UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

DIPLOMSKO DELO

UPORABA METODE KITAJSKEGA POŠTARJA NA DELU LJUBLJANSKE CESTNE MREŽE

Ljubljana, januar 2005 TOMAŽ JUVANČIČ

IZJAVA

Študent **TOMAŽ JUVANČIČ** izjavljam, da sem avtor tega diplomskega dela, ki sem ga napisal pod mentorstvom **Prof. ddr. Ludvik Bogataja** in dovolim objavo diplomskega dela na fakultetnih spletnih straneh.

V Ljubljani, dne 14.1.2005 Podpis:_____

KAZALO

UVOD

Namen te diplomske naloge je sestaviti matematični model – graf ulic, cest in stavb v enoti Ljubljana-Center. Na podlagi tega modela bomo skušali z različnimi pristopi teorije grafov odgovoriti na naslednja vprašanja:

1. Kako izračunati najkrajšo pot med dvema točkama v modelu?
2. Kako določiti optimalno (najkrajšo) pot po kateri lahko prehodimo vse ulice v modelu s pomočjo metod trgovskega potnika?
3. Kako določiti optimalno (najkrajšo) pot po kateri lahko prehodimo vse ulice v modelu s pomočjo metod kitajskega poštarja?

Omenjeni modeli precej dobro predstavljajo realne situacije. V praksi lahko najdemo kar nekaj primerov, pri katerih si je možno pomagati z modeliranjem na podlagi sodobnih konceptov teorije grafov. Takšen primer je tudi raznašanje reklamnih letakov po gospodinjstvih, ki ga bomo obravnavali v tej diplomski nalogi. Poskušali bomo najti optimalno pot po dani mreži ulic ob upoštevanju nekaterih predpostavk. V diplomski nalogi si bomo torej predstavljali poštarja, katerega želja je obiskati dani nabor naslovnikov in pri tem porabiti čim manj časa, torej prehoditi čim krajšo pot.

Iz teorije grafov bomo uporabili predvsem koncepte problema najkrajše poti[1], problem kitajskega poštarja[2] in njegove različice, dotaknili pa se bomo tudi problema trgovskega potnika[3].

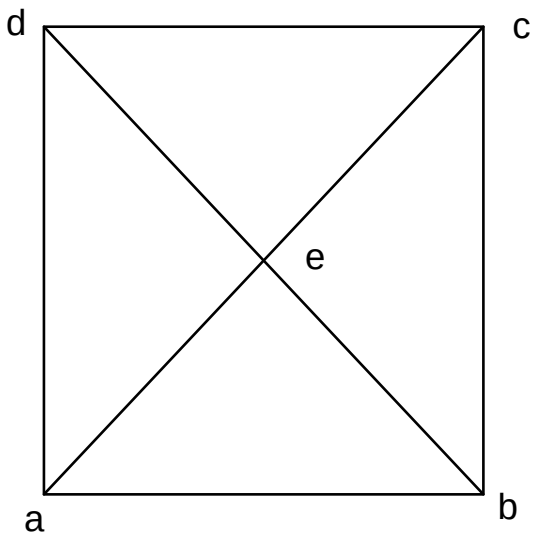
[1] Angl. Shortest Path Problem.
[2] Angl. Chinese Postman Problem.
[3] Angl. Traveling Salesperson Problem, kratko TSP.

Osnovni pojmi teorije grafov

1.1. Grafi in digrafi

Z grafom si v kontekstu te diplomske naloge predstavljamo množico točk v dvo- ali trodimenzionalnem prostoru in množico črt, ravnih ali ukrivljenih, vsaka črta pa povezuje dve točki ali pa eno točko samo s sabo.

Slika 1: Primer grafa



Vir: Lasten prikaz.

Grafi so zelo prilagodljivi modeli, primerni za analizo širokega spektra problemov, v katerih si določene entitete lahko predstavljamo kot točke, med katerimi obstajajo takšne ali drugačne povezave. Za potrebe modeliranja problemov iz realnega sveta je zato ključno, da podamo natančne definicije in terminologijo, ki jo podajamo po Hvalici (1994, str. 107) in Grossu (1999, str. 2).

Graf G=(V,E) je matematična struktura, sestavljena iz dveh množic V in E. Elementi množice V so točke[4], elementi množice E pa so **loki**[5]. Vsak lok ima pripadajoči točki (ali točko), ki sta **krajišči loka**[6].

[4] Tudi vozli ali vozlišča ali vrhovi, Angl. vertices, tudi nodes.
[5] Tudi veje ali povezave, Angl. edges.
[6] Angl. endpoints.

Usmerjen graf definiramo kot urejeno trojko **G = (V, E, φ).** Preslikava **φ** preslika vsak lok v urejen par točk:

$\varphi: l \rightarrow (v_1, v_2)$.

Pravimo, da lok **l** spaja točki $\mathbf{v_1}$ in $\mathbf{v_2}$, oziroma da sta ti točki njegovi **krajišči.** Točka $\mathbf{v_1}$ je začetna, točka $\mathbf{v_2}$ pa končna točka tega loka, točka $\mathbf{v_1}$ je **prednik** točki $\mathbf{v_2}$ oziroma točka $\mathbf{v_2}$ je naslednik točke $\mathbf{v_1}$.

Zanka je lok, ki povezuje točko v samo s sabo. Torej je $\varphi: l \rightarrow (v_1, v_1)$;

Drevo je graf, v katerem ima vsak vozel kvečjemu enega prednika. Neusmerjeno drevo je neusmerjen povezan graf, v katerem ni nobenega cikla. Podobno definiramo usmerjeno drevo kot usmerjen, vsaj šibko povezan graf, v katerem ni nobenega cikla (Saražin, 1984, str. 7).

Graf je **povezan,** če poljubnemu paru točk ustreza vsaj ena pot, ki ju povezuje (Hvalica, Rupnik, 1987, str. V/5).

V naši nalogi bomo v glavnem uporabljali naslednji pristop. V grafu bodo loki predstavljali ulice (oziroma natančneje dele ulic, saj bo ena ulica lahko sestavljena iz več zaporednih lokov), točke pa bodo predstavljale začetke in konce delov ulic, torej tudi vsa križišča. V nalogi se bomo omejili zgolj na neusmerjene loke. Sicer bi lahko upoštevali tudi enosmernost določenih ulic, te bi torej morali predstaviti z usmerjenimi loki. Vendar pa dva razloga govorita proti temu pristopu. Prvič se s tem kompleksnost modela zelo poveča, drugič pa enosmernost pogosto ni smiselna za naše potrebe, saj se v glavnem nanaša na motorna vozila. Ker naš model ni namenjen le-tem, bomo raje uporabljali samo neusmerjene loke. Naš graf bo torej v celoti neusmerjen.

Za potrebe našega modela moramo dodati še nekaj definicij (Gross, 1999, str. 6):

Sosednji točki sta točki, ki sta povezani z lokom.

Sosednja loka sta loka, ki imata skupno točko.

Pot po grafu je zaporedje lokov tega grafa, če sta dva zaporedna loka vedno sosednja. Nekateri avtoji ločijo poti, v katerih se loki lahko ponavljajo[7] in poti, v katerih se loki ne smejo ponoviti[8] (Locke, 2004).

Stopnja točke[9] je število, ki označuje število lokov, katerim je dana točka skupna.

V naši nalogi bodo točke s stopnjo 1 predstavljale konec slepe ulice. Točke s stopnjo 2 bodo predstavljale stičišča ulic oziroma pogosteje stičišča delov ulic, kadar bo ena ulica sestavljena iz več lokov. Točke s stopnjo 3 bodo predstavljale tako imenovana T-križišča, točke s stopnjo 4 pa bodo predstavljale običajna križna križišča. Točke s stopnjo višjo od 4 so redke, saj predstavljajo križišča, kjer se stika več cest ali ulic.

Na tem mestu moramo osnovni definiciji grafa dodati še eno množico, tako da je

G = (V,E, φ , ρ), kjer je

ρ: E → M,

preslikava, ki vsakemu loku **l** priredi njegovo utež **ρ(l)** – element množice **M**, ki daje o loku **l** neko dodatno informacijo (Hvalica, 1994, str.115).

V tej nalogi bodo uteži izražene v metrih kot dolžina loka in bodo torej predstavljale dejanske razdalje med posameznima točkama, če bosta točki povezani z lokom. V nasprotnem primeru, če te povezave ne bo, pa bo cena predstavljala neko veliko pozitivno število. Grafično lahko uteži predstavimo v matriki cen, kot tabelo velikosti n x n, ali pa kot tabelo velikosti (n x n) x 2, pri čemer je n število točk v grafu. Na naslednjih tabelah 1 in 2 podajamo možen prikaz matrike cen za enostaven graf.

[7] Angl. walk.
[8] Angl. trail.
[9] Tudi valenca, Angl. degree, valence.

Tabela 1: Prikaz matrike cen v obliki n x n

	a	b	c	d
a	0	1000	2	3
b	1000	0	4	3
c	2	4	0	1000
d	3	3	1000	0

Vir: Lasten prikaz.

Tabela 2: Prikaz matrike cen v obliki (n x n) x 2

a,a	0
a,b	1000
a,c	2
a,d	3
b,a	1000
b,b	0
b,c	4
b,d	3
c,a	2
c,b	4
c,c	0
c,d	1000
d,a	3
d,b	3
d,c	1000
d,d	0

Vir: Lasten prikaz.

2. Nekateri uporabni algoritmi v teoriji grafov

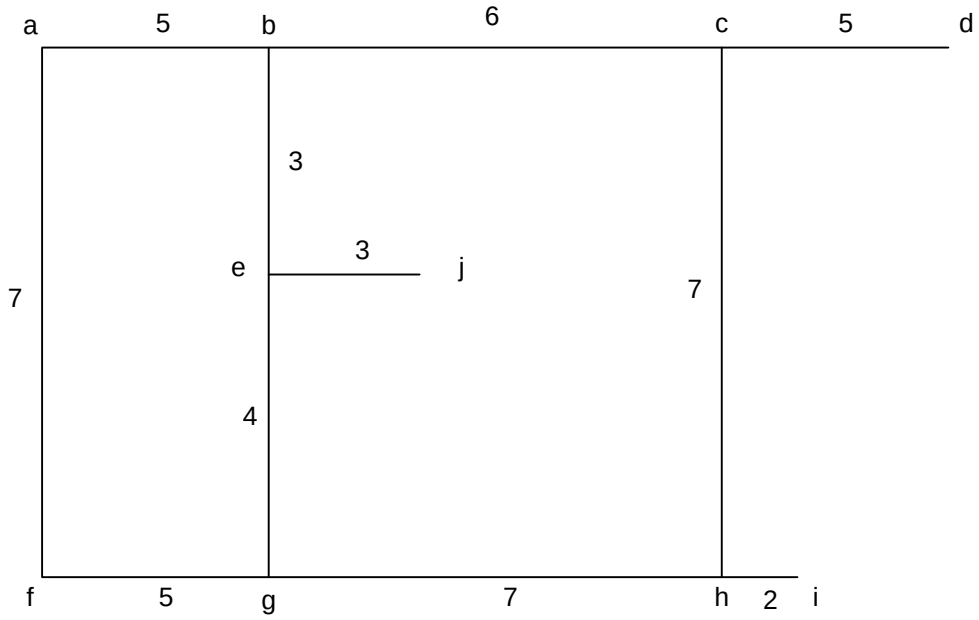
V diplomski nalogi se bomo osredotočili na iskanje optimalne poti po lokih in točkah danega grafa. Reševanje problema bomo razbili na manjše, toda zaključene celote, pri tem pa bomo potrebovali nekatere, v teoriji grafov dobro poznane algoritme. Nekaj teh algoritmov podajamo v tem poglavju.

2.1. Problem najkrajše poti

V tem delu bomo iskali najkrajšo možno pot med dvema točkama. Pri iskanju najkrajše poti med dvema točkama si lahko pomagamo z linearnim programiranjem, in sicer formuliramo problem odpošiljanja blaga[10]. V tem primeru skušamo minimizirati strošek pošiljanja ene enote z začetne točke v končno točko. Pri tem se vse ostale točke pojavljajo kot vmesne točke[11]. Strošek pošiljanja ene enote iz točke ***s*** v točko ***x*** je kar dolžina loka ***(s,x)***, če ta lok obstaja, oziroma neko veliko pozitivno število ***M***, če ta lok ne obstaja. Strošek pošiljanja ene enote iz neke točke samo vase je nič (Winston, 2004, str. 417).

Na spodnjem grafu poskusimo sedaj najti najkrajšo pot od točke ***a*** do točke ***j***.

Slika 2: Graf G, ki prikazuje enostavno mrežo ulic



Vir: Lasten prikaz.

[10] Angl. transshipment problem.
[11] Angl. transshipment points.

Najprej moramo podati matriko cen za loke grafa G:

Tabela 3: Matrika pripadajočih cen za dani graf G

	a	b	c	d	e	f	g	h	i	j
a	0	5	1000	1000	1000	7	1000	1000	1000	1000
b	1000	0	6	1000	3	1000	1000	1000	1000	1000
c	1000	1000	0	5	1000	1000	1000	7	1000	1000
d	1000	1000	1000	0	1000	1000	1000	1000	1000	1000
e	1000	1000	1000	1000	0	1000	4	1000	1000	3
f	1000	1000	1000	1000	1000	0	5	1000	1000	1000
g	1000	1000	1000	1000	1000	1000	0	7	1000	1000
h	1000	1000	1000	1000	1000	1000	1000	0	2	1000
i	1000	1000	1000	1000	1000	1000	1000	1000	0	1000
j	1000	1000	1000	1000	1000	1000	1000	1000	1000	0

Vir: Lasten prikaz.

Točki, ki predstavlja našo začetno točko dodelimo kapaciteto 1 in povpraševanje 0, točki, ki predstavlja našo končno točko pa dodelimo kapaciteto 0 in povpraševanje 1. Vsem ostalim točkam dodelimo vrednosti kapacitete in povpraševanja 1. To torej pomeni, da bomo pošiljali eno enoto iz začetne točke v končno, vse ostale točke pa bodo vmesne točke. Pri tem želimo minimizirati stroške pošiljanja:

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

Omenjeni problem lahko formuliramo v LINGU takole:

```
model:
! A 10 node Shortest Path Problem;

SETS:
   NODE1 / a,b,c,d,e,f,g,h,i,j/   : CAPACITY;
   NODE2   / a,b,c,d,e,f,g,h,i,j/ : DEMAND;
   ROUTES( NODE1, NODE2) : COST, VOLUME;
ENDSETS

! The objective;
 [OBJ] MIN = @SUM( ROUTES: COST * VOLUME);

! The demand constraints;
 @FOR( NODE1( J): [DEM]
  @SUM( NODE2( I): VOLUME( I, J)) >=
   DEMAND( J));

! The supply constraints;
```

```
 @FOR( NODE1( I): [SUP]
  @SUM( NODE2( J): VOLUME( I, J)) <=
   CAPACITY( I));

! Here are the parameters;
DATA:
   CAPACITY =   1, 1, 1, 1, 1, 1, 1, 1, 1, 0;
   DEMAND =     0, 1, 1, 1, 1, 1, 1, 1, 1, 1;
   COST =               0, 5, 1000, 1000, 1000, 7, 1000, 1000, 1000, 1000,
                                1000, 0, 6, 1000, 3, 1000, 1000, 1000, 1000, 1000,
                                1000, 1000, 0, 5, 1000, 1000, 1000, 7, 1000, 1000,
                                1000, 1000, 1000, 0, 1000, 1000, 1000, 1000, 1000, 1000,
                                1000, 1000, 1000, 1000, 0, 1000, 4, 1000, 1000, 3,
                                1000, 1000, 1000, 1000, 1000, 0, 5, 1000, 1000, 1000,
                                1000, 1000, 1000, 1000, 1000, 1000, 0, 7, 1000, 1000,
                                1000, 1000, 1000, 1000, 1000, 1000, 1000, 0, 2, 1000,
                                1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 0, 1000,
                                1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 0;

ENDDATA
end
```

Za dani problem dobimo rešitev *dist[j] = 11*, ta je podana v rezultatih LINGA pod *»objective value«.* To pomeni, da je razdalja med izbranima točkama ***a*** in ***j*** 11 enot. Če spreminjamo vrednosti v podatkih *CAPACITY in DEMAND* iz 1 v 0 in obratno, dobimo še rešitve za vse ostale kombinacije točk. Za podrobnejši pregled in obrazložitev rezultatov dobljenih z LINGOM glej prilogo 1.

2.2. Dijkstrov algoritem

Eden najbolj znanih algoritmov za iskanje najkrajše poti med dvema točkama je Dijkstrov algoritem. Ta je za naš problem še posebej uporaben, saj najde najkrajšo pot med dano točko in vsemi ostalimi točkami. Ta podatek bomo namreč potrebovali, ko bomo formulirali tako imenovan problem kitajskega poštarja. Algoritem nam prikaže drevo T, ki ima začetek v dani začetni točki ***s***, veje tega drevesa pa so najkrajše poti do vseh ostalih točk.

Dijkstrov algoritem je nastavljen takole (Gross, 1999, str. 147):

Vhod: obtežen povezan graf G, katerega teže lokov so nenegativne in točka s v G.
Izhod: Drevo T v G, zakoreninjeno v točki s, katerega vsaka pot od s do v je najkrajša pot od s do v v grafu G, in dolžine teh poti kot pripisane vrednosti nad v.

Nastavi začetek Dijkstra drevesa v točki s.
Določi množico mejnih lokov drevesa T za prazno množico.
dist[s] := 0
Dokler Dijkstra drevo T ne vsebuje vseg točk v G
Za vsak mejni lok e drevesa T
Naj bo x končna točka loka e, z že pripisano vrednostjo.

Naj bo y končna točka loka e, ki še nima pripisane vrednosti
Določi P[e] = dist[x] + w(e).
Naj bo e mejni lok drevesa T, ki ima najmanjšo vrednost P.
Naj bo x končna točka loka e, z že pripisano vrednostjo.
Naj bo y končna točka loka e, ki še nima pripisane vrednosti
Dodaj lok e in točko y drevesu T.
dist[y] := P(e)
Pripiši vrednost dist[y] nad točko y.
Povrni Dijkstra drevo in pripisane vrednosti točk.

Izdelajmo sedaj Dijsktra drevo za zgoraj obravnavani primer grafa. Želimo izračunati najkrajše poti od točke ***a*** do vseh ostalih točk.

Najprej nastavimo začetek Dijkstra drevesa v točko a. *Dist[a] = 0* , zato pripišemo točki a vrednost 0, takole: *a(0).* Točka a je zaenkrat edini element drevesa T. Mejna loka tega drevesa sta loka *ab* in *af.* Zato je:

P[ab] = 5 in P[af] = 7.

Ker je *af* mejni lok drevesa T z najnižjo vrednostjo P, dodamo ta lok k drevesu T. Zato je *dist[b] = 5,* točki b pa pripišemo vrednost 5 takole: *b(5).*

V naslednjem koraku zopet poiščemo mejne loke drevesa T. Sedaj so to loki *af, be in bc.* Za te loke zopet izračunamo vrednost P.

P[af] = 7, P[be] = 5 + 3 = 8, P[bc] = 5 + 6 = 11

Tokrat je lok z najnižjo P-vrednostjo lok af, zato ga dodamo k drevesu T, točki f pa priredimo vrednost 7. f(7).

Ta postopek ponavljamo toliko časa, da najdemo dist[y] za vse točke v grafu. V našem primeru tako dobimo naslednje vrednosti:

dist[a] = 0, dist[b] = 5, dist[f] = 7, dist[e] = 8, dist[c] = 11, dist[j] = 11,
dist[g] = 12, dist[d] = 16, dist[h] = 18, dist[i] = 20

Za točen potek in grafični prikaz vseh ostalih korakov glej prilogo 2.

2.3. Primov algoritem najmanjšega razvejanega drevesa

Primov algoritem je eden izmed bolj poznanih v teoriji grafov. Za potrebe te diplomske naloge je uporaben predvsem kot osnova za problem trgovskega potnika, katerega bomo predstavili v poglavju 2.6. Primov algoritem je po sestavi zelo podoben Dijkstrovemu, njegov končni rezultat pa je graf v obliki drevesa (torej brez krožne poti, oziroma zanke). Tako nastalo *Primovo drevo* vsebuje vse točke iz prvotnega grafa, hkrati pa minimizira vsoto cen vseh lokov drevesa. Glede na to, da je lahko število vseh možnih razvejanih dreves za graf z n točkami tudi do n^{n-2}, je nepraktično iskati vse možne kombinacije. Naslednji algoritem nam lahko precej olajša delo (Gross, 1999, str. 146).

Vhod: obtežen povezan graf G
Izhod: najmanjše razvejano drevo T.
Izberi neko začetno točko s.
Naj b s začetno Primovo drevo T.
Določi množico mejnih lokov drevesa T za prazno množico.
Dokler Primovo drevo T še ne vsebuje vseh točk v G
Določi množico mejnih lokov za T.
Naj bo e mejni lok T-ja za najmanjšo težo.
Naj bo v končna točka e-ja, ki še ni v drevesu T.
Dodaj lok e (in s tem točko v) v drevo T.
Zaključi Primovo drevo T.

Podoben algoritem navaja tudi Skiena (2004, str. 1). Ilustracija poteka algoritma Primovega drevesa je prikazana v prilogi 3.

S konceptom Primovega drevesa se bomo zopet srečali pri obravnavi problema trgovskega potnika.

2.4. Eulerjeve poti

Eulerjeva pot je dobila svoje ime po Švicarskemu matematiku 18. stoletja Leonardu Eulerju. Euler je z rešitvijo problema Koenigsbergovih mostov postavil temelje teorije grafov. Eulerjeva pot po definiciji pomeni naslednje (Gross, 1999, str. 156):

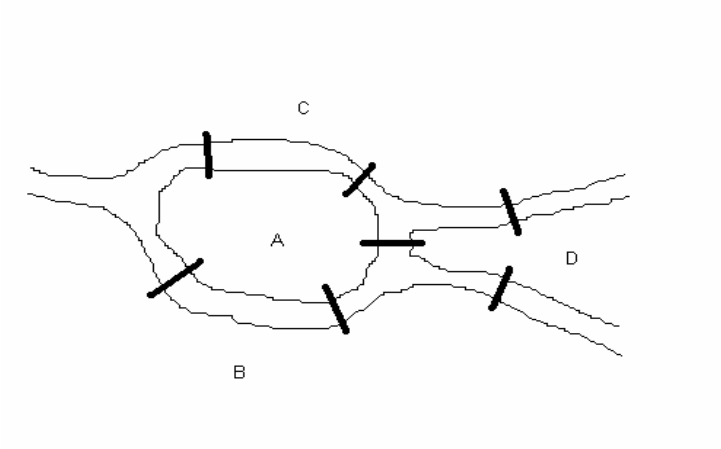
Eulerjeva pot v grafu je pot, ki vsebuje vsak lok tega grafa.

Eulerjeva krožna pot je zaprta Eulerjeva pot.

Eulerjev graf je graf, ki ima zaprto Eulerjevo pot.

Poglejmo si primer mesta Konigsberg, na osnovi katerega bomo predstavili uporabnost Eulerjevih poti. Mesto Konigsberg, ki je včasih spadalo pod Vzhodno Prusijo, leži na dveh bregovih reke Pregel in na dveh otočkih, ki jih ta reka obdaja. Štirje deli mesta so povezani s sedmimi mostovi, kot kaže naslednja slika:

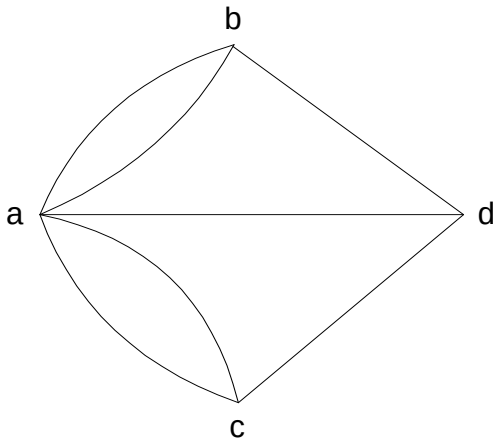
Slika 3: Mesto Koenigsberg - skica



Vir: Gross, 1999, str.156.

Eulerjev model problema je predstavljal graf s štirimi točkami, vsaka točka je predstavljala segment kopnega, in s sedmimi loki, ki ponazarjajo mostove.

Slika 4: Graf, ki ponazarja mostove Koenigsberga



Vir: Gross, 1999, str. 157.

Euler je pokazal, da ne obstaja pot, po kateri bi bilo možno prehoditi vsak most natanko enkrat. V sodobni terminologiji to pomeni, da graf nima Eulerjeve krožne poti.

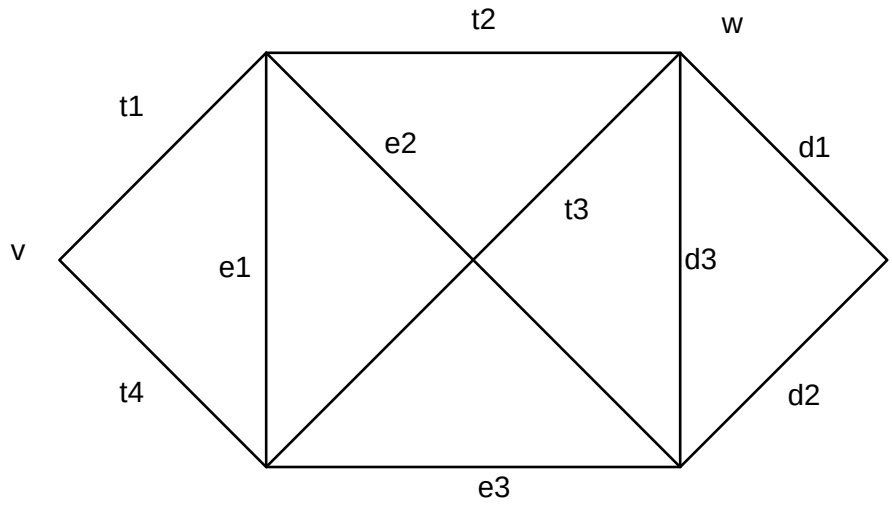
Euler je pokazal tudi, da graf, ki vsebuje točke z liho stopnjo (torej točke, ki so skupne lihemu številu lokov), ne more imeti Eulerjeve krožne poti. In obratno, vsak graf, ki ima vse točke s sodo stopnjo, mora imeti Eulerjevo krožno pot.

Algoritem, s katerim konstruiramo Eulerjevo krožno pot v grafu je naslednji (Gross, 1999, str. 205):

Začni v katerikoli točki v in konstruiraj krožno pot T v grafu G.
Dokler obstajajo loki grafa G, ki še niso v krožni poti T
izberi točko w v T, ki je začetna točka neuporabljenega loka (loka, ki ni v T).
Začenši v točki w, konstruiraj krožno pot D, sestavljeno iz neuporabljenih lokov.
Povečaj T z D (združi T in D, pot D je »obvoz«, ki se začne in konča v točki w).
Povrni T.

Izpeljimo zgornji algoritem na naslednjem grafu:

Slika 5: Primer enostavnega grafa



Vir: Gross, 1999, str. 206.

Začnimo v točki ***v***. Naša prva krožna pot ***T*** naj bo ***t1, t2, t3, t4***. Nato najdemo točko v ***T*** ki je začetna točka nekega neuporabljenega loka. Naj bo ta točka ***w***. Iz točke ***w*** začnemo krožno pot ***d1, d2, d3***, tako da pridemo spet v točko ***w***. Predstavljajmo si takole. Prvotno pot ***t1, t2, t3, t4*** smo povečali z obvozom, tako

da je nova pot: ***t1, t2, d1, d2, d3, t3, t4***. V naslednji iteraciji zopet najdemo točko, ki je začetna točka še neuporabljenega loku. Začenši v tej točki, konstruiramo krožno pot ***E(e1, e2, e3)***, ki jo kot obvoz vrinemo v pot ***T***.

Naj povemo še, da pri vseh modelih ni potrebno opraviti celotne krožne poti. Včasih je dovolj, da prehodimo vse loke grafa, brez da bi se vrnili v začetno pozicijo. Takšni poti rečemo **odprta Eulerjeva pot.** Pogoj za obstoj le-te je, da ima graf G natanko dve točki z liho stopnjo.

Koncept Eulerjevih poti bomo potrebovali v naslednjem poglavju, kjer bomo opredelili problem kitajskega poštarja.

2.5. Problem kitajskega poštarja

Kitajski matematik Meigu Guan se je ukvarjal s problemom, kako najti najkrajšo pot, po kateri bi prehodili vsak lok danega grafa vsaj enkrat. Guan si je predstavljal poštarja, ki želi prehoditi mrežo vseh ulic v kar najkrajšem času in se vrniti v začetno točko – poštno pisarno.

Če ima vsaka točka v grafu sodo stopnjo, potem bo vsaka eulerijska krožna pot hkrati tudi optimalna rešitev problema kitajskega poštarja, saj bo vsak lok prehojen natanko enkrat. Seveda pa v realnosti ni mogoče pričakovati takšnega grafa. Zato je potrebno nekatere loke podvojiti (stopnja njihovih končnih točk pa se bo s tem povečala). Optimalna rešitev problema je v tem, da podvojimo tiste loke, katerih skupna teža oziroma cena bo najmanjša, pri tem pa preslikamo prvotni graf ***G*** v ***G*,*** tako da bo G* Eulerjev graf.

Standardni algoritem s katerim najdemo optimalno poštarjevo pot je naslednji (Gross, 1999, str. 210):

Vhod: povezan obtežen graf G
Izhod: optimalna poštarjev pot W.
Najdi množico liho-stopenjskih točk v grafu G. Naj bo to nova množica S.
Za vsak par točk u in v iz S,
Najdi najkrajšo pot P v grafu G med točkama u in v
Naj bo d(uv) dolžina poti P.
Konstruiraj popolni graf K s točkami, ki so v S.
Za vsak lok e v K,
priredi loku e težo d(uv), u in v sta končni točki loka e.
Najdi popolno ujemanje M v grafu K, tako da bo skupna teža lokov v M najmanjša.
Za vsak e v popolno ujemanem grafu M
Naj bo P pripadajoča najkrajša pot v G med končnimi točkami loka e.

Za vsak lok f na poti P

Dodaj grafu G duplikat loka f, vključno z njegovo težo.

Naj bo G eulerjev graf, ki je nastal z dodajanjem reobov iz prejšnjega koraka grafu G. Konstruiraj Eulerjevo krožno pot W na grafu G*. W je optimalna poštarjeva pot prvotnega grafa G.*

To pomeni, da podvojimo loke po poti med dvema lihostopenjskima točkama ***u*** in ***v***. Stopnja teh točk tako postane soda (saj smo vsaki dodali en lok). Stopnja vseh vmesnih točk po poti med ***u*** in ***v*** pa se poveča za 2.

Izpeljimo zgornji algoritem na našem že obravnavanem primeru grafa iz slike 2.

Najprej najdemo podmnožico vseh lihostopenjskih točk v grafu ***G***. Ta podmnožica ***S*** vsebuje točke ***b, c, d, e, g, h, i, j***. Za dane točke izračunamo najkrajše poti za vsako kombinacijo dveh točk. Za to lahko uporabimo kar Dijkstrov algoritem. Tako dobimo matriko razdalj za vsako kombinacijo točk:

Tabela 4: Matrika razdalj za vsako kombinacijo točk v grafu G

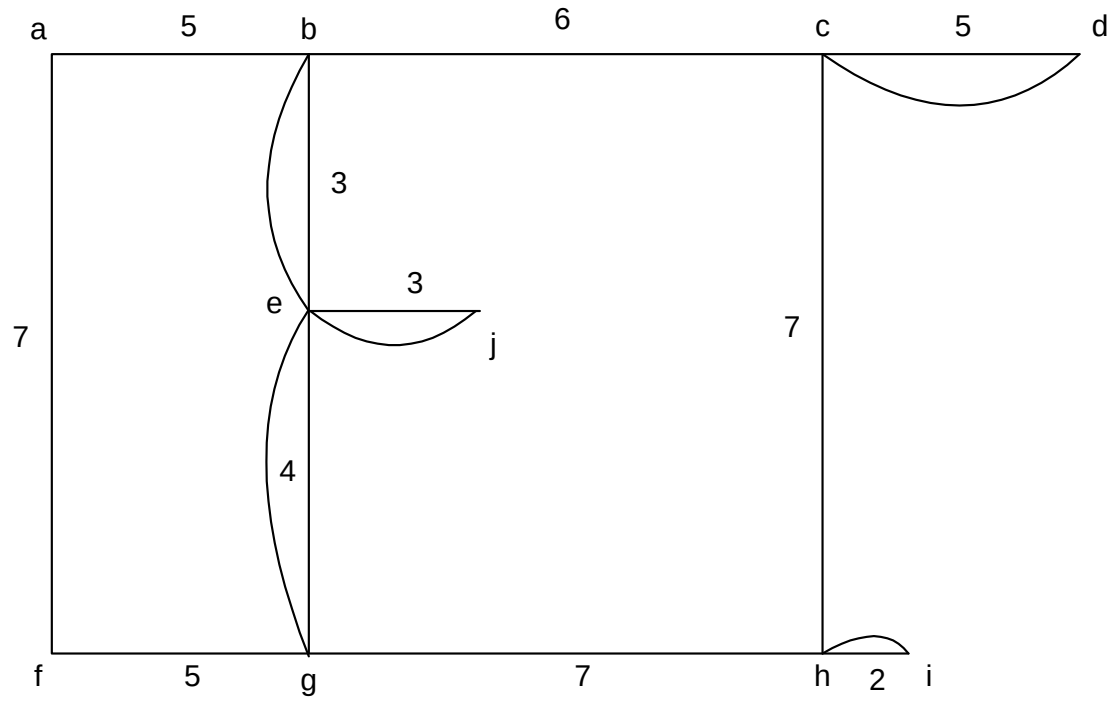
	b	c	d	e	g	h	i	j
b	0	6	11	3	8	13	15	6
c	6	0	5	9	14	7	9	12
d	11	5	0	14	19	12	14	17
e	3	9	14	0	5	12	14	3
g	8	14	19	5	0	7	9	8
h	13	7	12	12	7	0	2	15
i	15	9	14	14	9	2	0	17
j	6	12	17	3	8	15	17	0

Vir: Lasten prikaz.

Na podlagi teh točk narišemo popolni graf ***K*** z loki ***bc, bd, be,..., jg, jh, ji***. Teže lokov preberemo iz zgornje matrike cen. Sedaj moramo najti popolno ujemanje ***M*** v grafu ***K***, tako da bo teža lokov v ***M*** najmanjša. Zaenkrat naj povemo, da to pomeni, da moramo najti takšne loke, da bodo vsebovali vseh osem točk iz množice ***S***, vsota teže teh lokov pa bo najmanjša. Logično je, da bo v našem primeru osmih točk število lokov štiri. To so loki ***cd, hi, ej*** in ***bg***.

Za vsakega izmed teh lokov najdemo najkrajšo pot v originalnem grafu. Pri lokih ***cd, hi*** in ***ej*** je to kar sam lok, pri loku ***bg***, pa je najkrajša pot ***b-e-g***. Vse te loke sedaj dodamo originalnemu grafu ***G,*** tako da dobimo novi graf ***G*.*** Ta se torej od prvotnega razlikuje po podvojenih lokih ***be, ej, eg, cd*** in ***hi***.

Slika 6: Graf Eulerjev graf G* s podvojenimi loki



Vir: Lasten prikaz.

Na temu grafu sedaj konstruiramo Eulerjevo pot, na primer:

a-b-e-j-e-b-c-d-c-h-i-h-g-e-g-f-a

Celotna teža lokov te poti je 73. Glede na to, da je vsota dolžin lokov v grafu G 55, to pomeni, da smo 18 enot prehodili dvakrat.

Do rešitve problema lahko pridemo tudi enostavneje. Loke, ki se končajo v točki s stopnjo 1 lahko preprosto podvojimo, to pa zato, ker se moramo v vsakem primeru iz takšne točke vrniti po isti poti, po kateri smo prišli. Takšni loki namreč predstavljajo slepe ulice. S tem se znebimo precejšnjega števila lihostopenjskih križišč. Enostopenjske točke v naši množici ***S*** so točke ***d, i*** in ***j,*** pripadajoči loki pa ***cd, hi*** in ***ej***. Ko te loke podvojimo, se poveča tudi stopnja v točkah ***c, h*** in ***e*** in sicer na 4. Tako ostaneta zgolj dve lihostopenjski točki, in sicer točki ***b*** in ***g*** s stopnjo 3. Ko podvojimo loke najkrajše poti med tema točkama, dobimo Eulerjev graf ***G*.***

Problem kitajskega poštarja je tudi temelj za nekatere modernejše koncepte, ki iščejo optimalno pot po lokih grafa ob dodatnih predpostavkah. Nekaj teh konceptov navajamo v naslednjih vrsticah:

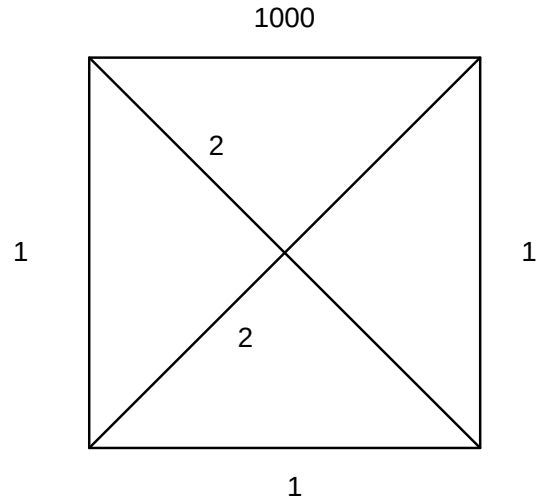
Usmejeni problem kitajskega poštarja[12] (Timbleby, 2003, str. 2) predstavlja situacijo, v kateri iščemo optimalno pot po vseh robovih grafa, v katerem je vsaj en rob usmerjen, se pravi, da dopušča prehod zgolj v eni smeri.

Nekateri avtorji še nadgradijo zgornji problem in iščejo najkrajšo pot, po kateri obiščemo dani nabor naročnikov, pri tem pa upoštevamo še število vozil (kurirjev), ki so na razpolago ter njihovo kapaciteto in povpraševanje posameznega naročnika (Lacomme, 2004). Takšen problem lahko z ustrezno transformacijo spremenimo v problem trgovskega potnika (Baldacci, Maniezzo, 2004, str.1).

2.6. Problem trgovskega potnika

Pri iskanju optimalne poti med vsemi točkami danega grafa lahko uporabimo še eno izmed bolj znanih metod, to je problem trgovskega potnika. Problem trgovskega potnika ponazarja situacijo, v kateri želi trgovski potnik obiskati vse točke v dani množici, pri tem pa minimizirati stroške tega potovanja. Tako so lahko točke v množici mesta ali stranke, stroški pa so dani v obliki kilometrov ali denarnih enot. Imamo torej graf G=(V,E), obiskati pa moramo vsako izmed točk v E. Graf si lahko predstavljamo kot popoln graf, pri čemer moramo prirediti velike vrednosti (velike glede na končno rešitev modela) za loke, ki dejansko ne obstajajo. Primer takšnega enostavnega grafa je prikazan na naslednji sliki.

Slika 7: Primer enostavnega popolnega grafa



Vir: Lasten prikaz.

[12] Angl. Directed Chinese Postman Problem.

Problem trgovskega potnika ima že dolgo zgodovino v svetu operacijskih raziskav. Najzgodnejša dela tako segajo v leto 1759, ko je Euler predstavil rešitev Problema skakačeve poti. Drugi zgodnji raziskovalci tega problema so bili A. T. Vandermonde, T. P. Kirkman in W. R. Hamilton. Prvi večji preboj v iskanju dokazljivo optimalne poti se je zgodil leta 1954, ko so G. B. Dantzig, D. R. Fulkerson in S. M. Johnson rešili problem trgovskega potnika na množici 49 mest v Združenih Državah Amerike. Naslednji dramatični uspehi so sledili leta 1980, ko sta Crowder in Padberg našla dokazljivo optimalno pot za 318 mest. Za poskus predstavitve naj povemo, da je število možnih kombinacij za ta problem 10^{655}. Če bi računalnik računal s hitrostjo 1.000.000.000 poti na sekundo, bi potreboval 10^{639} let (Gross, 1999, str. 228).

To dokazuje, da je problem trgovskega potnika izredno kompleksen, zato moramo najti algoritme, ki dovolj hitro najdejo približno optimalno pot, z natančnostjo, ki je zadovoljiva z vidika danega problema. Kasneje bomo predstavili model stavb in ulic v segmentu Ljubljana-Center, ki bo vseboval nekaj tisoč točk. Seveda je za tako obsežen problem nesmiselno iskati povsem optimalno rešitev, zadovoljili se bomo že s približkom na nekaj kilometrov natančno. V nadaljevanju bomo prikazali dva enostavna algoritma za reševanje problema trgovskega potnika.

2.6.1. Algoritem: Najbližji sosed

To je najenostavnejši TSP algoritem, ki ima v ozadju zelo kratkovidno filozofijo: v katerikoli točki si, izberi najkrajšo pot do nekam drugam. Zapišemo ga lahko takole (Gross, 1999, str. 229):

Vhod: popolni graf G s pripadajočimi težami lokov
Izhod: Zaporedje označenih točk, ki tvorijo Hamiltonov cikel.

```
Začni v katerikoli točki v.
Določi l(v)  =  0
Določi i = 1
Dokler obstajajo neobiskane točke
        Izberi najcenejšo pot, ki povezuje v z neoznačeno točko, npr. w
        Določi l(w) = i
        v := w
```

Algoritem najbližji sosed je tipični »pohlepni«[13] algoritem. Glavni prednosti sta njegova hitrost in enostavna implementacija. Včasih se algoritem obnese

[13] Angl. greedy algorithm.

presenetljivo dobro. V splošnem pa lahko prinese slabe rezultate, še posebej v grafih, kjer obstajajo velike razlike med težami posameznih lokov.

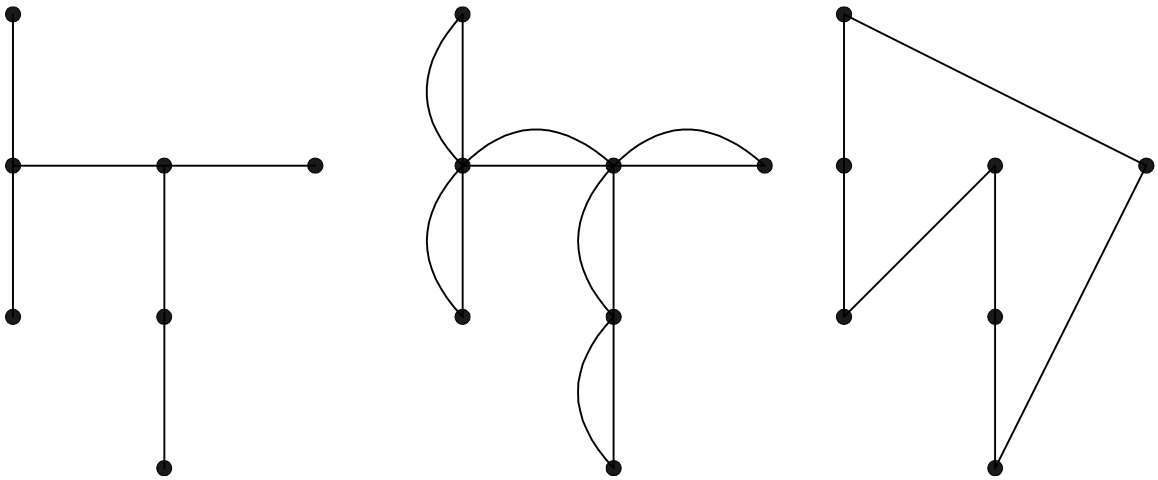
2.6.2. Algoritem: Podvojeno drevo

Ta algoritem ima še vedno nizko stopnjo kompleksnosti in prinese precej dobre rezultate za grafe z določenimi omejitvami (Gross, 1999, str. 230).

Vhod: obtežen popoln graf G.
Izhod: zaporedje točk in lokov, ki tvorijo Hamiltonov cikel.

> *Najdi najmanjše razvejano drevo T* v G.*
> *Izdelaj Eulerjev graf H tako, da podvojiš vsak lok v T*.*
> *Izdelaj eulerjevo pot W v grafu H.*
> *Konstruiraj pot v grafu G kot sledi:*
>
> > *Sledi zaporedju lokov in točk v W, dokler na prideš do točke, kjer je naslednji lok v zaporedju povezan z že obiskano točko. Na tej točki preskoči na naslednjo neobiskano točko po bližnjici, ki ni del W. Nadaljuj po poti W in uporabljaj bližnjice kjer je potrebno in mogoče, dokler ne obiščeš vseh točk.*

Slika 8: Ilustracija algoritma Podvojeno drevo podana na enostavnem grafu



Vir: Gross, 1999, str. 231.

Problem trgovskega potnika in problem minimalnega razvejanega drevesa je rešljiv tudi v LINGU, vendar je proces zelo počasen pri večjemu številu točk in lokov.

Nekateri avtorji na osnovi problema trgovskega potnika razvijejo bolj zapletene koncepte z dodatnimi omejitvami. Takšen je tudi **problem razporejanja vozil**[14], ki predstavlja situacijo, v kateri mora fiksno število vozil z znano kapaciteto zadostiti danemu povpraševanju strank, pri tem pa minimizirati transportne stroške (Ralphs, 2004).

3. Uporaba pristopov teorije grafov na delu ljubljanske cestne mreže

Naj ponovimo, da je cilj našega modela najti dolžino najkrajše poti med vsemi danimi stavbami in pa zaporedje, po katerem je treba dane stavbe obiskati. Seveda je natančnost rezultatov obratnosorazmerna s stroški analize, model, ki bi vključeval vse predpostavke realnega sveta, pa krepko presega zahtevnost te diplomske naloge. V nadaljevanju podajamo opis modela, uporabljenega v tej diplomski nalogi.

3.1. Osnovni opis modela ljubljanske cestne mreže

Za potrebe te naloge smo pod drobnogled vzeli podatke za mesto Ljubljana, enota Center. Podatke smo dobili od Geodetske uprave Republike Slovenije in zajemajo ulice in hišne številke na območju Ljubljana Center. Število hišnih številk je 2433, število ulic pa 218, razdeljene so v 3242 segmentov. Naš model, ki bo zajemal vse hišne številke in ulice je zaradi velikosti neprimeren (prevelik) za stoodstotne rešitve, ki jih dobimo z metodami linearnega programiranja.

Za ulice in hišne številke so podatki podani v tabelarični obliki in tudi kot vektor. Pri tabelarični obliki nas za hišne številke predvsem zanimajo naslednji atributi:

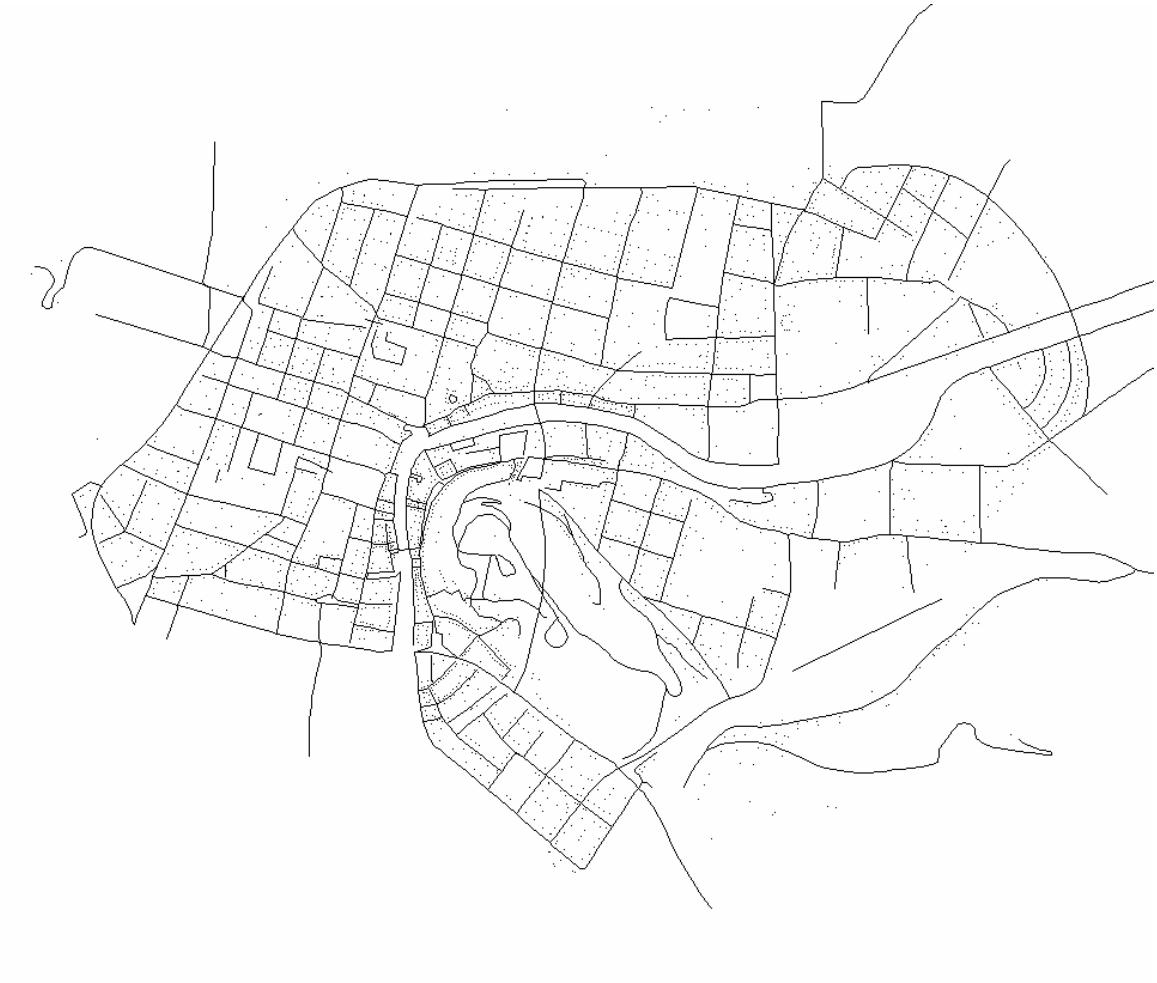
- hišna številka
- dodatek hišne številke (npr. A, B, C...)
- identifikator ulice
- koordinata Y
- koordinata X

[14] Angl. Vehicle Routing Problem.

Vektorski zapis za posamezno hišno številko je točka v danem dvodimenzionalnem prostoru.

Pri ulicah pa je pomemben predvsem vektorski podatek. Vsaka ulica je tako sestavljena iz enega ali več vektorjev, ki predstavljajo ravne črte v dvodimenzionalnem prostoru. Za vsakega od teh vektorjev imamo podano začetno in končno točko v koordinatah X in Y. Žal pa dani zapis ne podaja podatka, kateri vektorji tvorijo katero ulico, pač pa imamo zgolj podan zapis vektorjev na eni strani in tabelarični prikaz ulic na drugi. Na sliki 9 si lahko ogledamo grafični prikaz ulic in stavb zajetih v našem modelu.

Slika 9: Grafični prikaz ulic in stavb na območju Ljubljana-Center



Vir: Geodetska Uprava Republike Slovenije, 2004.

3.2. Transformacija vektorskih podatkov v modelu cestne mreže

Preden lahko ustvarimo pregleden model, primeren za nadaljnjo analizo, moramo predstaviti še nekaj osnovnih transformacij vektorskih podatkov. Tako bomo večkrat uporabili formulo za izračun zračne razdalje med dvema točkama, najsi bosta ti točki predstavljali stavbe (hišne številke), ali pa začetek oziroma konec vektorja ceste, ali pa kombinacijo obojega. Ta formula je:

$$d = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$$

Prvič bomo to uporabili, ko bomo odpravljali manjše nepravilnosti v izvornih podatkih. Opaziti je mogoče, da so določene ulice prekinjene, torej ne omogočajo prehoda tam, kjer z lastnih izkušenj vemo, da je prehod možen. Za poenostavitev predvidevamo, da je prehod možen med katerimakoli dvema križiščema (pri čemer je križišče lahko tudi konec ali začetek ulice), med katerima je manj kot 30 m zračne linije. To posplošitev si lahko privoščimo, ker je naš model v prvi meri prikrojen za osebo, ki se giblje peš ali s kolesom, zato lahko obidemo omejitve, ki veljajo za motorna vozila.

Druga formula, ki jo potrebujemo pa je formula za cepitev vektorja na dva dela oziroma iskanje središčne točke med dvema točkama. To bomo potrebovali zato, ker bo včasih neko daljšo ulico smiselno razdeliti na več krajših, ker bo model tako omogočal drugačne pristope. Cepitev vektorja na dva dela lahko izvajamo po naslednjih formulah:

$$x_s = \frac{1}{2}(x_1 + x_2)$$
$$y_s = \frac{1}{2}(y_1 + y_2)$$

Tako bomo nekatere daljše ceste preslikali iz $(x_1,y_1),(x_2,y_2)$ v $(x_1,y_1),(x_s,y_s)$ in $(x_s,y_s),(x_2,y_2)$.

3.3. Analiza modela s pomočjo metod TSP.

V terminologiji teorije grafov si lahko naš model predstavljamo kot graf **G**, ki ga tvorita podgrafa $\mathbf{G_u = (V,E)}$ in $\mathbf{G_h = (V_h)}$. Prvi tvori množico točk V, ki predstavljajo

križišča in začetek / konec ulice in množico lokov E, ki ponazarjajo povezave med temi točkami, torej ulice. V_h pa je množica točk stavb.

Če bi hoteli zares točen rezultat dolžine optimalne poti, bi se naslednji postopek že zelo približal dejanskemu rezultatu.

- Za vsako stavbo najdi najbližjo pot do pripadajoče ulice in jo izračunaj.
- To pot dodaj množici lokov E.
- Poišči najkrajšo pot v kateri obiščeš vse točke iz V_h, po lokih iz E.

Ker je ta pristop prezahteven, poskušajmo najti omejitve kompleksnosti operacij, ki nam še vedno dajejo zadovoljiv rezultat. Na začetku lahko eliminiramo prvi korak. To pa zato, ker lahko predpostavimo, da razdalja od posamezne stavbe do njene najbližje ulice ne variira preveč. To pomeni, da dejansko obiskujemo stavbo le do mesta, kjer smo še na ulici. Na koncu lahko rezultat popravimo tako, da predpostavimo nek konstanten čas, ki je potreben za to, da naredimo pot od ulice do vrat izbrane stavbe. Poleg tega pa ta razdalja glede na celotni model ni prevelika in jo lahko zanemarimo tudi zato, ker ne moremo točno vedeti, kako daleč do stavbe dejansko moramo priti, saj ne vemo kje se nahaja poštni nabiralnik. Ta problem bomo najenostavneje rešili tako, da bomo vsaki stavbi poiskali najbližjo točko, ki se nahaja na cesti in bomo koordinate stavbe kar preslikali v to točko. To dejansko pomeni, da bomo vsako stavbo »prestavili« na cesto.

Druga težava pa je večja in težje premostljiva. Upoštevati bi namreč morali, da so možni le prehodi med točkami (križišči), med katerimi je dejansko ulica. Naš graf je torej nepopolno povezan. Metode, ki jih znamo uporabljati za takšne modele učinkovito rešujejo modele do velikosti 10 ali nekaj 10 točk (npr. LINGO). V našem modelu pa je več kot 2000 točk.

Za reševanje TSP-jev takšne velikosti je eden izmed boljših programov program Concorde, ki lahko rešuje probleme, ki so še 100-krat večji od našega, zato smo ga tudi mi uporabili v naši analizi. Žal pa Concorde predvideva le popolnoma povezane grafe, to pomeni, da omogoča prehod med katerimakoli točkama v modelu (Applegate, 2004).

Da bi iztržili kar čim več od omenjenega programa si lahko pomagamo z naslednjim pristopom: vsako ulico ali njen del razdelimo na dovolj majhne segmente (npr. 10 metrov). S tem seveda število točk v modelu še naraste, vendar pa pridobimo bistveno prednost. S tem namreč TSP algoritem, ki ga uporablja

Concorde umetno prisilimo, da uporablja večinoma le poti, ki so dejansko ulice. Vsak prehod stran od dejanske poti (na primer na sosednjo vzporedno ulico) bo v primerjavi z manjšim, na primer desetmetrskim segmentom precej večji kot je bil pred transformacijo.

Po teh prilagoditvah lahko končno zaženemo program Concorde in si ogledamo rezultate (Slika 10):

Slika 10: Najkrajša pot po dani mreži ulic, določena s programom Concorde

Vir: Lasten prikaz.

Celotna dolžina tako dobljene poti je 53.870 metrov. Iz slike lahko vidimo, da pride do nepravilnosti predvsem tam, kjer je gostota točk večja. Kljub temu pa je dobljeni rezultat uporaben predvsem za to, da dobimo okvirno oceno dolžine poti.

Pričakujemo pa, da bomo točnejši rezultat dobili v naslednjem poglavju, ko bomo skušali model transformirati v problem kitajskega poštarja – ta namreč upošteva dejanske poti in povezave med točkami.

3.4. Analiza modela, ki ga rešujemo kot problem kitajskega poštarja

V tem delu bomo formulirali model kot problem kitajskega poštarja. Za razliko od prejšnjega dela, kjer smo se ukvarjali s posameznimi točkami, se bomo tokrat osredotočili na loke.

Spomnimo se, da moramo najprej iz prvotnega grafa G z ustreznimi transformacijami priti do Eulerjevega grafa G*. Najenostavnejši način, da dobimo Eulerjev graf iz danega grafa je, da preprosto podvojimo vse loke v izvornem grafu. Tako vsem točkam v grafu priredimo sodo stopnjo. Celotna Eulerjeva krožna pot je v tem primeru dvakrat daljša od vsote dolžin vseh ulic in znaša 2 x 62.368 = 124.736 metrov. Seveda pa to ni optimalna rešitev.

Boljšo rešitev bomo dosegli tako, da bomo podvojili le določene loke v grafu **G** tako, da dobimo graf, katerega točke bodo vse imele sodo vrednost. Za to moramo najprej za vsako točko v grafu ugotoviti njeno stopnjo. Število točk, ki imajo določeno stopnjo prikazuje spodnja tabela:

Tabela 5: Število točk z določeno stopnjo

Stopnja	Število točk
1	56
3	199
4	82
5	6

Vir: Lasten prikaz.

Logično je, da moramo podvojiti tiste loke, katerih začetna ali končna točka ima stopnjo 1 (to namreč pomeni, da moramo tak lok prehoditi dvakrat). Ko to naredimo, dobimo nove vrednosti za zgornjo tabelo.

Namesto da bi iskali kombinacije vseh točk z liho stopnjo in med njimi popolno ujemanje lahko postopamo tudi takole: najdemo vse tiste lihostopenjske točke, ki se z enim ali več loki povezujejo še z eno lihostopenjsko točko. Izmed teh lokov izberemo najkrajšega in ga podvojimo.

S tem smo prvotno število lihostopenjskih točk občutno zmanjšali, tako da med preostalimi mnogo lažje najdemo popolno ujemanje z najmanjšo skupno težo lokov.

Pri konstrukciji Eulerjevega grafa »na roke« bomo uporabili naslednji pristop:

1. označi stopnjo vsakega križišča z ustrezno vrednostjo: 1 za konce ulic, 3 za T-križišča in 4 ali več za ostala križišča,
2. križišča s sodo stopnjo označi tako, da obkrožiš številko, ki označuje pripadajočo stopnjo,
3. ustrezno podvajaj ulice med križišči z lihimi stopnjami, ki še niso obkrožene in označi podvojitev tako, da pobarvaš podvojene ulice.
4. številke, ki prikazujejo stopnje, med katerimi si podvojil ulico, obkroži, da s tem označiš transformacijo iz lihe v sodo stopnjo
5. ponavljaj od tretjega koraka, dokler niso vsa števila, ki ponazarjajo stopnjo križišč, obkrožena.

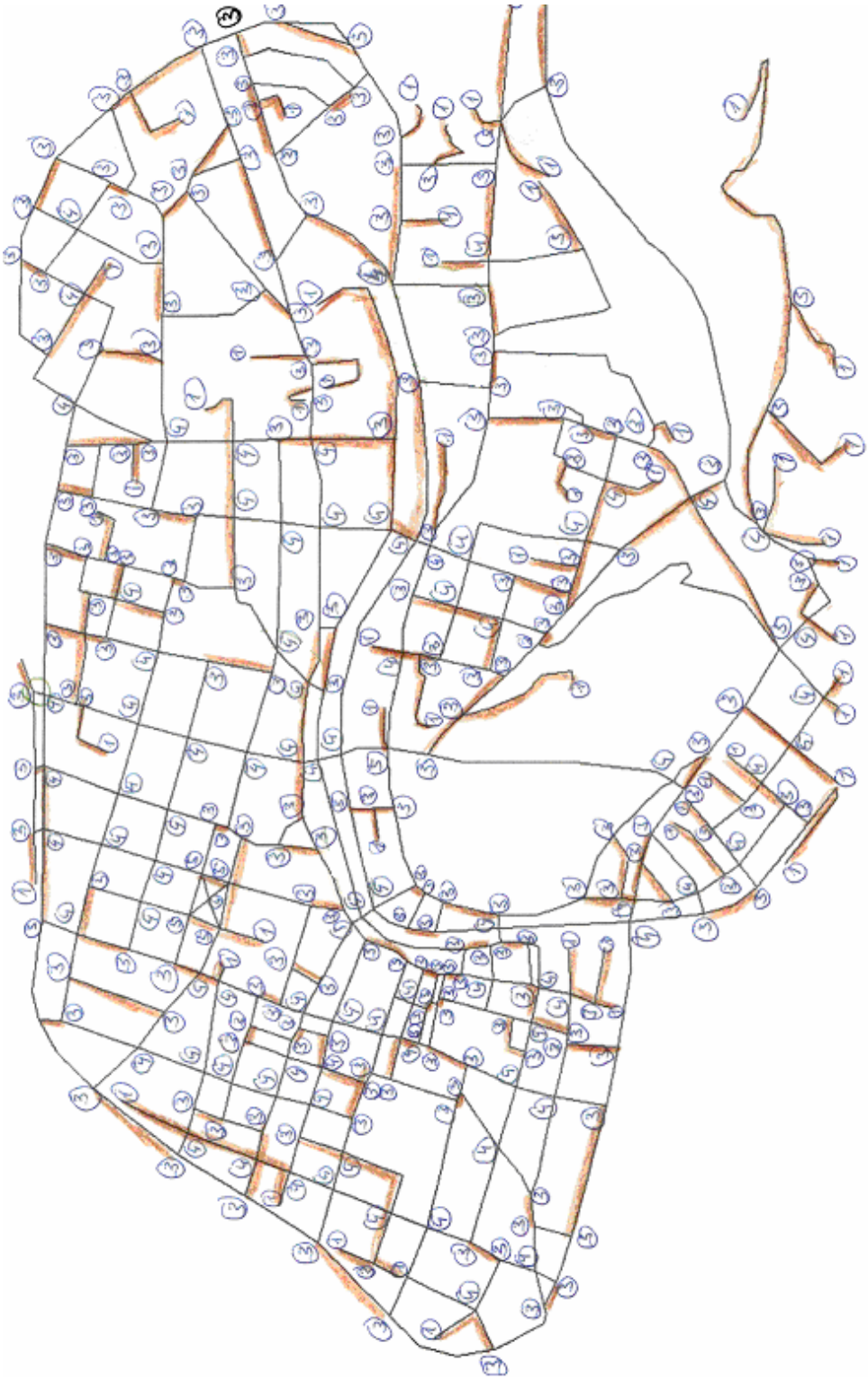
Na sliki 11 je prikazan graf ***G***, ki prikazuje obravnavane ulice na območju Ljubljana–Center, in križišča kot točke s pripadajočimi stopnjami.

Slika 11: prikaz mreže ulic na območju Ljubljana – Center in stopnje točk

Vir: Lasten prikaz.

Po končani transformaciji tako dobimo Eulerjev graf G*, ki ga prikazuje slika 12 (obarvani loki pomenijo podvojitev posameznega loka).

Slika 12: Eulerjev graf G* - prikaz mreže ulic na območju Ljubljana – Center s podvojenimi loki



Vir: Lasten prikaz.

Podvojene loke lahko prikažemo tudi v vektorski obliki. To bomo namreč potrebovali za izračun dodatne razdalje, ki jo je treba prišteti skupni dolžini vseh ulic, da dobimo celotno dolžino Eulerjeve poti.

Slika 13: Vektorski prikaz podvojenih ulic v danem grafu G

Vir: Lasten prikaz.

Vsota dolžin vseh lokov v izvornem grafu ***G*** je 62.368 metrov. Vsota dolžin tistih lokov, ki smo jih podvojili pa znaša 22.651 metrov. Tako da je celotna dolžina Eulerjeve poti 85.019 metrov, kar je 39.717 metrov manj kot dolžina poti, ki jo dobimo s podvajanjem vseh lokov v grafu. Pri tem naj še omenimo, da je možnih Eulerjevih poti po danem grafu mnogo, izbira poti pa seveda ne vpliva na njeno dolžino.

Določanje Eulerjeve poti oziroma podvajanje lokov smo opravili »na roke«, saj izdelava in implementacija algoritmov v računalniški obliki presega okvir te diplomske naloge. Kljub temu je prav, da možen postopek programiranja nakažemo v naslednjih korakih:

1. Najprej moramo pridobiti vektorske podatke za ulice in hišne številke. Te lahko naročimo pri Geodetski Upravi Republike Slovenije, ali pa jih izdelamo sami s pomočjo natančnega zemljevida in nekaterih programskih orodij.
2. Nato je potrebno preveriti, če so v podatkih kakšne napake in jih prečistiti glede na potrebe.
3. Naslednji korak je izdelava Dijkstrovega algoritma za iskanje najkrajše poti med kombinacijami točk. Pri tem lahko omejimo kompleksnost modela tako, da ne računamo razdalj za vse možne kombinacije točk, ampak zgolj za tiste, ki so dovolj skupaj, saj s podvajanjem poti med zelo oddaljenimi točkami zagotovo ne bomo dosegli optimalne rešitve.
4. Za vsako točko v grafu ugotovimo njeno stopnjo.
5. Točke z liho stopnjo povezujemo tako, da podvajamo loke med njimi. Pri tem moramo minimizirati skupno dolžino podvojenih lokov. Najkrajše poti iz točke 3 nam bodo v tem koraku v veliko pomoč.
6. Po grafu, ki ga dobimo v točki 5 izpeljemo Eulerjevo pot. S tem dobimo zaporedje točk, ki je tudi končna rešitev problema.

4. Sklep

V diplomski nalogi smo prikazali nekaj možnih pristopov pri iskanju optimalne poti po danem grafu, ki ponazarja mrežo ulic iz realnega sveta. Izmed obravnavanih konceptov se je kot najbolj uporaben in tudi natančen izkazal pristop, ki se uporablja pri reševanju problema kitajskega poštarja. S tem pristopom smo ugotovili, da je možna krožna pot po danih ulicah dolga 85.019 metrov in je 22.651 metrov daljša od vsote dolžin vseh ulic, ki znaša 62.368 metrov. Glede na najenostavnejšo krožno pot po mreži ulic, ki jo dobimo s podvajanjem vseh lokov v grafu, je dana rešitev krajša za 39.717 metrov. Zato lahko predpostavljamo, da je izračunana Eulerjeva krožna pot približno 10 do 30 kilometrov krajša od neke intuitivne poti, ki bi jo morebitni raznašalec izbiral med obhodom po danih ulicah.

Hkrati smo tudi predstavili algoritme, ki jih lahko uporabimo pri izdelavi aplikacije, s katero bi bilo možno izračunati optimalno krožno pot po mreži ulic tudi za problem

večjih dimenzij. Ob tem smo tudi navedli možne poenostavitve modela, s katerimi si lahko prihranimo veliko programiranja, še vedno pa dajejo zadovoljiv rezultat.

Ugotovili smo torej, da se pri opravljanju kurirskih oziroma poštnih storitev pri dovolj velikem naboru naslovnikov splača mrežo ulic definirati kot graf, na tem modelu pa uporabimo metode za reševanje problema kitajskega poštarja.

Literatura

1. Applegate David: Solving Traveling Salesman Problems. [URL:http://www.tsp.gatech.edu/index.html], 19.8.2004.
2. Baldacci Roberto, Maniezzo Vittorio: Exact methods based on node routing formulations for arc routing problems. Bologna : University of Bologna, 2004. 16 str.
3. Gross Jonathan, Yellen Jay: Graph Theory and its Applications. Boca Raton : CRC Press, 1999. 585 str.
4. Hvalica Dušan: Matematika 2. II.del. Ljubljana : Ekonomska Fakulteta, 1994. 241. str.
5. Hvalica Dušan, Rupnik Viljem: Matematika II. III.snopič: Grafi, Algoritmi. Ljubljana : Ekonomska Fakulteta, 1987. 98 str.
6. Lacomme P., Prins C., Sevaux M.: Multiobjective Capacitated Arc Routing Problem. [URL:www.univ-valenciennes.fr/sp/ sevaux/Publications/inp-lacomme-03.pdf], 18.11.2004.
7. Locke Steven C.: Graph Theory. Boca Raton : Florida Atlantic University. [URL:http://www.math.fau.edu/locke/graphthe.htm], 1.9.2004.
8. Ralphs Ted: The Vehicle Routing Problem. Lehigh : Lehigh University. [URL:http://www.lehigh.edu/~tkr2/research/applications.html], 25.5.2004.
9. Saražin Ines: Teorija grafov in uporaba pri poslovnem odločanju. Diplomska naloga. Ljubljana : Ekonomska fakulteta, 1984. 57 str.
10. Skiena Steven S.: The Algorithm Design Manual. [URL:http://www2.toki.or.id/book/AlgDesignManual/BOOK/BOOK2/NODE74.HTM#SECTION024710000000000000], 16.11.2004.
11. Timbleby Harold: The Directed Chinese Postman Problem. London : University College London Interaction Centre, 2003. 17 str.
12. Winston Wayne L.: Operations Research: Applications and Algorithms. Fourth Edition. Belmont : Thomson Brooks/Cole, 2004. 1418 str.

Priloga 1: Rezultati iz Linga za rešitev problema najkrajše poti za graf G

Global optimal solution found at iteration: 0
Objective value: 11.00000

Tabela 1: Rezultati iz Linga za rešitev problema najkrajše poti za graf G

VOLUME(A, A)	0,000	0,000
VOLUME(A, B)	1,000	0,000
VOLUME(A, C)	0,000	1000,000
VOLUME(A, D)	0,000	1000,000
VOLUME(A, E)	0,000	992,000
VOLUME(A, F)	0,000	7,000
VOLUME(A, G)	0,000	1000,000
VOLUME(A, H)	0,000	1000,000
VOLUME(A, I)	0,000	1000,000
VOLUME(A, J)	0,000	989,000
VOLUME(B, A)	0,000	1005,000
VOLUME(B, B)	0,000	0,000
VOLUME(B, C)	0,000	11,000
VOLUME(B, D)	0,000	1005,000
VOLUME(B, E)	1,000	0,000
VOLUME(B, F)	0,000	1005,000
VOLUME(B, G)	0,000	1005,000
VOLUME(B, H)	0,000	1005,000
VOLUME(B, I)	0,000	1005,000
VOLUME(B, J)	0,000	994,000
VOLUME(C, A)	0,000	1000,000
VOLUME(C, B)	0,000	995,000
VOLUME(C, C)	1,000	0,000
VOLUME(C, D)	0,000	5,000
VOLUME(C, E)	0,000	992,000
VOLUME(C, F)	0,000	1000,000
VOLUME(C, G)	0,000	1000,000
VOLUME(C, H)	0,000	7,000
VOLUME(C, I)	0,000	1000,000
VOLUME(C, J)	0,000	989,000
VOLUME(D, A)	0,000	1000,000
VOLUME(D, B)	0,000	995,000
VOLUME(D, C)	0,000	1000,000
VOLUME(D, D)	1,000	0,000
VOLUME(D, E)	0,000	992,000
VOLUME(D, F)	0,000	1000,000
VOLUME(D, G)	0,000	1000,000
VOLUME(D, H)	0,000	1000,000
VOLUME(D, I)	0,000	1000,000
VOLUME(D, J)	0,000	989,000
VOLUME(E, A)	0,000	1008,000

Nadaljevanje tabele 1

VOLUME(E, B)	0,000	1003,000
VOLUME(E, C)	0,000	1008,000
VOLUME(E, D)	0,000	1008,000
VOLUME(E, E)	0,000	0,000
VOLUME(E, F)	0,000	1008,000
VOLUME(E, G)	0,000	12,000
VOLUME(E, H)	0,000	1008,000
VOLUME(E, I)	0,000	1008,000
VOLUME(E, J)	1,000	0,000
VOLUME(F, A)	0,000	1000,000
VOLUME(F, B)	0,000	995,000
VOLUME(F, C)	0,000	1000,000
VOLUME(F, D)	0,000	1000,000
VOLUME(F, E)	0,000	992,000
VOLUME(F, F)	1,000	0,000
VOLUME(F, G)	0,000	5,000
VOLUME(F, H)	0,000	1000,000
VOLUME(F, I)	0,000	1000,000
VOLUME(F, J)	0,000	989,000
VOLUME(G, A)	0,000	1000,000
VOLUME(G, B)	0,000	995,000
VOLUME(G, C)	0,000	1000,000
VOLUME(G, D)	0,000	1000,000
VOLUME(G, E)	0,000	992,000
VOLUME(G, F)	0,000	1000,000
VOLUME(G, G)	1,000	0,000
VOLUME(G, H)	0,000	7,000
VOLUME(G, I)	0,000	1000,000
VOLUME(G, J)	0,000	989,000
VOLUME(H, A)	0,000	1000,000
VOLUME(H, B)	0,000	995,000
VOLUME(H, C)	0,000	1000,000
VOLUME(H, D)	0,000	1000,000
VOLUME(H, E)	0,000	992,000
VOLUME(H, F)	0,000	1000,000
VOLUME(H, G)	0,000	1000,000
VOLUME(H, H)	1,000	0,000
VOLUME(H, I)	0,000	2,000
VOLUME(H, J)	0,000	989,000
VOLUME(I, A)	0,000	1000,000
VOLUME(I, B)	0,000	995,000
VOLUME(I, C)	0,000	1000,000
VOLUME(I, D)	0,000	1000,000

Nadaljevanje tabele 1

VOLUME(I, E)	0,000	992,000
VOLUME(I, F)	0,000	1000,000
VOLUME(I, G)	0,000	1000,000
VOLUME(I, H)	0,000	1000,000
VOLUME(I, I)	1,000	0,000
VOLUME(I, J)	0,000	989,000
VOLUME(J, A)	0,000	1011,000
VOLUME(J, B)	0,000	1006,000
VOLUME(J, C)	0,000	1011,000
VOLUME(J, D)	0,000	1011,000
VOLUME(J, E)	0,000	1003,000
VOLUME(J, F)	0,000	1011,000
VOLUME(J, G)	0,000	1011,000
VOLUME(J, H)	0,000	1011,000
VOLUME(J, I)	0,000	1011,000
VOLUME(J, J)	0,000	0,000

Vir: Lastni računalniški izračuni.

V vrstici »Objective value« preberemo dolžino izračunane najkrajše poti, ki v našem primeru znaša 11 enot. V vrsticah »VOLUME« pa razberemo, kateri rob grafa je dejansko vključen v izračunani poti. Vrednost 1 v drugem stolpcu pomeni, da je ta rob vključen v izračunani poti, vrednost 0 pa pomeni, da ta rob ni vključen v izračunani poti.

Priloga 2: Izračun najkrajše poti v primeru 2 na podlagi Dijkstrovega algoritma.

Iteracija 1:

dist[a] = 0 → a(0) P(ab) = 5 P(af) = 7

Rob ab dodamo k drevesu T.

Iteracija 2:

dist[b] = 5 → b(5) P(af) = 7 P(bc) = 11 P(be) = 8

Rob af dodamo k drevesu T.

Iteracija 3:

dist[f] = 7 → f(7) *P(bc) = 11* *P(be) = 8* *P(fg) = 12*

Rob be dodamo k drevesu T.

Iteracija 4:

dist[e] = 8 → e(8) *P(bc) = 11* *P(ej) = 11* *P(fg) = 12* *P(eg) = 13*

Rob bc dodamo k drevesu T.

Iteracija 5:

dist[c] = 11 → c(11) *P(ej) = 11* *P(fg) = 12* *P(ch) = 18* *P(cd) = 16*
P(eg) = 13

Rob ej dodamo k drevesu T.

Iteracija 6:

dist[j] = 11→ j(11) *P(fg) = 12* *P(ch) = 18* *P(cd) = 16* *P(eg) = 13*

Rob fg dodamo k drevesu T.

Iteracija 7:

dist[g] = 12 → g(12) *P(gh) = 19* *P(ch) = 18* *P(cd) = 16*

Rob cd dodamo k drevesu T.

Iteracija 8:

dist[d] = 16 → d(16) *P(gh) = 19* *P(ch) = 18*

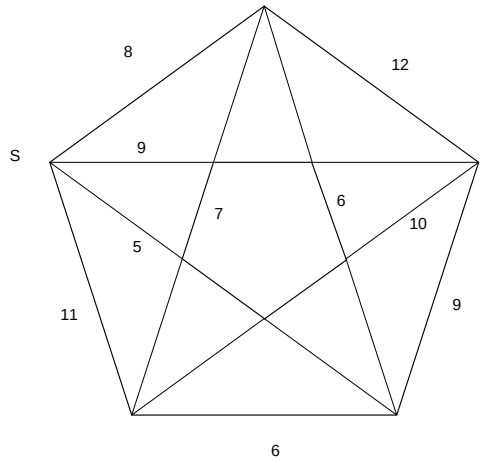
Rob ch dodamo k drevesu T.

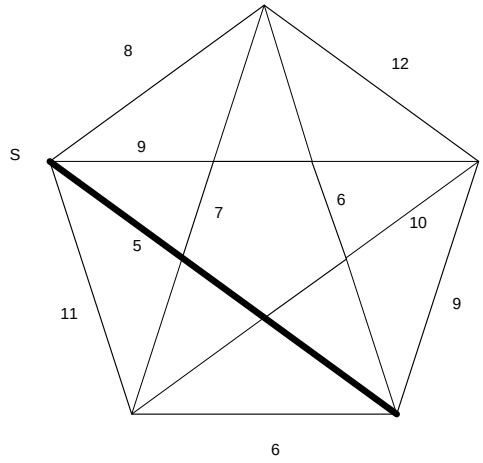
Iteracija 9:

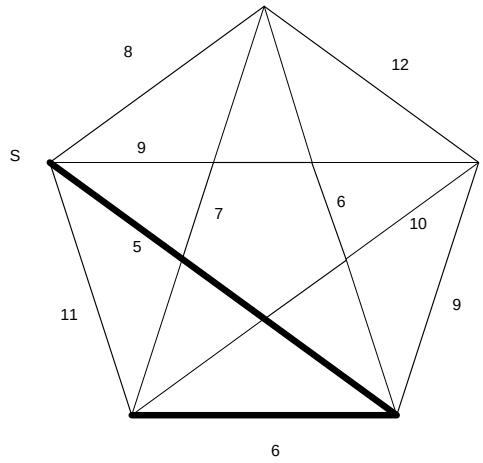
dist[h] = 18 → h(18) *P(hi) = 20*

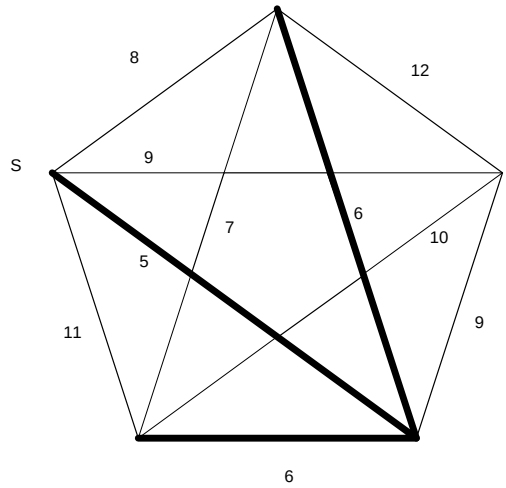
Rob hi dodamo k drevesu T. In dist[i] = 20 → i(20)

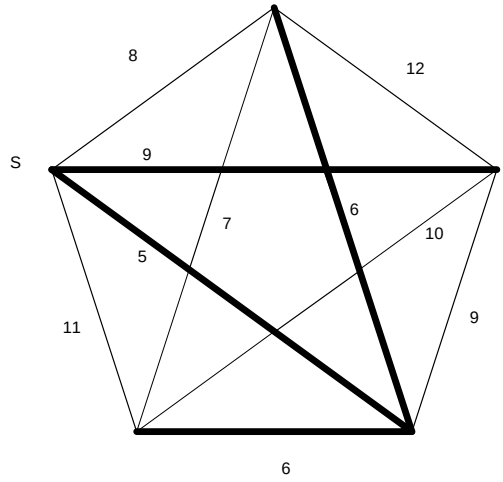
Priloga 3: Ilustracija poteka algoritma Primovega drevesa.











Vir: Gross, 1999, str. 147.

Priloga 4: Primer obravnave stroškovnega vidika problema s pomočjo teorije iger

Predstavljajmo si situacijo, v kateri želi naročnik mesečno poslati neko naslovljeno pošiljko do vsakega gospodinjstva v Sloveniji naslednjih 15 let. Predpostavljajmo, da je možno pošiljko dostaviti na dva načina: kot običajno poštno pošiljko, ali pa pri kurirski službi, katera je specializirana zgolj za dostavo v urbanih naseljih. V drugem primeru lahko kurirska služba odda zgolj tisti del pošiljk, kjer so naslovniki v urbanih naseljih, preostali del pa v vsakem primeru oddamo kot navadno poštno pošiljko. Število gospodinjstev v Sloveniji ob popisu leta 2002 je bilo 684.847, od

tega 370.932 v mestih (Statistični Letopis 2003, Ljubljana: Statistični Urad Republike Slovenije).

Predpostavljamo torej dva igralca: Pošta in Kurir. Igralec Pošta ima na voljo pet strategij pri oblikovanju cene poštne pošiljke. Ceno lahko nastavi na 45, 40, 35, 30 ali 25 SIT po pošiljki, vendar le za celotno količino pošiljk (694.847). Igralec Kurir pa ima na voljo dve strategiji: da investira v projekt ali pa da ne investira v projekt. Predpostavimo, da znaša Kurirjeva začetna investicija izražena v mesečni anuiteti (recimo, da bo vzel kredit za celotno dobo pričakovanih prihodkov projekta – 15 let) 600.000 SIT. Njegovi skupni mesečni prihodki, v primeru, da bo izbran v poslu bodo 7.000.000 SIT mesečno. Naslovnik se bo odločil med dvema možnostma – 1.) vse pošiljke prevzame Pošta ali 2.) del pošiljk prevzame Pošta, del Kurir. Naročnik se bo odločil na podlagi celotnih skupnih stroškov. Celotni stroški v primeru, da vse pošiljke prevzame Pošta so produkt cene in količine:

$C = P_{pošta} * N$
$C_{45} = 45 * 684.847 = 30.818.115$
$C_{40} = 40 * 684.847 = 27.393.880$
$C_{35} = 35 * 684.847 = 23.969.645$
$C_{30} = 30 * 684.847 = 20.545.410$
$C_{25} = 25 * 684.847 = 17.121.175$

Stroški alternativne variante, da del pošiljk preda Kurirju pa znašajo:

$C_K = 7.000.000 + (684.847 – 370.932) * 45 = 21.726.175$

Izraz v oklepaju je namreč število pošiljk neurbanim gospodinjstvom, ki jih v vsakem primeru oddamo na pošto.

Če Kurir ne investira, je seveda njegov prihodek enak 0. Omenjene vrednosti lahko prikažemo v igri dimenzije 2x5:

Tabela 2: Primer igre 2 x 5 za obravnavani problem

		Kurir			
		Investiraj		Ne investiraj	
Pošta	1 (P=45)	14.126.175	7.600.000	30.818.115	0
	2 (P=40)	14.126.175	7.600.000	27.393.880	0
	3 (P=35)	14.126.175	7.600.000	23.969.645	0
	4 (P=30)	20.545.410	-600.000	20.545.410	0
	5 (P=20)	17.121.175	-600.000	17.121.175	0

Vir: Lasten prikaz.

Če bo Kurir investiral, bo dobil posel v primerih, da Pošta izbere strategijo 1, 2 ali 3 (to so strategije visoke cene). Če Pošta dovolj zniža ceno, bo izbrana v vsakem primeru, ne glede na strategijo Kurirja. Hitro ugotovimo, da za Pošto strategija 1 dominira strategijam 2 in 3, ter da strategija 4 dominira strategiji 5. Tako dobimo zgolj igro dimenzije 2x2:

Tabela 2: Primer igre 2 x 2 za obravnavani problem

		Kurir			
		Investiraj		Ne investiraj	
Pošta	1 (P=45)	14.126.175	7.600.000	30.818.115	0
	4 (P=30)	20.545.410	-600.000	20.545.410	0

Vir: Lasten prikaz.

To je igra z nekonstantno vsoto. Vidimo, da igra nima Nashovega ravnovesja, da torej v vsaki kombinaciji strategij vsaj eden izmed igralcev lahko pridobi, če spremeni svojo strategijo. Kljub temu pa lahko na podlagi te igre predvidimo razmišljanje obeh igralcev. Recimo, da igralca izbereta strategiji tako, da se nahajata v desnem spodnjem kotu. Igralec Pošta lahko sedaj spremeni strategijo v 1, tako da poveča svoj prihodek. Vendar nato lahko igralec Kurir spremeni svojo strategijo v Investiraj in s tem tudi on pridobi, Pošta pa izgubi. Sedaj lahko Pošta zopet spremeni svojo strategijo in zniža ceno v svojo korist. Ker je Kurir že investiral se ne more več odločiti za spremembo strategije, tako da je končno stanje v spodnjem levem kvadratu. Pošta tako zniža ceno, da pridobi posel, Kurir pa investira kljub svoji izgubi.

Priloga 5: Slovarček angleških izrazov

Chinese Postman problem	problem kitajskega poštarja
degree	stopnja, valenca
edges	loki, veje, povezave
endpoints	krajišči loka
greedy algorithm	pohlepni algoritem
nodes	točke, vozli, vozlišča, vrhovi
Shortest Path Problem	problem najkrajše poti
tour (Eulerian)	(Eulerjeva) krožna pot
trail (Eulerian)	(Eulerjeva) pot
transshipment points	vmesne točke
transshipment problem	problem odpošiljanja blaga
Traveling Salesperson Problem, TSP	problem trgovskega potnika
valence	stopnja, valenca
vertices	točke, vozli, vozlišča, vrhovi