Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nika Eržen

Obhod trgovskega potnika po zemljevidu Slovenije

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: prof. dr. Gašper Fijavž

Ljubljana, 2018

Zahvala

Za pomoč, sodelovanje in nasvete pri nastajanju magistrskega dela se najlepše zahvaljujem mentorju prof. dr. Gašperju Fijavžu.

Prav tako se iskreno zahvaljujem družini in prijateljem, ki so mi pri delu pomagali in me vzpodbujali.

Nika Eržen, 2018

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
TSP	traveling salesman problem	problem trgovskega potnika
LK	Lin-Kernighan heuristics	hevristika Lin-Kernighan
LKH	Lin-Kernighan-Helsgaun Heuristics	hevristika Lin-Kernighan-Helsgaun
NP	nondeterministic polynomial time	nedeterministični polinomski čas

Povzetek

Naslov: Obhod trgovskega potnika po zemljevidu Slovenije

Problem trgovskega potnika je dobro znan NP-težak problem. Cilj problema je obresti določeno množico mest tako, da pri tem prehodimo čim krajšo pot in se vrnemo v izhodišče. V magistrski nalogi smo poiskali obhod trgovskega potnika po 6007 naseljih Slovenije glede na geografske razdalje med naselji. Za iskanje smo uporabili programa LKH in Concorde. S programom LKH smo poiskali zgornjo mejo obhoda trgovskega potnika. Nato smo s programom Concorde poiskali spodnjo mejo za obhod trgovskega potnika in jo nato izboljševali, dokler nismo dosegli izenačenja spodnje in zgornje meje. Našli smo obhod dolžine 7733,125km in pokazali njegovo optimalnost.

Ključne besede

problem trgovskega potnika, geografski podatki

Abstract

Title: Traveling salesman problem on the map of Slovenia

The traveling salesman problem is a well known NP-hard problem. Given a list of cities and the distances between each pair of cities, the goal is to find the shortest possible route that visits each city and returns to the origin city. In the thesis we found the solution to the traveling salesman problem on 6007 settlements in Slovenia using the geographical distance between the settlements. To find the solution we used programs LKH and Concorde. With LKH we obtained the upper bound of traveling salesman tour. Than we used Concorde to obtain and improve the lower bound of the traveling salesman tour. We have found the tour of length 7733,125km and have shown that it is optimal.

Keywords

traveling salesman problem, geographical data

Poglavje 1

Uvod

Problem trgovskega potnika je klasičen algoritmični problem na področjih operacijskih raziskav, matematične optimizacije in teoretičnega računalništva. Trgovski potnik mora obresti določeno množico mest tako, da pri tem prehodi čim krajšo pot in se vrne v izhodišče. Problem trgovskega potnika je matematični problem in ga najlažje opišemo z grafom, ki opisuje lokacije mest in razdalje med njimi.

Za problem trgovskega potnika obstaja več baz testnih podatkov. Med njimi je tako imenovana nacionalna TSP družina 27 problemov [1], kjer je izziv najti najkrajši obhod po mestih v določeni državi ali delu države. Problemi vsebujejo od 28 mest v Zahodni Sahari pa do 71009 mest na Kitajskem. Od vseh 27 problemov so trije še vedno rešeni zgolj približno. Prvotno kolekcijo dopolnjujejo dodatni primeri obhodov po mestih drugih držav, npr. Romuniji in Švici [2]. Primera za Slovenijo nismo našli, zato smo se v magistrski nalogi odločili poiskati podatke za obhod trgovskega potnika po naseljih Slovenije in problem tudi poskusiti rešiti. Za naselja smo se odločili, ker je število mest premajhno. Pridobili smo imena 6007 naselij Slovenije in njihove geografske koordinate. S programom LKH smo najprej poiskali obhod, ki določa zgornjo mejo za vrednost optimalnega obhoda. S programom Concorde pa smo potrdili, da je ta zgornja meja natančna — krajši obhod ne obstaja. Dodatno smo poiskali še optimalni obhod po občinah Slovenije in

ga primerjali z optimalnim obhodom po naseljih.

V poglavju *TSP in uvodne definicije* povemo nekaj več o samem problemu trgovskega potnika in njegovih različicah. Na kratko opišemo tudi nekatere načine reševanja tega problema. Definiramo tudi nekatere matematične pojme, ki jih potrebujemo za nadaljno razumevanje magistrske naloge. V poglavju *LKH - iskanje zgornje meje* natančneje opišemo idejo programa LKH in orišemo njegovo implementacijo. V poglavju *Concorde - iskanje spodnje meje* podobno naredimo za program Concorde. V poglavju *Pridobitev podatkov* natančno opišemo pridobitev podatkov in zapis v standardno obliko, ki jo sprejmeta programa Concorde in LKH. Za zaključek si v poglavju *Analiza rezultatov* pogledamo optimalni obhod po naseljih Slovenije in ga primerjamo z optimalnim obhodom po občinah Slovenije.

Poglavje 2

Problem trgovskega potnika in uvodne definicije

V tem poglavju bomo najprej definirali problem trgovskega potnika in nekatere njegove različice. Na kratko bomo opisali tudi nekaj načinov za njegovo reševanje. Nato bomo uvedli nekaj uvodnih definicij iz področij grafov, linearnih programov in iskalnih dreves.

2.1 Problem trgovskega potnika

Problem trgovskega potnika (TSP - Traveling Salesman Problem) je dobro znan NP-poln problem [3], ki ga lahko na kratko povzamemo s sledečo nalogo. Pri danem seznamu mest in cenami povezav med njimi, poišči najcenejšo možno pot, ki gre skozi vsako mesto natanko enkrat in se na koncu vrne v izhodiščno mesto. TSP modeliramo kot graf, kjer mesta predstavljajo točke, cene pa utežene povezave v grafu. Če velja, da je cena med mestom A in mestom B enaka ceni med mestom B in mestom A, je graf *neusmerjen* in TSP *simetričen*. Če to ne velja, je graf *usmerjen* in TSP *asimetričen*. Če cena obstaja za vsak par mest, je graf *poln*. V splošnem TSP predstavlja neusmerjen poln graf. Posebna oblika problema trgovskega potnika je *metrični problem trgovskega potnika*, kjer za uteži na povezavah velja triko-

tniška neenakost, t.j. cena povezave od mesta A do mesta C je manjša ali enaka vsoti cen povezav, ki gresta od mesta A do poljubnega mesta B in od mesta B do mesta C. Uteži v metričnem problemu trgovskega potnika imenujemo *razdalje*.

Rešitev TSP z n mesti lahko najdemo, če preverimo $\frac{(n-1)!}{2}$ različnih možnih obhodov. Že za relativno majhno število mest je preverjanje vseh možnih obhodov zamudno in pretirano. Za iskanje rešitve TSP so se tako razvili različni pristopi. V grobem jih delimo na eksaktne algoritme, ki zagotovo vrnejo optimalni obhod, in hevristične algoritme, za katere pričakujemo, da bodo vrnili dober približek optimalnemu obhodu. Hevristične algoritme nadalje delimo na algoritme, ki obhod gradijo, in algoritme, ki ga izboljšujejo. Ker je TSP NP-poln problem, znani eksaktni algoritmi niso bistveno hitrejši od preverjanja vseh možnih obhodov.

Held in Karp sta predstavila dinamični algoritem, ki rekurzivno išče aproksimacijo optimalnega obhoda [4]. Algoritem sloni na predpostavki, da je vsaka podpot minimalne poti minimalna. Zaradi visoke prostorske zahtevnosti je algoritem primeren le za manjše probleme. Za večje probleme se uporablja metoda, ki z linearno relaksacijo iterativno znižuje ceno obhoda, dokler ne najde rešitve. Tovrstne metode reševanja linearnih programov z velikimi in kompleksnimi sistemi neenakosti so predstavili Dantzig, Fulkerson in Johnson [5], imenujemo jih *metode presečnih ravnin*. Ko z metodo presečnih ravnin obhoda ne moremo več izboljšati, ga lahko razvejimo na dva podproblema in ju rešujemo ločeno. Podproblem nadalje zopet razvejamo ali ugotovimo, da je irelevanten, in ga opustimo. Ta metoda sodi med metode razveji in omeji [6], imenujemo jo razveji in odreži [7]. Trenutno najbolj učinkovita implementacija te metode je program Concorde [8, 9, 10].

Med hevristične algoritme, ki gradijo obhod, uvrščamo algoritem najbližjega soseda in algoritem z vstavljanjem. Algoritem najbližjega soseda je požrešni algoritem, pri katerem naključno izberemo začetno mesto v obhodu. Obhodu nato na vsakem koraku dodamo zadnjemu mestu najbližje še ne obiskano mesto. Pri algoritmu z vstavljanjem prav tako naključno iz-

beremo začetno mesto v obhodu. Nato na vsakem koraku dodamo še ne obiskano mesto in ga vstavimo med tisti dve zaporedni mesti na obhodu, kjer je novi obhod najcenejši. Najnatančnejši in najcenejši algoritem z vstavljanjem lahko povežemo z minimalnim vpetim drevesom [11].

λ-opt hevristike, spadajo med hevristične algoritme, ki izboljšujejo obhod. Bistvo take hevristike je, da v že obstoječem obhodu poiščemo λ mnogo povezav in jih zamenjamo z λ drugimi povezavami tako, da izboljšamo obhod. V praksi uporabljamo 2-opt [12] in 3-opt [13] hevristiki. Lin in Kernighan sta razvila algoritem Lin-Kernighan (LK), ki za razliko od λ-opt hevristik ne menjava fiksnega števila povezav, ampak poskuša na vsakem koraku najti najprimernejšo število povezav za menjavo [14]. Najboljša implementacija algoritma LK je njena modificirana verzija imenovana LKH [15].

2.2 Uvodne definicije

Za potrebe magistrske naloge moramo poznati nekaj osnovnih pojmov iz področij grafov, linearnih programov in iskalnih dreves. V naslednjih razdelkih bomo definirali in razložili določene pojme iz teh področij.

2.2.1 Osnove grafov

Naj bo $G = (V, E)$ graf, kjer je $V = \{v_1, v_2, \ldots, v_n\}$ množica vozlišč in $E = \{ e_1, e_2, \ldots, e_m \} = \{ (v_i, v_j) \mid v_i \in V, v_j \in V \}$ množica povezav. Naj bo $W \subset V$, potem je *inducirani podgraf* graf katerega množica vozlišč je enaka W in je množica povezav enaka vsem povezavam, ki imajo obe krajišči v množici W. Podgraf induciran z množico W označimo z G^W.

Graf G ima $n = |V|$ točk in $m = |E|$ povezav. S c_e ali $c(e)$ označimo utež na povezavi e. Za vsako podmnožico $S \subseteq E$ z $L(S)$ označimo njeno *dolžino* in jo definiramo kot $L(S) = \sum_{e \in S} c_e$. *Incidenčna matrika* je matrika velikosti $n \times m$, ki predstavlja graf G. Za usmerjen graf je (i, j)-ti element enak -1, če je i-to vozlišče končna točka j-te povezave in 1, če je i-to vozlišče začetna točka j-te povezave. V neusmerjenem grafu je v obeh primerih (i, j)-

ti element enak 1. Drugače je (i, j)-ti element enak 0. Iz stolpca j tako lahko razberemo, kateri dve vozlišči povezuje j-ta povezava in morebitno smer povezave. Iz vrstice i pa izvemo, krajišče katerih povezav je i-to vozlišče.

Sprehod v grafu je zaporedje točk $v_0v_1 \dots v_k$, kjer velja $v_iv_{i+1} \in E$ za $i = 0, \dots, k-1$. *Pot* v grafu je sprehod v katerem so vse točke različne. *Cikel* v grafu je sprehod, kjer je prvih $k-1$ točk različnih, zadnja točka pa je enaka prvi. Graf G je *povezan*, če za vsak par vozlišč obstaja pot med njima. Graf G je *2-povezan*, če je povezan in ostane povezan, če odstranimo katerokoli vozlišče. *Hamiltonova pot* je pot v grafu, ki gre skozi vsa vozlišča. *Hamiltonov cikel* je cikel, ki gre skozi vsa vozlišča. Odločitveni problem, ali obstaja Hamiltonov cikel, je NP-poln problem, imenovan *problem Hamiltonovega cikla*. Problem trgovskega potnika lahko formuliramo kot iskanje najcenejšega Hamiltonovega cikla. *Obhod* v grafu (nestandardno) definiramo kot Hamiltonov cikel v grafu. *Obhod trgovskega potnika* ali krajše *optimalni obhod* je najcenejši Hamiltonov cikel. Hamiltonov cikel lahko najdemo, če nanj gledamo kot na problem trgovskega potnika. Če iščemo Hamiltonov cikel v grafu $H = (U, F)$, vsem povezavam v grafu predpišemo ceno 1. Nato graf dopolnimo do polnega grafa $H' = (U, F')$ in vsem dodanim povezavam predpišemo ceno 2. Obhod trgovskega potnika na dopolnjenem grafu H' dolžine $|U|$ je enak Hamiltonovemu ciklu v grafu H.

2.2.2 Linearni program

Naj bo M množica v $\mathbb{R}^n$. Množica M je *konveksna*, če za poljubni točki m_i in m_j iz množice M daljica, ki povezuje točki m_i in m_j, v celoti leži v množici M. *Konveksna ovojnica* množice M je najmanjša konveksna množica, ki vsebuje M.

Polieder je presek končnega števila (zaprtih) polprostorov, t.j. množica rešitev končnega sistema linearnih neenačb. *Politop* je konveksna ovojnica končne množice točk. Vsak politop je polieder in vsak omejen polieder je politop. *Faceta* simplicialnega kompleksa je maksimalni simpleks. V splošnem v teoriji poliedrov in politopov poznamo dve definiciji. *Faceta poliedra* je ka-

terikoli mnogokotnik, katerega vogali so oglišča poliedra. *Faceta n-politopa* je $(n-1)$ razsežna stranska ploskev. Kocka je primer poliedra, katerega facete so njegove stranske ploskve.

Simpleksna metoda [16] je metoda, ki jo uporabljamo za reševanje linearnih programov standardne oblike:

$$\begin{aligned} \max \quad & c^T x \\ \text{p.p.} \quad & Ax \leq b, \\ & x_i \geq 0 \end{aligned} \tag{2.1}$$

kjer je $x = (x_1, \ldots, x_n)$ spremenljivka, $c = (c_1, \ldots, c_n)$ koeficienti kriterijske funkcije, $A \in \mathbb{R}^{p \times n}$ matrika in $b = (b_1, \ldots, b_p)$. Če so vsi $b_j \geq 0$, je linearni program zagotovo dopusten. Vsak linearni program lahko brez izgube splošnosti pretvorimo v standardno obliko. Geometrijsko gledano je množica dopustnih rešitev $F = \{\, x \mid Ax \leq b, \forall i : x_i \geq 0 \,\}$ konveksen politop. Če katera od spremenljivk x_i ni navzgor omejena, je množica F neomejen konveksen politop. Točka $x = (x_1, \ldots, x_n) \in F$ je ekstremna točka oz. oglišče politopa natanko takrat, ko je podmnožica stolpičnih vektorjev A_i, ki pripadajo neničelnim koordinatam x, linearno neodvisna. Tako točko imenujemo *bazična dopustna rešitev*.

Poglejmo si primer. Če iščemo minimalno vrednost funkcije $2x_1 + x_2 + 3x_3$ na območju

$$\{\, (x_1, x_2, x_3) \mid x_1 \in [0,1] \wedge x_2 \in [0,1] \wedge x_3 \in [0,1] \,\},$$

lahko problem zapišemo kot linearni program

$$\begin{aligned} \max \quad & -2x_1 - x_2 - 3x_3 \\ \text{p.p.} \quad & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \\ & x_1 \geq 0, x_2 \geq 0, x_3 \geq 0 \end{aligned} \tag{2.2}$$

in ga rešimo s simpleksno metodo. Množica dopustnih rešitev je kocka z oglišči $(0,0,0)$, $(1,0,0)$, $(0,1,0)$, $(0,0,1)$, $(1,1,0)$, $(1,0,1)$, $(0,1,1)$ in $(1,1,1)$.

Celoštevilski linearni program je optimizacijski problem, podoben linearnemu programu, pri čemer za vrednosti spremenljivk dodatno zahtevamo, da so cela števila. Poznamo tudi *mešani celoštevilski linearni program*, kjer so le nekatere spremenljivke omejene na podmnožico celih števil. *Linearna relaksacija* ali *relaksacija* je linearni program, ki ga dobimo iz celoštevilskega (mešanega) linearnega programa, če zanemarimo omejitve celoštevilskosti. Vsaka dopustna rešitev (mešanega) celoštevilskega linearnega programa je tudi dopustna rešitev njegove relaksacije.

2.2.3 Iskalna drevesa

Dvojiško ali *binarno drevo* je v računalništvu drevesna podatkovna struktura, kjer ima vsako vozlišče največ dva otroka. Po navadi se otroka imenujeta levi in desni sin. Če dvojiško drevo opazujemo kot graf, ima ta stopnjo največ tri. Podatkovna struktura *iskalno drevo* je posebna vrsta dvojiškega drevesa. Iskalna drevesa so namenjena predstavitvi podatkov, po katerih želimo učinkovito iskati. Iskalno drevo ima v vozliščih shranjene podatke, označene s ključi, ki jih lahko primerjamo po velikosti. Ključ je del podatka, lahko je celo ves podatek. V iskalnem drevesu velja, da je ključ v vsakem vozlišču večji od vseh ključev v levem poddrevesu in manjši od vseh ključev v desnem poddrevesu. V iskalnem drevesu lahko podatke vstavljamo, iščemo, obnovimo in (redkeje) brišemo. Vsa iskalna drevesa niso enako dobra. Drevo, ki je zelo globoko in „suho“, je slabo, nizko in „debelo“ drevo pa je dobro. Običajno poskušamo obdržati drevo uravnoteženo. Eno od rešitev ponujajo *uravnotežena drevesa* npr. *AVL drevesa*. *Iskalno dvojiško drevo* ponavadi zgradimo takrat, kadar potrebujemo hitro iskanje v množici urejenih elementov — v že zgrajenem dvojiškem drevesu, ki je približno uravnoteženo, je čas iskanja logaritemsko odvisen od števila elementov v drevesu.

Poglavje 3

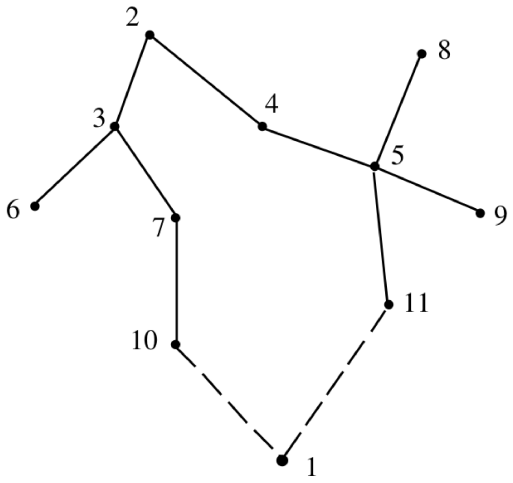
LKH - iskanje zgornje meje

V tem poglavju bomo opisali hevristični algoritem Lin-Kernighan (LK) za iskanje zgornje meje obhoda trgovskega potnika. Nadalje bomo opisali algoritem Lin-Kernighan-Helsgaun (LKH), ki je izboljšana verzija algoritma LK. Pri obeh algoritmih bomo opisali idejo in orisali implementacijo.

3.1 Hevristika LK

Naj bo $G = (V, E)$ neusmerjen utežen graf. *Drevo* je povezan graf brez ciklov. *Vpeto drevo* grafa G z n vozlišči je drevo z $n-1$ povezavami iz G. *Minimalno vpeto drevo* je vpeto drevo z najmanjšo vsoto dolžin povezav. 1-*drevo grafa* G je vpeto drevo na množici vozlišč $V \setminus \{ v_i \}$ z dodatnima povezavama iz E, ki se dotikata vozlišča v_i. Opazimo, da 1-drevo ni drevo, saj vsebuje cikel (glej Sliko 3.1). *Minimalno* 1-*drevo* je 1-drevo z najmanjšo vsoto dolžin povezav. *Stopnja vozlišča* je število povezav, ki so vezane na vozlišče. Lahko je videti, da je optimalni obhod enak najcenejšemu 1-drevesu, v katerem ima vsako vozlišče stopnjo dva. Prav tako je lahko videti, če je minimalno 1-drevo obhod, potem je obhod optimalen. TSP lahko sedaj formuliramo kot: Poišči najcenejše 1-drevo, katerega vsa vozlišča imajo stopnjo dva.

Za λ-opt algoritem velja, da je obhod λ-*optimalen* (ali λ-*opt*), če z zamenjavo katerihkoli λ mnogo povezav ne moremo dobiti krajšega obhoda. Iz



Slika 3.1: Primer 1-drevesa. Črtkani povezavi sta dodatni povezavi.

zgornje definicije je jasno razvidno, da je vsak λ-opt obhod tudi λ'-opt za vsako vrednost λ' , ki ne presega λ. Prav tako je razvidno, da je obhod, ki gre skozi n mest, optimalen natanko takrat, ko je n-opt. Testiranje λ-zamenjave ima v naivni implementaciji časovno zahtevnost $O(n^\lambda)$, zato se v praksi največkrat uporabljata $\lambda = 2$ in $\lambda = 3$. Poleg tega je algoritem λ-opt vezan na nespremenljivo število λ, ki ga izberemo na začetku. Lin in Kernighan sta uvedla algoritem LK, λ-opt algoritem s spremenljivo vrednostjo λ (Algoritem 1). Algoritem LK na vsakem koraku iteracije preveri, katero vrednost λ se splača uporabiti. Začetna vrednost je vedno $\lambda = 2$, naslednja vrednost λ je za ena večja od prejšnje. Vrednost λ zvišuje, dokler ne najde zamenjave, ki vrne boljšo rešitev, ali brez uspeha preizkusi vse dovoljene zamenjave. V prvem primeru algoritem ponastavi vrednost λ na 2 in začne novo iteracijo. V drugem primeru pa je dobljena rešitev lokalni (lahko, da tudi globalni) minimum. Algoritem LK vrednost λ zvišuje do največ 3.

Najdeni zamenjavi parov množic povezav (F, H) v Algoritmu 1 rečemo *r-opt premik*. *Nedopusten r-opt premik* je premik pri katerem ne dobimo dopustnega obhoda.

Množici F in H gradimo postopoma, začnemo s praznima množicama,

Algoritem 1 Lin-Kernighan (LK)

1: $x \leftarrow$ začetni obhod
2: **repeat**
3: **repeat**
4: preizkušaj pare množic povezav $F = \{ f_1, \ldots, f_r \}$ in $H = \{ h_1, \ldots, h_r \}$
5: **until** z zamenjavo povezav F s povezavami H v obhodu x, dobimo boljši obhod x' ali preizkusimo vse dopustne pare
6: **if** najdemo ustrezni množici F in H **then**
7: $x = x'$ in $r = 2$
8: **else**
9: $r = r + 1$
10: **end if**
11: **until** so izpolnjeni ustaitveni pogoji

katerima dodajamo povezave parov (f_i, h_i). Za boljše delovanje algoritma, lahko v množici dodajamo le povezave, ki zadoščajo naslednjim kriterijem:

1. kriterij zaporedne menjave:
   Povezavi f_i in h_i imata skupno vozlišče, enako velja za povezavi h_i in f_{i+1}. Zamenjava je zaporedna, če je veriga $(f_1, h_1, f_2, h_2, \ldots, f_r, h_r)$, ki jo tvorijo povezave F in H sklenjena, t.j. $h_r = (t_{2r}, t_1)$. S tem kriterijem dovoljujemo le zaporedne zamenjave.

2. kriterij dopustnosti:
   Naj bo $f_i = (t_{2i-1}, t_{2i})$ povezava iz F za $i \geq 3$. Če povežemo točki t_{2i} in t_1, dodane in odstranjene povezave tvorijo cikel.

3. kriterij pozitivnega dobička:
   Naj bo $g_i = c(f_i) - c(h_i)$ dobiček zamenjave f_i z h_i. Potem je vsak delni dobiček vsote izmenjav $G_i = g_1 + g_2 + \cdots + g_i$ pozitiven.

4. kriterij disjkunktnosti:
   Množici F in H sta disjunktni. Ta pogoj poenostavi zapis algoritma, zmanjša čas delovanja in je učinkovit pogoj ustavitve.

Tako omejimo iskalno območje na množice povezav, ki bolj verjetno izboljšajo obhod. Iskanje povezav, ki jih dodamo v množici F in H, je ozko grlo

algoritma. Zato sta Lin in Kernighan še bolj omejila iskalno območje. Poleg pravil 1–4 sta uvedla naslednjih pet dodatnih pravil:

5. Iskanje povezave $h_i = (t_{2i}, t_{2i+1})$ je omejeno na pet najbližjih sosedov t_{2i}.

6. Če se povezava e nahaja v nekaj prejšnjih (tipično dva do pet) obhodih, jo lahko odstranimo le, če je ena izmed prvih treh odstranjenih povezav.

7. Iskanje boljših obhodov se zaključi, če je trenutno najkrajši obhod enak zadnjemu obhodu.

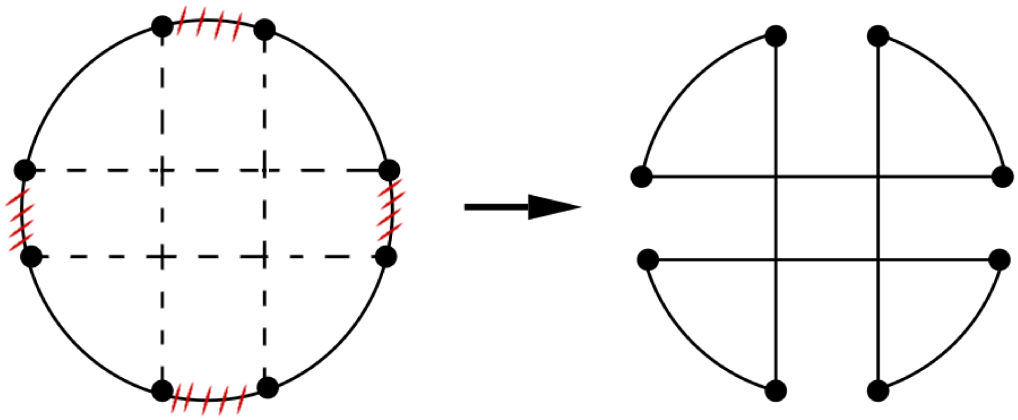
8. Pri izbiri povezave h_i za $i \geq 2$ vsaka izbira dobi prednostno vrednost $c(f_{i+1}) - c(h_i)$, t.j. raje izberemo lažje povezave.

9. Če za f_4 (algoritem LK zamenja največ 4 povezave) obstajata dve možnosti, izberemo tisto z višjo vrednostjo $c(f_4)$, t.j. raje odstranimo težje povezave.

Pravila 5–7 še bolj omejijo iskalno območje, pravili 8 in 9 pa usmerjata iskanje. Pravilo 6 krši znani tabu iskanja po katerem se nočemo vračati k prejšnjim rešitvam, a je potrebno, da odrežemo iskanje in tako omejimo iskalno območje. Poleg novih pravil sta Lin in Kernighan uvedla še omejeno obrambo pred primeri, kjer do krajšega obhoda lahko pridemo le preko nezaporednih zamenjav. Ko najdemo lokalni optimum, algoritem LK poskuša obhod izboljšati z nezaporedno 4-opt zamenjavo povezav, ki jih lahko zamenja (glej Sliko 3.2).

3.2 Dopolnitev LK do LKH

Pravilo 5 iz razdelka 3.1 usmeri iskanje in zmanjša zahtevnost algoritma, a se lahko zgodi da, ravno zaradi tega pravila, optimalne rešitve ne najdemo. Če optimalna rešitev vsebuje povezavo, katere vozlišči nista povezani z enim od petih najbližjih sosedov, bo algoritem težko našel optimalno rešitev. Pravilo je osnovano na ideji, da krajše kot so povezave, večja je verjetnost, da



Slika 3.2: Primer nezaporedne 4-opt zamenjave.

so del optimalne poti. Helsgaun je razvil algoritem Lin-Kernighan-Helsgaun (LKH), ki ohrani to idejo, vendar definira novo mero za bližino. Nova mera je imenovana α-bližina in je osnovana na analizi občutljivosti z uporabo najmanjših vpetih 1-dreves.

Keld Helsgaun izpostavi, da optimalni obhod ponavadi vsebuje med 70% in 80% povezav minimalnega 1-drevesa [15]. Povezave, ki bolj verjetno pripadajo minimalnemu 1-drevesu, imajo večjo verjetnost, da pripadajo optimalnemu obhodu. Formalno novo mero bližine definiramo takole:

Naj bo T minimalno 1-drevo dolžine $L(T)$ in naj bo $T^+(v_i, v_j)$ minimalno 1-drevo, ki vebuje povezavo (v_i, v_j). Potem α-bližino povezave (v_i, v_j) označimo z $\alpha(v_i, v_j)$ in jo definiramo kot $\alpha(v_i, v_j) = L(T^+(v_i, v_j)) - L(T)$.

Za α-bližino veljata naslednji lastnosti:

- $\alpha(v_i, v_j) \geq 0$,
- če (v_i, v_j) pripada minimalnemu 1-drevesu, potem je $\alpha(v_i, v_j) = 0$.

Z α-bližino lahko ocenimo, ali povezava pripada optimalnemu obhodu. Manjša kot je vrednost $\alpha(v_i, v_j)$, večja je verjetnost, da je povezava (v_i, v_j) del optimalnega obhoda. Iskanje povezav tako lahko omejimo na α-najbližjih sosedov. Že sama α-bližina dobro omeji iskalni prostor a jo lahko še izboljšamo s

preprosto modifikacijo matrike uteži na povezavah. Modifikacijo prilagodimo naslednjim opažanjem:

1. Vsak obhod je 1-drevo, zato je dolžina minimalnega 1-drevesa spodnja meja za optimalni obhod.

2. Minimalno 1-drevo T ima lahko vozlišča velikih stopenj. Če je vozlišče v velike stopnje v T, potem vse cene povezav s krajšim v povečamo za isto vrednost w. S tem obhoda trgovskega potnika ne spremenimo, le njegovo dolžino povečamo za $2w$. Spremeni pa se dolžina drevesa T za vrednost $w \deg_T(v)$. S primerno izbiro w lahko dosežemo, da nova stopnja vozlišča v v poceni 1-drevesu pade. Novo matriko uteži na povezavah označimo z D.

3. Če je T_w minimalno 1-drevo glede na D, potem je njegova dolžina $L(T_w)$ spodnja meja optimalnega obhoda za D

Cilj je poiskati transformacijo $C \to D$, ki maksimizira spodnjo mejo dolžine optimalnega obhoda za C.

Algoritem LKH tako namesto petih najbližjih sosedov vzame pet α-najbližjih sosedov glede na transformirano matriko uteži. Če imata dve povezavi enako α-vrednost, je bližja tista, ki ima manjšo začetno utež. Povezave do pet α-najbližjih sosedov imenujemo *kandidatne povezave*. Da bi pospešili proces iskanja, algoritem LKH dinamično ureja kandidatne povezave. Vsakič, ko najdemo krajši obhod, algoritem LKH povezave, ki se pojavijo v prejšnjem in trenutnem najkrajšem obhodu, uvrsti na prvo mesto v lestvici kandidatnih povezav.

Algoritem LK odreže iskanje na stopnji 4 ali več, t.j. po tretji izbrani odstranjeni povezavi, in to samo, če povezava, ki jo želimo odstraniti, ni skupna številu (2-5) obhodov (glej pravilo 6 v razdelku 3.1). Algoritem LKH uvede novo pravilo, ki odreže iskanje že na prvi stopnji. Pravilo pravi, da prva povezava, ki jo želimo odstraniti, ne sme pripadati trenutno najboljšemu obhodu. Ko trenutnega najboljšega obhoda ne poznamo, povezava ne sme pripadati minimalnemu 1-drevesu.

Algoritem LKH nadomesti tudi pravilo 4 z njegovo relaksacijo, ki pravi, da zadnja povezava, ki jo želimo izključiti, ne sme biti med povezavami, ki smo jih predhodno dodali.

Algoritem LK uporablja le 2-opt, 3-opt in v dveh posebnih primerih 4-opt zamenjave. Algoritem LKH pa uporablja tudi 5-opt zamenjave. Algoritem LKH sproti preverja, če zaprtje trenutnega novega obhoda izboljša trenutno najboljši obhod. Če je temu tako, algoritem LKH iskanje novega obhoda zaključi. Uporaba 5-opt zamenjav sicer poveča čas iskanja, a zaradi dobre omejitve iskalnega prostora ta ni bistveno različen od časa iskanja pri algoritmu LK. Algoritem LKH uvaja tudi splošnejši način izhoda iz lokalnega optimuma. Namesto enostavnega 4-opt premika uporabi močnejši nabor dveh nezaporednih premikov, ki vsebuje:

- Katerikoli nedopusten 2-opt premik, ki mu sledi katerikoli 2- ali 3-opt premik, ki vrne dopusten obhod.
- Katerikoli nedopusten 3-opt premik, ki mu sledi katerikoli 2-opt premik, ki vrne dopusten obhod.

V algoritmu LK začetni obhod izberemo naključno. Čeprav kvaliteta rešitve, ki jo dobimo z algoritmom LKH, v praksi ni odvisna od prvotnega obhoda, dober začetni obhod znatno zmanjša čas računanja. Algoritem LKH zato uporablja enostavno hevristiko za izračun začetnega obhoda (Algoritem 2). Ta algoritem lahko začne ali brez znanega obhoda ali pa z na kakršenkoli način pridobljenim obhodom. Obhod, ki ga Algoritem 2 vrne, bo dober začetni obhod za nadaljevanje LKH lokalne optimizacije.

Najprej izberemo naključno vozlišče v. Če smo algoritmu LKH podali že obstoječi obhod, želimo vozlišče v povezati s tistim vozliščem v', za katerega povezava (v, v') pripada danemu obhodu, je kandidatna povezava in je vrednost $\alpha(v, v')$ enaka 0. Če obhoda nismo podali, želimo vozlišče v povezati s tistim vozliščem v', za katerega je povezava (v, v') kandidatna povezava. Če takega vozlišča ni, vozlišče v povežemo z naključno izbranim vozliščem v'. Nadaljujemo z vozliščem v'. Postopek ponavljamo, dokler ne izberemo vseh

točk v grafu G.

Algoritem 2 Hevristika za začetni obhod v LKH

1: naključno izberi vozlišče v
2: **repeat** {izbiramo novo vozlišče v'}
3: **if** obstaja vozlišče v', za katerega je (v, v') kandidatna povezava, $\alpha(v, v') = 0$ in (v, v') pripada vhodnemu obhodu **then**
4: izberi vozlišče v'
5: **else if** obstaja vozlišče v', za katerega je (v, v') kandidatna povezava **then**
6: izberi vozlišče v'
7: **else**
8: izberi naključno vozlišče v', ki še ni med izbranimi vozlišči
9: **end if**
10: naj bo $v = v'$
11: **until** izbrana vsa vozlišča

Algoritem LKH je učinkovita implementacija hevristike Lin-Kernighan, ki z zamenjavo od dva do pet povezav krajša obhod v danem grafu. Algoritem se na vsakem koraku odloči, koliko povezav bo zamenjal. Povezave za zamenjavo so izbrane po skrbno predpisanih kriterijih, ki zagotavljajo hitro in učinkovito iskanje ustreznih povezav. Eden od kriterijev za povezave, ki jih želimo dodati, je α-bližina, ki je osnovana na analizi občutljivosti z uporabo najmanjših vpetih 1-dreves. Manjša kot je vrednost α-bližine, večja je verjetnost, da je povezava del optimalnega obhoda. Algoritem LKH vrne obhod in z njim zgornjo mejo obhoda trgovskega potnika.

Poglavje 4

Concorde - iskanje spodnje meje

V tem poglavju bomo opisali idejo in orisali implementacijo algoritma Concorde, s katerim iščemo in iterativno višamo spodnjo mejo obhoda trgovskega potnika.

4.1 Klasično reševanje

Naj bo $G = (V, E)$ poln neusmerjen graf z $n = |V|$ vozlišči in $m = |E|$ povezavami. Za $W \subset V$ z $E(W)$ označimo množico povezav med vozlišči iz W. Naj bo $x \in [0, 1]^m$ vektor indeksiran s povezavami grafa G. Z x_e označimo koordinato, ki ustreza povezavi e. Koordinato x_e interpretiramo kot delež prisotnosti povezave e, $x_e = 1$ pomeni popolno prisotnost, $x_e = 0$ pa popolno odsotnost povezave e. Za $F \subseteq E$ definiramo vsoto deležev povezav $x(F)$ na podmnožici povezav. Vsoto izračunamo kot $x(F) = \sum_{e \in F} x_e$. Vektor x lahko interpretiramo kot podgraf grafa G, ki vsebuje vse točke V in tiste povezave $e \in E$, za katere je $x_e > 0$.

Naj bo $x_e \in \{0, 1\}$ indikator prisotnosti povezave e in $\mathcal{S}$ množica vseh obhodov. Poiskati rešitev problema trgovskega potnika je enako kot poiskati

$$\min_{x \in \mathcal{S}} \sum_{e \in E} c_e x_e = \min_{x \in \mathcal{S}} c^T x. \tag{4.1}$$

Program Concorde namesto opisane minimizacije rešuje sorodni problem:

$$\begin{aligned} \min \quad & c^T x \\ \text{p.p.} \quad & Ax \leq b, \end{aligned} \tag{4.2}$$

z ustrezno izbranim sistemom linearnih neenačb $Ax \leq b$, ki ga izpolnjuje vsak $x \in \mathcal{S}$. Optimalna vrednost linearnega programa (4.2) je spodnja meja za optimalno vrednost problema (4.1). Ena od značilnosti linearnega programa je, da je optimalna rešitev x^* linearnega programa (4.2) hkrati ekstremna točka poliedra, ki ga definirajo neenačbe $Ax \leq b$. Natančneje, če x^* ne pripada $\mathcal{S}$, potem leži zunaj konveksne ovojnice $\mathcal{S}$. V tem primeru lahko x^* ločimo od $\mathcal{S}$ s hiperravnino, t.j. linearno neenačbo, ki ji zadoščajo vse točke iz $\mathcal{S}$ in jo krši x^*. Tovrstno neenakost imenujemo presečna ravnina. Presečno ravnino nato dodamo sistemu $Ax \leq b$ in s tem pridobimo tesnejšo relaksacijo, ki jo zopet rešimo. Proces ponavljamo dokler rešitev relaksacije ne pripada $\mathcal{S}$.

Ker ima vsako vozlišče v obhodu natanko dve povezavi, je konveksna ovojnica množice $\mathcal{S}$ (označimo jo z Q^n) vsebovana v politopu

$$Q^n_A = \left\{ x \in \mathbb{R}^E \mid Ax = \mathbf{2}, \mathbf{0} \leq x \leq \mathbf{1} \right\},$$

kjer je A incidenčna matrika grafa G, odebeljeno število $\mathbf{2}$ predstavlja vektor dolžine m, katerega komponente so vse enake 2 (podobno za ostala odebeljena števila). Po izreku Weyla in Minkowskega [17, 18], je podmnožica R evklidskega prostora konveksni politop natanko takrat, ko je R omejen polieder. Zato lahko Q^n zapišemo kot sistem linearnih neenačb

$$Q^n = \left\{ x \in Q^n_A \mid \ell x \leq \ell_0 \text{ za vse } (\ell, \ell_0) \in \mathcal{L} \right\},$$

kjer je $\mathcal{L}$ končna družina linearnih neenakosti, ki jih lahko izberemo tako, da vsaka neenakost inducira drugo faceto Q^n. Linearni program (4.2) lahko

sedaj prepišemo v

$$\begin{aligned} \min \quad & c^T x \\ \text{p.p.} \quad & Ax = \mathbf{2}, \\ & \ell x \leq \ell_0 \text{ za vse } (\ell, \ell_0) \in \mathcal{L}, \\ & \mathbf{0} \leq x \leq \mathbf{1}. \end{aligned} \tag{4.3}$$

Družina $\mathcal{L}$ je zelo velika in je ne poznamo v celoti, zato ponavadi uporabimo le del družine, označimo ga z $\overline{\mathcal{L}}$. Ena od poddružin družine $\mathcal{L}$ so neenačbe podobhodov. Te zagotavljajo, da rešitev ne vsebuje podobhodov. Linearni program (4.3) je ekvivalenten celoštevilskemu linearnemu programu

$$\begin{aligned} \min \quad & c^T x \\ \text{p.p.} \quad & Ax = \mathbf{2}, \\ & x(E(W)) \leq |W| - 1 \text{ za vse } \emptyset \subset W \subset V, \\ & \mathbf{0} \leq x \leq \mathbf{1}, \\ & x \in \mathbb{Z}, \end{aligned} \tag{4.4}$$

kjer je drugi pogoj družina neenačb podobhodov, označimo jo z $\overline{\mathcal{L}}$. Število neenakosti v $\overline{\mathcal{L}}$ je preveliko, da bi vključili vse naenkrat. Zato namesto sistema (4.4) rešimo iterativni Algoritem 3, ki se zaključi po končnem številu

Algoritem 3 Iterativni postopek

1: $\mathcal{L}' = 0 \leftarrow$ inicializiraj poddružino $\mathcal{L}$
2: reši sistem za $\mathcal{L} = \mathcal{L}'$, rešitev sistema označimo z x^*
3: **if** x^* ne krši nobene neenakosti iz $\overline{\mathcal{L}}$ **then**
4: končaj
5: **else**
6: dodaj kršene neenakosti v $\mathcal{L}'$ in ponovi postopek od koraka 2
7: **end if**

iteracij, saj je družina $\overline{\mathcal{L}}$ končna. Rešitev sistema v posamezni iteraciji je bodisi rešitev TSP bodisi množica podobhodov. V slednjem primeru obstaja $W \subset V$, ki krši neenakost $x^*(E(W)) \leq |W|-1$. Kršeno neenakost za najdeno množico vozlišč dodamo med pogoje linearnega programa in nadaljujemo postopek. Opisani način reševanja TSP sta prvič uporabila Crowder in Padberg

[19], poznamo ga kot klasični način reševanja tovrstnih optimizacijskih problemov. Osrednji problem reševanja Algoritma 3 je poiskati neenakosti, ki jih krši rešitev x^*. Ta problem poznamo pod imenom Problem separacije:

Problem 1 *(Problem separacije) Za dano točko $x \in \mathbb{R}^E$ in $\mathcal{L}'$ družino neenakosti v $\mathbb{R}^E$, poišči eno ali več neenakosti iz $\mathcal{L}'$, ki jih krši x ali dokaži, da tovrstna neenakost ne obstaja.*

Prednost klasičnega reševanje je, da lahko za reševanje linearnega sistema v posamezni iteraciji uporabimo katerikoli komercialni program tipa razveji in omeji. Z uporabo tovrstnega programa strogo ločimo fazi iskanja krštev neenakosti (faza presečnih ravnin) in reševanja sistema (faza številčenja) v iteraciji. V fazi presečnih ravnin poiščemo neenakosti, ki jih krši rešitev, ki smo jo dobili v fazi številčenja. Najdene neenakosti dodamo linearnemu sistemu. Naslednja faza številčenja ne bo upoštevala rešitve pridobljene v prejšnji fazi številčenja, čeprav bi ta lahko bila blizu optimalni rešitvi. Metoda razveji in odreži poveže fazi presečnih ravnin in številčenja.

4.2 Metoda razveji in odreži

Družino $\overline{\mathcal{L}}$ razširimo z drugimi tipi neenačb, ki jih bomo opisali kasneje. Za dano množico $\mathcal{L}' \subset \overline{\mathcal{L}}$ neenakosti, ki inducirajo faceto Q_n in disjunktni množici povezav $F_0, F_1 \subset E$ definiramo linearni program $P(\mathcal{L}', F_0, F_1)$

$$\begin{aligned} \min \quad & c^T x \\ \text{p.p.} \quad & Ax = \mathbf{2}, \\ & \ell x \leq \ell_0 \text{ za vse } (\ell, \ell_0) \in \mathcal{L}, \\ & x_e = 0 \text{ za vse } e \in F_0, \\ & x_e = 1 \text{ za vse } e \in F_1, \\ & \mathbf{0} \leq x \leq \mathbf{1}. \end{aligned} \tag{4.5}$$

Tudi ta linearni program rešujemo z Algoritmom 3. Rešitev postopka je rešitev TSP ali pa krši neko neznano neenakosti iz $\mathcal{L} - \overline{\mathcal{L}}$. V prvem primeru smo rešili linearni program (4.3), v drugem primeru pa smo našli

spodnjo mejo linearnega programa (4.3), ki jo lahko nadalje uporabimo za iskanje optimalne rešitve. Metoda razveji in omeji, Algoritem 4, je različica Algoritma 3 s katero rešimo linearni program (4.5).

Z $\mathcal{F}$ označimo družino urejenih parov (F_0, F_1) disjunktnih množic povezav, z $\overline{x}$ pa označimo vektor obhoda v grafu G. Množica F_1 predstavlja vse povezave e, ki morajo biti v celoti prisotne v rešitvi ($x_e = 1$). Množica F_0 pa predstavlja vse povezave e, ki zagotovo niso prisotne v rešitvi ($x_e = 0$). Na koraku 5 Algoritma 4 odstranimo urejeni par (F_0, F_1) iz množice $\mathcal{F}$. Odstranjenega para nikoli več ne dodamo v $\mathcal{F}$. Opišimo delovanje algoritma razveji in odreži s terminologijo, ki se uporablja pri algoritmu razveji in omeji. Vozlišče v drevesu, ki je vezano na urejen par (F_0, F_1), predstavlja problem $P(\mathcal{L}, F_0, F_1)$, t.j. problem iskanja najkrajšega Hamiltonovega cikla, katerega rešitev zadošča enačbam

$$\begin{aligned} x_e &= 0 \text{ za vse } e \in F_0 \\ x_e &= 1 \text{ za vse } e \in F_1. \end{aligned} \tag{4.6}$$

Vozlišče je rešeno, če je rešitev x^* celoštevilska. Če rešitev x^* ni celoštevilska in na koraku 16 ne najdemo nobene kršitve neenakosti, algoritem izbere spremenljivko razvejanja in problem razveji na dve novi veji. Tako dobimo novi vozlišči $(F_0 + \{e\}, F_1)$ in $(F_0, F_1 + \{e\})$, ki sta otroka vozlišča (F_0, F_1). Vozlišča generirana z Algoritmom 4 tvorijo iskalno drevo globine največ m. Zato in ker je $\mathcal{L}$ končna množica, se algoritem zaključi v končno mnogih korakih. Koren iskalnega drevesa je vozlišče $(\emptyset, \emptyset)$. Vozlišča iskalnega drevesa, ki so vsebovana v $\mathcal{F}$ so aktivna vozlišča. Linearni program $P(\mathcal{L}', F_0, F_1)$ je posebna vrsta relaksacije problema $P(\mathcal{L}, F_0, F_1)$, saj je vsaka neenakost iz $\mathcal{L}'$ ne le validna ampak tudi inducira facete za celoten politop $\mathcal{Q}^n$. To dovoljuje vozliščem iskalnega drevesa, da si delijo isto množico $\mathcal{L}'$, kar je ogromna prednost pred tradicionalnim pristopom presečnih ravnin/razveji in omeji v smislu implementacije in alokacije spomina. Poleg tega z rešitvijo vozlišča izboljšamo tudi relaksacijo povezano z vsemi ostalimi vozlišči.

Delovanje Algoritma 4 je močno odvisno od začene rešitve $\overline{x}$. Nižja kot je vrednost zgornje meje, manjše je število vozlišč v iskalnem drevesu, saj veje s

Algoritem 4 Razveji in odreži

1: $\mathcal{F} = \{< \emptyset, \emptyset >\}$, $\mathcal{L}' = \emptyset \leftarrow$ inicializacija
2: **if** $\mathcal{F} = \emptyset$ **then**
3: zaključi
4: **else**
5: izberi urejeni par (F_0, F_1) iz $\mathcal{F}$ in zamenjaj $\mathcal{F}$ z $\mathcal{F} - (F_0, F_1)$
6: **end if**
7: reši linearni program $P(\mathcal{L}', F_0, F_1)$
8: **if** obstaja rešitev linearnega programa **then**
9: naj bo x^* optimalna rešitev
10: **else**
11: pojdi na korak 2
12: **end if**
13: **if** $cx^* \geq c\overline{x}$ **then**
14: pojdi na korak 2
15: **end if**
16: poišči eno ali več neenakosti iz $\mathcal{L}$, ki jih krši x^*
17: **if** x^* ne krši nobene neenakosti iz $\mathcal{L}$ **then**
18: pojdi na korak 22
19: **else**
20: dodaj kršene neenakosti v $\mathcal{L}'$ in ponovi postopek od koraka 2
21: **end if**
22: **if** x^* celoštevilska rešitev **then**
23: zamenjaj $\overline{x}$ z x^* in pojdi na korak 2
24: **end if**
25: izberi povezavo $e \in E$, za katero velja $0 < x_e^* < 1$. Zamenjaj $\mathcal{F}$ z $\mathcal{F} + (F_0 + e, F_1) + (F_0, F_1 + e)$ in pojdi na korak 2

preveliko spodnjo mejo odrežemo (glej Sliko 4.1, levo vozlišče 3). V ta namen ima Concorde implemenitran hevristični algoritem za iskanje zgornje meje, ki je osnovan na hevristiki izmenjav (Lin Kernighan) in prilagojen za obsežne primere. V magistrskem delu smo namesto Concordovega hevrističnega algoritma uporabili hevristični algoritem LKH (glej poglavje 3).

Poiskati neenakosti, ki jih krši rešitev x^* na koraku 16 Algoritma 4 ni enostaven problem. Postopek reševanja tega problema imenujemo generator pogojev.

4.3 Generator pogojev

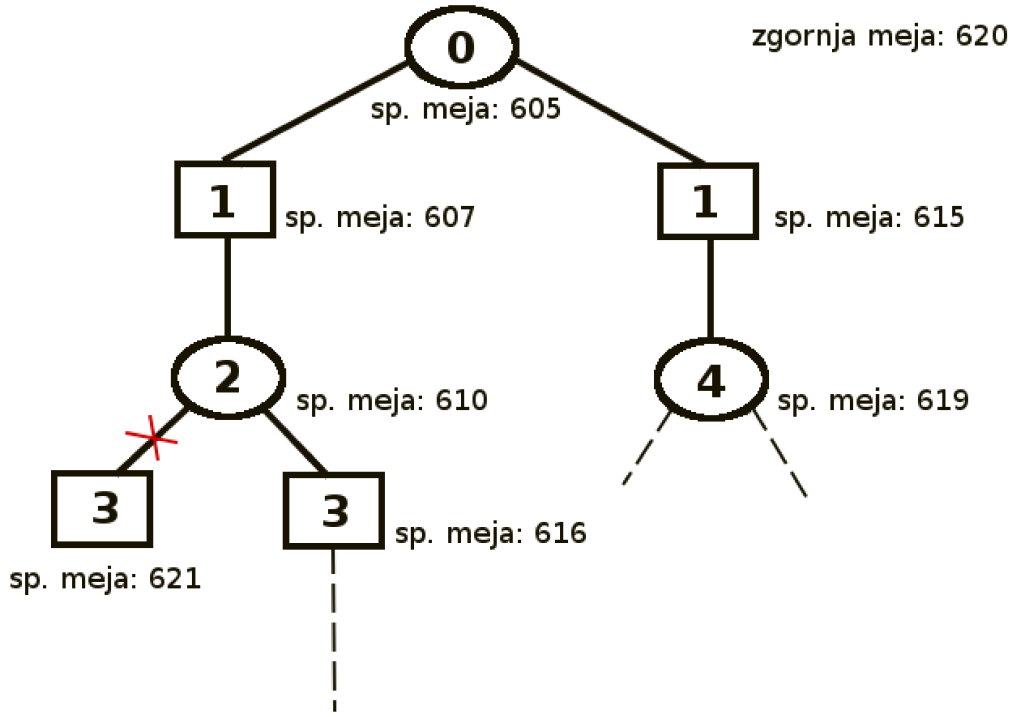
Najpomembnejši del algoritma poliedrskih presečnih ravnin je generator pogojev. Concorde implementira prirejeno različico znanega Rinaldijevega generatorja pogojev [20]. Boljši kot je generator pogojev na koraku 16 Algoritma 4, višja je vrednost cx^* na vsakem koraku 9. Posledično lahko število vozlišč v iskalnem drevesu drastično upade, če izboljšamo generator pogojev. V programu Concorde vse neenačbe inducirajo facete za $\mathcal{Q}^n$ in padejo v eno od štirih kategorij: neenačbe podobhodov, neenačbe 2-faktorizacije, neenačbe glavnika in neenačbe drevesa klik [21].

Dobljena rešitev x mora vsebovati le en cikel, ki zajema vse točke V. Zato mora za vsako množico $H \subset V$, veljati $x(E(H)) \leq |H| - 1$. V nasprotnem primeru je rešitev x na množici $E(H))$ cikel oz. cikel vsebuje in je zato x vsota podpoti (Slika 4.2c). Neenačbe oblike

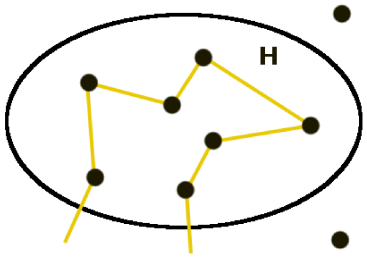
$$x(E(H)) \leq |H| - 1 \tag{4.7}$$

imenujemo neenačbe podobhodov.

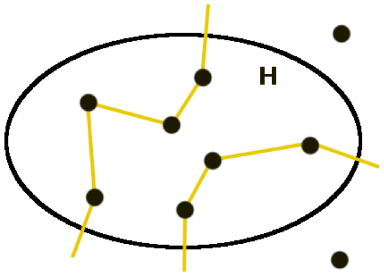
Za rešitev zahtevamo, da je za vsako vozlišče v vsota vseh vrednosti x_e povezav e, ki so povezane z vozliščem v enaka 2. Zato se lahko zgodi, da obstaja vozlišče v, ki je krajišče več kot dveh povezav prisotnih v rešitvi. Pogoj, ki ga dodamo, določa, da je vozlišče v krajišče natanko dveh povezav prisotnih v rešitvi. Ta pogoj je tipa neenačbe 2-faktorizacije, neenačbe glavnika ali neenačbe drevesa klik.



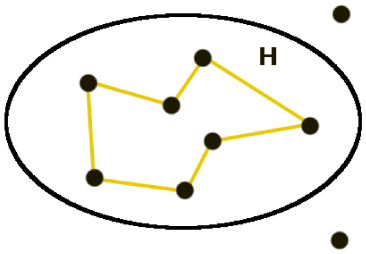
Slika 4.1: Shema delovanja programa Concorde. Ovalna vozlišča predstavljajo razvejanje, pravokotna pa reze. Številke v vozliščih označujejo zaporedje razvoja vozlišč v drevesu. V vsakem vozlišču Concorde na novo reši sistem linearnih neenačb, da dobi novo spodnjo mejo. Če spodnja meja v vozlišču u preseže zgornjo mejo, veje, ki gre iz vozlišča u, ne razvijamo naprej.



(a) Množica H zadošča neenačbi podobhodov.



(b) Množica H zadošča neenačbi podobhodov.



(c) Množica H ne zadošča neenačbi podobhodov.

Slika 4.2: Primeri neenačb podobhodov, glej (4.7).

Naj bo $H \subset V$ in $T_i \subset V$ za $i = 1, \ldots, t$. Množico H imenujemo ročaj, množice T_i pa zobje. Ročaj in zobje naj zadoščajo naslednjim pogojem:

- $|T_i| = 2$ in $|T_i - H| \geq 1$,
- vsak ročaj seka liho število (≥ 3) zob,
- število vseh zob t je liho,
- vsak par zob ima prazen presek.

Skozi točke ročaja in zob rešitev problema trgovskega potnika tvori eno samo pot (Slika 4.3a) ali pa drevo poti (Slika 4.3b). Največje število povezav dobimo, če imamo $\frac{t-1}{2}$ poti, ki skupaj zajemajo vse točke v H in vse točke v $t-1$ zobeh. Bolj natančno, če velja

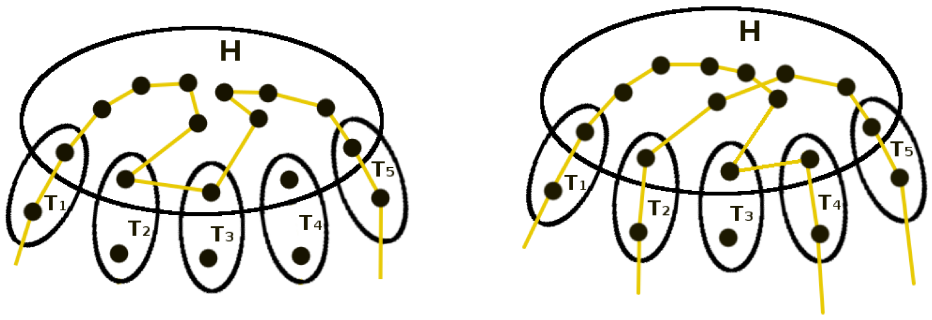
$$x(E(H)) + \sum_{i=1}^{t} x(E(T_i)) \leq |H| + (t-1) - \frac{t-1}{2} = |H| + \frac{t-1}{2}.$$

Neenačbe oblike $x(E(H)) + \sum_{i=1}^{t} x(E(T_i)) \leq |H| + \frac{t-1}{2}$ imenujemo neenačbe 2-faktorizacije. Neenačbe 2-faktorizacije so posebna oblika neenačb glavnika za katere dopuščamo, da je $|T_i| \geq 2$. V splošnem so neenačbe drevesa klik posebna oblika neenačb glavnika, kjer imamo lahko tudi več kot en ročaj. Concorde strogo loči neenačbe glavnika od neenačb drevesa klik, saj neenačbe drevesa klik definira za vsaj dva ročaja in vsaj pet zob. Vse štiri tipe neenačb (vključno z neenačbo podobhodov) lahko zapišemo s splošno formulo

$$\sum_{q=1}^{h} x(E(H_q)) + \sum_{i=1}^{t} x(E(T_i)) \leq \sum_{q=1}^{h} |H_q| + \sum_{i=1}^{t} (|T_i| - t_i) - \frac{t+1}{2},$$

kjer so množice H_q ročaji, T_i pa glavniki, za katere velja:

- $2 \leq |T_i| \leq n-2$ in $|T_i - \bigcup H_q| \geq 1$,
- vsak ročaj seka liho število (≥ 3) zob,
- t_i je število ročajev, ki sekajo zob T_i; število vseh zob t je liho,



(a) Primer ene same poti.

(b) Primer drevesa poti.

Slika 4.3: Primeri neenačb 2-faktorizacije.

- graf presekov sistema množic, ki ga podajajo ročaji in zobje, tvori drevo.

Če je $h = 1$ in $t = 0$, imamo neenačbo podobhodov. Če je $h = 1$ in $t \geq 3$ imamo neenačbo glavnika. Če dodamo še omejitev $|T_i| = 2$ imamo neenačbo 2-faktorizacije. Če je $h \geq 2$ in $t \geq 5$, imamo neenačbo drevesa klik (ki ni neenačba 2-faktorizacije).

4.4 Redukcija neenačb

Hitrost izvajanja Algoritma 4 je odvisna od števila neenakosti v $\mathcal{L}'$. Še več, odvisna je tudi od števila spremenljivk prisotnih v vsaki neenačbi v $\mathcal{L}'$. Manjše kot je število neenakosti, hitreje rešimo linearni program $P(\mathcal{L}', F_0, F_1)$ (4.5). Enako velja za število spremenljivk, ki nastopajo v vsaki neenakosti. Program Concorde ima zato implementiran postopek redukcije neenakosti, s katerim nadzorujemo število neenačb in število spremenljivk v njih. V ta namen definira mero za kvaliteto reza, imenovano vrednost odstopanja.

Naj bo x rešitev trenutnega linearnega programa, *vrednost odstopanja* neenačbe je vrednost desne strani neenačbe, ki ji odštejemo levo stran enačbe, pri tem pa upoštevamo vrednosti x. Negativna vrednost odstopanja pomeni, da je neenačbe kršena. Bolj kot je vrednost odstopanja negativna, boljši je rez. Posledično redukcija ohranja vrednost odstopanja karseda negativno,

hkrati pa zavrne redukcijo neenačbe, če je njena absolutna vrednost odstopanja premajhna. Za primer poglejmo redukciji neenačbe podobhodov. Naj bo

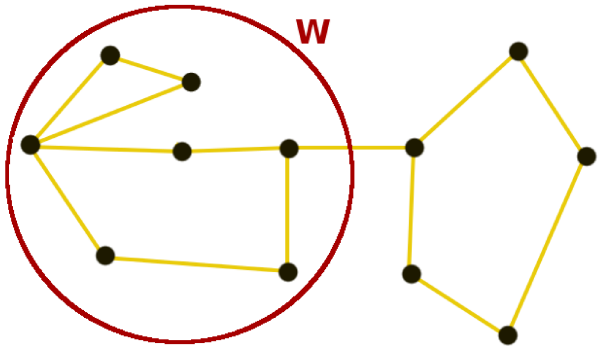
$$x(E(W)) \leq |W| - 1 \tag{4.8}$$

neenačba podobhodov za množico $W \subset V$, $3 \leq |W| \leq |V| - 3$. Če v neenačbi (4.8) nadomestimo W z $V - W$, dobimo ekvivalentno neenačbo, t.j. obe neenačbi definirata isto faceto v Q^n. Kadar je $|W| \geq \frac{n}{2}$, zamenjamo W z $V - W$ (glej Sliko 4.4b). Če neenačba (4.8) inducira povezan podgraf G^W, ki ni 2-povezan, podgraf G^W razbijemo na bloke. Naj bodo $W_1, W_2, \ldots, W_k$ množice vozlišč blokov podgrafa G^W. Potem velja

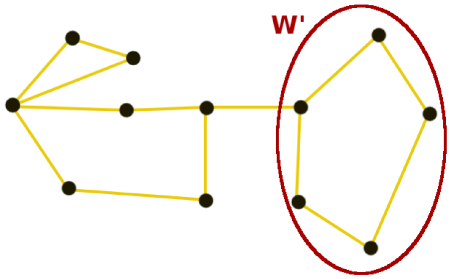
$$x(E(W)) = \sum_{i=1}^{k} x(E(W_i)).$$

Vsaj ena od neenačb $x(E(W_i)) \leq |W_i| - 1$, $i = 1, \ldots, k$ ima torej vrednost odstopanja manjšo ali enako vrednosti odstopanja neenačbe (4.8). Po drugi strani nekatere od teh neenakosti morda sploh niso kršene. Izmed vseh k neenačb izberemo tiste, ki imajo vrednost odstopanja manjšo od neke vnaprej določene vrednosti (glej Sliko 4.4c).

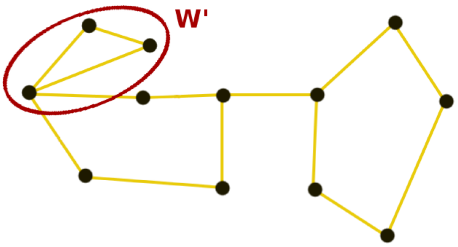
Algoritem Concorde je torej implementacija metode razveji in odreži, katere podlaga je algoritem razveji in omeji. Metoda razveji in odreži je iterativni postopek, kjer v vsaki iteraciji poiščemo rešitev linearnega programa na iskalnem območju, podanim z linearnimi neenakostmi. Za najdeno rešitev poiščemo morebitne kršitve neenakosti iz vnaprej znane družine neenakosti $\overline{\mathcal{L}}$. Z metodo presečnih ravnin odrežemo del iskalnega območja, ki ustreza najdenim kršitvam. Če kršitev ne najdemo, vzamemo primerno spremenljivko in razvejimo problem na dva podproblema. V enem ima spremenljivka vrednost 0, v drugem pa 1. Podproblema nato ločujemo ločeno. Iskanje v določeni veji prekinemo, če je vrednost rešitve večja od vnaprej poračunane zgornje meje obhoda trgovskega potnika. Hitrost delovanja algoritma je zato močno vezana na kvaliteto zgornje meje. Končni rezultat algoritma Concorde je obhod trgovskega potnika in hkrati tudi zagotovilo, da krajši obhod ne obstaja.



(a) Množica W, ki krši neenačbo podobhodov.



(b) Množico W nadomestimo z $W' = V - W$.



(c) Množico W nadomestimo z blokom W', ki ima dovolj majhno stopnjo odstopanja.

Slika 4.4: Primer redukcije neenačbe podobhodov.

Poglavje 5

Pridobitev podatkov

Tako Concorde kot LKH sprejmeta podatke o problemu v formatu TSPLIB [22]. TSPLIB je knjižnica TSP in sorodnih problemov iz različnih virov in različnih tipov. Vsaka datoteka sestoji iz dveh delov. Specifikacije vsebujejo informacije o formatu datoteke in njeni vsebini. Podatki pa vsebujejo eksplicitne informacije (npr. koordinate točk ali razdalje med točkami) v obliki, ki jo določimo v specifikaciji.

Podatki, ki smo jih želeli pridobiti, so koordinate 6007 slovenskih naselij in razdalje med njimi. Razdaljo med naselji lahko določimo na več različnih načinov. Odločili smo se za geografsko razdaljo, ki izračuna najkrajšo razdaljo med dvema točkama po zemljini površini. Lahko bi se odločili poiskati cestne razdalje med naselji v obliki časa ali dolžine potovanja, vendar tovrstnih razdalj ni enostavno izračunati. Podatke bi lahko pridobili iz spletnih virov (kot npr. Google Maps^{TM}), a je vseh povezav več kot 18 milijonov. Prav tako so razdalje tega tipa usmerjene, saj ni nujno da iz naselja A v naselje B lahko pridemo po isti poti. To pomeni, da bi reševali asimetričen problem trgovskega potnika, katerega reševanje bi vzelo dlje časa. TSPLIB format ima že vgrajeno opcijo, ki iz koordinat točk sama izračuna geografsko razdaljo. Format TSPLIB sprejme koordinate v obliki $DD.MM$, kjer so DD geografske stopinje in MM geografske minute. Geografske koordinate naselij Trata in Suha v občini Škofja Loka, navajamo samo en primer, se razlikujejo

za manj kot eno geografsko minuto tako v dolžini kot v širini. Natančnost geografskih koordinat formata TSPLIB je torej premalo natančna, zato smo se odločili, da razdalje poračunamo vnaprej. Izbrali smo razdaljo po velikem krogu (ortodroma). Razdalja po velikem krogu je najkrajša pot med dvema točkama po površini krogle. Izračunamo jo s formulo Heversine [23], ki je numerično boljša izbira za računanje majhnih razdalj. Naj bo r radij krogle, $\Delta\phi$ razlika v geografski širini in $\Delta\lambda$ razlika v geografski dolžini med točkama. Potem je razdalja po velikem krogu enaka

$$\Delta\sigma = 2r \arcsin \sqrt{\sin^2 \frac{\Delta\phi}{2} + \cos\phi_1 \cos\phi_2 \sin^2 \frac{\Delta\lambda}{2}}.$$

Ker Concorde lahko računa le s celoštevilskimi razdaljami smo razdalje shranili v metrih.

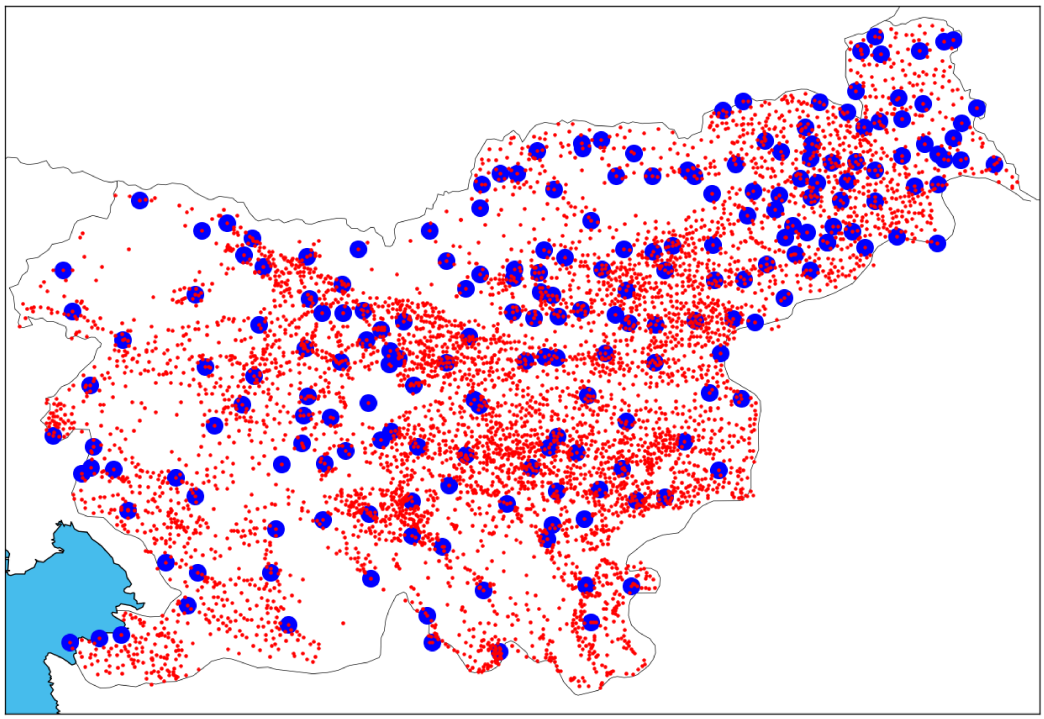
Imena naselij, občin in koordinate naselij smo pridobili na spletni strani Wikipedia [24]. Za vsako črko slovenske abecede imamo na Wikipediji stran z naselji, ki se začnejo na to črko. Vsako naselje ima podatke o imenu naselja, občini, poštni številki in pošti. Za naše potrebe smo shranili le ime naselja in občino, kateri pripada. Večina naselij ima povezavo na stran na Wikipediji, kjer se nahaja več informacij o tem naselju. Iz teh strani smo pridobili informaciji o geografski širini in dolžini, če sta bili podani. Za naselja Besnica, Marendol, Plešivec, Presika, Preska, Medvode, Stara Fužina, Dren, Drenje, Dragovica in Snežnik podatkov o lokaciji nismo našli na Wikipediji, zato smo informacije za ta naselja pridobili ročno s pomočjo aplikacije Google Maps. Izkaže se, da čeprav obstajajo posebne strani na Wikipediji, ki vsebujejo več informacij o posameznih občinah, te ne vsebujejo podatkov o geografski legi. Za geografsko lego občine smo zato vzeli kar geografsko lego enega od naselij, ki ji pripada. Če v občini obstaja istoimensko naselje, smo vzeli njegovo geografsko lego (npr. naselje Selnica ob Dravi v istoimenski občini). Če istoimensko naselje ne obstaja, obstaja pa naselje, katerega ime je vsebovano v imenu občine, smo vzeli njegovo geografsko lego (npr. naselje Polhov Gradec v občini Dobrova - Polhov Gradec). Drugače pa smo vzeli geografsko lego prvega naselja, ki se pojavi v seznamu naselij občine

(npr. naselja Babna Polica v občini Loška Dolina). Položaje naselij in občin na zemljevidu Slovenije vidimo na Sliki 5.1. Za nekatera naselja izgleda, da ležijo izven slovenske meje. Razlog je slaba natančnost meje, še posebej s Hrvaško.

Nadalje smo izračunali razdalje po velikem krogu med naselji in jih shranili v format TSPLIB (glej spodnji primer). Izbrali smo eksplicitni zapis uteži na povezavah (EDGE_WEIGHT_TYPE: EXPLICIT) s formatom zgornje trikotne matrike brez diagonale (EDGE_WEIGHT_FORMAT : UPPER_ROW). Na enak način smo shranili podatke o razdaljah med občinami.

NAME : slo
TYPE : TSP
DIMENSION : n
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT : UPPER_ROW
EDGE_WEIGHT_SECTION
$c_{1,2}$, $c_{1,3}$, . . . , $c_{1,n}$
$c_{2,3}$, $c_{2,4}$, . . . , $c_{2,n}$
. . .
$c_{n-1,n}$
EOF

Za pridobitev podatkov smo napisali tri programe. Prvi pridobi informacije o naseljih, občinah in geografskih koordinatah naselij, drugi poišče ustrezne geografske koordinate občin, tretji pa iz geografskih koordinat izračuna razdalje med točkami in podatke zapiše v datoteko formata TSPLIB. Vsi programi so spisani v programskem jeziku Node.js. Node.js je odprtokodno, večplatformno JavaScript okolje, ki je primarno namenjeno izvajanju JavaScript kode zunaj brskalnika.



Slika 5.1: Naselja (manjše rdeče pike) in občine (večje modre pike) na zemljevidu Slovenije.

Poglavje 6

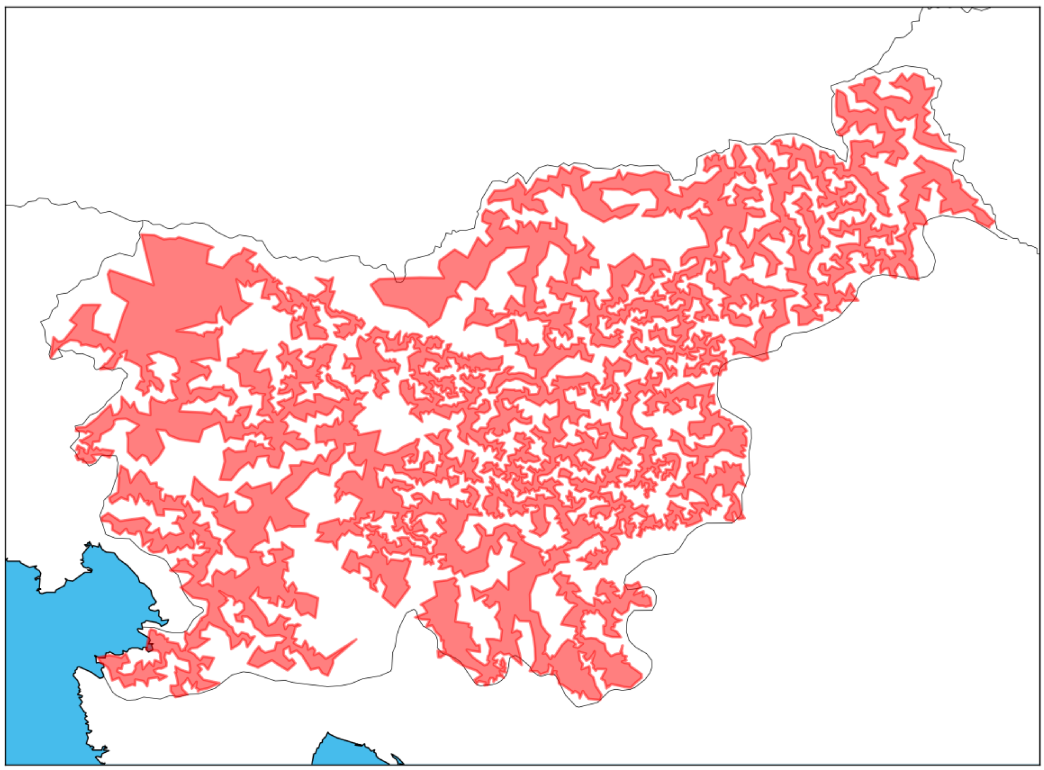
Analiza rezultatov

Za iskanje obhoda po naseljih smo uporabili programa LKH in Concorde. Programa smo zaganjali na računalniku z operacijskim sistemom ubuntu 16.04 LTS (64-bit), 16GB spomina, procesorjem Intel® Core™ i7-3770K CPU @ 3.50GHz × 8 in grafično kartico Intel® Ivybridge Desktop. Najprej smo s programom LKH poiskali zgornjo mejo obhoda. Program je našel zgornjo mejo dolžine 7733.125km v dobri uri in pol. Dobljeno zgornjo mejo smo nato posredovali programu Concorde, s katerim smo poskušali potrditi, da je dobljena zgornja meja tudi optimalna vrednost obhoda. Optimalno vrednost nam je uspelo potrditi v približno sedemdesetih dneh. Našli smo obhod po 6007 naseljih, ki ga na zemljevidu Slovenije lahko vidimo na Sliki 6.1. Vidimo, da obhod na določenih mestih prečka državno mejo ali pa gorsko verigo, kar je posledica izbire geografske razdalje, ki ne upošteva naravnih preprek in državnih mej. Obhod je prikazan kot sklenjena krivulja brez presečišč s pobarvano notranjostjo. Za metrične TSP obhode na točkah v ravnini velja, da ne vsebujejo samopresečišč. Slovenija zavzema dovolj majhno površino, da enako pričakujemo za TSP obhod po naseljih Slovenije.

Če iz obhoda odstranimo naselje, bi v splošnem morali zopet izračunati optimalni obhod, saj ni rečeno, da dobimo optimalni obhod, če povežemo preostali prosti vozlišči (glej Sliko 6.2). Za naselje Snežnik[1] (glej Sliko 6.3)

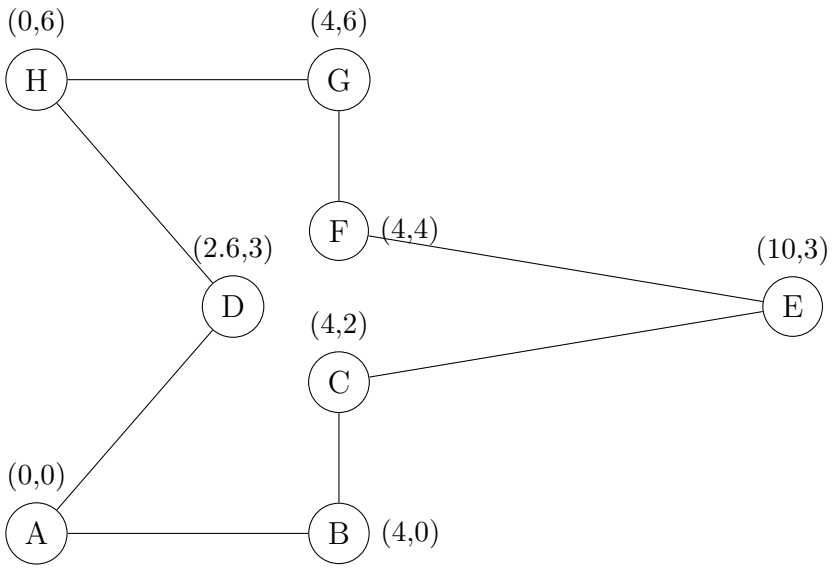
[1]Snežnik je tudi naselje h kateremu spadajo Gomance, Mašun in Sviščaki [25].

pa se zdi, da ponovno računanje ne bi bilo potrebno. Če odstranimo naselje Snežnik in povežemo naselji Kuteževo in Podgraje, se zdi, da smo pridobili optimalen obhod. Razlog je v ločenosti naselja Snežnik od vseh ostalih naselij. Zato se zdi, da njegova odstranitev ne bi imela vpliva na optimalen obhod. Da bi potrdili to trditev, bi seveda morali optimalen obhod izračunati od začetka na novih podatkih (brez naselja Snežnik).

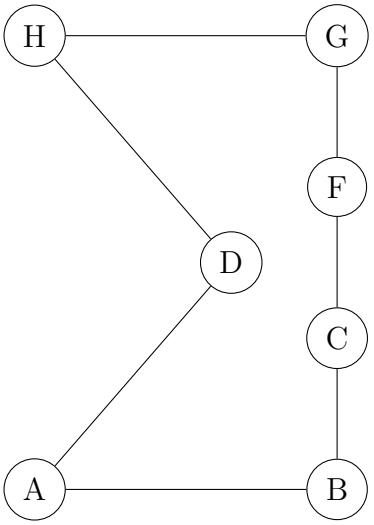


Slika 6.1: Optimalni obhod po naseljih Slovenije.

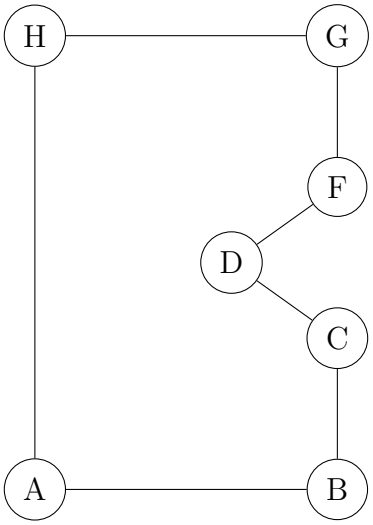
Dodatno smo izračunali še optimalni obhod po 210 občinah Slovenije. Kot pri naseljih smo uporabili program LKH za približek zgornje meje in program Concorde, da smo pokazali, da je dobljeni približek tudi optimalen obhod. Pričakovano sta oba programa porabila opazno manj časa za iskanje mej obhoda trgovskega potnika za občine kot za naselja (glej Tabelo 6.1). Zanimalo nas je, ali je optimalni obhod po naseljih zožan na obhod po občinah (Slika 6.4b) podoben optimalnemu obhodu po občinah (Slika 6.4a). Iz Slike



(a) Optimalni obhod za točke A, B, C, D, E, F, G in H.

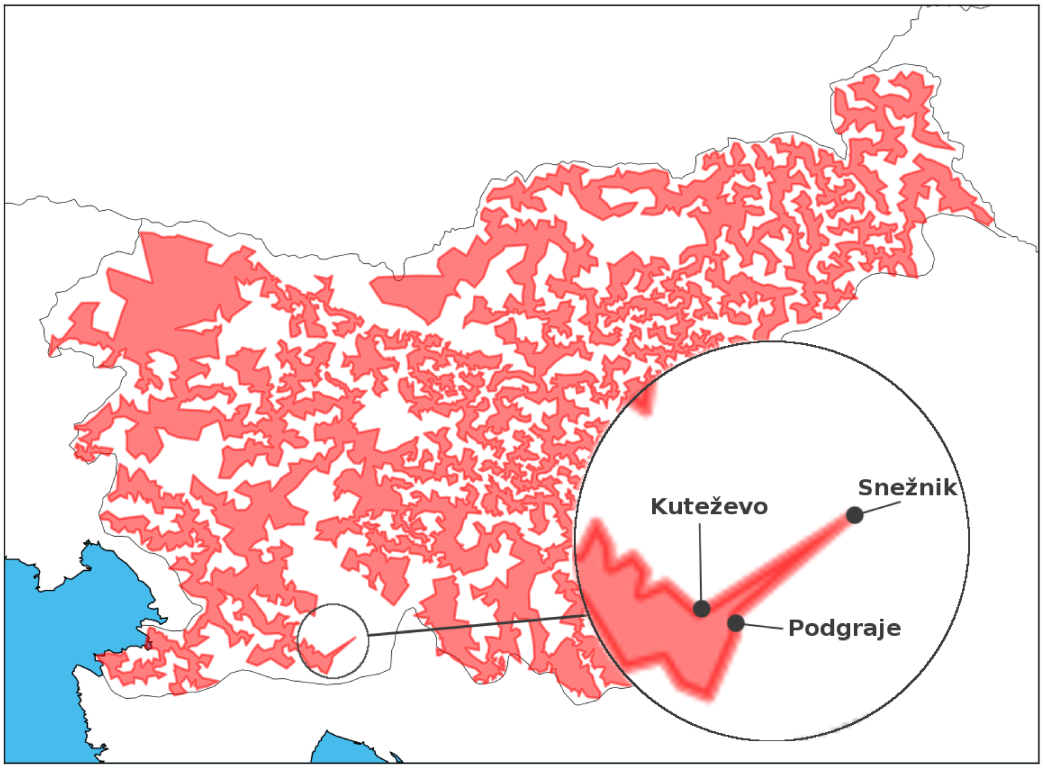


(b) Obhod brez točke E, dolžine 21,94.



(c) Optimalni obhod brez točke E, dolžine 21,44.

Slika 6.2: Odstranitev ene točke lahko spremeni optimalni obhod.

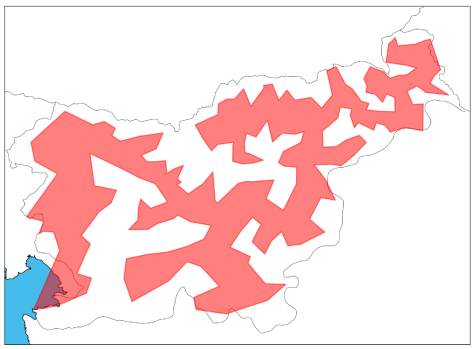


Slika 6.3: Posebnost v obhodu po naseljih Slovenije.

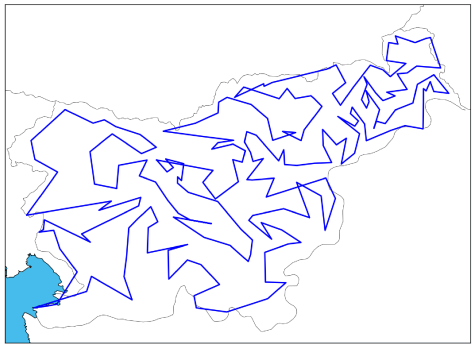
6.4b vidimo, da ima zožani obhod samopresečišča in zato ni optimalen. Če obhoda prekrijemo (Slika 6.4c), vidimo tudi, da zožani obhod močno odstopa od optimalnega obhoda po občinah skoraj povsod. Izkaže se, da je zožani obhod od optimalnega daljši kar za faktor 1.32.

	naselja	**občine**	**zožitev**
LKH	5981,34s (1.7h)	1,58s	-
Concorde	6081504,43s (70.4 dni)	0,22s	-
optimalni obhod(km)	7733,125	1752,764	2312,094

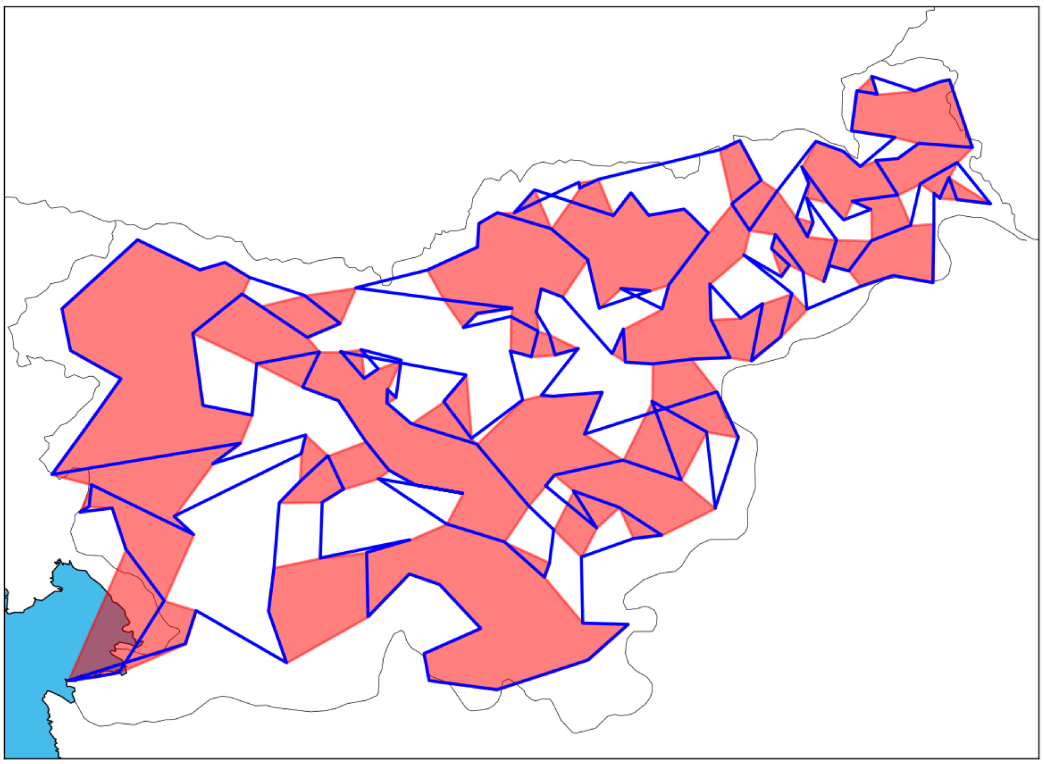
Tabela 6.1: Čas izvajanja in dožine obhodov.



(a) Optimalni obhod po občinah Slovenije.



(b) Zožitev optimalnega obhoda po naseljih Slovenije na obhod po občinah Slovenije.



(c) Primerjava optimalnega obhoda po občinah in zožanega obhoda po naseljih.

Slika 6.4: Optimalni obhod po občinah in zožani obhod po naseljih.

Poglavje 7

Zaključek

Problem trgovskega potnika je v teoriji težak problem. V magistrskem delu smo raziskali metodologijo za izračun obhoda trgovskega potnika po 6007 slovenskih naselijh. Najprej smo smo s programom LKH pridobili zgornjo mejo obhoda trgovskega potnika. Nato smo s programom Concorde izboljševali spodnjo mejo, dokler ni dosegla zgornje. Na začetku magistrske naloge smo si zadali cilj, najti takšno zgornjo in spodnjo mejo, da bo zgornja meja od spodnje slabša za faktor največ 1,0005. Meji nam je uspelo izenačiti, zato smo svoje cilje več kot izpolnili. Izenačenje mej je zagotovilo, da je obhod trgovskega potnika enak dobljeni zgornji meji dolžine 7733,125km.

Presenetilo nas je, da nam v celotnem postopku ni bilo treba spremeniti privzetih nastavitev programa Concorde. Zato pa smo več časa porabili, da smo ugotovili, kako program Concorde sploh deluje (npr. kako ponovno zagnati program, da nadaljuje od tam, kjer smo ga nazadnje ustavili), saj program nima uporabne dokumentacije ali primerov uporabe.

Dobljeni obhod smo vizualizirali na karti Slovenije ter omenili nekaj zanimivosti v obhodu. Obhod trgovskega potnika po naseljih Slovenije smo primerjali tudi z obhodom trgovskega potnika po občinah Slovenije. Izkazalo se je, da je optimalni obhod po občinah slaba aproksimacija za optimalni obhod po naseljih.

Problem trgovskega potnika na 100 točkah se je zdel pred 50 leti težak

problem v praksi. Metode in pristopi, razviti v zadnjih desetletjih, so naredile problem trgovskega potnika za enega najbolj obvladljivih NP-težkih problemov. V delu smo pokazali, da je dandanes mogoče velike praktične probleme trgovskega potnika reševati z domačim računalnikom — pa čeprav v mesecu ali dveh.

Literatura

[1] National TSP collection, [29. oktober 2017].
URL http://www.math.uwaterloo.ca/tsp/world/summary.html

[2] Romanian TSP instance and swiss TSP instance, [29. oktober 2017].
URL http://cadredidactice.ub.ro/ceraselacrisan/cercetare/

[3] C. H. Papadimitriou, Computational complexity, in: Encyclopedia of Computer Science, John Wiley and Sons Ltd., Chichester, UK, 2003, pp. 260–265.
URL http://dl.acm.org/citation.cfm?id=1074100.1074233

[4] M. Held, R. M. Karp, A dynamic programming approach to sequencing problems, Journal of the Society for Industrial and Applied Mathematics 10 (1) (1962) 196–210. arXiv:https://doi.org/10.1137/0110015, doi:10.1137/0110015.
URL https://doi.org/10.1137/0110015

[5] V. Chvátal, W. Cook, G. B. Dantzig, D. R. Fulkerson, S. M. Johnson, Solution of a Large-Scale Traveling-Salesman Problem, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 7–28. doi:10.1007/978-3-540-68279-0_1.
URL https://doi.org/10.1007/978-3-540-68279-0_1

[6] A. H. Land, A. G. Doig, An Automatic Method for Solving Discrete Programming Problems, Springer Berlin Heidelberg, Berlin, Heidelberg,

2010, pp. 105–132. doi:10.1007/978-3-540-68279-0_5.
URL https://doi.org/10.1007/978-3-540-68279-0_5

[7] D. K. Ly, K. Sugiyama, Z. Lin, M.-Y. Kan, Product review summarization based on facet identification and sentence clustering, CoRR abs/1110.1428.

[8] D. Applegate, R. Bixby, V. Chvátal, W. Cook, TSP Cuts Which Do Not Conform to the Template Paradigm, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 261–303. doi:10.1007/3-540-45586-8_7.
URL https://doi.org/10.1007/3-540-45586-8_7

[9] V. C. W. C. D. Applegate, R. Bixby, Concorde: a code for solving traveling salesman problem, [29. oktober 2017] (1999).
URL http://www.math.princeton.edu/tsp/concorde.html/

[10] M. Padberg, G. Rinaldi, A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, SIAM Rev. 33 (1) (1991) 60–100. doi:10.1137/1033004.
URL http://dx.doi.org/10.1137/1033004

[11] R. C. Prim, Shortest connection networks and some generalizations, The Bell System Technical Journal 36 (6) (1957) 1389–1401. doi:10.1002/j.1538-7305.1957.tb01515.x.

[12] G. A. Croes, A method for solving traveling-salesman problems, Operations Research 6 (6) (1958) 791–812. arXiv:https://doi.org/10.1287/opre.6.6.791, doi:10.1287/opre.6.6.791.
URL https://doi.org/10.1287/opre.6.6.791

[13] S. Lin, Computer solutions of the traveling salesman problem, The Bell System Technical Journal 44 (10) (1965) 2245–2269. doi:10.1002/j.1538-7305.1965.tb04146.x.

[14] S. Lin, B. W. Kernighan, An effective heuristic algorithm for the traveling-salesman problem, Operations Research 21 (2) (1973) 498–516.
URL http://www.jstor.org/stable/169020

[15] K. Helsgaun, An effective implementation of the lin–kernighan traveling salesman heuristic, European Journal of Operational Research 126 (1) (2000) 106 – 130. doi:https://doi.org/10.1016/S0377-2217(99)00284-2.
URL http://www.sciencedirect.com/science/article/pii/S0377221799002842

[16] F. Sullivan, J. Dongarra, Guest editors' introduction: The top 10 algorithms, Computing in Science Engineering 2 (2000) 22–23. doi:10.1109/MCISE.2000.814652.
URL doi.ieeecomputersociety.org/10.1109/MCISE.2000.814652

[17] H. Minkowski, Geometrie der Zahlen (Erste Licferung), Teubner, Leipzig, 1896.

[18] H. Weyl, Elementare theorie der konvexen polyeder., Commentarii mathematici Helvetici 7 (1934/35) 290–306.
URL http://eudml.org/doc/138641

[19] H. Crowder, M. W. Padberg, Solving large-scale symmetric travelling salesman problems to optimality, Manage. Sci. 26 (5) (1980) 495–509. doi:10.1287/mnsc.26.5.495.
URL https://doi.org/10.1287/mnsc.26.5.495

[20] M. W. Padberg, G. Rinaldi, Facet identification for the symmetric traveling salesman polytope, Math. Program. 47 (1990) 219–257.

[21] M. Grötschel, W. R. Pulleyblank, Clique tree inequalities and the symmetric travelling salesman problem, Math. Oper. Res. 11 (4) (1986)

537–569. doi:10.1287/moor.11.4.537.
URL http://dx.doi.org/10.1287/moor.11.4.537

[22] Ruprecht-Karls-Universität Heidelberg, Discrete and Combinatorial Optimization, TSPLIB, [29. oktober 2017].
URL https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

[23] Wikipedia contributors, Haversine formula — Wikipedia, the free encyclopedia, [10. februar 2018].
URL https://en.wikipedia.org/wiki/Haversine_formula

[24] Wikipedia contributors, Seznam naselij v sloveniji — Wikipedia, the free encyclopedia, [21. november 2017].
URL https://sl.wikipedia.org/wiki/Seznam_naselij_v_Sloveniji

[25] Wikipedia contributors, Naselje snežnik — Wikipedia, the free encyclopedia, [10.avgust 2018].
URL https://sl.wikipedia.org/wiki/Sne%C5%BEnik,_Ilirska_Bistrica