Egzaktno rješavanje problema cjelobrojnog linearnog programiranja

Križaić, Nikolina

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:597108

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Nikolina Križaić

EGZAKTNO RJEŠAVANJE PROBLEMA CJELOBROJNOG LINEARNOG PROGRAMIRANJA

Diplomski rad

Voditelj rada:
prof. dr. sc. Robert Manger

Zagreb, studeni, 2023.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Tijekom prve polovice 20. stoljeća formuliraju se na temelju praktičnih problema najpoznatiji problemi cjelobrojnog linearnog programiranja (CLP). Riječ je o problemima optimizacije gdje su i funkcija cilja i ograničenja linearni, a barem neke od varijabli odlučivanja moraju imati cjelobrojne vrijednosti. Problemi cjelobrojnog linearnog programiranja javljaju se u mnogim industrijama, kao što su logistika, proizvodnja, telekomunikacije. ekonomija, zdravstvo. U ovom radu proučavamo najpoznatije probleme CLP, kao što su problem naprtnjače, problem dodjeljivanja te problem trgovačkog putnika. Također se bavimo algoritmima za rješavanje problema CLP te korištenjem odgovarajućih softverskih paketa poput Gurobi

Problem naprtnjače (*engl. Knapsack problem*) formulira se kao pitanje kako odabrati određene predmete iz skupa predmeta tako da se maksimizira ukupna vrijednost tih predmeta, uzimajući u obzir ograničen kapacitet naprtnjače. U ekonomiji problem naprtnjače se može razmatrati na način da pojedinac ili firma koja treba donijeti odluku o tome koje investicije ili projekte da odabere kako bi maksimizirala očekivani profit, s obzirom na ograničene resurse kao što su budžet ili vrijeme.

Problem dodjeljivanja (*eng. assignment problem*) najčešće se formulira kao problema u kojem je cilj pridružiti određene resurse određenim zadacima na način koji minimizira ili maksimizira određenu funkciju troška ili dobiti. Zamislimo da se na fakultetu održavaju četri različita seminara, dani su nam podaci o tome koliki broj studenata može pohađati određeni seminar na određeni dan u tjednu. Svaki takav seminar treba održati jednom tjedno te u jednom danu smije biti samo jedan seminar. Cilj je pridružiti svakom seminaru određeni dan u tjednu te je to potrebno napraviti tako da čim više studenata može pohađati seminare.

Problem trgovačkog putnika (*eng. Traveling Salesman Problem*) formulira se kao problem gdje trgovac ili putnik treba obići niz gradova i to tako da minimizira ukupnu duljinu puta ili troškove putovanja. U industriji se problem trgovačkog putnika može primjeniti kod problema bušenja elektroničkih pločica. Rupe mogu biti različitih veličina. Za bušenje

dvije rupe različitih promjera, glava stroja mora se pomaknuti u kutiju s alatom i promijeniti svrdlo, što oduzima dosta vremena. Stoga je jasno da se prvo izabere promjer, izbuše se sve rupe istog promjera te se zatim odabere novo svrdlo. Stoga se ovaj problem bušenja može promatrati kao niz problema trgovačkog putnika, jedan za svaki promjer rupe, gdje su "gradovi" početni položaj i skup svih rupa koje se trebaju izbušiti s jednim svrdlom bušilice. "Udaljenost" između dva grada dana je s vremenom koje je potrebno za pomak glave stroja iz jednog položaja u drugi. Cilj je minimizirati vrijeme putovanja za glavu stroja.

Ovaj rad podijeljen je u sedam poglavlja. U prvom poglavlju ćemo navesti neke osnovne pojmove teorije grafova koje ce nam biti potrebne za razmatranje problema. U drugom poglavlju ćemo opisati osnovni problem naprtnjače te neke verzije toga problema. U trećem poglavlju opisat ćemo problem dodjeljivanja. U četvrtom poglavlju ćemo opisati osnovni problem trgovačkog putnika te različite formulacije toga problema, a navest ćemo i neke složenije verzije toga problema. U petom poglavlju objasnit ćemo *branch-and-bound* metodu za problem CLP-a. U šestom poglavlju upoznat ćemo se s načinom na koji se gradi model te rješava problem cjelobrojnog linearnog programiranja u softveru Gurobi. Dok ćemo u sedmom poglavlju implementirati rješenje za problem trgovačkog putnika koristeći Gurobi te ćemo navesti neke eksperimentalne rezultate.

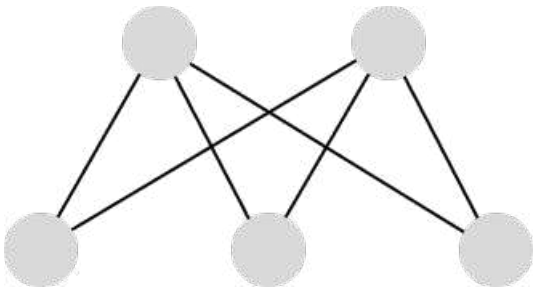
Poglavlje 1

Osnovni pojmovi iz teorije grafova

Niže ćemo navesti pojmove neophodne za razumijevanje osnova teorije grafova koji su potrebni za daljnje praćenje problema opisanih u ovom diplomskom radu. Sve definicije navedene u ovom poglavlju su prema Veljanu [15].

Definicija 1.0.1. ***Graf*** *G je uređeni par $G = (V, E)$. $V = V(G)$ je neprazan konačan skup, čiji su elementi* ***vrhovi*** *od G. $E = E(G)$ je skup* ***bridova*** *disjunktnih s V, a svaki brid $e \in E$ spaja dva vrha $u,\ v \in V$ koji se zovu* ***krajevi*** *od e.*

Definicija 1.0.2. *Graf G je* ***bipartitan graf*** *ako mu se skup vrhova može particionirat u dva skupa X i Y tako da svaki brid ima jedan kraj u X, a drugi u Y. Particija $(X,\ Y)$ zove se* ***biparticija*** *grafa.*



Slika 1.1: Primjer bipartitnog grafa $K_{2,3}$

Definicija 1.0.3. *Brid čiji se krajevi podudaraju zove se* ***petlja****, a ako su krajevi različiti -* ***pravi brid*** *ili* ***karika****. Graf G je* ***jednostavan*** *ako nema ni petlja ni višestrukih bridova.*

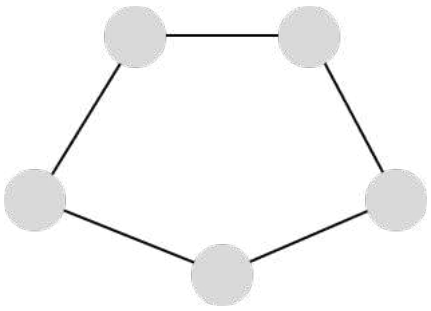
Definicija 1.0.4. ***Šetnja*** *u grafu G je niz $W = v_0e_1v_1e_2...e_kv_k$ čiji članovi su naizmjence vrhovi v_i i bridovi e_i, tako da su krajevi od e_i vrhovi v_{i-1} i v_i, $1 \leq i \leq k$. Dio šetnje $W = v_0e_1v_1e_2...e_kv_k$ je podniz $v_ie_{i+1}v_{i+1}...e_jv_j$ susjedni članovi od W. To se još zove i $(v_i,\ v_j)$-**dio** od W. Šetnja W je **zatvorena** ako je $v_0 = v_k$.*

Definicija 1.0.5. *Ako su svi bridovi $e_1, e_2, ..., e_k$ šetnje W međusobno različiti, onda se put u W zove **staza**, a ako su na stazi i svi vrhovi $v_0, ..., v_k$ međusobno različiti, ona se zove **put**.*

Definicija 1.0.6. *Dva vrha $u,\ v$ grafa G su **povezana**, ako postoji (u, v)-put u G. Graf je **povezan** ako su svaka dva njegova vrha povezana nekim putom.*

Definicija 1.0.7. ***Potpun graf*** *je jednostavan graf u kojem je svaki par vrhova spojen bridom.*

Definicija 1.0.8. ***Ciklus*** *C_n na n vrhova definiramo skupom vrhova $V = 1, 2, ..., n$ i skupom bridova $E = \{\{i,\ i + 1\}|i < n\} \cup \{1,\ n\}$.*



Slika 1.2: Primjer C_6 ciklusa

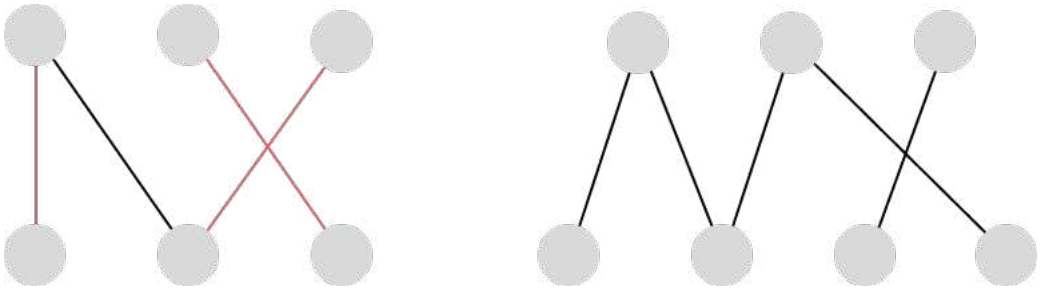
Definicija 1.0.9. ***Hamiltonov put*** *je put koji prolazi kroz sve vrhove grafa. Ukoliko je Hamiltonov put zatvoren, govorimo o **Hamiltonovom ciklusu**.*

U brojnim nam primjenama učestalo trebaju i dodatne strukture na grafu.

Definicija 1.0.10. *Svakom bridu $e \in E(G)$ pridružujemo njegovu **težinu** $w(e)$. Dakle, w je funkcija $w : E(G) \to A$, gdje je $A = \mathbb{R}$ ili $\mathbb{R}_+$ ili $\mathbb{Z}_m$ itd. Uređeni par (G, w) grafa G i težinske funkcije w zove se **težinski graf**.*

Definicija 1.0.11. ***Sparivanje*** *u grafu $G = (V, E)$ je podskup $M \subseteq E$ bridova koji su karike, a nikoja dva nisu susjedna, tj. nemaju zajednički vrh. Kažemo da su dva kraja brida u M **sparena** u M. Sparivanje u M-**zasićuje** vrh v, ili se kaže da je v M-**zasićen**, ako je neki brid iz M incidentan s v, a inače je v M-**nezasićen**.*

Definicija 1.0.12. *Sparivanje zovemo* ***savršenim sparivanjem*** *ako je svaki vrh iz G M-zasićen.*



Slika 1.3: Lijevo je primjer grafa u kojem postoji savšeno sparivanje, označeno crvenom bojom, a desno je primjer gdje ne postoji savršeno sparivanje, općenito vrijedi da grafovi s neparnim brojem vrhova ne mogu imat savršeno sparivanje

Poglavlje 2

Problem naprtnjače

2.1 Problem 0/1 naprtnjače

Najčešći primjer koji se veže uz ovaj problem je problem u kojem lopov ima jednu naprtnjaču te na odabir ima više predmeta koji želi ukrasti, ali nema dovoljno prostora u naprtnjači. Stoga, je potrebno izabrati predmete tako da oni stanu u naprtnjaču, a da vrijednost predmeta bude što veća.

Problem naprtnjače je probem iz područja kombinatorno optimizacijske matematike te se njime bave znanstvenici od početka prošlog stoljeća.

Matematički definiran problem prema Kellerer, Pferschy i Pisinger [4] . Neka je $N = \{1, 2, ..., n\}$ dani skup predmeta, vrijedi da je $n \in \mathbb{N}$, gdje predmet i ima težinu w_i i vrijednost (profit) p_i i neka je c kapacitet dane naprtnjače, gdje je c pozitivan realan broj. Klasični 0-1 problem naprtnjače se može definirati kao problem u kojem je potrebno izraz 2.1 maksimizirat:

$$z = \sum_{i=1}^{n} p_i x_i \tag{2.1}$$

pri čemu je

$$x_i = \begin{cases} 1 & \text{kada smo predmet } \ i \text{ stavili u naprtnjaču} \\ 0 & \text{inače} \end{cases} \tag{2.2}$$

te postoji ograničenje 2.3

$$\sum_{i=1}^{n} w_i x_i \leq c \tag{2.3}$$

Uz pretpostavku da su c, w_i te p_i za sve $i \in \{1, 2, ...\}$ pozitivni cijeli brojevi.

2.2 Inačice problema naprtnjače

Postoje i druge varijante problema naprtnjače osim *problema naprtnjače 0-1*, u svima njima postoje drugačija ograničenja u izboru količine predmeta koji se mogu staviti u naprtnjaču. Niže navedeni problemi opisali su Kellerer, Pferschy, Pisinger [4].

Ograničeni problem naprtnjače je problem gdje količina pojedinog predmeta kojeg možemo staviti u naprtnjaču može biti više od jednog, ali ujedno treba biti ograničeno mnogo. Matematički definirajmo problem. Varijable p_i, w_i i c su definirane na isti način kao i u 2.1, dakle c, w_i te p_i za sve $i \in \{1, 2, ...\}$ su pozitivni cijeli brojevi te je potrebno dodatno uvesti niz pozitivnih cijelih brojeva $b_1, b_2, \ldots, b_n$ koji predstavljaju maksimalnu količinu određenog predmeta. Potrebno je riješiti sljedeći izraz

$$\max \sum_{i=1}^{n} p_i x_i \tag{2.4}$$

te pri tome treba vrijediti

$$\sum_{i=1}^{n} w_i x_i \leq c \tag{2.5}$$

$$0 \leq x_i \leq b_i, \quad x_i \in \mathbb{Z}, \quad \forall i = 1, 2, \ldots, n \tag{2.6}$$

Uvjet 2.6 se odnosi na ograničenje da pojedini predmet se može više od jednom uzet, ali ne neograničeno puta. Sljedeći problem govori upravo o tome kada na rasplolaganjuimamo beskonačno mnogo svakog predmeta.

Neograničeni problem naprtnjače bavi se problemom kada su svi predmeti dostupni u neograničenim količinama. U probelemu je potrebno maksimizirati izraz 3.1 uz ograničenja

$$\sum_{i=1}^{n} w_i x_i \leq c \tag{2.7}$$

$$x_i \geq 0, \quad x_i \in \mathbb{Z}, \quad \forall i = 1, 2, \ldots, n \tag{2.8}$$

Sljedeća inačica problema odnosi se na problem kada je iz grupe predmeta potrebno odabrati točno jedan predmet, takav problem naziva se *problem naprtnjače višestrukog izbora* . Neka su $N_1, N_2, \ldots, N_m$ disjunktni skupovi predmeta te uvedimo matricu $X = \left(x_{ij}\right)$ pri čemu $x_{ij} = 1$ označava da smo odabrali j-ti predmet iz N_i-tog skupa predmeta. Također, p_{ij} predstavlja vrijednost j-tog predmeta iz N_i-tog skupa predmeta, a w_{ij} predstavlja težinu j-tog predmeta iz N_i-tog skupa predmeta. Potrebno je riješiti sljedeći izraz:

$$\max \sum_{i=1}^{m} \sum_{j \in N_i} p_{ij} x_{ij} \tag{2.9}$$

uz ograničenja

$$\sum_{i=1}^{m} \sum_{j \in N_i} w_{ij} x_{ij} \leq c \tag{2.10}$$

$$\sum_{j \in N_i} x_{ij} = 1 \quad (i = 1, 2, \ldots, m) \tag{2.11}$$

$$x_{ij} \in \{0, 1\} \quad (i = 1, 2, \ldots, m, \quad j \in N_i) \tag{2.12}$$

U mnogim primjenama prirodno se pojavljuje da vrijednost svih stvari u naprtnjači ovisi i o međusobnom utjecaju pojedinim predmeta jedan na drugi. Takav problem naziva se *kvadratni problem naprtnjače* (*eng. Quadratic knapsack problem*). Takav tip problema javlja se u telekomunikaciji, u situaciji kada je potrebno odabrati mjesta na kojima treba sagraditi satelitske stanice, takav da promet bude maksimalan, a potrošnja sredstva za izgradnju stanica je ograničena. Ovo je kvadratni problem naprtnjače jer jedna pozicija satelitske stanice prima signale na određenom području, stoga izgradnja jedne stanice utječe na izgradnju stanica na tom području.
Matematički zapišimo problem, neka je $N = \{1, 2, ..., n\}$ dani skup predmeta, vrijedi da je $n \in \mathbb{N}$, gdje predmet i ima težinu w_i, gdje je w_i pozitivan cijeli broj za svaki $i = 1, ..., n$ te neka je c kapacitet dane naprtnjače, gdje je c pozitivan realan broj. Definiramo, $n \times n$ matricu s prirodnim brojevima $P = \left(p_{ij}\right)$, gdje p_{ii} predstavlja vrijednost koji se ostvaruje, ako izaberemo predmet i, a $p_{ij} + p_{ji}$ je dodatna vrijednost koja se ostvaruje u slučaju da izaberemo predmet i i predmet j tj. izabirom predmeta i i j se ostvaruje vrijednost $p_{ii} + p_{jj} + p_{ij} + p_{ji}$. Potrebno je riješiti sljedeći izraz:

$$\max \sum_{i \in N} \sum_{j \in N} p_{ij} x_i x_j \tag{2.13}$$

te pri tome vrijedi

$$\sum_{i \in N} w_i x_i \leq c \tag{2.14}$$

$$x_i \in \{0, 1\}, \ j \in N \tag{2.15}$$

Uočimo da je matrica P simetrična, dakle treba vrijediti $p_{ij} = p_{ji}$ za sve $i, j \in N$. Navedeni problem je kompleksniji problem od drugih ovdje navedenih jer za razliku od ostalih verzija problema gdje je funkcija cilja linearna, ovdje je funkcija cilja kvadratična. Stoga ovaj problem spada u cjelobrojne kvadratične probleme.

Poglavlje 3

Problem dodjeljivanja

3.1 Linearan problem dodjeljivanja

Tipičan problem linearnog dodjeljivanja je pitanje kako dodijeliti n poslova radnicima, tako da jedan posao obavlja jedan radnik za što će on biti plaćen. Neka je c_{ij} cijena rada j-tog radnika da obavi i-ti posao. Problem glasi: kako rasporediti n poslova među radnicima tako da cijena obavljanja poslova bude najniža moguća. Matematički formuliran problem prema Conforti, Cornuejols, Zambelli [7]. Potrebno je riješit sljedeći izraz:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{3.1}$$

pri čemu je

$$x_{ij} = \begin{cases} 1 & \text{ako smo } i\text{-ti posao dodijelili } j\text{-tom radniku} \\ 0 & \text{inače} \end{cases} \tag{3.2}$$

iz same prirode problema slijedi da vrijede sljedeće jednakosti

$$\sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1, 2, ..., n) \tag{3.3}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1, 2, ..., n) \tag{3.4}$$

Jednakost 3.3 govori da jedan posao radi točno jedan radnik, a jednakost 3.4 govori da jedan radnik radi točno jedan posao. Jednakosti 3.2-3.4 nazivaju se ograničenja dodjeljivanja. Moguće je definirati problem dodjeljivanja pomoću znanja iz teorije grafova

prema Burkard, Dell'Amico i S. Martello [10]. Neka je $G = (U, V; E)$ bipartitan graf za koji vrijedi da su skupovi U i V n-člani, gdje je svaki vrh iz U povezan s točno jednim vrhom iz V te svaki brid $e \in E$ ima nenegativnu težinu. Vrhove iz skupa U možemo poistovjetiti s poslovima, a vrhove iz skupa V s radnicima. Neka je $(i, j) \in E$ tada težina tog brida odgovara vrijednosti c_{ij}. Tada prvotni problem linearnog dodjeljivanja postaje problem savršenog sparivanja u bipartitnom grafu tako da podgraf određen sparivanjem ima minimalnu težinu.

3.2 Kvadratni problem dodjeljivanja

U praksi se često pojavljuju problemi s funkcijom cilja koja je kvadratna kod problema dodjeljivanja. Takav problem se naziva kvadratni problem dodjeljivanja prema Burkard, Dell'Amico i S. Martello [10]. Kvadratni problem dodjeljivanja uveo je Koopmans i Beckmanna 1957. [13] za potrebe modeliranja problema smještaja djelatnosti na najpovoljnije lokacije koji je opisan u nastavku.

Primjer 3.2.1. *Trebamo premjestiti n djelatnosti na n lokacija. Dane su tri matrice $n \times n$:*

$A = (a_{ik})$, *gdje a_{ik} predstavlja mjeru koliko djelatnost i i k surađuju*

$B = \left(b_{jl}\right)$, *gdje b_{jl} predstavlja udaljenost između lokacije j i l*

$C = \left(c_{ij}\right)$, *gdje c_{ij} predstavlja trošak smještanja djelatnosti i na lokaciju j.*

Pretpostavljamo da ukupan trošak ovisi o umnošku mjere suradnje među djelatnostima, udaljenosti lokacija i trošku selidbe određene djelatnosti. Svaki produkt $a_{ik}b_{\varphi(i)\varphi(k)}$ reprezentira suradnju među djelatnostima i i k pomnoženo s udaljenosti između dodijeljene lokacije djelatnosti i koju označavamo s $\varphi(i)$ i dodijeljene lokacije djelatnosti k koju označavmo s $\varphi(k)$. Cilj je dodijeliti svakoj djelatnosti lokaciju tako da troškovi budu minimalni.

Prema Çela [16] funkcija cilja u **??** *može se zapisati kao*

$$\min_{\varphi \in \mathcal{S}_n} \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{\varphi(i)\varphi(k)} + \sum_{i=1}^{n} c_{i\varphi(i)} \tag{3.5}$$

gdje je $\mathcal{S}_n$ skup svih permutacija od $1, 2, ..., n$. Svaki pojedinačni produkt $a_{ik}b_{\varphi(i)\varphi(k)}$ je trošak transporta koji nastaje zbog suradnje djelatnosti i na lokaciji $\varphi(i)$ te djelatnosti k na lokaciji $\varphi(k)$. Dakle svaki izraz $\sum_{k=1}^{n} a_{ik}b_{\varphi(i)\varphi(k)} + c_{i\varphi(i)}$ predstavlja ukupan trošak, koji se pripisuje djelatnosti i. Navedeni trošak uključuje trošak montaže djelatnosti i na lokaciju $\varphi(i)$ te troškove transporta za sve djelatnosti k, ako se montiraju na lokacije $\varphi(1)$, $\varphi(2), ..., \varphi(n)$.

Iako, navedeni problem izlazi iz okvira teme diplomskog rada, naveden je jer problem opisan u sljedećem poglavlju se može tumačiti kao kvadratni problem dodjeljivanja, međutim u ovom radu ćemo tumačiti problem trgovačkog putnika kao linearni problem.

Poglavlje 4

Problem trgovačkog putnika

4.1 Opis problema trgovačkog putika

Jedan od najistaknutijih problema kombinatorne optimizacije je problem trgovačkog putnika. Naziv je 30-tih godina 20. stoljeća smislio američki matematičar H. Whitney, naime problem se može tumačiti kako umanjiti cjelokupnu razdaljinu koju trgovački putnik prođe da bi obišao svaki od n datih gradova isključivo jednom i vratio se u početni grad, prema V. Bosančić, A. Golemac [14]. Možemo modelirati problem trgovačkog putnika kao funkciju cilja 3.5. Međutim, kao i u potpoglavlju 3.1 problem se intuitivnije može predočiti koristeći pojmove iz teorije grafova.
Prema R. Manger [8] problem koristeći pojmove iz teorije grafova glasi ovako. Zadan je potpuni neusmjereni graf čijim bridovima su pridružene težine. Potrebno je naći Hamiltonov ciklus s minimalnim zbrojem težina bridova.

Prema T. C. Hu, A. B. Kahn [12] problem se može modelirati na sljedeći način. Dano je n gradova koje treba posjetiti trgovački putnik uz (simetričnu) udaljenost c_{i_j} među parom gradova i i j ($1 \leq i,\ i \leq j$), trgovački putnik treba posjetiti sve grdove točno jednom i vratiti se u grad iz kojeg je krenuo pri tom treba naći najkraći mogući put. Problem je cjelobrojne prirode jer trebamo odlučiti koji brid između dva grada sudjeluje u rješenju, a koji ne. Upravo zato definiramo binarnu varijeblu x_{i_j} kao:

$$x_{ij} = \begin{cases} 1 & \text{ako put ide direktno iz grada } i \text{ u grad } j \\ 0 & \text{inače} \end{cases} \tag{4.1}$$

Za $i = 1, 2, ..., n$ uvodimo varijablu t_i, za koju vrijedi $u_i = k$, ako je grad i posjećen u k-tom koraku ($k = 1, 2, ..., n$). Zatim se problem trgovačkog putnika može tumačiti kao problem cjelobrojnog linearnog programiranja zadan kao:

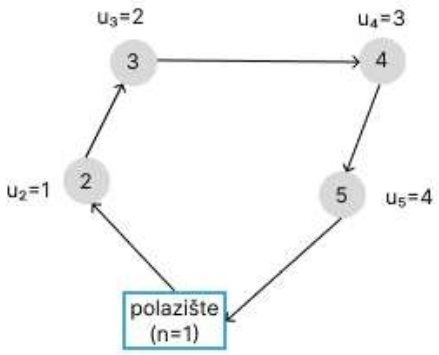
$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} \tag{4.2}$$

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \qquad j = 1, \cdots, n \tag{4.3}$$

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \qquad i = 1, \cdots, n \tag{4.4}$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad 1 \leq i \neq j \leq n \tag{4.5}$$

Izraz 4.2 govori da je potrebno naći rutu s najmanjom mogućom udaljenosti. Jednadžba 4.3 govori da je potrebno stići u svaki grad samo jednom, a jednadžba 4.4 da se iz svakog grada odlazi samo jednom. Nejednadžba 4.5 odnosi se na to da nisu dozvoljeni podture, odnosno ciklusi s manje od n vrhova. Opisan problem se naziva **Miller–Tucker–Zemlinova formulacija**.
Uvijet 4.5 se dobije na način da proučavama više mogućih slučaja. Ako trgovački putnik ide direktno iz grada i u grad j tada je $x_{ij} = 1$ te vrijedi $u_j = u_i + 1$. U slučaju da ne postoji brid $(i,\ j)$, odnosno vrijedi $x_{ij} = 0$, slijedi da vrijedi $u_i + 1 \leq u_j + M(1 - x_{ij})$ gdje je M dovoljno velik, treba vrijedit $u_i \leq M$. Dakle, dobar odabir za $M = n$ čime se dobiva izraz 4.5.



Slika 4.1: Primjer modeliranja problema trgovačkog putnika putem Miller–Tucker–Zemlinove formulacije

Navest ćemo još jednu formulaciju problema trgovačkog putnika, koja koristi podskupove vrhova da bi eliminirala moguću pojavu podtura u grafu (primjer 4.1). Formulacija se naziva **Dantzig–Fulkerson–Johnson** te glasi ovako:

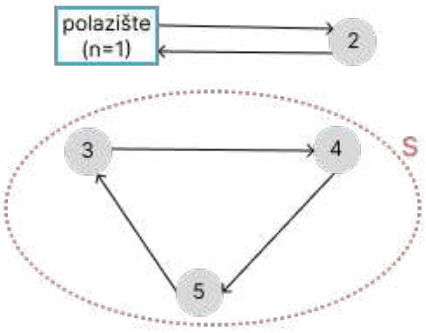
$$\min \sum_{i=1}^{n} \sum_{j\neq i, j=1}^{n} c_{ij} x_{ij} \tag{4.6}$$

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1 \qquad j = 1, \cdots, n \tag{4.7}$$

$$\sum_{j=1, j\neq i}^{n} x_{ij} = 1 \qquad i = 1, \cdots, n \tag{4.8}$$

$$\sum_{i\in S} \sum_{j\neq i, j\in S} x_{ij} \leq |S|-1 \qquad \forall S \subsetneq \{1, ..., n\},\ |S| \geq 2 \tag{4.9}$$

Dakle, obje formulacije su vrlo slične razlikuju se u tome na koji način se eliminiraju pojave podtura u grafu. Zahtjev 4.9 govori o tome da, ako uzmemo neki podskup vrhova koji ima barem dva člana da broj bridova među vrhovima iz podskupa ne smije biti veći od broja članova u podskupu umanjenog za jedan. Takav zahtjev treba vrijediti za sve prave podskupove vrhova od barem dva člana. Stoga, slika 4.1 ne zadovoljava Dantzig–Fulkerson–Johnson formulaciju jer kad za podskup vrhova uzmemo $S = \{3, 4, 5\}$ jer kad uvrstimo u izraz 4.9 dobivamo kontradikciju.



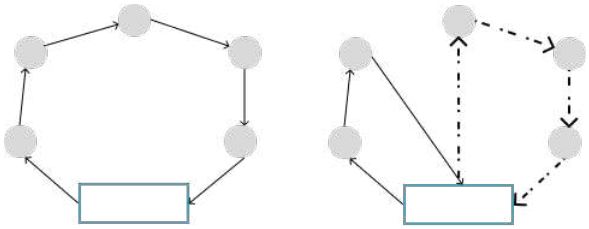
Slika 4.2: Primjer grafa koji nije dopušten zbog uvjeta 4.5 u formulaciji Miller–Tucker–Zemlin, odnosno 4.9 u formulaciji Dantzig–Fulkerson–Johnson te je uznačen podskup vrhova S

4.2 Inačice problema trgovačkog putnika

Kao i problem naprtnjače i problem trgovačkog putnika ima više verzija. Prema Greco [2] postoje tri verzije problema trgovačkog putnika:

1. **Simetrični problem trgovačkog putnika**
   Temeljni oblik problema, ranije naveden opis problema.

2. **Asimetrični problem trgovačkog putnika** Definiran je na usmjerenom grafu $G = (U, V)$ s matricom $A = (a_{i_k})$ koja sadrži udaljenosti medu gradovima, primijetimo da matrica A nije simetrična. Odnosno, bar jedan brid $\{i, j\} \in V(G)$ ima težinu brida različite vrijednosti ovisno o smjeru kretanja. Govoreći o problemu trgovačkog putnika kao o problemu obilaska gradova, to bih značilo da bar jedan par gradova ima drugačije vrijednosti troška putovanja ovisno o smjeru obilaska. Trošak putovanja se može tumačiti kao vrijeme putovanja, duljina puta, cijenu prijevoza itd.



Slika 4.3: S lijeve strane je primjer rješenja u simetričnom problemu trgovačkog putnika, s desne strane je primjer rješenja za višestruki problem trgovačkog putnika opisan pod a) za $m = 2$

3. **Višestruki problem trgovačkog putnika** Ovoj vrsti problema je zajedničko to da postoji više trgovačkih putnika. Postoji nekoliko varijanti takvog problema:

   a) m trgovačkih putnika započinje putovanje iz istog grada, svaki grad se treba posjetiti samo jednom. Potrebno je utvrditi obilaske za svih m trgovačkih putnika tako da trošak bude minimalan.

   b) Postoji više od jednog polaznih gradova. Odnosno, ne postoji jedno polazište za sve trgovačke putnike. Nakon svog putovanja svaki trgivački putnik vraća se u grad iz kojeg je krenuo ili u jedan od polaznih gradova, pri tome je važno da broj trgovačkih putnika na svakom polazištu bude jednak kao i na početku putovanja.

c) Postoji *m* trgovačkih putnika na raspolaganju, ali ne trebaju svi sudjelovati u putovanju. Potrebno je izabrati koji će trgovački putnici sudjelovati u putovanju tako da se minimalizira trošak. Obično kod ovakvog tipa problema postoji neki fiksni trošak koji je potrebno platit trgovačkom putniku da krene na putovanje.

d) Ponekad neke gradove treba posjetiti u točno određenom vremenskom periodu, stoga je potrebno da ga točno tada posjeti jedan od *m* trgovačkih putnika.

Naravno, moguće je uvoditi i dodatna ograničenja te kombinirati neke od gore navedenih problema.

Poglavlje 5

Egzaktni algoritmi za rješavanje problema CLP

Egzaktna metoda koja se prva nameće je pronalaženje svih mogućih rješenja te odabir onog optimalnog. Međutim, takve metode iziskuju previše računalnog vremena. Složenost za takve algoritme obično raste brže nego bilo koja eksponencijalna funkcija, npr. kao faktorijela s obzirom na neki parametar koji opisuje veličinu primjerka problema. Stoga, kad govorimo o egzaktnim metodama one su temeljene na tome da ne tražimo baš sva moguća rješenja već unaprijed odbacujemo ona za koja smo sigurni da ne vode do boljeg rješenja. Upravo na tome se temelji algoritam koji ćemo proučavati u ovom poglavlju.

5.1 *Branch-and-bound* metoda

Prema M. J. Brusco, S. Stahl [9] *branch-and-bound* pristup problemu optimizacije osmislili su neovisno A. H. Land i A. G. Doig (1960.) te K. G. Murty, C. Karel i J. D. C. Little (1962.). *Branch-and-bound* algoritam temelji na pretraživanju dijela prostora rješenja unutar kojeg se sigurno nalazi optimalno rješenje. Stoga, se kaže da su dopustiva rješenja nabrojana implicitno. Kao što samo ime algoritma govori, algoritam je temeljen na dva fundamentalna principa: grananju i ograničavanju. Pretpostavimo, da želimo riješiti sljedeći optimizacijski problem:

$$\max_{x \in S} f(x) \tag{5.1}$$

U ovom poglavlju ćemo koristiti maksimizacijski problem kako bi objasnili princip rada *branch-and-bound* algoritma, međutim veza između minimizacijski i maksimizacijskih problema je sljedeća: $\max_{x \in S} f(x) = -\min_{x \in S}(-f(x))$. Stoga, sve niže navedeno se odnosi i na probleme cjelobrojne linearne minimizacije.

Neka je S konačan prostor rješenja. Dio algoritma koji se usredotočuje na grananje (*eng. branching*), odnosi se na to da se podskup prostora rješenja $S' \subseteq S$ podijeli u manje podskupove $S_1, ..., S_m$. Ti podskupovi mogu se poklapati, ali u uniji trebaju tvoriti cijeli moguć prostor rješenja S', odnosno $S' = S_1 \cup S_2 \cup ... \cup S_m$. Proces se ponavlja tako dugo dok svaki podskup ne sadrži samo jedno moguće rješenje. Odabirom najboljeg od svih razmatranih rješenja, zajamčeno smo pronašli globalni optimum za navedeni problem na S'. Dio algoritma koji se odnosi na ograničavanje (*eng. bounding*), svodi se na postupak traženja gornje (*eng. upper bound*) i donje ograde (*eng. lower bound*) za moguć prostor rješenja S'. Donja ograda z^l odabire se kao najbolje trenutno nađeno rješenje. Gornja ograda z^u za dani prostor rješenja $S' \subseteq S$ zadovoljava sljedeće:

$$z^u \geq f(x), \quad \forall x \in S'. \tag{5.2}$$

Gornja granica se koristi za smanjenje dijelova prostora za pretraživanje. Pretpostavimo da vrijedi $z^u \leq z^l$ za dan podskup S'. Iz 5.2 dobivamo

$$f(x) \leq z^u \leq z^l, \quad \forall x \in S'. \tag{5.3}$$

što znači da bolje rješenje od z^l se ne može naći u S' stoga više ne trebamo pretraživat prostor rješenja S'. Iako se gornja ograda može računati kao $z^u = \max_{x \in S} f(x)$, često je takav pristup računski preskup. Umjesto toga, može se proširiti prostor rješenja ili modificirati funkcija cilja tako da se time dobiveni problem može učinkovito riješiti. Proširen prostor rješenja $Y \supseteq S'$ dobiva se popuštanjem nekih ograničenja, ako je Y prikladno odabran račun $z^u = \max_{x \in Y} f(x)$ se može bitno jednostavnije izvesti, a lako se vidi da je izvedena vrijednost doista gornja granica. Drugi pristup je razmatranje neke druge ciljne funkcije $g(x)$ koja zadovoljava $g(x) \geq f(x)$ za sve $x \in S'$. Lako se provjeri da zaista $z^u = \max_{x \in S'} g(x)$ daje valjanu gornju granicu. Za uspjeh ove metode bitno ovisi što ranije otkrivanje zadovoljavajuće donju ogradu. Algoritam *branch-and-bound* često se prikazuje kao stablo odluke i tada to stablo nazivamo ***branch-and-bound* stablo**, stoga ćemo u nastavku rada često koristiti termine stabla kao strukture podataka.

Standardni problem cjelobrojnog linearnog programiranja ima sljedeći oblik:

$$z_{\text{IP}} = \max\{cx : x \in S_{IP}\}, \quad \text{gdje } S_{IP} = \{x \in \mathbb{Z}_+^n : Ax \leqslant b\} \tag{5.4}$$

Uvjet $x \in Z_+^n$ se zove cjelobrojno ograničenje. Gornji problem bez cjelobrojnog ograničenja nazivamo relaksirani problem za zadani problem CLP. Taj relaksirani problem je problem (običnog) linearnog programiranja. linearnog programiranja. U čvoru *i branch-and-bound* stabla rješava se sljedeći relaksirani problem linearnog programiranja.

$$z_{\text{LP}}^i = \max\{cx : x \in S_{LP}^i\}, \quad \text{gdje } S_{LP}^i = \left\{x \in \mathbb{R}_+^n : A^i x \leqslant b^i\right\} \tag{5.5}$$

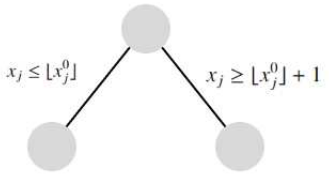
Ako u 5.5 pri i-tom čvoru nađemo do sada optimalnije rješenje, tada obilježimo pronađeno rješenje s x^i. (J. Lee [6])
Uvjeti za rezanje grane, odnosno daljnje ne grananje u i-tom čvoru su sljedeći:

1. *(nije dopustiv)* $S^i_{LP} = \emptyset$
2. *(optimalnost)* $x^i \in \mathbb{Z}^n_+$
3. *(dominiranost)* $z^i_{\text{LP}} \leq z_{\text{IP}}$ gdje je z_{IP} vrijednost poznatog dopustivog rješenja za 5.4

S obzirom na to da rješavamo relaksirani problem linearnog programiranja, procesom grananja se dodaju linearna ograničenja na pojedine varijable. To radimo tako da je $S = S^1 \cup S^2$, gdje je $S^1 = S \cap \{x \in \mathbb{R}^n_+ : dx \leq d_0\}$ te $S^2 = S \cap \{x \in \mathbb{R}^n_+ : dx \geq d_0 + 1\}$, $(d, d_0) \in Z^{n+1}$. Ako je x^0 rješenje relaksiranog sljedećeg problema

$$z^0_{\text{LP}} = \max\{cx : x \in \mathbb{R}^n_+,\ Ax \leqslant b\} \tag{5.6}$$

možemo odabrati (d, d_0) tako da vrijedi $d_0 < dx^0 < d_0 + 1$. Vrlo je poželjno birati (d, d_0) na takav način jer će tada sigurno vrijediti $x^0 \notin S^1_{LP} \cup S^2_{LP}$ također moguće je da za $i = 1, 2$ dobijemo $z^i_{\text{LP}} = \max\{cx : x \in S^i_{LP}\} < z^0_{\text{LP}}$.
U praksi prema G. Nemhauser, L. A. Wolsey [1], najčešće stavimo da je (d, d_0) takav da je $d = e_j$ za neki $j \in N$, gdje je e_j jedinični vektor kanonske baze za $\mathbb{R}^n$ te $d_0 = \lfloor x_0 \rfloor$.



Slika 5.1: Najčešći oblik grananja u *branch-and-bound* algoritmu

Propozicija 5.1.1. *Ako je* $P = \{cx : x \in \mathbb{R}^n_+,\ Ax \leqslant b\}$ *omeđen, stablo rješenja bit će konačno pod uvjetom da na svakom čvoru i gdje* x^i_j *nije cijeli broj se vrši grananje po principu* $x_j \leq \lfloor x^0_j \rfloor,\ x_j \geq \lfloor x^0_j \rfloor + 1$. *Posebno, ako je* $\omega_j = \lceil \max\{x_j : x \in P\} \rceil$ *niti jedan put ne može sadržavati više od* $\sum_{j \in N} \omega_j$ *vrhova.*

Dokaz. Jednom kad uvedemo ograničenje $x_j \leq d$ za neki $d \in \{0, ..., \omega_j - 1\}$ drugo ograničenje koje se može naknadno pojaviti na putu od korijena do lista stabla je $x_j \leq d'$ za $d' \in \{0, ..., d - 1\}$ te $x_j \geq \bar{d}$ za $\bar{d} \in \{0, ..., d\}$. Slijedi da najveći broj ograničenja koja uključuju x_j dogodit će se dodavanjem $x_j \leq d$ za sve $d \in \{0, ..., \omega_j - 1\}$, ili $x_j \geq d$ za sve $d \in \{0, ..., \omega_j\}$,

ili $x_j \geq d$ za sve $d \in \{0, ..., \alpha\}$ te posljednje $x_j \leq d$ za sve $d \in \{\alpha, ..., \omega_j - 1\}$. U svakom od ovih slučajeva zahtijevamo ω_j ograničenja na x_j stoga $\sum_{j \in N} \omega_j$ je ukupan broj ograničenja na bilo kojem putu. □

Primjer 5.1.2. *Zadan je problem cjelobrojnog linearnog programiranja:*

$$\begin{aligned} &\max z = -x_1 + x_2 \\ &12x_1 + 11x_2 \leq 63 \\ &-22x_1 + 4x_2 \leq -33 \\ &x_1, x_2 \geq 0,\ x_1, x_2 \in \mathbb{Z} \end{aligned}$$

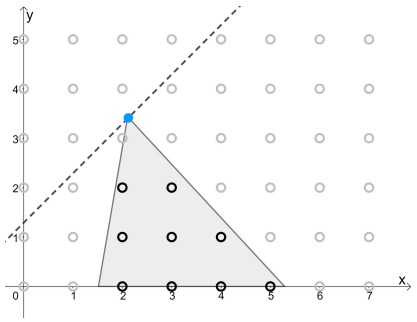
Riješimo relaksirani problem linearnog programiranja simplex metodom.

	x_1	x_2	
e_1	-12	-11	63
e_2	22	-4	-33
	-1	1	0

U dva koraka simplex algoritma dobivamo sljedeću simplex tablicu:

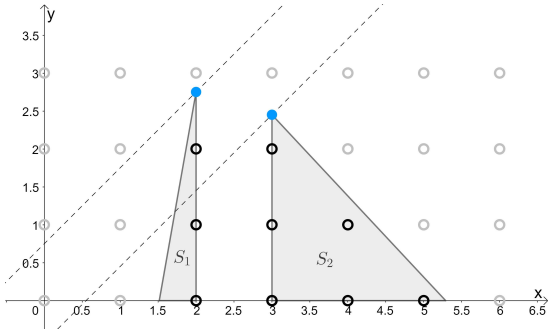
	e_1	e_2	
x_1	$\frac{-2}{145}$	$\frac{11}{290}$	$\frac{123}{58}$
x_2	$\frac{-11}{145}$	$\frac{-6}{145}$	$\frac{99}{29}$
	$\frac{-9}{145}$	$\frac{-23}{290}$	$\frac{75}{58}$

Rješenje relaksiranog problema je $z^* = \frac{75}{58} \approx 1.29,\ x_1^* = \frac{123}{58} \approx 2.12,\ x_2^* = \frac{99}{29} \approx 3.41.$



Slika 5.2: Prikaz prostora rješenja te optimalno rješenje za relaksirani problem

Očigledno je $x_1 = 2,\ x_2 = 0$ *moguće rješenje promatranog problema cjelobrojnog linearnog programiranja pa je* $z^l = -2 + 0 = -2$. *Kako je skup mogućih rješenja problema cjelobrojnog programiranja* S_{IP} *podskup skupa mogućih rješenja relaksiranog problema* S_{LP}, *na taj način smo dobili gornju ogradu* $z^u = 1.29$ *za funkciju cilja na* S_{IP}. *Varijabla* $x_1 = 2.12$ *je necjelobrojna pa ćemo pomoću nje izvršiti separaciju skupa mogućih rješenja* S_{IP}. *Dobivamo dva nova potproblema. Jedan potproblem se sastoji od početnog relaksiranog problema uz uvjet* $x_1 \leq 2$ *taj skup označimo s* S_1, *a drugi od početnog relaksiranog problema uz uvjet* $x_1 \geq 3$ *taj skup označimo s* S_2.



Slika 5.3: Prikaz prostora rješenja nakon prvog grananja, gdje je $S_1 = \{x_1 \leq 2, x_2 \geq 0 : 12x_1 + 11x_2 \leq 63,\ -22x_1 + 4x_2 \leq -33\}$ te $S_2 = \{x_1 \geq 3, x_2 \geq 0 : 12x_1 + 11x_2 \leq 63,\ -22x_1 + 4x_2 \leq -33\}$

Rješavanjem relaksiranih problema simplex algoritmom dobijemo sljedeće:

	z^*	x_1^*	x_2^*
$x_1 \leq 2$	0.75	2.00	2.75
$x_1 \geq 3$	-0.55	3.00	2.45

Vrijednosti trenutnih granica su $z^l = -2$ *te* $z^u = 0.75$. *U čvoru, gdje postoji uvjet* $x_1 \leq 2$ *vršimo grananje tako jedna grana uvjetuje* $x_2 \leq 2$ *predstavlja skup* S_3, *a druga* $x_2 \geq 3$ *predstavlja skup* S_4. *Rješavanjem relaksiranih problema simplex algoritmom uz uvjet* $x_2 \leq 2$ *dobivamo* $z^* = 0.14,\ x_1^* = 1.86,\ x_2^* = 2.00$. *Kao što je sa slike 5.3 vidljivo skup* S_1 *uz dodatan uvjet* $x_2 \geq 3$ *je prazan, stoga kažemo da je ta grana nije dopustiva. Vrijednosti trenutnih granica su* $z^l = -2$ *te* $z^u = 0.14$. *Aktivni potproblemi su sljedeći:* .

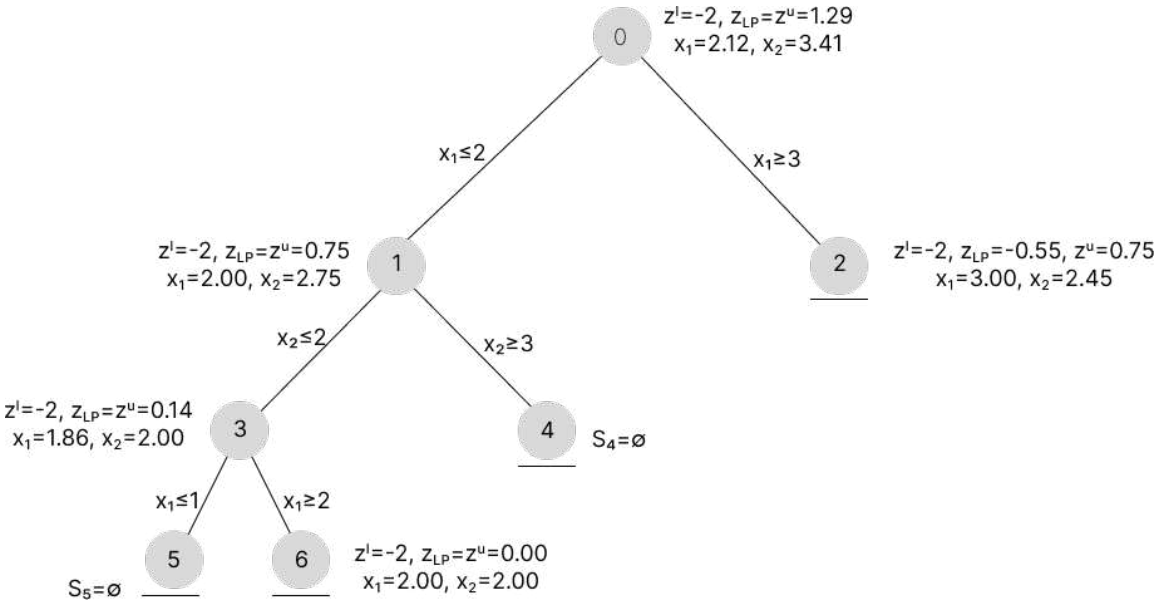
	z^*	x_1^*	x_2^*
$x_1 \geq 3$	-0.55	3.00	2.45
$x_1 \leq 2,\ x_2 \leq 2$	0.14	1.86	2.00

Prema tome potrebno je skup S_3 podijeliti na S_5 za kojeg vrijedi $x_1 \leq 1$ te na S_6 za kojeg vrijedi $x_2 \geq 2$. Primijetimo da je skup S_5 prazan, stoga ga isključujemo iz daljnjih razmatranja. Primijenimo simplex metodu za skup S_6, time dobivamo $z^* = 0.00$, $x_1^* = 2.00$, $x_2^* = 2.00$. Kako smo našli cjelobrojno rješenje $x_1 = 2$, $x_2 = 2$ na skupu S_6 te su svi drugi podskupovi isključeni, završavamo postupak.

Dan je popis $\mathcal{L}$ aktivnih potproblema ili, ekvivalentno, podstabla neobrezanih ili aktivnih čvorova, pitanje je odlučiti koji čvor treba detaljnije ispitati sljedeći. Postoje dvije su osnovne mogućnosti:

- *a priori pravilo* koje unaprijed određuju redoslijed kojim će se stablo granati odnosno rezati
- *adaptivno pravilo* koje bira čvorove koristeći informacije (npr. ograde) o statusu aktivnih čvorova

Metoda odabira a priori pravilo za biranje čvorova se temelji na dubinskom pretraživanju te na *backtracking* algoritmu, često se u literaturi opisuje i kao LIFO (*eng. last in, first out*) jer se temelji na tome da se čvorovi stavljaju na stog. Svaki korak algoritma uključuje uzimanje čvora s vrha stoga i provjeru predstavlja li taj čvor rješenje problema. Ako čvor predstavlja rješenje, poduzima se odgovarajuća akcija, na primjer, ispisivanje rješenja. Ako čvor nije rješenje, algoritam pokušava generirati njegovu djecu i dodaje ih na stog. Algoritam započinje s postavljanjem samo korijena stabla na stog, a završava kad pronađe rješenje ili kad se stog isprazni. Dubinsko pretraživanje čvorova odnosi se na to da, ako trenutni čvor nije odrezan, sljedeći čvor koji se razmatra je jedno od njegove dvoje djece. *Backtracking* znači da kada je čvor odrezan, vraćamo se stablom od čvora prema korijenu dok ne pronađemo prvi čvor (ako postoji) koji ima dijete koje još nije uzeto u razmatranje. Kombinacijom ova dva algoritma uz zadavanje redoslijeda kojim ćemo razmatrati djecu čvora, dobijemo *a priorno pravilo* jer unaprijed znamo izgled stabla. *A priori pravilo* se ne drži u potpunosti pretraživanja u dubinu jer, ako se nalazimo u nekom čvoru razmatraju se sva djeca tog čvora. Primjer korištenja *a priori pravila* za odabir čvorova uz pretpostavku da preferiramo lijevo dijete prikazano je na slici 5.4 koja prikazuje stablo iz primjera 5.1.2. Čvorovi su numerirani redoslijedom kojim se razmatraju. Ako je čvor podcrtan to znači da se on više ne grana zbog ograničenja.
Adaptivno pravilo temelji se na uspoređivanju donje, odnosno gornje ograde za aktivne čvorove. Kod ovog pravila važno je vršiti grananje u onim čvorovima koje smatramo vjerovatnijim za pronalaženjem konačnog rješenja. Dakle, kod problema maksimizacije uvijek ćemo preferirati čvor koji ima najveću gornju ogradu među svim aktivnim čvorovima. Primjer stabla koji za odabir čvorova koristi adaptivno pravilo je dano na slici 5.5.



Slika 5.4: Primjer korištenja *a priori* pravila za određivanje redoslijeda grananja u stablu iz primjera 5.1.2

Za kraj ovog potpoglavlja navedimo korake algoritma uz pretpostavku da se radi o optimizacijskom problemu maksimizacije te uz korištenje adaptivnog pravila.

1. korak: Određivanje donje ograde algoritma z^l. To se može postići, na primjer, određivanjem nekog dopustivog rješenja x, te stavimo $z^l = z(x)$. U situaciji kada nije dostupno valjano rješenje, može se pretpostaviti da je $z^l = -\infty$.

2. korak: Ovaj korak odnosi se na grananje skupova koji predstavljaju prostore rješenja. Na početku je potrebno podijeliti cijeli skup mogućih rješenja. Na primjer, ovu podjelu možemo postići tako da jedan dio bude podskup u kojem vrijedi $x_k \leq c$, dok je drugi podskup onaj u kojem vrijedi $x_k \geq c+1$, gdje je k odabrana varijabla te c cjelobrojna konstanta. Konstanta c se obično odabire na temelju rješenja varijable x_k^* relaksiranog problema, tako da vrijedi $c < x_k < c + 1$.

3. korak:Za svaki novi podskup, izračunajte gornju granicu z^u za maksimalnu vrijednost ciljne funkcije u tom podskupu.

4. korak:Moguće je isključiti neke od podskupova iz daljnjeg razmatranja ukoliko:

(i.) vrijedi $z^u < z^l$

(ii.) podskup nema mogućih rješenja

(iii.) nađeno je optimalnije cjelobrojno rješenje unutar tog podskupa, s vrijednošću ciljne funkcije z^*

U slučaju (iii.), ako je $z^* > z^l$, spremiti to rješenje i primijeniti test (i.) na sve preostale podskupove.

5. korak: Ako su svi podskupovi isključeni, zaustavite se. Tada, trenutno sačuvano rješenje je optimalno. U suprotnom, pređite na *korak 2*.

5.2 Rješavanje problema 0/1 naprtnjače

Prema L. Neralić, Z. Lukač [5], metoda *branch-and-bound* pokazala se posebno dobrim alatom za rješavanje problema cjelobrojnog programiranja s binarnim varijablama, koje mogu imati vrijednosti 0 ili 1. Među takve probleme svrstava se i problem 0/1 naprtnjače. Osnovni algoritam metode *branch-and-bound* sličan je onome opisanom u prethodnom odjeljku, ali se ovdje primjenjuje grananje na način da se za određenu varijablu x_k provodi podjela na dvije grane, jedna s vrijednošću $x_k = 0$ i druga s vrijednošću $x_k = 1$. U relaksiranom problemu, dodaju se ograničenja $0 \leq x_j \leq 1$ za sve binarne varijable x_j, gdje je $j = 1, 2, ..., n$. Redoslijed kojim ćemo izabirati čvorove temelji se na jednostavnoj ideji da preferiramo predmete koji donose najviše vrijednosti po jedinici težine i biramo ih dok ima dostupnog prostora u naprtnjači. Stoga je korisno analizirati predmete koji su sortirani i, prema potrebi, renumerirani tako da vrijedi

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq ... \geq \frac{p_n}{w_n} \tag{5.7}$$

Vrijeme potrebno za jednostavno sortiranje je $O(n \log n)$.

Primjer 5.2.1. *Neka je zadan problem 0/1 naprtnjače s pet predmeta s vrijednostima* $40, 42, 25, 12, 7$ *i težinama* $4, 7, 5, 3, 2$ *respektivno, uz nosivost naprtnjače od 13.*

Izračunajmo vrijednost pojedinog predmeta.

i	1	2	3	4	5
w_i	4	7	5	3	2
p_i	40	42	25	12	7
$\frac{w_i}{p_i}$	10	6	5	4	3.5

Sortirajmo predmete prema vrijednosti: $x_1 \geq x_2 \geq x_3 \geq x_4 \geq x_5$*, tim poretkom ćemo razmatrati stavljanje predmeta u naprtnjaču. Problem se može zapisati kao problem 0-1 linearnog programiranja:*

$$\begin{aligned} &\max z = 40x_1 + 42x_2 + 25x_3 + 12x_4 + 7x_5 \\ &\textit{uz ograničenja} \\ &4x_1 + 7x_2 + 5x_3 + 3x_4 + 2x_5 \leq 13 \\ &x_1, x_2, x_3, x_4, x_5 \in \{0, 1\} \end{aligned} \tag{5.8}$$

Očigledno je donja ograda funkcije cilja $z^l = 0$*, u slučaju prazne naprtnjače. Rješavamo relaksirani problem 5.8, gdje umjesto zadnjeg uvjeta o binarnosti varijabli, uvodimo uvjet* $0 \leq x_1, x_2, x_3, x_4, x_5 \leq 1$*. Koristeći simplex algoritam dobijemo rješenje* $x_1 = 1, x_2 = 1, x_3 = \frac{2}{5}, x_4 = x_5 = 0$*, dok je* $z^* = z^u = 92$ *gornja ograda na vrijednost funkcije cilja na skupu* S_{IP} *mogućih rješenja. Grananje provodimo tako da uzmemo najprije* $x_1 = 0$ *za jednu granu, te* $x_1 = 1$ *za drugu granu. Rješavanjem relaksiranih problema simplex algoritmom dobijemo sljedeće:*

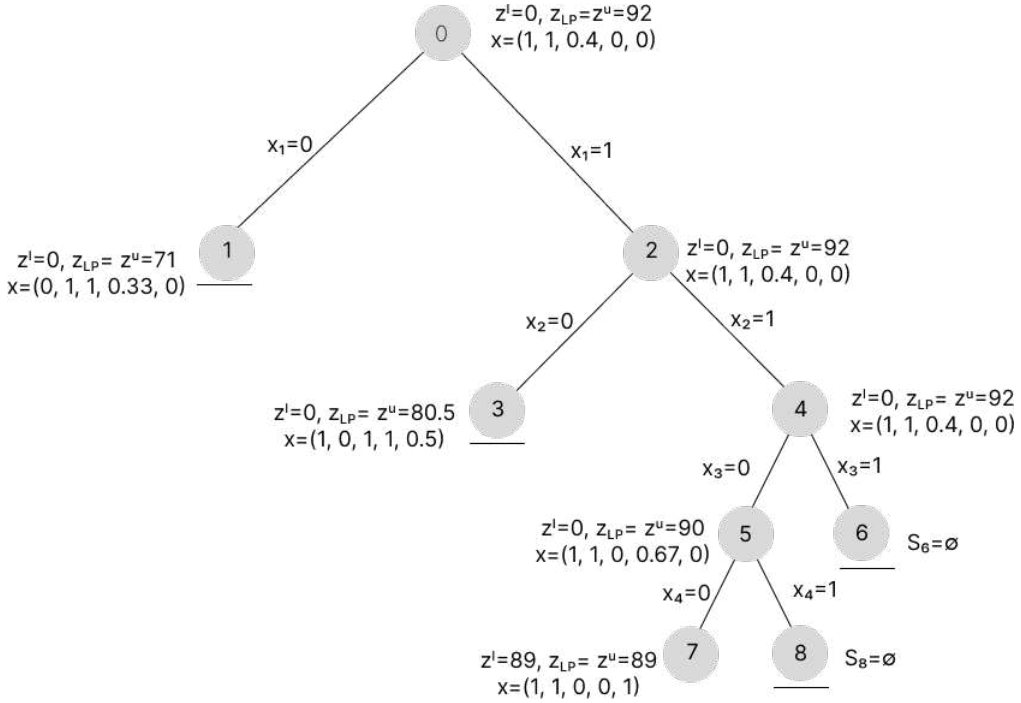
	z^*	x_1^*	x_2^*	x_3^*	x_4^*	x_5^*
$x_1 = 0$	71	0	1	1	$\frac{1}{3}$	0
$x_1 = 1$	92	1	1	$\frac{2}{5}$	0	0

U vrhu gdje je $x_1 = 0$ *gornja ograda je* $z^u = 71$*, a u vrhu gdje je* $x_1 = 1$ *gornja ograda je* $z^u = 92$*. Stoga, daljnje grananje nastavljamo u čvoru gdje je* $x_1 = 1$*. Pritom ćemo promatrati granu iz vrha* 2 *za koju je* $x_2 = 0$*, te dobiti vrh* 3 *i granu iz vrha* 2 *za koju je* $x_2 = 1$*, te dobiti vrh* 4*.Rješavanjem relaksiranih problema simplex algoritmom dobijemo sljedeće:*

	z^*	x_1^*	x_2^*	x_3^*	x_4^*	x_5^*
$x_1 = 1, x_2 = 0$	80.5	1	0	1	1	$\frac{1}{2}$
$x_1 = 1, x_2 = 1$	92	1	1	$\frac{2}{5}$	0	0

U vrhu 3 *gornja ograda je* $z^u = 80.5$*, a u vrhu* 4 *gornja ograda je* $z^u = 92$*.*
Stoga, daljnje grananje nastavljamo u vrhu 4*. Pritom ćemo promatrati granu iz vrha* 4 *za koju je* $x_3 = 0$*, te dobiti vrh* 5 *i granu iz vrha* 4 *za koju je* $x_3 = 1$*, te dobiti vrh* 6*. Primijetimo kako u vrhu* 6 *relaksirani problem nema mogućeg rješenja, stoga se ne provodi daljnje grananje iz vrha* 6*. Rješavanjem relaksiranog problema u vrhu* 5 *simplex algoritmom dobijemo sljedeće:* $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = \frac{2}{3}, x_5 = 0$*, te je* $z^* = z^u = 90$*.*
Daljnje, grananje provodimo iz čvore 5*, jedna grana sadrži uvjet* $x_4 = 0$ *te joj je pridružen*

vrh 7*, a druga* $x_4 = 1$ *te joj je pridružen vrh* 8*. Primijetimo da u vrhu* 8 *relaksirani problem nema mogućeg rješenja jer predmeti premašuju zadanu nosivost naprtnjače. Rješavanjem relaksiranog problema u vrhu* 7 *dobijemo rješenje:* $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0, x_5 = 1$ *te je* $z^* = z^u = 89$*.*
Jedini aktivni vrhovi su 1 *i* 3 *kako je donja ograda u oba čvora* 0*, a donja ograda u čvoru* 7 *je* 89 *stoga se podskupovi koje predstavljaju čvorovi* 1 *i* 3 *mogu isključiti iz daljnjeg razmatranja. Time smo sve podskupove isključili iz razmatranja te se algoritam zaustavlja. Rješenje je* $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0, x_5 = 1$ *s maksimalnom vrijednošću funkcije cilja* 89*. Dakle, potrebno je u naprtnjaču staviti prvi, drugi i peti predmet i pri tome ćemo ostvarit vrijednost* 89*.*



Slika 5.5: Stablo *branch-and-bound* za primjer 5.2.1

Poglavlje 6

Gurobi Optimization

Gurobi Optimization, često nazivan Gurobi, je vodeća komercijalna softverska biblioteka za rješavanje optimizacijskih problema. Gurobi omogućuje rješavanje linearnih, kvadratnih, kvadratno-ograničenih, mješovito cjelobrojnih linearnih, mješovito cjelobrojnih kvadratnih i mješovito cjelobrojno kvadratno-ograničenih optimizacijskih problema. Slika 6.1 prikazuje logo alata Gurobi optimization, slika preuzeta s [3].

Slika 6.1: Logo Gurobi Optimization

Gurobi se pokreće kroz neko programsko okruženje. Podržava pokretanje kroz sljedeće programske jezike: C, C++, Java, .NET, Python, MATLAB, R. Proizvođač preporučuje da se pokreće kroz Python te je najviše dokumentacije i primjera prilagođeno upravo tom programskom jeziku, vidi [3].

6.1 Gradnja modela u Gurobi-u

U ovom potpoglavlju objasnit ćemo sve naredbe koje ćemo koristiti u programu u poglavlju 7. Potrebno je zapisati početan problem i za to služi objekt iz klase Model u Gurobiu. Za kreiranje modela koristimo sljedeću naredbu:

```
Model(name="", env=defaultEnv)
```

Argumenti:

- **name**: naziv novog modela, naziv se bilježi kao ASCII string, stoga je potrebno pripaziti da se naziv modela može prikazati ASCII simbolima
- **env**: okruženje u kojem se stvara model

Povratna vrijednost:

- novi objekt model, model inicijalno ne sadrži varijable ili ograničenja

Fukcija kojom može mijenjati parametre modela je sljedeća:

```
setParam ( paramname, newvalue )
```

Argumenti:

- **paramname**: niz znakova (*eng. string*) koji sadrži naziv parametra koji želite izmijeniti, naziv može uključivati zamjenske znakove (*eng. wildcard characters*) poput '*', '?', '%', ako se podudara više od jednog parametra, navest će imena koja se podudaraju i nijedan odnjih neće promijeniti, mala i velika slova zanemaruje
- **newvalue**: nova vrijednost za parametar, može biti 'default', što znači da se parametar treba vratiti na zadanu vrijednost

Navest ćemo samo neke parametre koje ima smisla podešavati s obzirom na problem kojim se bavimo, lista svih parametra može se pronaći na [3].

- **TimeLimit** (def: Infinity) : postavlja maksimalno vrijeme u sekundama koje je dozvoljeno za rješavanje problema optimizacije
- **MIPGap** (def: 0.0001): postavlja dozvoljeno postotno odstupanje izračunatog rješenja od egzaktnog (zaista optimalnog) rješenja za MIP (*eng. Mixed-Integer Programming*) probleme
- **OutputFlag** (def: 1) : binaran parametar koji govori hoće li se ispisivati poruke tijekom rješavanja problema, 0 isključuje ispis, dok 1 omogućava ispis
- **Heuristics** (def: 0.05) : postavlja korištenje heuristika za pronalaženje boljih početnih rješenja, izraz u obliku postotka koji predstavlja postotak vremena izvođenja optimizacije koje se koristi za računanje heuristike
- **NodeLimit** (def: Infinity) : maksimalan broj čvorova u stablu za probleme CLP-a
- **Presolve** (def: -1) : parametar koji omogućava automatsko pojednostavljenje modela prije nego što se rješava, može se postaviti na 0 (isključeno), 1 (osnovni "presolve"), 2 (napredni "presolve"), -1 (odnosi se na početne postavke)

- **LazyConstraints** (def: 0) : binaran parametar koji govori da li se koriste uvjeti tipa lazy constraint (vidi potpoglavlje 6.3), 0 ne koriste se, dok 1 označava da se koriste

Naredbom Model samo stvaramo prazan model, potrebno je definirati koje ćemo varijable koristiti, to radimo naredbom Model.addVars().

```
addVars (*indices, lb=0.0, ub=float('inf'), obj=0.0,
         vtype=GRB.CONTINUOUS, name="" )
```

Argumenti:

- **indices**: indeksi za pristup novim varijablama
- **lb**: donja granica(e) za nove varijable
- **ub**: gornja granica(e) za nove varijable
- **obj**: koeficijenti funkcije cilja
- **vtype**: određuje tip varijabli, može poprimiti pet vrijednost
  1. GRB.CONTINUOUS : kontinuirane varijable
  2. GRB.BINARY : binarne varijable koje mogu imati vrijednost 0 ili 1
  3. GRB.INTEGER : cjelobrojne varijable
  4. GRB.SEMICONT : varijabla koja je u nekom rasponu cjelobrojna, a izvan tog raspona kontinuirana
  5. GRB.SEMIINT : varijabla koja može biti cijeli broj ili polovičan (npr. 0, 0.5, 1, 1.5 itd.)
- **name**: (neobavezno) naziv varijabli

Povratna vrijednost:

- **tupledict** : objekt sličan kao Pythonov dictionary

  ```
  tupledict (args, kwargs)
  ```

  - **args** : indeks varijable
  - **kwargs** : naziv varijable

Gornja naredba u sebi sadrži argument obj kojim je moguće mijenjati funkciju cilja međutim postoji i zasebna naredba koja služi tome da se namjesti funkcija cilja u modelu, ako koristimo obje mogućnosti funkcija cilja će imati vrijednosti koje su zadane sljedećom naredbom.

`setObjective ( expr, sense=GRB.MINIMIZE )`

Argumenti:

- **expr** : predstavlja ciljnu funkciju, izraz može sadržavati varijable koje se nalaze u modelu, konstante i matematičke operacije
- **sense** : parametar koji određuje da li se ciljna funkcija maksimizira ili minimizira, moguće vrijednosti su GRB.MAXIMIZE ili GRB.MINIMIZE

Povratna vrijednost: nema

U model se dodaju ograničenja pomoću sljedeće naredbe:

`addConstrs ( generator, name="" )`

Argumenti:

- **generator** : generator izraza, gdje svaka iteracija predstavlja jedno ograničenje
- **name** : naziv ograničenja

Povratna vrijednost:

- **tupledict** : objekt sličan kao Pythonov dictionary

  `tupledict (args, kwargs)`

  - **args** : indeks ograničenja
  - **kwargs** : generator izraza

Napomena 6.1.1. *Argument generator se piše kao u Pythonu generator izraza, to je značajka Python jezika koja omogućuje iteraciju kroz Python izraz. Nova Gurobi ograničenja dodaju se modelu za svaku iteraciju generatora izraza. Na primjer, ako je x Gurobi varijabla, tada naredba* `m.addConstr(x ≤ 1, name='c0')` *dodaje jedno linearno ograničenje. Dok naredba* `m.addConstrs((x[i] ≤ 1 for i in range(5)), name='c')` *modelu dodaje pet ograničenja. Prvo ograničenje koje proizlazi iz ovog izraza zvalo bi se c*[0] *i uključivalo bi varijablu x*[0].

Napomena 6.1.2. *Ograničenja u Gurobiu se mogu pisati samo pomoću jednog operatora usporedbe. Na primjer, za ograničenje* $0 \leq x \leq 1$ *potrebno je napisati dvije naredbe.*

Napomena 6.1.3. *Gurobi ne podržava striktne usporedbe manje od, veće od ili nejednako, zato da bi izbjegao potencijalne greške vezane uz numeričku toleranciju.*

Napomena 6.1.4. *Primijetimo da Gurobi slijedi objektno-orijentirani pristup u programiranju.*

6.2 Rješavanje problema u Gurobi-u

Algoritam koji se koristi u Gurobi-u za rješavanje problema cjelobrojnog linearnog programiranja je *branch-and-cut*. To je algoritam koji se temelji na algoritmu *branch-and-bound* te koristi algoritam *cutting planes* za vrednovanje i dodavanje ograničenja koja se dobiju rješavajući relaksirani problem linearnog programiranja.

Primjer 6.2.1 (Primjer korištenja *cutting planes* algoritma kod rješavanja problema CL-P-a)**.** *Pretpostavimo da problem cjelobrojnog linearnog programiranja ima slijeće ograničenje*

$$6x_1 + 5x_2 + 7x_3 + 4x_4 + 5x_5 \leq 15$$
$$x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}$$

Pretpostavimo da smo riješili neki relaksirani problem linearnog programiranja te smo dobili rješenje: $x_1 = 0, x_2 = 1, x_3 = x_4 = x_5 = \frac{3}{4}$*, primijetimo da nije moguće da* $x_3 = x_4 = x_5 = 1$ *jer* $5 + 7 + 4 = 16 > 15$*. Stoga u početna ograničenja dodajemo uvjet* $x_3 + x_4 + x_5 \leq 2$*. Nadalje, "režemo" dobiveno rješenje, odnosno više nećemo provoditi grananje u tom čvoru u sklopu algoritma branch-and-bound jer* $\frac{3}{4} + \frac{3}{4} + \frac{3}{4} = \frac{9}{4} > 2$*.*

Zbog toga što dodavanjem dodatnih ograničenja činimo relaksirani problem LP-a težim za rješavanje te zato što bi bilo preskupo tražiti sva moguća ograničenja, a ne znamo unaprijed hoće li nam ona pomoći u "rezanju" nekih rješenja, ne dodajemo takva ograničenja na početku već kao u primjeru 6.2.1 tek nakon što uvidimo da ona imaju povoljan učinak na "rezanje" nekih rješenja.

Prije izvršavanja samog algoritma, pokreće se *presolve* te se traži heurističko rješenje. *Presolve* ima za cilj smanjiti veličinu problema te pooštriti funkciju cilja, ako je to moguće. Kao što smo vidjeli u potpoglavlju 6.1 *presolve* je moguće isključiti ili birati između dvije razine rada *presolva*. Pretpostavimo da rješavamo problem minimizacije. Heuristika se temelji na tome da postavimo na početku rješavanja problema čim bolju gornju ogradu. Posljedica nalaženja zadovoljavajuće gornje ograde je veća vjerojatnost da će rješenje relaksiranih problema linearnog programiranja biti veća od heurističnog rješenja te će to dovesti do "rezanja" čvorova. Kao što smo vidjeli u potpoglavlju 6.1 možemo mijenjati parametar koji određuje koliki postotak vremena rješavanja problema će posvetiti traženju heurističkog rješenja, zadano je da to bude 5% vremena.

Uz sve gore navedene tehnike softver za rješavanje cjelobrojnog problema linearnog programiranja koristi i tehniku odabira varijabli grananja, otkrivanje simetrije, otkrivanje disjunktnih podstabala. Cilj je ograničiti veličinu razgranatosti stabla.

Naredba koja se u Gurobi-u koristi za pokretanje optimizacije kako bi pronašli rješenje koje minimizira ili maksimizira funkciju cilja, uz poštovanje svih ograničenja koja su definirana u modelu je sljedeća:

`optimize ( callback=None )`

Argumenti:

- **callback** : funkcija `Callback(model, where)` , tijekom optimizacije povremeno će se pokretati, ona ima dva parametra, gdje *model* predstavlja model koji optimiziramo, a *where* predstavlja kada će se u procesu optimizacije pokretati *callback* funkcija

Povratna vrijednost: nema
Nakon uspješnog završetka, ova će metoda popuniti atribute modela koji se odnose na rješenje, također popunit će i neke druge informacije o samom procesu rješavanja. Neki atributi modela koje ova naredba popunjava:

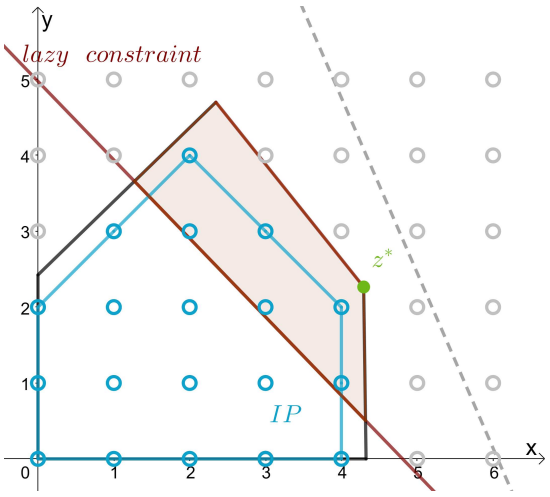
- **ObjVal** : optimalno nađeno rješenje
- **ObjBound** : najbolja nađena ograda za funkciju cilja (donja ograda za minimizaciju, gornja ograda za maksimizaciju)
- **Status** : trenutni status optimizacije modela, postoji 17 vrsta statusa, neki od njih su:
  - LOADED : model je učitan, ali informacije o rješenju nisu dostupne
  - OPTIMAL : model je riješen s obzirom na danu toleranciju rješenja te je dostupno optimalno rješenje
  - INF_OR_UNBD : model nije dopustiv ili je neograničen, potrebno je postaviti parametar DualReductions na 0 i ponoviti optimizaciju
  - INFEASIBLE : model nije dopustiv
  - UNBOUNDED :model je neograničen
  - NUMERICAL: optimizacija je prekinuta zbog nepopravljivih numeričkih poteškoća
  - SUBOPTIMAL : ne može se zadovoljiti tolerancija optimalnosti, dostupno je suboptimalno rješenje
- **PoolObjVal** : alternativna rješenja pohranjena tijekom procesa optimizacije
- **Runtime** : vrijeme potrebno da se izvede optimizacija
- **IterCount** : broj provođenja simplex algoritma tijekom izvođenja optimizacije
- **NodeCount** : broj čvorova u stablu rješenja koje se generiralo tijekom optimizacije
- **OpenNodeCount** : broj čvorova koji su ostali otvoreni u stablu rješenja jer je nađeno optimalno rješenje

Popis svih atributa dostupan je na [3].

6.3 *Lazy constraint* u Gurobi-u

Lazy constraint predstavlja skup ograničenja koja su zadana u početnom modelu, ali se ne dodaju na početku rješavanja problema, već se dodaju u model onda kada je pronađeno rješenje koje krši neko od ograničenja koja su zadana kao lazy constraint. Lazy constraint se koristi kada model uključuje iznimno puno ograničenja od kojih predviđamo da će mnoga biti suvišna, korištenjem takvih rješenja štedimo na memoriju te na vremenu za generiranje svih ograničenja na početku.

Prema Gurobi dokumentaciji, najbolje je provjeravati zadovoljava li rješenje uvjete *lazy constraint* kroz callback funkciju.



Slika 6.2: Primjer gdje se rješenje *LP* problema z^* odbacuje jer ne zadovoljava uvjete zadane kao *lazy constraint*

Za dohvaćanje privremenih rješenja optimizacije koristimo naredbu Model.cb GetSolution(). Ova se naredba može pozvati samo kada je vrijednost *where* u Callback funkciji GRB.Callback.MIPSOL, što znači da je proces optimizacije, našao novo privremeno rješenje.

```
cbGetSolution(vars)
```

Argumenti:

- **vars** : varijable modela, čije vrijednosti želimo dohvatiti, može biti dano kao jedna varijabla, matrica varijabli, lista varijabli ili rječnik varijabli

Povratna vrijednost:

- vrijednosti za navedene varijable u rješenju, format zapisa jednak je ulaznom parametru

Za dodavanje novih ograničenja tipa *lazy constraint* unutar funkcije Callback koristimo naredbu Model.cbLazy(). I ova naredba se može pozvati samo, ako je vrijednost *where* u Callback funkciji GRB.Callback.MIPSOL.

```
cbLazy(lhs, sense, rhs )
```

Argumenti:

- **lhs** : lijeva strana izraza novog ograničenja
- **sense** : operator usporedbe između lijeve i desne strane novog ograničenja (GRB.LESS_EQUAL, GRB.EQUAL ili GRB.GREATER_EQUAL)
- **rhs** : desna strana izraza novog ograničenja

Poglavlje 7

Implementacija rješenja za problem trgovačkog putnika

U ovom poglavlju govorit ćemo o implementaciji jednog egzaktnog rješanja za problem trgovačkog putnika. Problem je definiran pomoću Dantzig-Fulkerson-Johnson formulacije. Program sam pisala u interaktivnom okruženju *Jupyter Notebook 6.4.8.* te sam koristila besplatnu *Gurobi 10.0.3* studentsku licencu. Biblioteke koje je potrebno učitati su sljedeće:

```
import gurobipy as gp
from gurobipy import GRB
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import time
import math
```

Kod 7.1: Biblioteke potrebne za rad

Biblioteka **gurobipy** obuhvaća Gurobi Python modul koji sadrži funkcije i klase koje su opisane u poglavlju 6, također uvozimo i konstante i simbole iz Gurobi Python modula.

U varijablu *points* spremamo koordinate točaka (gradova) iz tekstualnih datoteka s [11]. Redoslijedom kojim učitavamo podatke, tako ih i numeriramo.

Potrebna nam je funkcija za određivanje udaljenosti između dva vrha, jer će to biti koeficijenti u funkciji cilja.

```
def distance (tocka1,tocka2):
    x1, y1 = tocka1
    x2, y2 = tocka2
    return math.sqrt((x1 - x2)**2 + (y1 - y2)**2)
```

Kod 7.2: Funkcija za računanje udaljenosti među dva vrha

Potrebno je stvoriti novi model. Postavili smo parametre tako da će model raditi bez ispisivanja poruka tijekom rješavanja te će se tražiti egzaktno optimalno rješenje. Dodajemo

varijable x_{ij}, $i > j$ u model gdje je $x_{ij} = 0$, ako brid koji povezuje vrhove i i j nije uključen u rješenje, a $x_{ij} = 1$, ako je uključen. Postavlja se funkcija cilja kao u (4.6) te ograničenja kao u (4.7) i (4.8).

```
# definiramo model
m = gp.Model("TSP")


V = range(len(points))
E = gp.tuplelist([(i, j) for i in V for j in V if i > j])

# postavljanje parametara
m.setParam(GRB.Param.OutputFlag, 0)
m.setParam(GRB.Param.MIPGap, 0)

# postavljanje varijabli
x = m.addVars(E, vtype=GRB.BINARY)

#postavljane fje cilja
m.setObjective(gp.quicksum([distance(points[i], points[j]) * x[i, j] for
    i, j in E.select("*", "*")]), GRB.MINIMIZE)

# postavljanje ogranicenja
for i in V:
    m.addConstr(gp.quicksum([x[i, j] for _, j in E.select(i, "*") if j
   != i]) + gp.quicksum([x[j, i] for j, _ in E.select("*", i) if j != i
   ]) == 2)
```

Kod 7.3: Definiranje početnog modela

Uvjet (4.9) iz Dantzig–Fulkerson–Johnson formulacije ne uvodimo za sve prave podskupove vrhova od barem dva člana već kao *lazy constraint*. Odnosno, tek kada potencijalno optimalno rješenje ne zadovoljava neki od uvjeta iz (4.9), tada taj uvjet uvodimo u model. Niže je dana funkcija koja po potrebi dodaje nova ograničenja u model te pokreće proces optimizacije te ispisuje vrijednost trenutno optimalne funkcije cilja. Funkcija vraća rječnik takav da ključ rječnika predstavlja brid $(i, j) : i > j$, a vrijednost rječnika je binarna vrijednost, govori je li brid uključen (1) ili ne (0).

```
def solve_tsp(points, model, subtours=[]):

    #dodavanje uvjeta za podture
    for subtour in subtours:
        model.addConstr(gp.quicksum([x[i, j] for i in subtour for j in
   subtour if i > j]) <= len(subtour) - 1)

    #pokretanje optimizacije
```

```
    model.optimize()

    #ispis rjesenja
    if model.status == GRB.status.OPTIMAL:
        print('Optimalno rjesenje je %g' % m.objVal)
        result = {edge: x[edge].X for edge in E}
        return result
    else:
        print("Greska")
        raise SystemExit
```

Kod 7.4: Funkcija koja dodaje u model ograničenja tipa *lazy constraint* te pokreće proces optimizacije

Potrebno je odrediti komponente povezanosti koje se javljaju u posljednjem rješenju koje smo dobili, kako bismo dodali dodatne uvjete tipa (4.9) u model za te komponente povezanosti. U biblioteci NetworkX postoji funkcija koja iz danog grafa vraća sve komponente povezanost, stoga je potrebno definirati graf trenutnog rješenja. Graf definiramo dodajući bridove koji su uključeni u posljednjem rješenju.

```
def add_edges_from_dict(graph, edge_dict):
    #definiranje grafa s bridovima gdje je value u rjecniku >=1
    for edge, value in edge_dict.items():
        if value >=1:
            graph.add_edge(*edge)


def conncom(solution):
    G = nx.Graph()
    add_edges_from_dict(G, solution)
    #prepoznavanje komponenta povezanosti
    components = [subgraph for subgraph in nx.connected_components(G)]
    return components
```

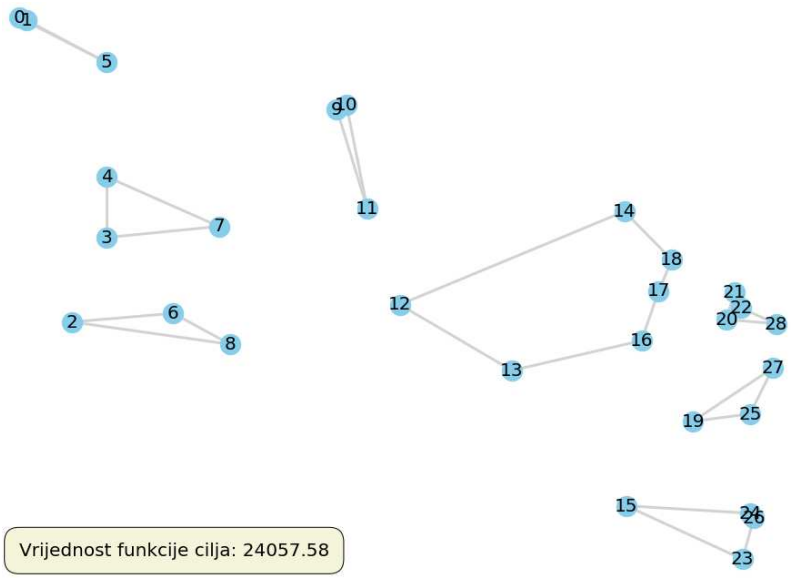
Kod 7.5: Funkcija `add_edges_from_dict` definira graf koji je zadan posljednjim rješenjem, a funkcija `conncom` vraća komponente povezanosti u danom grafu

Kako bi dobili konačno rješenje problema trgovačkog putnika potrebno je prvo pozvati funkciju `solve_tsp` na modelu koji nema nikakve uvjete tipa (4.9), zatim pomoćnu funkcije `conncom` koja vraća vrijednost svih komponenata povezanosti u grafu koji predstavlja rješenje problema. Ako postoji samo jedna komponenta povezanosti, to znači da je prvi poziv funkcije `solve_tsp` dao rješenje početnog problema trgovačkog putnika. Ako postoji više od jedne komponente povezanosti, tada je potrebno uvesti ograničenja tipa *lazy constraint* na temelju komponenta povezanosti u prethodnom rješenju te riješit takav problem. Potrebno je vidjeti da li novo rješenje ima jednu ili više od jedne komponente povezanosti te u skladu s tim proglasiti to rješenje početnog problema trgovačkog putnika ili nastaviti postupak koji je gore opisan dok ne dobijemo rješenje s jednom komponentom povezanosti

```
t0=time.time()
sol=solve_tsp(points, m)
com=conncom(sol)
while len(com) > 1:
    sol=solve_tsp(points, m, com)
    com=conncom(sol)
t1=time.time()
print("Vrijeme izvodenja algoritma:", t1 - t0, "sekundi")
```

Kod 7.6: Dobivanje konačnog rješenja početnog problema trgovačkog putnika

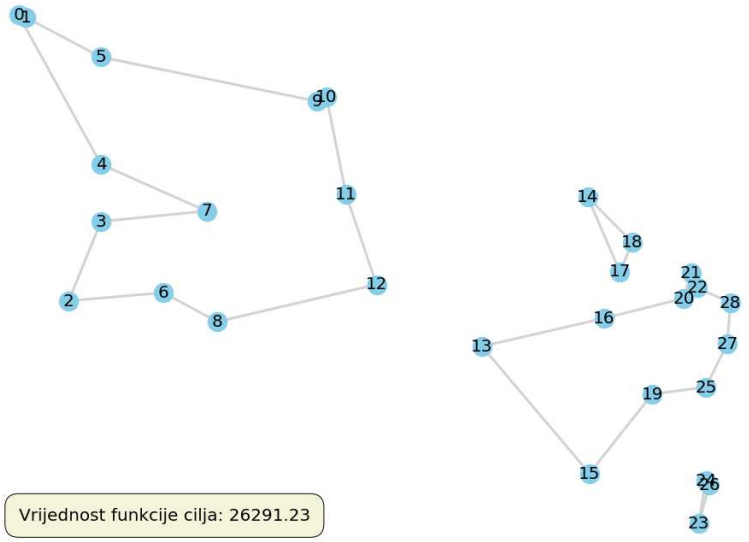
Primjer 7.0.1. *Na primjeru podataka wi29.txt preuzetih s [11] pogledajmo kako radi gore navedeni algoritam. Nakon definiranja početnog modela s minimizacijskom funkcijom cilja*



Slika 7.1: Rješenje nakon 1. iteracije

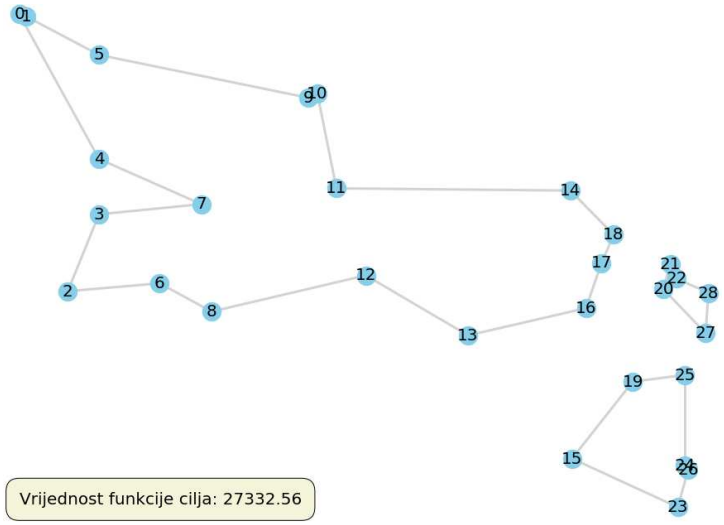
(4.6), te ograničenjima (4.7)-(4.8), pozivamo optimizaciju i dobijemo rješenje prikazano na slici7.1. Algoritam će na početan model dodati još 8 *ograničenja koji su formulirani kao (4.9) za svaku komponentu povezanosti, gdje vrhovi svake komponente povezanosti predstavljaju skupove* $S_i, i \in \{1, ..., 8\}$. *Takav model se optimizira te je rješenje u obliku grafa dana na slici 7.2.*

Uočimo da grafičko rješenje sa slike 7.2 ne zadovoljava ograničenje (4.9) jer postoje ciklusi s manje od $n = 29$ *vrhova. Stoga, je potrebno uvesti dodatna* 4 *uvjeta koji su formulirani kao (4.9) za svaku komponentu povezanosti, gdje vrhovi svake komponente*



Slika 7.2: Rješenje nakon 2. iteracije

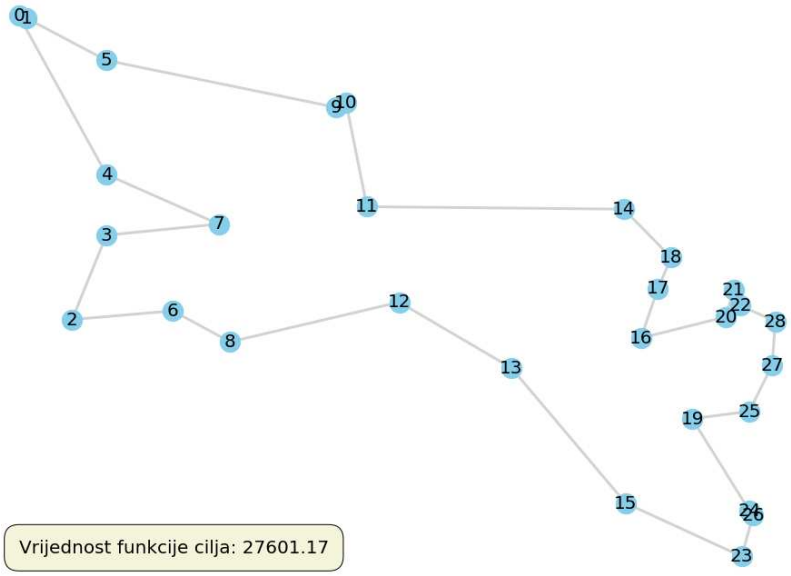
povezanosti predstavljaju skupove $S_i, i \in \{9, 10, 11, 12\}$. *Takav model se optimizira te je rješenje u obliku grafa dano na slici 7.3.*



Slika 7.3: Rješenje nakon 3. iteracije

Ni, grafičko rješenje sa slike 7.3 ne predstavlja rješenje početnog problema jer sadrži

podcikluse duljine manje od $n = 29$. *Stoga, je potrebno uvesti još* 3 *ograničenja koja su formulirana kao (4.9) za svaku komponentu povezanosti, gdje vrhovi svake komponente povezanosti predstavljaju skupove* $S_i, i \in \{13, 14, 15\}$. *Takav model se optimizira te je rješenje u obliku grafa dano na slici 7.4.*



Slika 7.4: Konačno rješenje početnog problema

Primijetimo da smo dobili konačno rješenje jer dano rješenje zadovoljava sve uvjete (4.7)-(4.9) u Dantzig–Fulkerson–Johnson formulaciji. Primijetimo da smo ovim algoritmom, uveli 15 *ograničenja tipa (4.9) za skupove* S_i, $i \in \{1, 2, ..., 15\}$ *umjesto* $2^n - 1 = 2^{29} - 1 = 536\,870\,911$.

Implementacija algoritma je testirana na osam grupa podataka s [11]. U datotekama su zapisane koordinate vrhova. Te su u tablici 7.1 dani podaci o tome koliko svaka grupa podataka ima zadanih koordinata vrhova (znamenke na kraju imena podataka), optimalno rješenje, vrijeme izvršavanja algoritma u sekundama, uzeta je prosječna vrijednost izvršavanja algoritma mjerenih 1000 puta, broj iteracija odnosno koliko puta se pokrenuo proces optimizacije dok nismo došli do krajnjeg rješenja, koliko je dodano ograničenja tipa (4.9) u model, koliko je bilo potrebno riješiti problema relaksiranog linearnog programiranja te koliko se iteracija simplex algoritma pokrenulo kroz cijeli proces optimizacije. Na stranici [11] dostupna su i egzaktna rješenja problema, stoga sam nakon procesa optimizacije provjerila svoja rješenja te se ona poklapaju.

podaci	**optimalno rješenje**	**vrijeme izvršavanja algoritma**	**broj iteracija**	**ukupan broj ograničenja vezanih uz podcikluse**	**ukupan broj relaksiranih LP problema**	**ukupan broj iteracija simplex algoritma**
wi29.txt	27601.17	0.055479	4	15	11	236
ulysses22.txt	75.31	0.0479146	5	13	14	211
eil51.txt	428.87	0.369947	5	20	81	1543
dj38.txt	6659.43	0.059878	4	10	11	254
burma14.txt	30.88	0.008328	2	3	2	41
ulysses16.txt	73.99	0.032803	5	11	21	172
berlin52.txt	7544.37	0.033908	2	7	7	163
att48.txt	33523.71	0.277849	7	22	45	1253

Tablica 7.1: Usporedba nekih varijabli rješenja problema trgovačkog putnika

Slika 7.5 primjer je ispisa Gurobia nakon svake optimizacije. Navest ćemo značenje nekih od stavki sa slike 7.5:

1. ***Found heuristic solution*** : ukazuje da je pronađeno početno heurističko rješenje s danom ciljnom funkcijom
2. ***Presolve time*** : vrijeme potrebno za *presolve* fazu optimizacije
3. ***Presolved*** : informacija o veličini problema nakon faze *presolve*
4. ***Root relaxation*** : rezultat relaksacije u korijenu stabla pretraživanja
    - objective : vrijednost ciljne funkcije nakon relaksacije
    - iterations : broj iteracija provedenih tijekom relaksacije
    - seconds : ukupno vrijeme u sekundama koliko je trajala relaksacija
    - work units : predstavlja radne jedinice utrošene tijekom relaksacije
5. ***Node*** : broj čvorova (nodova) u stablu pretraživanja koje se trenutno razmatra tijekom optimizacije
    - Expl : broj čvorova koji smo istražili u cijelom stablu pretraživanja
    - Unexpl : broj čvorova koji nismo istražili u cijelom stablu pretraživanja
6. ***Current Node*** : informacije o trenutnom čvoru u stablu pretraživanja

- Obj : vrijednost funkcije cilja
- Depth : dubina čvora
- IntInf : vrijednost ukazuje na to koliko su trenutne varijable cjelobrojnog rješenja udaljene od cjelobrojnih vrijednosti, primjer, ako model ima 406 varijabli te, ako je $IntInf = 16$, tada je $406 - 16 = 390$ varijabli cjelobrojne, a 16 ih nije

7. ***Objective Bounds*** : granice ciljne funkcije koje su trenutno poznate
   - Incumbent : najbolje poznato rješenje optimizacijskog problema (u poglavlju 5 definirano kao gornja granica)
   - BestBd : predstavlja najbolju donju granicu ciljne funkcije
   - Gap : relativna razlika između vrijednosti *Incumbent* i *BestBd* (izražena kao postotak)
8. ***Cutting planes*** : odnosi se na primijenjene metode za uklanjanje nekorisnih grana ili varijabli u stablu pretraživanja kako bi se ubrzao postupak optimizacije te su u nastavku nabrojene metode koje su se koristile i koliko puta
9. ***Explored*** : odnosi se na informacije o tijeku optimizacije
   - Nodes : broj istraženih čvorova u stablu pretraživanja
   - Simplex iterations : broj iteracija simplex metode koje su provedene tijekom optimizacije
10. ***Solution count*** : broj pronađenih rješenja tijekom optimizacije i njihove vrijednosti ciljnih funkcija

Napomena 7.0.2. *Ako se optimizacija izvršava na problemu s cjelobrojnim varijablama i IntInf ostaje različit od nule, to može ukazivati na problem s konvergencijom ili na to da postoji neka vrsta ograničenja koja onemogućava pronalazak cjelobrojnog rješenja.*

```
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (win64)

CPU model: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads

Optimize a model with 29 rows, 406 columns and 812 nonzeros
Model fingerprint: 0xc1cd9779
Variable types: 0 continuous, 406 integer (406 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [7e+01, 1e+04]
  Bounds range     [1e+00, 1e+00]
  RHS range        [2e+00, 2e+00]
Found heuristic solution: objective 117043.99782 (1)
Presolve time: 0.01s (2)
Presolved: 29 rows, 406 columns, 812 nonzeros (3)
Variable types: 0 continuous, 406 integer (406 binary)

Root relaxation: objective 2.359609e+04, 35 iterations, 0.00 seconds (0.00 work units) (4)

    Nodes (5)    |  Current Node (6)  |     Objective Bounds (7)      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 23596.0851    0   16 117043.998 23596.0851  79.8%     -    0s
H    0     0                      26342.939365 23596.0851  10.4%     -    0s
     0     0 23678.0602    0   10 26342.9394 23678.0602  10.1%     -    0s
     0     0 23678.0602    0   16 26342.9394 23678.0602  10.1%     -    0s
H    0     0                      25763.306527 23678.0602  8.09%     -    0s
*    0     0                0     24057.581480 24057.5815  0.00%     -    0s

Cutting planes: (8)
  Gomory: 2
  Zero half: 4

Explored 1 nodes (81 simplex iterations) in 0.13 seconds (0.01 work units) (9)
Thread count was 4 (of 4 available processors)

Solution count 4: 24057.6 25763.3 26342.9 117044 (10)

Optimal solution found (tolerance 0.00e+00)
Best objective 2.405758147967e+04, best bound 2.405758147967e+04, gap 0.0000%
```

Slika 7.5: Primjer ispisa nakon optimizacije (1. optimizacija za primjer *wi29.txt*)

Zaključak

Egzaktni algoritmi za rješavanje problema cjelobrojnog linearnog programiranja se koriste kada nam je od izrazite važnosti pronaći točno rješenje. Egzaktne metode često zahtijevaju puno vremena i resursa, pogotovo kako problem postaje složeniji. Problemi cjelobrojnog linearnog programiranja često su NP-teški, što znači da vrijeme izvođenja eksponencijalno raste s povećanjem veličine problema. Softver Gurobi koristi niz naprednih tehnika za egzaktno rješavanje problema cjelobrojnog linearnog programiranja kao što su heuristike za inicijalno rješenje, različite strategije pretraživanja, *cutting plane* tehnike kako bi vremenski ubrzao izvođenje algoritma. Zbog ograničenja studentske licence Gurobi softvera, koja omogućava da model ima maksimalno 60 vrhova u grafu nismo mogli istražiti sve mogućnosti Gurobi softvera za rješavanje problema trgovačkog putnika. Također daljnji smjerovi ispitivanja mogućnosti softvera Gurobi mogu ići u smjeru povećavanja složenosti samog modela implementirajući neke od inačica problema opisanih u potpoglavlju 2.2, 4.2.

Bibliografija

[1] L. A. Wolsey G. Nemhauser, *Integer and Combinatorial Optimization*, Wiley, 1999.

[2] F. Greco, *Traveling Salesman Problem*, In-Teh, 2008.

[3] LLC Gurobi Optimization, `https://www.gurobi.com/`, posjećena 25.10.2023.

[4] D. Pisinger H. Kellerer, U. Pferschy, *Knapsack Problems*, Springer, 2004.

[5] Z. Lukač L. Neralić, *Operacijska istraživanja*, Element, 2012.

[6] J. Lee, *A First Course in Combinatorial Optimization*, Cambridge University Press, 2004.

[7] G. Zambelli M. Conforti, G. Cornuejols, *Integer Programming*, Springer, Cham CH, 2014.

[8] R. Manger, *Strukture podataka i algoritmi*, Element, Zagreb, 2014.

[9] S. Stahl M.J. Brusco, *Branch-and-bound applications in combinatorial data analysis*, Springer, 2005.

[10] S. Martello R. Burkard, M. Dell'Amico, *Assignment Problems*, SIAM, 2009.

[11] Heidelberg Sveučilište, *Discrete and Combinatorial Optimization*, `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`, posjećena 30.10.2023.

[12] A. B. Kahn T. C. Hu, *Linear and Integer Programming Made Easy*, Springer, Cham CH, 2014.

[13] M. J. Beckmann T. C. Koopmans, *Assignment problems and the location of economic activities*, Econometrica **25** (1957), br. 1, 53–76.

[14] A. Golemac V. Bosančić, *Kako pomoći trgovačkom putniku*, Osječki matematički list **12** (2012), br. 2, 139–149.

[15] D. Veljan, *Kombinatorna i diskretna matematika*, Algoritam, Zagreb, 2001.

[16] E. Çela, *The Quadratic Assignment Problem - Theory and Algorithms*, Springer, 1998.

Sažetak

Cjelobrojno linearno programiranje se bavi problemom optimizacije gdje su funkcija cilja i ograničenja linearni, a neke od varijabli odlučivanja su cjelobrojne. Najpoznatije metode za egzaktno rješavanje ovakvih problema su *branch-and-bound* odnosno *cutting-plane* metoda. U ovom radu veći je naglasak na metodi *branch-and-bound*. U prvom poglavlju upoznali smo se osnovnim pojmovima iz teorije grafova koje su nam potrebne za razumijevanje modela pojedinih problema. U drugom, trećem i četvrtom poglavlju upoznali smo se s poznatim problemima cjelobrojnog linearnog programiranja: problemom naprtnjače, problemom dodjeljivanja te problemom trgovačkog putnika. U petom poglavlju opisali smo *branch-and-bound* algoritam te smo riješili jednostavan problem naprtnjače tom metodom. U šestom i sedmom poglavlju smo se upoznali s Gurobi softverom te smo implementirali rješenje za problem trgovačkog putnika koristeći Gurobi softver.

Summary

Integer linear programming deals with optimization problems where the objective function and constraints are linear and some decision variables are integers. The most common exact methods for solving such problems include the *branch-and-bound* method and the *cutting plane* method. In this work, there is more emphasis on the *branch-and-bound* method. In the first chapter, we have introduced some to basic concepts from graph theory that we need to understand the models of certain problems. In the second, third and fourth chapters, we have introduced the well-known integer linear programming problems: the knapsack problem, the assignment problem and the traveling salesman problem. In the fifth chapter, we have described the *branch-and-bound* algorithm and solved a simple knapsack problem with that method. In the sixth and seventh chapters, we have got acquainted with the Gurobi software and implemented a solution to the traveling salesman problem using the Gurobi software.

Životopis

Rodena sam 7. siječnja 2000. godine u Čakovcu. Nakon završene osnovne škole upisala sam Gimnaziju Josipa Slavenskog Čakovec, opći smjer, u kojoj sam maturirala 2018. godine. Zatim sam 2018. upisala Preddiplomski sveučilišni studij Matematika na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta u Zagrebu, koji sam završila 2021. godine. Iste godine upisala sam Diplomski sveučilišni studij Matematičke statistike na istom fakultetu.