za to je potrebno izvršiti njenu transformaciju na nove baze čiji su koraci opisani u Tabeli 8.1, odakle dobijamo kolonu za dodavanje u RMP

$$\begin{pmatrix} 1 \\ 1 \\ 2 \\ 1 \end{pmatrix}$$

Ubacimo za trenutak ovu kolonu u prethodnu simpleks tabelu, odakle imamo tabelu

	s_1	s_2	λ_1	λ_2	
s_1	1	0	0	1	$\frac{9}{4}$
s_2	0	1	0	2	0
λ_1	0	0	1	1	1
z	0	0	0	-1	0

Primijetimo da je cijena doprinosa dodate kolone promijenjena sa 1 na -1, jer je, zbog korištenja tabelarne forme simpleks metoda, optimalna vrijednost problema zapravo $-\hat{z}$, pošto se CG podproblem minimizuje. Sada koristimo novu kolonu kao ulaznu kolonu za transformacije – u njoj tražimo kandidata za pivota kako smo to i vidjeli u simpleks metodi. Odaberemo element čija je vrijednost 2 i pivotiramo oko njega. Prvo sve elemente u koloni 2 podijelimo sa 2, te potom izvršimo elementarne transformacije da bi ostali elementi novododatake kolone dobili vrijednost nula. Promjenljiva s_2 se isključuje iz baze, dok se λ_2 uključuje u bazu. Na kraju dobijamo (transformisanu) tabelu:

	s_1	s_2	λ_1	s_2	
s_1	1	$-\frac{1}{2}$	0	0	$\frac{9}{4}$
λ_2	0	$\frac{1}{2}$	0	1	0
λ_1	0	$-\frac{1}{2}$	1	0	1
z	0	$\frac{1}{2}$	0	0	0

Dakle, bazne promjenljive su sada iz skupa $\{s_1, \lambda_1, \lambda_2\}$, koji odgovara baznoj matrici $B^{-1} = \begin{pmatrix} 1 & -\frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & -\frac{1}{2} & 1 \end{pmatrix}$. Na osnovu prethodne simpleks tabele, dobijamo vrijednosti dualnih promjenljivih iz koeficijenta doprinosa, tj. $y_1 = 0, y_2 = \frac{1}{2}, \alpha = 0$.

Iteracija 2. Iz prethodnog dobijamo CG podproblem

$$\max_{x \in X} (1, 0)^T x - \left(0, \frac{1}{2}\right) \begin{pmatrix} 1 & 3 \\ 2 & -3 \end{pmatrix} x - 0 = \max_{x \in X} x_1 - x_1 + \frac{3}{2}x_2 = \max_{x \in X} \frac{3}{2}x_2. \quad (8.29)$$

Rješenje ovog problema se lako dobija, tj. optimum se dostiže u, recimo, tački $x^{(3)} = (0, 1)^T$, dok je optimalna vrijednost jednaka $\hat{z} = \frac{3}{2}$. Kako je $\hat{z} > 0$, treba da odaberemo novu kolonu koja se dodaje u RMP. Ova kolona,

ako posmatramo problem (8.25), je data sa

$$V = \begin{pmatrix} c^T x^{(3)} \\ Ax^{(3)} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \\ -3 \\ 1 \end{pmatrix}. \quad (8.30)$$

Kada pogledamo prethodne vrijednosti kolone V i master problem (8.25), jasno je da ona odgovara koeficijentima promjenljive λ_3 . Dakle, prethodna kolona je data u odnosu na promjenljivu λ_3 početnog problema, ali je sada izražavamo preko trenutne baze (podproblema) vektora s_1, s_2 i λ_2 . Dakle, nakon množenja kolone sa inverzom baze B^{-1} , na osnovu koraka opisanih u Tabeli 8.1, sljedeća kolona se dodaje u simpleks tabelu:

$$\begin{pmatrix} -\frac{3}{2} \\ \frac{9}{2} \\ \frac{2}{2} \\ -\frac{3}{2} \\ \frac{5}{2} \end{pmatrix}.$$

Dakle, dobijamo tabelu

s_1	1	$-\frac{1}{2}$	0	$\frac{9}{2}$	$\frac{9}{4}$
λ_2	0	$\frac{1}{2}$	0	$-\frac{3}{2}$	0
λ_1	0	$-\frac{1}{2}$	1	$\frac{5}{2}$	1
<hr/>	z	0	$\frac{1}{2}$	0	$-\frac{3}{2}$
					0

Sada nadimo pivota oko kolone koja je dodana. Kandidati za izlazak iz baze su ili s_1 ili λ_2 dok će mjesto njih ući promjenljiva λ_3 . Kako je $\frac{\frac{9}{4}}{\frac{9}{2}} = \frac{1}{2} > \frac{1}{\frac{5}{2}} = \frac{2}{5}$, imamo da je $\frac{5}{2}$ pivot element. Sada pivotiramo oko tog elementa, tako da se, elementarnim transformacijama po vrstama, kolona transformiše na kolonu sa svim nulama osim na poziciju pivota, gdje treba da stoji jedinica. Na taj način, simpleks tabela se transformiše u novu simpleks tabelu

s_1	1	$\frac{2}{5}$	$-\frac{9}{5}$	0	$\frac{9}{20}$
λ_2	0	$\frac{1}{5}$	$\frac{3}{5}$	0	$\frac{3}{5}$
λ_3	0	$-\frac{1}{5}$	$\frac{2}{5}$	1	$\frac{2}{5}$
z	0	$\frac{1}{5}$	$\frac{3}{5}$	0	$\frac{3}{5}$

Pročitamo vrijednost dualnih promjenjivih u posljednjoj vrsti: $y_1 = 0, y_2 = \frac{1}{5}$ i $\alpha = \frac{3}{5}$.

Iteracija 3. Prema tome, novi CG podproblem je dat sa

$$\begin{aligned} \max_{x \in X} (1, 0)^T x - (0, \frac{1}{5}) \begin{pmatrix} 1 & 3 \\ 2 & -3 \end{pmatrix} x - \frac{3}{5} &= \max_{x \in X} x_1 - \frac{2}{5}x_1 + \frac{3}{5}x_2 - \frac{3}{5} \\ &= \max_{x \in X} \frac{3}{5}x_1 + \frac{3}{5}x_2 - \frac{3}{5}. \end{aligned}$$

Optimalna vrijednost CG podproblema se dostiže u $x^{(4)} = (1, 1)^T$ i ona je jednaka $\hat{z} = \frac{3}{5}$. Kako je $\hat{z} > 0$, tražićemo novu kolonu koju treba dodati u podproblem. Ova kolona, ako posmatramo problem (8.25), je data sa

$$V = \begin{pmatrix} c^T x^{(4)} \\ Ax^{(4)} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 4 \\ -1 \\ 1 \end{pmatrix} \quad (8.31)$$

Kada pogledamo prethodne vrijednosti kolone V i master problem (8.25), jasno je da ona odgovara koeficijentima promjenljive λ_4 . Dakle, prethodna kolona je data u odnosu na promjenljivu λ_4 početnog problema, ali je sada izražavamo preko trenutne baze (podproblema) vektora s_1, λ_2 i λ_3 . Nakon množenja kolone sa inverzom baze $B^{-1} = \begin{pmatrix} 1 & \frac{2}{5} & -\frac{9}{5} \\ 0 & -\frac{1}{5} & \frac{3}{5} \\ 0 & -\frac{1}{5} & \frac{2}{5} \end{pmatrix}$ i na osnovu kora-ka opisanih u Tabeli 8.1, dobijamo da se sljedeća kolona dodaje u simpleks tabelu

$$\begin{pmatrix} -\frac{3}{5} \\ \frac{9}{5} \\ \frac{3}{5} \\ \frac{2}{5} \\ \frac{3}{5} \\ \frac{2}{5} \end{pmatrix}$$

Dakle, imamo simpleks tabelu

s_1	1	$\frac{2}{5}$	$-\frac{9}{5}$	$\frac{9}{5}$	$\frac{9}{20}$
λ_2	0	$\frac{1}{5}$	$\frac{3}{5}$	$\frac{2}{5}$	$\frac{3}{5}$
λ_3	0	$-\frac{1}{5}$	$\frac{2}{5}$	$\frac{3}{5}$	$\frac{2}{5}$
z	0	$\frac{1}{5}$	$\frac{3}{5}$	$-\frac{3}{5}$	$\frac{3}{5}$

Nadimo sada pivota u dodanoj koloni. Kako je $\frac{9}{20} < \frac{2}{5} < \frac{3}{5}$, slijedi da je pivot element $\frac{9}{5}$. Sada pivotiramo oko tog elementa, tako da se kolona transformiše elementarnim transformacijama po vrstama na kolonu sa svim nulama osim na poziciji pivota, gdje treba da stoji jedinica. Na taj način, simpleks tabela se transformiše u novu simpleks tabelu

λ_4	$\frac{5}{9}$	$\frac{2}{9}$	-1	1	$\frac{1}{4}$
λ_2	$-\frac{2}{9}$	$\frac{1}{9}$	1	0	$\frac{1}{2}$
λ_3	$-\frac{1}{3}$	$-\frac{1}{3}$	1	0	$\frac{1}{4}$
z	$\frac{1}{3}$	$\frac{1}{3}$	0	0	$\frac{3}{4}$

Iz prethodne tabele pročitamo vrijednost dualnih promjenljivih: $y_1 = \frac{1}{3} = y_2$, dok je $\alpha = 0$.

Iteracija 4. Na osnovu vrijednosti dualnih promjenljivih, formira se novi CG podproblem

$$\max_{x \in X} (1, 0)^T x - \left(\frac{1}{3}, \frac{1}{3} \right) \begin{pmatrix} 1 & 3 \\ 2 & -3 \end{pmatrix} x - 0 = \max_{x \in X} (x_1 - x_2) = 0$$

Iz ovoga zaključujemo da je $\hat{z} = 0$, pa iz (8.19) slijedi da imamo optimum za početni problem, gdje je optimalna tačka data sa

$$\begin{aligned}x^* &= \frac{1}{2}x^{(2)} + \frac{1}{4}x^{(3)} + \frac{1}{4}x^{(4)} = \frac{1}{2}\begin{pmatrix}1 \\ 0\end{pmatrix} + \frac{1}{4}\begin{pmatrix}0 \\ 1\end{pmatrix} + \frac{1}{4}\begin{pmatrix}1 \\ 1\end{pmatrix} \\&= \begin{pmatrix}\frac{1}{2} + 0 + \frac{1}{4} \\ 0 + \frac{1}{4} + \frac{1}{4}\end{pmatrix} = \begin{pmatrix}\frac{3}{4} \\ \frac{1}{2}\end{pmatrix}\end{aligned}$$

8.3 Zadaci

1. Uz pomoć Benderove dekompozicije riješiti sljedeći MILP problem

$$\begin{aligned}\min x + y \\ \text{t.d.} \\ 2x + y \geq 3 \\ x \geq 0 \\ y \in \{-5, -4, \dots, 3, 4\}.\end{aligned}$$

2. Uz pomoć Benderove dekompozicije riješiti sljedeći MILP problem

$$\begin{aligned}\max 8y_1 + 9y_2 + 5y_3 + 6y_4 - 15x_1 - 10x_2 \\ \text{t.d.} \\ y_1 + y_3 \leq 1 \\ y_1 + y_4 \leq 1 \\ y_2 + y_4 \leq 1 \\ -x_1 - x_2 \leq -1 \\ y_1 - x_1 \leq 0 \\ y_2 - x_1 \leq 0 \\ y_3 - x_2 \leq 0 \\ y_4 - x_2 \leq 0 \\ y_1, y_2, y_3, y_4, x_1, x_2 \geq 0 \\ x_1, x_2 \in \mathbb{Z}.\end{aligned}$$

3. Uz pomoć DW dekompozicije riješiti sljedeći MILP problem

$$\begin{aligned} & \min x_1 - 3x_2 \\ & \text{t.d.} \\ & -x_1 + 2x_2 \leq 6 \\ & x_1 + x_2 \leq 5 \\ & x_1, x_2 \geq 0. \end{aligned}$$

4. Uz pomoć DW dekompozicije riješiti sljedeći MILP problem

$$\begin{aligned} & \min -4x_1 - x_2 - 6x_3 \\ & \text{t.d.} \\ & 3x_1 + 2x_2 + 4x_3 = 17 \\ & x \in X = \begin{cases} 1 \leq x_1 \leq 2 \\ 1 \leq x_2 \\ 1 \leq x_3 \leq 2. \end{cases} \end{aligned}$$

5. Proučiti opšti slučaj za DW dekompoziciju kada poliedar koji odgovara problemu nije ograničen (u njemu postoji ekstremni zrak). Nakon toga, uz pomoć DW dekompozicije riješiti sljedeći MILP problem

$$\begin{aligned} & \min x_1 - x_2 - 2x_3 \\ & \text{t.d.} \\ & x_1 + x_2 + x_3 = 3 \\ & x \in X = \begin{cases} 0 \leq x_1 \leq 2 \\ 0 \leq x_2 \\ 0 \leq x_3 \leq 2. \end{cases} \end{aligned}$$

Glava 9

Metode za pronalaženje dopustivih rješenja

U prethodnim poglavljima prikazane su najvažnije tehnike za egzaktno rješavanje problema iz oblasti linearнog i cjelobrojnog programiranja. Sa druge strane, u teoriji i praksi se često javljaju i problemi koji se, zbog svoje složenosti i dimenzionalnosti, ne mogu optimalno riješiti uz razuman utrošak memorijskih i vremenskih resursa. Kao što je već i ranije napominjano, u takvim situacijama se postavlja drugačiji cilj u istraživanju, odnosno, traži se dovoljno dobro *dopustivo rješenje* u sklopu resursa sa kojima raspolažemo. Pod dopustivim rješenjem podrazumijevamo ono rješenje koje zadovoljava uslove samog problema, ali ne postoji garancija da je to rješenje i optimalno.

U ovom poglavlju ćemo objasniti neke od tehnika za pronalaženje dopustivih rješenja. Čak i više od toga, prikazaćemo i neke od moćnih optimizacionih tehnika, koje ne garantuju pronalazak optimalnog rješenja, ali su se u praksi pokazale kao izuzetno koristan alat za rješavanje problema iz oblasti operacionih istraživanja.

9.1 Pohlepni algoritmi

Pohlepni algoritmi su programska paradigma zasnovana na pristupu da se rješenje iterativno formira dio po dio (komponenta po komponenta) tako da se naredna komponenta, koja se dodaje u postojeće rješenje, bira po nekom pohlepnom kriterijumu. Preciznije, u svakoj iteraciji se u rješenje uključuje ona komponenta koja donosi najveću korist u odnosu na sve ostale komponente, koje se razmatraju za proširenje trenutnog parcijalnog rješenja, u nadi da će takvi izbori na kraju dovesti i do kvalitetnog cjelokupnog (komplettnog) rješenja.

Osnovna strategija pohlepnih algoritama isključuje „vraćanje“ na neko prethodno (parcijalno) rješenje, odnosno ne vodi se računa o „lošim“ odlukama, koje su eventualno napravljene u prethodnim iteracijama. Zbog toga se podrazumijeva da pohlepni algoritmi imaju polinomijalnu vremensku složnost, što je jedna od najvažnijih osobina ovakvih algoritama.

Zbog relativno jednostavne strategije na kojoj su zasnovani, ali i brzine izvršenja, ovi algoritmi se često koriste i u rješavanju teških optimizacionih problema i najčešće su prva strategija koja se primjenjuje, prije upotrebe neke složenije paradigmе. U nekim situacijama se može pokazati da se optimalno rješenje uvjek može dobiti primjenom pohlepnog algoritma. Primjer takvog problema je *razlomljeni problem jednodimenzionalnog ruksaka*, a u nastavku će biti prikazani još neki važni problemi koji se optimalno rješavaju upravo paradigmom pohlepnih algoritama.

Sa druge strane, većina optimizacionih problema se ne može optimalno riješiti pomoću pohlepnih algoritama. U tim slučajevima, ova paradigma se koristi da bi se u kratkom vremenskom periodu obezbijedila dobra približna rješenja, ili da se konstruiše neko polazno dopustivo rješenje, koje će se kasnije unapredijevati naprednjim i složenijim algoritamskim tehnikama.

U ovoj sekciji razmatramo nekoliko poznatih pohlepnih algoritama:

- *Primov algoritam* za pronalaženje minimalnog pokrivajućeg stabla grafa (eng. minimum spanning tree – MST);
- *Kruskalov algoritam* za pronalaženje MST;

-
- *Dajkstrin algoritam* za pronalaženje najkraćih puteva u grafu;
 - Pohlepni algoritam za rješavanje problema *najdužeg zajedničkog podniza*, za proizvoljan skup stringova u ulazu.

9.1.1 Algoritmi za pronalaženje MST

Definišimo prvo pojam pokrivajućeg stabla, pa potom i minimalnog pokrivajućeg stabla.

Neka je dat težinski graf $G = (V, E)$. Svaku težinu grane $(u, v) \in E$ u grafu G , označimo sa $w(uv)$.

Definicija 9.1. Stablo $T = (V_1, E_1)$ je pokrivajuće za graf $G = (V, E)$ akko $V_1 = V$ i $E_1 \subseteq E$. Pokrivajuće stablo T grafa G sa minimalnim zbirom težina svojih grana, tj. $\sum_{e \in E_1} w(e)$, se naziva minimalno pokrivajuće stablo (MST) grafa G .

MST ima primjene u različitim oblastima teorije i prakse. Izdvajamo neke od njih.

- (i) Dizajniranje mreža (npr. telefonske mreže), gdje je do svakog domaćinstva, koje je predstavljeno čvorom grafa, potrebno obezbijediti liniju, tako da se, na kraju, u mrežu povežu sva domaćinstva, uz minimalnu potrošnju materijala potrebnog za kreiranje infrastrukture mreže;
- (ii) Aproksimacija drugih teških optimizacionih problema, kao što je problem trgovackog putnika (TSP), gdje je potrebno naći najkraću kantu koja posjećuje sve čvorove tačno jednom u (težinskom) grafu. Primijetimo da su putevi generisani iz ovakvih kontura (brisanjem jedne od grana konture) takođe MST. Težina ovako dobijenog rješenja MST je manja nego težina rješenja TSP posmatranog grafa, jer je riječ o minimizaciji nad većim skupom rješenja. Na ovaj način se može pokazati da nalazak MST-a u težinskom grafu aproksimira nalazak TSP-a (nad euklidskim težinskim grafovima).

-
- (iii) Klaster analiza, konkretno za k -klaster problem, se može posmatrati kao problem nalaženja MST-a, a potom brisanja $k - 1$ najskupljih grana iz MST rješenja.

Primov algoritam

Primov algoritam je jedan od osnovnih algoritama koji se, zbog svoje važnosti, ali i jednostavnosti, izučava u teoriji algoritama. Navećemo ključne korake algoritma, bez detaljnije analize validnosti algoritma. Algoritam se sastoji od sljedećih koraka:

1. Kreirati skup $mstSkup = \emptyset$ da bi se pratili svi čvorovi koji su već uključeni u MST.
2. Dodijeliti vrijednost svakom čvoru u ulaznom grafu: inicijalizovati vrijednosti na $+\infty$. Dodijeli vrijednost 0 za čvor koji se prvi ubacuje u MST. Ideja dodjele vrijednosti čvorovima koji još nisu prisutni u skupu MST je u efikasnom odabiru grane minimalne težine, koja treba da se doda u trenutno parcijalno rješenje.
3. Dok god $mstSkup$ ne uključi sve čvorove iz V , ponavljamo sljedeće korake:
 - Izabrati čvor $u \notin mstSkup$ sa minimalnom vrijednošću (pohlepni korak);
 - $mstSkup = mstSkup \cup \{u\}$ (proširivanje parcijalnog rješenja);
 - Ažurirati vrijednosti svih susjeda čvora u na sljedeći način: Za svaki susjed v , ako je težina grane uv manja od trenutne vrijednosti za v , dodijeliti čvoru v vrijednost $w(uv)$.

Vremenska složenost. Ako se za predstavljanje grafa koristi lista susjedstva, složenost algoritma je $O(|V|^2)$. Ako koristimo prioritetni red za nalaženje grane minimalne težine, dobijamo složenost od $O(|E| \log(|V|))$.

Kruskalov algoritam

Kruskalov algoritam radi po principu izbora grana na osnovu njihove težine, gdje se u svakom koraku koristi pohlepna strategija da se bira odgovarajuća grana najmanje težine. Koraci algoritma su sljedeći.

1. Sortirati sve grane polaznog grafa u neopadajućem poretku; neka je tako formirana lista S svih grana u grafu.
2. Postaviti $mstSkup = \emptyset$.
3. Odabratи najlakšу granu iz skupa S (pohlepni korak). Provjeriti da li se formira ciklus pri dodavanju grane u skup grana koje su do sada ubačene u skup $mstSkup$; Ako to nije slučaj, ubaciti granu u trenutni skup $mstSkup$.
4. Iz liste S ukloniti odabranu granu;
5. Ponoviti korake 3 i 4, redom, dok se ne doda tačno $|V| - 1$ grana u $mstSkup$.

Vremenska složenost. Kruskalov algoritam se izvršava u $O(|E| \log |E|)$ ili $O(|E| \log |V|)$ vremenu. Sortiranje grana uzima $O(|E| \log |E|)$ vremena. Potom, za svaku granu se primjenjuje operacija nalaženja i dodavanja grane u skup mst . Ove dvije operacije se izvršavaju u $O(\log |V|)$ vremenu. Dakle, ukupno vrijeme izvršavanja algoritma je $O(|E| \log |E| + |E| \log |V|)$. Kako je $|E| = O(|V|^2)$, dobijamo složenost $O(|E| \log(|V|))$.

9.1.2 Dajkstrin algoritam

Ovaj algoritam se koristi za pronalaženje najkraćih puteva u grafu i sličan je Primovom algoritmu. Generiše se tzv. SPT (eng. *shortest path tree*) sa datim korjenim čvorom s , tako što se formiraju dva disjunktna skupa čvorova, jedan koji sadrži čvorove uključene u SPT, dok su u drugom svi ostali čvorovi.

Inicijalno, svim čvorovima, osim korjenog, je dodijeljena vrijednost beskonačno, tj. $dist[u] = +\infty, \forall u \in V \setminus \{s\}$, $dist[s] = 0$.

Neka je $Q = V$. U svakom koraku algoritma nalazimo čvor u koji je u skupu Q sa minimalnom udaljenošću $dist$ od korjenog čvora s . Zatim se posmatraju njegovi susjedi, te težine grana koje polaze od čvora u . U slučaju da je zbir vrijednosti vrijednosti $dist[u]$ i težine grane $w(u, v)$ manja od trenutne vrijednosti $dist[v]$, vrijednost $dist[v]$ se ažurira, jer je nađen novi najkraći put od s od v . Čvor u se zatim izbacuje iz skupa Q . Algoritam nastavlja sa istim koracima dok god $Q \neq \emptyset$.

Svi koraci algoritma su dati u Algoritmu 3. Nakon završetka algoritma, vrijednosti $dist[u]$ odgovaraju težini najkraćeg puta od korjenog čvora s do čvora u .

Algoritam 3 Dajkstra(G, s)

```

1: for svaki čvor  $v \in V$ : do
2:    $dist[v] \leftarrow +\infty$ 
3:    $previous[v] \leftarrow null$ 
4: end for
5:  $dist[s] \leftarrow 0$ 
6:  $Q \leftarrow V(G) = V$ 
7: while  $Q \neq \emptyset$  do
8:    $u \leftarrow$  čvor iz  $Q$  sa najmanjom vrijednošću  $dist$ 
9:    $Q \leftarrow Q \setminus \{u\}$ 
10:  for sve susjede  $v$  od  $u$  do
11:     $temp \leftarrow dist[u] + w(u, v)$ 
12:    if  $temp < dist[v]$  then
13:       $dist[v] \leftarrow temp$ 
14:       $previous[v] \leftarrow u$ 
15:    end if
16:  end for
17: end while
18: return  $previous$ 
```

Vremenska složenost. Implementacija Algoritma 3 ima vremensku složenost $O(|V|^2)$. Ako je graf predstavljen listom susjedstva, i skup Q je implementiran kao binarni heap, složenost se redukuje na $O(|E| \cdot \log |V|)$.

Napomenimo da Dajkstrin algoritam ne radi sa grafovima koji imaju negativne težine (pogledati liniju 12). Za ovakve grafove, primjenjuje se Bellman–Ford-ov algoritam koji je baziran na dinamičkom programiranju. Njega ne pominjemo u ovoj knjizi, ali čitaocu preporučujemo da se, koristeći literaturu ili internet, upozna i sa ovim važnim algoritmom.

9.1.3 Razlomljeni problem jednodimenzionalnog ruksaka

Definicija 9.2. *Neka je dato n proizvoda sa svojim težinama i vrijednostima, te ruksak čiji je kapacitet $W > 0$. Razlomljeni problem jednodimenzionalnog ruksaka podrazumijeva odabir proizvoljnih dijelova proizvoda koji se stavljuju u ruksak, tako da vrijednost proizvoda u ruksaku bude što veća, ali da se pri tom ne naruši ograničenje vezano za ukupan kapacitet ruksaka.*

U 0-1 problemu ruksaka, koji će biti razmatran i u Primjeru 9.4, ne dopušta se dijeljenje proizvoda. Dakle, ili se uzima cijeli proizvod (1) i stavlja u ruksak, ili se uopšte ne uzima (0). U razlomljenom problemu ruksaka, svaki proizvod pored svoje težine i vrijednosti, može da se podijeliti na proizvoljne dijelove (recimo dopušteno je da se uzme pola od ukupne količine proizvoda i ubaci u ruksak). Primjeri takvih proizvoda bi bili mlijeko, so, šećer, itd. Efikasno rješavanje ovog problema se izvodi primjenom pohlepnog algoritma na sljedeći način.

1. Računaju se odnosi vrijednost/težina za svaki proizvod, te se proizvodi sortiraju na osnovu tih vrijednosti, u opadajućem poretku.
2. Sve dok ruksak nije potpuno napunjeno ponavljamo sljedeći postupak:
 - Iz liste sortiranih proizvoda, uzmemmo onaj proizvod kod koga je izračunati odnos najveći;
 - Dodamo ga u ruksak u maksimalnoj količini, vodeći računa da ne narušimo kapacitet ruksaka.
 - Uklonimo taj proizvod iz liste.

Može se pokazati sljedeća teorema.

Teorema 9.1. Prethodno opisan pohlepni algoritam za razlomljeni problem jednodimenzionalnog ruksaka uvijek nalazi optimalno rješenje.

9.1.4 Pohlepni pristup za rješavanje LCS problema

String definišemo kao niz karaktera nad konačnom abzukom Σ .

Definicija 9.3. Problem najdužeg podniza (eng. Longest common subsequence – LCS), je problem koji u ulazu uzima n stringova proizvoljne dužine. Cilj je da se pronade string maksimalne dužine koji je zajednički podniz za sve stringove iz ulaznog skupa stringova.

Primjer 9.1. Neka su u ulazu data tri stringa dužine šest:

$$S = \{\text{abbccb}, \text{abccab}, \text{abbbcb}\}.$$

Rješenje problema je $s = \text{abcb}$. Primijetimo da je s podniz za sva tri stringa iz S , a može se (direktno) provjeriti da je to najduži takav string.

Ovaj problem se može riješiti u $O(n^m)$ vremenu uz pomoć dinamičkog programiranja, gdje je n dužina najdužeg stringa u ulazu, a m broj stringova u ulazu. Specijalno, za $m = 2$, problem je rješiv u $O(n^2)$ vremenu. Za proizvoljno m i n , ovaj problem je NP-težak.

Jedan od najjednostavnijih pristupa za rješavanje ovog problema je *pohlepni pristup*, tzv. *Best-Next* (BN) heuristika. Prije nego što navedemo detalje algoritma, uvedimo nekoliko definicija i oznaka koje će biti korištene u konstrukciji ovog algoritma.

Za cjelobrojni vektor $p^L = (p_1^L, \dots, p_m^L)$, definišemo kolekciju stringova $S[p^L] = \{s_i[p_i^L, |s_i|] \mid i = 1, \dots, m\}$, gdje $|s|$ označava dužinu stringa s , a $s[x, y]$ podstring stringa s koji počinje od karaktera na poziciji x , a završava sa karakterom na poziciji y . Konvencija je da se indeksi u stringu numerišu krenuvši od 1. Dakle, $s = [1, |s|]$ za svaki string s . Primjera radi, za (pozicioni) vektor $p^L = (2, 3, 4)$ i skup S iz Primjera 9.1 imamo $S[p^L] = \{s_1[1, |s_1|], s_2[2, |s_2|], s_3[3, |s_3|]\} = \{\text{bbccb}, \text{ccab}, \text{bcb}\}$.

Dalje, sa $p_{i,a}^L$ definišemo poziciju prvog pojavljivanja karaktera $a \in \Sigma$ takav da je $p_{i,a}^L \geq p_i^L$ (ukoliko karakter ne postoji, dodijelimo vrijednost

$n + 1$). Skup karaktera Σ_{p^L} koji se pojavljuju u svakom od stringova (u podproblemu) $S[p^L]$ su dopustive ekstenzije u odnosu na pozicioni vektor p^L .

U konstrukciji pohlepnog algoritma, odredimo skup dopustivih komponenti i (pohlepni) kriterijum odabira najbolje komponente u svakom koraku algoritma koja se dodaje u postojeće parcijalno rješenje. Inicijalno rješenje s je prazan string, koje je, trivijalno, dopustivo za svaku instancu LCS problema. Ideja se sastoji u iterativnom proširivanju trenutnog rješenja za jedan karakter, spajajući taj karakter sa rješenjem sa desna, dok god je to moguće. Pri tome prošireno rješenje mora uvjek biti zajednički podniz, dakle, dopustivo. Za spajanje stringa i karaktera ćemo koristiti standardni operator (\cdot) .

Prema navedenom, krenemo sa pozicijom $p^L = (1, \dots, 1)$ koja odgovara čitavom skupu $S = S[p^L]$ i praznim stringom $s = \varepsilon$ kao inicijalnim rješenjem. Dalje, da bismo očuvali dopustivost rješenja kada se rješenje proširi za jedan karakter, za komponente rješenja Σ_{p^L} ćemo uzeti sve karaktere koji se nalaze u svim stringovima iz $S = S[p^L]$, dakle u sufiks stringovima u odnosu na odgovarajuće pozicije vektora p^L . Na taj način, ekstenzija $s = s \cdot x$ će za bilo koji $x \in \Sigma_{p^L}$ dati dopustivo LCS rješenje. Među karakterima iz Σ_{p^L} treba odabrati (lokalno) najboljeg kandidata za proširivanje rješenja. Treba imati na umu da se kandidat $x \in \Sigma_{p^L}$ može pojaviti na nekoliko pozicija u svakom stringu iz $S = S[p^L]$. Kako ne želimo ekstenzije koje vode ka podoptimalnim rješenjima, za karakter x biramo onaj sa pozicijama pojavljivanja koje su najbliže početku svakog od stringova iz $S = S[p^L]$, dakle na pozicijama $p_{i,a}^L, i = 1, \dots, m$.

Ideja konstrukcije pohlepne funkcije $g()$ se sastoji u odabiru onog karaktera za proširivanje rješenja s koji, na neki način, izbacuje najmanji broj karaktera iz dalje pretrage. U našem slučaju, funkcija $g()$ procjenjuje relatičan broj izbačenih karaktera u odnosu na veličinu sufiks stringova. Sljedeći korak se sastoji u ažuriranju prostora pretrage za novi korak. Kako smo proširili rješenje za x , nove komponente rješenja koje su dopustive ekstenzije trenutnog rješenja tražimo u skupu (sufiks) stringova $S[p_{1,x}^L + 1, \dots, p_{m,x}^L + 1]$. Postupak nastavljamo dok god postoje komponente koje mogu proširiti tre-

nutno rješenje s .

Da rezimiramo, pohlepni (BN) pristup za rješavanje LCS problema se sastoji od sljedećih koraka.

- Polazi se od parcijalnog rješenja koje je prazna niska, $s = \varepsilon$;
- Iniciramo pozicioni vektor $p^L \leftarrow (1, \dots, 1)$;
- Dok god jedan od p_i^L nije jednak $n + 1$, ponavljamo sljedeće korake:
 - $\Sigma_{p^L} \leftarrow$ naći dopustive ekstenzije u $S[p^L]$;
 - Od svih karaktera u Σ_{p^L} , odaberemo onaj koji minimizuje vrijednost funkcije

$$g(p^L, a) = \sum_{i=1}^m \frac{p_{i,a}^L - p_i^L + 1}{|s_i| - p_i^L + 1}, a \in \Sigma_{p^L}.$$

Označimo taj karakter sa a^* .

- Ažuriramo: $s \leftarrow s \cdot a^*$, $p_i^L \leftarrow p_{i,a^*}^L + 1$, $i = 1, \dots, m$.

Algoritam vraća vrijednost s .

Primjetimo da ovakav algoritam ne može da garantuje optimalno rješenje, što je ilustrovano sljedećim jednostavnim primjerom.

Primjer 9.2. Neka su data dva stringa: $s_1 = \text{caba}$, $s_2 = \text{bacc}$. Prvo konstatujmo da su sva tri karaktera moguće opcije za proširenje (inicijalno praznog stringa ε). Izračunajmo za svakog od njih odgovarajući g -skor. Imamo: $g((1, 1), \text{a}) = \frac{2}{4} + \frac{2}{4} = 1$, $g((1, 1), \text{b}) = \frac{3}{4} + \frac{1}{4} = 1$, $g((1, 1), \text{c}) = \frac{1}{4} + \frac{3}{4} = 1$. Ovdje dolazi do situacije u kojoj, ukoliko ne postoji dodatni kriterij odlučivanja u slučaju jednakih vrijednosti minimalnog g -skora za više karaktera, algoritam nasumično donosi odluku o odabiru karaktera (u zavisnosti od internih koraka u algoritmu i korištenih struktura podataka). Ako bi izabrali c kao karakter koji proširuje trenutno (prazno) rješenje u prvom koraku algoritma, u narednom koraku bi algoritam prekinuo sa radom, jer sufiksi koji preostaju za proširenje rješenja c neće dati niti jedan karakter za naredno proširenje. Dakle, u tom slučaju algoritam vraća rješenje c , koje očigledno

nije optimalno. Ako bi izabrali recimo karakter **b** u prvom koraku, naredni (i jedini mogući) karakter koji proširuje trenutno rješenje je **a**, pa bi algoritam u tom slučaju vratio (optimalno) rješenje **ba**.

9.2 Aproksimativni algoritmi

Za veliki broj složenih optimizacionih problema u praksi ne postoje efikasni egzaktni algoritmi. Drugim riječima, da bi se takvi problemi egzaktno riješili, potrebno je izdvojiti ogromne vremenske i memorijске resurse. Sa druge strane, većina pohlepnih algoritama, ali i naprednijih heuristika, ne daju nikakvu garanciju kvaliteta nađenog rješenja. Negdje između egzaktnih i heurističkih algoritama su se pozicionirali *aproksimativni algoritmi*, koji su polinomijalne složenosti i pronalaze dopustiva rješenja uz garanciju njihovog kvaliteta.

Algoritam pripada klasi *aproksimativnih algoritma* ako:

1. se izvršava u polinomijalnom vremenu;
2. posjeduje garanciju da je pronađeno dopustivo rješenje približno optimalnom rješenju, odnosno garantuje se da je od optimalnog rješenja unutar nekog faktora kvaliteta (c), bez ikakvog prethodnog znanja o optimumu.

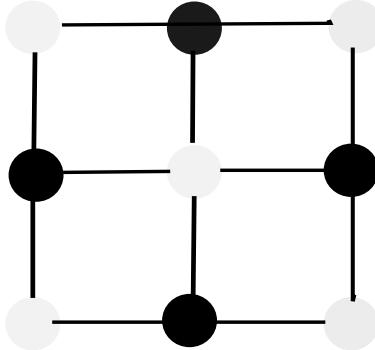
Neka je dat algoritam \mathcal{A} , problem P i instanca problema I . Sa $\mathcal{A}(I)$ označimo (dopustivo) rješenje koje se dobija kao izlaz algoritma \mathcal{A} , dok sa $\mathcal{A}^*(I)$ označimo optimalno rješenje za ulaznu instancu I . Prepostavimo da je riječ o problemu maksimizacije.

Algoritam \mathcal{A} se naziva *aproksimativni algoritam* problema P ako za bilo koji ulaz (instancu) I , \mathcal{A} vraća aproksimativno rješenje $\mathcal{A}(I)$ za $\mathcal{A}^*(I)$ u polinomijalnom vremenu.

Postoji dvije vrste aproksimativnih algoritama:

- *apsolutni aproksimativni algoritam* – ako je za svaku instancu problema P ispunjeno

$$|\mathcal{A}(I) - \mathcal{A}^*(I)| \leq c,$$



Slika 9.1: Primjer pokrivača čvorova grafa.

za neko $c \in \mathbb{R}^+$;

- *relativni aproksimativni algoritam* – ako je za svaku instancu problema P ispunjeno

$$\frac{\mathcal{A}(I)}{\mathcal{A}^*(I)} \geq c,$$

za neko $c \in (0, 1]$.

U tom slučaju kažemo da je algoritma \mathcal{A} c -aproksimativni algoritam.

U nastavku navodimo (relativne) aproksimativne algoritme za nekoliko poznatih problema kombinatorne optimizacije (eng. combinatorial optimization), podoblasti matematičke optimizacije, u kojoj se optimalno rješenje traži na diskretnom skupu rješenja.

9.2.1 Aproksimativni algoritmi za problem minimalnog pokrivanja čvorova

Definišimo problem minimalnog pokrivanja čvorova.

Definicija 9.4. Neka je dat graf $G = (V, E)$. Potrebno je naći podskup $C \subseteq V$ minimalne kardinalnosti, takav da barem jedan čvor svake grane pripada skupu C .

Skup C , koji zadovoljava uslove iz prethodne definicije se naziva *minimalni pokrivač* čvorova grafa G . Primjer minimalnog pokrivača jednog grafa je dat na Slici 9.1 gdje pokrivač uključuje sve čvorove crne boje.

Dopustivo rješenje za problem minimalnog pokrivanja čvorova je svaki podskup C' , ne obavezno minimalne kardinalnosti, koji sadrži barem jedan čvor svake grane datog grafa.

Posmatrajmo (pohlepni) algoritam za ovaj problem dat Algoritmom 4.

Algoritam 4 Prvi pohlepni algoritam za problem minimalnog pokrivanja čvorova

```
1:  $C \leftarrow \emptyset$ 
2: while  $E \neq \emptyset$  do
3:    $(u, v) \leftarrow$  bilo koja grana iz  $E$ 
4:    $C \leftarrow C \cup \{u, v\}$ 
5:    $E \leftarrow$  ukloniti sve grane incidentne sa  $u$  ili  $v$  iz grafa  $G$ 
6: end while
7: return  $C$ 
```

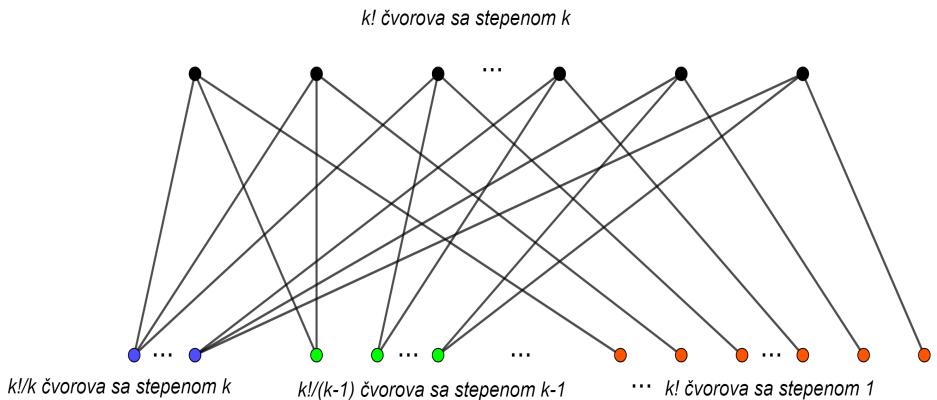
Pokazaćemo da je ovo aproksimativni algoritam. Izračunajmo koeficijent aproksimacije c za Algoritam 4. Grane koje se biraju u ovom algoritmu su grane maksimalnog uparivanja (M), pa je prema tome, skup C uistinu pokrivač čvorova, prema definiciji uparivanja u grafu. Dalje, jasno je da se algoritam izvodi u polinomijalnom vremenu. Neka je C^* pokrivač koji je rješenje problema. Tada C^* sadrži barem jedan kraj svake grane iz M , odakle slijedi $|C^*| \geq |M|$. Dalje, $|C| = 2 \cdot |M| \geq 2 \cdot |C^*|$. Prema tome, ovo je 2-aproksimativni algoritam za problem pokrivanja čvorova grafa. Jednakost u prethodnoj aproksimaciji se postiže, recimo, za bipartitne grafove $K_{n,n}$.

Posmatrajmo sada i drugi pohlepni Algoritam 5 za problem pokrivanja čvorova.

Algoritam 5 Drugi pohlepni algoritam za problem minimalnog pokrivanja čvorova

```
1:  $C \leftarrow \emptyset$ 
2: while  $E \neq \emptyset$  do
3:    $u \leftarrow \operatorname{argmin}_{u \in V(G)} \deg(u)$ 
4:    $C \leftarrow C \cup \{u\}$ 
5:    $E \leftarrow$  ukloniti sve grane incidentne sa  $u$  u  $G$ 
6: end while
7: return  $C$ 
```

Izračunajmo aproksimativni faktor Algoritma 5 posmatrajući graf sa Slike 9.2.



Slika 9.2: Specijalan bipartitan graf.

Optimalno rješenje je podskup čvorova C^* , koji se sastoji od svih čvorova sa vrha grafa (obojeni crnom bojom); ukupno ih ima $k!$. Međutim, prateći korake Algoritma 5, dobijamo

$$|C| = k! \cdot (1/k + 1/k - 1 + \cdots + 1/2 + 1) \approx k! \cdot \log k.$$

Dakle, imamo da je

$$|C| \approx \log k \cdot |C^*|.$$

Primijetimo da je u prethodnom zaključku korištena dobro poznata formula

$$\sum_{j=1}^k \frac{1}{j} = \log(k) + \gamma + O(1/k),$$

gdje je γ Ojler-Mašeronijeva konstanta ($0.577\dots$).

Prema tome, Algoritam 5 je $\log k$ -aproksimativni, $k \in \mathbb{N}$.

9.2.2 Aproksimativni algoritmi za problem trgovačkog putnika

Za problem trgovačkog putnika nije jednostavno konstruisati efikasan aproksimativni algoritam. Tome svjedoči sljedeća teorema, koja se odnosi na opšti slučaj ovog problema.

Teorema 9.2. *Ne postoji polinomijalan algoritam koji može aproksimirati TSP sa faktorom $c > 1$, osim ako je $P = NP$.*

Ipak, za neke specifične varijante TSP-a, postoje aproksimativni algoritmi, što ćemo vidjeti u nastavku ovog odjeljka.

Definicija 9.5. *Kažemo da grane grafa zadovoljavaju nejednakost trougla ako za sve različite čvorove u, v, r grafa G vrijedi*

$$w(u, v) + w(v, r) \geq w(u, r),$$

gdje su $w(\cdot, \cdot)$ težine grane koja spaja dva čvora.

Definicija 9.6. *TSP kod koga grane zadovoljavaju nejednakost trougla se naziva metrički TSP.*

Metrički TSP je i dalje NP-težak, ali se aproksimativni algoritam ipak može konstruisati.

Posmatrajmo *algoritam najблиžeg dodavanja* koji aproksimira TSP, dat Algoritmom 6.

Algoritam 6 Prvi aproksimativni algoritam za TSP.

```
1: Uzorak: Kompletan težinski graf  $G$ 
2: Izlaz: Hamiltonova kontura  $T$ 
3:  $u, v \leftarrow \operatorname{argmin}_{uv \in E} w(u, v)$ 
4:  $S \leftarrow \{u, v\}$ 
5:  $T \leftarrow uv$ 
6: while  $|S| \neq n$  do
7:    $(u, t) \leftarrow \operatorname{argmin}_{(u, t) \in (V(G) \setminus V(T)) \times V(T)} \{w(u, t)\}$ 
8:    $S \leftarrow S \cup \{u\}$ 
9:    $(u, q) \leftarrow$  čvor  $q$  slijedi  $u$  u  $T$ 
10:   $(u, q)$  se izbacuje iz  $T$ , ubacuje se  $(u, t), (t, q)$  // Ažuriranje  $T$  (rekonstrukcija)
11: end while
12: return  $T$ 
```

Ovo je očigledno polinomijalan algoritam. Odredimo aproksimacioni faktor koji se garantuje njegovim izvršavanjem.

Primjetimo da algoritam podsjeća na Primov algoritam. Grane koje se identificuju algoritmom konstruišu MST. Zaključujemo da je cijena konture T barem jednaka koliko i cijena MST u odnosu na iste ulazne podatke. Cijena konture za dva čvora v_i i v_j je $2 \cdot w(v_i, v_j)$ (jer računamo i povratnu granu). Posmatrajmo jednu iteraciju Algoritma 6, gdje se T proširuje za čvor t koji se ubacuje između čvorova u i q . Povećanje cijene (trenutne) konture T u svakoj iteraciji je

$$w(u, t) + w(t, q) - w(u, q).$$

Na osnovu nejednakosti trougla imamo $w(t, q) \leq w(t, u) + w(u, q)$. Prema tome, porast cijene konture T u svakoj iteraciji je najviše $w(t, u) + w(u, t) = 2 \cdot w(u, t)$. Dakle, završna kontura ima cijenu koja dostiže cijenu barem dva puta veću od cijene MST. Prema tome, ovo je 2-aproksimativni algoritam. Pokušajmo poboljšati aproksimativni faktor Algoritma 6 uz pomoć *algoritma dvostrukog drveta* (eng. double tree algoritma), čije detalje objašnjavamo u nastavku. Prije toga ćemo uvesti nekoliko definicija i teorema od bitnog značaja u konstrukciji algoritma.

Definicija 9.7. Neka je dat graf $G = (V, E)$. Ojlerova kontura T^o grafa G je kontura e_1e_2, \dots, e_me_1 koja obilazi sve grane grafa G .

Vrijedi sljedeća karakterizacija grafova koji sadrže Ojlerovu konturu.

Teorema 9.3. Graf G sadrži Ojlerovu konturu akko je povezan i stepen svakog njegovog čvora paran broj. Takva klasa grafova se naziva Ojlerovom klasiom grafova.

Posmatrajmo sada Algoritmom 7 i pokažimo da ovim algoritmom dobijamo 2-aproksimaciju metričkog TSP-a.

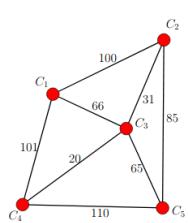
Algoritam 7 Drugi aproksimativni algoritam za TSP.

- 1: **Ulaz:** Kompletan graf G
 - 2: **Izlaz:** Hamiltonova kontura T
 - 3: $mst \leftarrow$ Izračunati MST grafa G
 - 4: $mst^{copy} \leftarrow$ Zamijeniti svaku granu u mst sa dvije kopije istog
 - 5: $T^o \leftarrow$ Naći Ojlerovu konturu T u mst^{copy}
 - 6: $T \leftarrow$ Izbaciti duple čvorove iz T^o
 - 7: **return** T
-

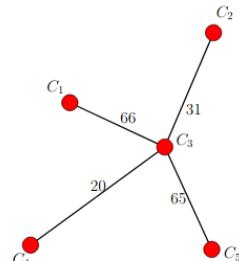
Jasno je da je algoritam polinomijalan i da je dobijeno rješenje dopustivo.

Neka je C^* cijena optimalne Hamiltonove konture. Tada je cijena MST $\leq C^*$. Lako se vidi da je cijena Ojlerove konture manja ili jednaka $2 \cdot C^*$. Na osnovu nejednakosti trougla, skraćivanje Ojlerove konture (linija 6 u

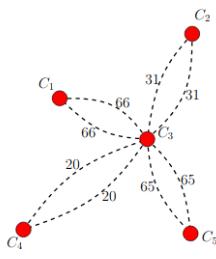
algoritmu) u svrhu dobijanja Hamiltonove konture T , ne može da bude veća od $2C^*$. Dakle, slijedi da je cijena Hamiltonove konture manja ili jednaka $2C^*$, pa je ovaj algoritam 2-aproksimativni.



(a) Graf G .

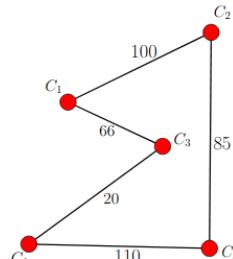


(b) MST grafa G



(c) Ojlerova
 $C_1C_3C_4C_3C_5C_3C_2C_3C_1$.

kontura: (d) Hamiltonova kontura (ciklus):
 $C_1C_3C_4C_5C_2C_1$.



Slika 9.3: Koraci u konstrukciji Kristofidjesovog algoritma.

Posmatrajmo još jedan aproksimativni algoritam za TSP koji garantuje aproksimativni faktor od 1.5. Algoritam je poznat pod nazivom *Kristofidesov* algoritam. Koraci su prikazani Algoritmom 8.

Algoritam 8 Kristofidesov aproksimativni algoritam za TSP.

- 1: **Ulaz:** Graf G
 - 2: **Izlaz:** Hamiltonova kontura T
 - 3: $T \leftarrow$ Naći MST grafa G
 - 4: $M \leftarrow$ Naći minimalno uparivanje svih čvorova u T neparnog stepena
 - 5: $T' \leftarrow$ Dodamo grane od M u T
 - 6: $T^o \leftarrow$ Nađimo Ojlerovu konturu u T'
 - 7: $T \leftarrow$ Izbaciti duple čvorove iz T^o
 - 8: **return** T
-

Lako je vidjeti da je ovaj algoritam polinomijalan i da je dobijeno rješenje dopustivo.

Teorema 9.4. *Kristofidesov algoritam ima faktor aproksimacije $c = 1.5$.*

Dokaz. Cijena TSP C^* dostiže najviše cijenu Ojlerove konture, tj. $C^* \leq \text{cost}(T^o) = \text{cost}(T) + \text{cost}(M)$. Takođe jasno je da vrijedi $\text{cost}(T) \leq C^*$. Pokažimo da je $\text{cost}(M) \leq \frac{C^*}{2}$, što je dovoljno za dokaz teoreme. Prijmetimo da u optimalnoj konturi T postoje dva uparivanja: prvi sa čvorovima iz M , a drugi za ostale čvorove kojeg označavamo sa M' . Kako je $C^* \geq \text{cost}(M) \leq \text{cost}(M')$ i $\text{cost}(M) + \text{cost}(M') = \text{cost}(T)$, slijedi $2\text{cost}(M) \leq \text{cost}(T) \leq C^*$, odakle dobijamo tvrđenje. \square

U literaturi je pokazano da ne postoji aproksimacioni algoritam sa faktorom $c < \frac{220}{219} \approx 1.0045$ (osim ako je $P = NP$).

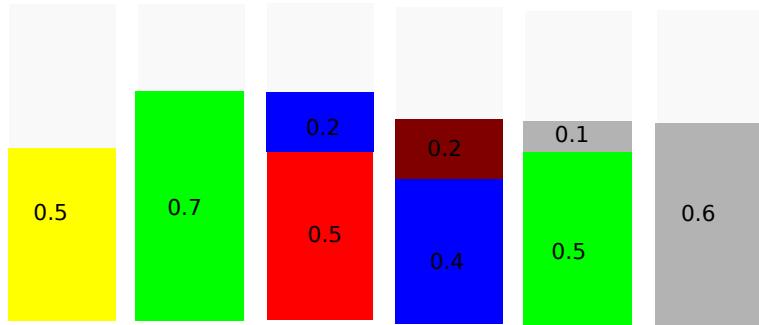
9.2.3 Aproksimativni algoritmi za problem binarnog pakovanja

Definišimo *problem binarnog pakovanja*.

Definicija 9.8. *Neka je dato n proizvoda veličine a_1, a_2, \dots, a_n ($0 < a_i \leq 1$). Potrebno je spakovati sve proizvode u pakovanja jediničnog kapaciteta, tako da je ukupan broj korištenih pakovanja minimalan.*

Primjer jedne instance problema pakovanja je dat na Slici 9.4.

0.5, 0.7, 0.2, 0.4, 0.2, 0.1, 0.6



Slika 9.4: Primjer binarnog pakovanja.

Navedimo *prvi odgovarajući* (eng. first-fit) pohlepni algoritam, koji se sastoji od sljedećih koraka.

- Raspoređujemo proizvode u onom redu u kojem su poredani za pakovanje, tj. a_i se pakuje prije a_j akko $i < j$;
- Stavljamo sljedeći proizvod u pakovanje sa najmanjim indeksom, u kojem je ostalo dovoljno prostora za taj proizvod.
- Ako proizvod ne može da se ubaci ni u jedno otvoreno pakovanje, uzimamo novo pakovanje i tu smještamo taj proizvod.

Lako se vidi da je ovaj algoritam polinimijalne složenosti i da kao rezultat vraća dopustivo rješenje. Neka je C^* optimalan broj pakovanja koji se treba otvoriti da bi se spakovalo svih n stavki. Pretpostavimo da algoritam vraća rješenje C (pakovanja). Sigurno vrijedi da je barem u $C - 1$ pakovanja iskorišteno više od polovine prostora. Ako to ne bi bio slučaj, onda bi postojala dva pakovanja u kojima je najviše pola prostora iskorišteno. U tom slučaju, stavke u oba pakovanja bi mogle da idu u jedno od njih, dok bi se

drugo oslobođilo. Prema tome, imamo

$$C^* \geq \sum_{i=1}^n a_i > \frac{C-1}{2} \Rightarrow 2 \cdot C^* > C - 1 \Rightarrow 2 \cdot C^* \geq C,$$

pa slijedi da je algoritam prvi-odgovarajući 2-aproksimacioni.

Pomenimo sada i tzv. *najbolji-odgovarajući* (eng. best-fit) algoritam, koji radi po sljedećem principu. Sljedeći proizvod se stavlja u ono pakovanje u kojem ostaje najmanji prazan prostor, u odnosu na sva ostala otvorena pakovanja. Ako takvo pakovanje ne postoji, otvara se novo pakovanje. Ovaj algoritam je takođe 2-aproksimacioni.

Interesantno je napomenuti da za problem binarnog pakovanja ne postoji aproksimativni algoritam čiji je faktor $c < \frac{3}{2}$ (osim ako je $P = NP$).

9.2.4 Aproksimativni algoritmi za problem pokrivanja skupova

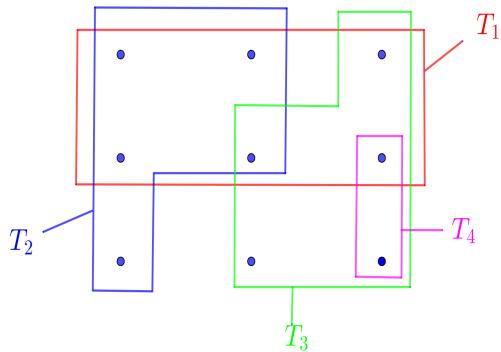
Problem pokrivanja skupova (eng. set cover problem) predstavlja uopštene probleme pokrivanja čvorova, i definisan je na sljedeći način.

Definicija 9.9. Neka je dat skup elemenata X te familija podskupova \mathcal{F} skupa X takva da je za svaki $f \in \mathcal{F}$, $|f \cap X| \geq 1$ i $\bigcup_{f \in \mathcal{F}} f = X$. Potrebno je naći pokrivač $C \subseteq \mathcal{F}$ minimalne kardinalnosti takav da je

$$\bigcup_{f \in C} f = X.$$

Motivacija za razmatranje ovog problema se može pronaći i u svakodnevnom životu. Npr. prepostavimo da imamo skup sposobnosti (kompetencija), te skup ljudi koji posjeduju pojedine kompetencije. Potrebno je formirati komisiju sa što manjim brojem osoba, ali tako da se za svaku kompetenciju može naći osoba iz komisije sa tom kompetencijom.

U nastavku ćemo objasniti pohlepni algoritam koji nema konstantan aproksimacioni faktor, već logaritamski zavisi od veličine ulaza. To znači da, kako veličina ulaza raste, kvalitet rješenja algoritma opada relativno u



Slika 9.5: Primjer pokrivanja skupa ($\mathcal{F} = \{T_1, T_2, \dots, T_4\}$, optimalno rješenje je skup $\mathcal{C} = \{T_2, T_3\}$)

odnosu na kvalitet optimalnog rješenja. Ipak, ovaj aproksimacioni algoritam daje dobre rezultate u praktičnim primjenama, pošto logaritamska funkcija sporo raste.

Pohlepni metod za problem pokrivanja skupa je prikazan Algoritmom 9. Skup U predstavlja skup preostalih nepokrivenih elemenata, dok skup \mathcal{C} predstavlja pokrivač koji se konstruiše kroz iteracije algoritma. Incijalno, $U = X$, te pokrivač $C = \emptyset$. U svakom koraku, algoritam nastoji da proširi pokrivač C na račun smanjenja skupa U . Kandidata za proširivanje pokrivača biramo na osnovu pohlepne funkcije date u liniji 6. U osnovi, biramo onaj podskup $f \in \mathcal{F}$ koji ima maksimalan presjek sa trenutnim skupom U . Dati skup potom ubacujemo u rješenje C , te izbacujemo elemente iz U koji se nalaze u skupu f . Algoritam se izvršava dok god ne pokrijemo sve elemente iz U , tj. dok god $U \neq \emptyset$.

Algoritam 9 GREEDY-SET-COVER metod za problem pokrivanja skupa.

```
1: Ulaz:  $X$  i familija podskupova  $\mathcal{F}$  od  $X$ 
2: Izlaz:  $C$  – pokrivač skupa  $X$ 
3:  $U \leftarrow X$ 
4:  $C \leftarrow \emptyset$ 
5: while  $U \neq \emptyset$  do
6:    $f \leftarrow \operatorname{argmax}_{f \in \mathcal{F}} |f \cap U|$ 
7:    $U \leftarrow U \setminus f$ 
8:    $C \leftarrow C \cup \{f\}$ 
9: end while
```

Algoritam se izvodi u $O(|\mathcal{F}| \cdot |X|)$ vremenu, što znači da je polinomijalne složenosti. Takođe, jasno je da je rješenje koje se vraća dopustivo, uz pretpostavku da instanca generiše dopustiv prostor, što je lako provjeriti prije pokretanja algoritma.

O aproksimativnom faktoru ovog pohlepnog algoritma nam govori naредna teorema, koju navodimo bez dokaza.

Teorema 9.5. *GREEDY-SET-COVER algoritam ima aproksimativni faktor*

$$H(\max\{|f| \mid f \in \mathcal{F}\}),$$

gdje je $H(n) = \sum_{i=1}^n \frac{1}{i}$.

Napomenimo da postoji i težinska verzija problema pokrivanja skupa. U ovoj verziji je svakom skupu iz $f \in \mathcal{F}$ dodijeljena težina w_f . Zadatak je pronaći pokrivač $C \subseteq \mathcal{F}$ skupa X sa najmanjom ukupnom težinom. Ovaj problem se, uz pomoć sličnog pohlepnog algoritma kao kod netežinske varijante, može aproksimirati do na faktor $H_d = \log d + O(1)$, gdje je $d = \max\{|f| : f \in \mathcal{F}\}$. U tom slučaju, pohlejni kriterijum je određen funkcijom

$$g(f) = \frac{w_f}{|f \cap U|}, f \in \mathcal{F} \setminus C, \quad (9.1)$$

gdje je w_f težina skupa f . U svakom koraku algoritma određuje se skup $f^* \in \mathcal{F}$ koji minimizuje vrijednost funkcije g , koji se zatim dodaje u trenutni (parcijalni) pokrivač C .

9.3 Heurističke metode

U prethodnom odjeljku smo vidjeli da za aproksimativne algoritme vrijedi važna osobina da se rješavanjem problema ne garantuje pronađak optimalnog rješenja, ali postoji garancija da se rješenje nalazi unutar nekog faktora kvaliteta (c), u odnosu na optimalno rješenje. Drugim riječima, za rješenja dobijena aproksimativnim algoritmima se zna koliko njihov kvalitet maksimalno može da odstupa od kvaliteta optimalnog rješenja.

Posebna vrsta neegzaktnih metoda, odnosno metoda koje ne daju i/ili ne garantuju pronađak optimalnog rješenja pri završetku su tzv. heurističke metode.

Za razliku od aproksimativnih algoritama, kod heurističkih metoda, u opštem slučaju, ne propisuje se faktor kvaliteta rješenja, već je primarni cilj razviti i primjeniti tehniku pomoći koje se u razumnom vremenu može doći do dovoljno dobrog (zadovoljavajućeg) rješenja. Heurističke metode mogu biti izvedene iz teorijskih razmatranja, ali i iz eksperimentalnih rezultata. Npr. u praksi je čest slučaj da se heurističke metode razvijaju na osnovu rješavanja problema manjih dimenzija ili uopštavanjem tehnika uspješno primijenjenih na uzorcima.

Snaga heurističkih algoritama leži u smanjenju prostora pretraživanja i usmjeravanju procesa pretraživanja u one regije koji sadrže bolja rješenja, čime se značajno ubrzava proces pronađenja rješenja. Naravno, ne postoji univerzalna strategija koja garantuje uspjeh heurističkog algoritma, tj. ne postoji garancija da će heuristički algoritam uvijek pronaći dovoljno dobro rješenje u razumnom vremenu. Ipak, višedecenijski razvoj i upotreba različitih heurističkih algoritama ukazuju na činjenicu da ove metode zauzimaju veoma važno mjesto u oblasti kombinatorne optimizacije, a njihova uspješna primjena u širokom spektru različitih problema dokazuje njihovu stvarnu upotrebnu vrijednost.

Svaka heuristička metoda mora da zadovolji dva pravila: da je rešenje „dovoljno dobro“ i da se algoritam pokrenut na ulazni problem izvršava u realnom, polinomijalnom vremenu. Treba napomenuti da se pod „dovoljno dobrim“ rješenjem podrazumijevaju različite procjene, koje mogu biti subjektivne (npr. zasnovane na ličnoj ili profesionalnoj procjeni stručnjaka

koji razvija ili koristi metodu), objektivne (npr. rješenje je dobro ako od optimalnog rješenja odstupa u dozvoljenim granicama) ili zavisne od drugih uslova (npr. traži se rješenje problema takvo da je ono ekonomski prihvatljivo, bez obzira na to da li je blizu optimalnog rješenja ili ne). Za neke metode, kvalitet dobijenog rješenja je moguće utvrditi samo eksperimentalno, npr. poređenjem sa rješenjima dobijenim uz pomoć drugih metoda. Nekada je cilj heurističke metode samo da se pronađe jedno dopustivo rješenje, dok se kvalitet samog rješenja ne razmatra.

Važna osobina dobro konstruisanih heurističkih metoda je mogućnost prevazilaženja lokalnog ekstrema (što nije slučaj sa pohlepnim algoritmima), u cilju dostizanja globalnog optimalnog rješenja.

Takođe, heurističke metode se često uspešno mogu integrisati i u egzaktne metode, čime se kombinuje brzina dobijanja rješenja i potvrđivanje optimalnosti.

9.4 Metaheurističke metode

Široku klasu heurističkih metoda čine tzv. „univerzalne“ heurističke metode, koje se, uz blage modifikacije i prilagođavanje, mogu koristiti za rješavanje više različitih problema. Takve metode nazivamo *metaheurističkim* metodama, ili metaheuristikama.

Metaheuristika se formalno definiše kao iterativni proces koji koristi približne metode, kombinujući na inteligentan način različite koncepte za pretraživanje prostora rješenja, da bi se u razumnom vremenu, „dohvatilo“ rješenje što bliže optimalnom.

Sa praktičnog aspekta, cilj metaheuristika je pretraživanje skupa dopustivih rješenja, pri čemu je, ponekad, dozvoljeno i proširenje tog skupa elementima koji nisu dopustivi, prelazak u rješenja slabijeg kvaliteta kako bi se izbjegli lokalni optimumi koji nisu i globalni, kombinovanje sa drugim heuristikama, itd.

9.4.1 Klasifikacija metaheuristika

S obzirom na veliki broj raznih metaheuristika koje su razvijene u posljednje vrijeme, prirodno je očekivati da se za njihov razvoj koriste vrlo različiti pristupi, počev od polazne motivacije, preko strategije pretraživanja, različitih namjena i klasa problema na kojima se primjenjuju. U savremenoj literaturi, sreće se nekoliko kriterijuma za klasifikaciju metaheuristika.

Klasifikacija metaheuristika prema polaznoj motivaciji identificira tzv. metaheuristike zasnovane na simulaciji prirodnih pojava (eng. *nature inspired metaheuristics*), kao što su na primjer genetski algoritmi, mravlje ili pčelinje kolonije ili elektromagnetizam. Nasuprot njima, veliki broj metaheuristika nije motivisan prirodnim procesima, kao npr. tabu pretraživanje, iterativna lokalna pretraga itd.

Prema broju tačaka kojima se raspolaže unutar jedne iteracije, metaheuristike se dijele na dvije klase: metaheuristike zasnovane na populaciji (eng. *population based*) i metaheuristike koje rade sa jednom tačkom unutar jedne iteracije (eng. *single point search*). U prvoj klasi, u datom trenutku algoritam raspolaže sa više tačaka – različitih rješenja problema. Pretraživanje prostora je zasnovano na primjeni različitih operatora kombinovanja, pomjeranja, izmjena i odabira pojedinih tačaka, kako bi se u svakoj novoj iteraciji formirala nova populacija, sa potencijalno boljim rješenjima. Takve metaheuristike su: genetski algoritmi, mravlje kolonije, elektromagnetizam, jata ptica, itd. U metaheuristikama koje raspolažu jednom tačkom, primjenjuje se strategija zasnovana na praćenju trajektorije (eng. *trajectory methods*), koja podrazumijeva da se kroz iteracije formira niz rješenja, od kojih je svako naredno (po pravilu, ali ne i obavezno) bolje od prethodnog. Ovi algoritmi najčešće, ali opet ne i obavezno, koriste i neke metode lokalnog pretraživanja, gdje se novo, potencijalno bolje rješenje bira unutar neke okoline trenutnog rješenja. Neke od metoda koje pripadaju ovoj klasi su metoda promjenljivih okolina, tabu pretraživanje, simulirano kaljenje, metoda promjenljivih okolina, itd.

Prema tipu funkcije cilja, metaheuristike se mogu klasifikovati na one koje koriste statičku, te one koje koriste dinamičku funkciju cilja. Kako i sam termin ukazuje, dinamičke funkcije cilja se ažuriraju prema trenutnim

zahtjevima u toku samog izvršenja metaheuristike. Suprotno njima, statičke funkcije se ne mijenjaju za vrijeme izvršenja metaheuristike.

Neke od najčešće korištenih metaheuristika su:

- pohlepni algoritam sa prilagođenom slučajnom pretragom (eng. *greedy randomized adaptive search procedure*) – GRASP;
- tabu pretraživanje (eng. *tabu search*) – TS;
- genetski algoritam (eng. *genetic algorithm*) – GA;
- metod promjenljivih okolina (eng. *variable neighborhood search*) – VNS;
- simulirano kaljenje (eng. *simulated annealing*) – SA;
- mravlje kolonije (eng. *ant colony optimization*) – ACO;
- pčelinje kolonije (eng. *artificial bee colony*) – ABC;
- elektromagnetizam (eng. *electromagnetism*) – EM;
- optimizacija rojem čestica (eng. *particle swarm optimization*) – PSO;
- upravljano lokalno pretraživanje (eng. *guided local search*) – GLS.

U nastavku ovog poglavlja, detaljnije će biti objašnjena po jedna metaheuristika iz grupe populacionih metoda (genetski algoritam) i iz grupe metoda koje u procesu pretrage raspolažu samo jednom tačkom (metoda promjenljivih okolina). Pored ove dvije metode, biće objašnjen i princip lokalnog pretraživanja. Čitaocu preporučujemo da se, koristeći se internet izvorima, manje ili više detaljno upozna i sa drugim metaheuristikama, koje su nabrojane u ovom odjeljku.

Algoritam 10 Lokalna pretraga (maksimizacija)

-
- 1: **Ulaz:** instanca problema, okolina \mathcal{N}
 - 2: **while** $\{x' \in \mathcal{N}(x) \mid f(x') > f(x)\} \neq \emptyset$ **do**
 - 3: $x \leftarrow$ izaberi $x' \in \mathcal{N}(x)$ ako vrijedi $f(x') > f(x)$
 - 4: **end while**
-

9.4.2 Metoda lokalne pretrage

Metoda lokalne pretrage je tehniku čija se strategija sastoji u izvođenju niza poboljšanja rješenja, počevši od nekog unaprijed zadanog, radeći „male” promjene u strukturi trenutno najboljeg rješenja. Uvedimo prvo koncept okolina i lokalnih rješenja kao baznih pojmove u metodologiji lokalne pretrage. Za dato rješenje x , definišimo skup (ne obavezno dopustivih) rješenja koja su „blizu” x u odnosu na neku mjeru. Formalno, okolina predstavlja preslikavanje \mathcal{N} gdje svakom (dopustivom) rješenju pridružujemo skup rješenja, koje zovemo okolina od x . Funkcija okoline se takođe može definisati preko operatora Δ , koji predstavlja kolekciju funkcija $\Delta : \Phi \mapsto \Phi$, gdje je Φ prostor pretrage (rješenja) razmatrane instance problema:

$$x' \in \mathcal{N}(x) \iff \exists \delta \in \Delta, \delta(x) = x'.$$

Prirodan izbor u mnogim aplikacijama je k -izmjenjena (eng. *k-exchange*) okolina: x i x' pripadaju istoj okolini akko se razlikuju na tačno k komponenti rješenja. Npr. za TSP problem, 2-izmjenjena okolina je prirodan izbor gdje su komponente rješenja grane datog grafa.

Rješenje x je *lokalni optimum* u odnosu na okolinu \mathcal{N} akko $f(x) \leq f(x')$ ($f(x) \geq f(x')$), za sve $x' \in \mathcal{N}(x)$, ako minimizujemo (maksimizujemo) ciljnu funkciju f . *Globalni optimum* je rješenje koje je lokalni optimum u odnosu na bilo koju okolinu. Iako je to većini čitalaca jasno, nije zgoreg još jednom pomenuuti da nije svaki lokalni optimum ujedno i globalni.

Pseudokod lokalne pretrage je dat u Algoritmu 10. Ulaz u algoritam je instanca problema i jedna unaprijed definisana okolina \mathcal{N} . U svakoj iteraciji, lokalna pretraga poboljšava trenutno (najbolje) rješenje ako rješenje u okolini $\mathcal{N}(x)$ ima bolju vrijednost ciljne funkcije. Algoritam se prekida ako

je trenutno rješenje lokalno najbolje u odnosu na definisanu okolinu. Postoje dvije strategije za izbor rješenja koje će zamijeniti trenutno najbolje rješenje, pod pretpostavkom da postoji više od jednog rješenja sa boljom vrijednošću ciljne funkcije u posmatranoj okolini. Te strategije su:

- *prvo poboljšanje* (eng. *the first improvement strategy*) – ova strategija prihvata prvo nađeno rješenje x' u okolini $\mathcal{N}(x)$ sa boljom vrijednošću ciljne funkcije od x . Pretraga se pomjera u tačku $x = x'$, odakle se pokreće nova iteracija lokalne pretrage. Lokalna pretraga se zaustavlja kada nema više poboljšanja.
- *najbolje poboljšanje* (eng. *the best improvement strategy*) – ova strategija provjerava sva rješenja iz okoline $\mathcal{N}(x)$ i za novo najbolje rješenje bira ono koje je lokalno najbolje u toj okolini. Nalazak najboljeg rješenja zahtijeva enumeraciju svih rješenja iz okoline. Stoga, upotreba ove strategije, u opštem slučaju, zahtijeva više procesorskog vremena, tako da ona nije uvijek primjenjiva. U ovom slučaju, lokalna pretraga se završava kada nema više poboljšanja.

Okolina kod koje se formira direktnom razmjrenom vrijednosti dvije promjenljive u rješenju se često naziva i *1-razmjena* (eng. *swap*) okolina.

Primjer 9.3. U problemu 0-1 ruksaka, za okolinu \mathcal{N} bi mogli uzeti 1-exchange okolinu. Recimo, za $n = 4$ i rješenje $x = 1100$ imali bismo

$$\mathcal{N}(x) = \{0100, 1000, 1110, 1101\}.$$

9.4.3 Genetski algoritmi

Genetski algoritmi spadaju u klasu populacionih i evolucijskih algoritama, čije je ponašanje inspirisano procesom evolucije koja se odvija u prirodi. Kao što je već rečeno, u osnovi se podrazumijeva rad sa populacijom rješenja (jedinki), gdje svaka jedinka predstavlja potencijalno rješenje polaznog problema, dok je populacija podskup ukupnog prostora pretraživanja. Populacija se u iterativnom postupku mijenja tako što se stare jedinke mijenjaju novim, potencijalno boljim jedinkama.

Osnovni okvir za funkcionisanje genetskih algoritama je prezentovan još 1975. godine od strane Džona Holanda, u čuvenoj knjizi „Adaptation in natural and artificial system”.

Svakoj jedinki se dodjeljuje brojna vrijednost, tzv. *prilagodenost* (eng. fitness), koja ocjenjuje kvalitet posmatrane jedinke. Cilj genetskog algoritma je da se, iz iteracije u iteraciju, pronalaze jedinke sa sve boljom prilagodenošću, pod čime se smatra i pojedinačno poboljšanje nekih jedinki, ali i prosječna prilagodenost kompletne populacije, što se postiže standardnim genetskim operatorima: *selekciom, ukrštanjem i mutacijom*.

Populacija je centralna struktura kod genetskog algoritma i ona obično broji od nekoliko, pa do nekoliko stotina jedinki (u rijetkim situacijama i do hiljadu). Svaka jedinka se predstavlja genetskim kôdom, koji se zapisuje nad nekom konačnom, najčešće binarnom ili cjelobrojnom azbukom. Segmenti genetskog kôda se nazivaju *geni*.

Početna populacija se obično kreira na slučajan način, čime se omogućava kreiranje raznovrsnog genetskog materijala, korisnog za kasnije faze algoritma i raznovrsnost samih jedinki. U nekim slučajevima se za generisanje polaznih jedinki koristi i neka druga heuristika, čime se pretraga od početka usmjerava ka onim oblastima pretraživanja za koje se prepostavlja da sadrže kvalitetnija rješenja.

Tokom rada algoritma, na populaciju se primjenjuju genetski operatori selekcije, mutacije i ukrštanja. Pored ovih osnovnih operatora, na neke jedinke se mogu primijeniti i drugi operatori, čijom se primjenom teži ka poboljšanju rješenja (na primjer, na neke, ili sve jedinke se mogu primijeniti i tehnike lokalnog pretraživanja).

Operator selekcije bira jedinke na koje se dalje primjenjuju genetski operatori ukrštanja i mutacije. Po pravilu, boljim jedinkama, odnosno jedinkama sa boljom prilagodenosti, se operatorom selekcije daje veća vjerojatnoća da će biti izabrane, dok slabije jedinke selekcijom dobijaju manju šansu. Postoji veći broj tehnika selekcije, a najpoznatije su: jednostavna (ili rulet) selekcija, turnirska, eliminacijska selekcija i elitizam. Rulet selekcija diktira da je vjerojatnoća da će jedinka biti izabrana proporcionalna prilagodenosti. Kod turnirske selekcije se iterativno formiraju skupovi jedinki, od kojih samo ona sa najboljom prilagodenosti opstaje i ulazi u dalji proces

kombinovanja sa drugim jedinkama. Eliminacijska selekcija je primarno zasnovana na izbacivanju loših jedinki, prije nego na odabiru boljih za narednu generaciju. Značajan broj savremenih implementacija genetskog algoritma uključuje i koncept *elitizma*. Elitizam podrazumijeva da će nekoliko najboljih (elitnih) jedinki direktno preći u narednu generaciju.

Ukrštanjem se kombinuju geni jedinki. Rezultat ukrštanja jedinki su nove jedinke, koje potencijalno sadrže „dobre” gene roditelja od kojih su nastale. Ovim mehanizmom i jedinke slabije prilagođenosti, ali koje sadrže dobre gene, dobijaju šansu da učestvuju u reprodukciji i prenesu „dobar” genetski materijal na sljedeću generaciju. Najčešće korišteni operatori ukrštanja su tzv. jednopoziciono, odnosno dvopoziciono ukrštanje, kod kojih se određuje jedna, odnosno dvije pozicije između kojih se odvija razmjena genetskih kôdova roditelja, respektivno.

Mutacijom se vrši slučajna promjena određenog gena (sa nekom malom vjerovatnoćom da će se mutacija desiti), čime se postiže mogućnost vraćanja dobrog genetskog materijala, ukoliko je on izgubljen primjenom operatora selekcije. Time se proces pretrage „izvlači” iz preuranjene konvergencije i usmjerava ka onim oblastima pretraživanja koje sadrže potencijalno bolje jedinke.

Nakon primjene genetskih operatora, formira se naredna generacija. Populacija se tako sastoji od potomaka i nekih starijih jedinki koje su preživjele, s obzirom na njihov kvalitet (elitne jedinke). Producija novih generacija se nastavlja sve dok ne budu ispunjeni uslovi za završetak algoritma. Najčešći kriterijumi za završetak algoritma su dostignuto maksimalno vrijeme izvršenja, dostignut maksimalan ukupan broj iteracija, ili dostignut maksimalan broj iteracija bez poboljšanja najboljeg rješenja.

Jedna od mogućih varijanti genetskog algoritma je prikazana Algoritmom 11.

Nakon što se generiše početna populacija P , pokreće se iterativni proces koji traje sve dok se ne ispuni kriterijum za zaustavljanje. Unutar iterativnog procesa, za svaku jedinku populacije P se računa funkcija prilagođenosti. Nakon toga se, redom, formira skup P' , koji inicijalno sadrži elitne jedinke i skup R koji je rezultat primjene operatora selekcije na populaciju P . Operatori ukrštanja i mutacije se primjenjuju na jedinke iz skupa R , a

Algoritam 11 Osnovna šema GA.

```
1:  $P \leftarrow \text{GenerisiInicijalnuPopulaciju}()$ 
2: while  $\text{IspunjeniKriterijZaustavljanja}()$  do
3:    $R \leftarrow \text{RacunajFunkcijuPrilagodjenosti}(P)$ 
4:    $P' \leftarrow \text{PrebacElitneJedinke}(P)$ 
5:    $R \leftarrow \text{IzvrsiSelekcijuRoditelja}(P)$ 
6:    $P'' \leftarrow \text{IzvrsiUkrstanjeIMutaciju}(R)$ 
7:    $P \leftarrow P' \cup P''$ 
8: end while
```

novodobijene jedinke se ubacuju u skup P'' . Po izvršenju ovih operatora, populacija P preuzima sve jedinke iz skupa P' i P'' i iterativni proces se nastavlja.

Primjer 9.4. U ovom primjeru posmatrajmo 0-1 problem jednodimenzijsnog ruksaka. Ponovimo da nam je, kod ovog problema, na raspaganju određen broj proizvoda, od kojih svaki proizvod ima svoju vrijednost i težinu. Ruksak je ograničenog maksimalnog kapaciteta (težine) C . Podsetimo se i da se u 0-1 problemu ruksaka svaki proizvod ili ne uzima nikako ili se uzima kompletan. Cilj je, kao i u drugim problemima ruksaka, odabrati onaj skup proizvoda, tako da je njihova ukupna vrijednost maksimalna, a da se ne naruši kapacitet ruksaka.

Formalna matematička definicija 0-1 problema ruksaka će nam pomoći u konstrukciji genetskog algoritma, pa ćemo je ovdje navesti.

Neka je dato n proizvoda, koji su opisani pomoću vektora p i w . Neka p_i i w_i predstavljaju vrijednost i težinu i -tog proizvoda. Neka je C maksimalan kapacitet ruksaka.

Neka je $f(\cdot)$ funkcija koja se maksimizuje i neka je x binarni niz dužine n koji predstavlja jedno rješenje problema: ako je i -ti proizvod uključen u rješenje, tada je $x_i = 1$, u protivnom, $x_i = 0$. Ovaj problem se definiše sa

Proizvod	Cijena	Težina
p1	5	7
p2	8	8
p3	3	4
p4	2	10
p5	7	4
p6	9	6
p7	4	4

Tabela 9.1: Primjer instance za 0-1 problem ruksaka

ILP modelom

$$\begin{aligned} \max \quad & \sum_{i=1}^n x_i p_i \\ \text{t.d.} \quad & \sum_{i=1}^n x_i w_i \leq C \\ & x_i \in \{0, 1\}, i = 1, 2, \dots, n \end{aligned}$$

Konstrukciju elemenata genetskog algoritma ćemo prikazati služeći se konkretnim primjerom. U Tabeli 9.1 su prikazani polazni podaci vezani za proizvode, a maksimalan kapacitet rukasaka je 22.

Kodiranje. Genetski kôd svake jedinke ćemo prikazati binarnim nizom dužine n (u primjeru je $n = 7$). Tako, na primjer, genetski kôd 1100100 označava da su proizvodi pod rednim brojem 1, 2 i 5 uključeni u ruksak, dok ostali proizvodi nisu uključeni u ruksak. Vrijednost funkcije cilja za ovo rješenje je $5+8+7=20$, dok je ukupna težina $7+8+4 = 19$. Primjetimo da ovakav sistem kodiranja može da dovede do nedopustivih rješenja. Npr. genetski kôd 0101010 označava da su proizvodi pod rednim brojem 2, 4 i 6 uključeni u rješenje, čija je ukupna težina $8+10+6 = 24$, što je više od maksimalno dozvoljenog kapaciteta ruksaka.

Treba napomenuti da se u rješavanju velikog broja problema kombinatorne optimizacije pomoću genetskih algoritama, ali i drugih heurističkih metoda, često dozvoljava rad sa nedopustivim rješenjima. Razlog za to leži u činjenici da nisu sva nedopustiva rješenja „podjednako loša”, a nekada je lakše popraviti nedopustivo rješenje u dopustivo rješenje visokog kvaliteta, nego vršiti pretragu samo na prostoru dopustivih rješenja slabijeg kvaliteta. Zbog toga se za prevazilaženje pojave nedopustivih rješenja najčešće koristi jedna od sljedeće dvije strategije:

- nedopustiva rješenja, po automatizmu, dobijaju minimalnu vrijednost funkcije prilagođenosti, čime se omogućava da sam genetski algoritam odbaci takva rješenja u procesu selekcije.
- uvodi se tzv. *kaznena funkcija* (eng. penalty function), kojom se „kažnjavaju” nedopustiva rješenja (najčešće smanjenjem vrijednosti funkcije prilagođenosti), ali se ona ne izbacuju iz razmatranja, u nadi da perspektivnija nedopustiva rješenja mogu prerasti u kvalitetna dopustiva rješenja.

U našem primjeru, pojavu nedopustivih rješenja, dakle, možemo riješiti automatskom dodjelom minimalne vrijednosti funkcije prilagođenosti, ili „kažnjavanjem” takvih rješenja smanjenjem vrijednosti funkcije prilagođenosti. Neke od strategije za „kažnjavanje” nedopustivih jedinki bi mogle biti zasnovane na procjeni broja proizvoda koje treba zamijeniti (ili izbaciti) iz datog rješenja, a da rješenje postane dopustivo. Na taj način bi se ona nedopustiva rješenja, koja se teže mogu popraviti u dopustiva, više kažnjavala (i samim tim češće i odbacivala), dok bi se perspektivnija nedopustiva rješenja zadržavala u populaciji, u nadi da će, primjenom nekog od genetskih operatora, prerasti u kvalitetna dopustiva rješenja. Uopšte, vrijednost kaznene funkcije bi trebalo da bude malo veća od minimalne cijene potrebne da se odgovarajuće nedopustivo rješenje „popravi” na dopustivo. Suprotno, ako bi vrijednost kaznene funkcije bila manja od cijene minimalne korekcije, nedopustiva rješenja se mogu javiti kao konačno rješenje, što naravno nije dopušteno. Sa druge strane, ako je vrijednost kaznene funkcije značajno veća od cijene minimalne popravke, nedopustiva rješenja su totalno

diskriminisana i algoritam ih odmah odbacuje.

Računanje funkcije prilagođenosti. Za posmatrani problem, prirodna funkcija prilagođenosti bi se mogla definisati na način da ona bude jednaka ukupnom zbiru vrijednosti proizvoda uključenih u rješenje (pod uslovom da je to rješenje dopustivo), a da je vrijednost funkcije prilagođenosti nedopustivih rješenja jednaka nuli. Imajući u vidu prethodno razmatranje o nedopustivim rješenjima, funkcija prilagođenosti može biti definisana i drugačije, tj. mogla bi da sadrži i dio vezan za „kažnjavanje“ nedopustivih rješenja, po nekom od predloženih (ili drugih) principa. Time bi nedopustiva rješenja bila uključena u pretragu.

Operator selekcije. Ilustrujmo rad rulet selekcije na našem primjeru 0-1 problemu ruksaka. Naprije pretpostavimo da je funkcija prilagođenosti jednaka vrijednosti funkcije cilja. Dalje, posmatrajmo populaciju od 4 jedinke, prikazane u Tabeli 9.2.

Ukupna prilagođenost svih jedinki (zbir vrijednosti funkcija prilagođenosti svih jedinki) iznosi 72. Vjerovatnoće da će jedinke preživjeti za sljedeću generaciju jednake su redom: $\frac{18}{72}$, $\frac{9}{72}$, $\frac{17}{72}$ i $\frac{28}{72}$.

Treba napomenuti da, iako se rulet selekcija na prvi pogled čini pogodnom za odabir jedinki koje preživljavaju u sljedećoj generaciji, ona ima i značajne nedostatke. Ovom selekcijom se jedinke sa slabom vrijednošću funkcije cilja često prerano odbacuju, iako možda sadrže potencijalno dobre gene, koji bi ukrštanjem sa drugim jedinkama mogli proizvesti još bolje potomke. Dalje, u kasnijim fazama algoritma, kada se javi više kvalitetnih jedinki koje imaju približno iste (ali ipak različite!) vrijednosti funkcije prilagođenosti, vjerovatnoće da će takve jedinke preživjeti su približno jednake, što dovodi do opasnosti da, nakon selekcije, najbolje jedinke i ne prežive u narednu generaciju. Stoga se u praksi značajno češće koristi turnirska selekcija, kojom se prevazilaze nedostaci rulet selekcije.

Operator ukrštanja. Kao što je već pomenuto, osnovni operatori ukrštanja su tzv. jednopoziciono i dvopoziciono ukrštanje. Objasnimo primjerom jednopoziciono ukrštanje, koje bi moglo biti realizovano na sljedeći način

	prilagođenost							težine
Jedinka1	1	0	0	0	0	1	1	18
Jedinka2	0	0	0	0	0	1	0	9
Jedinka3	0	1	0	1	1	0	0	17
Jedinka4	0	1	0	0	1	1	1	22

Tabela 9.2: Primjer korišten za ilustraciju upotrebe rulet selekcije

	Prije ukrštanja:	Poslije ukrštanja:
Jednika 1:	XXX XXXX	XXXYYYY
Jedinka 2:	YYY YYYY	YYYXXXXX

Tabela 9.3: Primjer operatora ukrštanja: jednopoziciono ukrštanje.

U ovom generičkom primjeru možemo vidjeti jednopoziciono ukrštanje u pozicijama (3, 3). Specifičnije, pretpostavimo da imamo dva rješenja (jedinke) konstruisane u algoritamskom procesu: 1100100 i 0100011. Na osnovu jednopozicionog ukrštanja, one produkuju dva nova rješenja (dva potomka), tako što se lijevi dio genetskog kôda prvog roditelja spaja sa desnim dijelom genetskog kôda drugog roditelja i obrnuto, lijevi dio genetskog kôda drugog roditelja se spaja sa desnim dijelom prvog roditelja. Treba napomenuti da se primjenom ovakvog operatora ukrštanja takođe mogu dobiti potomci koji su nedopustivi, bez obzira na to da li su jedinke roditelja dopustive ili ne. Njihova dalja egzistencija u populaciji ponovo zavisi od načina konstruisanja funkcije prilagođenosti.

Kod dvopozicionog ukrštanja postoje dvije pozicije u svakoj od jedinika na osnovu kojih se radi ukrštanje. Primjer ovakvog ukrštanja je data Tabelom 9.4

	Prije ukrštanja:	Poslije ukrštanja:
Jednika 1:	XXX XXXX XX	XXXYYYYYXX
Jedinka 2 :	YYY YYYY YY	YYYYXXXXYY

Tabela 9.4: Primjer operatora ukrštanja: dvopoziciono ukrštanje.

U ovom primjeru ukrštanje je izvedeno u pozicijama (3, 7) prve jedinke i pozicijama (3, 7) druge jedinke. Kao i kod jednopozicionog ukrštanja i ovdje je moguće da novonastale jedinke ne budu dopustive.

Maska:	110001100
Jednika 1:	XXXXXXXXXX
Jedinka 2:	YYYYYYYYYY
Poslije ukrštanja:	
Nove jedinke:	XXYYYYXXYY YYXXXXYYXX

Tabela 9.5: Primjer operatora ukrštanja: uniformno ukrštanje.

Kod uniformnog ukrštanja, za svaki par jedinki koje se ukrštaju, prvo se generiše maska na slučajan način, tj. binarni niz dužine genetskog kôda jedinki. Ukoliko je na poziciji i generisane maske vrijednost 1, onda prvi potomak za i -ti gen uzima vrijednosti prve jedinke, dok je i -ti gen drugog potomka vrijednost i -tog gena druge jedinke. Inače, vrijedi obrnuta situacija. Jedan primjer ovakvog ukrštanja je data Tabelom 9.5

Mutacija. Kao što je pomenuto, mutacije podrazumijevaju male promjene u genetskom kôdu, čija je osnovna namjena izlazak iz lokalnih podoptimalnih rješenja i omogućavanje dalje pretrage u cilju pronađenja boljih rješenja. Najčešće se mutacija realizuje kao promjena vrijednosti jednog gena. U slučaju 0-1 problemu ruksaka, predstavljenog u Primjeru 9.4, u zavisnosti od drugih elemenata algoritma, mogao bi se konstruisati veći broj različitih operatora mutacije:

- Mutacija 1: prelazak neke nule u jedinicu. Ova mutacija bi modifiovala trenutnu jedinku na način da se neki proizvod, koji nije bio u rješenju, uključi u rješenje. Time bi se povećala ukupna vrijednost ruksaka, ali postoji opasnost da jedinka postane nedopustiva.
- Mutacija 2: prelazak neke jedinice u nulu. Ova mutacija bi modifiovala trenutnu jedinku na način da se neki proizvod, koji je bio u

rješenju, isključi iz rješenja. Ovo bi dovelo do smanjenja ukupne vrijednosti ruksaka, ali bi, u slučaju rada sa nedopustivim jedinkama, postojala mogućnost da nedopustivo rješenje postane dopustivo (ili da se poboljša kvalitet nedopustivog rješenja).

- Mutacija 3: dva gena različitih vrijednosti mijenjaju svoje vrijednosti. Preciznije, biraju se dva gena, od kojih jedan ima vrijednost 0, a drugi ima vrijednost 1, te prvi gen mijenja vrijednost na 1, a drugi na 0. Time se postiže da se, simultano, jedan proizvod, koji nije bio u rješenju, uključuje u rješenje, a drugi proizvod, koji je prije mutacije bio u rješenju, sada izbacuje iz rješenja. Kao i u prethodnim slučajevima, i ovdje je, na sličan način, potrebno razmatrati potencijalnu nedopustivost novonastale jedinke.
- Mutacija 4: zasnovana na tzv. uniformnoj mutaciji. Slično kao kod uniformnog ukrštanja, definiše se maska na osnovu koje se vrši mutacija određenih gena. Primjer ove mutacije dat je u Tabeli 9.6. Nakon mutacije, kod jedinke će biti izmijenjeni geni koji su na pozicijama 1 i 5.

Maska:	1000100
Jedinka:	1101000
	<hr/>
Nakon mutacije	
	0101100

Tabela 9.6: Primjer operatora mutacije (uniformno mutiranje)

Na kraju ovog odjeljka treba pomenuti da je višedecenijska primjena genetskih algoritama u rješavanju velikog broja problema kombinatorne optimizacije dovela je do pojave velikog broja različitih varijanti algoritma, koje se, između ostalog, razlikuju po operatorima koji se primjenjuju, načinu formiranja početne populacije, primjenama različitih strategija lokalnog pretraživanja, tehnike keširanja međurezultata, itd.

9.4.4 Metoda promjenljivih okolina

Metoda promjenljivih okolina (VNS) je metaheuristika uvedena od strane Nenada Mladenovića i Pjera Hansena 1997. godine. Pretraga je zasnovana na sistematskoj promjeni okolina da bi se izbjegle situacije kada algoritam „upada” u podoptimalna rješenja. Efikasnost VNS algoritma je zasnovana na razmatranju da u mnogim praktičnim optimizacionim problemima postoji povezanost između lokalnih optimuma. Stoga, metoda promjenljivih okolina i ne prati unaprijed zadatu putanju, već „skače” na potencijalno bolja rješenja koja se nalaze u nekoj okolini trenutno najboljeg rješenja. Ovaj proces je poznat pod nazivom *diversifikacija* (eng. *diversification*). Sistem je dodatno ojačan procedurom lokalnog pretraživanja, kojom se iz trenutnog rješenja prelazi u najkvalitetnije rješenje u nekoj njegovoj okolini. Ovaj proces je poznat pod nazivom *intenzifikacija* (eng. *intensification*).

Ponovimo da algoritam zasnovan na metodi promjenljivih okolina raspolaze jednom tačkom u jednoj iteraciji, a ne populacijom tačaka kao što je to slučaj sa genetskim algoritmom.

Tako, ideja da se pretraga usmjeri sa jednog na (moguće bolji) drugi lokalni optimum, koji je smješten u nekoj okolini polaznog lokalnog optimuma, predstavlja razuman i, kao što je to slučaj u velikom broju raznih optimizacionih problema, opravдан pristup. Da bi se postigli pomenuti efekti „iskakanja” iz lokalnog (pod)optimuma, kao i detaljno pretraživanje nove okoline trenutnog najboljeg rješenja, metoda promjenljivih okolina se najčešće realizuje pomoću dvije važne procedure: procedure *razmrđavanja* (eng. shaking) i procedure *lokalnog pretraživanja* (eng. local search). Kao i što je to slučaj sa svim metaheurističkim metodama, dugogodišnja primjena metode promjenljivih okolina na rješavanje različitih problema dovela je do pojave velikog broja različitih varijanti, koje, pored osnovnih metoda razmrđavanja i lokalnog pretraživanja, uključuju i mnoge druge efektivne strategije.

Da bi se proširila pretraga, VNS u potrazi za boljim lokalnim optimumom, obično koristi okoline rastuće kardinalnosti. Da bi se definisao skup okolina, prepostavimo da je x proizvoljno rješenje i N_k , za $k = k_{\min}, \dots, k_{\max}$, konačan skup okolina. Tada se $N_k(x)$ definiše kao skup rje-

šenja u k -toj okolini tačke x . Jedna od implementacija VNS metode, kojom se rješava problem maksimizacije, prikazana je Algoritmom 12.

U svakom koraku algoritma, VNS započinje od nekog rješenja x i cijelog broja k , $k_{min} \leq k \leq k_{max}$, koji označava trenutnu okolinu N_k . U proceduri razmrdavanja se na slučajan način bira rješenje x' iz okoline $N_k(x)$. Zatim se na to rješenje primjenjuje procedura lokalnog pretraživanja. Najbolje rješenje koje se dobije u lokalnom pretraživanju se dalje poredi sa trenutno najboljim rješenjem i ukoliko je ono bolje od trenutno najboljeg rješenja, ono se proglašava novim najboljim rješenjem i algoritam nastavlja sa daljim izvršavanjem. Postoje različite strategije kada algoritam prelazi u narednu okolinu (u proceduri razmrdavanja), kao i kada se okoline „resetuju”, tj. kada se algoritam vraća na razmatranje najmanje okoline.

Najčešći kriterijumi za prekid algoritma su: dostignuto maksimalno vrijeme, dostignut maksimalan broj iteracija, maksimalni broj iteracija između dva poboljšanja najboljeg rješenja.

Primjer 9.5. Da bismo ilustrovali način funkcionisanja metode promjenljivih okolina, možemo posmatrati isti primjer kao i u slučaju genetskog algoritma: 0-1 problem ruksaka.

Reprezentacija rješenja. Kao i u slučaju genetskog algoritma, rješenje problema se može predstaviti binarnim nizom. Ako i -ti element niza ima vrijednost 1, to znači da je i -ti proizvod uključen u rješenje, a u suprotnom je izostavljen.

Vrijednost funkcije cilja. Slično kao u primjeru genetskog algoritma, vrijednost funkcije prilagodenosti se najlakše može definisati kao zbir vrijednosti proizvoda uključenih u rješenje. Eventualno, ako se u algoritmu dozvoljava rad sa nedopustivim rješenjima, funkcija prilagodenosti može da sadrži i dio za „kažnjavanje“ nedopustivih rješenja. Razmatranje o kažnjavanju nedopustivih rješenja je slično kao u slučaju genetskog algoritma.

Razmrdavanje. Može se primijeniti više različitih tehnika razmrdavanja, a u ovom primjeru ćemo objasniti jedan pristup. U okviru procedure razmrdavanja, kreira se novo rješenje x' , ($x' \in N_k(x)$) zasnovano na trenutno najboljem rješenju x . k -ta okolina se definiše na sljedeći način: Nekih k proizvoda se bira na slučajan način. Svakom izabranom proizvodu mijenja se

Algoritam 12 VNS metaheuristika (maksimizacija)

```
1: Ulaz: inicijalno rješenje  $s$ , okoline  $\mathcal{N}_{k_{\min}}, \dots, \mathcal{N}_{k_{\max}}$ 
2: Izlaz: (poboljšano) rješenje  $s$ 
3:  $k \leftarrow k_{\min}$ 
4: while !(IspunjeniKriterijZaustavljanja()) do
5:    $s' \leftarrow$  izabradi random rješenje iz  $\mathcal{N}_k(s)$  // shaking faza
6:    $s' \leftarrow$  LokalnaPretraga( $s'$ )
7:   if  $f(s') > f(s)$  then
8:      $s \leftarrow s'$ 
9:      $k \leftarrow k_{\min}$ 
10:   else
11:      $k \leftarrow k + 1$  // koristiti narednu (VNS) okolinu
12:     if  $k > k_{\max}$  then
13:        $k \leftarrow k_{\min}$ 
14:     end if
15:   end if
16: end while
17: return  $s$ 
```

status: Ako je izabran proizvod koji je uključen u ruksak, on se izbacuje iz ruksaka i obrnuto.

Za pravilno funkcionisanje procedure razmrdavanja, potrebno je definisati kardinalnost najmanje i najveće okoline, koja zavisi od k . Obično se okolina najmanje kardinalnosti zasniva na malim vijednostima k , (na primer $k_{\min} = 1$ ili $k_{\min} = 2$), dok se za maksimalnu vrijednost k_{\max} uzimaju vrijednosti 20, 30, ili čak i više, u zavisnosti od prirode i dimenzije problema.

Praktično gledano, u našem primjeru u k -tom koraku izvršenja proced ure razmrdavanja, naš algoritam bi generisao jedno novo rješenje (označimo ga sa x'), koje se od trenutno najboljeg rješenja razlikuje u tačno k ko ordinata, odnosno u tačno k proizvoda. Npr. u slučaju da je rješenje x predstavljeno nizom 1100100 i ako je $k = 3$, tada bi rješenje x' moglo da (između ostalog) izgleda 1000111. Odnosno, vidimo da se ova dva rješenja razlikuju na tačno tri indeksa (indeksima 2, 6 i 7).

Lokalno pretraživanje. Za rješenje x' dobijeno u proceduri razmrdavanja poziva se lokalno pretraživanje. Ova procedura podrazumijeva temeljno (iscrpno) pretraživanje svih (ili skoro svih) rješenja, koja se blago razlikuju od rješenja x' . Lokalno pretraživanje u ovom problemu bi moglo biti zasnovano na sličnim razmatranjima kao mutacije kod genetskog algoritma. Npr. jedno lokalno pretraživanje bi moglo da bude zasnovano na tzv. *1-zamjena* principu (eng. 1-swap, odnosno razmjeni vrijednosti parova koordinata. Preciznije, lokalno pretraživanje razmatra sva rješenja nastala od rješenja x' na način da je jedan proizvod koji je bio uključen u ruksak sada izbačen, a neki drugi proizvod, koji prvo bitno nije bio u ruksaku, sada jeste.

Postoje različite strategije koliko dugo se izvršava lokalna pretraga. U opštem slučaju, potrebno je napraviti balans između utrošenog vremena pretraživanja i potrebe za lokalnim poboljšanjem rješenja. Stoga je generalno dobar pristup konstrukcija procedura lokalnog pretraživanja koje se brzo izvršavaju, kako bi se, zbog velike brzine izvršavanja jedne iteracije lokalne pretrage, omogućilo što iscrpnije pretraživanje okoline datog rješenja u razumnom vremenu. Takođe, nekada se kombinuje više metoda lokalnog pretraživanja (ovo važi za većinu heurističkih metoda), koje, manje ili više temeljno, pretražuju okoline datog rješenja, u zavisnosti od brzine ili drugih parametara.

9.5 Zadaci

1. Na šahovskoj tabli dimenzije 8x8 je postavljeno nekoliko topova, tako da se nikoja dva topa međusobno ne napadaju. Dodati na tu tablu još topova, tako da ih bude maksimalno 8, a da se i dalje nikoja dva topa ne napadaju.
2. Istražiti na internetu problem vraćanja kusura pomoću minimalnog broja novčanica (kovаницa). Koristi google pretragu *change-making problem* ili *coin change problem*. Napisati pohlepni algoritam koji rješava ovaj problem. Ispitati pod kojim uslovima pohlepni algoritam uvijek daje optimalno rješenje i pronaći kontraprimjer, kada pohlepni algoritam neće dati optimalno rješenje.

-
3. Osmisliti pohlepni algoritam koji rješava problem trgovačkog putnika. Pronaći kontraprimjer kada taj algoritam ne pronalazi optimalno rješenje.
 4. Osmisliti pohlepni algoritam koji rješava problem određivanja maksimalnog nezavisnog skupa čvorova u grafu. Odredi neke klase grafova za koje taj pohlepni algoritam uvijek pronalazi optimalno rješenje. Pronaći kontraprimjer kada pohlepni algoritam ne pronalazi optimalno rješenje.
 5. Svaki razlomak se može napisati kao zbir razlomaka od kojih svaki ima jedinicu u brojiocu. Taj zapis su koristili još i stari Egipćani, po kojima se taj zapis i zove „egipatski razlomak”. Napisati pohlepni algoritam koji proizvoljan razlomak zapisuje kao „egipatski razlomak”, odnosno kao zbir razlomaka koji imaju jedinicu u brojiocu.
 6. Zadan je skup S koji se sastoji od n brojeva. Koristeći pohlepni algoritam, podijeliti skup S na dva podskupa, tako da se zbroovi elemenata u prvom i drugom podskupu što manje razlikuju.
 7. Neka je dato N kajakaša koji su teški redom $1 \leq w_1 \leq w_2 \leq \dots \leq W_N$. Cilj je rasporediti kajakaše u što manje kajaka–dvosjeda. Maksimalna težina koju jedan kajak može da izdrži je k . Prepostavka je da je za svako i , $w_i \leq k$. Napisati pohlepni algoritam koji rješava ovaj zadatak.
 8. Napisati pohlepni algoritam koji ispituje da li je jedna riječ podriječ druge riječi.
 9. Poznate su visine n momaka i n djevojaka. Napisati program koji određuje koliko se najviše plesnih parova može formirati tako da je momak uvijek viši od djevojke.
 10. Konstruisati primjer za 0-1 problem ruksaka, koji se ne može do optimalnosti rješiti pohlepnim algoritmom prikazanom u Odjeljku 9.1.3, koji se koristi za rješavanje frakcionog problema ruksaka.

-
11. U literaturi ili na internetu, pronaći primjere problema za koji postoje apsolutni apriksimativni algoritam (na primjer, bojenje planarnog grafa, eng. *Planar graph coloring*), odnosno primjer problema za koji postoji relativni apriksimativni algoritam.
 12. Implementirati aproksimativne algoritme pomenute u Odjeljku 9.2.3 i analizirati rješenja na različitim instancama problema.
 13. Implementirati genetski algoritam za rješavanje 0-1 problema ruksaka iz Primjera 9.4.
 14. Problem postavljanja kraljica na šahovsku tablu podrazumijeva pozicioniranje kraljica tako da se nikije dvije kraljice ne napadaju. Osmisliti genetski algoritam koji rješava ovaj problem.
 15. Bojenje grafa je procedura koja svakom čvoru grafa dodjeljuje jednu boju, poštujući pravilo da susjedni čvorovi nisu obojeni istom bojom.
 - (a) Napisati pohlepni algoritam pomoću kog se vrši bojenje grafa, tako da se iskoristi što je manje moguće boja. Okarakterisati neku klasu grafova za koju pohlepni algoritam uvijek daje optimalno rješenje. Konstruisati kontraprimjer kada pohlepni algoritam neće dati optimalno rješenje.
 - (b) Osmisliti genetski algoritam za rješavanje problema bojenja čvorova grafa.
 - (c) Osmisliti metodu promjenljivih okolina kojom se rješava ovaj problem
 16. Osmisliti genetski algoritam i metodu promjenljivih okolina kojima se rješava problem određivanja maksimalne klike u grafu.
 17. Osmisliti genetski algoritam i metodu promjenljivih okolina kojima se rješava problem određivanja maksimalnog nezavisnog skupa čvorova u grafu.

-
18. Neka je dat ruksak čija je veličina jednaka C i skup proizvoda $I = \{1, \dots, n\}$, gdje proizvod i ima svoju cijenu $c_i > 0$ i veličinu $s_i > 0$. Pokrivač ruksaka je skup proizvoda $I' \subseteq I$ tako da je $\sum_{i \in I'} s_i > C$. Problem pokrivača ruksaka traži podskup proizvoda koji je pokrivač ruksaka sa minimalnom cijenom. Prepostavimo da ne postoji proizvod čija je cijena veća od cijene optimalnog rješenja. Konstruisati 2-aproksimativni algoritam za ovaj problem pod prethodnom pretpostavkom.
 19. Pokazali smo da postoji 2-aproksimativni algoritam za problem pokrivanja čvorova. Da li se može konstruisati instanca na kojoj neki od dva aproksimativna algoritma prikazana u Odjeljku 9.2.1, ne daje bolju aproksimaciju od faktora 2?
 20. Neka je dat kompletan neusmjeren graf $G = (V, E)$ u kojem težine grane zadovoljavaju nejednakost trougla i neka je k pozitivan prirodan broj. Problem metričkog k -klasterovanja (eng. Metric k -clustering) se sastoji od partionisanja skupa V na k disjunktnih skupova V_1, \dots, V_k , na način da se minimizuje najveća cijena grane između dva čvora u istom skupu, simbolički zapisano

$$\min \max_{1 \leq i \leq k, u, v \in V_i} c(uv).$$

Konstruisati 2-aproksimativni algoritam za problem metričkog k -klasterovanja.

21. Neka je dat graf $G = (V, E)$. Naći particiju V na k skupova V_1, \dots, V_k tako da je broj grana koje su između dvije različite particije minimizovan. Konstruisati (pohlepni) algoritam koji je $(1 - \frac{1}{k})$ -aproksimativni.
22. U ovom zadatku se rješava Problem obilaska segmenata (eng. *Segment tour problem*). Neka je dat skup segmenata $S = \{s_0 = (p_0, p_1), s_1 = (p_2, p_3), \dots\}$ u ravni. Naći put u ravni, koji (potpuno) sadrži svaki od segmenata iz S , a pri tom je minimalne dužine. Konstruisati 3-aproksimativni algoritam. Napomenimo da nalaženje ovakvog puta odgovara pronalasku redoslijeda posjećivanja svakog od segmenata.

-
23. Neka je dat kompletan težinski graf $G = (V, E)$ čije grane zadovoljavaju nejednakost trougla. k poštara treba da prođu grafom tako da svaka grana u datom podskupu grana $E' \subseteq E$ bude pređena od strane barem jednog poštara. Startni i završni čvor za svakog od poštara je izabran na slučajan način. Zadatak ovog problema je naći šetnju u grafu sa minimalnom dužinom (za sve poštare). Konstruisati 2-aproksimativni algoritam za ovaj problem.
24. Neka je dat usmjereni graf $G = (V, E)$ sa nenegativnim težinama na granama, gdje je sa $w(uv)$ označena težina grane $uv \in E$. Zadatak je naći podskup čvorova $S \subseteq V$ tako da je suma težina grana iz S u $V \setminus S$ najveća moguća, tj. suma

$$\sum_{s \in S, t \notin S, st \in E} w(st)$$

je maksimizovana.

Konstruisati $\frac{1}{4}$ -aproximativni algoritam.

25. Neka je data instanca problema sa pozitivnim prirodnim brojem k i multi-skupom $T = \{t_1, \dots, t_n\}$ vremena (procesiranja poslova/zadataka), $t_i \in \mathbb{Q}$ za sve $i = 1, \dots, n$. Potrebno je dodijeliti zadatke svakoj od k mašini (dakle, funkcija $f : \{1, \dots, n\} \mapsto \{1, \dots, k\}$) tako da je vrijeme završetka svih zadataka na mašinama minimizovano, pod uslovom da one mogu izvršavati zadatke paralelno. Simbolički zapisano, funkcija cilja koja se minimizuje je data sa

$$\max \left\{ \sum_{i: f(i)=j} t_i \mid j = 1, \dots, k \right\}$$

Konstruisati 2-aproksimativni algoritam.

Glava 10

Optimizacioni rješavači

Kao što smo i pomenuli u uvodnom poglavlju, konstrukcija modela koji (najčešće) odgovara idealiziranoj verziji realnog problema je dio koji uzima najveći dio vremena u operacionim istraživanjima. Kada dođemo do modela koji približno odgovara potrebama, rješavanje modela se svodi na pozivanje odgovarajućih rješavača. Neki od najpoznatijih su: CPLEX koji može biti pozivan u Javi, C++ i Python-u uz pomoć odgovarajućih API-ja, zatim Lindo, Gurobi, te modul PuLP u Python-u.

U nastavku opisujemo korištenje CPLEX i PuLP rješavača za implementaciju (M)ILP modela i njihovog rješavanja.

10.1 Optimizacioni rješavač CPLEX

CPLEX alat je opšti rješavač koji se koristi za rješavanje optimizacionih problema. Iako se CPLEX može koristiti i za rješavanje nekih drugih klasa problema, u ovom poglavlju ćemo razmatrati LP, ILP i MILP probleme. Različite verzije ovog programa, uključujući i besplatnu verziju koja sadrži određena ograničenja u mogućnostima, mogu se preuzeti preko adrese koja je prikazana u Literaturi.

U nastavku opisujemo proces same implementacije modela u CPLEX-u kroz programske jezik C++ u Linux okruženju (Ubuntu 20). Da bismo

uključili CPLEX u *C++* kôd, nepohodno je koristiti sljedeći zapis

```
#include<ilcplex/ilocplex.h>
ILOSTLBEGIN
```

ili umjesto prve linije uključiti (pomoću `#include`) *ilocplex.h* fajl sa apsolutnom putanjom do date datoteke.

Da bi se inicijalizovao LP model, neophodno je prvo inicijalizovati okruženje za modelovanje. Dakle, konstruišemo objekat klase *IloEnv* sljedećim kôdom.

```
IloEnv env;
try{
//ovdje se deklarise model
}
catch(IloException& e){
    cerr >> "Greska: " >> e >> endl;
}
```

Unutar *try-catch* bloka deklarišemo prvo objekat modela (instanca klase *IloModel*), a zatim se model uvlači u rješavač CPLEX (kreira se objekat klase *IloCplex*) i rješava pozivanjem odgovarajuće metode *solve()*.

Model je instanca klase *IloModel*, čiji konstruktor uzima objekat klase *IloEnv* kao parametar. Model koji se rješava pomoću CPLEX-a je instanca klase *IloCplex*, koji uzima instancu klase *IloModel* kao vrijednost svog parametra.

```
IloModel model(env); //def. modela
// potrebno deklarisati ciljnu funkciju,
// promjenljive i ograničenja
IloCplex cplex(model);
if(!cplex.solve()){// poziv cplex rjesavaca
    env.error() << " Problem nije optimizovan." << endl;
```

```
        throw(-1);
    }
//cplex zavrsio sa optimizovanjem problema ... ispis rjesenja
```

Sljedeći korak je implementacija promjenljivih, funkcije cilja i ograničenja u modelu.

Implementacija promjenljivih u modelu. Ako želimo da deklarišemo realnu promjenljivu x gdje je, na primjer, $0 \leq x \leq 40$, koristimo klasu *IloNumVar* i pišemo sljedeći kôd

```
IloNumVar x = IloNumVar(env, 0.0, 40.0);
```

Ova definicija je ekvivalentna dodavanjem *ILFLOAT* konstante kao četvrtog argumenta u konstruktoru. Ako je potrebno implementirati promjenljivu za koju je $x \geq 0$ to radimo kao u prethodnom primjeru uz korištenje *IloInfinity* konstante.

Ako želimo da implementiramo promjenljivu koja je cijelobrojna binarna promjenljiva, koristimo sljedeći kôd

```
IloNumVar x = IloNumVar(env, 0.0, 1.0, ILOINT);
```

Ako je potrebno definisati vektor promjenjivih $x = (x_1, \dots, x_n)$, gdje je, recimo, $x_i \geq 1$ za sve i , onda se definiše objekat klase *IloNumVarArray*,

```
IloNumVarArray x(env);
for(int i = 0; i < n; ++i){
    x.add(IloNumVar(env, 1.0, IloInfinity));
}
```

Ako želimo da deklarišemo visedimenzionalne promjenjive kao što je $x_{i,j}$, $i = 1, \dots, n$ i $j = 1, \dots, m$, onda pozivamo šablon klasu *IloArray <>* na sljedeći način

```

typedef IloArray<IloNumVarArray> FloatMatrix; // x_ij
// 3D niz var: x_ijk
typedef IloArray<FloatMatrix> Float3Matrix;

FloatMatrix x(env, n);
for(int i = 0; i < n; i++){
    x[i] = IloNumVarArray(env, m);
    for(int j = 0; j < m; j++){
        x[i][j] = IloNumVar(env, 0.0) // x_ij>=0
    }
}

```

Implementacija funkcije cilja u modelu. Postoje dvije klase koje služe za deklarisanje funkcija cilja: *IloMaximize* i *IloMinimize*, zavisno od toga da li riječ maksimizaciji ili minimizaciji funkcije cilja. *IloMaximize* objektu se proslijeduju dva parametra: objekat *IloEnv* okruženja, te objekat klase *IloExpr* čime se definiše izraz koji odgovara funkciji cilja.

```

IloExpr expr(env);
//definisanje expr
IloObjective obj = IloMaximize(env, expr);
model.add(obj);

```

Za formiranje izraza *expr* možemo koristiti operator „ $+=$ ”, gdje se trenutni izraz sabira sa nekim objektom *IloNumVar* klase – dakle promjenljivom (eventualno pomnoženom sa nekim koeficijentom). Npr. ako želimo da implementiramo funkciju cilja $\sum_{i=1}^n c_i x_i$, koristili bismo sljedeću implementaciju:

```

IloExpr expr(env);
// pretp. da je niz c[] deklarisan
for(int i = 0; i < n; i++){

```

```

IloNumVar x = IloNumVar(env);
expr+= c[i] * x;
}
IloObjective obj = IloMaximize(env, expr);
model.add(obj); // dodavanje funkcije cilja u model

```

Implementacija ograničenja u modelu. Ograničenja se direktno mogu dodati u model uz pomoć funkcije *add()*, na primjer

```
model.add(expr1 <= expr2).
```

Umjesto relacije „ \leq ” mogu biti ravnopravno korištene i relacije „ $=$ ” ili „ \geq ”. Umjesto izraza *expr1* ili *expr2* takođe može da stoji i skalar. Pretpostavimo da želimo implementirati sljedeće ograničenje:

$$s_j = \sum_{t=1}^n t \cdot x_{jt}, \forall j \in \{1, \dots, N\}.$$

Implementacija koja odgovara ovim ograničenjima data je kôdom

```

// neka je prethodno definisan i popunjeno niz s[]
// neka su prethodno deklarisane promjenljive x[i][j]
for (j = 0; j < N; j++) {
    IloExpr expr(env);
    for (t = 1; t <= n; t++)
        expr += t * x[j][t];
    model.add(s[j] == expr);
    expr.end();
}

```

Još jedan od načina dodavanja ograničenja u model je implementacija preko kontejnera *IloRangeArray* u koji dodajemo ograničenja preko funkcije *add()*. Npr. pretpostavimo da je potrebno dodati ograničenja $x_i + x_j \leq 1$, za sve $i < j, i, j \in \{0, \dots, n - 1\}$ u *IloCplex* model

```

IloRangeArray constr(env);
// neka su prethodno definisane
// promjenljive x (IloNumVarArray)
for (i = 0; i < n-1; ++i)
    for(j = i+1; i < n; ++j)
        constr.add( x[i] + x[j] <= 1)
model.add(constr); //dodavanje ograničenja u model

```

Metode za izvoz rješenja (statistike) iz modela. Objasnimo najvažnije metode koje se koriste za dobijanje informacija o rješenju, statusu i ostalim parametrima samog izvršenja.

- Metod *getStatus()*, koji se poziva na instancu klase *IloCplex*, pokazuje da li je rješenje koje je nađeno dopustivo, ograničeno ili optimalno ili je CPLEX pokazao da je model nedopustiv ili neograničen.
- Metod *getObjValue()* vraća optimalnu vrijednost funkcije cilja (ili najbolju vrijednost nađenu u toku unaprijed definisanog intervala vremena).
- Metod *getValues(vals, var)*:
 - *vals* – numerički niz (najčešće instanca klase *IloNumArray*) u koju će se kopirati krajnja rješenja koje je CPLEX našao;
 - *var* – promjenljive u modelu.
- Metod *getDuals(vals, con)*:
 - *vals* – numerički niz (najčešće instanca klase *IloNumArray*) u koju će se kopirati rješenja odgovarajućih dualnih promjenjivih;
 - *con* – ograničenja u modelu (instanca klase *IloRangeArray*) za koje se računaju koeficijenti odgovarajućih dualnih promjenljivih;

-
- `getMIPRelativeGap()` vraća vrijednost relativne udaljenosti (realna vrijednost iz intervala $[0, 1]$) između najboljeg dopustivog rješenja (nadenog od strane CPLEX-a) i dokazane gornje granice optimalnog rješenja (koje su rezultat raznih internih tehnika – kao što je na primjer B&B – pozvane od strane rješavača).

Narednim kôdom je ilustrovana upotreba nekih od navedenih metoda.

```
// prethodno: kreiranje modela i poziv metoda solve()
env.out << cplex.getStatus() << endl;
IloNumArray xval(env); // ovdje ce se cuvati rjesenje
env.out() << "Status rjesenja = " << cplex.getStatus() << endl;
env.out() << "Vrijednost = " << cplex.getObjValue() << endl;
env.out() << "Vrijeme = " << timer.getTime() << endl;
cplex.getValues(xval, x);
env.out() << "x vrijednosti = " << xval << endl;
env.out() << "gap vrijednost: " << cplex.getMIPRelativeGap() * 100
// vraca gap (ako nije rjeseno do optimalnosti, onda > 0)
```

Izvoz modela za eventualno ponovno korištenje se može dobiti pozivom metoda `exportModel()`, čiji je očekivani argument naziv datoteke u koju će se model upisati (`.lp` ekstenzija).

Podešavanje parametara rješavača CPLEX. Podešavanje limita za vreme izvršavanja rješavača (recimo, na 2h) se definiše na sljedeći način

```
cplex.setParam(IloCplex::Param::TimeLimit, 7200)
```

dok recimo podešavanje broja jezgara pri izvršavanju CPLEX-a

```
cplex.setParam(IloCplex::Param::Threads, 1);
```

O ostalim parametrima kojima podešavamo rješavač može se pročitati iz CPLEX uputstva čija je internet adresa data među referencama na kraju ovog knjige.

Ukoliko za rad koristite Linux operativni sistem, za kompajliranje programa, koji je baziran na pozivanju CPLEX rješavača, neophodno je „uvućj“

ilocplex biblioteku, koja treba da se kompajlira sa glavnim programom. Prikaz odgovarajućeg *makefile*-a, koji sadrži neophodan kôd za uspješnu kompilaciju programa baziranih na pozivanju CPLEX rješavača, je dat u Apendiksu B.

10.2 PuLP alat

PuLP je Python-ova biblioteka visokog nivoa za implementaciju i rješavanje modela, sa fokusom na LP, ILP i MILP modele. Podržava veći broj komercijalnih i nekomercijalnih rješavača, a može se proširiti i na dodatne rješavače. Interesantno je i da PuLP ne zavisi od drugih softverskih paketa.

PuLP radi na principu modularnog pristupa prevođenja modela, napisanog prema definisanim pravilima, na njegovu vektorsku reprezentaciju (koja koristi vektore, rijetke matrice itd.). Konvertovani model se proslijeduje u interfejs klase nekog od rješavača. Kako je šablon mnogih rješavača sličan, ili se sa njima rukuje tako što se model predstavlja u standardnim *.lp* ili *.mps* formatima datoteka, pored specifičnih interfejsa popularnih rješavača, u PuLP biblioteku su uključene i generičke klase rješavača. Ove generičke klase se, uz minimalan trud, mogu korisnički proširiti u cilju pozivanja novih rješavača.

Da bismo koristili PuLP biblioteku, treba da je uvezemo na sljedeći način:

```
from pulp import *
```

Konkretno, ovaj paket se (u Linux-u) može instalirati pozivom: *sudo pip install pulp* ili *sudo apt-get install glpk-utils*. Kreiranje jedne instance problema se izvodi sljedećim kôdom:

```
prob = LpProblem("Naziv problema", LpMinimize)
```

Ako se rješava problem maksimizacije, drugi argument bi bio *LpMaximize*.

Implementacija promjenljivih u modelu. Za inicijalizaciju promjenljivih koristi se klasa *LpVariable*. Ona ima četiri atributa: prvi je naziv promjenljive, drugi označava donju granicu vrijednosti promjenljive, treći označava gornju granicu vrijednosti promjenljive, dok četvrti označava tip promjenljive. Tip promjenljive, koji se može odabratи je *LpContinuous* ili *LpInteger*, označavajući neprekidnu (realnu) i diskretnu promjenljivu, redom.

Npr. ako želimo deklarisati neprekidnu promjenljivu $x_1 \geq 0$, to radimo sljedećom linijom kôda:

```
x1 = LpVariable( "x1", 0, None, LpContinuous)
```

Implementacija funkcije cilja u modelu. Uz pomoć definisanih promjenljivih, koristeći operatore iz skupa $\{+, -, \cdot\}$, instanci problema dodajemo odgovarajuću funkciju koristeći operator $+=$. Primjera radi, ako želimo da modelujemo funkciju cilja $x_1 + x_2$, gdje su obje promjenljive neprekidne i pozitivne, to bismo uradili na sljedeći način

```
x1 = LpVariable("x1", 0, None)
x2 = LpVariable("x2", 0, None)
prob += (x1 + x2) # dodavanje funkcije cilja x1+x2
```

Implementacija ograničenja u modelu. Na sličan način kao i za funkciju cilja, ograničenja se u model dodaju pomoću operatora $+=$. Sintaksa ograničenja je bliska matematičkom zapisu ograničenja u modelu. Prvo se konstruišu izrazi koji su vezani nekim operatorom poređenja. Primjera radi, ako bismo u prethodni model željeli dodati ograničenja $x_1 + 2x_2 \leq 2$ i $x_1 - x_2 \geq 0$, koristimo sljedeći kôd

```
prob += x1 + 2 * x2 <= 2, "Ogranicenje 1"
prob += x1 - x2 >= 0, "Ogranicenje 2"
```

Upisivanje modela u *.lp* format se vrši pozivom funkcije *writeLP()*, čiji je (očekivani) argument naziv datoteke.

Metode za izvoz rješenja (statistike) iz modela. Rješavanje modela se inicira pozivom funkcije *solve()* na instancu modela. Kriterijumi za zaustavljanje izvršavanja rješavača postavljamo kao parametar metoda *solve()* uz pomoć sljedeće linije kôda

```
PULP_CBC_CMD(maxSeconds=1000, msg=1, fracGap=0)
```

koji kontroliše maksimalno vrijeme izvršavanja rješavača, poruke o napretku rješavača prilikom pronalaženja novih najboljih rješenja, te maksimalni željeni *gap* – nakon čijeg dostizanja rješavač prekida izvršavanje.

Napomenimo da je rješavač *COIN Branch and Cut* (CBC) difoltni (podrazumijevani) rješavač u PuLP modulu.

Čitanje svih promjenljivih iz modela se postiže pomoću funkcije *variables()*. Svaka promjenljiva za atribut *name*, te vrijednost, koju je dobila pri rješavanju modela nekim od rješavača (atribut *varValue*). Atribut *objective* instance modela čuva (najbolju) vrijednost funkcije cilja dobijene rješavanjem modela. Status rješavanja se dobija na osnovu atributa *status* objekta modela.

```
print "Status:", LpStatus[prob.status]
for v in prob.variables():
    print(v.name, "=", v.varValue)
print("Vrijednost funkcije cilja: ", value(prob.objective))
```

10.3 Primjena alata PuLP na rješavanje jednog problema lokacija sa ograničenim kapacitetima

Posmatrajmo već modelovan problem i riješimo ga uz pomoć PuLP rješavača. Model ovog problema je dat sa:

$$\begin{aligned} & \min \sum_{i=1}^n f_i x_i + \sum_{i,j} c_{ij} y_{ij} d_j \\ t.d \\ & \sum_i y_{ij} = 1, \forall j \in \{1, \dots, m\} \\ & \sum_j d_j y_{ij} \leq u_i x_i, \forall i \in \{1, \dots, n\} \\ & x_i \in \{0, 1\}, \forall i \in \{1, \dots, n\} \\ & y_{ij} \geq 0, \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\}. \end{aligned}$$

Ulagni parametri problema su konstante m, n , vektor f koji odgovara cijeni otvaranja postrojenja, vektor d koji odgovara potražnji svakog od kupaca, vektor u koji odgovara kapacitetu postrojenja, te matrica c_{ij} koja odgovara cijeni zadovoljenja potreba kupca j od strane postrojenja i (kao što je, recimo, cijena isporuke). Definišimo kroz program jednu instancu (obično se ona čita iz zasebne datoteke).

```
from pulp import *
locations = [1, 2, 3]
customers = [1, 2, 3]
f = { 1 : 50,
      2 : 32,
      3 : 40
}
u = {
      1 : 220,
      2 : 100,
```

```

            3 : 150
        }
        d = {
            1 : 25,
            2 : 44,
            3 : 48
        }
        c = {
            1 : { 1 : 2.0, 2 : 1.5, 3 : 0.75 },
            2 : { 1 : 1.75, 2 : 1.5, 3 : 2.2 },
            3 : { 1 : 1.1, 2 : 2.2, 3 : 2.1 }
        }
    # modelovanje:
    model = LpProblem("Lokacijski problem", LpMinimize)
    # promjenljive:
    x = LpVariable.dicts("x_vars", locations, 0, 1, LpBinary)
    y = LpVariable.dicts("y_vars", \
                         [ (i, j) for i in locations \
                           for j in customers ],\
                           0, None, LpContinuous)

    #funkcija cilja:
    model += lpSum( f[i] * x[i] for i in locations )\
        + lpSum( y[ (i, j) ] * c[ i ][ j ] * d[ j ] \
                  for i in locations for j in customers )

    #ograničenja:
    for j in customers:
        model += lpSum( y[ (i, j) ] for i in locations ) == 1
    for i in locations:
        model += lpSum( y[ (i, j) ] * d[ j ] for j in customers ) \
            <= u[ i ] * x[ i ]

    # solve:
    model.solve(PULP_CBC_CMD(maxSeconds=1000, msg=1, fracGap=0))
    # rješenja:
```

```

print("Otvorene lokacije su \n:")
for i in locations:
    if x[ i ].varValue > 0:
        print(str(i) + "\t")
print("y =>\n")
for i in locations:
    for j in customers:
        print(y[ (i, j) ].varValue)
print("Vrijednost: ", value(model.objective))

```

Optimalna vrijednost ove instance je 202.0, a dostiže se za vrijednost promjenljivih $x = (1, 0, 0)$, te $y_{11} = y_{12} = y_{13} = 1$, dok su ostale vrijednosti 0.

Čitaocu se preporučuje da na linku <https://www.programcreek.com/python/example/96845/pulp.LpProblem> pronađe i analizira još nekoliko problema, koji su riješeni uz pomoć PuLp modula.

10.4 Primjena CPLEX rješavača na problem najdužeg zajedničkog podniza bez ponavljanja

U ovoj sekciji modelujemo problem *najdužeg zajedničkog podniza bez ponavljanja*. Ovaj problem kao ulaz prihvata dva (proizvoljna) stringa s_1 i s_2 nad proizvoljnim alfabetom, a kao izlaz vraća string s maksimalne dužine sa sljedećim karakteristikama:

- s je zajednički podniz oba stringa;
- s ne sadrži karakter koji se pojavljuje više od jednom u njemu samom.

Za ovaj problem je poznato da je NP-težak, ili preciznije pripada \mathcal{APX} klasi. Zainteresovanom čitaocu preporučujemo da se, koristeći se internet resursima, upozna sa specifičnostima \mathcal{APX} klase problema.

Npr. neka su data dva stringa $s_1 = \text{abccccab}$ i $s_2 = \text{abccacd}$, rješenje ovog problema je $s = \text{abc}$. Primijetimo da je s zajednički podniz za oba

ulazna stringa te da ispunjava uslov o „neponavljačim” karakterima u sebi. Takođe, može se provjeriti da ne postoji duži zajednički podniz sa takvim svojstvom za ove ulazne stringove, što znači da je ovo rješenje optimalno. Konstruišimo sada jedan model binarnog cjelobrojnog programiranja ovog problema.

Nazovimo par (a, b) , $1 \leq i \leq |s_1|$, $1 \leq j \leq |s_2|$ odgovarajući akko je $s_1[i] = s_2[j]$, $a < b$. Dalje, sa \mathcal{M} označimo skup svih odgovarajućih parova karaktera za dati ulaz $S = \{s_1, s_2\}$. Takođe, označimo sa $\mathcal{M}_a \subseteq \mathcal{M}$ skup svih odgovarajućih parova koji pokrivaju slovo a , tj. $\mathcal{M}_a = \{(i, j) \mid s_1[i] = s_2[j] = a\}$, $a \in \Sigma$. Svakom odgovarajućem paru $p = (p_1(a), p_2(a))$, pridružimo jednu binarnu promjenljivu $z_p \in \{0, 1\}$. Označimo $l(z_p) = s_1[p_1(a)]$, $Z_a = \{z_p \mid p \in \mathcal{M} \wedge l(z_p) = a\}$, $a \in \Sigma$, te skup svih promjenljivih sa $Z = \cup_{a \in \Sigma} Z_a$.

Za svake dvije binarne promjenljive z_q i z_r kažemo da su u *konfliktu* akko vrijedi sljedeće

$$q_1(a) \geq r_1(b) \text{ i } q_2(b) \leq r_2(a)$$

ili

$$q_1(a) \leq r_1(b) \text{ i } q_2(b) \geq r_2(a)$$

ili

$$l(z_{p_1}) = l(z_{p_2}).$$

Sada možemo formirati ograničenja u modelu. Uslov da je svaki karakter prisutan najviše jednom u rješenju problema se modeluje sa

$$\sum_{z \in Z_a} z \leq 1, \forall a \in \Sigma \quad (10.1)$$

Takođe, s obzirom da rješenje problema podrazumijeva pronalazak skupa (međusobno) nekonfliktnih odgovarajućih parova karaktera (odgovarajućih z -promjenljivih) maksimalne kardinalnosti, dodajemo ograničenje

$$z_p + z_q \leq 1, \forall p, q \in Z, p \neq q, p \text{ je u konfliktu sa } q. \quad (10.2)$$

Kako tražimo skup maksimalne kardinalnosti, funkciju cilja (koja se maksimizuje) je data sa:

$$\max \sum_{z \in Z} z \quad (10.3)$$

Dakle, uz funkciju cilja (10.3), ograničenja (10.1)–(10.2) sa uslovom $z \in \{0, 1\}$, $z \in Z$, dobijamo model problema najdužeg stringa bez ponavljanja.

Implementirajmo ovaj model, uz pretpostavku da je svaki string u ulazu narednog programa dat svojom numeričkom reprezentacijom. To znači da je svakom karakteru stringa pridružen jedinstven broj i kao takav se importuje i koristi u programu. Primjera radi, za string $s_1 = \text{abbcba}$, pridružen je numerički niz (vektor) 011210.

Model je implementiran u programskom jeziku C++.

```
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <limits>
#include <unordered_map>
#include <map>

#include "path-to-ilocplex.h" // Ovdje ide staza do ilocplex.h

ILOSTLBEGIN
struct Point2d // point for structure
{

    public:
    int a,b;
    Point2d(): a(0), b(0){}; // default constructor...
    Point2d(int _a,int _b): a(_a), b(_b){};

    bool operator == (const Point2d & p2) const
    {
        return (a == p2.a and b == p2.b);
    }
    friend std::ostream& operator << (std::ostream & os, const
    Point2d& p)
```

```
{  
    os<< "(" << p.a << ", " << p.b << ")" " << endl;  
    return os;  
  
};  
  
class Hash2d //heširanje 2D tačaka  
{  
    // radi za stringove dužine <= 5000 (može se adaptirati)  
    public:  
        std::size_t operator()(const Point2d & pl ) const  
    {  
        return 5000*pl.a + pl.b;  
    }  
};  
  
bool conflict(const Point2d& p1, const Point2d& p2)  
{  
    if( ( p1.a > p2.a and p1.b > p2.b ) or  
        ( p1.a < p2.a and p1.b < p2.b ) )  
        return false;  
    return true;  
}  
  
int main( int argc, char **argv )  
{  
    vector<int> s1;  
    vector<int> s2;  
    // import stringova u s1 i s2 (numericka prezentacija)  
    IloEnv env;  
    env.setOut(env.getNullStream());  
    try  
    {
```

```
cout << "model creation..." << endl;
IloModel model(env);
int vars_num = 0;
IloObjective obj = IloMaximize(env);
// definisanje Z promjenljivih
unordered_map<Point2d, IloNumVar, Hash2d> Z;
for (int i = 0; i < s1.size(); ++i){
    for (int j = i+1; j < s2.size(); ++j)
    {
        if( s1[i] == s2[j] )
        {
            Point2d p(i, j);
            IloNumVar myIntVar(env, 0, 1, ILOINT);
            Z[p] = myIntVar;
            obj.setLinearCoef(Z[p], 1.0); // funkcija cilja
            ++vars_num;
        }
    }
}
map<int, IloExpr> cs;
// repetition-free ograničenja:
for(unordered_map<Point2d, IloNumVar, Hash2d>::iterator itx
= Z.begin(); itx != Z.end(); ++itx)
{
    if(cs.find(s1[(*itx).first.a]) == cs.end() ){
        IloExpr xpr(env);
        cs.insert({s1[(*itx).first.a], xpr });
    }
    cs[s1[(*itx).first.a]] += (*itx).second;
}
for(int i = 0; i < alphabet_size; i++)
    if(cs.find(i) != cs.end())
        model.add(cs[i] <= 1);
```

```
for(unordered_map<Point2d, IloNumVar, Hash2d >
::iterator it1 = Z.begin(); it1 != Z.end(); ++it1)
{
    for(unordered_map<Point2d, IloNumVar, Hash2d >
::iterator it2 = Z.begin(); it2 != Z.end(); ++it2)
    {
        if(conflict( (*it1).first, (*it2).first ) and
           ! ((*it1).first == (*it2).first) ){
            // conflict constraints
            model.add( (*it1).second + (*it2).second <= 1);
        }
    }
}
model.add(obj);
IloCplex cplex(model);
int time_limit = 900;
//pass the time limit to \text{Cplex}
cplex.setParam(IloCplex::TiLim, time_limit);
IloNum lastObjVal = std::numeric_limits<double>::max();
cplex.solve();
// rjesenja:
if (cplex.getStatus() == IloAlgorithm::Optimal
    or cplex.getStatus() == IloAlgorithm::Feasible)
{
    if(cplex.getStatus() == IloAlgorithm::Optimal)
        cout << "Cplex je našao optimum" << endl;
    else
        cout << "Cplex je našao dopustivo rješenje" << endl;

    double lastVal = double(cplex.getObjValue());
    // printaj rješenja
    cout << "komponente u rješenju: {" << endl;
    bool first = true;
    for(unordered_map<Point2d, IloNumVar, Hash2d >::iterator
```

```
    it = Z.begin(); it != (Z.end()); ++it)
{
    IloNum xval = cplex.getValue((*it).second);
    if (xval > 0.9) {
        cout << (*it).first;
    }
}
cout << " } \n";
}
cout << "Objektivna vrijednost: " << lastVal << endl;
cout << "Dualna vrijednost: "
     << double(cplex.getBestObjValue()) << endl;
catch(IloException& e) {
    cerr << " ERROR: " << e << endl;
}
env.end();
}
}
```

10.5 Zadaci

1. Istražiti na internetu dostupne rješavače za probleme linearног, cje-ljivo-рног (i mješovitog cjelobrojno-linearног) programiranja. Proci-jeniti njihov kvalitet, upotrebnu vrijednost i prepoznati prednosti i nedostatke svakog od njih.
2. Istražiti mogućnosti PuLP biblioteke, na stranici <https://coin-or.github.io/pulp/main/index.html> i drugim relevantnim internet resursima.
3. Koristeći neki od rješavača, provjeriti ispravnost rješenja zadataka iz Poglavlja 4 i 6.
4. Neka je na ulazu dat graf $G = (V, E)$ kao instanca problema. Imple-

mentirati sljedeći ILP model

$$\begin{aligned} \max \sum_{v \in V} x_v \\ \text{t.d.} \\ x_{v'} + x_{v''} \leq 1, \forall (v' v'') \in E \\ x_v \in \{0, 1\}, \forall v \in V. \end{aligned}$$

Eksportovati rješenje problema u obliku skupa odabralih čvorova koji su komponenete optimalnog rješenja. Riješiti relaksaciju datog ILP-a. Zaokružiti dobijeno rješenje, tj. transformisati ga u najbliže cjelobrojno rješenje (gledajući po koordinatama). Model implementirati:

- (a)) uz pomoć CPLEX-a;
 - (b) uz pomoć PuLP modula.
5. Neka je dat usmjeren težinski graf $G = (V_1 \cup V_2, E)$, gdje su V_1 i V_2 disjunktni skupovi. Neka je dat prirodan broj $p \geq 1$. Težine grana između čvorova i i j je data sa $d_{i,j} \geq 0$ i postoje samo grane između čvorova iz skupa V_1 u skup V_2 . Svakom čvoru iz skupa V_1 pridružena je težina h_i . Neka je MILP model dat sa:

$$\begin{aligned} \min \sum_{j \in V_2} \sum_{i \in V_1} h_i d_{i,j} Y_{i,j} \\ \text{t.d.} \\ \sum_{j \in V_2} Y_{i,j} = 1, \forall i \in V_1 \\ Y_{i,j} - X_j \leq 0, \forall i \in V_1, \forall j \in V_2 \\ \sum_{i \in V_2} X_j = p \\ X_j \in \{0, 1\}, \forall j \in V_2 \\ Y_{i,j} \in \{0, 1\}, \forall i \in V_1, \forall j \in V_2. \end{aligned}$$

Model implementirati

-
- (a)) uz pomoć CPLEX-a;
- (b) uz pomoć PuLP modula.
6. Neka je na ulazu dat skup V poslova koji treba da budu urađeni na mašini. Svaki posao j je dat intervalom $[r_j, d_j]$, $r_j < d_j$, gdje r_j označava dopustivo vrijeme puštanja posla u rad (na mašini) dok d_j označava vrijeme do kada se posao mora završiti. Vrijeme procesiranja posla j je dato sa p_j . Odrediti za sve poslove $j \in V$ dopustiv redoslijed njihovog izvršavanja (na mašini) tako što treba odrediti startna vremena izvršenja s_j ($s_j \geq r_j \wedge s_j + p_j \leq d_j$) za sve $j \in V$. Problem je modelovan na sljedeći način

$$\begin{aligned} & \max D - R \\ & \text{t.d.} \\ & R \geq r_i, \forall i \in V \\ & D \leq d_i, \forall i \in V \\ & R \geq r_i + \sum_{\{j \in V \setminus \{i\} | r_j \geq r_i\}} p_j x_j^+, \forall i \in V \\ & D \leq d_i - \sum_{\{j \in V \setminus \{i\} | d_j \leq d_i\}} p_j x_j^-, \forall i \in V \\ & x_j^+ + x_j^- = 1, \forall i \in V \\ & x_j^+, x_j^- \in \{0, 1\}, \forall i \in V \\ & D, R \in \mathbb{Z}. \end{aligned}$$

Model implementirati

- (a)) uz pomoć CPLEX-a;
- (b) uz pomoć PuLP modula.
7. Neka je data instanca koja se sastoji n brojeva iz skupa \mathbb{R}^p , $p \in \mathbb{N}$. Sa $w_i = (w_{i1}, \dots, w_{ip})$ označimo i -ti element u skupu S . Neka je $s_j =$

$\sum_{i=1}^n w_{ij}, j = 1, \dots, p$ i neka je dat model

$$\min t$$

t.d.

$$-0.5 \cdot t + \sum_{i=1}^n w_{i,j} x_i \leq 0.5 \cdot s_j, j = 1, \dots, p$$

$$0.5 \cdot t + \sum_{i=1}^n w_{i,j} x_i \geq 0.5 \cdot s_j, j = 1, \dots, p$$

$$x_j \in \{0, 1\}, j = 1, \dots, n.$$

Model implementirati

(a)) uz pomoć CPLEX-a;

(b) uz pomoć PuLP modula.

8. Neka je dat $G = (V, E)$ kao instanca problema. Neka je dat ILP model

$$\min \sum_{i \in V} x_i + \sum_{i \in V} y_i$$

t.d.

$$x_i + \sum_{j \in N_i} y_j \geq 1, i \in V$$

$$y_i \leq x_i, i \in V$$

$$x_i, y_i \in \{0, 1\}, i \in V,$$

gdje N_i označava susjede čvora i u grafu G .

Model implementirati

(a) uz pomoć CPLEX-a;

(b) uz pomoć PuLP modula.

Dodatak A

Metod unutrašnje tačke

U ovoj sekciji ćemo prikazati konstrukciju *Metoda unutrašnje tačke*, koji bilo koji problem LP-a rješava u polinomijanom vremenu.

Metod unutrašnje tačke (eng. *Interior point method*) je dizajniran 50-ih godina XX vijeka. Sve do 80-ih godina XX vijeka nije smatran algoritmom koji efikasno rješava probleme velikih dimenzija. Tek 1984. godine, N. Karmakar, inžinjer u IBM-u, formalno dokazuje da se ovaj algoritam izvršava u polinomijalnom broju koraka za unaprijed zadatu preciznost optimalne vrijednosti. U današnje vrijeme, zbog ekstremnog poboljšanja performansi mašina, pokazalo se da je metod unutrašnje tačke kompetentan sa simpleks metodom u situacijama kada se radi o problemima izuzetno velikih dimenzija, gdje matrice problema sadrže 10^9 i više elemenata.

Za razliku od simpleks metoda, koji „šeta” po granicama poliedra (dovoljstivog regiona) u potrazi za optimalnim rješenjem, metod unutrašnje tačke, kako i samo ime kaže, se kreće po unutrašnjim tačkama poliedra, dok ne nađe na optimalno rješenje.

Postoji nekoliko metoda ove vrste, kao što su: metod centralne putanje, redukcija potencijala, afino skaliranje, itd. U ovoj sekciji ćemo razmatrati metod centralne putanje. Napomenimo i da ćemo se baviti rješavanjem problema LP-a koji je dat u obliku (4.1). Napomenimo da razumijevanje procesa konstrukcije metode unutrašnje tačke zahtjeva poznavanje osnov-

nih pojmova i teorema klasične matematičke analize sa više promjenjivih, kao i iterativnih numeričkih metoda za rješavanje (nelinearnih) sistema jednačina. Dalje, kako je konstrukcija ovog algoritma iznimno zahtjevna, neke teoreme, ili dijelovi teorema koji koriste ovoj svrsi, će biti navedeni bez dokaza i užaženja u detalje. Dakle, više ćemo se fokusirati na ideju i razumijevanje glavnih koraka u konstrukciji ovog algoritma.

Neka je par Primal-dual linearnih programa dat sa

$$\begin{array}{ll} \text{Primal} & \text{Dual} \\ \min c^T x & \max b^T y \\ \text{t.d. } Ax = b & \text{t.d. } A^T y + s = c \\ x \geq 0 & s \geq 0 \end{array}$$

Definicija A.1. Lagranžijan primal-dual linearnih programa se definiše sa

$$L(x, y) = c^T x - y^T (Ax - b) - s^T x. \quad (\text{A.1})$$

Postoji niz teorema koje obezbeđuju dovoljne uslove za nalazak optimalnog rješenja LP-a. U nastavku navodimo najvažnije od njih.

Teorema A.1. Prepostavimo da postoje $y \in \mathbb{R}^m, s \in \mathbb{R}^n$ tako da je

$$\begin{aligned} A^T y + s &= c \\ s &\geq 0. \end{aligned}$$

Ako je x dopustivo rješenje za Primal, onda vrijedi

$$c^T x - b^T y = s^T x \geq 0.$$

Dokaz. Prepostavimo da je x dopustivo za Primal (P). Tada vrijedi

$$c^T x = (A^T y + s)^T x = s^T x + y^T b \geq b^T y.$$

Dakle, pokazali smo tvrdnju. Štaviše, vrijednost $b^T y$ je donja granica za optimalnu vrijednost Primal-a. Takođe, ako je $s^T x = 0$ donja granica je dostignuta, te je x je optimalno rješenje Primal-a. \square

Teorema A.2. Prepostavimo da Primal ima barem jedno optimalno rješenje. Vektor $x \in \mathbb{R}^n$ je optimalno rješenje za Primal akko postoji $y \in \mathbb{R}^m, s \in \mathbb{R}^n$ tako da

$$Ax = b \quad (\text{A.2})$$

$$x \geq 0 \quad (\text{A.3})$$

$$A^T y + s = c \quad (\text{A.4})$$

$$s \geq 0 \quad (\text{A.5})$$

$$s^T x = 0. \quad (\text{A.6})$$

Optimalna vrijednost je data sa $b^T y$.

Dokaz. Za potrebne uslove, neka je \tilde{x} optimalno rješenje za Primal. Zbog uslova dopustivosti za Primal, treba da vrijedi (A.2) i (A.3). Iz Teoreme 5.4, imamo da je optimalna vrijednost Primala jednaka $b^T y$ te dodatno da je \tilde{x} optimalno samo ako je $s^T \tilde{x} = 0$, pa vrijedi i (A.6).

Dovoljni uslovi se mogu pokazati na osnovu koraka konstrukcije simpleks metode i teoreme o komplementarnosti. Ovaj dokaz ne navodimo ovdje, jer je tehnički dosta zahtjevan.

Prema tome, ako postoji x, y, s koji zadovoljavaju uslove optimalnosti, onda je x optimalno rješenje. \square

Drugi način da se dođe do uslova optimalnosti posmatrati Lagranžijan L para Primal-Dual i naći stacionarne tačke za L , odakle nalazimo optimalno rješenje oba problema. Uslovi za nalaženje stacionarne tačke se dobijaju iz sistema

$$\nabla_x L(x, y) = 0 \quad (\text{A.7})$$

$$\nabla_y L(x, y) = 0, \quad (\text{A.8})$$

što nas, uz uslov komplementarnosti za x i y , ponovo dovodi do sistema

$$Ax = b \quad (\text{A.9})$$

$$A^T y + s = c \quad (\text{A.10})$$

$$XS\mathbf{1} = 0 \quad (x_j \cdot s_j = 0, \forall j) \quad (\text{A.11})$$

$$(x, s) \geq 0, \quad (\text{A.12})$$

gdje je $X = \text{diag}(x_1, \dots, x_n)$, $S = \text{diag}(s_1, \dots, s_m)$, $\mathbf{1} = (1, \dots, 1)$. Ovaj sistem nazivamo *uslov optimalnosti prvog reda za primal-dual par*.

U nastavku navodimo pojam barijernog problema preko kojeg razvijamo metod unutrašnje tačke. Zamijenimo ograničenja nenegativnosti u primalnom problemu sa $-lnx_j$, $j = 1, \dots, n$. Tada vrijedi

$$\min e^{-\sum_{j=1}^n lnx_j} \Leftrightarrow \max \prod_j x_j$$

Dakle, minimizacija izraza $-\sum_{j=1}^n x_j$ je ekvivalentna maksimizaciji proizvoda udaljenosti od svih hiperravnih koje definišu prvi ortant (skup svih vektora kojima su koordinate nenegativne). Dakle, ovako definisan optimizacioni problem će onemogućiti da se vrijednost neke od promjenljivih x_j približi nuli (uprkos nenegativnim ograničenjima). Na osnovu ovih činjenica, definišimo sljedeći optimizacioni problem.

Definicija A.2. *Barijerni program primala je dat u obliku*

$$\begin{aligned} \min & c^T x - \mu \sum_{j=1}^n ln(x_j) \\ \text{t.d. } & Ax = b \\ & x \geq 0 \end{aligned}$$

za neki $\mu > 0$. Lagranžian barijernog problema je dat sa

$$L(x, y, \mu) = c^T x - \mu \sum_{j=1}^n ln(x_j) - y^T (Ax - b)$$

Sljedeća teorema koja razmatra skalarne funkcije sa više promjenjivih sa uslovnim ekstremima je važna u daljem razmatranju; navodimo je bez dokaza.

Teorema A.3. *Neka je data skalarna funkcija $f : \mathbb{R}^n \mapsto \mathbb{R}$ na skupu $g_1(x) = g_2(x) = \dots = g_m(x) = 0$ koja ima minimum u x^* . Onda vrijedi*

$$\nabla f(x^*) = \sum_{i=1}^m y_i \nabla g_i(x^*),$$

gdje se y_i nazivaju Lagranžovi množitelji.

Primijetimo da u barijarnom problemu imamo sljedeće:

- $f(x) = c^T x - \mu \sum_{j=1}^n \ln(x_j)$;
- $g_i(x) = a_i x - b_i$.

Izračunajmo Lagranžove množitelje ovih funkcija. Vrijedi:

$$c + \mu \left(\frac{1}{x_1}, \frac{1}{x_2}, \dots, \frac{1}{x_n} \right) = \nabla f(x) = \sum_{i=1}^m y_i \nabla g_i(x) = \sum_i y_i a_i = A^T y.$$

Uvedimo nenegativni vektor $s = \mu \cdot \left(\frac{1}{x_1}, \frac{1}{x_2}, \dots, \frac{1}{x_n} \right)$, pa imamo da bilo koje optimalno rješenje početnog problema treba da zadovoljava *uslove optimalnosti prvog reda za barijerni problem*:

$$\begin{aligned} Ax &= b \\ A^T y + s &= c \\ (s_1 x_1, \dots, s_m x_m) &= (\mu, \dots, \mu) \\ x, s &\geq 0. \end{aligned} \tag{A.13}$$

Primijetimo da ovaj uslov aproksimira iste uslove za primal-dual par. Interesantno je da parametar μ kontroliše udaljenost od optimalnosti, tj.

$$c^T x - b^T y = c^T x - x^T A y = x^T (c - A^T y) = x^T s = n\mu.$$

Primijetimo da uslovi (A.13) ne daju linearni sistem, jer treće ograničenje nije linearno, što otežava rješavanje sistema.

Definicija A.3. *Primal-dual centralni put $\{(x(\mu), y(\mu), s(\mu)) \in \mathbb{R}^{2n+m} \mid \mu > 0, x(\mu) > 0, s(\mu) > 0\}$ je (jedinstveno) rješenje sistema (A.13) parametrizovanog sa μ .*

Primijetimo sljedeće, ako se u problemu (A.13) stavi $\mu = 0$, onda imamo

$$\begin{aligned} (s_1 x_1, \dots, s_m x_m) &= (0, \dots, 0) \\ \iff s^T x &= 0 \iff (A^T y - c)^T x = 0 \\ \iff y^T A x - c^T x &= 0 \iff y^T b = c^T x \end{aligned}$$

jer je $Ax = b$. Pimijetimo da vrijednost (optimalnih) dualnih promjenjivih y upravo odgovara vrijednosti Lagranžovih množitelja. Takođe, kako μ teži ka nuli, rješenja (nelinearnog) sistema (A.13) sve preciznije aproksimira JU rješenje početnog problema (4.1).

Da bi došli do koraka samog algoritma, uvedimo notaciju

$$F(x, y, s) = \begin{pmatrix} Ax - b \\ A^T y + s - c \\ X S \mathbf{1} \end{pmatrix},$$

gdje je $X = \text{diag}(x_1, \dots, x_n)$, $S = \text{diag}(s_1, \dots, s_n)$ i $\mathbf{1} = \text{diag}(1, \dots, 1)$. Primijetimo da je $X S \mathbf{1} = \sum_i s_i x_i$. Potrebno je riješiti $F(x, y, s) = 0, x, s \geq 0$. Korjen ove (nelinearne) jednačine ćemo naći uz pomoć *Njutnovog metoda* (za nelinearni sistem jednačina).

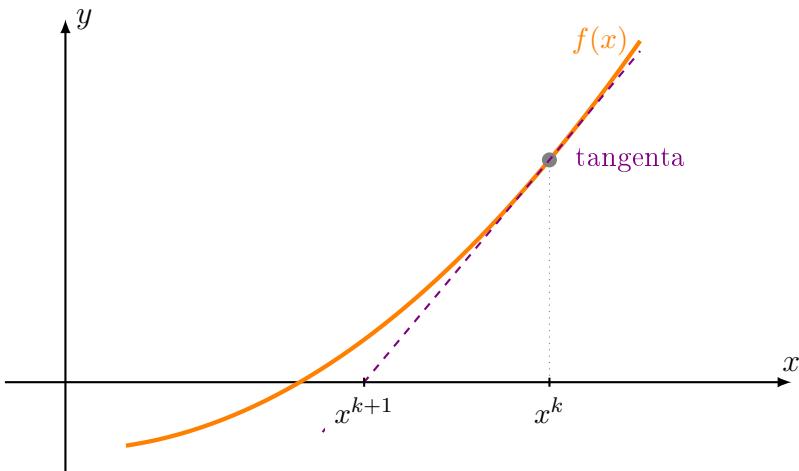
U nastavku u sažetom obliku izlažemo ideju Njutnovog metoda, jednog od najefikasnijih metoda za rješavanje ovakvih (nelinearnih) sistema jednačina. Neka je data diferencijalna funkcija $f : \mathbb{R}^n \mapsto \mathbb{R}$. Tangentna linija

$$z - f(x^k) = \nabla f(x^k)(x - x^k)$$

je lokalna aproksimacija grafa funkcije $f(x)$ oko date tačke. Stavajući $z = 0$, definiše se nova tačka

$$x^{k+1} = x^k - (\nabla f(x^k))^{-1} f(x^k). \quad (\text{A.14})$$

Pokazuje se da niz $\{x^k\}_{k \in \mathbb{N}}$ definisan na osnovu (A.14) konvergira ka stacionarnoj tački funkcije $f(x)$. Simulacija koraka Njutnovog metoda je data na Slici A.1.



Slika A.1: Njutnova metoda u ravni.

Rješavanje nelinearnih sistema se može generalizovati na funkcije više promjenjivih.¹ Primijenimo ovu metodu na nalaženje nule funkcije $F(x, y, s)$. Tada imamo

$$\nabla F = \begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{pmatrix}$$

Prema tome, u fiksnoj tački (x, y, s) , Njutnov pravac (pomjeraj) $\begin{pmatrix} \Delta x \\ \Delta y \\ \Delta s \end{pmatrix}$ se izračunava rješavajući sistem linearnih jednačina

¹Za više informacija pogledati na <http://www.ohiouniversityfaculty.com/youngt/IntNumMeth/lecture13.pdf>.

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta s \end{pmatrix} = \begin{pmatrix} b - Ax \\ c - A^T y - s \\ -XS\mathbf{1} \end{pmatrix} \quad (\text{A.15})$$

iz koga se potom računaju tačke naredne iteracije. Njutnov metod u nekoj iteraciji obično narušava uslove nenegativnosti $x, s \geq 0$, ali to se može izbjegći kako će biti prikazano u nastavku. Prethodno se može lako proširiti i na slučaj kada ne vrijedi $s_i x_i = 0$, ili preciznije kada je $\frac{1}{n} s_i x_i = \mu, \mu > 0$, $i = 1, \dots, n$. Na sličan način računamo Njutnov pravac kada je $s_i x_i \approx \mu\theta$ za $\theta > 0$, iz sistema

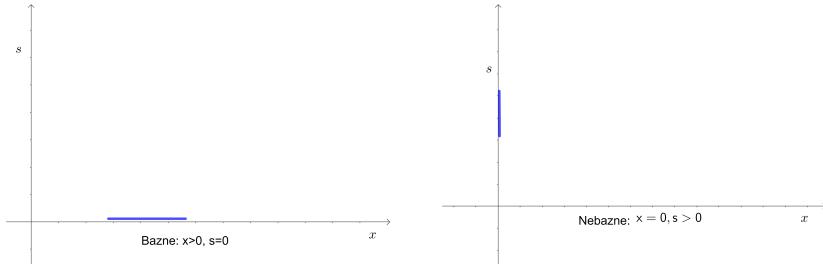
$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{pmatrix} \cdot \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta s \end{pmatrix} = \begin{pmatrix} b - Ax \\ c - A^T y - s \\ \theta\mu\mathbf{1} - XS\mathbf{1} \end{pmatrix} \quad (\text{A.16})$$

Prema tome, *algoritam unutrašnje tačke* izvršava sljedeće korake:

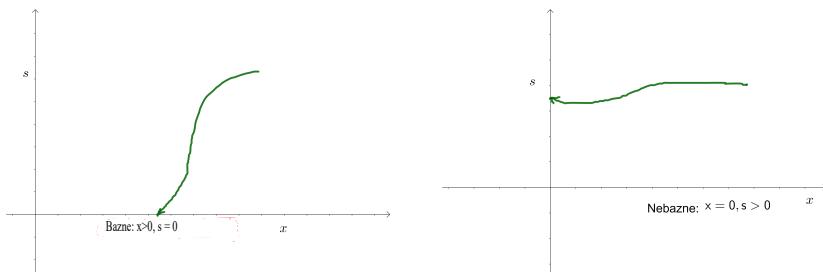
1. Inicijalizujemo $k = 0$, te proizvoljno $(x^k, y^k, s^k), x^k, s^k \geq 0, x^k, s^k \in \mathbb{R}^n, y^k \in \mathbb{R}^m$.
2. Izaberimo $\theta^k \in [0, 1)$ te riješimo sistem (A.16) za (x^k, y^k, s^k) i odabran $\theta = \theta_k$ te $\mu = \mu^k = (y^k)^T s^k$, odakle dobijamo $\begin{pmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{pmatrix}$
3. $\begin{pmatrix} x^{k+1} \\ y^{k+1} \\ s^{k+1} \end{pmatrix} \approx \begin{pmatrix} x^k \\ y^k \\ s^k \end{pmatrix} + \alpha_k \begin{pmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{pmatrix}$, gdje je α_k izabran na taj način da $(x^{k+1}, s^{k+1}) > 0$, tj. da bi se očuvala dopustivost rješenja.
4. Ažurirajmo $k = k + 1$, te se vratimo na korak 2.

Primijetimo da $\mu \rightarrow 0$, kako $k \rightarrow \infty$. U praksi se često stavlja $\theta^k = 1 - \frac{\beta}{n}$, za $\beta \in (0, 1)$, a najčešće $\beta = 0.1$. Parametar β upravlja brzinom progresa prema optimalnom rješenju.

Sljedeća teorema karakteriše *metod unutrašnje tačke*.



Slika A.2: Prilazak optimalnosti kod Simpleks metode.



Slika A.3: Prilazak optimalnosti kod Metoda unutrašnje tačke.

Teorema A.4. Za metod unutrašnje tačke vrijedi sljedeće:

- Algoritam unutrašnje tačke konvergira ka optimalnom rješenju problema (4.1).
- U svim iteracijama, algoritam generiše dopustive tačke.
- Kompleksnost algoritma je $O(\sqrt{n})$.

Da bi se uvidjeli sličnosti i razlike između Simpleks metode i Metode unutrašnje tačke, potrebno je analizirati Slike A.2 i A.3. U njima se mogu vidjeti kako ova dva metoda pristupaju pretrazi optimalnih rješenja.

Paralelizovani objektno orjentisani optimizacioni softver baziran na metodi unutrašnje tačke se može naći na linku koji je dat pod referencom [25].

Dodatak B

Makefile za kompajliranje CPLEX programa

Za pokretanje progama uz pomoć *makefile* datoteke (komande *make*), potrebno je pratiti sljedeće korake.

- Instalirati C++, sa kompajlerom GCC (verzija 9, recimo).
- Intalirati je CPLEX.
- U promjenljivim CPLEXDIR i CONCERTDIR (linije 8 i 9), poresiti putanje ka cplex i concert direktorijumima, redom.
- Napisati program cplex_rfrcs.cpp i dodatni .cpp za mjerenje vremena izvršavanja algoritma, nazvan Timer.cpp. (Pisanje Timer.cpp nije obavezno. U tom slučaju sadržaj promjenljive OBJS na liniji 3 je prazan.)

```
1 TARGET = cplex_rflcs
2 CXXFLAGS = -ansi -O3
3 OBJS = Timer.o #Random.o
4
5 SYSTEM      = x86-64_linux
6 LIBFORMAT   = static_pic
7
8 CPLEXDIR     = /home1/share/ILOG/cplex-12.7.0/cplex#path to the cplex dir
9 CONCERTDIR   = /home1/share/ILOG/cplex-12.7.0/concert#path to the concert dir
10 CCC = g++ -std=c++11 #g++
11 CCOPT = -m64 -O -fPIC -fexceptions -DNDEBUG -DIL_STD
12 CPLEXBINDIR = $(CPLEXDIR)/bin/$(BINDIST)
13 CPLEXLIBDIR = $(CPLEXDIR)/lib/$(SYSTEM)/$(LIBFORMAT)
14 CONCERTLIBDIR = $(CONCERTDIR)/lib/$(SYSTEM)/$(LIBFORMAT)
15 CCLNFLAGS = -L$(CPLEXLIBDIR) -lilocplex -lcplex -L$(CONCERTLIBDIR) -lconcert -lm -pthread
16 CLNFLAGS = -L$(CPLEXLIBDIR) -lcplex -lm -pthread
17 CONCERTINCDIR = $(CONCERTDIR)/include
18 CPLEXINCDIR = $(CPLEXDIR)/include
19 CCFLAGS = $(CCOPT) -I$(CPLEXINCDIR) -I$(CONCERTINCDIR)
20
21 all: ${TARGET}
22
23 cplex_rflcs: cplex_rflcs.o ${OBJS}
24         $(CCC) $(CCFLAGS) cplex_rflcs.o ${OBJS} -o cplex_rflcs $(CCLNFLAGS)
25
26 cplex_rflcs.o: cplex_rflcs.cpp
27         $(CCC) -c $(CCFLAGS) cplex_rflcs.cpp -o cplex_rflcs.o
28
29 clean:
30         @rm -f *~ *.o ${TARGET} core
31
```

Slika B.1: Prikaz makefile-a za kompajliranje CPLEX programa

Bibliografija

- [1] M. Leila, The big picture of Operations Research, preuzeto 24.7.2022. <https://towardsdatascience.com/the-big-picture-of-operations-research-8652d5153aad>
- [2] Hillier, Frederick S. Introduction to operations research. Tata McGraw-Hill Education, 2012.
- [3] M. Lavrov, Linear programming, skripta, preuzeto 24.7.2022. <https://faculty.math.illinois.edu/~mlavrov/>
- [4] Bisschop, Johannes. AIMMS optimization modeling. Lulu.com, 2006.
- [5] Feillet, Dominique. „A tutorial on column generation and branch-and-price for vehicle routing problems.” 4or 8.4 (2010): 407-424.
- [6] Rebeka Čordaš. Linearno programiranje i primjene. Sveučilište J.J.Strossmayera u Osijeku, diplomski rad (2012)
- [7] Manual, CPLEX User'S. „Ibm ilog cplex optimization studio.” Version 12 (1987): 1987-2018.
- [8] Mitchell, Stuart, Michael OSullivan, and Iain Dunning. "PuLP: a linear programming toolkit for python."The University of Auckland, Auckland, New Zealand (2011): 65.

-
- [9] Talbi, El-Ghazali. Metaheuristics: from design to implementation. Vol. 74. John Wiley & Sons, 2009.
 - [10] Vazirani, Vijay V. Approximation algorithms. Springer Science & Business Media, 2013.
 - [11] Blum, Christian, Maria J. Blesa, and Manuel Lopez-Ibanez. „Beam search for the longest common subsequence problem.” Computers & Operations Research 36.12 (2009): 3178-3186.
 - [12] Raidl, Günther R., and Jakob Puchinger. „Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization.” Hybrid metaheuristics (2008): 31-62.
 - [13] Rahamanian, Ragheb, et al. "The Benders decomposition algorithm: A literature review." European Journal of Operational Research 259.3 (2017): 801–817.
 - [14] Raidl, Günther R. „Decomposition based hybrid metaheuristics.” European journal of operational research 244.1 (2015): 66-76.
 - [15] Mladenović, N., and Hansen P. „Variable neighborhood search.” Computers & Operations Research 24.11 (1997): 1097-1100.
 - [16] CPLEX User Manual, preuzeto 24.7.2022. <https://www.tu-chemnitz.de/mathematik/discrete/manuals/cplex/doc/userman/html/cpxUserAppendixA.html>
 - [17] Mitchell, S., OSullivan, M., and Dunning, I. (2011). PuLP: a linear programming toolkit for python. The University of Auckland, Auckland, New Zealand, 65.
 - [18] S. A. Mitchell and J.S. Roy. PuLP. PuLP Case Studies. preuzeto 24.7.2022. <https://www.coin-or.org/PuLP/CaseStudies/index.html>

-
- [19] S. A. Mitchell and J.S. Roy. PuLP. preuzeto 24.7.2022. <http://www.coin-or.org/PuLP/>.
- [20] Sergiy Butenko, Dantzig-Wolfe Decomposition for LPs with Block Diagonal Structure: An Example, Nov 17, 2020. YouTube <https://www.youtube.com/watch?v=wxz0NJvKZNM>
- [21] Sergiy Butenko, Dantzig-Wolfe Decomposition: Intro, Nov 10, 2020. YouTube <https://www.youtube.com/watch?v=IposxyVBUnY>
- [22] Shahidehpour, Mohammad, and Yong Fu. "Benders decomposition in restructured power systems." IEEE Techtorial April (2005). <http://motor.ece.iit.edu/ms/benders.pdf>
- [23] S.Sankaranarayanan, Integer Linear Programming, skripta, preuzeto 24.7.2022. <https://home.cs.colorado.edu/~srirams/courses/csci5654-fall13/ilpLectures.pdf>
- [24] Cplex II. V12. 1: User's Manual for CPLEX. International Business Machines Corporation. 2009;46(53):157. <https://www.ibm.com/products/ilog-cplex-optimization-studio>
- [25] Jacek Gondzio, Exploiting Structure with Interior Point Method, preuzeto 24.7.2022. <https://www.maths.ed.ac.uk/~gondzio/parallel/solver.html>
- [26] John E. Mitchell, Gomory Cutting Planes, skripta, preuzeto 24.7.2022. <https://homepages.rpi.edu/~mitchj/handouts/gomorycuts/gomorycuts.html>
- [27] Williamson, David P., and David B. Shmoys. „The design of approximation algorithms.” Cambridge university press, 2011.

-
- [28] Adi, Said S., et al. „Repetition-free longest common subsequence.” *Discrete Applied Mathematics* 158.12 (2010): 1315–1324.
- [29] Cornuéjols, Gérard, George Nemhauser, and Laurence Wolsey. „The uncapacitated facility location problem.” Cornell University Operations Research and Industrial Engineering, 1983.
- [30] Asghari, M., Fathollahi-Fard, A. M., Mirzapour Al-e-hashem, S. M. J., & Dulebenets, M. A. (2022). Transformation and Linearization Techniques in Optimization: A State-of-the-Art Survey. *Mathematics*, 10(2), 283.
- [31] Lozano, L., Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers & Operations Research*, 40(1), 378-384.
- [32] Jovanović, M. (2013): Konveksne funkcije i ekstremi, Prirodno-matematički fakultet, Univerzitet u Banjoj Luci.
- [33] Dugošija, Đ., Savić, A. (2018). Operaciona istraživanja: linearno i celobrojno programiranje, grafovi i algoritmi (1. izd.). Matematički fakultet.
- [34] Čangalović, M., Dugošija, Đ., Kovačević-Vujčić, V., Simić, S., & Vučeta, J. (1996). Kombinatorna optimizacija: matematička teorija i algoritmi.