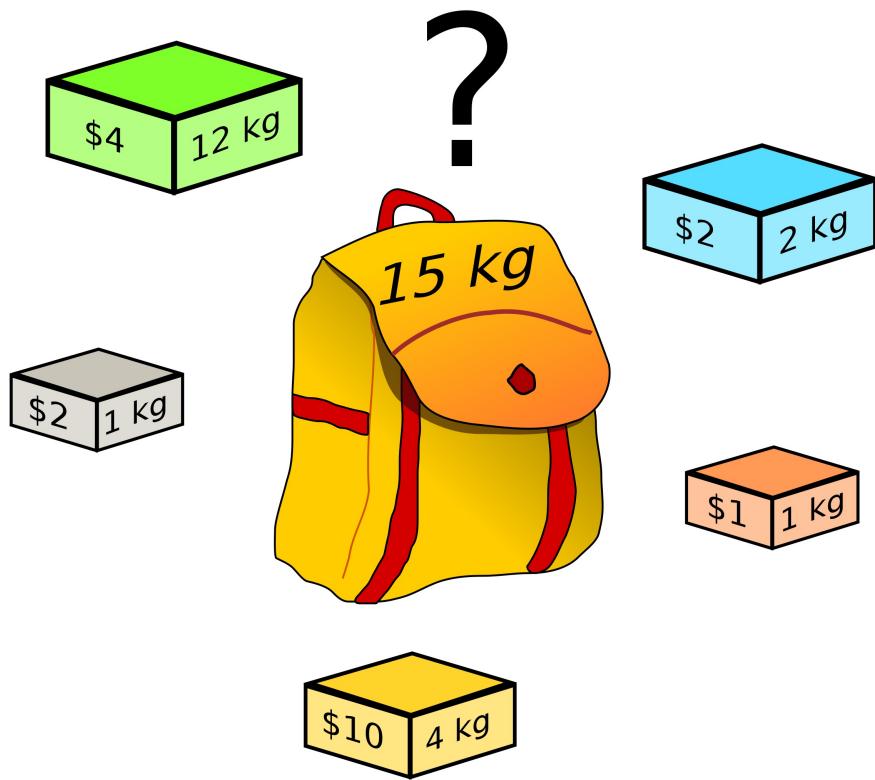


Uvod u Operaciona Istraživanja

Marko Đukanović & Dragan Matić



Univerzitet u Banjoj Luci
Prirodno-matematički fakultet
2022.

Sadržaj

1 Predmet izučavanja Operacionih istraživanja	3
2 Principi Modelovanja u Operacionim Istraživanjima	9
2.1 Definisanje problema i sakupljanje relevantnih podataka	9
2.2 Proces formulisanja matematičkog modela	11
2.3 Izvođenje rješenja iz modela	14
2.4 Testiranje modela	16
2.5 Pripremanje modela za primjenu	17
2.6 Implementacija	18
3 Linearno Programiranje	21
3.1 Grafički metod	23
3.2 Standardna forma linearног programiranja	24
3.3 Modelovanje nekih problema paradigmom linearног programiranja	29
3.4 Teorija linearног programiranja	30
3.5 Neke tehnike u modelovanju problema linearног programiranja	37
3.6 Zadaci	40
4 Simpleks Metod	45
4.1 Konstrukcija simpleks metode	45
4.2 Tabelarni oblik simpleks metoda	48
4.3 Primjena simpleks metoda	49
4.4 Dvofazni simpleks metod	54
4.5 Primjena dvofaznog simpleks metoda	56
4.6 Geometrija i simpleks metod	58
4.7 Zadaci	61
5 Dualnost & Dopustivost	63
5.1 Teoreme dualnosti	64
5.2 Metod unutrašnje tačke	69
5.3 Farkašova lema	76
5.4 Fourier–Motzkin-ova eliminacija	77
5.5 Zadaci	80

6 Cjelobrojno Programiranje	83
6.1 Modelovanje nekih problema	83
6.2 Tehnike transformacija u ILP-u	90
6.3 Kompleksnost cjelobrojnog programiranja	97
6.4 Zadaci	98
7 Algoritamske Tehnike za Rješavanje ILP-a	103
7.1 Metoda grananja i ograničavanja (B&B)	103
7.1.1 Bazni B&B za rješavanje problema ILP-a	104
7.2 Implicitna enumeracija	106
7.3 Metoda odsjecajućih ravni	109
7.4 B&C algoritam za rješavanje ILP-a	116
7.5 Lagranžova relaksacija	118
7.6 Metod generisanja kolona	120
7.7 Zadaci	127
8 Dekompozicioni Metodi	131
8.1 Benderova dekompozicija	131
8.2 Dantzig–Wolfe dekompozicija	144
8.3 Zadaci	155
9 Metode za pronalaženje dopustivih rješenja	159
9.1 Pohlepni algoritmi	159
9.1.1 Primov algoritam	160
9.1.2 Kruskalov algoritam	161
9.1.3 Dajkstrin algoritam	162
9.1.4 Frakcioni problem ruksaka	162
9.1.5 Pohlepni pristup za rješavanje LCS problema	164
9.2 Aproksimativni algoritmi	165
9.2.1 Aproksimativni algoritmi za problem pokrivanja čvorova	166
9.2.2 Aproksimativni algoritmi za problem trgovačkog putnika	168
9.2.3 Aproksimativni algoritmi za problem binarnog pakovanja	171
9.2.4 Aproksimativni algoritmi za problem pokrivanja skupa	172
9.2.5 Aproksimativni algoritam za problem najkraćeg nad-stringa	176
9.3 Heurističke metode	177
9.4 Metaheurističke metode	178
9.4.1 Klasifikacija metaheuristika	178
9.4.2 Genetski algoritmi	180
9.4.3 Metoda lokalne pretrage	186
9.4.4 Metoda promjenljivih okolina	187
9.5 Zadaci	190

10 Optimizacioni Rješavači	195
10.1 Optimizacioni rješavač CPLEX	195
10.2 PuLP alat	200
10.3 Primjena alata PuLP na rješavanje jednog problema lokacija sa ograničenim kapacitetima	202
10.4 Primjena CPLEX rješavača na problem stringova	204
10.5 Zadaci	208

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 predmet Uvod u operaciona istraživanja, za studente treće godine informatičkog smjera na Prirodno-matematičkog fakulteta Univerziteta u Banjoj Luci.

Pretpostavljamo 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 u uvodnim kursevima iz programiranja, te poznavanje 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 odgovaraјуćih tehnika. U udžbeniku je prezentovan veliki broj primjera realnih problema koji potiču iz prakse, a na različitim mjestima su objašnjene različite faze rješavanja tih problema, počev od identifikovanja samog problema, njegove formalne matematičke formulacije, analizu potencijalnih metoda za rješavanje, pa do implementacije rješenja u odgovaraјućem rješavaču ili pak pomoću samostalno dizajniranog algoritma.

Uz pomoć znanja stečenog iz ovog udžbenika, polaznik će biti sposoban da sam identificira i formuliše probleme koji spadaju u oblast operacionih istraživanja (sa naglaskom na probleme linearног programiranja), analizira različite metode za rješavanje datih problema, te, primjenjujući odgovaraјuće 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. Na primjer, teorija dualnosti se koristi u dekompozicionim metodama, simpleks metoda se može kombinovati sa drugim metodama za rješavanje problema linearog i cjelobrojnog linarnog 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 preskače na neka naredna.

Za uspješno savladavanje gradiva iznesenog u udžbeniku potreban je strpljiv i studiozan rad. Čitaocu se preporučuje i da sve prezentovane primjere detaljno razradi i po potrebi testira na svom računaru, da dodatno analizira ponuđena rješenja, formira svoje mišljenje o rješenjima u udžbeniku i predloži svoja rješenja. Neki zadaci podrazumijevaju i dodatni angažman, koji uključuje prikupljanje potrebnih informacija i iz drugih izvora (na primjer iz pouzdanih internet izvora), što je u praksi standardan i uobičajen metod za učenje i napredovanje u struci.

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 računarstvo i informatiku Prirodno-matematičkog fakulteta Univerziteta u Banjoj Luci.

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, ove tehnike 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 varijabli 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 karakteristike operacionih istraživanja su

- *Optimizacija*: uključuje poređenja i navođenje pretrage prema potencijalno boljim opcijama.
- *Simulacija*: podrazumijeva izgradnju modela problema iz koga se dobijaju rješenja koja se akon toga provjeravaju prije nego se implementiraju u realnoj situaciji.
- *Vjerovatnoća i statistika*: podrazumijeva (i) upotreba matematičkih algoritama i podataka u smislu otkrivanja korisnih informacija koje služe u davanju pouzdanih predviđanja u realnom svijetu; (ii) testiranja kvaliteta i upotrebljivosti rješenja.

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

Pojam Operacionih istraživanja nije jednostavno opisati. U osnovi, Operaciona istraživanja su nauka koja se bavi ispitivanjem kako doći do niza efikasnih odluka za skup operacija koje se provode. Operaciona istraživanja rješavaju probleme iz prakse (nazovimo ih biznis problemi) koje 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, do-nošenje niza odluka da bi se došlo do cilja na što efikasniji način. Jedna od najmoćnijih tehniki 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, već ono u ovom kontekstu označava optimizaciju.

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

1. Biznis problemi;
2. Generički problemi;
3. Paradigma modelovanja;
4. Algoritamska rješenja.

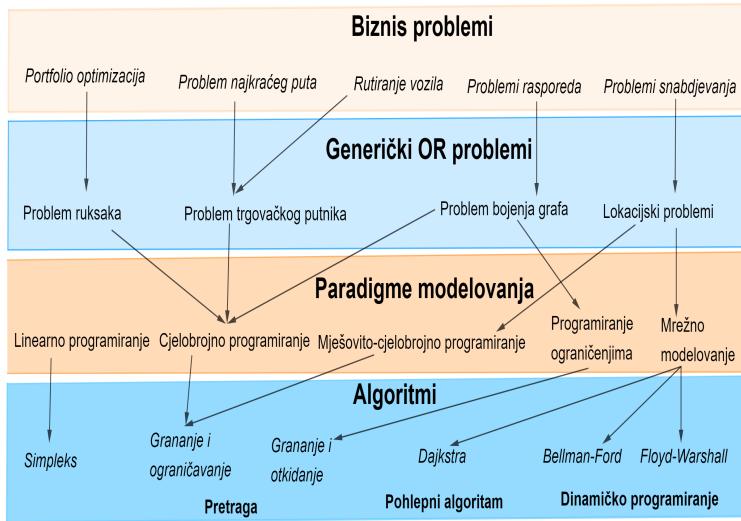
Pod *biznis problemima* se ne podrazumijevaju samo problemi iz domena ekonomije, već svi problemi koji imaju primjenu u stvarnom svijetu. Pojam "biznis" više odgovara tome da problem nema svoju preciznu matematičku formulaciju, nego je neformalno definisan kroz jezički (deskriptivan) opis.

Najčešće, ovi problemi proizilaze iz problem u industriji i kao takvi se i opisuju. Na primjer, 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*.

Na početku razvoja Operacionih istraživanja kao nauke, pri konceptualizaciji u pozadini nije bilo jakog matematičkog formalizma. Matematičari su prepoznali da se većina biznis problema može mapirati u generičke familije problema nižeg nivoa. Ovom mapiranju je posvećeno dosta vremena, da bi se na kraju završilo sa nekoliko standardnih generičkih klasa na koje se mapira gotovo svaki problem iz operacionih istraživanja. U praksi, najveći dio vremena se provodi u konvertovanju biznis problema u jedan od poznatih generičkih problema. Nakon toga, rješavanje problema je manje-više standardna procedura. Generički problemi operacionih istraživanja su dovoljno koncizni da se mogu predstaviti preko matematičke notacije. Međutim, u svrhu boljeg razumijevanja, u praksi se koristi jezik modelovanja koji je na višem nivou apstrakcije od njih; tako npr. problemi raspoređivanja se obično opisuju upotrebotom štvarnih "termina, kao što su resursi, aktivnosti, prioriteti, uslovi raspoređivanja i sl.

Paradigma modelovanja. Kod generičkih operacionih problema koji su definisani jezikom višeg nivoa, notacija je najčešće vezana uz problem koji se razmatra. Na primjer, resursi i aktivnosti nisu nešto što se može definisati u, recimo, poznatom problemu trgovачkog putnika. U osnovi, svaki od ovih generičkih problema se može opisati nekom od paradigm modelovanja. Paradigma modelovanja predstavlja skup pravila koja omogućuju da predstavimo probleme višeg nivoa, koristeći strukture nižeg nivoa, kao što su, recimo, 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 je sposoban za generisanje rješenja. Najpoznatije paradigmе modelovanja su sljedeće: Linearno, Cjelobrojno i Mješovito-cjelobrojno programiranje. Postoje i druge paradigmе poput Modelovanje ograničenjima (eng. *Constraint Programming*) i mrežno modelovanje (eng. *Network models*).

Algoritamska rješenja. Algoritam je procedura ili konačan niz instrukcija čijim izvršavanjem možemo riješiti problem. 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. Specifično, među ove algoritme



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

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 u operacionim istraživanjima. Ova familija algoritama rješava probleme tako što eksploratiše optimalnu pod-strukturu problema. Drugim riječima, polazni problem razbijemo na manje potprobleme, koje rješavamo rekurzivno. U rješavanju glavnog problema, koristimo pronađena optimalna rješenja potproblema, izbjegavajući da potprobleme koji se ponavljaju rješavamo više puta. Ova programska paradigma ima značajnu ulogu u rješavanju grafovskih problema. Jedan od najpoznatijih takvih algoritama je *Bellman-Ford*-ov algoritam, nazvan po izumitelju paradigmе dinamičkog programiranja, Richardu Belmanu.

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 vremena i memorijskih resursa. Za potrebe dobijanja brzog (ne obavezno optimalnog) rješenja nam koriste *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 paradigmе modelovanja i najznačajnije

algoritme za rješavanje navedenih problema. Odnos između sva četiri nivoa apstrakcije su data 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 operacionih istraživanja.

Glava 2

Principi Modelovanja u Operacionim Istraživanjima

Uobičajeno je izraz Operaciona istraživanja zapisivati skraćenom akronimom OR (eng. *operational research*). U ovoj glavi ćemo objasniti glavne faze tipičnog OR istraživanja 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 uprave koja naručuje/finansira sistem.
- 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 susreću OR timovi, koji razvijaju projekat 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 krucijalan korak, jer

uvetliko utiče na relevantnost zaključaka istraživanja. Izvlačenje odgovora ili dobijanje rezultata iz pogrešne postavke polaznog problema sigurno neće dovesti do ispravnog rješenja.

Zbog toga je važno da OR tim, koji modeluje problem, ne radi samostalno, već blisko sarađuje sa naručionicom zadatka. Članovi tima, koji su zaduženi za formulisanje problema, savjetuju se sa upravom firme za koju se razvija sistem, poslije čega tim izrađuje detaljnu tehničku analizu u vezi problema i prezentuje preporuke za rješenje osobama koje su ključne u donošenju odluka.

Najčešće, izvještaj koji se daje upravi će sadržavati osnovno rješenje, ali i brojne alternative, koje odgovaraju različitim uslovima ili vrijednostima parametara za problem (na primjer, balansiranje između cijene i dobiti, stabilnosti poslovanja i dobiti, itd.). Na upravi je da zatim vrednuje prijedloge, uzimajući u obzir niz faktora i da nakon toga donosi odluku zasnovanu na najboljoj procjeni. Važno je i da OR tim bude sličnog mišljenja kao i uprava. Utvrđivanje odgovarajućih ciljeva je neophodno za ispravno definisanje problema. Za to je potrebno neprestano komunicirati sa osobom iz uprave koja je donosilac odluka u vezi sistema koji se razvija, a potom pristupiti realizaciji svakog od ciljeva.

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 rijечima, analiza bi mogla postati pretjerano glomazna ako bi zadani ciljevi bili previše uopšteni. Umjesto toga, ciljevi bi trebalo biti specifični, koliko god je to moguće, ali da se pri tome obuhvate glavni ciljevi donosilaca odluka.

Kada su u pitanju profitne organizacije, jedan od mogućih pristupa zaobilazeњa problema pod-optimizacije je korištenje dugoročne maksimizacije profita. Riječ "dugoročno" ukazuje da cilj pruža fleksibilnost u razmatranju aktivnosti koje ne donose momentalnu dobit, ali se na moraju uključiti kako bi omogućili krajnji profit. Ovakav cilj je dovoljno specifičan, a ipak i dovoljno širok da obuhvati osnovni cilj profitnih organizacija, dolazak do (zadovoljavajuće) zarade. Zapravo, svi drugi legitimni ciljevi ovakvih organizacija mogu se podvesti u prethodno pomenuti. Međutim, u praksi profitne organizacije rijetko koriste ovaj pristup. Velik broj profitnih organizacija prilagođava zadovoljavajući profit sa drugim ciljevima, umjesto fokusiranja na maksimizovanje dugoročne zarade. Neki od takvih ciljeva su: izgradnja stabilnog profita, proširivanje tržišta, 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.

Timovi zaduženi za modelovanje obično provode veliku količinu vremena prikupljući relevantne podatke o problemu. Mnogo podataka je potrebno prikupiti kako bi se došlo do tačnog razumijevanja problema i dolaska do potrebnih ulaznih podataka za matematički model, 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 pogrešnom formatu. Članovi tima komuniciraju sa ključnim osobama u upravi, kako bi pronašli sve podatke značajne za projekat. Međutim, mnogi podaci mogu biti prilično "neprecizni", tj. mogu biti grube procjene zasnovane na istraživanjem pretpostavaka. Tada se izdvaja značajna količina vremena da bi se poboljšala preciznost podataka, a zatim se proces nastavlja sa najboljim mogućim resursima koji su prikupljeni.

Primjer. OR studija je urađena za policijsku upravu jednog grada na 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 željenog nivoa zaštite. Matematički model koji je razvijen je nametnuo uslov da će se postići potreban nivo zaštite preko funkcije cilja. Da bi se postigao drugi cilj, u samom modelu je uključen i uslov o ravноправno uravnotežnim 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. Konvencionalni OR pristup se sastoji od konstrukcije matematičkog modela koji na najbolji

način odgovara realnom problemu. Prije nego što objasnimo kako konstruisati takav model, recimo nešto o modelima u opštem slučaju, a posebno o matematičkim modelima.

Modeli (idealizovani prikazi) su sastavni dio svakodnevnog života. 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 analize.

Matematički modeli su obično izraženi preko matematičkih simbola, kao što je, na primjer, poznata formula $F = ma$ u fizici. Slično, matematički model biznis problema se sastoji od sistema jednačina i matematičkih izraza koji opisuju osnov problema. Na primjer, postoji n mjerljivih odluka koje treba da budu donesene i koje su predstavljene pomoću nepoznatih (varijabli) x_1, \dots, x_n . Odgovarajuća mjera učinka (profita) je izražena preko matematičkih funkcija nepoznatih koje odgovaraju odlukama (npr. $3x_1 + 5x_2 + x_3$). Ovaj funkcija se naziva *ciljna* (objektivna) funkcija, ili *funkcija cilja*. Ograničenja vrijednosti promjenjivih 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 varijabli (odluka), 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 polju. Određivanje odgovarajućih vrijednosti, koje se dodjeljuju parametrima modela, je kritični dio procesa konstrukcije modela.

Za razliku od problema koji su već definisani u udžbenicima, u kojima 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 "pravi" model; u praksi često dva ili više potpuno različitih modela odgovaraju jednom problemu. Ovo je stvar razvoja modela koji je iterativni proces koji proizilazi 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 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 tehnika 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. Kako je model “apstraktna idealizacija problema”, često su neophodne aproksimacije i pojednostavlјivanja pretpostavki, ako želimo da model bude izvršiv (tj. sposoban za rješavanje problema). Stoga se mora paziti da model ostane valjan prikaz problema nakon pojednostavljenja. To znači da treba da postoji značajna korelacija između onoga što model predviđa i onoga što bi se zapravo i dobilo 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.

U razvoju modela, dobar pristup je započeti sa pojednostavljenom verzijom, te krenuti evolucionim načinom prema složenijim modelima koji će bolje odražavati stvarni problem. Ovaj proces obogaćivanja modela se nastavlja dok god je model rješiv. Osnovni 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 mjerne obično transformišu i kombinuju u jednu složenu mjeru. Ukupna mjera može biti nešto opipljivo (npr. dobit) koja odgovara višem cilju organizacije. Nakon što se razvije cjelokupna mjera, ciljna funkcija se izražava kao matematička funkcija čiji su argumenti varijable odluke.

Primjer. 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 $100l$ multimitaminskog 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 $500l$ tekućine, a za bistri sok bure kapaciteta $400l$. Na kraju dana dolazi preprodavač koji kupuje sok. Cilj proizvođača sokova je da napravi koliko god može soka, uzimajući u obzir data ograničenja, kako bi maksimizovao svoj profit. Drugim riječima, treba napraviti plan proizvodnje

koji će maksimizovati profit.

Model za ovakav problem bi išao ovako:

- Varijable odluke: x_1 i x_2 koje označavaju koliko litara gustog soka i koliko litara bistrog soka proizvođač treba da napravi.
- 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: $0 \leq x_1 \leq 5$, $0 \leq x_2 \leq 4$, $3x_1 + 6x_2 \leq 30$, $8x_1 + 4x_2 \leq 44$.

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 modela. Sa druge strane, nekada su matematički modeli toliko složeni da se ne mogu riješiti direktnom primjenom standardnih algoritama, već se podrazumijeva razvoj i primjenu drugih, sofisticiranih tehnika. Dodatni posao dolazi u sljedećim koracima, koji uključuje analizu dobijenih rezultata, koju ćemo objasniti kasnije.

Veći dio ove knjige posvećujemo metodama koje pronalaze rješenja iz modela i prirodi dobijenih rješenja. Uobičajeni zadatak u OR polju je potraga za optimalnim (najboljim) rješenjem i razvoj odgovarajućih postupaka koji služe za pronalaženje optimalnih rješenja zadatog modela. Treba imati na umu da su ta rješenja optimalna samo za dati model. Kako model ne mora u potpunosti odgovarati stvarnom problemu, već je on najčešće njegova idealizovana varijanta, ne postoji garancija da će se nađeno optimalno rješenje modela pokazati 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 ispuni, a rješenje koje zadovoljava te uslove najčešće se uzima kao prihvatljivo. U procesu donošenja odluka, pokušava se sa što više koristi naučni pristup, sa primarnim ciljem pronalaska globalnog optimuma za posmatrani model. Međutim, uslov da se u razumnom vremenskom roku dođe do modela koji generiše rješenje zadovoljavajućeg kvaliteta, najčešće ima prednost u odnosu na nalazak optimuma u nerazumnom vremenu. Tim koji rukovodi procesom razmatra troškove procesa razvoja modela, a zatim pokušava maksimizovati korist u oba smjera. OR timovi povremeno koriste samo približne (heurističke) pristupe rješavanju problema. Ovakvi pristupi su intuitivno dizajnirani

postupci kojima se ne garantuje optimalno rješenje. 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 dobija sve više na značaju.

Kao što je već nekoliko puta pomenuto, optimalno/podoptimalno rješenje za konstruisani model može biti daleko od idealnog za stvarni problem, pa su potrebne dodatne analize. Stoga je analiza "postoptimalnosti" (analiza provedena nakon pronađaska (pod)optimalnog rješenja) vrlo važan dio OR studija. Ova analiza se ponekad naziva i šta–ako analiza, 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 uprava (firme) koja će donijeti konačne odluke u vezi procesa dodavanja uslova.

Pojava softvera koji rade sa proračunskim tabelama (prvenstveno Excel-a) igra centralnu ulogu u provođenju analize postoptimalnosti. Jedna od velikih prednosti ovakvih softvera je jednostavnost u korišćenju, gdje se lako dobija analiza što 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 uvjeravanja 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 narušavanje 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 primjeni 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 određena 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 premala da bi se nastavilo sa ovakvim iterativnim procesom. Alternativna rješenja (rješenja koja su optimalna za jedan od nekoliko probližnih modela) se takođe mogu predstaviti upravi 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 zaključuju da trenutni program sada generalno daje valjane rezultate. Može se desiti da neke manje greške ostanu skrivene u programu te se možda nikada i neće ukloniti. Slično tome, prva verzija matematičkog modela neizbjegno 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 s obzirom na potencijalne probleme u komunikaciji, teškoću razumijevanja svih aspekata problema, te poteškoća 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. Na kraju, nakon određenog broja iteracija poboljšanja modela, OR tim koji radi na modelu zaključuje da trenutni model sada daje razumno valjane rezultate. Iako neki manji nedostaci nesumnjivo ostaju skriveni u modelu (a možda nikada neće ni biti otkriveni), glavni nedostaci su otklonjeni, te se može tvrditi da je model pouzdan. Ovaj postupak ispitivanja i poboljšanja modela radi povećanja njegove valjanosti se naziva *validacija modela*.

Budući da OR tim može provesti mjesecce na izradi svih detalja modela, 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 pogrešaka. Takođe, korisno je da su svi matematički izrazi dimenzionalno konzistentni. Ponekad se dodatni uvid u valjanost modela dobija i mijenjanjem vrijednosti parametara i/ili varijabli odluke, kao i provjerom da li se izlaz iz modela ponaša na očekivan način. To se često može otkriti kada se parametrima ili varijablama 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 utvrđuje koliko bi dobro funkcionali model i rezultujuće rješenje, ako bi ono bilo korišteno u praksi. Ovaj pristup može da ukaže na dijelove na kojima model ima nedostatke, što zahtijeva dalje izmjene i poboljšanja. S druge strane, nedostatak retrospektivnog ispitivanja je

taj što koristi iste podatke kao i oni koji vode ka formulaciju modela. Važno je još dokumentovati postupke koji se koriste u testiranju valjanosti modela. Ovo može povećati povjerenje u model za buduće korisnike. Takođe, ako se u budućem vremenu pojave pitanja u vezi sa modelom, ova dokumentacija će biti korisna u dijagnosticiranju problema.

2.5 Pripremanje modela za primjenu

Ako želimo da model bude iskorišten više puta, sljedeći korak je instaliranje dobro dokumentovanog sistema za primjenu modela po propisima naručioca (uprave). 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 ažuran unos podataka u model, te je zbog toga potrebno obezbijediti i odgovarajući interfejs. Nakon što se na model primjeni postupak rješavanja (drugi program), 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 upravi da koriste podatke i modele kako bi, po potrebi, podržali (a ne zamjenili) donošenje odluka. Drugi program može generisati menadžerske izvještaje, koji tumače izlazne podatke modela i njegove 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. Dio ovog napora uključuje razvoj i izvođenje postupka održavanja sistema tokom njegove buduće upotrebe. Kako se uslovi 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.

Primjer. Sistem nazvan *Optimizer* pruža optimalnu kontrolu usluga i zaliha rezervnih dijelova u cijeloj IBM-ovoj mreži za distribuciju dijelova u SAD-u, koja uključuje dva središnja automatizovana skladišta, desetine terezenskih distribucijskih centara i stanica za dijelove, te nekoliko hiljada vanjskih (povezanih) lokacija. Inventar 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 koji traje i sedmicu dana kako bi se optimizovala kontrola zaliha. Četvrti modul uključuje šest programa koji integrišu *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 propisala uprava (koja naručuje razvoj sistema). Ova faza je ključna jer se ovdje “ubiru plodovi rada” obje strane. Stoga je važno da OR tim učestvuje u realizaciji ove faze, kako bi se osiguralo da se rješenja modela precizno prevedu u operativni postupak, ali i da bi se ispravili svi nedostaci u rješenjima koja se otkriju. Uspjeh faze implementacije uveliko zavisi i od podrške vrha uprave (firme). Mnogo je vjerovatnije da će OR tim dobiti podršku ako je dobro informisao upravu i podsticao njeno aktivno učestvovanje tokom razvoja rješenja. Dobra komunikacija pomaže da istraživanje postigne ono što je uprava željela. Faza implementacije uključuje nekoliko koraka. Prvo, OR tim daje operativnom rukovodstvu opis novog sistema koji treba da bude usvojen. Dalje, ove dvije strane dijele odgovornost za razvijanje koraka potrebnih za pokretanje ovog 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 narednim godinama. Imajući to na umu, razvojni tim nadgleda iskustva korisnika sistema i nastoji da utvrdi koje izmjene u budućnosti bi trebalo uraditi na sistemu u cilju što lakošeg korištenja istog. Tokom vremena u kojem se koristi novi sistem, važno je i dalje dobijati povratne informacije o funkcionisanju sistema, te 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 da OR tim dovoljno jasno i precizno dokumentuje svoju metodologiju, kako bi se čitav posao mogao ponoviti. Repliciranje bi trebalo biti dio profesionalnog i etičkog kodeksa projektovanja (informacionih) sistema.

Primjer. Vratimo se na realni IBM projekat pomenut u prethodnoj sekciji. Tri su se glavna faktora pokazala posebno važnim za uspješan završetak projekta. Kao što je raspravlјano u prethodnim poglavljima, prvo je uključivanje korisničkog tima i (operativnog) rukovodstva koji je savjetovao 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 bio je dobijanje vrlo široke povratne informacije korisnika sistema, gdje su identifikovane problematične tačke u sistemu koje su trebale biti ispravljene prije potpune implementacije. Treći faktor je bio taj da je novi sistem uvođen postepeno, fazu po fazu, uz pažljivo testiranje u svakoj fazi, kako bi se veće greške mogle ukloniti prije nego je sistem pušten 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, kako 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

Linearno Programiranje

Početak razvoja linearнog programiranja počinje od sredine 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 troškova u vojsци. 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 u raznim zemljama svijeta. Stoga se razvoj linearнog programiranja smatra jednim od najvažnijih naučnih dostignuća XX vijeka. Napisano je na desetine, pa i stotine udžbenika o linearном programiranju i objavljeno je na hiljade naučnih radova koji ovu paradigmu primjenjuju na probleme iz raznih naučnih i praktičnih domena.

Linearно programiranje koristi matematički model za opisivanje problema; pridjev "linearно" 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, linearно programiranje uključuje planiranje aktivnosti u cilju postizanja optimalnog rezultata, tj. rezultata koji najbolje postiže navedeni cilj (prema matematičkom modelu) među svim izvedivim 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. U ovom primjeru opisujemo jedan industrijski projekat naziva WYNDOR–GLASS CO projekat, prikazujući kako OR može da riješi ozbiljne probleme u industriji.

Firma WYNDOR–GLASS proizvodi visokokvalitetne staklene proizvode, uključujući prozore i staklena vrata. Postoje tri pogona. Aluminijski okviri i oko-

vi izrađuju se u pogonu 1, drveni okviri izrađuju se u pogonu 2, a pogon 3 proizvodi staklo i sastavlja proizvode. Zbog pada zarade, rukovodstvo firme odlučilo je da obnovi liniju proizvoda koje nudi. Nefitabilni 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 postrojenjima 1 i 3, ali ne korisiti postrojenje 2;
- Proizvod 2 za proizvodnju koristi postrojenja 2 i 3 .

Rukovodstvo kompanije je zaključilo da bi kompanija mogla prodati sve proizvode koje bi mogla proizvesti u postrojenjima. 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, glavni ciljevi istraživanja su utvrđeni, gdje se utvrdila definicija (idealizovanog) problema koji treba da se riješi:

- 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 drugog proizvoda na osnovu utvrđenih kapaciteta.

Da bi se razvio model, informacije koje je bilo neophodno prikupiti prije modelovanja su:

- (i) Broj radnih sati u sedmici dostupan u svakom pogonu za nove proizvode;
- (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 dobit po proizvedenoj seriji bude kao odgovarajuća mjera – zaključeno je da dobit svake dodatne proizvedene serije raste približno konstantno bez obzira 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, treba procijeniti konstante (parametre) u modelu. Ovo je ostvareno u komunikaciji između uprave firme i tima zadužen za rješavanje problema. Za dobijanje razumnih procjena, pomoć ključnog osoblja u raznim odjelima firme je odigrala ključnu ulogu. Osoblje u proizvodnom odjelu informisalo je o podacima za stavku (*i*). Za procjenu slučaja (*ii*), ključna je bila analiza proizvodnih inženjera uključenih 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.

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

Tabela 3.1: Procjene konstanti modela.

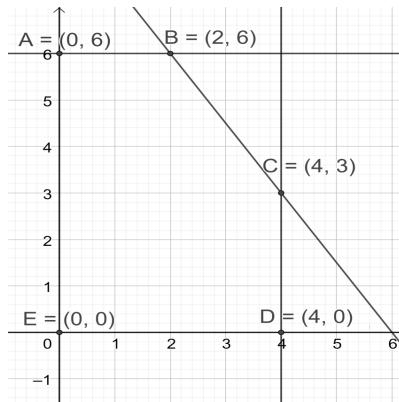
U nastavku konstruišemo model problema. Uvedimo varijablu x_1 za broj serija proizvoda 1 proizvedenog na sedmičnom nivou, te x_2 broj serija proizvoda 2 koji je proizведен na sedmičnom nivou. Funkcija profita je jednaka $f(x_1, x_2) = 3x_1 + 5x_2$, dok se rješenja traže pod ograničenjima

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

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 ima maksimalnu vrijednost.

3.1 Grafički metod

Problem iz prethodne sekcije je malih dimenzija jer sadrži samo dvije varijable odluke i samim tim sva moguća (dopustiva) rješenja mogu biti predstavljena u dvije dimenzije. U ovom slučaju, možemo koristiti *grafički metod*



Slika 3.1: Dopustiv region.

za rješavanje problema. Ovaj 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 ograničenja. Naglasimo još jednom da ona rješenja koja zadovoljavaju sva ograničenja modela problema zovemo *dopustivim rješenjima*. Grafički predstavimo svako od ograničenja, odakle dobijamo prostor koji uključuje svih 5 ograničenja; vidjeti Sliku 3.1. Rezultujući region rješenja (x_1, x_2) se naziva *dopustivi region* problema.

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

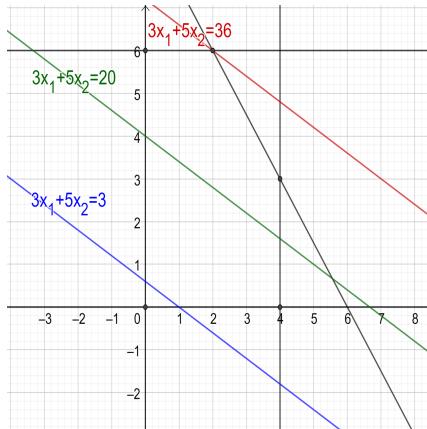
Pogledajmo sada niz (implicitnih) funkcija $3x_1 + 5x_2 = c$, za $c \in \{10, 20, 36\}$, kao na Slici 3.2. Jasno se vidi da prava $f = 36$ prolazi kroz tačku (vrh) regiona $(2, 6)$, koja je svakako dopustiva. Takođe, vidimo da niti jedna druga funkcija $f = c$, $c > 36$ ne presijeca dopustiv region, odakle zaključujemo da je tačka $(2, 6)$ 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 varijable. Uz znatan angažman, ovu metodu je moguće proširiti na problem sa tri varijable, 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

U ovoј sekciji govorimo o opštoj formi modela linearног programiranja. Uvedimo neke ključne pojmove koji se standardno koriste u modelovanju i rješavanju problema linearног programiranja.

Podimo od pojmove *resursi* i *aktivnosti*. Pretpostavimo da u našem razmatranju imamo m resursa i n aktivnosti. Standardni resursi su vrijeme,



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

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 najčešće ograničena, pa aktivnosti trebaju biti pažljivo raspoređene u odnosu na odgovarajuće resurse. Utvrđivanje ove raspodjele uključuje odabir vrsta i nivoa aktivnosti za koje se postiže najbolja moguća vrijednost ukupne mјere učinka. Ukupna mјera učinka može biti mјerenje dobiti (profita), mјerenje vremena realizacije aktivnosti, mјerenje broja proizvedenih proizvoda itd.

U nastavku navodimo notaciju koja se najčešće koriste u formiranju opštег 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 količina aktivnosti j podigne za jediničnu vrijednost;
- b_i : ukupna količina resursa i ($i = 1, \dots, m$) koja je dozvoljena za alociranje aktivnosti;
- a_{ij} : količina resursa i koja se uzima od strane jedinične (vrijednosti) aktivnosti j .

Prema ovoj notaciji, matematički modeli problema linearног programiranja ima sljedeću postavku:

$$f = c_1x_1 + \cdots + c_nx_n \rightarrow \max \quad (3.1)$$

s.t.

$$a_{11}x_1 + a_{12}x_{12} + \cdots + a_{1n}x_n \leq b_1 \quad (3.2)$$

...

$$a_{m1}x_1 + a_{12}x_{m2} + \cdots + a_{mn}x_n \leq b_m \quad (3.3)$$

$$x_i \geq 0, \quad i = 1, \dots, n. \quad (3.4)$$

Postavka (3.1)–(3.4) se još naziva i *standardna forma* problema linearne programiranja. Funkcija (3.1) se naziva funkcija cilja koju je potrebno maksimizovati pod uslovima (3.2)–(3.3), koji se nazivaju *funkcionalna ograničenja*. Ograničenja (3.4) se nazivaju *nenegativna ograničenja*.

Kompaktiniji oblik standardne forme je dat sa:

$$f = c^T x \rightarrow \max \quad (3.5)$$

s.t.

$$Ax \leq b \quad (3.6)$$

$$x \geq 0, \quad (3.7)$$

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 ograničenja (3.6) mogu da imaju i oblike:

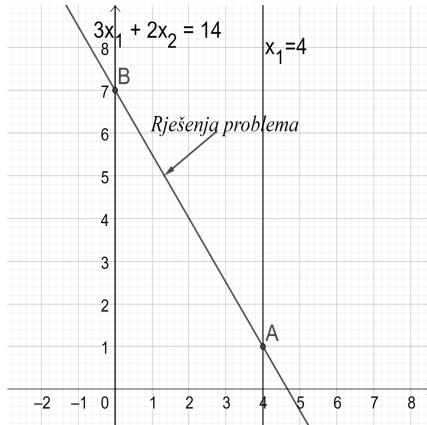
- $Ax \geq b$;
- $Ax = b$;
- neke od varijabli $x_i \leq 0$.

Bilo koji problem koji koristi neke od prethodna tri ograničenja predstavlja i dalje problem linearne programiranja. Dodavajući tzv. *izravnajajuće varijabe* $s \in \mathbb{R}^m$ 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 izravnajajućih varijabli kao $Ax - s = b, s \geq 0$.

Forma linearne programske funkcije sa ograničenjima $Ax = b$ se naziva *kanonska forma* linearne programiranja.

Takođe, ako u linearnom programu postoji varijabla koja ne zadovoljava uslov nenegativnosti, ona se može zamijeniti razlikom dve nove nenegativne promenljive. Kanonski oblik linearne programiranja je pogodan za efikasno izvođenje Gausovih transformacija nad matricom problema, što će biti od koristi u konstrukciji simpleks metoda.

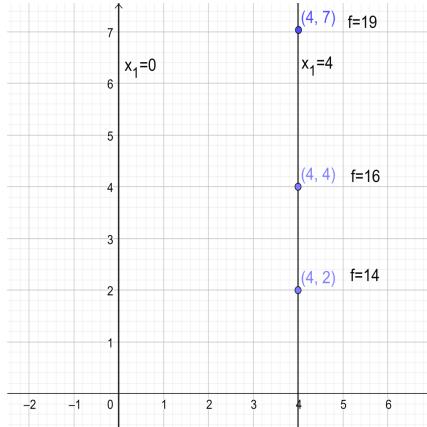
U nastavku pratimo sljedeću terminologiju u vezi linearne programiranja (LP):



Slika 3.3: LP sa više od jednog optimalnog rješenja.

- *Rješenje* u standardnom matematičkom značenju najčešće označava krajnje rješenje (u LP-u bi to bilo najbolje rješenje). Međutim, u rješavanju problema LP-a to nije slučaj. Bilo koja specifikacija vrijednosti varijabli (x_1, \dots, x_n) se smatra rješenjem bez obzira da li je riječ o željenom rješenju ili čak o nekom rješenju koje i ne zadovoljava sva organičenja. 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 pronaleta najboljeg rješenja modela. Takođe, može se desiti da problem nema niti jedno dopustivo rješenje. Kažemo da je taj problem *nedopustiv*.
- *Optimalno rješenje* je ono rješenje iz dopustivog skupa za koje je funkcija cilja najveća (pod pretpostavkom da maksimizujemo funkciju cilja), odnosno najmanja (pod pretpostavkom da minimizujemo funkciju cilja). Ova vrijednost ne mora biti jedinstvena, pogledati Sliku 3.3 gdje je dopustiv region isti kao u primjeru sa Slike 3.1, dok je funkcija cilja data sa $f = 3x_1 + 2x_2$.

Za razliku od nedopustivog problema, može se desiti i sasvim druga situacija, da je problem neograničen. U tom slučaju vrijednosti (jedne od) varijabli se mogu povećavati tako da funkcija cilja (u problemu maksimizacije) teži ka beskonačno, tj. nije ograničena odozgo nekom vrijednošću. Za taj problem kažemo da je *neograničen* problem.



Slika 3.4: Primjer LP-a koji je neograničen.

Primjer takvog 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, što se više povećava vrijednost varijable x_2 , to funkcija f više raste. Ako bi x_2 bilo ∞ , to bi i funkcija f bila neograničena ($f = \infty$).

Uvedimo sada pojam *ekstremnih tačaka*, koje igraju bitnu ulogu u simpleks metodi. Za rješenje x kažemo da je *ekstremna tačka* ako se nalazi u vrhu dopustivog regiona (u presjeku granica dopustivog regiona).

Postoji važne veze između ovih tačaka i optimalnog rješenja LP-a, o čemu će biti riječi u nastavku ove knjige. Prije toga, za početak, razmotrimo bilo koji problem LP-a sa dopustivim rješenjima i ograničenim dopustivim regionom. Tada će važiti, što ćemo uskoro i pokazati, da problem u tom slučaju posjeduje barem jednu tačku koja je vrh tog regiona i barem jedno optimalno rješenje. Nadalje, jedan od vrhova mora biti optimalno rješenje. Dakle, ako problem ima tačno jedno optimalno rješenje, to rješenje je tačka koja je vrh regiona. Ako problem ima više optimalnih rješenja, barem dva moraju biti rješenja koja su tačke vrhova dopustivog regiona.

Napomena. Treba imati na umu da (optimalno) rješenje LP-a ne mora da bude cjelobrojno. Budući da svaka varijabla odluke predstavlja nivo odgovarajuće aktivnosti, pretpostavlja se da se aktivnosti mogu izvoditi na razlomljenim nivoima (aktivnost se ne dijeli samo na diskretne dijelove), na osnovu nenegativnih ograničenja.

Ako su sve varijable u LP-u cjelobrojne, riječ je o modelu *Cjelobrojnog programiranja*. Ako pored cjelobrojnih, postoe i neke varijable koje su neprekidne, riječ je o modelima *Mješovitog-cjelobrojnog linearogn programiranja*. Veliki broj realnih problema koji dolaze iz prakse su upravo problemi Mješovitog-cjelobrojnog linearogn programiranja.

3.3 Modelovanje nekih problema paradigmom linearog programiranja

Problem optimalne prehrane. Prepostavimo da su nam na raspolaganju namirnice N_1, \dots, N_n . Cijena namirnice N_j po jedinici je $c_j > 0$, $j = 1, \dots, n$. U namirnicama su prisutni nutritivni elementi e_j , $j = 1, \dots, m$ pri čemu u namirnici N_j ima $a_{ij} \geq 0$ 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 prvo varijable $x_j \geq 0$ za količinu hrane namirnice N_j , $j = 1, \dots, n$ koja će da bude korištena za prehranu. Funkcija cilja treba da odgovara minimizaciji cijene prehrane, pa je, prema tome

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

Dalje, ograničenja su sljedeća: treba da se zadovolje dnevne potrebe za unos svakog nutritivnog elementa, odakle 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.

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. U prevodu, 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 linearnu funkciju (već je linearna po dijelovima). Međutim, elementarnim transformacijama, polazni problem možemo da preformulišemo na sljedeći problem linearnog programiranja:

$$f(x) = c^T w, \text{ pod uslovima}$$

- $-w_i \leq kx_i + l - y_i$
- $kx_i + l - y_i \leq w_i$

za sve $i = 1, \dots, n$ gdje je $w_i = z_i, i = 1, \dots, n, w_{n+1} = k, w_{n+2} = l$, dok $c_i = 1, i = 1, \dots, n$, i $c_{n+1} = 0, c_{n+2} = 0$.

Kada je riječ o linearnim programima, oni imaju nekoliko osnovnih implicitnih pretpostavki.

1. *Proporcionalnost.* Doprinos svake aktivnosti ciljnoj funkciji f proporcionalan je njenom koeficijentu doprinosa, tj. $c_i x_i$. Doprinos svake aktivnosti svakom od ograničenja u modelu proporcionalan je njenom koeficijentu doprinosa, tj. $a_i x_i$.
2. *Aditivnost.* Svaka funkcija u modelu je zbir doprinosa pojedinih aktivnosti, npr. $f(x) = \sum_i a_i x_i$.
3. *Djeljivost.* Varijable odluke se mogu podijeliti na vrijednosti koje nisu cijele, uzimajući u obzir razlomljene vrijednosti, dok god su ograničenja zadovoljena.
4. *Sigurnost.* Pretpostavljamo da su vrijednosti parametara u modelu poznate sa nekom dozom sigurnosti ili se barem tako tretiraju. Dobijeno optimalno rješenje je optimalno za specifični formulisani problem. Ako su vrijednosti parametara pogrešne, tada rezultirajuće rješenje po pravilu ima malu (upotrebnu) vrijednost.

3.4 Teorija linearog programiranja

U ovom poglavlju ćemo formalno izvesti generalne uslove pod kojima se problem linearog programiranja može riješiti. U osnovi, pokazaćemo da ako je problem linearog programiranja dopustiv i postoje vrhovi, onda postoji i optimalno rješenje problema LP-a. Kako smo ranije neformalno pomenuli, to rješenje se nalazi upravo u jednom od vrhova poliedra.

Prije nego što ovo i formalno pokažemo, uvedimo nekoliko osnovnih definicija i pratećih teorema.

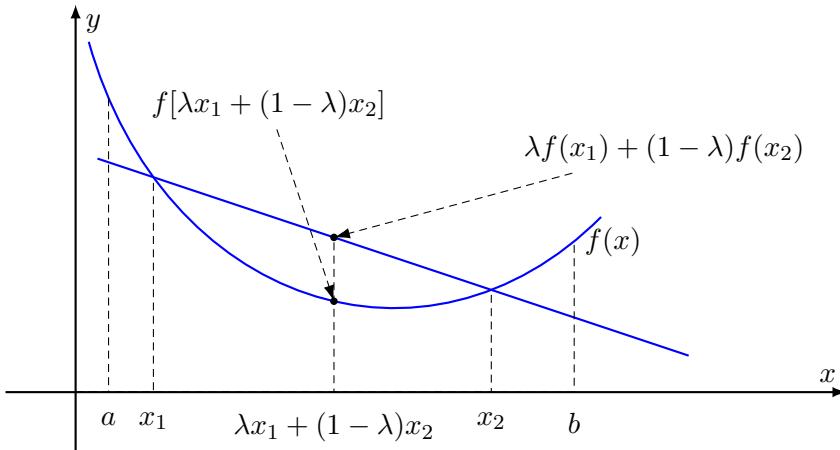
Definicija 1 Za skup $S \in \mathbb{R}^n$ kažemo da je konveksan akko za sve $x, y \in S$ i $\lambda \in [0, 1]$ vrijedi $\lambda x + (1 - \lambda)y \in S$

Definicija 2 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 $\lambda \in [0, 1]$.

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



Slika 3.5: Konveksna funkcija: geometrijska interpretacija.

Definicija 3 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 .

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

Teorema 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 je ta tačka predstavlja i tačku globalnog minimuma.

Definicija 4 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 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 konveksan skup. To je važna karakteristika, jer se uslovima problema LP-a formira odgovarajući poliedar, a kako je on konveksan skup, prethodna teorema nam garantuje da, uz pretpostavku da postoji lokalni minimum (maksimum), onda postoji i globalni minimum (maksimum), tj. optimalno rješenje zadatog problema LP-a.

Propozicija 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 $y \in P$, to je $a^T y \geq b$. Sada imamo sljedeće:

$$a^T(\lambda x) \geq \lambda b \quad (3.8)$$

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

Sabirajući posljednje dvije nejednakosti, dobijamo

$$a^T(\lambda x + (1 - \lambda)y) \geq b, \quad (3.10)$$

odakle slijedi da $\lambda x + (1 - \lambda)y \in P$, pa imamo tvrdnju.

(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 teoreme vrijedi sljedeća posljedica.

Posljedica 1 *Poliedar P nekog problema LP-a je konveksan skup.*

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

Nadalje, formalno definišimo ekstremne tačke, vrhove i bazna dopustiva rješenja koja su već ranije pomenuta u ne tako strogom kontekstu.

Definicija 6 *Neka je P poliedar u \mathbb{R}^n . Vektor $x \in P$ se naziva ekstremna tačka poliedra P ako ne postoji vektori $y, z \in P$, $y \neq x, z \neq x$ i skalar $\lambda \in [0, 1]$ takvi da je $x = \lambda y + (1 - \lambda)z$.*

Drugim riječima, ekstremna tačka x ne leži između neke dvije tačke y, z poliedra P . Možemo reći i ovako: One tačke koje se ne mogu napisati kao netrivijalna konveksna kombinacija tačaka iz poliedra čine skup ekstremnih tačaka.

Definicija 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 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 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 aktivan u x^* .

Definicija 9 Neka je zadan poliedar iz Definicije 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 nezavisan onda se x^* naziva bazno rješenje.

Definicija 10 Bazno dopustivo rješenje x^* je bazno rješenje koje zadovoljava uslove linearног programa.

Pokažimo sada da su vrhovi, ekstremne tačke poliedra i bazna dopustiva rješenja predstavljaju istu stvar kada je riječ o LP-u.

Teorema 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 ekvivalentiju sve tri tvrdnje.

$1 \Rightarrow 2$. Neka je x^* vrh poliedra. Iz definicije slijedi da postoji $c \in \mathbb{R}^n$ tako da $c^T x^* < c^T x$, za sve $x \in P, x \neq x^*$. Pretpostavimo 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 pretpostavke je $c^T x^* < c^T y$. Sada je

$$\begin{aligned} c^T x^* &= c^T(\lambda y + (1 - \lambda)z) = c^T(\lambda y) + c^T((1 - \lambda)z) > \\ &> \lambda c^T y + (1 - \lambda)c^T z = c^T y. \end{aligned} \tag{3.11}$$

što je nemoguće, odakle slijedi tvrdnja.

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

$$a_i^T x \geq b_i, i \in C_1 \tag{3.12}$$

$$a_i^T x = b_i, i \in C_2. \tag{3.13}$$

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

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

Pokažimo još da za $i \notin I$ vrijedi $a_i^T x^* \geq b_i$, 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 $a_i^T x^* > b_i$, odakle slijedi da je $y \in P$. 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 $\epsilon \in (0, \frac{b_i - a_i^T x^*}{a_i^T d})$ 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 ona takođe vrh poliedra P . Iz prepostavke, za BDR, 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

$$c^T x = \sum_{i \in I} a_i^T x \geq c^T x^* \tag{3.14}$$

Iz toga vidimo da je x^* rješenje LP problema minimizacije. Jednakost u (3.14) 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 uslove koji treba da vrijede za postojanje BDR-a, tj. ekstremne tačke poliedra. Takođe, daćemo i postupak za konstrukciju BDR-a, te ćemo uvesti pojam degenerativnog rješenja.

Teorema 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 tako 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 zaključujemo da bismo konstruisali bazno rješenje, sljedeći koraci se izvršavaju:

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

$$x = (x_1, \dots, x_n) := \begin{cases} 0, j \notin \{p_1, \dots, p_m\} \\ x_{p_i} = (x_B)_i, j \in \{p_1, \dots, p_m\} \end{cases},$$

čime dobijamo bazno rješenje.

Varijable x_{p_1}, \dots, x_{p_m} nazivamo *bazne varijable*, a vektore A_{p_1}, \dots, A_{p_m} *bazni vektori*, indekse p_1, \dots, p_m *bazni indeksi*, dok matricu B matrica baze.

Definišimo sada pojam *degenerativnog* rješenja i degenerativnog baznog rješenja. Pokazaćemo kako se na primjeru pronalazi bazno rješenje te kako izgleda degenerativno bazno rješenje.

Definicija 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 12 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 neka je x bazno rješenje. Vektor x nazivamo *degenerativno bazno rješenje* ako je više od $n - m$ komponenti u vektoru x jednako 0.

Primjer. 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 , recimo prve dvije, odakle je matica $B = \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}$. Riješimo sistem jednačina (stavka 3), odakle je rješenje $x_B = [1 \ 1]^T$, a krajnje rješenje je $x^* = [1 \ 1 \ 0]^T$. Kako je x^* aktivan 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 odabrat 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. U narednoj definiciji objašnjavamo pojam pravca u poliedru.

Definicija 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 $\lambda \in \mathbb{R}$.*

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

Teorema 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 vektore među kolonama matrice A .*

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

Teorema 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 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 $O^* = \{x \in \mathbb{R}^n \mid Ax \geq b, c^T x = v\}$ je takođe poliedar, $O^* \subset P$. Prethodna teorema nam kaže 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 prethodne teoreme), označimo je 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 postoje $y, z \in P$ tako da $q^* = \lambda y + (1 - \lambda)z$, $y \neq q^*, z \neq q^*, \lambda \in [0, 1]$. Kako je $q^* \in O^*$, imamo

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

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

Ova važna 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.

Posljedica. 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.

Kako ćemo vidjeti u narednom poglavlju, upravo ove tvrdnje motivišu konstrukciju simpleks metode, koja je bazirana na iterativnim posjećivanju vrhova poliedra u cilju pronašlaska optimalnog rješenja.

3.5 Neke tehnike u modelovanju problema linearog programiranja

Funkcija absolutna vrijednost. Prepostavimo da imamo model dat sa

$$\begin{aligned} \min & \sum_{j \in J} c_j |x_j| \\ & \sum_{j \in J} a_{ij} x_j \leq b_i, \forall i \in I \\ & x_i \in \mathbb{R}, \forall i \in I. \end{aligned}$$

Kao što znamo, absolutna vrijednost nije linearna funkcija, ali se ovaj model može transformisati u model linearog programiranja na sljedeći način: svaku slobodnu varijablu ćemo zapisati kao razliku dvije pozitivne varijable, tj. $x_i = x_i^+ - x_i^-$, a absolutnu vrijednost kao zbir dvije pozitivne varijable, tj. $|x_i| = x_i^+ + x_i^-$. Prema tome, prethodni model se transformiše u

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

Ostavljamo čitaocu da pokaže da ova dva modela imaju jednaka optimalna rješenja.

Min-max problem. Neka je dat sljedeći model:

$$\begin{aligned} & \min_x \max_{k \in K} \sum_{j \in J} c_{kj} x_j \\ & \text{s.t.} \\ & \sum_{j \in J} a_{ij} x_j \leq b_i \quad \forall i \in I \\ & x_j \geq 0 \quad \forall j \in J. \end{aligned}$$

Ovaj model se transformiše u model LP-a uvodeći varijablu

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

koja označava maksimalnu cijenu. Ograničenja koja treba da budu dodana u početni model poslije smjene da bi ograničili vrijednost varijable z su:

$$\sum_{j \in J} a_{kj} x_j \leq z \quad \forall k \in K. \quad (3.15)$$

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 varijable z će težiti da bude što manja moguća i jednaka maksimalnoj cijeni nad skupom K . Prema tome, početni model se transformiše u:

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

Frakcionala funkcija cilja. Neka je dat sljedeći model:

$$\begin{aligned} & \min \frac{\sum_{j \in J} c_j x_j + \alpha}{\sum_{j \in J} d_j x_j + \beta} \\ & \sum_{j \in J} a_{ij} x_j \leq b_i, \forall i \in I \\ & x_j \geq 0, \forall j \in J. \end{aligned}$$

Ovdje je funkcija cilja data kao količnik dvije linearne funkcije. Potrebno je transformisati ovu funkciju u linearnu da bismo imali linearni program. Ovakvi modeli su česta pojava u modelovanju finansijskog planiranja. 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 varijable y_j i parametra t , tako da je $y_j = tx_j$. U nastavku, prepostavljamo da je imenilac u funkciji cilja pozitivan (slično je izvođenje i sa negativnim imeniocem, s tim da imamo znak “-” ispred, što će promijeniti znak nejednakosti u ograničenjima krajnjeg modela). Dakle, uvedemo $t = \frac{1}{\sum_{j \in J} d_j x_j + \beta}$. Time dobijamo model

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

Množeći prvo ograničenje sa $t > 0$ te uvodeći smjenu $y_j = tx_j$, dobijamo (ekvivalentan) model:

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

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

Domenske granice u ograničenjima. Neka je dat model

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

Direktnom transformacijom dobijamo sljedeći LP (konjunkcija ograničenja):

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

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:

$$\begin{aligned} & s_i + \sum_{j \in J} a_{ij} x_j = u_i, \forall i \in I \\ & 0 \leq s_i \leq u_i - l_i, \forall i \in I. \end{aligned}$$

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

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

3.6 Zadaci

1. Kompanija Healthy Pet Food Company proizvodi dvije vrste hrane za pse: Meaties i Yummies. Svako pakovanje Meaties hrane sadrži 2 kilograma žitarica i 3 kilograma mesa; svako pakovanje Yummies -a sadrži 3 kilograma žitarica i 1,5 kilograma mesa. Kompanija vjeruje da može prodati onoliko hrane za pse koliko može napraviti. Meaty se prodaje 2,80 \$ po paketu, a Yummies 2,00 \$ po paketu. Proizvodnja kompanije Healthy ograničena je na nekoliko načina. Prvo, kompanija može kupiti samo do 400.000 kg žitarica svakog mjeseca za 0,20 \$ po kg; može kupiti samo do 300.000 kg mesa mjesečno za 0,50 \$ po kg. Osim toga, za proizvodnju Meatiesa potrebna je posebna mašina, a proizvodni kapacitet ove mašine je 90.000 paketa mjesečno. Varijabilni troškovi miješanja i pakovanja hrane za pse iznose 0,25 \$ po pakovanju

za Meaties i 0,20 \$ po paketu za Yummies. Pretpostavimo da ste vođa odjela hrane za pse ove kompanije. Na koji način bi trebali upravljati odjelom proizvodnje kako bi maksimizovali profit?

2. Kompanija proizvodi velike industrijske ključeve za cijevi u jednoj od svojih fabrika. Odjel za marketing procjenjuje potražnju za ovim proizvodom za sljedećih 6 mjeseci, koja je data 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 \$ po ključu u standardnom režimu proizvodnje. Dodatnih 80 ključeva mjesečno može se izraditi prekovremenim radom po cijeni od 45 \$ po ključu. Ključevi se mogu napraviti unaprijed i držati u zalihamama za kasniju isporuku po cijeni od 3 \$ mjesečno po ključu. Mjesečna potražnja za ključevima mora biti zadovoljena svakog mjeseca. Pred kraj decembra (početak januara) firma 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 minimiziranjem ukupnih troškova nastalih u proizvodnji i isporuci ključeva. Modelovati ovaj problem.

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

Januar	30
Februar	15
Mart	15
April	25

Proizvodni kapacitet kompanije je 30000 računara mjesečno. Moguće je proizvesti i više proizvoda, ali po višoj cijeni, gdje je tada cijena računara 300 \$ po komadu umjesto 250 \$. U startu na lageru je 2000 računara. Troškovi skladištenja su 25 \$ po jedinici proizvoda koji ostanu na zalihi na kraju mjeseca. Pretpostavljamo da su skladišni kapaciteti neograničeni. Proizvodnja počinje 1. januara. Kako planirati proizvodnju za sljedeća četiri mjeseca tako da se zadovolje svi zahtjevi, a troškovi svedu na minimum?

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

Odjeljak	Težinski kapacitet	Prostorni kapacitet
Prednji	10	6800
Središnji	16	8700
Stražnji	8	5300

Osim toga, težine tereta koji su smješteni u odgovarajućim pregradama moraju biti istog udjela kapaciteta težina da bi se održala ravnoteža aviona. 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 koliko (ako postoji i jedan) svakog tereta C_1, C_2, C_3 i C_4 treba prihvati i kako ga raspoređiti po odjeljcima da bi ukupna dobit na letu bila maksimalna.

Uputstvo. Definisati varijable odluke $x_{i,j}$ kao broj (tona) tereta i (za $C_i, i = 1, \dots, 4$, respektivno) koji se skladišti u odjeljak j , $j = 1, 2, 3$ (prednji, središnji, stražnji).

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

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 \$, respektivno. Proizvod 1 mora biti proizведен na obje mašine X i Y (u početku proizvodnje obje mašine treba da budu pokrenute), ali ostali proizvodi mogu biti proizvedeni na bilo kojoj mašini.

Tvornica je vrlo mala i to znači da je podni prostor vrlo ograničen. Proizvodnja na nivou sedmice se skladišti na $50m^2$ podne površine pri čemu svaki proizvod zauzima 0.1, 0.15, 0.5 i 0.05 (kvadratnih metara) proizvoda 1, 2, 3 i 4, respektivno.

Zahtjevi kupaca su da količina proizvedenog proizvoda 3 treba biti povezana s količinom proizvedenog proizvoda 2. Preciznije, tokom sedmice trebalo bi se proizvesti približno dvostruko više jedinica proizvoda 2 nego proizvoda 3. Mašina X je van pogona (zbog održavanja/zbog kvara) 5% vremena, a za mašinu Y je to 7% vremena. Pretpostavljajući radnu sedmicu koja iznosi 35 sati, formulišite model proizvodnje ovih proizvoda preko linearног programa da bi se maksimizovao profit.

Uputstvo. Varijable odluke se tiču količine proizvoda i koje se proizvode na svakoj od mašina. Dakle, varijabla x_i označava broj proizvoda i proizvedeni na sedmičnom nivou na mašini X , dok je y_j – broj proizvoda j proizvedeni na sedmičnom nivou na mašini Y , $i \in \{1, 2, 3, 4\}$, $j \in \{2, 3, 4\}$. Varijablu y_1 ne definišemo jer se proizvod 1 izvršava na obje mašine.

6. Grafičkom metodom riješiti sljedeći problem

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

7. Grafičkom metodom riješiti sljedeći problem

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

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 odrediti jesu li ona degenerativna.

Glava 4

Simpleks Metod

Simpleks metod je jedan od najpoznatijih i najefikasnijih algoritama za rješavanje problema linearog programiranja. Osnovna ideja leži u traženju optimalnog rješenja LP-a posjećivanjem vrhova dopustivog regionala problema, na način da se vrhovi uskcesivno posjećuju 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 linearog programiranja, spada među najznačajnije algoritme XX vijeka.

Neka je dat problem LP-a u kanonskom obliku:

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

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

Definicija 14 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 , a gdje su na mjesto nebaznih indeksa N postavljene 0. Kao što je rečeno, ideja simpleks metode je “šetanje” po vrhovima poliedra dok ne nađemo optimalno rješenje. Krenimo pravcem d u početnom BDR 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*, treba da je 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 ; neka je to $x_j, j \in N$. Definišemo $d = (d_B, d_N)$, gdje su koordinate od (nebaznog dijela) d_N svugdje 0, osim na koordinati j na koju je postavljena vrijednost 1. Bazni dio vektora d , koji je označen sa d_B , 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 je 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 protivnom, 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 $x_{p_i} + \lambda d_{p_i} > 0$, pa imam zadovoljen uslov nenegativnosti.
- Ako je x degenerativno bazno dopustivo rješenje, slijedi da više od $n - m$ komponenti od x je jednako 0. Za nebazni slučaj kao i u prvom slučaju 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 $d_{p_i} < 0$, to bi slijedilo da ne možemo naći $\lambda > 0$ tako da $x_{p_i} + \lambda d_{p_i} > 0$ što stvara problem (dopustivost narušena). Pojava ovog slučaja se na sreću može zaobići primjenom određenih pravila izbora nove bazne koordinate umjesto koordinate j (pokazaćemo to u nastavku ove sekcije).

Ostaje nam pitanje odabira nebazne varijable x_j . U pomoć pozivamo 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, odaberimo (nebazni) indeks j pri čemu 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 vrijednost \bar{c}_j *doprinos nebazne varijable* x_j .

Iz sljedeće teoreme dobijamo uslove optimalnosti rješenja.

Teorema 6 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 sljedeća dva slučaja:

- Ako $d > 0$, onda je $x + \lambda d \geq 0$, za sve $\lambda > 0$, pa je $\lambda^* = \infty$, tj. ciljna funkcija je neograničena.
- Ako postoji indeks i , td. $d_i < 0$, onda za dovoljno malo $\lambda > 0$ tako da $x_i + \lambda d_i \geq 0$ dobijamo $\lambda \leq -\frac{x_i}{d_i}$. Prema tome

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

Za indeks l za koji se postiže ovaj minimum, vrijedi $x_{p_l} + \lambda^* d_{p_l} = 0$. Da je ovo upravo bazno dopustivo rješenje problema, govori nam sljedeća teorema.

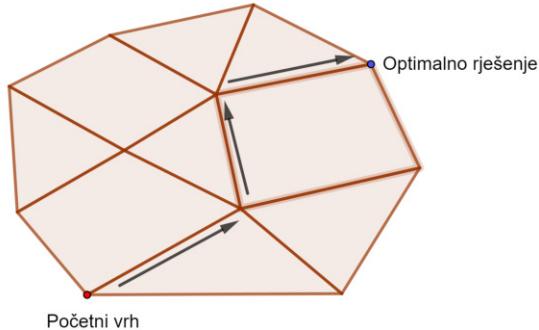
Teorema 7 Kolone $A_{p_i}, i \neq l, i = 1, \dots, m$ i A_j su linearne nezavisne, pa je matrica $[A_{p_1}, \dots, A_{p_{l-1}}, A_j, A_{p_{l+1}}, \dots, A_{p_m}]$ regularna. Vektor $x + \lambda^* d$ predstavlja bazno dopustivo rješenje problema (4.1).

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 nađen, algoritam staje 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 je $f = -\infty$, te algoritam staje.
4. Tada je za barem neki i , $d_i < 0$, pa računamo

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

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



Slika 4.1: Iteracije simpleks metode (vizuelizacija).

4.2 Tabelarni oblik simpleks metoda

U osnovi, simpleks algoritam se sastoji od stalne izmjene jedne kolone baze B sa nekim nebaznim vektorom u iterativnoj šemi u cilju smanjenja trenutne vrijednosti cilje funkcije, dok god je to moguće. U praktičnom računanju, rad sa tabelarnim prikazom koraka simpleks metoda je u mnogome lakši (kompaktniji i prirodniji) nego koristiti standardan oblik LP-a. Tabelarni simpleks prikaz se može vidjeti u Tabeli 4.1.

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

Tabela 4.1: Simpleks tabela

Na osnovu ovakve tabele, te koraka simpleks metode opisanih u prethodnoj sekciji, sljedeći koraci se izvršavaju u tabelarnoj simpleks formi:

- 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, odabrati 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 $-\infty$, algoritam se prekida.
- Za svaki $u_i > 0$, računati x_{p_i}/u_i , i neka je omjer najmanji za indeks l . Kolona A_j ulazi u bazu, dok kolona A_{p_l} izlazi 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 postoji dva ili više redova koji su kandidati za ulazak u bazu. Ako se odabere nepovoljan kandidat, postupak može da proizvede degenerativno BDR ili čak da dođe do ciklusa. Da bi se to spriječilo, Robert Bland (1977. godine) je uveo pravilo za pivotiranje danas poznato pod nazivom *Blandovo pravilo* koje:

1. Među svim nebaznim varijablama koje su pogodne za ulazak u bazu odaberemo onu najmanjeg indeksa.
2. Među svim baznim varijablama koje su pogodne za izbacivanje iz baze odaberemo onu sa najmanjim indeksom.

Može se pokazati da poštujući ovakva pravila, osigurava se da simpleks metoda završava u konačno mnogo koraka. Blandovo pravilo nije jedino pravilo za izbjegavanje ciklusa, postoji i leksikografsko pravilo pivotiranja kojeg, 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 imamo problem LP-a gdje treba da maksimizujemo vrijednost funkcije cilja, pa onda primijenimo simpleks metod, koraci metoda ostaju isti, uz malu izmijenu u koraku 2, gdje se mijenja znak nejednakosti (jer se mijenja znak koeficijenata u funkciji cilja). Dakle, posmatraju se doprinosi za koje je $\bar{c}_j > 0$.

4.3 Primjena simpleks metoda

Transformišimo standardni oblik problema LP-a

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

na kanonski oblik pogodan simpleks metodi ($Ax = b$) dodavanjem “izjednačavajućih” varijabli s na sljedeći način:

$$c^T x \rightarrow \max \quad (4.2)$$

$$Ax + s = b \quad (4.3)$$

$$x \geq 0, s \geq 0. \quad (4.4)$$

$$(4.5)$$

što u matričnom obliku odgovara zapisu

$$c^T x \rightarrow \max \quad (4.6)$$

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

$$(x, s) \geq 0. \quad (4.8)$$

$$(4.9)$$

Primjer. Neka je dat problem LP-a:

$$f = x_1 + x_2 \rightarrow \max$$

$$x_1 + 3x_2 \leq 9$$

$$2x_1 + x_2 \leq 8$$

$$x_1, x_2 \geq 0.$$

Pretvorimo ga u kanonski oblik dodavajući izjednačavajuće varijable $s = (s_1, s_2) > 0$, pa dobijamo

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

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 uzmemo za bazu vektore s_1 i s_2 . Prema tome, bazno dopustivo rješenje je $x = (0, 0, 9, 8)$, za koji je $f = 0$. Sada ažiriramo simpleks tabelu izvodeći pivotiranje. Izaberimo indeks j td. $\bar{c}_j > 0$. Neka je $j = 1$, što

odgovara varijabli x_1 čiji doprinos želimo da povećamo (jer maksimizujemo) u funkciji cilja. Izaberimo pivot vrstu tako što posmatramo odnos $\bar{b}_i/\bar{a}_{1,j}$, za sve $\bar{a}_{1,j} > 0$, $B^{-1}A_1 = \bar{a}_{1,j}$ i 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 dat elementom $\bar{a}_{1,2} = 2$. Sada izvršimo elementarne operacije pivotiranja oko ovog elementa, tako da sve elemente u koloni 1 simpleks matrice načinimo nulama, dok element u drugoj vrsti postaje 1 (Gausove eliminacije u rješavanju sistema jednačina). Primjera radi, vrsti 1 se dodaje druga vrsta pomnožena sa $-\frac{1}{2}$. Druga vrsta se dijeli sa $\frac{1}{2}$. Dok trećoj vrsti se dodaje takođe 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 transformacija.

Bazne varijable prepoznjemo jer odgovarajući vektori (kolone) zajedno čine jediničnu matricu. Prema tome, nebazne varijable su x_2 i s_2 , koje dobijaju vrijednost 0, dok ostale lako izračunavamo, odakle je $x = (4, 0, 5, 0)$. Ponavljamo prethodne korake sa novom simpleks tabelom, nalazimo pivot, tj. poziciju (i, j) . Lako je vidjeti da kolona 2 jedina odgovara nalasku odgovarajućeg pivot elementa. Dalje, za pogodnu vrstu i , imamo da je $5/(5/2) < 4/(1/2)$, prema tome $i = 1$. Dakle, pivot je $\bar{a}_{1,2} = \frac{5}{2}$. Radimo ponovo Gausove eliminacije oko prve vrste, pa se dobija simpleks tabela:

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

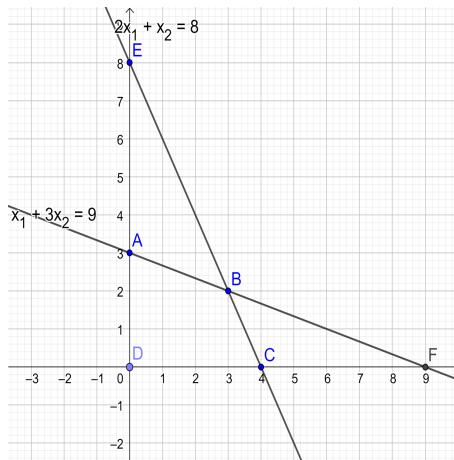
Tabela 4.4: Simpleks tabela: korak 2.

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

Komentari na zadatak. Na Slici 4.2, tačke A, B, C , i D su bazna dopustva rješenja (BDR). Tačke E i F su takođe bazna rješenja, ali nisu dopustiva. Simpleks algoritam prvo kreće od tačke A prema tački D , te završava u tački C (optimum).

Komentari na simpleks tabelu:

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



Slika 4.2: Region (bazna rješenja označena).

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

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

Primjer. Navedimo primjer jednog linearnog programa koji je neograničen, te ga riješimo simpleks metodom. Neka je dat problem LP-a:

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

Prevedimo problem na kanonski oblik, dodavanjem izjednačavajućih varijabli, pa imamo

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

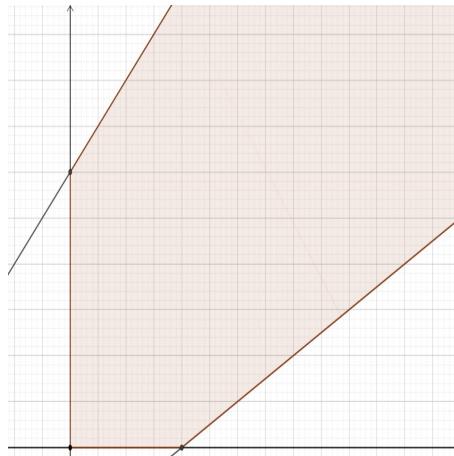
Ovom programu odgovara simpleks tabela:

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

Kako minimizujemo, kolona 2 je ona u kojoj tražimo pivota. Pivot je element $\bar{a}_{1,2} = 1$, odakle 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, ali kako su svi negativni, to znači da se vrijednost varijable x može povećavati proizvoljno, što znači da nije ograničena. Ovo se može vidjeti i na osnovu grafičke reprezentacije dopustivog regiona na Slici 4.3 (koji je neograničen).



Slika 4.3: Neograničen region.

Primjer. U nastavku dajemo još jedan primjer kada pivotiranje u simpleks metodi generiše degenerativno rješenje. Posmatrajmo sljedeći linearni program:

$$\begin{aligned} & \max 2x + 3y \\ & \text{s.t.} \\ & \quad -x + y \leq 3 \\ & \quad x - 2y \leq 2 \\ & \quad x + 6y \leq 18 \\ & \quad x, y \geq 0. \end{aligned}$$

Slično kao i u prethodnom primjeru, dodavanjem izjednačavajućih varijabli, 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

Recimo da iz nekog razloga biramo varijablu y za baznu, tj. pivotiramo oko nekog od elemenata u koloni 2. Za pivota odaberemo $\bar{a}_{1,2} = 1$ element, te nakon pivotiranja dobijamo novu simpleks tabelu:

-1	1	1	0	0	3
-1	0	2	1	0	8
7	0	-6	0	1	0
5	0	-3	0	0	-9

Ovdje počinju problemi. Pivotiranje bi trebalo da ide sa elementima u koloni 1, ali tamo nema niti jedan element strogo veći od 0 (samo jedan koji je 0, a to je $\bar{a}_{3,1}$) i trebalo bi oko njega da pivotiramo. Kada se izvrši pivotiranje, varijable \bar{b} ne mijenjaju svoje vrijednosti:

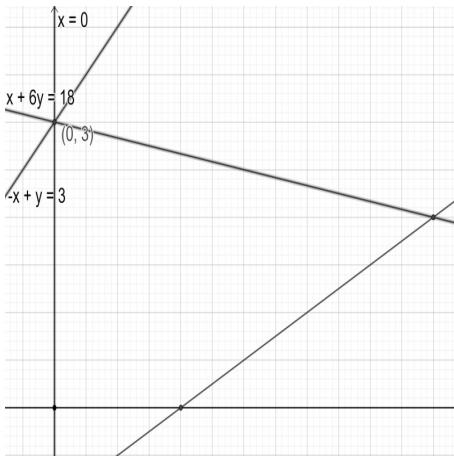
0	1	$\frac{1}{7}$	0	$\frac{1}{7}$	3
0	0	$\frac{6}{7}$	1	$\frac{1}{7}$	8
1	0	$-\frac{6}{7}$	0	$\frac{1}{7}$	0
0	0	$\frac{9}{7}$	0	$-\frac{5}{7}$	-9

Razlog za to je jer se tri prave $x = 0$, $-x + y = 3$ i $x + 6y = 18$ susreću u tački $(0, 3)$ (Slika 4.4). U prošlom koraku, kada su x i s_1 bile nebazne varijable, 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 presjeku $-x + y = 3$ i $x + 6y = 18$, što je opet ista tačka (i nema napretka vrijednosti funkcije cilja). Ovo pivotiranje se naziva *degenerativno pivotiranje*. U ovom primjeru, napustićemo vrh $(0, 3)$ nakon sljedećeg pivotiranja. U komplikovanim primjerima, kada se mnoga ograničenja mogu sresti u jednom vrhu, ostajanje u istom vrhu na nrednim simpleks koracima je velika opasnost. Ovaj slučaj se može izbjegći poštovanjem Bladovog pravila. Primjetimo da koristeći Bladovo 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:

- Da li se uvijek (kao i koliko lako) može naći bazno dopustivo rješenje za inicijalni korak simpleks algoritma?



Slika 4.4: Tačka presjeka $(0, 3)$ gdje se javlja degenerativno pivotiranje.

- Da li se simpleks algoritam uvijek prekida sa nalaskom optimalnog rješenja ili je funkcija cilja neograničena?

Odgovor na prvo pitanje se može posmatrati u ekvivalentnom obliku: da li postoji jedinična podmatrica I_m u gornjoj lijevoj blok matrici simpleks tabele? Ako je odgovor da, onda je bazu jednostavno odrediti (kako smo to i vidjeli u prethodnoj sekциji). Linearnom programu koji sadrži ograničenja $Ax \leq b, x \geq 0$ se dodaju izjednačavajuće varijable da bi uslovi nejednakosti u ograničenjima postali potrebne jednakosti. Na taj način se i formira ova jedinična matrica. Međutim, šta ako je linearan program direktno dat u kanonskom obliku, tj. sa ogarničenjima 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 *umjetne* varijable $s = (s_1, \dots, s_m)$ nakon čega treba da riješimo sljedeći problem LP-a:

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

Postoje dva moguća slučaja:

1. Ako početni problem LP-a sa jednakosnim ulovima (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, rješimo prvo problem LP-a (4.10) simpleks metodom i zaključujemo:

-
- Ako se desi slučaj 2, odustanemo od daljeg rješavanja problema.
 - Ako se desi 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 *dvoфaznoj simpleks metodi*. U nastavku dajemo primjer rješavanja jednog problema LP-a pomoću ove metode.

4.5 Primjena dvofaznog simpleks metoda

Riješimo sljedeći problem LP-a:

$$\begin{aligned} & \min x_1 + x_2 - x_3 - x_4 \\ \text{s.t. } & x_1 + 2x_2 + x_3 + x_4 = 7 \\ & 2x_1 - x_2 - x_3 - 3x_4 = -1 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

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

$$\begin{aligned} & \min x_1 + x_2 - x_3 - x_4 \\ \text{s.t. } & x_1 + 2x_2 + x_3 + x_4 + s_1 = 7 \\ & -2x_1 + x_2 + x_3 + 3x_4 + s_2 = -1 \\ & x_1, x_2, x_3, x_4, s_1, s_2 \geq 0. \end{aligned}$$

Prva faza metoda je formirati problem

$$\begin{aligned} & \min s_1 + s_2 \\ \text{s.t. } & x_1 + 2x_2 + x_3 + x_4 + s_1 = 7 \\ & -2x_1 + x_2 + x_3 + 3x_4 + s_2 = 1 \\ & x_1, x_2, x_3, x_4, s_1, s_2 \geq 0. \end{aligned}$$

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 drugi vrstu prethodne tabele sa (-1) te sabirajući sa trećom vrstom, dobijamo tabelu

1	2	1	1	1	0	7
-2	1	1	3	0	1	1
1	-3	-2	-4	0	0	-8

Iz prethodne tabele, kako minimizujemo, potražimo varijablu sa negativnim koeficijentom doprinosa, a prva takva je varijabla 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 napravimo sve nule u datoј koloni, odakle dobijamo 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 $\frac{5}{5}$, dakle pivot je $\bar{a}_{1,1} = 5$. Izvršavajući elementarne transformacije oko ovog elementa, dobijamo tabelu

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

Kako su koeficijenti doprinosa svi veći ili jednaki 0, našli smo minimum. Prema tome, bazno dopustivo rješenje je jednako $(x, s) = (1, 3, 0, 0, 0, 0)$. Dakle, nastavljamo dalje sa algoritmom, tj. 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 varijablama ω izbrišemo iz simpleks tabele;
- ostale elemente tabele ostavimo kakvi jesu.

Iz ovih koraka, dobijamo simpleks tabelu

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

Uradimo 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

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

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.

4.6 Geometrija i simpleks metod

Sljedeći primjer demonstrira odgovarajuće geometrijske korake (“šetanje”) po dopustivom regionu shodno iteracijama simpleks metode koji se izvršavaju. Neka je dat problem:

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

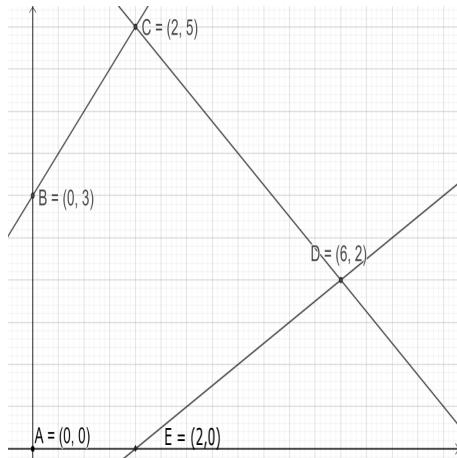
Dodajmo izjednačavajuće varijable u početni LP, odakle dobijamo

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

Kao što smo pomenuli kod simpleks metode, izjednačavajuće varijable čine pogodne bazne varijable od kojih krećemo sa simpleks metodom (jedinična podmatrica u gornjoj blok matrici simpleks tabele). Bazno rješenje, prema tome, ima oblik $(x, y, s_1, s_2, s_3) = (0, 0, 3, 2, 26)$, koje je i dopustivo. 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

Ova situacija odgovara geometrijskoj interpretaciji na Slici 4.5, što znači da se nalazimo u tački $A = (0, 0)$, koja je dopustiva, te je najbolja vrijednost za sada jednaka 0.



Slika 4.5: Korak 1

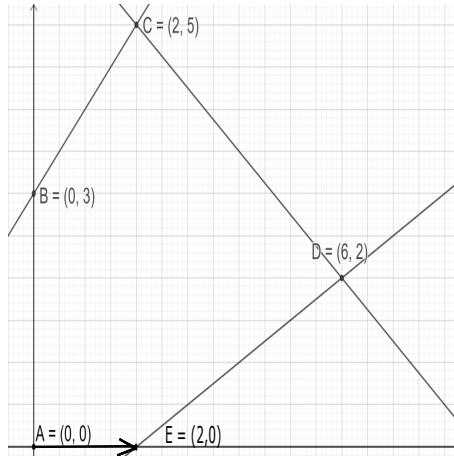
Izvršimo sada promjenu baznih vektora na osnovu simpleks koraka. Dakle, kako maksimizujemo, tražimo one koeficijente dopirnosa koji su veći od 0. Biramo prvu kolonu koja odgovara varijabli x , te nadimo pivot element: kandidati su s_2 i s_3 . Kako je $\frac{26}{3} > \frac{2}{1}$, onda je pivot $\bar{a}_{2,1} = 1$. 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

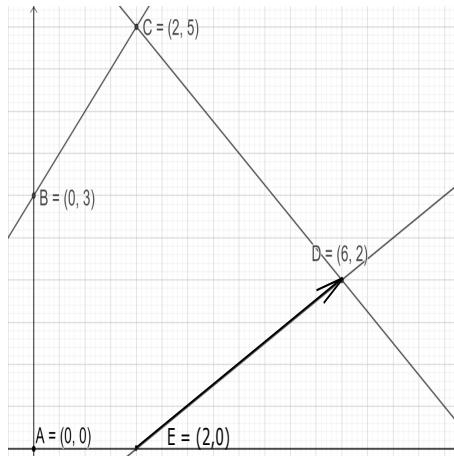
Dakle, ova tableau odgovara Slici 4.6, tj. šetnji do baznog rješenja $(x, y, s_1, s_2, s_3) = (2, 0, 5, 0, 20)$, tj. do tačke $E = (2, 0)$.

Dalje, narednog pivota nalazimo u koloni 2 (koja odgovara varijabli 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 4.6: Korak 2

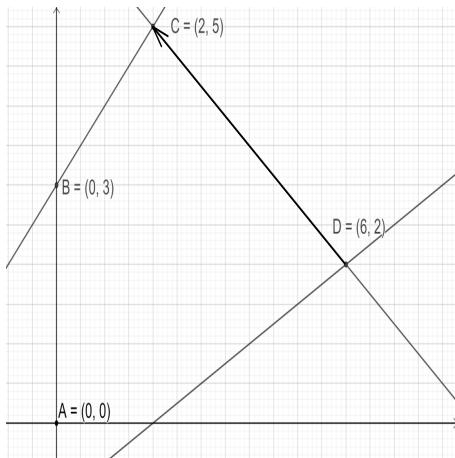


Slika 4.7: Korak 3

Ova tabela odgovara kretanju od tačke $E = (2,0)$ do tačke $D = (6,2)$, kao što vidimo na Slici 4.7. Zaključujemo da je jedina koordinata koja odgovara pozitivnom doprinosu kolona 4 simpleks tabele. Pivotiranje radimo oko elementa $\bar{a}_{1,4} = \frac{7}{10}$, pa 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 i on je $C = (2,5)$. Prethodna simpleks tabela odgovara Slici 4.8, dok svi koraci u simpleks metodi odgovaraju hodu $A \rightarrow E \rightarrow D \rightarrow C$.



Slika 4.8: Korak 4

Primijetimo da ako bi u prvom koraku simpleks metode birali mjesto varijable x ubacivanje varijable y u bazu, došli bi do optimuma tako što bi se pomijerili prvo u tačku B , pa potom u C , koja je optimum. Takođe, u ovom slučaju je potrebno generisati samo dvije simpleks tabele (a ne tri, kao što smo mi imali u ovom primjeru).

4.7 Zadaci

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

$$\max 3x + 2y$$

s.t.

$$2x + y \leq 18$$

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

$$3x + y \leq 24$$

$$x, y \geq 0.$$

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

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

$$\begin{aligned} \max z &= 2x + 3y + z \\ s.t. \\ 3x + 5y &\leq 5 \\ 2x + y - z &\leq 13 \\ z &\leq 4 \\ x, y, z &\geq 0. \end{aligned}$$

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

$$\begin{aligned} \min x_1 + x_2 + x_3 \\ s.t. \\ x_1 + 2x_2 + 3x_3 &= 3 \\ -x_1 + 2x_2 + 6x_3 &= 2 \\ -4x_2 - 9x_3 &= -5 \\ 3x_3 + x_4 &= 1 \\ x_1, x_2, x_3, x_4 &\geq 0. \end{aligned}$$

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

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

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

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

U ovom primjeru detektovati degenerativno bazno rješenje koje je nađeno u nekom koraku simpleks metode. Diskutovati zašto je ono degenerativno.

Glava 5

Dualnost & Dopustivost

Teorija dualnosti povezuje dva problema linearnog programiranja – jedan od njih je problem linearog programiranja koji maksimizuje funkciju cilja, dok je drugi linearan program koji minimizuje (ne obavezno istu) funkciju cilja. Svaki problem LP-a se može navesti u drugom, ekvivalentnom obliku na osnovu istih ulaznih podataka. Ponekad rješavanje dualnog problema može da bude puno lakše nego rješavanje početnog LP-a. Dualnost implicira da svaki problem linearног programiranja možemo analizirati na dva potpuno različita načina, ali sa ekvivalentnim (optimalnim) rješenjima.

Prije nego što uvedemo pojam dualnog problema, objasnimo motivaciju koja stoji iza nastanka teorije dualnosti. Posmatrajmo jedan LP i probajmo naći što bolju gornju procjenu za optimalnu vrijednost ciljne funkcije na osnovu ograničenja u datom modelu.

$$\begin{aligned} \max \quad & x_1 + x_2 \\ \text{s.t.} \quad & 4x_1 + 3x_2 \leq 9 \\ & x_1 + x_2 \leq 8 \\ & x \geq 0 \end{aligned}$$

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

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

kao jedna (bolja) gornja granica optimalne vrijednosti ciljne funkcije. Uradimo sada slično izvođenje u opštijem obliku, tj. iskoristimo ograničenja data u modelu zajedno u kombinaciji sa koeficijentima funkcije cilja. Neka su $y_1, y_2 \geq 0$ koeficijenti sa kojima množimo ograničenja u početnom modelu, odakle dobijamo

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

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 varijable 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 bila što bolja u (5.1), dakle koeficijent na desnoj strani što veći, potrebno je minimizovati izraz $9y_1 + 8y_2$, pod gore navedenim uslovima. Prema tome, početnom problemu, kojeg označavamo sa (P) , odgovara dualni problem

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

Generalno posmatrajući, za problem (P)

$$\max c^T x \quad (5.2)$$

s.t.

$$Ax \leq b \quad (5.3)$$

$$x \geq 0. \quad (5.4)$$

njegov dualni problem, označen sa (D) , je dat sa

$$\min b^T y \quad (5.5)$$

s.t.

$$A^T y \geq c \quad (5.6)$$

$$y \geq 0. \quad (5.7)$$

5.1 Teoreme dualnosti

Teorema 8 (Teorema o slaboj 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 $x \geq 0$ i $A^T y \geq c$, transponovanjem obje strane dobijemo $y^T A \geq c^T$. Množenjem obje strane posljednje nejednakosti sa x , dobijamo

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

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

Posljedica. 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 ovo vrijedi kada problemi (P) i (D) zamijene uloge.

Propozicija 2 Ako je x^* dopustivo rješenje za (P) , y^* dopustivo rješenje za (D) i pri tome vrijedi $c^T x^* = b^T y^*$, onda je x^* optimalno rješenje za problem (P) , dok je y^* optimalno rješenje za problem (D) .

Dokaz: Iz teoreme o slaboj dualnosti imamo: $c^T x \leq b^T y^* = c^T x^*$, za bilo koje dopustivo rješenje x od (P) , pa je x^* upravo optimalno rješenje problema (P) . Slično imamo da vrijedi 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 9 (Teorema o jakoj 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 (optimalnu) simpleks tabelu (iz koje je dobijeno rješenje x^*):

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

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 , pa dobijamo

$$f^* = (y^*)^T b \quad (5.8)$$

(posljedica kad uvrstimo $x = 0$) odakle posljedično slijedi $c = A^T y^* + c^*$. Kako je $c^* \leq 0$, imamo $A^T y^* \geq c$. Kako je $-y^* \leq 0$ pa je y^* dopustivo rješenje za (D) . Iz (5.8) je pokazano da je vrijednost ciljne funkcije u y^* upravo f^* . Iz teoreme o slaboj dualnosti slijedi da je y^* optimalno rješenje problema (D) . \square

Primjetimo da koeficijenti y^* posljednje vrste završne simpleks tabele odgovaraju "izjednačavajućim" varijablama i one nam daju rješenje problema (D) .

Primjer. Moguće je da i problem (P) i odgovarajući dual (D) nemaju dopustivo rješenje. Pogledajte sljedeći primjer:

$$\begin{aligned} \max \quad & 2x_1 - x_2 \\ \text{s.t.} \quad & x_1 - x_1 \leq 1 \\ & -x_1 + x_2 \leq -2 \\ & x_1, x_2 \geq 0. \end{aligned}$$

i pokažite da je to slučaj.

Pozabavimo se sada pojmom *komplementarnosti* (eng. *complementary slackness*).

Teorema 10 (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*

$$(Ax - b)_i y_i = 0, i = 1, \dots, m \quad (5.9)$$

$$\sum_i (A^T y - c)_j x_j = 0, j = 1, \dots, n \quad (5.10)$$

Uslovi (5.10) i (5.9) se nazivaju *uslovi komplementarnosti*.

Dokaz: Iz teoreme o slaboj dualnosti imamo:

$$c^T x \leq (A^T y)^T x = y^T A x \leq b^T y. \quad (5.11)$$

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 imamo ekvivalenciju da 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 imamo ekvivalenciju da je $(Ax - b)_j y_j = 0$ za sve j . \square

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

Primjer. Posmatrajmo problem (P) i (D) gdje su:

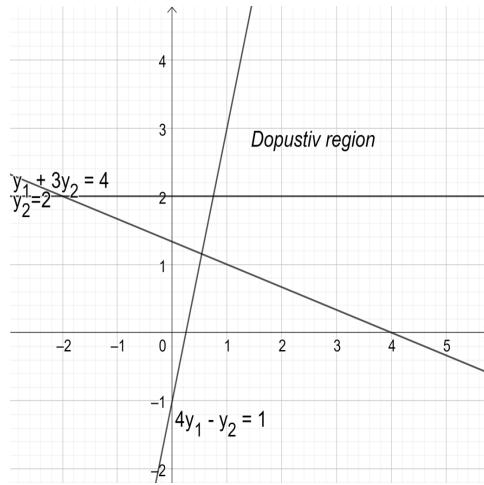
$$A = \begin{pmatrix} 1 & 4 & 0 \\ 3 & -1 & 1 \end{pmatrix}, b = \begin{pmatrix} 1 \\ 3 \end{pmatrix}, c = \begin{pmatrix} 4 \\ 1 \\ 3 \end{pmatrix}$$

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 bi imali 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.



Slika 5.1: Dualni problem: dopustiv region

Optimalno rješenje je u presjeku pravih $y_2 = 3$ i $4y_1 - y_2 = 1$, i ono odgovara $y = (1, 3)$. Ispitajmo na osnovu komplementarnosti šta je 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 $x = (0, \frac{1}{4}, \frac{13}{4})$.

Primjer. Posmatrajmo problem LP-a:

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

Ovaj problem ima 4 varijable i 2 ograničenja. Prema tome, ima smisla razmatrati dualni problem (D) datog primala (P) jer u tom slučaju dobijamo

linearni program od samo dvije varijable koji se može riješiti grafički. Odgovarajući problem (D) je dat sa:

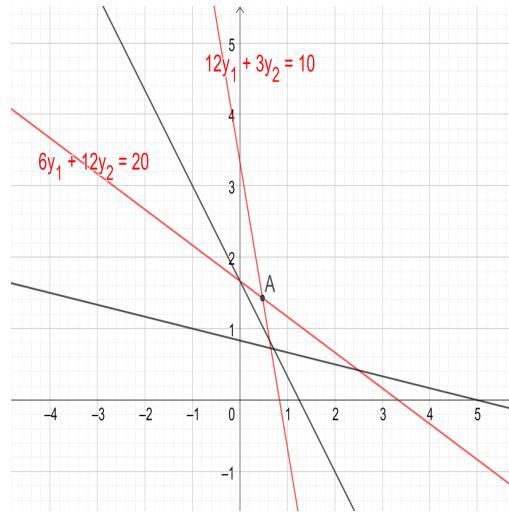
$$\begin{aligned} \min \quad & 210y_1 + 210y_2 \\ s.t. \quad & 12y_1 + 3y_2 \geq 10 \\ & 8y_1 + 6y_2 \geq 10 \\ & 6y_1 + 12y_2 \geq 20 \\ & 4y_1 + 24y_2 \geq 20 \\ & y_1, y_2 \geq 0. \end{aligned}$$

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 optimuma (vrijedi stroga nejednakost nakon uvrštavanja tačke optimuma 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 $x^* = (10, 0, 15, 0)$.

Zadatak. Vrijednost 210 u drugom ograničenju u prethodnom problemu (P) zamijenite sa 420 i diskutujte optimalna rješenja (P) i (D) na sličan način kako je izvedeno u primjeru gore.



Slika 5.2: Dopustiv region duala.

Najlakše je opisati dualnost kao odnos između problema maksimizacije i problem minimizacije, bez obzira koji je od njih bio problem (P) a koji od njih dual (D). U nastavku, kompletanu listu mogućih odnosa između ograničenja jedanog problema i varijabli u drugom problemu je data preko sljedeće tabele

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

Tabela 5.1: Odnos u varijablama i ograničenjima problema (P) i odgovarajućeg problema (D).

5.2 Metod unutrašnje tačke

Simpleks metod ne garantuje rješavanje proizvoljnog problema LP u polinomijalnom broju koraka u odnosu na veličinu ulaza. *Metod unutrašnje tačke* rješava bilo koji problem LP-a u polinomijanom vremenu. U ovoj sekciji ćemo dati konstrukciju ovog metoda, koji pokazuje da rješavanje problema LP-a pripada klasi \mathcal{P} . Iako u praksi često ne postiže dobre performanse kao simpleks metod, te se obično preporučuje korištenje simpleks metoda u rješavanju LP-a prije korištenja komplikovanih metoda, ovaj metod je veoma važan sa teoretskog stanovišta. *Metod unutrašnje tačke* (eng. *Interior point method*) je dizajniran 50-ih godina XX vijeka i sve do 80-ih godina nije bio smatran za algoritam koji se efikasno može primijeniti na probleme velikih dimenzija. Tek 1984.godine (N. Karmakar, inžinjer u IBM-u) je formalno dokazao da se ovaj algoritam izvršava u polinomijalnom broju koraka za unaprijed zadatu preciznost (optimalne vrijednosti) za probleme linearнog programiranja. U današnje vrijeme, uslijed ekstremnog poboljšanja performansi mašina, pokazalo se da je metod unutrašnje tačke kompetentan sa simpleks metodom kada je riječ o problemima LP-a velikih dimenzija (sa matricom A čiji je broj elemenata 10^9 i više).

Za razliku od simpleks metoda, koji “hoda” po granicama poliedra (dopustivog regiona) u potrazi za optimalnim rješenjem, metod unutrašnje tačke, kako i samo ime kaže, se kreće po unutrašnjim tačkama poliedra dok ne nađe optimalno rješenje problema.

Postoji nekoliko metoda ove vrste, kao što su: metod centralne putanje, redukcija potencijala, afino skaliranje, itd. Mi ćemo u ovoj sekciji obratiti

pažnju na metod centralne putanje. Da ponovimo, ova sekcija se bavi rješavanjem problema LP-a koji je dat u obliku (4.1).

Neka je par Primal-dual linearnih programa dat sa

$$\begin{array}{ll} \text{Primal} & \text{Dual} \\ \min c^T x & \max b^T y \\ \text{s.t. } Ax = b & \text{s.t. } A^T y + s = c \\ x \geq 0 & s \geq 0 \end{array}$$

Definicija 15 Lagranžijan primal-dual linearnih programa definišemo sa

$$L(x, y) = c^T x - y^T (Ax - b) - s^T x. \quad (5.12)$$

Postoji niz teorema koje obezbeđuju dovoljne optimalne uslove za LP-a. U nastavku navodimo najvažnije od njih.

Teorema 11 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 na optimalnu vrijednost Primal-a. Takođe, ako je $s^T x = 0$ donja granica je dostignuta, i x je optimalno rješenje Primal-a. \square

Teorema 12 Prepostavimo da Primal ima barem jedno optimalno rješenje. Vektor $x \in \mathbb{R}^n$ je optimalno rješenje za Primal akko postoje $y \in \mathbb{R}^m, s \in \mathbb{R}^n$ tako da

$$Ax = b \quad (5.13)$$

$$x \geq 0 \quad (5.14)$$

$$A^T y + s = c \quad (5.15)$$

$$s \geq 0 \quad (5.16)$$

$$s^T x = 0. \quad (5.17)$$

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 (5.13) i (5.14). Iz prethodne teoreme je pokazano 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 (5.17).

Dovoljni uslovi se mogu pokazati na osnovu koraka konstrukcije simpleks metode i teoreme o komplementarnosti. 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 je preko Lagranžijana L para Primal-Dual, tj. nalazak stacionarne tačke za L , odakle nalazimo optimalno rješenje (oba problema). Uslovi za nalazak stacionarne tačke se mogu izvesti iz sistema

$$\nabla_x L(x, y) = 0 \quad (5.18)$$

$$\nabla_y L(x, y) = 0, \quad (5.19)$$

što nas (uz uslov komplementarnosti za x i y) ponovo dovodi do sistema:

$$Ax = b \quad (5.20)$$

$$A^T y + s = c \quad (5.21)$$

$$XS\mathbf{1} = 0 \quad (x_j \cdot s_j = 0, \forall j) \quad (5.22)$$

$$(x, s) \geq 0, \quad (5.23)$$

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 koga razvijamo metod unutrašnje tačke. Zamjenimo ograničenja nenegativnosti u primalnom problemu sa $-lnx_j$, $j = 1, \dots, n$. Primjetite da 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. Dakle, ovakav optimizacioni problem (na prvom ortantu) nas sprečava da se vrijednost neke od varijabli x_j približi nuli (uprkos nenegativnim ograničenjima). Na osnovu ovih činjenica, definišimo sljedeći optimizacioni problem.

Definicija 16 *Barijerni program primala je dat u obliku*

$$\min c^T x - \mu \sum_{j=1}^n \ln(x_j)$$

$$s.t. \quad Ax = b$$

$$x \geq 0$$

za neki $\mu > 0$. Lagranžijan barijernog problema je dat sa

$$L(x, y, \mu) = c^T x - \mu \sum_{j=1}^n \ln(x_j) - y^T(Ax - b)$$

Sljedeću teoremu iz teorije optimizacije za skalarne funkcije sa uslovnim ekstremima, koja je od bitne važnosti u daljem razmatranju, navodimo bez dokaza.

Teorema 13 *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*:

$$\begin{aligned} Ax &= b \\ A^T y + s &= c \\ (s_1 x_1, \dots, s_m x_m) &= (\mu, \dots, \mu) \\ x, s &\geq 0. \end{aligned} \tag{5.24}$$

Primijetimo da ovaj uslov aproksimira iste uslove za primal-dual par. Interesantno je da parametar μ kontroliše udaljenost od optimalnosti, tj.

$$c^T x - b^T y = c^T x - x^T A y = x^T(c - A^T y) = x^T s = n\mu.$$

Problem u rješavanju leži u činjenici da uslovi (5.24) ne daju linearni sistem, jer treće ograničenje nije linearno.

Definicija 17 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 koje zadovoljava uslove (5.24) parametrizovano sa μ .

Primijetimo sljedeće, ako se u problemu (5.24) 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 $Ax = b$. Takođe, primijetimo da vrijednost (optimalnih) dualnih promjenjivih y upravo odgovara vrijednosti Lagranžovih množitelja. Takođe, kako μ teži ka nuli, rješenja (nelinearnog) sistema (5.24) sve preciznije aproksimira 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 $F(x, y, s) = 0, x, s \geq 0$. Ideja je 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. Mijenjući $z = 0$, definiše se nova tačka

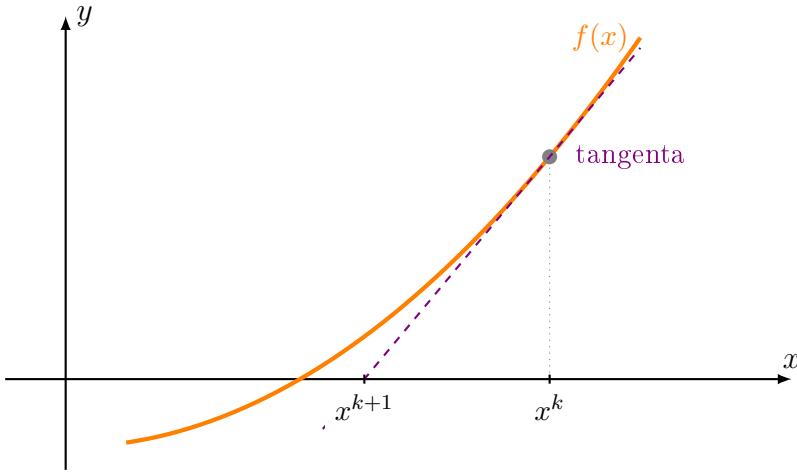
$$x^{k+1} = x^k - (\nabla f(x^k))^{-1} f(x^k). \quad (5.25)$$

Pokazuje se da niz $\{x^k\}_{k \in \mathbb{N}}$ definisan uz pomoć (5.25) konvergira ka stacionarnoj tački funkcije $f(x)$. Simulacija koraka Njutnovog metoda je data na Slici 5.3.

Rješavanje nelinearnih sistema se može generalizovati na funkcije više promjenjivih, za više informacija pogledati <http://www.ohiouniversityfaculty.com/youngt/IntNumMeth/lecture13.pdf>.

Primijenimo ovu metodu na nalaženje nule funkcije $F(x, y, s)$. Tada imamo

$$\nabla F = \begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{pmatrix}$$



Slika 5.3: Njutnova metoda u ravni.

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:

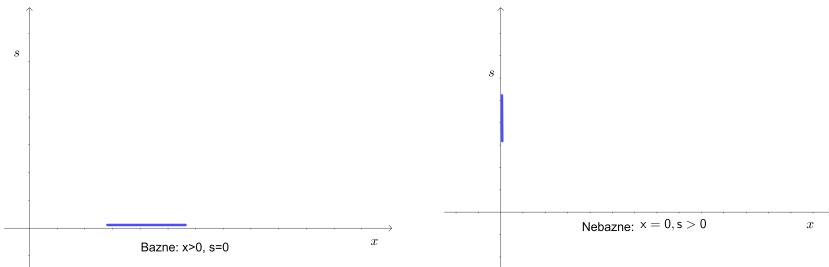
$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta s \end{pmatrix} = \begin{pmatrix} b - Ax \\ c - A^T y - s \\ -XS\mathbf{1} \end{pmatrix} \quad (5.26)$$

iz koga se potom računaju tačke naredne iteracije. Njutnov metod u nekoj iteracija obično narušava uslove nenegativnosti $x, s \geq 0$, ali to se može izbjegići (biće prikazano u nastavku). Prethodno izloženo se može lako proširiti i na slučaj kada ne vrijedi $s_i x_i = 0$, tj. kada je $\frac{1}{n} s_i x_i = \mu, \mu > 0, i = 1, \dots, n$. Na sličan način računamo Njutnov pravac u smjeru $s_i x_i \approx \mu \theta$, za $\theta > 0$, iz sistema

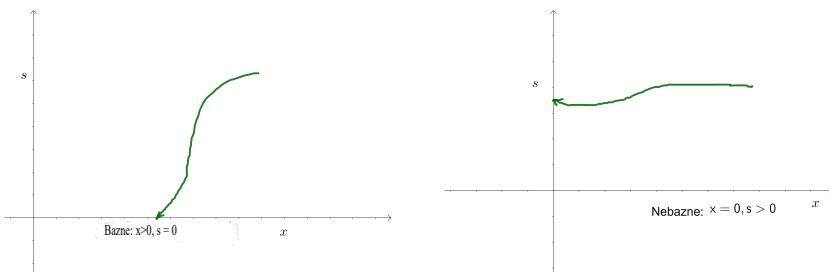
$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{pmatrix} \cdot \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta s \end{pmatrix} = \begin{pmatrix} b - Ax \\ c - A^T y - s \\ \theta \mu \mathbf{1} - XS\mathbf{1} \end{pmatrix} \quad (5.27)$$

Prema tome, *algoritam unutrašnje tačke* se sastoji od sljedećih koraka:

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 (5.27) za (x^k, y^k, s^k) i odabran $\theta = \theta_k$ te $\mu = \mu^k = (y^k)^T s^k$, odakle dobijamo $\begin{pmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{pmatrix}$



Slika 5.4: Prilazak optimalnosti kod Simpleks metode.



Slika 5.5: Prilazak optimalnosti kod Metoda unutrašnje tačke.

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.

Može se pokazati sljedeća teorema koja karakteriše *metod unutrašnje tačke*.

Teorema 14 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, pogledajte Slike 5.4 i 5.5. U njima možete vidjeti kako ova dva metoda pristupaju pretrazi optimalnih rješenja.

Paralelizovani objektno orijentisani optimizacioni softver baziran na metodi unutrašnje tačke se može naći na linku koji je dat pod referencom [25].

5.3 Farkašova lema

Još jedna primjena teorije dualnosti u LP-u se ogleda u dokazivanju sljedeće teoreme alternative.

Teorema 15 Neka je $A \in \mathbb{R}^{m \times n}$ matica i $c \in \mathbb{R}^n$. Onda jedan i samo jedan od dva sljedeća sistema ima rješenje:

1. $Ax \geq 0$ i $c^T x < 0$;
2. $A^T y = c$ i $y \geq 0$.

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

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

Šta se može reći o ovom LP-u? Ako pogledamo desnu stranu ograničenja, vidimo da je vektor $x = 0$ dopustivo rješenje problema, tako da problem ima barem jedno dopustivo rješenje. Zaključujemo da naš LP može da bude ili optimalan ili neograničen. Posmatrajmo njegov dual dat sa

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

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

Ono što zaključujemo iz odnosa definisanog primala i njegovog duala je:

- primal je optimalan akko je dual optimalan;
- primal je neograničen akko je njegov dual nedopustiv.

Odgovorimo sada na pitanje kada je primal optimalan. 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 je primal optimalan, ta vrijednost mora biti 0, i dostiže se za $x = 0$; inače je problem neograničen. Dalje, ako je primal optimalan, 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 je optimalan 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.

□

Geometrijska interpretacija Farkašove leme. Neka je dat sistem:

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

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

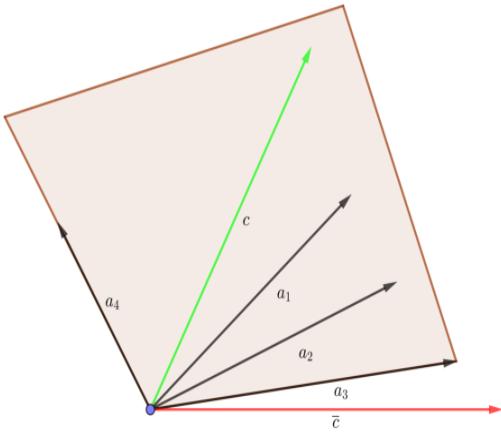
$$\begin{aligned} & 2y_1 + 3y_2 + 4y_3 - y_4 = 1 \\ & 3y_1 + 2y_2 + y_3 + 3y_4 = 4 \\ & y_1, y_2, y_3, y_4 \geq 0. \end{aligned}$$

U osnovi geometrijska interpretacija nam kaže da se vektor $c = (1, 4)^T$ može predstaviti kao linearna kombinacija vekotora vrsta matrice A , kao što vidimo na Slici 5.6, jer se nalazi unutar prostora upregnutog vektorima a_1, \dots, a_n . To implicira da početni sistem nema rješenje. Kada bi posmatrali $c = (5, 0)^T$, zaključili bi da se on ne može izraziti preko vektora vrste matrice A (jer je van prostora upregnutog sa vektrima vrste), pa prema tome postoji rješenje za početni problem u slučaju ove konfiguracije.

5.4 Fourier–Motzkin-ova eliminacija

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

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 Fourier–Motzkin-ove eliminacije, koja je strukturalno drugačija metoda od simpleks metode pa je zbog toga i navedena u ovoj knjizi. Ideja algoritma se sastoji od procedure koja će redukovati problem sa n varijabli na ekvivalentan problem sa $n - 1$ varijabli. Ekvivalentnost se podrazumijeva u pogledu



Slika 5.6: Geometrijska interpretacija Farkašove leme.

da početi problem ima riješenje akko ga ima i redukovani problem. Ideja procedure se sastoji u iterativnom izvršavanju, smanjujući broj varijabli u problemu u svakom koraku. Na kraju ćemo eventualno ostati sa sistemom od jedne varijable za koji se lako može provjeriti da li postoji barem jedna tačka između najveće gornje granice i najmanje donje granice koja zadovoljava uslove.

Glavni korak ovog metoda je redukcija problema na ekvivalentan problem sa manjim brojem varijabli. Pretpostavimo da su ograničenja data u varijablama x_1, \dots, x_n i da želimo da eliminišemo varijablu x_n . Za svaku od ograničenja

$$a_{i,1}x_1 + \dots + a_{i,n}x_n \leq b_i \quad (5.28)$$

vrijedi jedna od sljedećih alternativa

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

$i = 1, \dots, m$, 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 (5.28) kakvo i jeste (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 $n - 1$ varijabli, 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

$$\max\{L_1, \dots, L_k\} \leq x_n \leq \min\{U_1, \dots, U_l\}, \quad (5.29)$$

jer bi tada postojalo nešto između najveće donje granice i najmanje gornje granice za za vrijednost varijable x_n . Međutim, kako su sve donje i gornje granice izražene preko nepoznatih varijabli x_1, \dots, x_{n-1} , ne može se reći 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 (5.29), 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 (5.29), pa se može izabrati neki x_n između max i min gornje granice.

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

Primjer. Pretpostavimo da nam je dat sistem nejednačina:

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

Eliminišimo varijablu y iz sistema. Kako je $x - y \leq 1 \Rightarrow y \leq x - 1$, $-x + 2y \geq 1 \Rightarrow y \geq \frac{x+1}{2}$, te $3x - 5y \geq 1 \Rightarrow y \leq \frac{3x-1}{5}$, uparimo gornje i donje granice te 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 , dato

sa:

$$\begin{cases} x \geq 3 \\ x \geq 1 \\ x \geq 7 \\ x \geq \frac{1}{3} \\ x \geq 0. \end{cases}$$

Sve ove nejednakosti su donje granice. Prema tome, ako $x = 7$, vidimo da će ta vrijednost proći. Tada, donje i gornje granice za y postaju uslovi (pogledati gore): $y \leq 6$, $y \geq 4$, $y \leq 4$ te $y \geq 0$. Prema tome, možemo staviti $y = 4$ i naći dopustivo rješenje.

5.5 Zadaci

1. Napišite odgovarajući dual linearog programa:

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

2. Napišite odgovarajući dual linearog programa:

$$\begin{aligned} & \min 3x_1 - 2x_2 + 4x_3 \\ & s.t. \\ & 3x_1 + 5x_2 + 4x_3 \geq 7 \\ & 6x_1 + x_2 + 3x_3 \geq 4 \\ & 7x_1 - 2x_2 - x_3 \leq 10 \\ & x_1 - 2x_2 + 5x_3 \geq 3 \\ & 4x_1 + 7x_2 - 2x_3 \geq 2 \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

-
3. Napišite dual sljedećeg linearnog programa:

$$\begin{aligned} & \min x_1 - 3x_2 - 2x_3 \\ & s.t. \\ & 3x_1 - x_2 + 2x_3 \leq 7 \\ & 2x_1 - 4x_2 \geq 12 \\ & -4x_1 + 3x_2 + 8x_3 = 10 \\ & x_1, x_2 \geq 0 \\ & x_3 \in \mathbb{R}. \end{aligned}$$

4. Pokazati da je dual dualnog problema ponovo primalni problem (P).
5. Ako je primalni problem (P) neograničen (ima neograničenu funkciju cilja), njegov dual nije dopustiv.
6. Pokazati da vrijede odnosi dati u Tabeli 5.1.
7. Istražiti verziju simpleks metoda zvanu *Dualni simpleks metod*.
8. Koristeći Farkaš-ovu lemu, pokazati sljedeći rezultat.

Tačno jedan od sljedećih sistema ima rješenje:

- (a) $Ax > 0$;
- (b) $y^T A = 0, y \geq 0, y \neq 0$.
9. Koristeći Farkaš-ovu lemu, pokazati sljedeći rezultat. Tačno jedan od sljedećih sistema ima rješenje:
- (a) $Ax \leq b$;
- (b) $y^T A = 0, y^T b < 0, y \geq 0$.
10. Koristeći Furijer-Motzkin-ovu eliminaciju, pokazati da postoji rješenje za sljedeći sistem nejednačina:

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

-
11. Pomoću Furijer-Motzkin-ovu eliminacije, pokazati sa je sistem

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

nedopustiv.

12. 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

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

13. Formirati dualan LP za problem

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

pri uslovima

$$x_1 - x_2 + 2x_3 = 1 \tag{5.30}$$

$$2x_1 + x_2 \leq 3 \tag{5.31}$$

$$x \geq 0 \tag{5.32}$$

$$x_1 \leq 0, x_2 \geq 0, x_3 \geq 0 \tag{5.33}$$

pa riješiti oba problema.

14. Koristeći teoriju dualnosti, riješiti zadatak linearнog programiranja

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

pri uslovima

$$x_1 \geq 1$$

$$x_1 + x_2 \geq 2$$

...

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

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

Glava 6

Cjelobrojno Programiranje

Varijable odluke u mnogim realnim problemima ne dobijaju neprekidne vrijednosti. Recimo, ograničenja u problemu rasporeda ljudskih resursa po smjenama koje se tiču minimizacije broja osoblja potrebnog u svakoj od smjena pod određenim dodatnim uslovima, zahtijevaju varijable koje su diskretnog tipa. U slučaju da su sve varijable odluke u modelu problema cjelobrojne, riječ je o problemu *cjelobrojnog programiranja* (eng. *Integer programming*). U ovom odjeljku ćemo se pozabaviti *cjelobrojnim linearnim programiranjem* (ILP) gdje se još pored integralnosti (cjelobrojnosti) varijabli zahtijeva da funkcija cilja kao i ograničenja u modelu budu linearni u svojim varijablama. Prema tome, ILP ima opšti oblik:

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

U nastavku modelujmo nekoliko poznatih problema uz pomoć paradigme ILP-a.

6.1 Modelovanje nekih problema

Problem 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 proizvode treba da stavimo u ruksak čiji je kapacitet $C > 0$ tako da vrijednost proizvoda u ruksaku bude maksimalna?

Rješenje. Definišimo varijable x_i koje dobijaju vrijednost 1 ako odaberemo i -ti proizvod kojeg stavimo u ruksak, u protivnom 0, za $i = 1, \dots, n$. Ograničenje kapaciteta ruksaka se modeluje ograničenjem $\sum_{i=1}^n w_i x_i \leq C$, dok je funkcija cilja ona koja maksimizuje profit odabranih proizvoda, dakle modeluje se uz pomoć linearne kombinacije $f(x) = \sum_{i=1}^n c_i x_i$. Time je ovaj problem modelovan.

Problem rasporeda. Uprava bolnice želi da napravi raspored dežurstva po sedmičnim noćnim smjenama za svoje medicinske sestre. Potrebno je da broj medicinskih sestara za noćnu smjenu na j -ti dan bude barem (cij broj) $d_j, j = 1, \dots, 7$. Svaka medicinska sestra radi pet dana zaredom u noćnoj smjeni.

Zadatak: Pronaći minimalan broj medicinskih sestara koje bolnica treba zaposliti da bi se moglo napraviti takve smjene.

Rješenje. Označimo sa x_i broj medicinskih sestara koje radnu 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

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

Primjetimo da su u ovom problemu varijable cijelobrojne.

Problem oglašavanja. Kompanija želi da se oglašava u medijima. Postoji nekoliko alternativa oglašavanja: televizijsko, novinsko i radio oglašavanje. Cijena svakog medija sa pokrivenošću publike navedena je u sljedećoj tabeli.

	Televizija	Novine	Radio
Cijena po oglašavanju	2000	600	300
Broj gledalaca/slušalaca/čitalaca	1000	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 18.200.

Zadatak: 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 uspešnost oglašavanja, tj. funkciju $f(x) = 10x_1 + 4x_2 + 1.8x_3$. Uslovi pod kojima tražimo rješenja su modelovani na sljedeći način:

- $2000x_1 + 600x_2 + 300x_3 \leq 18.200$ (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)
- $x_i \geq 0, x_i \in \mathbb{Z}, i = 1, 2, 3$.

U ovom problemu možemo još primijetiti da u funkciji cilja nisu korišteni egzaktni koeficijenti (za broj gledalaca) već brojevi koji su proporcionalni nekoj vrijednosti (1000). Takve transformacije su invarijantne i ne utiču na potragu za optimalnim rješenjem problema, a doprinose većoj čitljivosti modela.

Problem pokrivanja. Telefonska kompanija želi da instalira antene na neka mjesta kako bi pokrila šest oblasti. Postoji pet mogućih mjesto 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 daje 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 $d > 0$. Nādalje, ne može da se desi da više od jednog signala dostiže jačinu signala d u istoj oblasti, inače bi to prouzrokovalo smetnje u prijemu. Kompanija želi da odredi gdje treba postaviti antene kako bi se pokrio maksimalan broj oblasti sa signalom.

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 može da prepozna (u našoj instanci imamo da je $N = 1$);
 - x_i : binarna varijabla koja uzima vrijednost 1 ako je antena smještena u mjestu $i \in I$, 0 inače;
 - z_j : binarna varijabla koja uzima vrijednost 1 ako je oblast $j \in J$ pokrivena signalom, 0 inače;
 - M_j : dovoljno veliki parametar, npr. za svako mjesto $j \in J$, $M_j = |\{i \in I \mid \sigma_{ij} \geq d\}|$.

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

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

pod sljedećim uslovima

- $\sum_{\{i \in I \mid \sigma_{ij} \geq d\}} x_i \geq z_i, \forall j \in J$ (barem jedna antena treba da bude uključena da bi se oblast pokrila signalom)
- $\sum_{\{i \in I \mid \sigma_{ij} \geq d\}} x_i \leq N + M_j(1 - z_j), \forall j \in J$ (maksimalan broj signala koje prijemnici mogu obraditi)
- $x_i, z_j \in \{0, 1\}, \forall i \in I, \forall j \in J$

Problem lokacija. Problemi s lokacijom često se postavljaju na sljedeći način: pretpostavimo da postoji n postrojenja (prodavnica) i m kupaca. Dva pitanja koja treba da budu obuhvaćena riješenjem 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 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 .

Rješenje. Funkcija cilja ima sljedeći oblik:

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

gdje je

- x_i : binarna varijabla koja označava da li je (1) ili nije (0) otvoreno postrojenje i ;
- $y_{ij} \in \mathbb{R}^+$ realna varijabla, koja označava dio potražnje d_j kupca j koja je ostvarena u prodavnici i .

Ograničenja ovog modela su sljedeća:

- $\sum_i y_{ij} = 1, \forall j \in \{1, \dots, m\}$ (zahtjevi kupaca treba da se ispune);
- $\sum_j d_j y_{ij} \leq u_i x_i, \forall i \in \{1, \dots, n\}$.

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

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

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

Vrijednost $M > 0$ je konstanta koja uzima neku veliku vrijednost (zbir količine potražnji svih kupaca). Međutim, u praksi se često izbjegava upotreba M -konstante, te se tako ovo ograničenje zamjenjuje sa $z_{ij} \leq x_i$ za $i = 1, \dots, n, j = 1, \dots, m$ koje osigurava bolje performanse jer se generišu preciznije linearne relaksacije o kojima će biti riječi u narednim sekcijama. Prethodni model spada u modele *binarnog linearнog programiranja*.

Problem rasporeda. Čitava klasa problema koja se naziva sekvenciranje, raspoređivanje i usmjeravanje u osnovi su predstavljeni modelima cjelobrojnog programiranja. Kao primjer, recimo da je potrebno rasporediti studente u učioniku na takav način da se minimizuje broj učenika koji ne mogu pohađati nastavu koja je njihov prvi izbor među svim ostalim izborima. Postoje ograničenja u pogledu broja i kapaciteta učionica dostupnih u bilo kom trenutku, dostupnosti predavača na fakultetu u određeno vrijeme i preferencijama studenata za određene rasporede. Jasno je da imamo varijablu $X_{i,j,n}$ za i -toga studenta koji pohađa j -to predavanje tokom n -toga 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 modeovanje, 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 Cirih (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 varijable

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

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

$$\begin{aligned} \min & \sum_{i=1}^n c_j x_j \\ & \sum_{j=1}^n a_{ij} x_j \geq 1, \forall i \in \{1, \dots, n\} \end{aligned} \tag{6.2}$$

$$\begin{aligned} & x_i \in \{0, 1\}, \forall i \in \{1, \dots, n\} \\ & a_{ij} \in \{0, 1\}, \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\} \end{aligned} \tag{6.3}$$

Ograničenje (6.2) je poznato ograničenje koje se javlja u pokrivanju problema skupa (eng. *set covering problem*).

Problem trgovackog putnika (engl. *Traveling Salesman Problem - TSP*). Problem je formulisan na sljedeći način. Na ulazu je dano n gradova. Krenuvši od svog mjesta, putnik želi da posjeti ostalih $n - 1$ gradova pod minimalnim troškovima a da se na kraju vratи na isto mjesto. 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) varijable

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

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 pojavu rješenja koja predstavljaju uniju disjunktnih podruta, što nije dopustivo rješenje za TSP. Ako prepostavimo da je rješenje dato u obliku podruta R_1, \dots, R_k , gdje je $R_i = \{v_{i_1}, \dots, v_{i_k}\}$, onda se može izbjegći pojava podruta dodavanjem dodatnih ograničenja u početni model problema, te ga takvog sukcesivno riješavati dodavanjem ograničenja dok god ne dobijemo rješenje koje nije sastavljenod više od jedne rute. Recimo, eliminacija pojave podrute R_2 (u odnosu na podrutu R_1) se vrši dodavanjem ograničenja:

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

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

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 2, \forall S \subset [n]. \quad (6.4)$$

6.2 Tehnike transformacija u ILP-u

Binarne varijable. Prepostavimo da je potrebno uključiti sljedeće aktivnosti: (i) izgraditi novo postrojenje ili (ii) poduzeti reklamnu kampanju ili (iii) razviti novi proizvod. Jasno je da su ove odluke korespondiraju ka logičkim odlukama da ili ne, pa ih, prema tome, modelujemo uvodeći (binarne) varijable $x_i \in \{0, 1\}$, $i = 1, 2, 3$. Često se nameće uslov da se najviše jedna od ovih odluka izvrši, što se modeluje sa $\sum_{i=1}^3 x_i \leq 1$. 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.

Prepostavimo da proizvođač lijekova treba da odluči hoće li koristiti spremnik za fermentaciju u proizvodnji. Ako koristi spremnik, tehnologija obrade zahtijeva da se napravi K jedinica lijekova. Dakle, količina proizvodnje y dobija vrijednosti 0 ili K , a problem se može modelovati binarnom varijablom $x_j \in \{0, 1\}$, a potom zamjenom Kx_j za y .

Logička ograničenja. Jedno od najprostijih logičkih pitanja u matematičkom programiranju je da li dati izbor varijabli odluka zadovoljava ograničenje

$$f(x_1, \dots, x_n) \leq b \quad (6.5)$$

ili ograničenje vrijedi u generalnom slučaju.

Uvedimo binarnu varijablu y na sljedeći način:

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

te napišimo

$$f(x) - My \leq b, \quad (6.6)$$

gdje je M (dovoljno velika) konstanta tako da je ograničenje zadovoljeno za sve x ako je $y = 1$. Kad god je $y = 0$ i ako je uslov (6.5) zadovoljen, onda je i ograničenje (6.6) zadovoljeno. U praksi određivanje konstante M često slijedi iz samog problema. Inače, veoma je bitno da uzmemo što manju vrijednost za M da bismo izbjegli potencijalne numeričke probleme prilikom računanja.

Alternativna ograničenja. Pogledajmo sljedeću situaciju sa alternativnim ograničenjima:

$$\begin{aligned} f_1(x) &\leq b_1 \vee \\ f_2(x) &\leq b_2. \end{aligned} \quad (6.7)$$

gdje je u modelu traženo da se zadovolji barem jedno od ograničenja. Ovo ograničenje se može dobiti kombinovanjem logičkih ograničenja te ograniče-

nja 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 odabiru da oba ograničenja u (6.7) budu (trivijano) zadovoljena, te time postanu invarijante kad je $y = 1$.

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

Pojednostavimo ovu formulaciju koristeći ograničenje $y_1 + y_2 = 1$ mjesto ograničenja nejednakosti (jer ovo povlači da je ili y_1 ili y_2 jednako 0), tj. uvrštavanjem $y_2 = 1 - y_1$, odakle dobijamo

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

Ustolovna ograničenja. Prepostavimo da želimo modelovati sljedeći scenario sa ograničenjima

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

Kako je implikacija $p \Rightarrow q$ ekvivalentna sa $\neg p \wedge q$, imamo da je ovaj scenario ekvivalentan sa

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

odakle slijedi da koristeći alternativna ograničenja možemo da modelujemo ovakav tip ograničenja.

K-alternative. Pokušajmo da modelujemo scenario da od m ograničenja

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

barem k od njih treba da budu zadovoljena. Modelovanje ovakvog scenarija vršimo na način da prvo odredimo konstante koje će ignorisati ograničenja, nazovimo ih $M_1, \dots, M_m > 0$, redom. Generalni problem u tom slučaju može biti definisan na sljedeći način:

$$f_j(x) + (1 - y_j)M_j \leq b_j, j = 1, \dots, m \quad (6.8)$$

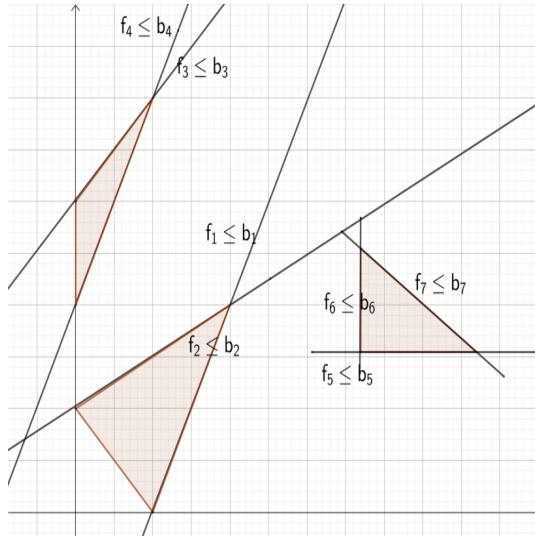
$$\sum_{i=1}^m y_i \geq k \quad (6.9)$$

$$y_j \in \{0, 1\}, j = 1, \dots, m. \quad (6.10)$$

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.

Složene alternative. Prepostavimo da se dopustiv region problema LP sastoji od tri disjunktna regiona kao na Slici 6.1. Ovakav scenario se može modelovati uz pomoć konstante veliko $M > 0$, te ograničenjem sa višestrukim izborom, odakle dobijamo:

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



Slika 6.1: Dopustiv region sa više disjunktnih regiona.

Uslov $y_1 + y_2 + y_3 \leq 2$ nam upravo ograničava da rješenje (x_1, x_2) pripada tačno jednom od (disjunktnih) dopustivih regiona.

Reporezentovanje nelinearnih funkcija. (i) Fiksne cijene. Ciljna funkcija problema minimizacije može da sadrži i neke fiksne troškove (fiksni troškovi ulaganja, fiksni troškovi postavljanja opreme itd.). Na primjer, 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. Neka je y binarna varijabla koja pokazuje kada nastaje fiksni trošak, pri čemu je $y = 1$ ako je $x > 0$, dok je $y = 0$ ako je $x = 0$. Tada doprinos cijeni proizvodnje od x jedinica proizvoda se može predstaviti sa

$$yB + cx$$

pod uslovima:

$$x \leq By \quad (6.11)$$

$$x \geq 0 \quad (6.12)$$

$$y \in \{0, 1\}. \quad (6.13)$$

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

(ii) Po dijelovima linearna funkcija. Druga vrsta nelinearne funkcije koja se može predstaviti cjelobrojnim varijablama je po-dijelovima-linearna funkcija. Radi jednostavnosti, posmatramo takve funkcije u ravni. To znači da funkcija sama po sebi nije linearна, ali kada joj se domen rastavi na komade (intervale), ona postaje linearna na svakom od tih komada. Recimo da se ova funkcija sastoji od tri po dijelovima (nezavisne) linearne funkcije, tj.

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

gdje su I_1, I_2 i I_3 disjunktni intervali na \mathbb{R} .

Da bi se modelovala ovakva funkciju cilja u modelu, varijablu x izrazimo preko tri nove varijable

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

pod uslovima

$$\begin{aligned} 0 \leq \beta_1 &\leq \delta_2^1 - \delta_1^1 = b_1 \\ 0 \leq \beta_2 &\leq \delta_2^2 - \delta_1^2 = b_2 \\ 0 \leq \beta_3 &\leq \delta_2^3 - \delta_1^3 = b_3, \end{aligned} \quad (6.14)$$

dakle slijedi da je funkcija cilja jednaka

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

Primijetimo da imamo problema na graničnim uslovima intervala, dakle 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 varijabli

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

te

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

pa se ograničenja u (6.14) mogu napisati u obliku

$$\begin{aligned} b_1\omega_1 &\leq \beta_1 \leq b_1 \\ b_2\omega_2 &\leq \beta_2 \leq \omega_1 b_2 \\ 0 &\leq \beta_3\omega_2 \leq b_3 \\ \omega_1, \omega_2 &\geq 0. \end{aligned} \tag{6.15}$$

Primijetimo da ako $\omega_1 = 0$, onda je $\omega_2 = 0$ kako bi se održala dopustivost za ograničenja nametnuta od strane β_2 , pri čemu se uslovi u (6.15) 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 (6.15) prevode 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 (6.15) prevode 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 :

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

Eliminacija proizvoda varijabli. Generalno, proizvod dvije varijable se može zamijeniti sa jednom varijablom sa dodatnim ograničenjima. Metod prezentovan ovdje se na isti način može proširiti na proizvod više od dvije varijable. Razlikujemo nekoliko slučajeva:

- x_1 i x_2 su binarne varijable. Uvedemo smjenu $y = x_1x_2$. Sljedeći uslovi će primorati da y primi vrijednost proizvoda:

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

-
- x_1 je binarna, x_2 je realna (pozitivna) varijabla, $0 \leq x_2 \leq u$. Opet uvodimo istu smjenu $y = x_1 x_2$ koja je neprekidna, te dodamo sljedeća ograničenja da primoramo y da primi 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 u obje neprekidne varijable, $l_1 \leq x_1 \leq u_1$, $l_2 \leq x_2 \leq u_2$. Uvedemo varijable $y_1 = \frac{1}{2}(x_1 + x_2)$ i $y_2 = \frac{1}{2}(x_1 - x_2)$. Tada se izraz $x_1 x_2$ može predstaviti u separabilnoj formi $y_1^2 - y_2^2$ koja se zatim može aproksimirati po-dijelovima-linearnim funkcijama. Granice za vrijednosti novih varijabli 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 stavci, ako je funkcija cilja neprekidna i nelinearna, ona se može aproksimirati uz pomoću niza po-dijelovima-linearnih funkcija. Na tu aproksimaciju potom primjenimo transformaciju ograničenja data u stavci o po-dijelovima-linearnim funkcijama.

Da bi primijenili veći broj gore navedenih transformacija ograničenja u praksi, posmatrajmo sljedeći primjer kojeg modelujemo.

Primjer. Pretpostavimo da je stanovništvo koncentrirano u I okruga unutar grada i da u okrugu $i \in [I]$ stanuje p_i ljudi. Preliminarna analiza (premjeri zemljista, politička, socijalna situacija, itd.) ograničila 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 .

Zadatak: *Potrebno je odabrati lokacija 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 varijable

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

i

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

Svaki okrug treba da bude pridružen tačno jednom vatrogasnog domu, pa je to ograničenje modelovano sa

$$\sum_{j \in J} x_{ij} = 1, \forall i \in I \quad (6.16)$$

Takođe, ne postoji okrug koji je pridružen nekorištenoj lokaciji (za vatrogasnog dom) j , tj. ako je $y_j = 0$, onda $\sum_{i \in I} x_{ij} = 0$. Ovo je uslovno ograničenje, koje se modeluje sa

$$\sum_{i \in I} x_{ij} \leq y_j |I|. \quad (6.17)$$

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

$$s_j = \sum_{i \in I} p_i x_{ij}. \quad (6.18)$$

Prepostavimo da je središnji okrug posebno osjetljiv na požar i da lokacije za dom 1 i 2 ili lokacije 3 i 4 treba da budu iskorištene da zaštite okrug. Ovakva situacija se modeluje sa

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

što je ekvivalentno ograničenjima

$$\begin{aligned} y_1 + y_2 &\geq 2y \\ y_3 + y_4 &\geq 2(1 - y) \\ y &\in \{0, 1\}. \end{aligned} \quad (6.19)$$

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

$$\sum_{j \in J} f_j(s_j) \leq B. \quad (6.20)$$

Funkcija cilja koja bi se mogla optimizovati u ovoj varajnti problema je minimizacija udaljenosti okruga koja se nalazi najdalje od svog pridruženog vatrogasnog doma (d_i):

$$\min D$$

gdje je $D = \max_i d_i$. Ekvivalentno se ova funkcija cilja može napisati kao:

$$\begin{aligned} &\min D \\ &D \geq d_i, \forall i \in I \\ &s.t. \\ &(6.16) - (6.20). \end{aligned}$$

Još je ostalo zamijeniti svaku funkciju $f_j(s_j)$ aproksimacijom cjelobrojnog programiranja kako bismo izvršili cjelokupno modeliranje. Detalje ostavljamo čitaocu kao zadatak. Ako bi funkcija $f_j(s_j)$ sadržala fiksne troškove, tada se ne bi trebale uvoditi nove varijable fiksnih troškova – u tu svrhu već služi ranije uvedena varijabla y_j .

6.3 Kompleksnost cjelobrojnog programiranja

Što se tiče rješivosti problema cjelobrojnog programiranja, ona spada u NP-teške probleme, za razliku od LP-a koji je, kako smo pokazali, polinomijalno rješiv. U nastavku ove sekcije dajemo dokaz da je rješivost ILP-a zaista NP-težak problem. Ideja je da (polinomijalno) svedemo problem 3-SAT 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 sastavljena 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; u njoj postoje dvije klauzule sastavljene od po dva literala. 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 proizvoljna klauzula može redukovati u 3-CNF, u smislu rješivosti. Dakle, svaka klauzula $l_1 \vee l_2 \vee \dots \vee l_n$ se prevodi u 3-CNF na sljedeći način:

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

Imajte na umu da ove formule nisu logički ekvivalentne ali su ekvivalentne u smislu zadovoljenja. Pokazano je da je problem 3-SAT, gdje je ulaz 3-CNF formula, u opštem slučaju NP-kompletan problem.

Teorema 16 *Rješivost ILP-a je NP-težak problem.*

Dokaz: Izvedimo redukciju sa 3-SAT problema. Neka je dat 3-SAT izraz sa literalima x_1, \dots, x_n . Konstruišimo ILP instancu sa varijablama z_1, \dots, z_n gdje imamo:

- Svaka klauzula $\phi_i(x)$ je konvertovana u nejednakost gdje $z_i = x_i$, ako je x_i pozitivan literal u klauzuli ϕ , odnosno $z_i = 1 - x_i$ ako je x_i negativan literal u klauzuli ϕ (negacija ispred). 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 polinomijalnom vremenu) na instancu ILP-a

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

Dakle, iz gornje redukcije, nalazak rješenja ILP instance odgovara nalasku dodjele \bar{x} literalima tako da je formula ϕ zadovoljena. Dakle, treba da se provjeri da li je ILP instanca dopustiva. Takođe, lako je vidjeti da je ova redukcija polinomna. Prema tome, ILP je NP-težak, s obzirom da je 3-SAT problem NP-kompletan. \square

Može se pokazati i sljedeća teorema.

Teorema 17 Neka je $k \in \mathbb{Z}$. Problem odlučivanja da li postoji cijelobrojno rješenje u ILP-u (pod uslovom da minimizujemo) 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 dnevno, a deponije B 45KM dnevno. 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 najveća.
3. Marko se sprema 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 tako da ukupna kalorijska vrijednost ponesenih namirnica bude maksimalna?

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

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. Zadat je skup S koji se sastoji od n brojeva. Koristeći tehnike cjelobrojnog linearнog programiranja, podijeliti skup S na dva podskupa, tako da se zbroji 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. Problem totalnog pokrivanja. Sa D_c označimo maksimalnu duljinu koja se može pokriti instalacijom (skladišta, antene). Pod "pokrivanjem" podrazumijevamo ispunjavanje zahtjeva klijenata (koji žive u nekom mjestu). Sa c_i označimo cijenu instalacije (antene, skladišta isl.) na mjesto i . Sa d_{ij} označimo udaljenost između mjesta i i j . Zadatak ovog problema je koja mjesta odabrati (za instaliranje antene, skladišta isl.) tako da su sva mjesta pokrivena, ali da pri tome minimizujemo troškove instalacija. Sa jednog odabranog mesta se pokrivaju sva susjedna koja su u njegovom radiusu od D_c .

-
9. Problem maksimalnog pokrivanja (eng. *maximum coverage problem*). Neka je u 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 varijablu koja dobija vrijednost 1 ako je skup S_i odabran u rješenje S' , inače 0. Dodatno, sa $y_j \in \{0, 1\}$ definisemo varijablu koja dobija vrijednost 1 ako je element e_j pokriven rješenjem.
10. *P-center* problem čvorova (eng. *vertex p-center problem*). Formulacija problema je predstavljena neusmjerenim grafom $G = (I, J, E)$, gdje su čvorovi potražnje predstavljeni vrhovima $i \in I$, moguće lokacije (servisnih) objekata date su drugim skupom vrhova $j \in J$, a grane $e_{i,j} \in E$ postoje samo između čvorova $i \in I$ i čvorova $j \in J$. Nadalje, čvorova $d_{i,j} \geq 0$ dodjeljujemo pozitivne težine, što predstavlja udaljenost između čvorova i i j . Imajte 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 tog čvora.

U ovom problemu želimo postaviti p objekata kako bismo smanjili maksimalnu udaljenost između bilo kojeg čvora potražnje i njegovog servisnog objekta.

11. Prepostavimo 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 ubačeni u ova dva kofera.

12. Višedimenzionalni više-direkcioniji problem particionisanja brojeva (eng. *multidimensional multi-way number partitioning problem*). Neka je dat skup vektora S . Inače, 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 podjednake (što manje moguće).
13. Problem maksimalnog nezavisnog skupa (eng. *independent set problem*). U ulazu je dat graf $G = (V, E)$. Potrebno je naći podskup $V' \subseteq V$ maksimalne kardinalnosti tako da niti jedan par čvorova iz V' nisu susjedni.

-
14. Problem maksimalne klike (eng. *maximum clique problem*). U ulazu je 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 (postoji grana između bilo koja dva čvora iz V').
15. Proizvođač igračaka planira proizvodnju novih igračaka. Troškovi pripreme proizvodnih pogona i jedinični profit za svaku igračku dati su u nastavku:

Igračka	Troškovi pripreme	Profit
1	45000	12
2	76000	16

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

Stopi proizvodnje svake igračke date su u nastavku (u jedinicama/satu):

	Igračka 1	Igračka 2
Tvornica 1	52	38
Tvornica 2	42	23

Fabrika 1 i 2, imaju 480 i 720 sati radnog vremena za proizvodnju ovih igračaka, redom. 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. Sirova ulja dolaze u dvije kategorije:
- vegansko ulje:
 - (a) VEG1
 - (b) VEG2
 - obično ulje:
 - (a) Ulje1
 - (b) Ulje2
 - (c) Ulje3

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

VEG1	VEG2	Ulje1	Ulje2	Ulje3
115	128	132	109	114

Finalni proizvod prodaje se po 180 funti 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 ne-biljnih ulja. Pretpostavimo da u procesu rafiniranja nema gubitka težine, a troškovi prerade mogu se zanemariti.

Postoji tehničko ograničenje koje se odnosi na tvrdoću konačnog proizvoda. U jedinicama u kojima se mjeri tvrdoća, dopušteno je da ona bude između 3.5 i 6.2. Pretpostavlja se da se tvrdoća odnosi linearno sa miješanjem vrsta ulja. Tvrdoća sirovog ulja je data sljedećom tabelom:

VEG1	VEG2	Ulje1	Ulje2	Ulje3
8.8	6.2	1.9	4.3	5.1

Potrebno je odrediti šta kupiti i kako miješati sirova ulja tako da kompanija maksimizuje svoj profit.

U osnovni model uključiti i sljedeća ograničenja:

- Hrana nikad ne smije biti napravljena miješanjem više od 3 vrste ulja.
- Ako je ulje korišteno, onda se mora iskoristiti barem 30 t ulja.
- Ako su ili VEG1 ili VEG2 ulja korištena, onda i Ulje2 mora biti korišteno.

Glava 7

Algoritamske Tehnike za Rješavanje ILP-a

Za razliku od problema LP-a, koji se efikasno rješava simpleks metodom, za rješavanje problema Cjelobrojnog programiranja ne postoji generalna efikasna metoda, a efikasnost metoda kojima se rješava ovaj problem uveliko zavisi od karakteristika samog problema. Dosadašnje metode prisutne u literaturi se mogu klasifikovati na sljedeći način:

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

U nastavku ćemo opisati i analizirati ove nabrojane metode. Za model ILP-a (6.1), zanemarujući uslov da varijable treba da budu cjelobrojne (već da uzimaju realne, pozitivne vrijednosti), dobijamo tzv. odgovarajuću LP *relaksaciju* ILP-a (6.1). LP relaksacija se može efikasno riješiti uz pomoć simpleks metoda; označimo dobijeno rješenje sa x^* . Posmatrajući vektor \bar{x} , koji se dobije zaokruživanjem svake koordinate vektora x^* na njenu najbližu cjelobrojnu vrijednost, dobijamo cjelobrojni vektor. Međutim, iako je uslov o cjelobrojnosti (kakav je u odgovarajućem ILP-u potreban) ispunjen, može se desiti da ovakvo rješenje izvedeno iz LP relaksacije problema (6.1) nije dopustivo za početni ILP. U osnovi, skaliranjem desnih strana 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.

7.1 Metoda grananja i ograničavanja (B&B)

Metoda grananja i ograničavanja (engl. branch and bound, odakle potiče i skraćenica B&B) je zasnovana na poznatom principu “zavadi pa vladaj”

(engl. divide and conquer) gdje se problem rastavlja na manje dijelove, i tako rekurzivno, po potrebi. Kasnije se (optimalna rješenja) manjih problema kombinuju za dobijanje optimalnog rješenja početnog problema. U B&B metodi važan dodatak je upotreba strategije da se problem ne rastavlja na manje potprobleme ako se može procijeniti da će rješenje potproblema biti suboptimalno (lošije od trenutno najboljeg dopustivog rješenja razmatranog problema). Očigledno, na taj način dolazi do uštete u broju operacija koje je potrebno izvršiti u toku samog 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), i tako rekurzivno, dok se ne dobiju potproblemi čije relaksacije obezbjeđuju cjelobrojno rješenje ili se zaključi da potproblemi 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 će nam biti od pomoći:

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

Na bazi 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 uvodeći ograničenja: $z_i \geq \lfloor x_i^* \rfloor + 1$ ili $z_i \leq \lfloor x_i^* \rfloor$ i to je upravo mjesto podjеле dopustivog regiona. Ako ima više takvih koordinata, biramo onu koordinatu i gdje 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 (pot)problema.

-
- Ako je neko od rješenja cjelobrojno, popravljamo LB (ako smo dobili bolje dopustivo rješenje) i odgovarajući potproblem (njegov region) se dalje ne dijeli.
 - Ako rješenje x nije cjelobrojno, i pri tome je vrijednost optimuma manja od LB , taj potproblem se dalje više ne dijeli (*bound procedura*).
 - Dalje, ako je potproblem nedopustiv, problem se ne dijeli (i ne razmatra više).
 - Inače, idemo na korak 2 dijeljeći podregion na nove (manje) podregione nekog od potproblema 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 raste eksponencijalno u odnosu na veličinu ulaznog problema.

Ovo je samo okvirna shema B&B procedure. Ono što je ostalo nerazjašnjeno je koji od neriješenih potproblema 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 (potproblema) koji se rješava su sljedeći:

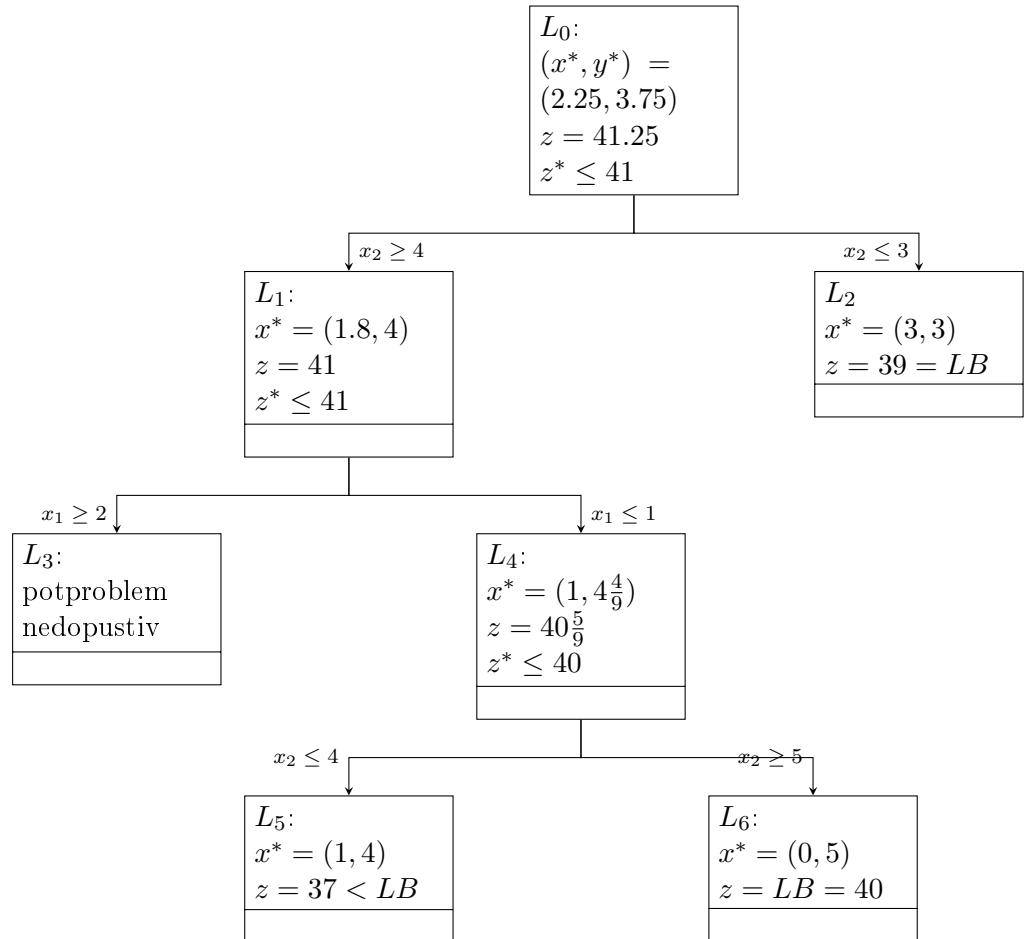
- Izabratи one aktivne čvorove koji su najdublje u drvetu, da bi se što prije stiglo do listova i potencijalnih dopustivih rješenja.
- Birati aktivne čvorove nivo po nivo. Dakle, sve aktivne čvorove (tj. odgovarajuće potprobleme) jednog nivoa rješavati, pa onda preći na čvorove idućeg nivoa.
- Koristiti heurističku funkciju u odluci koji aktivni čvor (tj. odgovarajući potproblem) prvo rješavati.

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

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

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

- Ako su koeficijenti ciljne funkcije cijelobrojni, onda $z^* \leq \lfloor z^* \rfloor$;
- Dodavanjem važećih ograničenja u sam model (fokus sljedećeg poglavljja).



Slika 7.1: Primjer Branch & Bound Metoda za ILP.

7.2 Implicitna enumeracija

Posmatrajmo sada specijalnu B&B proceduru koja se može primijeniti na rješavanje problema binarnog cjelobrojnog programiranja, gdje diskretne varijable u suštini modeluju logičke odluke (da ili ne). Ovaj algoritam ne zahtjeva da se posmatraju rješenja LP-relaksacija, već se radi sa enumeracijom. U naivnoj enumeraciji bismo izlistali sve moguće kombinacije vrijednosti koje varijable mogu da prime, pa bi se izabrala ona koja je dopustiva i daje najveću vrijednost za ciljnu funkciju. Ovaj pristup radi dobro na problemima malih dimenzija, gdje postoji svega nekoliko binarnih varijabli. Primijetimo da je u problemu sa n binarnih varijabli broj kombinacija za varijable jednak 2^n (dakle, 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

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

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

primjenjuje suprotna taktika, zadržavaju se 0-1 restrikcije za varijable, ali ignoriraju se linearna ograničenja u modelu. Ideja implicitne enumeracije je uključiti B&B proces tako što se fiksiraju neke od varijabli na 0 ili 1; varijable koje su još neodređene se nazivaju *slobodne varijable*. B&B kreće od slučaja da ne postoje varijable koje nisu slobodne. U svakom koraku fiksiramo jednu od varijabli 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) varijable 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 varijabla *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 potproblem, te $x_i = 1$, za drugi pot-

problem koji treba da se riješe.

3. Provjerimo za oba potproblema da li je rješenje (sa fiksnim dijelom) koje maksimizuje rješenje potproblema (kao u koraku 1) dopustivo. U slučaju da jeste, odgovarajući protproblem se više ne dijeli; po mogućnosti ažuriramo LB , ako je nađeno novo najbolje rješenje. Ukoliko fiksiranje dijela rješenja vodi narušavanju nekog od ograničenja bez obzira na vrijednosti ostalih varijabli, potproblem je nedopustiv i više se ne dijeli.
4. U protivnom, ažuriramo $i = i + 1$ i idemo na korak 3 sa potproblemima koji mogu dalje da se dijele, dok god je $i \leq n$. Inače, izlazimo iz petlje sa optimalnim rješenjem (LB).

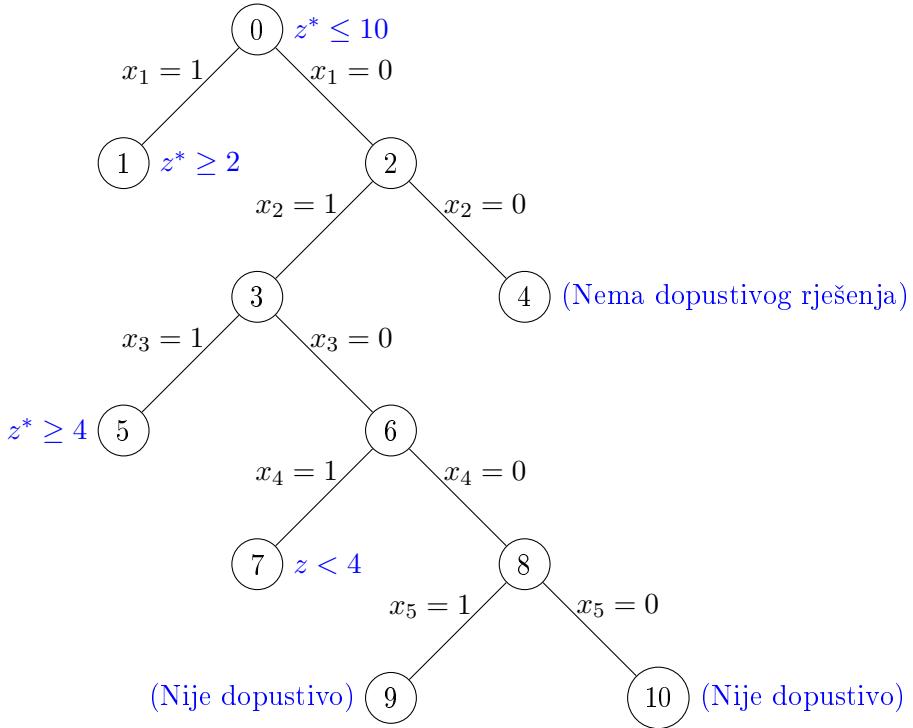
Primjetimo da i u ovom slučaju dobijamo drvo odluke (binarno drvo) gdje su na granama koje spajaju čvor na dubini i sa čvorom na dubini $i + 1$ ugrađene odluke o tome da li je varijabla x_i fiksirana na 0 ili 1.

Primjer. Na Slici 7.2 je dato drvo odlučivanja za problem

$$\begin{aligned} & \max -8x_1 - 2x_2 - 4x_3 - 7x_4 - 5x_5 + 10 \\ & \text{s.t.} \\ & \quad -3x_1 - 3x_2 + x_3 + 2x_4 + 3x_5 \leq -2 \\ & \quad -5x_1 - 3x_2 - 2x_3 - x_4 + x_5 \leq -4 \\ & \quad x_i \in \{0, 1\}, \text{ za } i = 1, \dots, 5. \end{aligned}$$

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 potproblema (odgovaraju čvorovima 1 i 2). U čvoru 2 imamo potproblem 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, odakle zaključujemo da je za ovaj potproblem $z^* = f(x^*) \geq 2$. Dakle, imamo $LB=2$.
- Problem u čvoru 2 je isti kao problem u čvoru 1, pa nastavljamo dalje sa dijeljenjem problema fiksirajući slobodne varijable.
- Sada fiksiramo promjenjivu x_2 (uz već fiksiranu odluku $x_1 = 0$). Za potproblem u čvoru 4, provjerimo da ne postoji dopuna do dopustivog rješenja za ovako fiksirane vrijednosti (narušeno ograničenje 1), pa tu



Slika 7.2: Drvo odluke za rješavanje problema binarnog programiranja.

stajemo sa njim. Za potproblem u čvoru 3, imamo sada podjelu na dva potproblema fiksirajući varijablu x_3 , tj. dva nova čvora 5 i 6. Za problem u čvoru 5, lako se vidi da za ovakve fiksirane varijable 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 varijabli x_4), odakle dobijamo dva nova potproblema, koji odgovaraju čvoru 7 i 8. Lako se zaključi da je $z^* < LB$ za potproblem u čvoru 7. Za čvor 8, imamo novo grananje (po varijabli x_5). Kako su ovdje sve varijable kompletno fiksne, primijetimo da niti jedna od ovih tačaka nije dopustiva.

- Prema tome, optimalno rješenje početnog problema je $LB = 4$ i dostiže se u tački $x = (0, 1, 1, 0, 0)$.

Potpuna enumeracija bi generisala 32 čvora za gornji problem. Implicitna enumeracija generiše svega 11 čvorova.

7.3 Metoda odsjecajućih ravnih

Metoda odsjecajućih ravnih rješava cjelobrojne programe mijenjanjem rješenja njegove relaksacije sve dok se ne dobije model čije je optimalno rješenje

cjelobrojno. Metoda ne radi na principu dijeljenja dopustivog regional problema na podregione, kao u B&B metodama, već radi sa jednim linearnim programom, kojeg poboljšava iterativnim dodavanjem novih ograničenja. Nova ograničenja sukcesivno smanjuju dopustiv region dok se ne pronađe cjelobrojno optimalno rješenje. Istorijски gledano, to je prvi algoritam razvijen za probleme cjelobrojno programiranja za koji se formalno dokazalo da konvergira u konačnom broju koraka ka optimumu. Iako se algoritam generalno smatra neefikasnim, ovaj metod je doveo do konstrukcije drugih, učinkovitijih algoritama.

Definicija 18 *Validna nejednakost za ILP je svako ograničenje koje ne eliminira niti jedno moguće cjelobrojno rješenje problema.*

Drugo ime za validnu nejednakost je *odsjecajuća ravan*. Recimo, ako imamo sljedeći problem ILP-a:

$$\begin{aligned} \max z &= 3x_1 + 4x_2 \\ \text{s.t.} \\ 5x_1 + 8x_2 &\leq 24 \\ x_1, x_2 &\geq 0, x_1, x_2 \in \mathbb{Z}, \end{aligned}$$

jedna njegova validna nejednakost je $z \leq 5$.

Definicija 19 *Konveksni omotač za problem (6.1) je najmanji (dopustivi) poliedar nekog LP-a koji sadrži sve 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 ovaj problem je takođe NP-težak, pa prema tome nije očekivano da se takav konveksni omotač može naći u generalnom slučaju. Nalazak korisnih ograničenja (ne svih) koja učestvuju u izgradnji konveksnog omotača je takođe veoma težak zadatak, mada je ovo nekad i moguće (kao u slučaju problema trgovackog putnika). Prema tome, ostaje nam naći korisne odsjecajuće ravni nezavisno od problema nalaska konveksnog omotača, što se najčešće i radi u praksi.

Posmatrajmo *problem pakovanja skupa* dat na sljedeći način. Neka je data kolekcija D rombova (u \mathbb{R}^2). Zadatak se sastoji od odabira maksimalnog broja rombova tako da se niti jedan od njih ne presjeca. 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:

$$\begin{aligned} \max \sum_{d \in D} x_d \\ x_d + x_{d'} \leq 1, \forall (d, d') \in O \\ x_d \in \{0, 1\}, \forall d \in D, \end{aligned}$$

gdje je x_d binarna varijabla koja dobija vrijednost 1 ako je romb d izabran u riješenju, inače 0. Kao što vidimo, broj binarnih varijabli u praksi može biti veoma velik, što nas rješava mogućnosti efikasne primjene neke od B&B tehnika.

Pokušajmo dodati odsjecajuće ravni u prethodni ILP. Za svaku tačku c , enumerišimo sve one (moguće) rombove iz D koji sadrže ovu tačku. Označimo takav skup sa $D(c)$. Prema tome, za svaki skup $D(c)$, najviše jedan romb može da bude odabran. Sada dodajemo sljedeća ograničenja u prethodni model

$$\sum_{d \in D(c)} x_d \leq 1, \forall c. \quad (7.1)$$

Ove odsjecajuće ravni će ubrzati rješavanje ovog problema za red veličine.

Pozabavimo se sada *Gomorijevim odsjecajućim ravnima*, generalnom metodom za dodavanje odsjecajućih ravni za ILP. Ideja je da se dobiju ravni iz nekog od ograničenja optimalne tabele 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, a s_j nebazne promjenjive. Napišimo ovu jednakost u drugačijem obliku tako da su na lijevoj strani 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, pa kako je lijeva strana cjelobrojna u tom slučaju, dobijamo da je ona 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 nebazne varijable jednake 0 u bilo kom baznom dopustivom rješenju, tada imamo

$$\bar{b}_i - \lfloor \bar{b}_i \rfloor - \sum_{ij} (\bar{a}_{i,j} - \lfloor \bar{a}_{i,j} \rfloor) s_j = \bar{b}_i - \lfloor \bar{b}_i \rfloor \geq 0.$$

Prema tome, pokazali smo da odsjecajuće ravni (7.2) ne eliminišu bazna dopustiva rješenja, pa ispunjava 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. Neka je dat sljedeći ILP problem:

$$\begin{aligned} & \max \quad 3x_1 + 4x_2 \\ & s.t. \\ & \frac{2}{5}x_1 + x_2 \leq 3 \\ & \frac{2}{5}x_1 - \frac{2}{5}x_2 \leq 1 \\ & x_1, x_2 \geq 0, \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

Riješimo ga uz pomoć Gomorijevih odsjecajućih ravni.

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

Dovedimo drugu varijablu (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 odrogarajuće 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:

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	$\frac{5}{2}$	$\frac{2}{7}$	-1	$\frac{3}{7}$
0	0	5	$\frac{5}{2}$	0	$\frac{35}{2}$

Načinimo sada jediničnu podmatricu u gornjoj lijevoj matrici \bar{A} , za treću kolonu (i odgovarajuću varijablu s_1), pivotirajući oko elementa $\bar{a}_{3,3}$, čime dobijamo

$$\begin{array}{cccccc|c} 0 & 1 & 0 & -1 & 1 & 1 \\ 1 & 0 & 0 & \frac{3}{2} & 1 & \frac{7}{2} \\ 0 & 0 & 1 & \frac{5}{2} & -\frac{7}{5} & \frac{3}{5} \\ \hline 0 & 0 & 0 & \frac{1}{2} & 7 & \frac{29}{2} \end{array}$$

Kao što vidimo, rješenje problema je $(\frac{7}{2}, 1, 0, 0, \frac{3}{5})$, što i dalje nije cjelobrojno 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:

$$\begin{array}{cccccc|c} 0 & 1 & 0 & -1 & 1 & 0 & 1 \\ 1 & 0 & 0 & \frac{3}{2} & 1 & 0 & \frac{7}{2} \\ 0 & 0 & 1 & \frac{5}{2} & -\frac{7}{5} & 0 & \frac{3}{5} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & -1 & \frac{1}{2} \\ \hline 0 & 0 & 0 & \frac{1}{2} & 7 & 0 & \frac{29}{2} \end{array}$$

Sada za varijablu x_4 (četvrta kolona), pivotiramo oko elementa $\bar{a}_{4,4}$, pa dobijamo simpleks tabelu

$$\begin{array}{cccccc|c} 0 & 1 & 0 & 0 & 1 & -2 & 2 \\ 1 & 0 & 0 & 0 & 1 & 3 & 2 \\ 0 & 0 & 1 & 0 & -\frac{7}{5} & \frac{4}{5} & \frac{1}{5} \\ 0 & 0 & 0 & 1 & 0 & -2 & 1 \\ \hline 0 & 0 & 0 & 0 & 7 & 1 & 14 \end{array}$$

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.

Rješenje ILP problema gdje su svi koeficijenti u ograničenjima cjelobrojni, je prikazano sljedećim primjerom.

Primjer. Riješimo sljedeći problem ILP-a:

$$\max 3x_1 + 4x_2$$

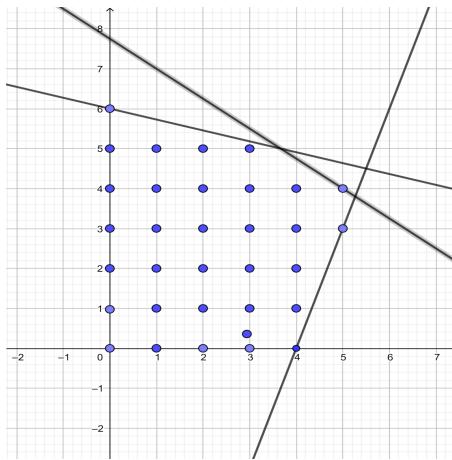
s.t.

$$3x_1 - x_2 \leq 12$$

$$3x_1 + 11x_2 \leq 66$$

$$x_1, x_2 \geq 0$$

$$x_1, x_2 \in \mathbb{Z}$$



Slika 7.3: Dopustiv skup tačaka problema.

Rješenje. Na Slici 7.3 dat je skup tačaka koje su dopustive za problem, za koji je optimalna tačka $x^* = (5, 4)$.

Riješimo sada ovaj problem uz pomoć Gomorijevih odsjecajućih ravnih. Početna simpleks tabela je data sa

$$\begin{array}{rrrr|r} 3 & -1 & 1 & 0 & 12 \\ 3 & 11 & 0 & 1 & 66 \\ \hline -3 & -4 & 0 & 0 & 0 \end{array}$$

Nakon dvije iteracije pivotiranja, dobijamo simpleks tabelu

$$\begin{array}{rrrr|r} 1 & 0 & \frac{11}{36} & \frac{1}{36} & \frac{11}{2} \\ 0 & 1 & -\frac{1}{12} & \frac{1}{12} & \frac{9}{2} \\ \hline 0 & 0 & \frac{7}{12} & \frac{5}{12} & \frac{69}{2} \end{array}$$

Optimalno rješenje je $x^* = (\frac{11}{2}, \frac{9}{2}, 0, 0)$ koje nije cijelobrojno. Dodajmo gomorijevu odsjecajuću ravan generisanu iz ograničenja u prvoj vrsti 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

$$\begin{array}{rrrrr|r} 1 & 0 & \frac{11}{36} & \frac{1}{36} & 0 & \frac{11}{2} \\ 0 & 1 & -\frac{1}{12} & \frac{1}{12} & 0 & \frac{9}{2} \\ 0 & 0 & \frac{11}{36} & \frac{1}{36} & -1 & \frac{1}{2} \\ \hline 0 & 0 & \frac{7}{12} & \frac{5}{12} & 0 & \frac{69}{2} \end{array}$$

Kako bazno dopustivo rješenje nije očigledno, primijenimo dvofazni simpleks

metod da bi izabrali bazu, odakle dobijamo tabelu

$$\begin{array}{cccccc|c} 1 & 0 & 0 & 0 & 1 & 5 \\ 0 & 1 & 0 & \frac{1}{11} & -\frac{3}{11} & \frac{51}{11} \\ 0 & 0 & 1 & \frac{1}{11} & -\frac{36}{11} & \frac{18}{11} \\ \hline 0 & 0 & 0 & \frac{4}{11} & \frac{21}{11} & \frac{369}{11} \end{array}$$

Kako su svi koeficijenti u posljednjoj vrsti veći od 0, slijedi da je optimalno bazno rješenje jednako $x = (5, \frac{51}{11}, \frac{18}{11}, 0, 0)$, koje opet nije cijelobrojno. 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

$$\begin{array}{ccccccc|c} 1 & 0 & 0 & 0 & 1 & 0 & 5 \\ 0 & 1 & 0 & \frac{1}{11} & -\frac{3}{11} & 0 & \frac{51}{11} \\ 0 & 0 & 1 & \frac{1}{11} & -\frac{36}{11} & 0 & \frac{18}{11} \\ 0 & 0 & 0 & \frac{1}{11} & \frac{8}{11} & -1 & \frac{7}{11} \\ \hline 0 & 0 & 0 & \frac{4}{11} & \frac{21}{11} & 0 & \frac{369}{11} \end{array}$$

I ovdje nije očigledno šta su bazne varijable, pa ovdje primijenimo dvofazni simpleks metod, te dobijamo

$$\begin{array}{ccccccc|c} 1 & 0 & 0 & -\frac{1}{8} & 0 & \frac{11}{8} & \frac{33}{8} \\ 0 & 1 & 0 & \frac{1}{8} & 0 & -\frac{3}{8} & \frac{39}{8} \\ 0 & 0 & 1 & \frac{1}{2} & 0 & -\frac{9}{2} & \frac{9}{2} \\ 0 & 0 & 0 & \frac{1}{8} & 1 & -\frac{11}{8} & \frac{7}{8} \\ \hline 0 & 0 & 0 & \frac{1}{8} & 0 & \frac{21}{8} & \frac{255}{8} \end{array}$$

Odgovarajuće optimalno bazno dopustivo rješenje i dalje nije cijelobrojno – niti jedna komponenta nije cijelobrojna.

Prema tome, dodajemo 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

$$\begin{array}{ccccccc|c} 1 & 0 & 0 & -\frac{1}{8} & 0 & \frac{11}{8} & 0 & \frac{33}{8} \\ 0 & 1 & 0 & \frac{1}{8} & 0 & -\frac{3}{8} & 0 & \frac{39}{8} \\ 0 & 0 & 1 & \frac{1}{2} & 0 & -\frac{9}{2} & 0 & \frac{9}{2} \\ 0 & 0 & 0 & \frac{1}{8} & 1 & -\frac{11}{8} & 0 & \frac{7}{8} \\ 0 & 0 & 0 & \frac{1}{8} & 0 & \frac{5}{8} & -1 & \frac{255}{8} \\ \hline 0 & 0 & 0 & \frac{1}{8} & 0 & \frac{21}{8} & 0 & \frac{255}{8} \end{array}$$

I ovdje nije očigledno koju varijablu treba dodati u bazne varijable (bazu), pa primijenimo dvofazni simpleks metod, odakle dobijamo

$$\begin{array}{ccccccc|c} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 4 \\ 0 & 0 & 1 & 0 & -\frac{7}{2} & 0 & \frac{1}{2} & 1 \\ 0 & 0 & 0 & 1 & \frac{5}{2} & 0 & -\frac{11}{2} & 7 \\ 0 & 0 & 0 & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 2 & 31 \end{array}$$

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 metod granjanja i odsijecanja (engl. branch and cut - B&C). Za razliku od B&B procedure koja rješava ILP, B&C daje i metod odsjecajućih ravni u rješavanju potproblema koji odgovaraju čvorovima (drveta). Dakle, ako uzimamo trenutno aktivni čvor i rješavamo njegov potproblem, dijelimo ga na nekoliko potproblema (podjelom dopustivog regiona na više disjunktnih – dodavanjem ograničenja) ili ga rješavamo rekurzivnim dodavanjem (gomorijevih) odsjecajućih ravni. Demonstrirajmo ovu ideju na primjeru rješavanja jednog ILP-a.

Primjer. Riješimo sljedeći ILP:

$$\begin{aligned} & \max 6x_1 + 5x_2 \\ & s.t. \\ & 3x_1 + x_2 \leq 11 \\ & -x_2 + 2x_2 \leq 5 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{aligned}$$

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 potproblema

$$\begin{aligned}
& \max 6x_1 + 5x_2 \\
s.t. \quad & \\
& 3x_1 + x_2 \leq 11 \\
& -x_2 + 2x_2 \leq 5 \quad (P1) \\
& x_1 \geq 3 \\
& x_2 \geq 0 \\
& x_1, x_2 \in \mathbb{Z}
\end{aligned}$$

$$\begin{aligned}
& \max 6x_1 + 5x_2 \\
s.t. \quad & \\
& 3x_1 + x_2 \leq 11 \\
& -x_2 + 2x_2 \leq 5 \quad (P2) \\
& x_1 \leq 2 \\
& x_2 \geq 0 \\
& x_1, x_2 \in \mathbb{Z}
\end{aligned}$$

Rješenje relaksacije problema $(P1)$ daje riješenje $x^* = (3, 2)$, koje je cijelobrojno, pa prema tome i dopustivo u početnom problemu. Njena vrijednost 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 cijelobrojno. Primjenjujući metod Gomorijevih odsjecajućih ravni na problem $(P2)$, konstruišemo odsjecajuću ravan $2x_1 + x_2 \leq 7$ koju dodajemo u problem $(P2)$ pa dobijamo problem

$$\begin{aligned}
& \max 6x_1 + 5x_2 \\
s.t. \quad & \\
& 3x_1 + x_2 \leq 11 \\
& -x_2 + 2x_2 \leq 5 \\
& 2x_1 + x_2 \leq 7 \quad (P3) \\
& x_1 \leq 2 \\
& x_2 \geq 0 \\
& x_1, x_2 \in \mathbb{Z}
\end{aligned}$$

Rješavajući LP relaksaciju problema $(P3)$, dobijamo rješenje $x^* = (\frac{9}{5}, \frac{17}{5})$, tj. optimalna vrijednost je 27.8. Kako je $LB \geq 28.8$, niti jedno cijelobrojno rješenje problema $(P3)$ ne može biti bolje od trenutno najbolje donje graniče ($LB = 28$), pa zaključujemo da je optimalno rješenje početnog problema $x^* = (3, 2)$.

7.5 Lagranžova relaksacija

U rješavanju problema optimizacije, nalazak optimalnog rješenja je primarni cilj. Međutim, za mnoge probleme nije realno očekivati da se garantuje nalazak optimalnog rješenja u realnom vremenu za realne instance problema. Umjesto toga, najbolje što se može učiniti je da se ponudi dobra donja granica za optimalnu vrijednost. Ako je ta granica blizu optimuma, u mnogo slučajeva ona može da zamijeni optimum, štедеći resurse kojima raspolažemo.

Ideja *Lagranžove relaksacije* se sastoji u tome da u modelu ILP-a koji ima ograničenja koja čine problem teškim (ova procjena je najčešće stvar praktičnog iskustva), posmatramo kao dio ciljne funkcije koju penalizujemo nekim parametrom. Ovo ograničenje potom brišemo iz skupa ograničenja modela, a zatim rješavamo novi problem. Na ovaj način, rješenje ovakvog problema je donja granica (za problem minimizacije) optimalnog rješenja inicijalnog problema. U praksi, ona je često jako blizu vrijednosti optimuma.

Primjer. 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 $t_{i,j}$. Prepostavimo da je čvor $i = 1$ startni čvor (izvorišni čvor), a čvor $n = |V|$ završni čvor (terminalni čvor). Zadatak je da se nađe put od izvorišnog do terminalnog čvora koji je najefтинiji ali tako da se čitav put pređe u vremenskom periodu T .

Ovaj problem se modeluje pomoću sljedećeg ILP-a:

$$z^* = \min \sum_{(i,j) \in E} c_{i,j} x_{i,j} \quad (7.4)$$

$$s.t. \quad (7.5)$$

$$\sum_i x_{i,j} - \sum_i x_{j,i} = \begin{cases} 1, & \text{ako } i = 1 \\ -1, & \text{ako } i = n \\ 0, & \text{inače} \end{cases} \quad (7.6)$$

$$\sum_{(i,j) \in E} t_{i,j} x_{i,j} \leq T \quad (7.7)$$

$$x_{i,j} \in \{0, 1\}, \forall (i, j) \in E. \quad (7.8)$$

Binarna varijabla $x_{i,j}$ dobija vrijednost 1 ako je odabrana grana (i, j) na putu, a 0 inače. Prvo ograničenje u modelu nam ograničava 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. Ovo ograničenje spada pod "komplikovanim" ograničenjima jer, kao što znamo, pronalazak najkraćeg puta u direktnim grafovima sa unaprijed

definisanim izvorišnim čvorom je polinomijalno rješiv (uz pomoć Dajkstri-nog algoritma), dok dodavanjem ograničenja o vremenu taj problem postaje NP-težak. Ideja je sada relaksirati ovaj problem uz prebacivanje ograničenja za vrijeme u funkciju cilja koja se zatim penalizuje. Dakle, Lagranžova relaksacija problema je data sa:

$$L(\lambda) = \min \sum_{(i,j) \in E} c_{i,j} x_{i,j} + \lambda \left(\sum_{(i,j) \in E} t_{i,j} x_{i,j} - T \right) = \min \sum_{(i,j) \in E} (c_{i,j} + t_{i,j}) x_{i,j} - \lambda T$$

s.t.

$$\sum_i x_{i,j} - \sum_i x_{j,i} = \begin{cases} 1, & \text{ako } i = 1 \\ -1, & \text{ako } i = n \\ 0, & \text{inače} \end{cases}$$

$$x_{i,j} \in \{0, 1\}, \forall (i, j) \in E.$$

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 multiplikatora*. Može se pokazati da vrijedi

$$L(\lambda) \leq L^* \leq z^*.$$

U praksi se pokazalo da je Lagranžova relaksacija dobra tehnika za dobijanje granica za optimalnu vrijednost problema.

Pokušajmo sada primijeniti ovu tehniku na *problem trgovačkog putnika*. Pogledajmo ovaj problem sa grafovske strane. Znamo da kolekcija grana formira rutu ako su svaka dva čvora grafa pokrivena sa dvije grane. Takođe, ako obrišemo grane incidentne sa čvorom 1, ostatak rute čini stablo koje pokriva graf $G \setminus \{1\}$. Definišimo sa $A(j)$ skup svih čvorova koji su incidentni sa čvorom j .

Relaksacija: definišimo sa T_1 skup svih ruta koje nakon brisanja grana koje pokrivaju čvor 1 formiraju drvo.

Shodno tome, konstruišemo sljedeći model

$$z^* = \min \sum_{(i,j) \in E} c_{i,j} x_{i,j}$$

$$\sum_{\{i \mid (i,j) \in E\}} x_{i,j} = 2, \forall i \in \{1, \dots, n\}$$

$$x \in T_1,$$

gdje $x_{i,j} = 1$ ako grana (i, j) pripada ruti, a 0 inače.

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 $c_{i,j}^\lambda = c_{i,j} + \lambda_i + \lambda_j$.

Rješenje problema Lagranžovog multiplikatora u praksi daje izvrsna rješenja i obično su rješenja blizu prave rute. Optimalno pokrivajuće stablo Lagranžovog problema $L(\lambda^*)$ za optimalni λ^* će nam često u praksi dati rješenje sa nekoliko visećih čvorova.

Konstruišimo još jednu Lagranžovu relaksaciju za ovaj problem uvodeći *ograničenja pokrivanja*, tj. da u svakoj ruti broj grana koje imaju oba kraja u skupu čvorova S kardinalnosti $|S|$ je najviše $|S| - 1$, pa imamo:

$$z^* = \min \sum_{(i,j) \in E} c_{i,j} x_{i,j} \\ \sum_{\{i|(i,j) \in E\}} x_{i,j} = 2, \forall i \in \{1, \dots, n\} \\ \sum_{(i,j) \in E \wedge i, j \in S} x_{i,j} \leq |S| - 1, \forall S \subseteq V \\ x_{i,j} \in \{0, 1\}, (i, j) \in E.$$

Lagranživa relaksacija prethodnog ILP-a ima oblik

$$L(\lambda) = \min \sum_{(i,j) \in E} c_{i,j}^\lambda x_{i,j} - 2 \sum_i \lambda_i \\ \sum_{(i,j) \in E} x_{i,j} \leq |S| - 1, \forall S \subseteq V \\ \sum_{(i,j) \in E} x_{i,j} = n$$

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

Često se u praksi nailazi na situaciju da je u LP-u (ili ILP-u) broj varijabli puno (eksponencijalno) veći u odnosu na broj ograničenja. U osnovi, riječ je o ekstremno velikim modelima sa matricom A velikih dimenzija. Kada je riječ o rješavanju ovakvih problema, veliki dio matrice (tj. njene kolone) nam obično nikada i neće biti relevantan u rješavanju. Da bi se riješila specifična instanca problema, ono što nam je potrebno su:

-
- Ograničenja koja se vezuju uz optimalnost.
 - Varijable koje su bazne u optimalnom rješenju.

Dakle, ono što je potrebno su ograničenja čije su odgovarajuće dualne vrijednosti pozitivne, kao i varijable koje imaju pozitivne vrijednosti koeficijenata doprinosa za optimalno rješenje (tj. vektor \bar{c} iz simpleks tabele). Ako bismo imali ove varijable i ograničenja na raspolaganju od samog početka, problem bi bio riješen veoma brzo. Međutim, ova informacija nam nije besplatna. 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.

- Krenimo sa “obećavajućim” podskupom S kolona matrice A ;
- Riješimo restrikovani LP na odabrnem podskupu kolona S ;
- Procijenimo ostale kolone i dodajmo u S one sa negativnim cijenama doprinosa;
- Itrerativno pozivamo perthodna dva koraka dok god postoje kolone sa negativnim doprinosom koje nisu u S .

Tehnički zapisano, krećemo od *restrikovanog master problema* (RMP)

$$\begin{aligned} & \min c^T x \\ \text{s.t. } & \sum_{i \in I} A_i x_i \leq b \\ & x \geq 0. \end{aligned} \tag{RMP}$$

gdje I inicijalizujemo kao skup kolona koje čine inicijalno BDR za RMP (i po mogućnosti neke druge kolone za koje smatramo da su “bitne” u kontekstu problema). Riješimo RMP, te izračunamo odgovarajuće optimalno dualno rješenje. Sada je potrebno naći varijablu 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. potproblema generisan 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 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.

Postoji mnogo pristupa koji se tiču ažuriranja skup S tokom iteracija. Najjednostavniji se tiče zadržavanja svake kolone koja je do sada izgenerisana u skup S i dodavanja nove u postojeći skup. Naprednija strategija se tiče izbacivanja onih kolona iz skupa S koje su nebazne u svakoj iteraciji.

Bazna column generation šema. U osnovi, šema se sastoji od dva koraka:

1. Za ulazne kolone prvo izaberemo one sa najvećim negativnim koeficijentom doprinosa među varijablama koje odgovaraju skupu I .
2. Jednom kada su svi koeficijenti nenegativni (tj. kada smo našli optimalnu vrijednost za RMP), tražimo nove varijable (kolone) van skupa I (rješavajući CGSP) koje će biti ubačene u I .

Ako u koraku 1 za skup I uzmemos 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 efikasna primjena CG frejmворка je teško ostvariva.

Iterativni proces rješavanja RMP problema se može prekinuti i ranije (bez dostizanja optimalnog rješenja početnog problema), uslijed velikog broja iteracija, iako je moguće 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. Obruto ne vrijedi: ako bi iskoristili neku heuristiku za rješavanje CGSP, onda se optimalnost početnog LP-a ne može garantovati. Razlog je vrlo jednostavan, heuristika treba da nađe bilo koju kolonu sa negativnim koeficijentom doprinosa koja će biti ubačena u RMP, čine se proces (uspješno) nastavlja. Ali, ako heuristika ne uspije u nalasku takve kolone, to ne znači da takva kolona ne postoji.

U nastavku dajemo primjenu *metode generisanja kolona* na poznatom *Cutting stock* problemu.

Problem. Tvornica papira proizvodi rolne papira fiksne širine W . Kupci naručuju različit broj rolni različite širine $w \in \mathbb{R}^m$, dok je sa vektorom $b \in \mathbb{R}^m$ dat zahtjev (broj naručenih 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. Što se tiče kolona za ovom problemu, one predstavljaju dopustive obrasce (kombinacije rezanja). Jedan dio obrasca sa slike bi bio 1820 x 2 (rolne) u prvoj koloni. Primijetimo da broj različitih obrazaca raste eksponencijalno u odnosu na broj narudžbi. Ako bismo izvršili enumeraciju svih mogućih obrazaca, trebali bi riješiti sljedeće ILP modele:

$$\begin{aligned} \min & \sum_i x_i \\ \sum_{ij} a_{i,j} x_j & \geq b_i, \forall i \\ x_j & \geq 0, x_j \in \mathbb{Z} \end{aligned} \quad (CSILP)$$

gdje je $a_{i,j}$ broj puta da se narudžba i pojavi u obrascu j (vrsta na slici), dok je x_j broj puta koliko je obrazac j korišten. Primjera radi, na Slici 7.4 je $a_{1,1} = 3$ i $x_1 = 2$, dok je $a_{1,2} = 0$; $a_{1,5} = 1$ i $x_5 = 12$.

Ideja je ne stvarati obrasce na početku, već ih generisati prema potrebi (po kolonama) koristeći *metod generisanja kolona*. Kolona a odgovara dopustivom obrascu akko vrijedi

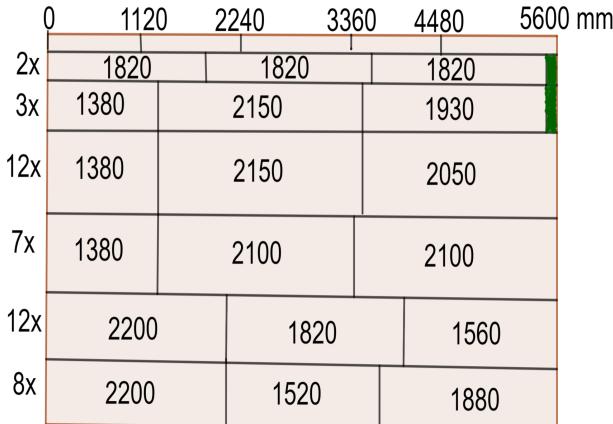
$$\sum_{i=1}^m a_i w_i \leq W,$$

gdje a sadrži samo nenegativne (cjelobrojne) vrijednosti. Primijetimo da je na osnovu prethodnog (razmatrajući relaksaciju problema *CSILP*), vrijednosti koeficijenata doprimosa jednaki $\bar{c} = 1 - \sum_i a_{j,i} y_i$. Dakle, CG potproblem izgleda ovako:

$$\begin{aligned} \max & \sum_{i=1}^m y_i a_i \\ \sum_i w_i a_i & \leq W \\ a_i & \geq 0 \\ a_i & \in \mathbb{Z}, \end{aligned} \quad (CGP)$$

gdje su y_i vrijednosti optimuma duala. Ovaj problem je poznat pod nazivom *cjelobrojni problem ruksaka* (eng. integer knapsack problem). Iako je NP-težak, on se u praksi rješava na vrlo efikasan način 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 formira inicijalnu dopustivu bazu. Svakako, malo je vjerovatno da će ove kolone biti korištene u optimalnom rješenju. Prema tome, algoritam za ovaj problem bi išao ovako:



Slika 7.4: Rješenje jedne instance.

1. Konstruišimo inicijalno bazno dopustivo rješenje (jedinični vektori) i kolone ubacimo u skup S .
2. Riješimo relaksaciju restrikovanog problema ($CSILP$) i vratimo dualno optimalno rješenje.
3. Riješimo (CGP).
4. Ako je vrijednost optimalnog rješenja negativna, ubaciti novu kolonu a (koja je rješenje potproblema) 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. Riješimo sada jednu instancu ovog problema pomocu CG frejm-vorka. Recimo da imamo $W = 15$, te tri narudžbe za sječenje:

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

$$\text{waste}_1 = 3, \text{waste}_2 = 3, \text{waste}_3 = 1, \text{waste}_4 = 0, \text{waste}_5 = 1, \text{waste}_6 = 2,$$

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 varijable 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 bi problem gledali sa jednostavnije strane, izbacimo uslove integralnosti (cjelobrojnosti promjenjivih). Tada ovaj problem možemo da riješimo uz pomoć simpleks metoda gdje je $B^{-1} = \begin{pmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{pmatrix}$ a završna simpleks tabela ima oblik

x_1	$\frac{1}{3}$	0	0	$\left \frac{80}{3} \right.$
x_2	0	$\frac{1}{2}$	0	25
x_3	0	0	$\frac{1}{2}$	50
z	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{2}$	$\left \frac{305}{3} \right.$

Sljedeći korak je naći kolonu čija 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 varijablama. 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 varijablu (i odgovarajuću kolonu) sa minimalnim negativnim doprinosom koju potom ubacujemo u RMP. Kako je svaki paterni dat u opštem slučaju kao $A_j = \begin{pmatrix} a_4 \\ a_6 \\ a_7 \end{pmatrix}$, CG potproblem 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 $(a_4^*, a_6^*, a_7^*)^T$ ulazi kao nova kolona u RMP. 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:

$$\begin{aligned} & \max \frac{1}{3} a_4 + \frac{1}{2} a_6 + \frac{1}{2} a_7 \\ & 4a_4 + 6a_6 + 7a_7 \leq 15 \\ & a_4, a_6, a_7 \geq 0, a_4, a_6, a_7 \in \mathbb{Z}. \end{aligned}$$

Uz pomoć algoritma dinamičkog programiranja riješimo ovaj problem, pa dobijamo rješenje $(a_4^*, a_6^*, a_7^*)^T = (2, 0, 1)^T = A_4$ optimalne vrijednosti $\frac{7}{6}$.

Prekonvertujmo 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 vektor

$$B^{-1} A_4 = \begin{pmatrix} \frac{2}{3} \\ 0 \\ \frac{1}{3} \end{pmatrix}$$

ulazi u novu simpleks tabelu (uz redukovanoj 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 novododanoj koloni tražimo pivot element (kao i u simpleks metodi). Lako je vidjeti da je pivot element $\frac{2}{3}$, pa radeći 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 potproblem 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 potproblem:

$$\frac{1}{4}a_4 + \frac{1}{2}a_6 + \frac{1}{2}a_7$$

uz uslove nenegativnosti i kapaciteta i integralnosti (cjelobrojnosti). Rješavajući problem uz pomoć dinamičkog problema, dobijamo optimalno rješenje $A_5 = (2, 1, 0)^T$, dok je vrijednost optimuma za CG potproblem jednak 1, odakle slijedi da je optimum početnog problema nađen sa prethodnim rješenjem (ne postoji više ulaznih kolona koje se dodaju u RMP), i ono je jednako 95, dok je rješenje dato sa $x_1^* = 0, x_2^* = 25, x_3^* = 30, x_4^* = 40$. Dakle, koristimo samo 3 različita paterna za sječenje.

7.7 Zadaci

- Pomoću metode grananja i ograničavanja (B&B) riješiti sljedeći problem ILP-a:

$$\begin{aligned} & \max 100x_1 + 150x_2 \\ & \text{s.t.} \\ & 8000x_1 + 4000x_2 \leq 40000 \\ & 15x_1 + 30x_2 \leq 200 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

-
2. Pomoću metode grananja i ograničavanja (B&B) riješiti sljedeći problem ILP-a:

$$\begin{aligned} & \max 5x_1 + 4x_2 \\ & s.t. \\ & 3x_1 + 4x_2 \leq 10 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

Nakon primjene B&B metode, riješiti metod grafički dijeljeći region na podregione i uoprediti dobijena rješenja.

3. Pomoću metode grananja i ograničavanja (B&B) riješiti sljedeći problem ILP-a:

$$\begin{aligned} & \max 5x_1 + 6x_2 + 4x_3 \\ & s.t. \\ & 5x_1 + 3x_2 + 6x_3 \leq 20 \\ & x_1 + 3x_2 \leq 12 \\ & x_1, x_2, x_3 \geq 0 \\ & x_2 \in \mathbb{Z}. \end{aligned}$$

4. Pomoću metode implicitne enumeracije riješiti sljedeći ILP:

$$\begin{aligned} & \max 1000x_1 + 700x_2 + 800x_3 \\ & s.t. \\ & 5000x_1 + 6000x_2 + 4000x_3 \leq 10000 \\ & x_1, x_2, x_3 \in \{0, 1\}. \end{aligned}$$

5. Pomoću metode implicitne enumeracije riješiti sljedeći ILP:

$$\begin{aligned} & \max 20x_1 + 30x_2 + 10x_3 + 40x_4 \\ & s.t. \\ & 2x_1 + 4x_2 + 3x_3 + 7x_4 \leq 10 \\ & 10x_1 + 7x_2 + 20x_3 + 15x_4 \leq 40 \\ & x_1 + 10x_2 + x_3 \leq 10 \\ & x_1, x_2, x_3, x_4 \in \{0, 1\}. \end{aligned}$$

6. Metodom gomorijevih odsjecajućih ravnih, riješiti sljedeći ILP:

$$\begin{aligned} & \min -x_1 - x_2 \\ & s.t. \\ & 2x_1 + 5x_2 \leq 20 \\ & 4x_1 + 3x_2 \leq 17 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

7. Metodom gomorijevih odsjecajućih ravnih, riješiti sljedeći ILP:

$$\begin{aligned} & \max x_1 + 4x_2 \\ & s.t. \\ & 2x_1 + 4x_2 \leq 7 \\ & 5x_1 + 3x_2 \leq 15 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

8. Uz pomoć algoritma B&C riješiti sljedeći problem ILP-a:

$$\begin{aligned} & \max x_1 + 2x_2 \\ & s.t. \\ & -2x_1 + 2x_2 \leq 5 \\ & 6x_1 + 4x_2 \leq 25 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

9. Za sljedeći problem

$$\begin{aligned} & \min c^T x \\ & s.t. \\ & Ax = b \\ & Dx \geq q \\ & x \geq 0 \end{aligned}$$

napisati odgovarajuću lagranžovu relaksaciju.

10. Neka je dad problem ILP-a:

$$\begin{aligned} & \max x_2 \\ & -2x_1 + 2x_2 \leq 1 \\ & 2x_1 + 2x_2 \leq 7 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

Definisati Lagranžovu relaksaciju za ovaj problem (npr. uzeti prvo ograničenje za “teško”). Riješiti početni problem sa jednom od tehnika za rješavanje ILP-a. Potom, riješiti Lagranžovu relaksaciju problema za $\lambda = 0.25$. Šta možemo zaključiti na osnovu rješenja Lagranžove relaksacije i pročetnog problema?

11. Pomoću Column generation okvira, 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

Dekompozicioni Metodi

Dekompozicione metode u matematičkom programiranju su tehnike pomoću kojih se polazni problem (na primjer, problem linearog programiranja) 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 i za primjenu i razvoj dekompozicionih metoda. Čak se i tradicionalne metode za rješavanje problema cjelobrojnog programiranja, kao što su B&B ili odsjecačuće 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 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 dekompozicije 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 zove i master problem) se rješava nad prvim skupom promjenljivih, dok se vrijednosti promjenljivih iz drugog skupa, za dato rješenje prvog potproblema, određuju rješavanjem drugog potproblema. Uspjehnu primjenu Benderove dekompozicije, između ostalog, obezbjeđuje i teorija dualnosti, odnosno teoreme slabe i jake dualnosti.

Posmatrajmo sada sljedeću formulaciju MILP programa

$$\begin{aligned} & \min f^T x + g^T y \\ & \text{s.t. } Ax + By \geq c \\ & \quad x \geq 0, y \in \mathbb{Z} \end{aligned} \tag{8.1}$$

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 odgovaraajućih dimenzija.

Pretpostavimo da raspolazemo jednom početnom vrijednošću vektora y . Za tu vrijednost (označimo je sa \bar{y}) dobijamo sljedeći potproblem

$$\begin{aligned} & \min f^T x + g^T \bar{y} \\ & \text{s.t. } Ax \geq c - B\bar{y} \\ & \quad x \geq 0 \end{aligned} \tag{8.2}$$

Posmatrajmo dualni problem potproblema (8.2), ignorišući term $g^T \bar{y}$ koji je konstantan.

$$\begin{aligned} & \max u^T(c - B\bar{y}) \\ & \text{s.t. } u^T A \leq f \\ & \quad u \geq 0 \end{aligned} \tag{8.3}$$

Sada uključujemo teoriju dualnosti i razmatramo sljedeće mogućnosti, koje mogu da se jave.

1. Ako je optimalno rješenje problema (8.2) z^* , a optimalno rješenje duala (8.3) u^* , na osnovu teoreme o jakoj dualnosti imamo

$$z^* - g^T \bar{y} = (u^*)^T(c - B\bar{y}) \tag{8.4}$$

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 LP dualnosti dobijamo

$$z \geq g^T y + (u^*)^T(c - B\bar{y}) \tag{8.5}$$

Posljednja nejednakost propisuje 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 $y = \bar{y}$, funkcija cilja bi bila veća ili jednaka vrijednosti \bar{z} , koja se dobije u potproblemu, 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 da je dualni potproblem neograničen. U toj situaciji važi nejednakost

$$(u^*)^T(c - By) > 0 \quad (8.6)$$

gdje je $(u^*)^T$ zrak mogućih rješenja duž kog funkcija neograničeno raste (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.

$$\begin{aligned} & \min z \\ & s.t. \\ & z \geq g^T y + (u^k)^T(c - By), \quad k \in K \\ & (u^k)^T(c - By) \leq, \quad k \in L \\ & y \in \mathbb{Z}, \end{aligned} \quad (8.8)$$

gdje je K skup ekstremnih tačaka, a L skup ekstremnih zrakova od potproblema. 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 potproblem (8.2) restrikcija originalnog problema (8.1), pa je njegova optimalna vrijednost z^* jedna gornja granica za funkciju cilja polaznog problema. Takođe je master problem (8.8) relaksacija polaznog problema, jer se pri rješavanju master problema ne posmatraju sva ograničenja. Stoga je optimalna vrijednost master problema donja granica za funkciju cilja originalnog problema.

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 potproblem)

$$\begin{aligned} & \max u^T(c - B\bar{y}) \\ & \text{s.t. } u^T A \leq f \\ & \quad u \geq 0 \end{aligned}$$

- 5: **if** Potproblem nema dopustivo rješenje **then**
- 6: KRAJ // nema rješenja
- 7: **end if**
- 8: **if** Potproblem 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** Potproblem 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

$$\begin{aligned} & \min z \\ & \text{s.t. } g^T y + (u^k)^T(c - By), k \in K \\ & \quad (u^k)^T(c - By), k \in L \\ & \quad y \in \mathbb{Z} \end{aligned}$$

- 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 preformulisati tako da se on može primjeniti 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 potproblema dobijati donje granice.

Primjer. Pretpostavimo da raspolaćemo 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.

S obzirom na postavku zadatka, jasno je da bi optimalno rješenje trebalo da podrazumijeva da se 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 kao

$$\begin{aligned}
& \max && 1.01x_1 + 1.02x_2 + \dots + 1.10x_{10} + 1.045y \\
& \text{s.t.} && x_1 + x_2 + \dots + x_{10} + y \leq 1000 \\
& && x_1 \leq 100 \\
& && \vdots \\
& && x_{10} \leq 100 \\
& && y \in \mathbb{Z} \\
& && x_1, x_2, \dots, x_{10}, y \geq 0
\end{aligned} \tag{8.9}$$

Zapišimo sada model u obliku koji odgovara postavci metoda Benderove dekompozicije.

$$\begin{aligned}
& \max && f^T x + g^T y \\
& \text{s.t.} && Ax + By \leq c \\
& && x \geq 0, y \in \mathbb{Z}^+
\end{aligned} \tag{8.10}$$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{10} \end{pmatrix}, f = \begin{pmatrix} 1.01 \\ 1.02 \\ \vdots \\ 1.10 \end{pmatrix}$$

Primijetimo da je su u našem primjeru g i y jednodimenzioni vektori, koje ćemo u daljem zapisu identifikovati sa njihovom (jedinom) koordinatom, odnosno $y = (y)$ i $g = (1.045)$. Takođe je i

$$A = \begin{pmatrix} 1 & \dots & 1 \\ 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{pmatrix}, B = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, c = \begin{pmatrix} 1000 \\ 100 \\ \vdots \\ 100 \end{pmatrix}$$

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 $\bar{y} = 1500$. 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 potproblema). 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

$$\begin{aligned} u_1 + u_2 &\geq 1.01 \\ u_1 + u_3 &\geq 1.02 \\ &\vdots \\ u_1 + u_{11} &\geq 1.1 \\ u_1 &\geq 1.045 \\ u &\geq 0 \end{aligned}$$

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

$$\begin{aligned} &\max z \\ 1000 - y &\geq 0 \\ y &\in \mathbb{Z}^+ \end{aligned}$$

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

$$\begin{aligned} z &\leq 1.045y + \left(\begin{pmatrix} 1.1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \right)^T \left(\begin{pmatrix} 1000 \\ 100 \\ \vdots \\ 100 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} y \right) \\ z &\leq 1.045y + \left(\begin{pmatrix} 1.1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \right)^T \begin{pmatrix} 1000 - y \\ 100 \\ \vdots \\ 100 \end{pmatrix} \end{aligned}$$

Odnosno

$$z \leq 1100 - 0.055y$$

Sada novodobijenu odsjecajuću ravan pridodajemo ograničenjima master problema i rješavamo master problem

$$\begin{aligned} \max z \\ 1000 - y \geqslant 0 \\ z \leqslant 1100 - 0.055y \\ y \in \mathbb{Z}^+ \end{aligned}$$

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

$$u^T(c - B\bar{y}) = (u_1 \dots u_{11}) \cdot \left(\begin{pmatrix} 1000 \\ 100 \\ \vdots \\ 100 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \right) = 1000u_1 + 100u_2 + \dots + 100u_{11}$$

koja, s obzirom na ograničenja u dualnom problemu, 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

$$z \leqslant 1.045y + \begin{pmatrix} 0 \\ 1.01 \\ \vdots \\ 1.1 \end{pmatrix}^T \left(\begin{pmatrix} 1000 \\ 100 \\ \vdots \\ 100 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} y \right)$$

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, pomocu nekog od računarskih programa) dobija se optimalno rješenje za cijelobrojnu 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 uvek 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

$$LB = \max\{1055, 1.045 * 41 + 1013.59\} = \max\{1055, 1056.435\} = 1056.435$$

Dalje, dodajemo novu odsjecajuću ravan

$$z \leq 1.045y + \begin{pmatrix} 1.01 \\ 0 \\ 0.01 \\ 0.02 \\ 0.09 \end{pmatrix}^T \begin{pmatrix} 1000 \\ 100 \\ \vdots \\ 100 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} y$$

Odnosno

$$z \leq 1.045y + 1010 - 1.01y + 100 \cdot 0.45$$

t.j.

$$z \leq 0.035y + 1055$$

Ovo ograničenje pridodajemo 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 cijelobrojnu 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 pridodamo master problemu, dobijamo novi master problem

$$\begin{aligned} & \max z \\ & 1000 - y \geq 0 \\ & z \leq 1100 - 0.055y \\ & z \leq 1.045y + 1055 \\ & z \leq 0.035y + 1055 \\ & z \leq -0.005y + 1065 \\ & y \in \mathbb{Z}^+ \end{aligned}$$

čije se optimalno rješenje dobija za vrijednost za cijelobrojnu vrijednost $\bar{y} = 250$, koje iznosi $\bar{z} = 1063.75$.

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 uvijek 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 pridodajemo je master problemu, koji izgleda

$$\begin{aligned} \max z \\ 1000 - y &\geq 0 \\ z &\leq 1100 - 0.055y \\ z &\leq 1.045y + 1055 \\ z &\leq 0.035y + 1055 \\ z &\leq -0.005y + 1065 \\ z &\leq 0.015y + 1058 \\ y &\in \mathbb{Z}^+ \end{aligned}$$

Rješavanjem master problema dobijamo novu vrijednost $\bar{y} = 350$, a gornja granica se ažurira na

$$UB = 1063.25$$

Ponavljajući 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

$$\begin{aligned}
& \max z \\
& 1000 - y \geq 0 \\
& z \leq 1100 - 0.055y \\
& z \leq 1.045y + 1055 \\
& z \leq 0.035y + 1055 \\
& z \leq -0.005y + 1065 \\
& z \leq 0.015y + 1058 \\
& z \leq 0.005y + 1061 \\
& y \in \mathbb{Z}^+
\end{aligned}$$

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.

Sada 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

$$\begin{aligned}
& \max f^T x \\
& s.t. Ax \leq c - B\bar{y} \\
& x \geq 0
\end{aligned} \tag{8.11}$$

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 linearog programiranja – *Dantzig–Wolfe* (DW) dekompoziciju. Ova tehnika je originalno dizajnirana za rješavanje problema

linearnog programiranja velikih dimenzija. Osnovna ideja DW dekompozicije je da se na podskup ograničenja polaznog problema primijeni *Teorema Minkovskog*, odnosno *Teorema o reprezentaciji za konveksne poliedre*, te da se polazni problem razbije na više manjih problema.

Teorema 18 Neka je $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ ograničeni polieder i neka je $\{x^{(1)}, \dots, x^{(p)}\}$ skup ekstremnih tačaka od 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

$$\begin{aligned} & \min c^T x \\ & s.t. Ax = b \\ & x \in X \end{aligned} \tag{8.12}$$

gdje je skup X polieder. Iako za DW dekompoziciju nije neophodno, u daljem razmatranju, radi lakše analize DW dekompozicije, smatraćemo da je X ograničen polieder, odnosno politop.

Dalje, navećemo 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, koje ćemo zbog toga i zvati "komplikovanim ograničenjima".

Posmatrajmo sada konfiguraciju za primjenu Teoreme o reprezentaciji na skup X .

Neka su tačke $x^{(1)}, x^{(2)}, \dots, x^{(t)}$ vrhovi politopa X . Prema Teoremi o reprezentaciji, 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 postoje $\lambda_1, \lambda_2, \dots, \lambda_t$, takvi da važi:

$$\begin{aligned} x &= \sum_{j=1}^t \lambda_j x^{(j)} \\ \sum_{j=1}^t \lambda_j &= 1 \\ \lambda_j &\geq 0, j = 1, \dots, t \end{aligned} \tag{8.13}$$

Ponovimo da Teorema o reprezentaciji ima svoju formulaciju i u slučaju da polieder 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

$$\begin{aligned} & \min c^T \cdot \sum_{j=1}^t \lambda_j x^{(j)} \\ & \text{s.t. } A \cdot \sum_{j=1}^t \lambda_j x^{(j)} = b \\ & \quad \sum_{j=1}^t \lambda_j = 1 \\ & \quad \lambda_j \geq 0, \quad j = 1, \dots, t \end{aligned} \tag{8.14}$$

odnosno

$$\begin{aligned} & \min \sum_{j=1}^t (c^T x^{(j)}) \lambda_j \\ & \text{s.t. } \sum_{j=1}^t (A x^{(j)}) \lambda_j = b \\ & \quad \sum_{j=1}^t \lambda_j = 1 \\ & \quad \lambda_j \geq 0, \quad j = 1, \dots, t \end{aligned} \tag{8.15}$$

Vidimo da se početni problem (8.12) sada može predstaviti ekvivalentnom reprezentacijom (8.15), u kojem se, umjesto polaznih promjenljivih (vektora x), sada posmatraju promjenljive λ_j , $j = 1, \dots, t$ (skup ovih promjenljivih ćemo zvati λ promjenljive), uz pretpostavku da su vrhovi $x^{(1)}, x^{(2)}, \dots, x^{(t)}$ konveksnog politopa X poznati. 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) jednako optimalnom rješenju reformulisanog problema (8.15).

Samom reformulacijom polaznog problema (8.12) u DW formulaciju (8.15) nismo postigli mnogo, 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, primjeničemo tehniku "generisanja kolona" (eng. *column generation*), kako bismo u razmatranje

uključili nove vrhove politopa X , koje 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.

$$\begin{aligned} \min \quad & \sum_{j \in I} (c^T x_j) \lambda_j \\ \text{s.t.} \quad & \sum_{j \in I} (Ax_j) \lambda_j = b \\ & \sum_{j \in I} \lambda_j = 1 \\ & \lambda_j \geq 0, \quad j \in I \end{aligned} \tag{8.16}$$

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 doprinosa svih nebaznih promjenjivih nenegativni, tj. postoji li neka nebazna varijabla koja može doprinijeti dodatnom smanjenju vrijednosti funkcije cilja. Kao što znamo iz simpleks tablele i ranije sekcije o dualnosti linearнog programiranja, 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 promjenjive koja se odnosi na ograničenje $\sum_{j \in I} \lambda_j = 1$.

Da bi se odredili koeficijenti doprinosa 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

$$\begin{pmatrix} c^T x^{(j)} \\ Ax^{(j)} \\ 1 \end{pmatrix}$$

i sada rješavamo *Column generation potproblem* (CG), čiji je cilj da se pronađe najmanji koeficijent doprinosa. Ukoliko je on manji od nule, dobija se nova kolona koja treba da se doda u potproblem RMP. U slučaju da je on nenegativan, tada je optimalno rješenje RMP ujedno i optimalno rješenje MP.

CG potproblem 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 potproblem sada kao problem maksimizacije, gdje ćemo za funkciju cilja uzeti negativnu vrijednost funkcije cilja minimizacije, tj.

$$\max_{j=1,\dots,t} \{y^T A x - c^T x + \alpha\} \quad (8.18)$$

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

$$z^* \geq \bar{z} - \hat{z}$$

odnosno, razlika $\bar{z} - \hat{z}$ je donja granica za optimalnu vrijednost funkcije cilja polaznog problema. Dalje, prisjetimo se da je \bar{z} gornja granica za z^* , odnosno važi

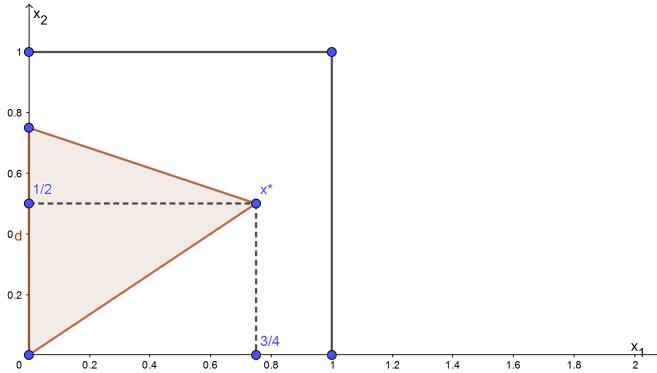
$$\bar{z} \geq z^* \geq \bar{z} - \hat{z} \quad (8.19)$$

Iraz (8.19) se može koristiti za određivanje koliko daleko 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.

Posmatrajmo problem linearног programiranja



Slika 8.1: Grafička interpretacija problema. Optimalno rješenje je označeno sa x^* .

$$\begin{aligned}
 & \max x_1 \\
 & \text{s.t. } x_1 + 3x_2 \leq \frac{9}{4} \\
 & \quad 2x_1 - 3x_2 \leq 0 \\
 & \quad x \in X = \{(x_{1,2}) : 0 \leq x_{1,2} \leq 1\}.
 \end{aligned} \tag{8.20}$$

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 trougao, a da se maksimalna vrijednost funkcije cilja dostiže u vrhu $(\frac{3}{4}, \frac{1}{2})$ tog trougla, odnosno,

$$z^* = \frac{3}{4} \quad \text{za } x^* = \begin{pmatrix} \frac{3}{4} \\ \frac{1}{2} \end{pmatrix} \tag{8.21}$$

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

$$\begin{aligned} & \max c^T x \\ & \text{s.t. } Ax \leq b \\ & \quad x \in X \end{aligned} \tag{8.22}$$

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

$$\begin{aligned} & \max \sum_{j=1}^4 (c^T x^{(j)}) \lambda_j \\ & \text{s.t. } \sum_{j=1}^4 (Ax^{(j)}) \lambda_j \leq b \\ & \quad \sum_{j=1}^4 \lambda_j = 1 \\ & \quad \lambda_j \geq 0, \quad j = 1, \dots, 4 \end{aligned} \tag{8.23}$$

Odnosno, nakon sređivanja

$$\begin{aligned} & \max \lambda_2 + \lambda_4 \\ & \text{s.t. } \begin{pmatrix} 1 \\ 2 \end{pmatrix} \lambda_2 + \begin{pmatrix} 3 \\ -3 \end{pmatrix} \lambda_3 + \begin{pmatrix} 4 \\ -1 \end{pmatrix} \lambda_4 \leq \begin{pmatrix} \frac{9}{4} \\ 0 \end{pmatrix} \\ & \quad \sum_{j=1}^4 \lambda_j = 1 \\ & \quad \lambda_j \geq 0, \quad j = 1, \dots, 4 \end{aligned} \tag{8.24}$$

Da bismo u ograničenjima izbjegli znak \leq u prva dva ograničenja, dodajmo modelu dvije izjednačavajuće promjenljive s_1 i s_2 , te model zapišimo

\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

kao

$$\begin{aligned}
 & \max \lambda_2 + \lambda_4 \\
 & s.t. \quad \lambda_2 + 3\lambda_3 + 4\lambda_4 + s_1 = \frac{9}{4} \\
 & \quad 2\lambda_2 - 3\lambda_3 - \lambda_4 + s_2 = 0 \\
 & \quad \sum_{j=1}^4 \lambda_j = 1 \\
 & \quad \lambda_j \geq 0, \quad j = 1, \dots, 4 \\
 & \quad s_1, s_2 \geq 0
 \end{aligned} \tag{8.25}$$

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 potproblemu, RMP izgleda ovako

$$\begin{aligned}
 & \max 0 \\
 & s.t. \quad s_1 = \frac{9}{4} \\
 & \quad s_2 = 0 \\
 & \quad \lambda_1 = 1 \\
 & \quad \lambda_1, s_1, s_2 \geq 0
 \end{aligned} \tag{8.26}$$

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

$$\max_{x \in X} (1, 0)^T x + 0 - (0, 0) \begin{pmatrix} 1 & 3 \\ 2 & -3 \end{pmatrix} x = \max_{x \in X} x_1. \quad (8.27)$$

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 potproblem postoji, te je treba da odaberemo. 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 varijable 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

Primjetimo da je cijena doprinosa dodate kolone promijenjena sa 1 na -1 (zbog korištenja tabelarne forme simpleks metoda, optimalna vrijednost problema je $-\hat{z}$, jer se CG potproblem u stvari 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, pa pivotirajmo oko njega. Prvo sve elemente u koloni 2 podijelimo sa 2, te potom izvršimo elementarne transformacije u kojima načinimo nule u ostalim elementima novododatake kolone. Varijabla s_2 izlazi iz baze, dok λ_2 ulazi 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 varijable su sada iz skupa $\{s_1, \lambda_1, \lambda_2\}$, koja 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 varijabli iz koeficijenta doprinosa, tj. $y_1 = 0, y_2 = \frac{1}{2}, \alpha = 0$.

Iteracija 2. Iz prethodnog dobijamo CG potproblem

$$\max_{x \in X} (1, 0)^T x - \left(0, \frac{1}{2}\right) \begin{pmatrix} 1 & 3 \\ 2 & -3 \end{pmatrix} x - 0 = \max_{x \in X} x_1 - x_1 + \frac{3}{2}x_2 = \max_{x \in X} \frac{3}{2}x_2. \quad (8.29)$$

Rješenje ovog problema je lako dobiti, 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 varijable λ_3 . Dakle, prethodna kolona je data u odnosu na varijablu λ_3 početnog problema, ali je sada izražavamo preko trenutne baze (potproblema) 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 je ulazna koja se dodaje u simpleks tabelu:

$$\begin{pmatrix} -\frac{3}{2} \\ \frac{9}{2} \\ \frac{1}{2} \\ -\frac{3}{2} \\ \frac{5}{2} \end{pmatrix}$$

Dakle, dobijamo tabelu

	s_1	s_2	λ_2	λ_1	
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
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 varijabla λ_3 . Kako je $\frac{9/4}{9/2} = \frac{1}{2} > \frac{1}{5/2} = \frac{2}{5}$, imamo da je $\frac{5}{2}$ pivot element. Sada pivotiramo oko tog elementa na način 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 tabelu 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 potproblem 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 potproblema 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 potproblem. 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 varijable λ_4 . Dakle, prethodna kolona je data u odnosu na varijablu λ_4 početnog problema, ali je sada izražavamo preko trenutne baze (potproblema) vektora s_1, λ_2 i λ_3 . Dakle, nakon množenja kolone sa inverzom baze $B^{-1} = \begin{pmatrix} 1 & \frac{2}{5} & -\frac{9}{5} \\ 0 & -\frac{1}{5} & \frac{3}{5} \\ 0 & -\frac{1}{5} & \frac{3}{5} \end{pmatrix}$, koraci opisani u Tabeli 8.1, sljedeća kolona je ulazna koja se dodaje u simpleks tabelu:

$$\begin{pmatrix} -\frac{3}{5} \\ \frac{9}{5} \\ \frac{2}{5} \\ \frac{3}{5} \\ \frac{3}{5} \end{pmatrix}$$

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 $\frac{9/20}{9/5} < \frac{2/5}{3/5} < \frac{3/5}{2/5}$, slijedi da je pivot element $\frac{9}{5}$. Sada pivotiramo oko tog elementa na način 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 varijabli: $y_1 = \frac{1}{3} = y_2$, dok je $\lambda = 0$.

Iteracija 4. Na osnovu vrijednosti dualnih varijabli, formira se novi CG potproblem:

$$\max_{x \in X} (1, 0)^T x - \left(\frac{1}{3}, \frac{1}{3}\right) \begin{pmatrix} 1 & 3 \\ 2 & -3 \end{pmatrix} x - 0 = \max_{x \in X} (x_1 - x_1) = 0$$

Iz ovoga zaključujemo da je $\hat{z} = 0$, pa iz (8.19) slijedi da smo dobili optimum početnog problema i optimalna tačka je 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

- Uz pomoć Benderove dekompozicije riješiti sljedeći MILP problem:

$$\begin{aligned} &\min x + y \\ &s.t. \\ &2x + y \geq 3 \\ &x \geq 0 \\ &y \in \{-5, -4, \dots, 3, 4\}. \end{aligned}$$

2. Uz pomoć Benderove dekompozicije riješiti sljedeći MILP problem:

$$\begin{aligned} & \max 8y_1 + 9y_2 + 5y_3 + 6y_4 - 15x_1 - 10x_2 \\ & s.t. \\ & y_1 + y_3 \leq 1 \\ & y_1 + y_4 \leq 1 \\ & y_2 + y_4 \leq 1 \\ & -x_1 - x_2 \leq -1 \\ & y_1 - x_1 \leq 0 \\ & y_2 - x_1 \leq 0 \\ & y_3 - x_2 \leq 0 \\ & y_4 - x_2 \leq 0 \\ & y_1, y_2, y_3, y_4, x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

3. Uz pomoć DW dekompozicije riješiti sljedeći MILP problem:

$$\begin{aligned} & \min x_1 - 3x_2 \\ & s.t. \\ & -x_1 + 2x_2 \leq 6 \\ & x_1 + x_2 \leq 5 \\ & x_1, x_2 \geq 0. \end{aligned}$$

4. Uz pomoć DW dekompozicije riješiti sljedeći MILP problem:

$$\begin{aligned} & \min -4x_1 - x_2 - 6x_3 \\ & s.t. \\ & 3x_1 + 2x_2 + 4x_3 = 17 \\ & x \in X = \begin{cases} 1 \leq x_1 \leq 2 \\ 1 \leq x_2 \\ 1 \leq x_3 \leq 2. \end{cases} \end{aligned}$$

5. Proučiti opšti slučaj za DW dekompoziciju kada poliedar koji odgovara problemu nije ograničen (u njemu postoji zrak). Nakon toga, uz pomoć

DW dekompozicije riješiti sljedeći MILP problem:

$$\begin{aligned} & \min x_1 - x_2 - 2x_3 \\ & s.t. \\ & x_1 + x_2 + x_3 = 3 \\ & x \in X = \begin{cases} 0 \leq x_1 \leq 2 \\ 0 \leq x_2 \\ 0 \leq x_3 \leq 2. \end{cases} \end{aligned}$$

Glava 9

Metode za pronalaženje dopustivih rješenja

U prethodnim poglavljima prezentovane su najvažnije tehnike za egzaktno rješavanje problema iz oblasti Operacionih istraživanja. Sa druge strane, u teoriji i praksi se često javljaju i problemi koji se, zbog svoje složenosti i dimenzionalnosti, ne mogu riješiti do optimalnosti 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 ovom poglavlju ćemo prezentovati neke od tehnika za pronalaženje dopustivih rješenja. Čak i više od toga, prikazaćemo i neke od moćnih optimizacionih tehnika, koje ne garantuju uvijek pronalazak optimalnog rješenja, ali su se u praksi pokazale kao izuzetno moćan 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 razumnom (pohlepnom) kriterijumu. Preciznije, u svakoj iteraciji se u rješenje uključuje ona komponenta koja donosi najveći benefit 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 globalnog rješenja.

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 složenost.

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, pohlepni algoritmi mogu svojim rješenjem garantovati optimalnost, odnosno, za neke probleme se može pokazati da se optimalno rješenje uvijek može dobiti primjenom pohlepnog algoritma. Primjer takvog problema je *frakcioni problem ruksaka*, a u nastavku će biti prikazani još neki važni problemi koji se do optimalnosti rješavaju pohlepnim algoritmima.

Sa druge strane, većina optimizacionih problema se ne može optimalno riješiti pomoću pohlepnih algoritama. U tim slučajevima, ova paradigmа se koristi da bi se u kratkom vremenskom periodu obezbijedila dobra aproksimaciona rješenja, ili da se konstruiše neko polazno (dopustivo) rješenje, koje će se kasnije unapređivati naprednjim i složenijim algoritamskim tehnikama.

U ovoj sekciji razmatramo nekoliko poznatih pohlepnih algoritama koji se mogu naći u literaturi:

- *Kruskalov algoritam* za nalazak minimalnog pokrivajućeg stabla (MST);
- *Primov algoritam* za nalazak minimalnog pokrivajućeg stabla;
- *Dajkstrin algoritam* za nalaženje najkraćih puteva;
- Pohlepni algoritam za problem *najdužeg zajedničkog podniza* za proizvoljan skup stringova u ulazu.

Definšimo prvo pojam pokrivajućeg stabla, pa potom i minimalnog pokrivajućeg stabla (eng. *minimum spanning tree* - MST).

Neka je dat težinski graf $G = (V, E)$. Svaku težinu grane $uv \in E$ u grafu G , označimo sa $w(uv)$.

Definicija 20 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 .

Postoje nekoliko efikasnih algoritama za pronađak MST, koje u narednim sekcijama i opisujemo.

9.1.1 Primov algoritam

Ovo je jedan od osnovnih algoritama koji se, zbog svoje važnosti, ali i jednostavnosti, izučava u teoriji algoritama. Ovdje navodimo korake algoritma, bez ugađanja u detalje o njegovoj validnosti. Algoritam se sastoji od sljedećih koraka:

- Kreirati skup $mstSkup = \emptyset$ da bi se pratili svi čvorovi koji su već uključeni u MST.

-
- Dodijeliti vrijednosti svakom čvoru u ulaznom grafu: inicijalizovati vrijednosti na $+\infty$. Dodijeli vrijednosti 0 za čvor koji je izabran prvi za MST;
 - Dok god $mstSkup$ ne uključi sve čvorove iz V , radimo sljedeće korake:
 - Izabrati čvor $u \notin mstSkup$ sa minimalnom vrijednošću (pohlepni korak);
 - $mstSkup = mstSkup \cup \{u\}$;
 - Ažurirati vrijednosti svih susjeda čvora u na sljedeći način:
 - * Za svaki susjed v , ako je težina grane uv manja od ranije vrijednosti za v , dodijeliti vrijednost čvoru v na $w(uv)$.

Ideja dodjele vrijednosti čvorovima koji još nisu prisutni u MST je u tome da se efikasno izabere grana minimalne težine koja treba da se doda u trenutni parcijalno rješenje. Dakle, ove vrijednosti ukazuju na grane minimalne težine koje su povezane sa skupom čvorova koji su već uključeni u MST.

Kompleksnost algoritma. Lista susjedstva i pretraga: $O(|V|^2)$. Ako koristimo prioritetni red za nalaženje grane minimalne težine, imamo kompleksnost od $O(|E| \log(|V|))$.

MST ima primjene u (i) dizajniranju mreža (na primjer, telefonske mreže), gdje je do svakog domaćinstva (koje je predstavljeno jednim čvorom) potrebno obezbijediti liniju, tako da se u mrežu povežu sva domaćinstva, uz potrošnju što manje materijala za infrastrukturu mreže; (ii) aproksimaciji drugih teških optimizacionih problema kao što je problem trgovackog putnika (TSP). Ponovimo, u problemu TSP je potrebno naći najkraću konturu koja posjećuje sve čvorove u (težinskom) grafu. Primijetimo da su takvi putevi takođe MST. Težina MST je manja nego težina TSP, jer minimizujemo nad većim skupom rješenja, tako da se može pokazati da je nalazak MST-a u težinskom grafu aproksimira nalazak TSP-a (nad euklidskim težinskim grafovima); (iii) Klaster analiza. k -klaster problem se može posmatrati kao problem nalaženja MST-a, a potom brisanja $k - 1$ najskupljih grana iz MST.

9.1.2 Kruskalov algoritam

Ovo je još jedan pohlepni algoritam za nalaženje MST-a. Radi na principu izbora grana na osnovu njihove težine, gdje se u svakom koraku koristi pohlepna strategija da se bira odgovarajuća grana najmanje težine.

Neka je $mst = \emptyset$. U detalje, sljedeći koraci se izvršavaju ovim algoritmom:

1. Sortiranje svih grana u neopadajućem poretku;
2. Odabir najlakše grane i provjera da li se formira ciklus ako se ova grana doda u skup grana koje su do sada ubačene u skup mst ; Ako to nije

slučaj, ubaciti granu u trenutni skup mst . Inače, provjeriti granu koja je sljedeća u sortiranom nizu grana (pohlepni korak);

3. Ponoviti korak 2, dok se ne doda tačno $|V| - 1$ grana u mst .

Vremenska kompleksnost. 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, za izvršavanje algoritma treba $O(|E| \log |E| + |E| \log |V|)$ vremena. Kako je $|E| = O(|V|^2)$, dobijamo kompleksnost $O(|E| \cdot \log(|V|))$.

9.1.3 Dajkstrin algoritam

Ovaj algoritam se koristi za nalaženje 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 zbirna vrijednost $dist[u]$ i težine grane $w(uv)$ manja od trenutne vrijednosti $dist[v]$, vrijednost $dist[v]$ se ažurira jer je nađen novi najkraći put od s do 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 .

Vremenska kompleksnost. Implementacija Algoritma 3 ima vremensku kompleksnost $O(|V|^2)$. Ako je graf predstavljen listom susjedstva, i skup Q (linija 6, Algoritam 3) je implementiran kao binarni hip, kompleksnost 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 se literaturom ili internetom, upozna i sa ovim važnim algoritmom.

9.1.4 Frakcioni problem ruksaka

Definicija 21 Neka je dato n proizvoda sa svojim težinama i vrijednostima, te ruksak čiji je kapacitet W . Frakcioni problem ruksaka traži da se odaberu dijelovi (nekih) proizvoda i stave u ruksak a da vrijednost proizvoda u ruksaku bude što veća ali da se pri tom ne naruši kapacitet ruksaka.

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  $previous$ 
```

U 0-1 problemu ruksaka se ne dopušta dijeljenje proizvoda. Dakle, ili se uzima cijeli proizvod (1) i stavlja u ruksak, ili se uopšte ne uzima (0). U problemu frakcionog ruksaka, svaki proizvod posjeduje težinu i vrijednost, te se kao takav može podijeliti na dijelove (recimo dopušteno je da se uzme pola od ukupne količine proizvoda). Primjer takvih proizvoda bi bili mljeko, so, šećer itd. Efikasno riješavanje ovog problema se izvodi primjenom pohlepnog algoritma kako je u nastavku i opisano.

- Računajmo podjele količina/vrijednost za svaki proizvod, te se proizvodi sortiraju na osnovu tih vrijednosti.
- Uzmemo proizvod sa najvećom količničkom vrijednošću te ga dodamo u ruksak.
- Nastavljamo proces dodavanja sa sljedećim proizvodom dok ga ne budemo mogli da dodamo cijeli proizvod u ruksak. Ako je to slučaj, dodamo dio njega (onoliko koliko god je moguće, a da se pri tome kapacitet ruksaka ne naruši).

Može se pokazati sljedeća teorema.

Teorema 19 *Pohlepni algoritam opisan prethodno za frakcioni problem ruksaka uvijek nalazi optimalno rješenje.*

9.1.5 Pohlepni pristup za rješavanje LCS problema

String definišemo kao niz karaktera nad (konačnom) azbukom Σ . Problem najdužeg podniza (eng. *Longest common subsequence*), ili skraćeno LCS, je problem koji uzima u ulazu n stringova (označimo ovaj skup sa S) proizvoljne dužine. Njegov zadatak je da se nađe string maksimalne dužine tako da je zajednički podniz za sve stringove iz skupa S .

Neka su u ulazu data tri stringa dužine šest: $S = \{\text{abbccb}, \text{abccab}, \text{abbbcb}\}$. Rješenje problema je $s = \text{abcb}$. Primjetimo 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 proizvoljan ulaz, za problem je poznato da je NP-težak.

Jedan od najjednostavnijih pristupa u rješavanu ovog problema je *pohlepni pristup*, tzv. *Best-Next* (BN) heuristika. Prije nego 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 sa $S[p^L] = \{s_i[p_i^L, |s_i|] \mid i = 1, \dots, m\}$, gdje $|s|$ označava dužinu stringa s , a $s[x, y]$ podstring (sufiks) 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 $p^L = (1, 2, 2)$ i skup S sa početka ove podsekcije imamo $S[(1, 2, 3)] = \{s_1[1, |s_1|], s_2[2, |s_2|], s_3[3, |s_3|]\} = \{\text{bbccb}, \text{ccab}, \text{bc}\}$.

Dalje, sa $p_{i,a}^L$ definišemo poziciju prvog pojavljivanja karaktera $a \in \Sigma$ koja 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 potproblemu) $S[p^L]$ se nazivaju 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 od koraka algoritma. Inicijalno rješenje s je prazan string, koji je, trivijalno, dopustivo rješenje za svaku instancu LCS problema. Ideja je u svakom koraku proširiti rješenje za jedan karakter tako da spojimo taj karakter (sa desna) sa s dok god je to moguće i pri tome da je prošireni string i dalje LCS. Prema tome, krenemo sa pozicijom $p^L = (1, \dots, 1)$ koja odgovara čitavom skupu $S = S[p^L]$ i praznim stringom $s = \varepsilon$ kao inicijalno rješenje. Dalje, da bi 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 $S = S[p^L]$, jer na taj način ekstenzija $s = s \cdot x$ za bilo koji $x \in \Sigma_{p^L}$ će 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, biramo karakter x koji je najbliže početku u stringovima 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 preskače najmanji broj karaktera iz dalje pretrage. Sljedeći korak se sastoji u ažuriranju prostora pretrage za novi korak. Kako smo proširili rješenje za x , nove komponente rješenja koje su dopustive tražimo u skupu stringova $S[p_{1,x}^L + 1, \dots, p_{m,x}^L + 1]$. Postupak nastavljamo dok god postoji komponente koje mogu proširiti trenutno rješenje s .

Da sumiramo, pohlepni (BN) pristup za rješavanje LCS problema izvršava sljedeće korake:

- Kreće 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 $n + 1$, radimo sljedeće:
 - $\Sigma_{p^L} \leftarrow$ naći dopustive ekstenzije u $S[p^L]$
 - Od svih karaktera u Σ_{p^L} , odaberemo onaj koji minimizuje vrijednost funkcije

$$g(p^L, a) = \sum_{i=1}^m \frac{p_{i,a}^L - p_i^L + 1}{|s_i| - p_i^L + 1}, a \in \Sigma_{p^L}.$$

Neka je taj karakter označen 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 .

Primijetimo da ovakav algoritam ne može da garantuje optimalno rješenje.

Primjer. Neka su data dva stringa: $s_1 = \text{acb}$, $s_2 = \text{bac}$. Ako uzmemo b kao prvi karakter, rješenje koje dobijamo je b . Međutim, optimalno rješenje je string $s = \text{ac}$.

9.2 Aproksimativni algoritmi

Za veliki broj kompleksnih optimizacionih problema u praksi jednostavno ne postoje efikasni egzaktni algoritmi, te je za njihovo rješavanje potrebano izdvojiti ogromne vremenske/memorijske resurse. Možemo reći da za ovakve probleme nije poznat (egzaktan) algoritam koji se izvršava u polinomijalnom vremenu $O(|P|)$, gdje je $P \in \text{Poly}$ neki polinom iz vektorskog prostora polinoma Poly . Sa druge strane, većina pohlepnih algoritama, ali i naprednijih heuristika, ne daje nikakvu garanciju na kvalitet nađenog rješenja. Negdje između egzaktnih i heurističkih algoritama su se pozicionirali *aproksimativni algoritmi*, koji su polinomijalni, daju dopustiva rješenja uz garanciju o njihovom kvalitetu.

Algoritam pripada klasi *aproksimativnih algoritma* ako:

-
1. se izvršava u polinomijalnom vremenu;
 2. posjeduje garanciju da je nađeno (dopustivo) rješenje uvijek unutar nekog faktora (c) kvaliteta optimalnog rješenja, 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 ulaz I . Pretpostavimo da je riječ o problemu maksimizacije.

\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 apriksimativni algoritam* – ako je za svaku instancu problema P ispunjeno

$$|\mathcal{A}(I) - \mathcal{A}^*(I)| \leq c,$$

za neko $c \in \mathbb{R}^+$;

- *relativni apriksimativni algoritam* – ako je za svaku instancu problema P ispunjeno

$$\frac{\mathcal{A}(I)}{\mathcal{A}^*(I)} \geq c,$$

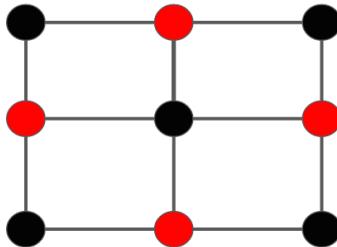
za neko $c \in \mathbb{R}^+$.

U tom slučaju kažemo da je algoritma \mathcal{A} c -aproksimativni algoritam. U literaturi ili na internetu, čitalac može pronaći primjere problema za koji postoji absolutni apriksimativni algoritam (na primjer, bojenje planarnog grafa, eng. *Planar graph coloring*), odnosno primjer problema za koji postoji relativni apriksimativni algoritam. U sljedećoj sekciji će biti prikazan jedan takav, problem, problem pokrivanja čvorova.

9.2.1 Aproksimativni algoritmi za problem pokrivanja čvorova

Neka je dat graf $G = (V, E)$. Potrebno je naći podskup $C \subseteq V$ najmanje kardinalnosti, takad da barem jedan čvor svake grane pripada skupu C . Dopustivo rješenje je svaki podskup C koji zadovoljava uslov problema. Takav skup se naziva *pokrivač* čvorova grafa G . Primjer pokrivača jednog grafa je dat na Slici 9.1 gdje pokrivač uključuje sve čvorove crvene boje.

Posmatrajmo (pohlepni) algoritam za ovaj problem dat Algoritmom 4.



Slika 9.1: Primjer pokrivača čvora.

Algoritam 4 Pohlepni algoritam za Problem 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:   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 mečinga (M), pa je prema tome, skup C uistinu pokrivač čvorova, prema definiciji mečinga. 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 sljedeći pohlepni Algoritam 5 za problem pokrivanja čvorova.

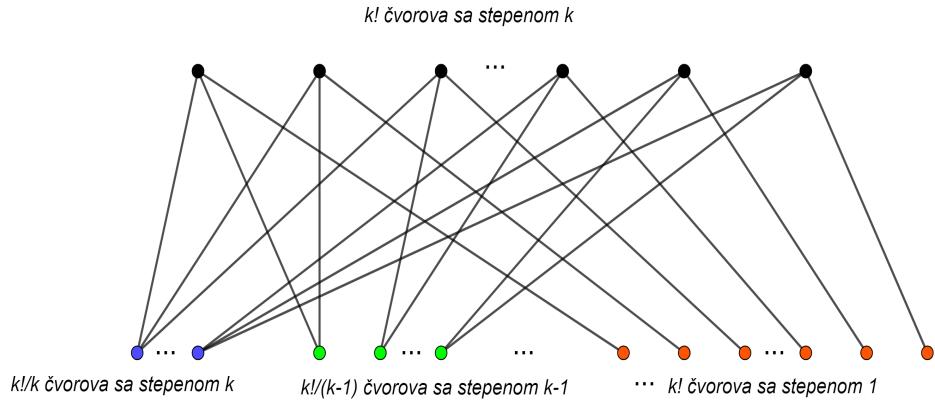
Algoritam 5 Pohlepni algoritam 2 za Problem pokrivanja čvorova

```

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:    $G \leftarrow$  ukloniti sve grane incidentne sa  $u$  u  $G$ 
6: end while
7: return  $C$ 
```

Izračunajmo aproksimativni faktor Algoritma 5 posmatrajući jedan specijalni graf sa Slike 9.2.

Optimalno rješenje je podskup čvorova C^* koji se sastoji od svih čvorova



Slika 9.2: Specijalan bipartitan graf.

sa vrha grafa (obojeni crnom bojom); dakle, ukupno ih je k ! Međutim, ako pratimo korake Algoritma 5, dobijemo da je

$$|C| = k! \cdot (1/k + 1/k - 1 + \dots + 1/2 + 1) \approx k! \cdot \log k.$$

Dakle, imamo da je

$$|C| \approx \log k \cdot |C^*|.$$

Prema tome, ovaj algoritam je $\log k$ -aproksimativni algoritam, za $k \in \mathbb{N}$. U literaturi, najbolji faktor je $2 - \frac{1}{\sqrt{\log n}}$ (Karakostas, 2009). Ono sto se takođe zna (Hastad, 2001) je da ne postoji aproksimativni faktor koji garantuje da je $c < \frac{7}{6}$ (osim ako $P \neq NP$).

9.2.2 Aproksimativni algoritmi za problem trgovackog putnika

Ovaj problem je težak kada je u pitanju konstrukcija aproksimativnog algoritma. Tome svjedoči sljedeća teorema.

Teorema 20 Ne postoji polinomijalan algoritam koji može aproksimirati TSP sa faktorom $c > 1$, osim ako je $P = NP$.

Kažemo da grane grafa zadovoljavaju nejednakost trougla akko za sve različite čvorove u, v, r grafa G vrijedi

$$w(u, v) + w(v, r) \geq w(u, r),$$

Algoritam 6 Aproksimativni algoritam 1 za TSP.

```
1: Ulaz: 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:   $uq$  se izbacuje iz  $T$ , ubacuje se  $utq$  // Ažururanje  $T$  (rekonstrukcija)
11: end while
12: return  $T$ 
```

gdje su $w(\cdot, \cdot)$ težine grane koja spaja dva čvora. Ovaj problem je poznat pod nazivom *Metrički TSP*. On je i dalje NP-težak, ali se aproksimativni algoritam ipak može konstruisati. U nastavku aproksimativne algoritme izvodimo za metrički TSP. Posmatrajmo *algoritam najbližeg dodavanja* koji aproksimira TSP, dat u Algoritmu 6.

Ovo je očigledno polinomijalan algoritam. Ispitajmo koji aproksimacioni faktor se njegovim izvršavanjem garantuje. Primijetimo 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 koja je barem dva puta veća od cijene MST. Prema tome, ovo je 2-aproximativni algoritam.

Pokušajmo poboljšati 2-aproximativni Algoritam 6 uz pomoć *Double tree algoritma*, čije detalje objašnjavamo u nastavku. Prije toga ćemo uvesti nekoliko definicija i teorema od bitnog značaja u konstrukciji algoritma.

Definicija 22 Neka je dat graf $G = (V, E)$. Ojlerova kontura T^o grafa G je kontura $e_1 e_2 \dots, e_m e_1$ koja obilazi sve grane grafa G .

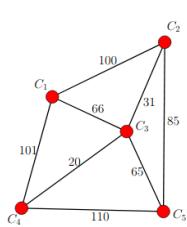
Vrijedi sljedeća karakterizacija grafova koji imaju Ojlerovu konturu.

Teorema 21 Graf G sadrži Ojlerovu konturu akko je povezan i svaki njegov čvor je paran. Takva klasa grafova se naziva *Ojlerovom klasom grafova*.

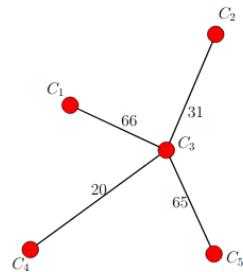
Algoritam 7 Aproksimativni algoritam 2 za TSP.

- 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
-

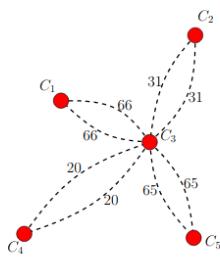
Pokažimo da ovim algoritmom dobijamo 2-aproksimaciju. Jasno je da je algoritam polinomijalan i da je dobijeno rješenje dopustivo. Pokažimo sada da ovim ostvarujemo faktor 2. Neka je C^* cijena optimalne Hamiltonove konture. Tada je $MST \leq C^*$. Lako se vidi da je cijena Ojlerove konture $\leq 2 \cdot C^*$. 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 $\leq 2C^*$, pa je ovaj algoritam 2-aproksimativni.



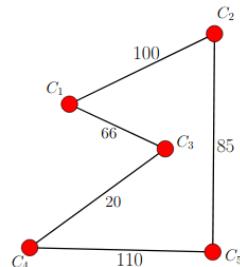
(a) Graf G .



(b) MST grafa G



(c) Ojlerova kontura:
 $C_1C_3C_4C_3C_5C_3C_2C_3C_1$.



(d) Hamiltonova kontura (ciklus):
 $C_1C_3C_4C_5C_2C_1$.

Slika 9.3: Koraci u konstrukciji Kristofidjesovog algoritma.

Posmatrajmo sljedeći (aproksimativni) algoritam koji umanjuje aproksimativni faktor TSP sa 2 na 1.5. Algoritam je poznat pod nazivom *Kristofidjesov* algoritam i dat je u Algoritmu 8.

Algoritam 8 Kristofidjesov algoritam.

-
- 1: **Ulaz:** Graf G
 - 2: **Izlaz:** Hamiltonova kontura T
 - 3: $T \leftarrow$ Naći MST grafa G
 - 4: $M \leftarrow$ Naći minimalan mečing 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 22 Kristofidesov algoritam ima faktor aproksimacije $c = 1.5$.

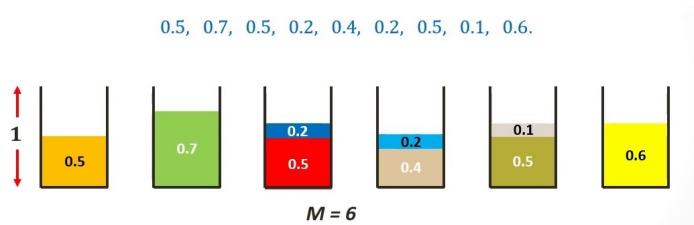
Dokaz: Cijena TSP C^* dostiže najviše cijenu Ojlerove konture, tj. $C^* \leq \text{cost}(T^o) = \text{cost}(T) + \text{cost}(M)$. Takođe jasno je da vrijedi $\text{cost}(T) \leq C^*$. Pokažimo da je $\text{cost}(M) \leq \frac{C^*}{2}$, odakle bi slijedila tvrdnja. Primijetimo da u optimalnoj konturi T postoje dva mečinga: 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

Definicija 23 Neka je dato n proizvoda veličine a_1, a_2, \dots, a_n ($0 < a_i < 1$). Potrebno je spakovati sve proizvode u jedinična pakovanja, tako da je broj korištenih pakovanja minimalan.

Primjer jedne instance problema pakovanja je dat na Slici 9.4.



Slika 9.4: Primjer Binarnog Pakovanja.

First-fit algoritam. Ovaj pohlepni algoritam se sastoji do sljedećih koraka:

-
- Stavljamo proizvode u 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 koji može da stane.
 - Ako ne odgovara niti jednom od otvorenih pakovanja, otvoriti novo pakovanje i smjestiti taj proizvod u novo pakovanje.

Analiza algoritma. Ovo je polinomijalan algoritam koji vraća dopustivo rješenje. Neka je C^* optimalan broj pakovanja koji se otvara da bi se spakovalo svih n stavki. Pretpostavimo da algoritam vraća rješenje C (pakovanja). Iz toga je barem $C - 1$ pakovanja više od polovine mesta je iskorišteno. Odатле je

$$C^* \geq \sum_{i=1}^n a_i > \frac{C-1}{2} \Rightarrow 2 \cdot C^* > C - 1 \Rightarrow 2 \cdot C^* \geq C,$$

pa slijedi da je algoritam 2-aproksimacioni.

Još jedan pohlepni algoritam za ovaj problem je *Best fit* koji radi na sljedeći princip. Sljedeći proizvod se stavlja u pakovanje koje ostavlja najmanju prazninu od svih pakovanja (koja su otvorena) kada se taj prizvod smjesti u odgovarajuće pakovanje. Ako takvo pakovanje ne postoji, otvoriti 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 $P = NP$), kako je pokazano u literaturi.

9.2.4 Aproksimativni algoritmi za problem pokrivanja skupa

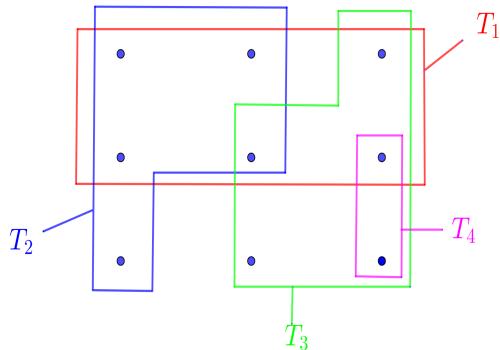
Ovaj problem predstavlja uopštenje Problema pokrivanja čvorova, i definisan je na sljedeci način.

Definicija 24 Neka je dat skup elemenata X te familija podskupova \mathcal{F} skupa X tako 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 tako da je

$$\bigcup_{f \in C} f = X.$$

U nastavku izlažemo pohlepnu heuristiku koja predstavlja aproksimativni algoritam sa logaritamskim faktorom. To znači, kako je veličina ulaza veća, kvalitet rješenja aproksimativnog algoritma opada relativno u odnosu na veličinu optimalnog rješenja. Kako logaritam kao funkcija raste sporim intenzitetom, ovaj aproksimacioni algoritam ipak daje korisne rezultate.

Što se tice motivacije za ovaj problem, pretpostavimo da imamo skup karakteristika, te skup ljudi koji posjeduju pojedine karakteristike. Potrebno je formirati komisiju sa što manjim brojem osoba tako da se za svaku



Slika 9.5: Primjer Pokrivanja Skupa ($\mathcal{F} = \{T_1, T_2, \dots, T_4\}$), optimalno rješenje je skup $\mathcal{C} = \{T_2, T_3\}$

Algoritam 9 GREEDY-SET-COVER metod za Problem Pokrivanja Skupa.

```

1: Ulaz:  $X$  i familija podskupova  $\mathcal{F}$  od  $X$ 
2: Izlaz: pokrivač skupa  $X$ 
3:  $U \leftarrow X$ 
4:  $C \leftarrow \emptyset$ 
5: while  $U \neq \emptyset$  do
6:    $f \leftarrow \operatorname{argmax}_{f \in \mathcal{F}} |\{f' \mid f' \in U \wedge |f \cap f'| \geq 1\}|$ 
7:    $U \leftarrow U \setminus f$ 
8:    $C \leftarrow C \cup \{f\}$ 
9: end while

```

karakteristiku može naći osoba iz komisije da je stručna u tome. Pohlepni algoritam za ovaj problem je dat u Algoritmu 9. Skup U predstavlja skup preostalih nepokrivenih elemenata dok skup \mathcal{C} predstavlja pokrivač koji se konstruiše kroz iteracije algoritma. U svakom koraku, algoritam nastoji da proširi pokrivač \mathcal{C} na račun smanjenja skupa U . Kandidata za proširivanje pokrivača biramo na osnovu pohlepne funkcije data u liniji 6. U osnovi, biramo onaj $f' \in \mathcal{F}$ koji ima najviše elementima iz U sa kojim ima neprazan presjek sve dok je $U \neq \emptyset$. Algoritam se izvodi u $O(|\mathcal{F}| \cdot |X|)$ vremenu, dakle polinomijalnom. Takođe, jasno je da je rješenje koje se vraća dopustivo. Izračunajmo još aproksimativni faktor algoritma. Prije toga, označimo d -ti harmonijski broj $= H_d = \sum_{i=1}^d 1/i$ sa $H(d)$. Vrijedi sljedeća teorema.

Teorema 23 *GREEDY-SET-COVER algoritam ima aproksimativni faktor*

$$H(\max\{|f| \mid f \in \mathcal{F}\}).$$

Dokaz: Označimo sa f_i , i -ti podskup koji je selektovan od strane algoritma: algoritam dodjeljuje cijenu 1 kad se f_i doda u \mathcal{C} . Cijenu selektovanja f_i distribuiramo podjednakom vjerovatnoćom na sve elemente koji su po prvi put pokriveni sa f_i . Posmatrajmo primjer dat na Slici 9.5. U prvoj iteraciji pohlepnog algoritma, 6 elemenata je pokriveno sa T_1 ; prema tome, cijena svakog od ovih elemenata je $c_x = \frac{1}{|T_1 \setminus \emptyset|} = \frac{1}{6}$. Dalje, u drugoj iteraciji algoritam odabire skup T_3 , koji pokriva 2 nova elementa; njihova cijena je $c_x = \frac{1}{|T_3 \setminus T_1|} = \frac{1}{2}$. U iteraciji broj 3, odabere se skup T_2 , te se jedan novi element pokrije; njegova cijena je jednaka $c_x = \frac{1}{|T_2 \setminus (T_1 \cup T_3)|}$.

Neka c_x označava cijenu za pokrivanje elementa $x \in X$. Svakom elementu se cijena dodjeli najviše jednom, onog puta kada se pokrije po prvi put. Ako se x pokriva prvi put sa f_i , tada imamo:

$$c_x = \frac{1}{f_i \setminus (f_1 \cup \dots \cup f_{i-1})}.$$

Algoritam generiše rješenje \mathcal{C} , koje je cijene $|\mathcal{C}|$ i ova cijena se raspoređuje elemenatima iz X . Kako optimalni pokrivač C^* takođe pokriva skup X , dobijamo

$$|\mathcal{C}| = \sum_{x \in X} c_x \leq \sum_{f \in C^*} \sum_{x \in f} c_x. \quad (9.1)$$

Pokazimo sljedeću ključnu nejednakost koja služi u dokazu ove teoreme. Za svaku $f \in \mathcal{F}$, vrijedi

$$\sum_{x \in f} c_x \leq H(|f|). \quad (9.2)$$

Iz formula (9.1) i (9.2) bi dobili da je

$$|\mathcal{C}| \leq \sum_{f \in C^*} \sum_{x \in f} c_x \leq \sum_{f \in C^*} H(|f|), \quad (9.3)$$

odakle bi slijedila tvrdnja o vrijednosti aproksimacionog faktora algoritma. Ostaje nam još da pokažemo formulu (9.2).

Neka je $f \in \mathcal{F}$ tako da za $i = 1, \dots, |\mathcal{C}|$

$$u_i(f) = |f \setminus (f_1 \cup \dots \cup f_i)|$$

označava ostale elemente u $f \in \mathcal{F}$ koji su još uvijek nepokriveni nakon što je algoritam dodojao skupove f_1, \dots, f_i u pokrivač C , posmatrano do i -te iteracije. Definišimo $u_0(f) = |f|$ za broj elemenata skupa f koji je inicijalno nepokriven. Označimo sa k prvi indeks tako da je $u_k(f) = 0$ – što znači da

su svi elementi skupa f u k -toj iteraciji pokriveni prvi put nekim skupovima koji su odabranu u ranijim koracima algoritma. Primijetimo da vrijedi

$$u_o(f) \geq u_1(f) \geq \dots \geq u_{k-1}(f) \geq u_k(f).$$

Prema tome, $u_{i-1}(f) - u_i(f)$ predstavlja broj elemenata u skupu f koji su pokriveni po prvi put sa f_i u i -toj iteraciji za $i = 1, \dots, k$. Dakle, vrijedi

$$\sum_{x \in f} c_x = \sum_{i=1}^k (u_{i-1}(f) - u_i(f)) \cdot \frac{1}{f \setminus (f_1 \cup \dots \cup f_{i-1})}, \quad (9.4)$$

što je jednako sumi brojeva elemenata skupa f pokriven sa f_i pomnožen sa cijenom koja je dodjeljena svakom elementu u i -toj iteraciji. Primijetimo da vrijedi:

$$|f_i \setminus (f_1 \cup \dots \cup f_{i-1})| \geq |f \setminus (f_1 \cup \dots \cup f_{i-1})| := u_{i-1}, \quad (9.5)$$

jer je f_i pohlepni izbor (njegovim izborom pokrićemo čitav (nepokriveni dio) skupa f_i) u i -toj iteraciji i svaki drugi izbor ne može pokriti više novih (nepokrivenih elemenata).

Prema tome, iz jednakosti (9.1), te nejednakosti (9.2), slijedi

$$\sum_{x \in f} c_x \leq \sum_{i=1}^k (u_{i-1} - u_i) \cdot \frac{1}{u_{i-1}}$$

Izraz sa desne strane nejednakosti se može ograničiti na sljedeći način:

$$\begin{aligned} \sum_{i=1}^k (u_{i-1} - u_i) \cdot \frac{1}{u_{i-1}} &= \sum_{i=1}^k \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{u_{i-1}} \leq \sum_{i=1}^k \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{j} \\ &= \sum_{i=1}^k \left(\sum_{j=1}^{u_{i-1}} \frac{1}{j} - \sum_{j=1}^{u_i} \frac{1}{j} \right) \\ &= \sum_{i=1}^k (H(u_{i-1}) - H(u_i)) \\ &= H(u_0) - H(u_k) = H(u_0) = H(|f|), \end{aligned}$$

odakle slijedi tvrđenje. □

Napomenimo da postoji i težinska verzija Problema pokrivanja skupa. U ovoj verziji je svakom skupu iz $f \in \mathcal{F}$ data težina w_f . Zadatak je naći pokrivač $C \subseteq \mathcal{F}$ skupa X sa najmanjom težinom. Ovaj problem na sličan način kao i ne-težinska varijanta može aproksimirati do na faktor $H_d =$

$\log d + O(1)$, gdje je $d = \max\{|f| : f \in \mathcal{F}\}$ uz pomoć sličnog pohlepnog algoritma. U tom slučaju, pohlepni kriterij je određen funkcijom

$$g(f) = \frac{w_f}{|f \cap U|}, f \in \mathcal{F} \setminus C, \quad (9.6)$$

gdje je w_f težina skupa. U svakom koraku algoritma određuje skup $f^* \in \mathcal{F}$ koji minimizuje vrijednost funkcije g , koji se zatim dodaje u trenutni (parcijalni) pokrivač C .

9.2.5 Aproksimativni algoritam za problem najkraćeg nadstringa

Definicija 25 Neka je dat skup stringova $S = \{s_1, \dots, s_n\}$. Problem najkraćeg nadstringa traži string s minimalne dužine koji sadrži sve stringove iz S za njegove podstrigove.

Bez gubljenja opštosti, mozemo pretpostaviti da niti jedan string s_i nije podstring stringa s_j , $i \neq j$. Ovaj problem je NP-težak. Ideja je riješiti ga korišteći Problem pokrivanja skupa i njegove algoritame iz prethodne sekcije. Pokazaćemo da postoji $2 \cdot H_n$ -aproksimacija za problem najkraćeg nadstringa.

Svakom stringu θ dodijelimo skup $set(\theta)$ tako da $set(\theta)$ pokriva string $s \in S$ akko je s podstring od θ . Prema tome

$$set(\theta) = \{s \in S \mid s \text{ podstring od } \theta\}.$$

Odatle bi pokrivač skupa bila familija takvih skupova $set(\theta)$, iz kojih se konstruiše nadstring skupa S spajajući sve θ stringove u jedan string. U svakom slučaju familija \mathcal{F} se ne može definisati kao skup skupova $set(\theta)$ jer bi u suprotnom \mathcal{F} bio beskonačan. Sa druge strane, ne možemo ograničiti sve stringove θ samo na one iz S , jer bi u suprotnom dobili rješenje koje je jednako konkatenaciji svih stringova iz S , što je svakako rješenje koje nije korisno. Za uspostavljanje balansa, neka iz skupa $S = \{s_1, \dots, s_n\}$ uzmemo par stringova s_i i s_j , nađemo $k > 0$ tako da je posljednjih k karaktera stringa s_i jednaka prvih k karaktera stringa s_j (poklapanje). Označimo sa θ_{ijk} string koji se dobije preklapanjem stringova s_i i s_j na k karaktera. Oznacimo sa I skup strinova θ_{ijk} za validne izbore i, j i k . Zatim definišemo $\mathcal{F} = \{set(\theta) \mid \theta \in S \cup I\}$. Dodijeljena cijena svakog skupa $set(\theta)$ je jednaka $|\theta|$. Na osnovu prethodnog, dobijamo algoritam za rješavanje problema Najkraćeg nadstriga Algoritmom 10.

Analiza algoritma.

Ako je OPT_{ns} vrijednost optimalnog rješenja problema nad instancom (X, \mathcal{F}) za problem najkraćeg nadstringa, a OPT_{ss} dužina najkraćeg nadstringa, može se pokazati da vrijedi

$$\text{OPT}_{ns} \leq 2 \cdot \text{OPT}_{ss}.$$

Iz prethodnog bi slijedilo da Algoritam 10 daje $2H_n$ -aproksimaciju.

Algoritam 10 Gridi heuristika za Problem Najkraćeg Nadstringa.

- 1: **Ulaz:** skup $S = \{s_1, \dots, s_n\}$
 - 2: **Izlaz:** aproksimacioni nadstring s
 - 3: Izračunati instancu (X, \mathcal{F}) problema najkraćeg nadstringa
 - 4: $\mathcal{C} = \{set(\theta_1), set(\theta_2), \dots, set(\theta_p)\} \leftarrow$ Primijeniti GREEDY-SET-COVER(X, \mathcal{F})
 - 5: $s \leftarrow \theta_1 \dots \theta_p$
-

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 maksimalno mogu da odstupaju od optimalnog rješenja, a to odstupanje je određeno faktorom kvaliteta.

Posebna vrsta približnih metoda (metoda koje ne daju i/ili ne garantuju pronađak optimalnog rješenja pri smo 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 primijeniti tehniku pomoću 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 (na primjer, na osnovu rješavanja problema manjih dimenzija ili uopštavanjem tehnika uspješno primjenjenih na uzorcima).

Snaga heurističkih algoritama leži u smanjenju prostora pretraživanja i usmjeravanju procesa pretraživanja u one regije koji sadrže bolje 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 optimal-

nog rješenja odstupa u dozvoljenim granicama) ili zavisne od drugih uslova (na primjer, 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, na primjer, poređenjem sa rješenjima dobijenim uz pomoć drugih metoda. Nekada je cilj heurističke metode samo da se pronađe jedno dopustivo rješenje, dok se u kvalitet samog rješenja uopšte i ne ulazi.

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 i iskorištavanje čitavog prostora rješenja, da bi se u razumnom vremenu, na efikasan način, odredilo rješenje što bliže optimalnom.

Sa praktičnog, upotrebnog aspekta, cilj metaheuristika je pretraživanje skupa dopustivih rješenja, pri čemu je (ukoliko je potrebno) dozvoljeno i proširenje tog skupa i 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 se razvile 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 klase problema na kojima se primjenjuju. U savremenoj literaturi, sreće se nekoliko kriterijuma za klasifikaciju metaheuristika.

Klasifikacija metaheuristika prema polaznoj motivaciji identificuje 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 (na primjer, tabu pretraživanje, iterativna lokalna pretraga itd.)

Prema broju tačaka kojima se raspolože 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 istom trenutku algoritam raspolože sa više tačaka – različitim rješenja problema. Pretraživanje prostora je zasnovano na primjeni različitih operatora rekombinovanja, pomjeranja, izmjena i odabira pojedinih tačaka, kako bi se u svakoj novoj iteraciji formirala nova populacija, sa potencijalno boljim jedinkama. Takve metaheuristike su: genetski algoritmi, mravlje kolonije, elektromagnetizam, jata ptica itd. U metaheuristikama koje raspoložu jednom tačkom, primjenjuje se strategija zasnovana na praćenju trajektorije (eng. *trajectory methods*), koji 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 itd.

Prema tipu funkcije cilja, može se metaheuristike se mogu klasifikovati na one koje koriste staticku, te one koje koriste dinamičku funkciju cilja.

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;
- upravljanje lokalno pretraživanje (eng. *Guided Local Search*) – GLS.

U nastavku ovog poglavlja, detaljnije će biti objašnjena po jedna metaheuristica iz grupe populacionih metoda (genetski algoritam) i iz grupe metoda koje u procesu pretrage raspoložu 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.

9.4.2 Genetski algoritmi

Genetski algoritmi spadaju u klasu populacionih i evolucijskih algoritama, čije je ponašanje inspirisano procesom evolucije koji se odvija u prirodi. Kao što je već rečeno, u osnovi se podrazumijeva rad sa populacijom 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 jo 175. godine od strane Džona Holanda, u čuvenoj knjizi "Adaptation in natural and artificial system".

Svakoj jedinki se dodjeljuje brojevna vrijednost, tzv. prilagođenost (engl. *fitness*), koja ocjenjuje kvalitet posmatrane jedinke. Cilj genetskog algoritma je da se, iz iteracije u iteraciju, pronalaze jedinke sa sve boljom prilagođenošću, pod čime se smatra i pojedinačno poboljšanje svake jedinke, ali i prosječna prilagođenost kompletne populacije, što se postiže standardnim genetskim operatorima: *selekcijom, 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 kodom, koji se zapisuje nad nekom konačnom azbukom (najčešće binarnom ili pak cjelobrojnom).

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 pretpostavlja 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 jedinke sa boljom prilagođenošću se mogu primijeniti i tehnikе lokalnog pretraživanja).

Operator selekcije bira jedinke koje preživljavaju u procesu evolucije. Po pravilu, boljim jedinkama (jedinkama sa boljom prilagođenošću) se operatom selekcije daje veća vjerovatnoća da će preživjeti u narednoj generaciji, dok slabije jedinke selekcijom dobijaju manju šansu da prežive, te se tako izbacuju iz populacije. Postoji veći broj tehnika selekcije, a najpoznatije su: jednostavna (ili rulet) selekcija, turnirska, eliminacijska selekcija i elitizam. Rulet selekcija diktira da je vjerovatnoća da će jedinka preživjeti za narednu generaciju proporcionalna prilagodenosti. Kod turnirske selekcije se formiraju skupovi jedinki, od kojih samo ona sa najboljom prilagođenošću opstaje i ulazi u sljedeću generaciju. Eliminacijska selekcija je primarno zasnovana na izbacivanju loših jedinki, prije nego na odabiru boljih za narednu generaci-

ju. Elitizam podrazumijeva da će nekoliko najboljih (elitnih) jedinki direktno preći u narednu generaciju, čime se daje pouzdana pretpostavka da algoritam teži globalnom optimumu.

Ukrštanjem se rekombinuju geni jedinki. Rezultat ukrštanja jedinki su nove jedinke, koje potencijalno sadrže "dobre" gene roditelja od kojih su nastale. Ovim mehanizmom i jedinke slabije prilagodenosti, 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 tačke prekida na kojima se odvija razmjena genetskih kodova 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 i ukrštanja. 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.

Osnovna šema funkcionalisanja genetskog algoritma je prikazana Algoritmom 11.

N_{pop} označava ukupan broj jedinki u populaciji. N_{elite} je broj elitnih jedinki, dok su promjenljivima i i obj_i predstavljene odgovarajuća jedinka sa njenom vrijednošću funkcije cilja (fitnes vrijednost).

Algoritam 11 Osnovna šema GA.

```
1:  $P \leftarrow \text{GenerisiInicijalnuPopulaciju}()$ 
2: while  $\text{!IspunjeniKriterijZaustavljanja}()$  do
3:   for  $i = N_{elite} + 1; i < N_{pop}; ++i$  do
4:      $Obj_i \leftarrow \text{RacunajFunkcijuCilja}(P_i)$ 
5:   end for
6:    $\text{RacunajFunkcijuPrilagodjenosti}(P)$ 
7:    $\text{IzvrsiSelekciju}(P)$ 
8:    $\text{IzvrsiMutaciju}(P)$ 
9:    $\text{IzvrsiUkrstanje}(P)$ 
10: end while
```

Primjer. U ovom primjeru posmatrajmo 0-1 problem ruksaka. Ponovimo da nam je, kod ovog problema, na raspolaganju određen broj proizvoda, od

kojih svaki proizvod ima svoju vrijednost i težinu. Ruksak je ograničenog maksimalnog kapaciteta (težine) C . Podsjetimo 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 vrijenost 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, $x_i = 0$. Ponovimo iz Sekcije 6, ovaj problem se definiše sa ILP modelom:

$$\max\{f(x)\} = \max\left\{\sum_{i=1}^n x_i p_i\right\}$$

pod uslovima

$$\sum_{i=1}^n x_i w_i \leq C$$

$$x_i \in \{0, 1\}, i = 1, 2, \dots, n$$

Konstrukciju elemenata genetskog algoritma ćemo prikazati služeći se konkretnim primjerom. U tabeli su prikazani polazni podaci vezani za proizvode, a maksimalan kapacitet rukasaka je 22.

Kodiranje. Genetski kod svake jedinke ćemo prikazati binarnim nizom dužine n (u primjeru je $n = 7$). Tako, na primjer, genetski kod 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 $7+8+4 = 19$. Primijetimo da ovakav sistem kodiranja može da dovede do nedopustivih rješenja. Na primjer, genetski kod 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 za 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 minimimalnu 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 fitnesa. 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 ćešće 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 u startu odbacuje.

Računanje funkcije prilagođenosti. Za posmatrani problem, najprirodnija funkcija prilagođenosti bi se mogla definisati na način da ona bude jednak 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 ukrštanja. Kao što je već pomenuto, osnovni operatori ukrštanja su tzv. jednopoziciono i dvopoziciono ukrštanje. Na našem primjeru ćemo objasniti jednopoziciono ukrštanje, koje bi moglo biti realizovano na sljedeći način:

	Prije ukrštanja:	Poslije ukrštanja:
Jednika 1:	XXX XXXX	XXXYYYY
Jedinka 2:	YYY YYYY	YYYXXXX

Tabela 9.1: 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 koda prvog roditelja spaja sa desnim dijelom genetskog koda drugog roditelja i obrnuto, lijevi dio genetskog koda drugog roditelja se ukršta sa desnim dijelom prvog roditelja. Treba napomenuti da se primjenom ovakvog operatora ukrštanja takođe mogu dobiti potomci koji su nedopustive jedinke (bez obzira na to da li su jedinke roditelja dopustive ili ne), a njihova dalja egzistencija u populaciji ponovo zavisi od načina konstruisanja funkcije prilagođenosti.

Kod dvopozicionog ukrštanja postoje dvije pozicije u svakoj od jedinki na osnovu kojih se radi ukrštanje. Generička procedura ovakvog ukrštanja je data sljedećom tabelom:

	Prije ukrštanja:	Poslije ukrštanja:
Jednika 1:	XXX XXXX XX	XXXYYYYYXX
Jedinka 2 :	YYY YYYY YY	YYYYXXXXYY

Tabela 9.2: 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.

Kod uniformnog ukrštanja, za svaki par jedinki koje se ukrštaju, prvo se generiše maska na slučajan način, tj. binarni niz dužine genetskog koda 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 druge jedinke vrijednost i -tog gena druge jedinke. Inače, vrijedi obrnuta situacija. Jedan primjer ovakvog ukrštanja je data sljedećom tabelom:

Maska:	110001100
Jednika 1:	XXXXXXXXXX
Jedinka 2:	YYYYYYYYYY
Poslije ukrštanja:	
Nove jedinke:	XXYYYYXXYY YYXXXXYYXX

Tabela 9.3: Primjer operatora ukrštanja: uniformno ukrštanje.

Mutacija. Kao što je pomenuto, mutacije podrazumijevaju male promjene u genetskom kodu, čija je osnovna namjena izlazak iz lokalnih podoptimalnih rješenja i omogućavanje dalje pretrage u cilju pronalaženja boljih rješenja. Najčešće se mutacija realizuje kao promjena vrijednosti jednog gena. U našem slučaju, 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 modifikovala 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 modifikovala trenutnu jedinku na način da se neki proizvod, koji je bio u rješenju, islujuči 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.

Maska:	10001000100
Jedinka:	11001001000
<hr/>	
Nakon mutacije	01000001100

Tabela 9.4: Primjer operatora mutacije (uniformno mutiranje)

Operator selekcije. Da bismo ilustrovali princip rada najjednostavnije rulet selekcije, naprije pretpostavimo da je funkcija prilagođenosti jednaka vrijednosti funkcije cilja. Dalje, posmatrajmo populaciju od 4 jedinke, prikazane u Tabeli 9.5.

Ukupan fitnes 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

	fitnes							težine	
Jedinka1	1	0	0	0	0	1	1	18	17
Jedinka2	0	0	0	0	0	1	0	9	6
Jedinka3	0	1	0	1	1	0	0	17	22
Jedinka4	0	1	0	0	1	1	1	28	22

Tabela 9.5: Primjer korišten za ilustraciju upotrebe rulet selekcije

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 jednakе, 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.

Na kraju ovog odjeljka treba pomenuti da je višedecenijska primjena genetskih algoritama u rješavanju velikog broja problema kombinatorne optimizacije doveo 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.

9.4.3 Metoda lokalne pretrage

Metoda lokalne pretrage je tehnika čija se strategija sastoji u izvođenju niza poboljšanja rješenja počevši od nekog unaprijed zadanog, radeći male promjene u trenutno najboljem rješenju. Uvedimo prvo koncept okolina i lokalnih rješenja kao baznih pojmova u metodologiji lokalne pretrage. Za dato rješenje x , definisimo 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 -promjenjiva (eng. *k-exchange*) okolina: x i x' pripadaju istoj okolini akko se razlikuju na tačno k komponenti rješenja. Npr. za TSP problem, 2-promjenjiva 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) u

Algoritam 12 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**
-

odnosu na 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 pomenuti da nije svaki lokalni optimum ujedno i globalni.

Pseudokod lokalne pretrage je dat u Algoritmu 12. Ulaz u algoritam je instanca problema i jedna definisana okolina. 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 rekursivno seli u tačku $x = x'$, odakle se pokreće nova iteracija lokalne pretrage.
- *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 zahtjeva enumeraciju svih rješenja iz okoline pa često ovakav proces u praksi traje dosta dugo, tako da ona nije uvijek primjenjiva.

Primjer. U problemu 0-1 ruksaka, za okolinu \mathcal{N} bi mogli uzeti 1-swap okolinu. Recimo, za $n = 4$ i rješenje $x = 1100$ imali bismo

$$\mathcal{N}(x) = \{0100, 1000, 1110, 1101\}.$$

9.4.4 Metoda promjenljivih okolina

Metoda promjenljivih okolina (VNS) je metaheuristika uvedena od strane Nenada Mladenovića i Pjera Hansena u drugoj polovini devedesetih godina prošlog vijeka. 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 veza između lokalnih minimuma. Stoga metod promjenljivih okolina i ne prati unaprijed zadatu putanju, već “skče” na potencijalno bolja rješenja koja se nalaze u nekoj okolini trenutno najboljeg rješenja. (Ovaj proces je poznat pod nazivom diversifikacija – eng.

diversification.) Na ovaj se način očuvavaju dobre osobine trenutnog rješenja (ako su neke promjenljive već dostigle optimalne vrijednosti), dok se u novim okolinama pokušavaju poboljšati preostale promjenljive. Ovaj sistem je dodatno ojačan sistemom lokalnog pretraživanja, kojim se prelazi iz trenutnog rješenja u najkvalitetnije rješenje u nekoj njegovoј 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 nekoј okolini polaznog lokalnog optimuma, predstavlja razuman i, kao što je to slučaj u velikom broju raznih optimizacionih problema, opravdan pristup. Da bi se postigli pomenuti efekti “izlaska” 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 strategije.

Da bi se proširila pretraga, VNS u potrazi za boljim lokalnim optimumom, obično koristi okoline rastuće kardinalnosti. Da bi se definisao skup okolina, pretpostavimo da je x proizvoljno rješenje u N_k , za $k = k_{min}, \dots, k_{max}$, konačan skup struktura okolina. Tada se $N_k(x)$ definiše kao skup rješenja u k -toј okolini tačke x . Način funkcionisanja ove metode prikazan je Algoritmom 13.

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 razmrđavanja 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 razmrđavanja), 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. 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

Algoritam 13 VNS metaheuristika

```
1: Ulaz: inicijalno rješenje  $s$ , okoline  $\mathcal{N}_1, \dots, \mathcal{N}_{k_{\max}}$ 
2: Izlaz: (poboljšano) rješenje  $s$ 
3: while  $\neg(IspunjeniKriterijZaustavljanja())$  do
4:    $k \leftarrow k_{\min}$ 
5:   while  $k \leq k_{\max}$  do
6:      $s' \leftarrow$  izabrati random rješenje iz  $\mathcal{N}_k(s)$  // shaking phase
7:      $s' \leftarrow LokalnaPretraga(s')$ 
8:     if  $f(s') > f(s)$  then
9:        $s \leftarrow s'$ 
10:       $k \leftarrow 1$ 
11:    else
12:       $k \leftarrow k + 1$  // koristiti narednu (VNS) okolinu
13:    end if
14:   end while
15: end while
16: return  $s$ 
```

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, nije.

Vrijednost funkcije cilja. Slično kao u primjeru genetskog algoritma, vrijednost funkcije cilja se najlakše može definisati kao zbir vrijednosti proizvoda uključenih u rješenje. Eventualno, ako se u algoritmu dozvoljava rad sa nedopustivim rješenjima, funkcija cilja može da sadrži i dio za "kažnjavanje" nedopustivih rješenja. Razmatranje o kažnjavanju nedupistivih 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 status: Ako je izabran proizvod koji je uključen u ruksak, on se izbacuje iz ruksaka i obrnuto. Formalno, k -ta okolina rješenja x se zapisuje kao $N_k(x) = \{x' : \{i_1, i_2, \dots, i_k\} \subset \{1, 2, \dots, n\} x'_{i_j} = 1 - x_{i_j}\}$.

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 vrijednostima 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 procedure razmrdavanja, naš algoritam bi generisao jedno novo rješenje (označimo ga sa x'), koje se od trenutno najboljeg rješenja razlikuje u tačno k koordinata,

odnosno u tačno k proizvoda. Na primjer, 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 (ispaprano) 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. Na primjer, jedno lokalno pretraživanje bi moglo da bude zasnovano na tzv. *1-swap* pristupu (razmjena 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 (u idealnom slučaju u $O(1)$ vremenu), 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. Istraži na internetu problem vraćanja kusura pomoću minimalnog broja novčanica (kovanicu). Koristi google pretragu *change-making problem* ili *coin change problem*. Napiši pohlepni algoritam koji rješava ovaj problem. Ispitaj pod kojim uslovima pohlepni algoritam uvijek daje optimalno rješenje. Pronađi kontraprimjer kada pohlepni algoritam neće dati optimalno rješenje.
3. Bojenje grafa je procedura koja svakom čvoru grafa dodjeljuje jednu boju, poštujući pravilo da susjedni čvorovi nisu obojeni istom bojom. Napiši pohlepni algoritam pomoću kog se vrši bojenje grafa, tako da se iskoristi što je manje moguće boja. Pokušaj da okarakterišeš neku klasu grafova za koju tvoj pohlepni algoritam uvijek daje optimalno

rješenje. Konstruiši kontraprimjer kada pohlepni algoritam neće dati optimalno rješenje.

4. Osmisli pohlepni algoritam koji rješava problem trgovčkog putnika. Pronađi kontraprimjer kada taj algoritam ne pronalazi optimalno rješenje.
5. Osmisliti pohlepni algoritam koji rješava problem određivanja maksimalnog nezavisnog skupa čvorova u grafu. Odredi neke klase grafova za koje tvoj pohlepni algoritam uvijek pronalazi optimalno rješenje. Pronađi kontraprimjer kada pohlepni algoritam ne pronalazi optimalno rješenje.
6. 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.
7. Zadat 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.
8. 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 . Pretpostavka je da je $w_i \leq k$. Napisati pohlepni algoritam koji rješava ovaj zadatak.
9. Napisati pohlepni algoritam koji ispituje da li je jedna riječ podriječ drugе riječi.
10. 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.
11. Konstruisati primjer za 0-1 problem ruksaka, koji se ne može do optimalnosti rješiti pohlepnim algoritmom, koji se koristi za rješavanje frakcionog problema ruksaka.
12. Problem postavljanja kraljica na šahovsku tablu podrazumijeva pozicioniranje kraljica tako da se nikije dvije kraljice ne napadaju. Osmisliti genetski algoritam koji rješava ovaj problem.
13. Osmisliti genetski algoritam i metodu promjenljivih okolina kojima se rješava problem bojenja čvorova grafa.
14. Osmisliti genetski algoritam i metodu promjenljivih okolina kojima se rješava problem određivanja maksimalne klike u grafu.

-
15. Osmisliti genetski algoritam i metodu promjenljivih okolina kojima se rješava problem određivanja maksimalnog nezavisnog skupa čvorova u grafu.
16. 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. Pretpostavimo da ne postoji proizvod čija je cijena veća od cijene optimalnog rješenja. Konstruisati 2-aproksimativni algoritam za ovaj problem pod prethodnom pretpostavkom.
17. Pokazali smo da postoji 2-aproksimativni algoritam za problem pokrivanja čvorova. Da li se može konstruisati instanca na kojoj (jedan od dva aproksimativna algoritma koja su obrađena u prethodnoj glavi za ovaj problem) ne daje bolju aproksimaciju od faktora 2?
18. Neka je dat kompletan nedirektni graf $G = (V, E)$ u kojem težine grana zadovoljavaju nejednakost trougla i neka je k pozitivan prirodan broj. Probem se sastoji u particionisanju skupa V na k disjunktnih skupova V_1, \dots, V_k da bi se minimizirala grana najveće cijene između dva čvora u istom skupu, simbolički zapisano sa
- $$\max_{1 \leq i \leq k, u, v \in V_i} c(uv).$$
- Konstruisati 2-aproksimativni algoritam za ovaj problem (metričko k -klasterovanje).
19. Neka je dat netežinski 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.
20. U problemu bojenja grafa, u ulazu nam je dat neusmjeren graf $G = (V, E)$. Kažemo da je funkcija $f : V \mapsto \{1, \dots, k\}$ k -bojenje akko ne postoje susjedni čvorovi grafa kojima je pridružen isti broj, tj. $f(u) \neq f(v)$, za $uv \in E$. Problem bojenja grafa traži za k -bojenje na grafu G , gdje je k minimalan. Konstruisati pohlepni algoritam za koji je $k = \Delta + 1$, gdje je Δ maksimalni stepen čvora u G .
21. 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 \mathbb{R}^2 . Problem traži da se nađe (geometrijski) obilazak minimalne dužine koji sadrži sve segmente iz skupa S . Konstruisati 3-aproksimativni algoritam.

-
22. 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.
23. Neka je dat usmjereni graf $G = (V, E)$ sa nenegativnim težinama grana $w : E \mapsto \mathbb{R}$. Zadatak je naći podskup čvorova $S \subseteq V$ tako da je suma grana iz S u $V \setminus S$ najveća moguća, tj.

$$\sum_{s \in S, t \notin S, st \in E} w(st)$$

je maksimizovana.

Konstruisati $\frac{1}{4}$ -aproksimativni algoritam.

24. 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 $i = 1, \dots, n$. Problem se svodi na dodijelu zadataka 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 zadataka paralelno, tj. 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 uvodnoj sekciji, 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 uprave koja naručuje sistem, 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, među ostalima.

U nastavku opisujemo korištenje CPLEX i PuLP rješavača za implementaciju modela i njihovog rješavanja.

10.1 Optimizacioni rješavač CPLEX

CPLEX alat je generalni rješavač za rješavanje optimizacionih problema, a među njima izdvajamo klase problema LP, ILP i MILP-a. Konstruisan je za komercijalnu upotrebu od strane kompanije IBM. Različite verzije ovog programa, uključujući i besplatnu verziju koja sadrži određena ograničenja u mogućnostima, mogu se preuzeti preko linka koji je dat u referenci [24].

IBM ILOG CPLEX Optimization Studio uključuje i druge opcije poput *Constraint Programming* (CP) solver.

U nastavku opisujemo proces same implementacije modela u CPLEX-u kroz programski jezik C++ u Linux okruženju. Da bismo uključili CPLEX u C++ kod, nepohodno je koristiti sljedeći kod:

```
#include<ilcplex/ilocplex.h>
ILOSTLBEGIN
```

ili umjesto prve linije uključiti (pomoću #include) ilocplex.h fajl sa apsolutnom putanjom do njega.

Da bi se inicijalizovao LP model, neophodno je prvo inicijalizovati okruženje za modelovanje. Dakle, konstruišemo objekat klase *IloEnv* sa:

```
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 parameter dok je model koji se rješava pomoću CPLEX-a instanca klase *IloCplex* koji uzima instancu klase *IloModel* kao vrijednost svog parametra. Ako bismo željeli da riješimo model pomoću CP solvera, mjesto *IloCplex* bismo pozvali klasu *IloCP*.

```
IloModel model(env); //def. modela
// potrebno deklarisati ciljnu funkciju, varijable i ogranicenja
IloCplex cplex(model);
if(!cplex.solve()){ // poziv cplex solver-a
    env.error() << " Problem nije optimizovana." << endl;
    throw(-1);
}
//cplex zavrsio sa optimizovanjem problema ... ispis rjesenja
```

Sljedeći korak je implementacija varijabli, funkcije cilja i ograničenja u modelu.

Implementacija varijabli u modelu. Ako želimo da deklarisemo (realnu) varijablu x gdje je, na primjer, $0 \leq x \leq 40$, koristimo klasu *IloNumVar*, što odgovara sljedećoj liniji koda:

```
IloNumVar x = IloNumVar(env, 0.0, 40.0);
```

Ova definicija je ekvivalentna dodavanjem **ILOFLOAT** konstante kao četvrti argument u prethodnom konstruktoru. Ako je potrebno implementirati varijablu za koju je $x \geq 0$ to radimo kao u prethodnom primjeru uz korištenje *IloInfinity* konstante.

Ako želimo da implementiramo varijablu koja je cjelobrojna, recimo binarna varijabla, imamo sljedeći kod:

```
IloNumVar x = IloNumVar(env, 0.0, 1.0, ILOINT);
```

Ako je potrebno definisati vektor promjenjivih $x = (x_1, \dots, x_n)$, gdje je, recimo, $x_i \geq 1$, onda se definiše objekat klase *IloNumVarArray*, tj.:

```
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 templejt klasu **IloArray** <> na sljedeći način:

```
typedef IloArray<IloNumVarArray> FloatMatrix; // x_ij
typedef IloArray<FloatMatrix> Float3Matrix; // 3D niz var: x_ijk

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 deklarisanje izraza *expr* možemo koristiti operator “ $+=$ ”, gdje se trenutni izraz sabira sa nekim objektom *IloNumVar* klase – dakle varijabom (eventualno pomnoženom sa nekim koeficijentom). Recimo, ako želimo da implementiramo funkciju cilja $\sum_{i=1}^n c_i x_i$, implementacija bi bila ovakva:

```

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*, tj. kao *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:

$$s_j = \sum_{t=1}^n t \cdot x_{jt}, \forall j \in \{1, \dots, N\}.$$

Sljedeća implementacija odgovara ovim ograničenjima:

```

// neka je prethodno definisan i popunjeno niz s[]
// neka su prethodno deklarisane varajible x_ij
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*. Recimo da je potrebno dodati ograničenja $x_i + x_j \leq 1$ za sve $i < j, i, j \in [n]$ u *IloCplex* model

```

IloRangeArray constr(env);
// neka su prethodno definisane varijable 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 eksport rješenja (statistike) iz modela. 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.

```
// prethodno: kreiranje modela i poziv metoda solve()
env.out << cplex.getStatus() << endl;
IloNumArray xval(env); // ovdje će 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 << endl;
// vraca gap (ako nije rjeseno do optimalnosti, onda > 0)
```

- Metod `getObjValue`: vraća optimalnu vrijednost funkcije cilja (ili najbolju vrijednost nađenu u toku unaprijed definisanog intervala vremena).
- Metod `getStatus`: vraća status rješenja koje je CPLEX našao u zadanim vremenima (dopustivo, optimalno, nedopustivo itd.).
- Metod `getValues(vals, var)`:
 - `vals` – numerički niz (najčešće instanca klase `IloNumArray`) u koju će se kopirati krajinja rješenja koje je CPLEX našao;
 - `var` – varijable 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 promjenjivih;
 - `getMIPRelativeGap()`: vraća vrijednost relativne udaljenosti (realna vrijednost iz intervala $[0, 1]$) između najboljeg dopustivog rješenja (nađenog 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).

Eksportovanje modela za eventualno ponovno korištenje se može dobiti pozivom metoda `exportModel`, čiji je očekivani argument naziv fajla u koji će se model upisati (.lp ekstenzija).

Podešavanje parametara rješavača CPLEX. Podešavanje limita za vrijeme 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 manuala čiji je link dat među referencama na kraju knjige.

10.2 PuLP alat

PuLP je biblioteka visokog nivoa za implementaciju modela i njihovo rješavanje, izgrađena u programskom jeziku Python. Ona omogućava korisniku da kreira programe koristeći izraze prirodne za ovaj programski jezik. Biblioteka PuLP je fokusirana na podršku modelovanju i rješavanju LP i MILP modela. Interesantno je da PuLP ne zavisi od drugih softverskih paketa. Podržava veći broj komercijalnih i nekomercijalnih rješavača, te može biti proširena da podržava i dodatne rješavače.

PuLP radi na principu modularnog pristupa prevođenja modela napisanog prema definisanim pravilima na njegovu vektorsku prezentaciju (koja koristi vektore, rijetke matrice itd.), nakon čega je tako konvertovani model proslijeđen u interfejs klase nekog od rješavača. Kako je interfejs mnogih rješavača sličan, ili se sa njima rukuje tako što se model napiše u standardnim *.lp* ili *.mps* formatima datoteka, generičke klase rješavača su uključene u PuLP biblioteku, pored njenih specifičnih interfejsa. Ove generičke klase se, uz minimalan trud, mogu proširiti od strane korisnika koji žele da koristi nove rješavače.

Da bismo koristili PuLP biblioteku, treba da je importujemo 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 kodom:

```
prob = LpProblem("Naziv problema", LpMinimize)
```

Ovdje znači da je riječ o modelu čija se funkcija cilja minimizuje. Ako bi bilo potrebno da maksimizujemo, drugi argument bi tada bio *LpMaximize*.

Implementacija varijabli u modelu. Za inicijalizaciju varijabli koristi se klasa *LpVariable*. Ona ima četiri atributa: prvi je naziv varijable, drugi označava donju vrijednost varijable, treći označava gornju granicu vrijednosti varijable, dok četvrti označava tip varijable. Tip varijable, koji je

se može odabratи je `LpContinuous` ili `LpInteger`, označavajući neprekidnu (realnu) i diskretnu varijablu, redom. Ako želimo deklarisati varijablu $x_1 \geq 0$, koja je neprekidna, to radimo sljedećom linijom koda:

```
x1 = LpVariable( "x1", 0, None, LpContinuous)
```

Implementacija funkcije cilja u modelu. Uz pomoć definisanih varijabli, koristeći operatore iz skupa $\{+, -, \cdot\}$, dodajemo takvu funkciju na instancu problema (koristeći operator `+=`). Primjera radi, ako želimo da modelujemo funkciju cilja $x_1 + x_2$, gdje su obje varijable neprekidne i pozitivne, to bi 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 `+=`. Implementacija ograničenja je bliska ograničenjima u modelu, dakle prvo se konstruišu izrazi koji su vezani nekim operatorom poređenja. Primjera radi, ako bi željeli dodati ograničenja $x_1 + 2x_2 \leq 2$ i $x_1 - x_2 \geq 0$ u prethodni model, imamo sljedeći kod:

```
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 fajla.

Metode za eksport rješenja (statistike) iz modela. Rješavanje modela se inicira pozivom funkcije `solve()` na instancu modela. Kriterijumi za zauzstavljanje izvršavanja rješavača postavljamo kao parametar metoda `solve()` uz pomoć sljedeće linije koda:

```
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 po defaultu *COIN Branch and Cut* (CBC) rješavač izabran za rješavanje u PuLP modulu. Više o podešavanju solvera pročitati na linku [19]. Čitanje svih varijabli iz modela se postiže pomoću funkcije `variables()`. Svaka varijabla za attribute ima svoj naziv (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 lokacija iz Sekcije 3.3 uz pomoć PuLP rješavača. Model ovog problema je dat sa:

$$\begin{aligned}
& \min \sum_{i=1}^n f_i x_i + \sum_{i,j} c_{ij} y_{ij} d_j \\
& s.t. \\
& \sum_i y_{ij} = 1, \forall j \in \{1, \dots, m\} \\
& \sum_j d_j y_{ij} \leq u_i x_i, \forall i \in \{1, \dots, n\} \\
& x_i \in \{0, 1\}, \forall i \in \{1, \dots, n\} \\
& y_{ij} \geq 0, \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\}.
\end{aligned}$$

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 fajla).

```

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)
# varijable:
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 varijabli $x = (1, 0, 0)$, te $y_{11} = y_{12} = y_{13} = 1$, dok su ostale vrijednosti 0.

Preporučujemo da na linku <https://www.programcreek.com/python/example/96845/pulp.LpProblem> pronađete još nekoliko problema, koji su riješeni uz pomoć PuLp modula.

10.4 Primjena CPLEX rješavača na problem stringova

U ovoj sekciji modelujemo problem *najdužeg zajedničkog podniza bez ponavljanja*. Ovaj problem kao ulaz prihvata dva (proizvoljna) stringa s_1 i s_2 nad proizvoljnom abecedom, a kao izlaz vraća string s maksimalne dužine sa sljedećim karakteristikama:

- s je zajednički podniz oba stringa;
- s ne sadrži karakter koji se u njemu pojavljuje dva ili više puta.

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.

Za primjer, neka su data dva stringa $s_1 = \text{abdcccab}$ 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 "neponavljućim" 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 za ovaj problem.

Nazovimo par (a, b) , $1 \leq i \leq |s_1|$, $1 \leq j \leq |s_2|$ mečing akko je $s_1[i] = s_2[j]$, $a < b$. Dalje, sa \mathcal{M} označimo skup svih mečinga za dati ulaz $S = \{s_1, s_2\}$. Takođe, označimo sa $\mathcal{M}_a \subseteq \mathcal{M}$ skup svih mečinga koji pokrivaju isto slovo, tj. $\mathcal{M}_a = \{(i, j) \mid s_1[i] = s_2[j] = a\}$, $a \in \Sigma$. Za svaki mečing $p = (p_1(a), p_2(a))$, pridružimo jednu binarnu varijablu $z_p \in \{0, 1\}$. 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 varijabli sa $Z = \bigcup_{a \in \Sigma} Z_a$.

Za svake dvije binarne varijable 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

$$\sum_{z \in Z_a} z \leq 1, \forall a \in \Sigma \tag{10.1}$$

Takođe, s obzirom da rješenje problema odgovara nalasku skupa (međusobno) nekonfliktnih mečinga (odgovarajućih z -varijabli) 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 $z \in \{0, 1\}, z \in Z$, dobili smo 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 . kao što i prepostavljate, ovo je moguće riješiti i na drugačiji način, bez korištenja konverzije stringa.

Model je implementiran u programskom jeziku C++, a riješen uz pomoć CPLEX rješavača.

```
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <limits>
#include <unordered_map>
#include <map>

#include "path-to-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 //hashing the points
    {
        // works if the length of strings <= 5000 (can be adapted)
        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);
            // defining the set of binary variables Z
            unordered_map<Point2d, IloNumVar, Hash2d> Z;
            for (int i = 0; i < s1.size(); ++i){
                for (int j = 0; 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); // define objective function
        ++vars_num;
    }
}
}

map<int, IloExpr> cs;
// repetition-free constraints:
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);

// define a set of constraints:
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 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 finds optimal" << endl;
    else
        cout << "CPLEX finds feasible solution" << endl;

    double lastVal = double(cplex.getObjValue());
    // print the objective point
    cout << "components in the solution: {" << 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 << "value: " << lastVal << endl;
cout << "dual bound: " << double(cplex.getBestObjValue()) << endl;
catch(IloException& e) {
    cerr << " ERROR: " << e << endl;
}
env.end();
}
}

```

10.5 Zadaci

1. Istraži na internetu dostupne optimizacione rješavače za probleme linearne, cjelobrojne (i mješovite cjelobrojno-linearnog) programiranja. Procijeni njihov kvalitet, upotrebnu vrijednost i prepoznaj njihove prednosti i nedostatke u svakog od njih u odnosu na ostale.
2. Koristeći neki od rješavača, provjeri ispravnost rješenja zadataka iz Glave 4 i 6.
3. Neka je u ulazu dat graf $G = (V, E)$ kao instanca problema. Implementiraj rješavač za ovaj problem.

tirati sljedeći ILP model:

$$\begin{aligned} & \max \sum_{v \in V} x_v \\ & s.t. \\ & x_{v'} + x_{v''} \leq 1, \forall (v'v'') \in E \\ & x_v \in \{0, 1\}, \forall v \in V. \end{aligned}$$

Izbaciti rješenje problema u obliku skupa odabranih čvorova koji su komponeneta (optimalnog) rješenja. Riješiti relaksaciju datog ILP-a. Zaokružiti dobijeno rješenje, tj. transformisati ga u najbliže cjelobrojno rješenje (gledući po koordinatama). Model implementirati:

(a) uz pomoć CPLEX-a; (b) uz pomoć PuLP modula.

4. Neka je dat usmjerenim težinskim grafom $G = (V_1 \cup V_2, E)$ kao instanca problema modelovan sljedećim modelom, 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:

$$\begin{aligned} & \min \sum_{j \in V_2} \sum_{i \in V_1} h_i d_{i,j} Y_{i,j} \\ & s.t. \\ & \sum_{j \in V_2} Y_{i,j} = 1, \forall i \in V_1 \\ & Y_{i,j} - X_j \leq 0, \forall i \in V_1, \forall j \in V_2 \\ & \sum_{i \in V_1} X_i = p \\ & X_j \in \{0, 1\}, \forall j \in V_2 \\ & Y_{i,j} \in \{0, 1\}, \forall i \in V_1, \forall j \in V_2. \end{aligned}$$

Model implementirati:

(a) uz pomoć CPLEX-a; (b) uz pomoć PuLP modula.

5. Neka je u 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 vrijeđanje 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 ćete odrediti startna vremena izvršenja s_j ($s_j \geq r_j \wedge s_j + p_j \leq d_j$) za sve $j \in V$. Ovaj problem je modelovan

iz pomoć sljedećeg modela:

$$\begin{aligned}
& \max D - R \\
& s.t. \\
& R \geq r_i, \forall i \in V \\
& D \leq d_i, \forall i \in V \\
& R \geq r_i + \sum_{\{j \in V \setminus \{t\} | r_j \geq r_i\}} p_j x_j^+, \forall i \in V \\
& D \leq d_i - \sum_{\{j \in V \setminus \{t\} | d_j \leq d_i\}} p_j x_j^-, \forall i \in V \\
& x_j^+ + x_j^- = 1, \forall i \in V \\
& x_j^+, x_j^- \in \{0, 1\}, \forall i \in V \\
& D, R \in \mathbb{Z}.
\end{aligned}$$

Model implementirati:

(a) uz pomoć CPLEX-a; (b) uz pomoć PuLP modula.

6. 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:

$$\begin{aligned}
& \min t \\
& s.t. \\
& -0.5 \cdot t + \sum_{i=1}^n w_{i,j} x_i \leq 0.5 \cdot s_j, j = 1, \dots, p \\
& 0.5 \cdot t + \sum_{i=1}^n w_{i,j} x_i \geq 0.5 \cdot s_j, j = 1, \dots, p \\
& x_j \in \{0, 1\}, j = 1, \dots, n.
\end{aligned}$$

Model implementirati:

(a) uz pomoć CPLEX-a; (b) uz pomoć PuLP modula.

7. Neka je data instanca koja se sastoji od grafa $G = (V, E)$. Neka je dat ILP model:

$$\begin{aligned}
& \min \sum_{i \in V} x_i + \sum_{i \in V} y_i \\
& s.t. \\
& x_i + \sum_{j \in N_i} y_j \geq 1, i \in V \\
& y_i \leq x_i, i \in V \\
& x_i, y_i \in \{0, 1\}, i \in V,
\end{aligned}$$

gdje N_i označava susjede čvora i u grafu G .

Model implementirati:

(a) uz pomoć CPLEX-a; (b) uz pomoć PuLP modula.

Bibliografija

- [1] <https://towardsdatascience.com/the-big-picture-of-operations-research-8652d5153aad>
- [2] Hillier, Frederick S. Introduction to operations research. Tata McGraw-Hill Education, 2012.
- [3] <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 O'Sullivan, 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] Rahmaniani, 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] Hansen, Pierre, Nenad Mladenović, and José A. Moreno Pérez. "Variable neighbourhood search: methods and applications." 4OR 6.4 (2008): 319–360.
- [16] <https://www.tu-chemnitz.de/mathematik/discrete/manuals/cplex/doc/>
- [17] https://coin-or.github.io/pulp/main/the\optimisation_process.html
- [18] <https://www.coin-or.org/PuLP/CaseStudies/index.html>
- [19] https://coin-or.github.io/pulp/guides/how_to_configure_solvers.html
- [20] <https://www.youtube.com/watch?v=wxzONJvKZNM>
- [21] <https://www.youtube.com/watch?v=IposxYVBUnY>
- [22] <http://motor.ece.iit.edu/ms/benders.pdf>
- [23] <https://home.cs.colorado.edu/\~srirams/courses/csci5654-fall13/ilpLectures.pdf>
- [24] <https://www.ibm.com/products/ilog-cplex-optimization-studio>
- [25] <https://www.maths.ed.ac.uk/~gondzio/parallel/solver.html>
- [26] <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.