Implementacija mrežne simpleks metode

Milašinović, Jure

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Jure Milašinović

IMPLEMENTACIJA MREŽNE SIMPLEKS METODE

Diplomski rad

Voditelj rada:
dr. sc. Marko Vrdoljak, izv. prof.

Zagreb, rujan, 2016

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

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

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Diplomski rad posvećujem roditeljima Miri i Zvonki, braći Stjepanu i Anti i sestrama Tomislavi i Mariji.

Zahvaljujem se dragom Bogu koji me vodio kroz cijelo obrazovanje i koji mi je davao snagu i ljubav. Zahvaljujem se svojoj obitelji koja mi je pružala potporu i koja je bila sa mnom cijelo vrijeme. Posebno se zahvaljujem mentoru profesoru Marku Vrdoljaku koji mi je pomogao da napišem rad. Također se zahvaljujem i profesoru Lavoslavu Čakloviću koji mi je, uz mentora, stalno pomagao i bio na raspolaganju.

Sadržaj

Uvod

Za praktične aplikacije, daleko najkorisniji algoritam za rješavanje linearnog programiranja je proslavljena simpleks metoda. S profesionalnom implementacijom doseže izvanredne performanse: problemi sa ≈ 1000 varijabli i ≈ 1000 uvjeta mogu se riješiti unutar 0.1 do 0.5 sekundi([3]). Ta spoznaja sugerira primjenu metode i na probleme iz teorije grafova. Štoviše, bitne zadaće mrežne optimizacije mogu se iskazati kao zadaće linearnog programiranja. Primjer takvih zadaća su: određivanje najkraćeg puta, određivanje maksimalnog toka, optimalnog toka i optimalne cirkulacije. Valjalo bi ipak napomenuti da za jako velike probleme linearnog programiranja metoda unutrašnje točke može biti u prednosti u odnosu na simpleks metodu. Što se tiče problema u grafovima, tu je općenito prilagođena simpleks metoda (mrežna simpleks metoda) u prednosti.

Standardna primjena simpleks metode za rješavanje problema u grafovima je često vrlo nepraktična. Razlog tome je to što bi linearno programiranje bilo dosta nezgrapno i vrlo degenerirano. S druge strane, posebnost zadaće mrežne optimizacije čini osnovni korak simpleks metode vrlo jednostavnim za izvođenje i vrlo preglednim u terminima grafa.

Mrežna simpleks metoda se obično formulira u terminima standardnog problema koji će biti uveden u sljedećem poglavlju pod nazivom *problem minimizacije troškova toka na mreži*. Svi ostali praktični problemi od interesa se mogu laganom transformacijom svesti na ovaj problem. Također ćemo vidjeti primjer takvih transformacija.

Dugo vrijeme je egzistencija dokaza efikasnosti mrežne simpleks metode bila vodeći otvoreni problem u teoriji složenosti, iako je očito bila najefikasnija praktična metoda za problem minimizacije troškova toka na mreži. Problem je napokon riješio Orlin koji je dao implementaciju složenosti $O(|V|^2|E|\log(|V|C))$, gdje je $C = \max\{|c(e)|: e \in E\}$ maksimalna vrijednost jedinične cijene koja se pojavljuje u grafu $G = (V, E)$. Za više o složenosti valjalo bi pogledati [3].

U ovom diplomskom radu, odlučili smo naglasak staviti na teoriju grafova, dok ćemo izbjegavati, koliko to bude moguće, teoriju linearnog programiranja. Gledajući iz te perspektive, imamo dosta sreće što se mrežna simpleks metoda može u cijelosti riješiti u terminima teorije grafova, bez potrebe pozivanja na linearno programiranje.

Opisat ćemo mrežnu simpleks metodu. Dat ćemo konkretan algoritam koji rješava zadaću mrežne optimizacije i koji će se zasnivati na uvođenje pomoćnog problema. Riješit

ćemo slučaj pojave degeneracije tako da ćemo uvesti pravilo zadnjeg blokirajućeg luka za odabir izlaznog luka iz pivotnog ciklusa. Definirat ćemo indekse stabla koje će nam olakšati pronalaženje pivotnog ciklusa i mnoge druge operacije. Zatim ćemo dati konkretne efikasne implementacije glavnih dijelova algoritma. To su pronalaženje pivotnog ciklusa, odabir izlaznog luka po spomenutom pravilu i ažuriranje potencijala. Nakon što opišemo efikasne implementacije slijedi demonstracija PyQt5 aplikacije koja rješava, u interakciji s korisnikom, zadaću mrežne optimizacije. Aplikacija je napravljena u programskom jeziku Python, koristeći Qt5 framework za GUI i korištenjem biblioteke NetworkX za rad s grafovima. Demonstrirat ćemo rad aplikacije rješavanjem konkretnog problema minimizacije troškova toka na mreži.

Poglavlje 1

Mrežna simpleks metoda

1.1 Minimizacija troškova toka na mreži

Problem minimizacije troškova toka na mreži (MCFP) je među najosnovnijim problemima toka i cirkulacije. Svaki takav problem se može lagano transformirati na problem minimizacije troškova toka na mreži, koji se vrlo učinkovito može riješiti pomoću mrežne simpleks metode. Problem minimizacije troškova toka na mreži je često generalizacija transportnog problema i problema minimizacije cirkulacije na mreži. Sada slijedi definicija našeg problema.

Definicija 1.1.1 (Problem minimizacije troškova toka na mreži)**.** *Neka je $G = (V, E)$ povezani usmjereni graf. Neka su:*

- *donja i gornja funkcija kapaciteta $l\colon E \to \mathbb{R}$ i $u\colon E \to \mathbb{R}$, respektivno,*
- *funkcija troška $c\colon E \to \mathbb{R}$,*
- *funkcija napajanja $b\colon V \to \mathbb{R}$ za koju vrijedi $\sum_{v \in V} b(v) = 0$.*

Problem minimizacije troškova toka na mreži (MCFP) se svodi na određivanje optimalnog toka $x\colon E \to \mathbb{R}$ tako da je ukupan trošak toka minimiziran, tj.

$$c(x) = \sum_{e \in E} c(e)x(e) \to \min$$

i koja zadovoljava sljedeća dva uvjeta:

(F1) $l(e) \leq x(e) \leq u(e), \forall e \in E,$

(F2) $\sum_{e^- = v} x(e) - \sum_{e^+ = v} x(e) = b(v), \forall v \in V,$

Vrhovi s negativnim napajanjem su vrhovi potražnje, a vrhovi s nenegativnim napajanjem su vrhovi ponude. Svi ostali vrhovi su Kirrchoffovi [1]. Zapišimo to formalno:

Neka je n broj vrhova $V = \{v_1, v_2, \cdots v_n\}$, m broj lukova $E = \{e_1, e_2, \cdots e_m\}$.

- c_{ij} - cijena jediničnog transporta na luku $(i, j) \in E$,
- l_{ij} - donja ograda za tok na luku $(i, j) \in E$,
- u_{ij} - gornja ograda za tok na luku $(i, j) \in E$,
- b_i - napajanje luka i:
  - $b_i > 0$ vrh i je vrh ponude (dobitak na vrhu i),
  - $b_i < 0$ vrh i je vrh potražnje (gubitak na vrhu i),
  - $b_i = 0$ vrh i je Kirchoffov (tranzitni).
- x_{ij} varijabla odluke - tok na luku $(i, j) \in E$.

Problem minimizacije troškova toka na mreži možemo kratko zapisati na sljedeći način:

$$\begin{cases} \sum\limits_{e \in E} c(e)x(e) \to \min \\ l(e) \leq x(e) \leq u(e), \forall e \in E \\ \sum\limits_{e^- = v} x(e) - \sum\limits_{e^+ = v} x(e) = b(v), \forall v \in V \end{cases} \tag{P}$$

Tok je preslikavanje $x\colon E \to \mathbb{R}$ koje zadovoljava uvjete za napajanje za svaki vrh $v \in V$. Ako dodatno zadovoljava uvjete na kapacitet za svaki $e \in E$, tada govorimo o dopustivom toku. Problem (P) traži dopustivi tok takav da je ukupan trošak minimalan.

U Tablici 1.1 smo naveli neke standardne probleme koji se mogu transformirati u problem (P).

Slika 1.1 prikazuje jedan primjer problema (P). Svaki je luk označen uređenom trojkom (l_{ij}, u_{ij}, c_{ij}). Svaki je vrh označen s nazivom i b_i. Zadebljanje linije označava smjer luka.

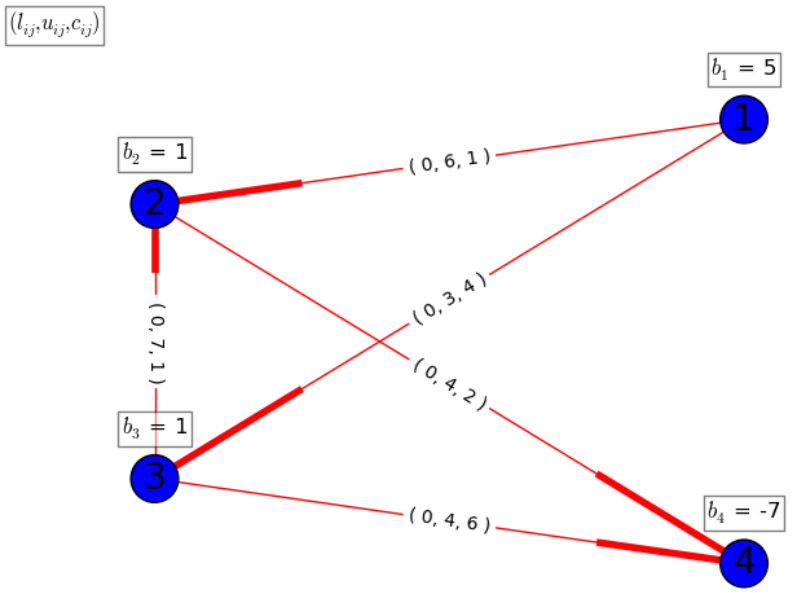
Prije nego što pokušamo naći takav tok da je ukupan trošak minimalan, prvo moramo provjeriti postoji li uopće dopustiv tok. To nije nimalo trivijalno zbog l i u. Osnovni uvjet postojanja dopustivog toka je $\sum_i b_i = 0$, tj. ukupna potražnja je jednaka ukupnoj ponudi.

U tu svrhu ćemo iskazati teorem koji nam govori kada će za problem (P) postojati dopustiv tok. Prije iskaza teorema, slijedi definicija.

[1] Vrh v je Kirrchoffov ako je ukupan tok koji ulazi u vrh v jednak ukupnom toku koji izlazi iz vrha v. Drugim riječima: $\sum_{e^+ = v} x(e) = \sum_{e^- = v} x(e)$

Problem	Specifikacija
Najkraći put od vrha s do vrha t uzimajući u obzir duljinu $w(e)$	$l \equiv 0$; $u \equiv 1$; $c(e) = w(e)$; $b(s) = 1$; $b(t) = -1$; $b(v) = 0$ za svaki $v \neq s, t$
Maksimalni tok u transportnoj mreži $N = (G, c_1, i, p)$	$l \equiv 0$; $u(e) = c_1(e)$; $b(v) = 0$ za svaki $v \in V$; $c(e) = 0$ za svaki $e \in E$; dodati novi luk (p, i) s $c(p, i) = -1$, $l(p, i) = 0$ i $u(p, i) = \sum_{e^- = i} c_1(e)$
Problem cirkulacije	$b(v) = 0$ za svaki $v \in V$; ostali podaci kako su zadani

Tablica 1.1: Transformacija nekih problema



Slika 1.1: Problem minimizacije troškova na mreži

Definicija 1.1.2. ***Kapacitet reza*** (S, S^c) *za podskup* $S \subset V$ *je:*

$$c(S, S^c) = \sum_{\substack{e^- \in S \\ e^+ \in S^c}} u(e) - \sum_{\substack{e^- \in S^c \\ e^+ \in S}} l(e)$$

pri čemu je $c\colon E \to [0, +\infty >$.

Sada slijedi teorem o postojanju dopustivog toka.

Teorem 1.1.3. *Problem (P) ima dopustivi tok ako i samo ako za svaki rez $V = S \dot\cup T$ vrijedi:*

$$c(S,T) \geq \sum_{v\in S} b(v). \tag{1.1}$$

Dokaz. 1. Neka je G polazni graf s pripadnim l, u, b. Definirajmo novi graf tako da l reduciramo na nulu:

$$\begin{aligned} &l'(e) = 0, e \in E \\ &u'(e) = u(e) - l(e), e \in E \\ &b'_i = b_i - \sum_{e^-=i} l(e) + \sum_{e^+=i} l(e), i \in V \end{aligned}$$

Za svaki luk činimo transformaciju $b'_i = b_i - l_{ij}$, $b'_j = b_j + l_{ij}$. Novi graf označimo s G'.

2. Uvedimo dva nova vrha koje nazivamo izvor i ponor s oznakama i i p. Zatim, dodajmo nove lukove iz i prema j ako je $b'_j > 0$ (kapacitet luka b'_j), odnosno iz vrhova j za koje je $b'_j < 0$ prema p (kapacitet luka $-b'_j$).

Transportna mreža je usmjereni graf s nenegativnim težinama u kojem postoji točno jedan vrh u kojeg niti jedan luk ne ulazi (izvor - i) i točno jedan luk iz kojeg niti jedan luk ne izlazi (ponor - p). Više pogledati točku 6.1 u [1]- ključna je Lema 6.1.2.
Nije teško uočiti: dopustivi tok x' na G' generira tok na transportnoj mreži tako da novim lukovima stavimo b'_j odnosno $-b'_j$, a to je očito maksimalan tok[2] za transportnu mrežu. Dakle,
x' je dopustivi tok $\Leftrightarrow$ pripadni tok na transportnoj mreži ima (maksimalnu) vrijednost
$\sum\limits_{\substack{j\\ b'_j>0}} b'_j = -\sum\limits_{\substack{j\\ b'_j<0}} b'_j$ (suma b-ova je 0)
$\Leftrightarrow$ za svaki rez (S,T) od G' ($V = S \dot\cup T$) vrijedi (slaba dualnost)
$c(S\cup\{i\}, T\cup\{p\}) \geq \underbrace{\sum\limits_{\substack{j\\ b'_j>0}} b'_j}_{(=\text{val } f^*)}$.
maksimalna tok

($\boxed{\Rightarrow}$ je zapravo slaba dualnost, a $\boxed{\Leftarrow}$ vrijedi jer za i-p rez $(\{i\}, \{i\}^c)$ imamo jednakost, pa je to min i-p rez opet prema teoremu slabe dualnosti[3])
$c(S\cup\{i\}, T\cup\{p\}) = \underbrace{c'(S,T)}_{\text{kapacitet u } G'} + \sum\limits_{\substack{b'_j>0\\ j\notin S}} b'_j + \sum\limits_{\substack{b'_j<0\\ j\in S}} (-b'_j) \geq \sum\limits_{\substack{j\\ b'_j>0}} b'_j \Leftrightarrow \exists$ dopustivi tok

[2]Tok s najvećom vrijednosti toka.

[3]Za svaki i-p rez kapacitet je $\geq$ od vrijednosti bilo kojeg toka, pa tako i maksimalnog.

$$\Leftrightarrow \sum_{\substack{e^- \in S \\ e^+ \in T}} (u(e) - l(e)) \geq \sum_{\substack{b'_j > 0 \\ j \in S}} b'_j + \sum_{\substack{b'_j < 0 \\ j \in S}} b'_j = \sum_{j \in S} b'_j$$

$$\Leftrightarrow \sum_{\substack{e^- \in S \\ e^+ \in T}} (u(e) - l(e)) \geq \sum_{j \in S} [b_j - \sum_{e^- = j} l(e) + \sum_{e^+ = j} l(e)] = \sum_{j \in S} b_j - \sum_{e^- \in S} l(e) + \sum_{e^+ \in S} l(e)$$

$$\Leftrightarrow \underbrace{\sum_{\substack{e^- \in S \\ e^+ \in T}} u(e) - \sum_{\substack{e^+ \in S \\ e^- \in T}} l(e)}_{c(S,T)} \geq \sum_{j \in S} b_j \qquad \square$$

1.2 Bazičan tok

Razmotrimo naš problem (P) u terminima usmjerenoga grafa G.

Definicija 1.2.1. *Neka je T generirajuće stablo za usmjereni graf G. Dopustivi tok x nazivamo **bazičan tok** (uz zadani T) ako vrijednost toka $x(e)$ za svaki luk e izvan T odgovara donjoj $l(e)$ ili gornjoj $u(e)$ ogradi za tok.*

Vidjet ćemo da postojanje dopustivog toka povlači postojanje optimalnog bazičnog toka. Štoviše, ne-optimalan bazičan tok se uvijek može poboljšati boljim rješenjem tako da zamijenimo luk koji pripada T lukom koji ne pripada. Mrežna simpleks metoda koristi operacije toga tipa prilikom traženja optimalnog bazičnoga toka. To su jednostavne operacije koje se mogu efikasno implementirati i koje predstavljaju glavnu prednost mrežne simpleks metode u odnosu na ostale.

Definirat ćemo osnovne pojmove koji su nam bitni u nastavku rada.

Definicija 1.2.2. *Neka je x dopustiv tok. Kažemo da je luk $e_k = (i, j)$ **vezan luk** ako je $x(e_k) = l(e_k)$ ili $x(e_k) = u(e_k)$. Ukoliko luk nije vezan tada kažemo da je **slobodan luk**.*

Napomena 1.2.3. *Uvođenjem definicije slobodnog luka možemo napisati sljedeće: x je bazičan tok ako i samo ako postoji generirajuće stablo T koje sadrži sve slobodne lukove.*

Ne zahtijevamo da svi lukovi stabla T moraju biti slobodni.
Sada ćemo dokazati sljedeći važan rezultat koji smo već spomenuli.

Teorem 1.2.4. *Neka je zadan problem (P) s usmjerenim grafom G, kao iz Definicije 1.1.1. Tada vrijedi:*
Ako za problem (P) postoji dopustivi tok, tada postoji i optimalan bazičan tok.

Dokaz. Kako je skup dopustivih tokova za problem (P) kompaktan podskup od $\mathbb{R}^{|E|}$ i kako je funkcija troškova neprekidna, tada postoji optimalan dopustivi tok x. (v. Teorem 6.1.6 u [1])
Pretpostavimo da nisu svi slobodni lukovi, s obzirom na x, sadržani u generirajućem stablu

za G, inače, nemamo što dokazati.
Tada postoji (neusmjereni) ciklus γ u G koji sadrži samo slobodne lukove, stoga možemo povećati x tako da prekinemo ciklus. To je neovisno o tome koja je, od dvije moguće, orijentacija ciklusa γ. (Više o tome u sljedećem potpoglavlju). Nakon povećanja toka na tom ciklusu, barem će jedan luk postati vezan. Novi tok x' je optimalniji od prethodnog. Ta operacija smanjuje broj slobodnih lukova barem za jedan. Nastavimo primjenjivati tu operaciju dok god u G postoji slobodan ciklus[4] i na taj način ćemo doći do optimalnog toka x' čiji skup slobodnih lukova ne sadrži ciklus. Kako je G povezani usmjereni graf koji ne sadrži slobodne cikluse, x' je (optimalan) bazičan tok. □

Postoji konačno mnogo izbora generirajućeg stabla za G. Mi ćemo odabrati ono generirajuće stablo T za koje vrijedi sljedeće: Svaki luk $e \in E \setminus T$ je vezan. Vezano time, iskazat ćemo lemu s kojom ćemo definirati dva nova skupa.

Lema 1.2.5. *Neka je zadan problem* (P) *s usmjerenim grafom G. Neka je T generirajuće stablo za G, te neka je $L \dot{\cup} U = E \setminus T$ bilo koja particija skupa $E \setminus T$. Tada postoji jedinstveni tok x takav da vrijedi sljedeće:*

- $x(e) = l(e)$, $\forall e \in L$,
- $x(e) = u(e)$, $\forall e \in U$.

Dokaz. Stavimo:

$$x(e) = l(e) \text{ za svaki } e \in L \text{ i } x(e) = u(e) \text{ za svaki } e \in U \tag{1.2}$$

Neka je v bilo koji list stabla T, i neka je e jedinstveni luk stabla T koji je incidentan s v. Tada napajanje luka v zajedno s (1.2) jedinstveno određuje vrijednost toka $x(e)$. Isto obrazloženje se zatim može primijeniti na svaki list stabla $T \setminus e$, itd. Na taj način, rekurzivno smo definirali vrijednost toka $x(e)$ za sve lukove $e \in T$, istodobno zadovoljavajući sva ograničenja napajanja. □

Bilo koju particiju $L \dot{\cup} U$ od $E \setminus T$ nazivamo strukturom bazičnog toka. Tok definiran kao u Lemi 1.2.5 strukture (T, L, U) ne mora nužno biti dopustiv. Zato ćemo strukturu (T, L, U) zvati **optimalnom ili dopustivom** ako odgovarajući tok ima tražena svojstva.

Postavlja se pitanje: Kada ćemo znati da je tok optimalan? To nije trivijalno pitanje, te ćemo u tu svrhu iskazati i dokazati sljedeći teorem koji se često naziva *teorem slabe dualnosti*.

[4] Slobodan ciklus je ciklus sačinjen od slobodnih lukova.

Teorem 1.2.6. *Dopustivi bazičan tok strukture* (T, L, U) *za problem* (P) *je optimalan ako i samo ako postoji potencijal* $\pi\colon V \to \mathbb{R}$ *takav da reducirani troškovi*

$$c_{ij}^{\pi} := c_{ij} - \pi_i + \pi_j$$

zadovoljavaju:

$$\begin{aligned} &(\forall (i,j) \in T) \quad c_{ij}^{\pi} = 0 \\ &(\forall (i,j) \in L) \quad c_{ij}^{\pi} \geq 0 \\ &(\forall (i,j) \in U) \quad c_{ij}^{\pi} \leq 0 \end{aligned} \tag{1.3}$$

Dokaz. Neka je $y\colon E \to \mathbb{R}$ proizvoljan tok. Tada je:

$$\begin{aligned} c^{\pi}(y) &= \sum_{(u,v)\in E} y(u,v)(c(u,v) - \pi(u) + \pi(v)) \\ &= \sum_{(u,v)\in E} y(u,v)c(u,v) - \sum_{u\in V}\sum_{e^-=u} y(e)\pi(u) + \sum_{v\in V}\sum_{e^+=v} y(e)\pi(v) \\ &= c(y) + \sum_{v\in V} \pi(v)\left(\sum_{e^-=v} y(e) - \sum_{e^+=v} y(e)\right) \\ &= c(y) + \sum_{v\in V} \pi(v)b(v) \end{aligned}$$

U zadnjem koraku smo iskoristili definiciju napajanja. $c^{\pi}(y)$ i $c(y)$ se razlikuju samo za konstantni sumand koji ne ovisi o y. Stoga, dopustivi tok je optimalan za funkciju troškova c ako i samo ako je optimalan za c^{π}.

Neka je $x\colon E \to \mathbb{R}$ dopustivi tok definiran s dopustivom strukturom bazičnog toka (T, L, U) koja zadovoljava (1.3).
Po definiciji,

$$x(e) = l(e) \text{ za svaki } e \in L \text{ i } x(e) = u(e) \text{ za svaki } e \in U \tag{1.4}$$

Neka je y proizvoljan dopustiv tok. Iz (1.3) i (1.4),

$$\begin{aligned} c^{\pi}(y) &= \sum_{e\in T} y(e)c^{\pi}(e) + \sum_{e\in L} y(e)c^{\pi}(e) + \sum_{e\in U} y(e)c^{\pi}(e) \\ &= \sum_{e\in L} y(e)c^{\pi}(e) + \sum_{e\in U} y(e)c^{\pi}(e) \\ &\geq \sum_{e\in L} l(e)c^{\pi}(e) + \sum_{e\in U} u(e)c^{\pi}(e) \\ &= \sum_{e\in E} x(e)c^{\pi}(e) = c^{\pi}(x) \end{aligned}$$

x je optimalan za reducirane troškove c^{π}, te stoga i za c. □

Potencijal s uvjetima optimalnosti (1.3) ima svojstvo da je vrijednost reduciranih troškova 0 na svim lukovima razapinjućeg stabla T. Mrežna simpleks metoda koristi samo potencijale s tim svojstvom. Sada ćemo pokazati da bilo koje razapinjuće stablo određuje takav potencijal, koji je jedinstven do na vrijednosti u jednom vrhu.

Lema 1.2.7. *Neka je zadana struktura* (T, L, U) *za problem* (P)*, i neka je* v *bilo koji vrh. Tada postoji jedinstveni potencijal* π *koji zadovoljava:*

$$\pi(v) = 0 \text{ i } c^{\pi}(e) = 0 \text{ za svaki } e \in T \tag{1.5}$$

Dokaz. Eksplicitno, (1.5) zahtjeva:

$$c(u, v) - \pi(u) + \pi(v) = 0 \text{ za svaki luk } e = (u, v) \in T \tag{1.6}$$

Zbog toga što je $\pi(v) = 0$, vrijednost $\pi(u)$ je određena s (1.6) za svaki susjedni vrh u vrha v. U isto vrijeme, π je također određen susjednim vrhovima tih vrhova itd.. Kako T sadrži jedinstveni put od v do svakoga drugog vrha od G, uvjet (1.6) uistinu pridonosi jedinstvenom potencijalu π s $\pi(x) = 0$. □

1.3 Konstrukcija strukture dopustivog bazičnog toka

Prije nego što napišemo algoritam za mrežnu simpleks metodu, treba nam dopustiva struktura bazičnog toka za dani problem (P). Tu će nam pomoći odgovarajuća transformacija polaznog problema (P). No, prije toga slijedi napomena o numeriranju vrhova grafa, te o novom pomoćnom vrhu.

Napomena 1.3.1. *Početne vrhove grafa* $G = (V, E)$ *numerirat ćemo s* $1, 2, \ldots, n$*. Novi pomoćni vrh koji ćemo uvesti u daljnjem tekstu ćemo numerirati s* 0*.*

Za problem (P) na usmjerenom grafu $G = (V, E)$ pomoćni problem minimizacije troškova toka na mreži je dan Tablicom 1.2. Pomoćni problem (P') možemo zapisati na sljedeći način:

$$\begin{cases} \sum\limits_{e \in E'} c'(e)x'(e) \to \min \\ l'(e) \le x'(e) \le u'(e), \forall e \in E' \\ \sum\limits_{e^- = v} x'(e) - \sum\limits_{e^+ = v} x'(e) = b'(v), \forall v \in V' \end{cases} \tag{P'}$$

$V' = V \cup \{0\}$, pri čemu $0 \notin V$
$b'_0 = 0$; $b'_i = b_i$, $i \in V$
$E' = E \cup \{(0,i) : b_i + \sum_{e^+=i} l(e) - \sum_{e^-=i} l(e) < 0\} \cup \{(i,0) : b_i + \sum_{e^+=i} l(e) - \sum_{e^-=i} l(e) \geq 0\}$
$l(0,i) = 0$, $(0,i) \in E'$
$l(i,0) = 0$, $(i,0) \in E'$
$l'(e) = l(e)$, $e \in E$
$u'(0,i) = -b(i) - \sum_{e^+=i} l(e) + \sum_{e^-=i} l(e) + 1$, $(0,i) \in E'$
$u'(i,0) = b(i) + \sum_{e^+=i} l(e) - \sum_{e^-=i} l(e) + 1$, $(i,0) \in E'$
$u'(e) = u(e)$, $e \in E$
$c'(0,i) = M$, $(0,i) \in E'$; $c'(i,0) = M$, $(i,0) \in E'$, gdje je $M := 1 + \frac{1}{2}|V| \max\{|c(e)|: e \in E\}$
$c'(e) = c(e)$, $e \in E$;

Tablica 1.2: Pomoćni problem P'

Teorem 1.3.2. *Neka je dan pomoćni problem* (P′) *Tablicom 1.2 i problemom* (P). *Tada postoji optimalno rješenje x' za* (P′). *Štoviše, imamo jedan od dva moguća slučaja:*

- *Ako je $x'(0,i) > 0$ za neki $(0,i) \in E'$, tada ne postoji dopustivi tok za* (P).
- *Ako je $x'(0,i) = 0$ za sve $(0,i) \in E'$, tada je $x'|_E$ optimalni tok za* (P).

Dokaz. Dokaz pogledati u [1]. □

Kao što vidimo, pomoćni problem P' uvijek dopušta dopustivi tok i štoviše po Teoremu 1.3.2, također i dopustivu strukturu bazičnog toka.
Sada ćemo eksplicitno pokazati jednu takvu strukturu.

Lema 1.3.3. *Neka je zadan pomoćni problem* (P′). *Definirajmo:*

$$T = \{(0,i) : (0,i) \in E'\} \cup \{(i,0) : (i,0) \in E'\} \quad L = E,\ i\ U = \emptyset.$$

Tada je (T, L, U) struktura dopustivog bazičnog toka za (P′).

Dokaz. Kako je vrh 0 spojen sa svakim vrhom skupa V s točno jednim lukom, T je uistinu razapinjuće stablo za $G' = (V', E')$. Neka je x' tok zadan s (T, L, U) iz Leme 1.2.5. Stoga je $x'(e) = l(e)$ za svaki $e \in E$. Tada ograničenje napajanja jedinstveno određuje vrijednosti $x'(0,i)$ i $x'(i,0)$ i štoviše x' se slaže sa dopustivim tokom definiranim u prvom dijelu dokaza Teorema 1.3.2 (v. dokaz Teorema 11.3.1 iz [1]). Dakle, (T, L, U) je uistinu dopustiva struktura bazičnog toka za (P′). □

U ovoj točki rada napisali smo potrebnu teoriju za mrežnu simpleks metodu, te smo spremni napisati algoritam mrežne simpleks metode. Algoritam će rješavati problem (P) tako da će definirati pomoćni problem (P′) i njega rješavati. Ukoliko nađemo dopustivi optimalni tok za (P′), našli smo i za (P).

1.4 Algoritam

U ovom poglavlju dat ćemo opis generalne strukture mrežne simpleks metode, te ćemo biti više precizniji što se tiče prekida algoritma.

Algoritam 1.4.1 (Mrežne simpleks metode)**.** *Neka je problem* (P) *s usmjerenim grafom G kao iz Definicije 1.1.1. Neka je pridružen pomoćni problem* (P′) *zadan Tablicom 1.2.*

1. ***Inicijalizacija.*** *Definirajmo:*
   - $T = E' \setminus E; L = E; U = \emptyset$
   - $x(e) = l(e), e \in E$
   - $x(e) = u'(e) - 1, e \in E' \setminus E$
   - $\pi(0) = 0$
   - $\pi(i) = -M, i \in V$ *pri čemu je* $(0, i) \in E'$
   - $\pi(i) = M, i \in V$ *pri čemu je* $(i, 0) \in E'$
2. ***Test optimalnosti.*** *Ukoliko ne postoji* $e \in L \cup U$ *takav da vrijedi:*

   $$e \in L \; i \; c^{\pi}(e) < 0$$

   $$e \in U \; i \; c^{\pi}(e) > 0$$

   Stop. Ako je $x(e) > 0$ *za neki* $e \in E' \setminus E$*, tada ne postoji dopustivi tok originalne zadaće. Inače,* $x|_E$ *je optimalan bazičan tok originalne zadaće.*
3. ***Određivanje ciklusa.*** *Izaberi* $e \in L \cup U$ *koji narušava uvjet optimalnosti i odredi jedinstveni ciklus* γ *u* $T \cup \{e\}$*. Označimo taj luk sa s;* $s = e$*.*
4. ***Određivanje luka koji napušta stablo.*** *Prvo utvrdimo orijentaciju ciklusa:*
   - *Ako je dodan luk* $s \in L$ *orijentacija ciklusa je u skladu s orijentacijom novog luka,*
   - *Ako je dodan luk* $s \in U$ *orijentacija ciklusa je suprotna orijentaciji novog luka.*

U ciklusu γ s $\overline{\gamma}$ označimo direktne lukove, a s $\underline{\gamma}$ obrnute lukove obzirom na orijentaciju ciklusa.
Za lukove iz ciklusa $\gamma = \overline{\gamma} \dot{\cup} \underline{\gamma}$ definiramo:

$$\delta(e) = \begin{cases} x(e) - l(e) & e \in \underline{\gamma} \\ u(e) - x(e) & e \in \overline{\gamma} \end{cases} \tag{1.7}$$

U (1.7) *dana je maksimalna vrijednost toka koju možemo dodati na direktnim lukovima, odnosno oduzeti na obrnutim lukovima, tako da novi tok bude opet dopustiv. Definirajmo novi tok:*

$$x'(e) = \begin{cases} x(e) - \delta & e \in \underline{\gamma} \\ x(e) + \delta & e \in \overline{\gamma} \end{cases} \tag{1.8}$$

Gdje je $\delta = \min_{e \in \gamma} \gamma(e) > 0$. Na barem jednom luku iz γ dobivamo jednakost:

$$x'(e) = l(e) \tag{1.9}$$

ili

$$x'(e) = u(e) \tag{1.10}$$

Izaberimo taj luk i označimo ga s o; $o = e$.

5. ***Ažuriranje**. Ažuriraj tok i strukturu (T, L, U) bazičnog toka.*

   - $x(e) = x'(e)$, $e \in \gamma$
   - $T = (T \setminus \{o\}) \cup \{s\}$

   Izračunaj potencijal π prateći strukturu (T, L, U) i uzimajući $\pi(0) = 0$, kako je navedeno u lemi 1.2.7. Idi na korak 2.

Sada ćemo razmotriti korake iteracije koji se odvijaju u algoritmu. Nazovimo luk koji smo označili u 3. koraku *ulazni luk*, jedinstveni ciklus $\gamma \subset T \cup \{s\}$ *pivotni ciklus* i luk o iz 4. koraka *izlazni luk*.

Ovakva gore napisana generička verzija algoritma ne mora nužno stati u konačno mnogo koraka, tj. završiti. Razlog tome je mogućnost pojave degenerirane strukture bazičnog toka: struktura bazičnog toka (T, L, U) gdje T ne sadrži samo slobodne lukove. U tome slučaju, odgovarajuće povećanje uz ciklus iz 3. koraka može biti nemoguće: možemo imati $\delta = 0$ u 4. koraku. Iako ćemo promijeniti stablo T (kako je u ovom slučaju ulazni luk različit od izlaznog luka), možda ćemo nakon niza promjena doći ponovno do iste strukture bazičnog toka (T, L, U). Drugim riječima, moguće je da algoritam završi u beskonačnoj

petlji. To je realan problem budući da u praktičnim primjerima imamo pojavu degeneracije strukture u 90% od svih struktura bazičnih tokova. Međutim, kruženje se može izbjeći tako da se na odgovarajući način izabere izlazni luk. Na sreću, dodatni napor, koji je potreban da se to učini, čak teži da se algoritam ubrza na način da se smanji potreban broj iteracija.

Sada ćemo detaljno objasniti kako ćemo izbjeći da algoritam ne završi u beskonačnoj petlji. U tu svrhu ćemo odabrati jedan fiksni vrh w koji ćemo koristiti kao korijen za sva razapinjuća stabla konstruirana tijekom algoritma.

Definicija 1.4.2. *Dopustiva struktura bazičnog toka* (T, L, U) *je* ***jako dopustiva*** *ako za svaki vrh* $i \neq w$*, jedinstveni put od* i *do* w *u stablu* T *je zapravo put povećanja bazičnog toka* x *strukture* (T, L, U)*. Posebno, dopustiva struktura stabla* (T, L, U) *je jako dopustiva ako je nedegenerirana, tj. ako su svi lukovi stabla* T *slobodni.*

U fazi inicijalizacije (1. korak) biramo $w = 0$. Tada je inicijalna struktura $(T, L, U) = (E' \setminus E, E, \emptyset)$ jako dopustiva za (P′).
Po Lemi (1.3.3) je dopustiva i imamo:

$$x(0, i) = u'(0, i) - 1 > 0, (0, i) \in E'$$

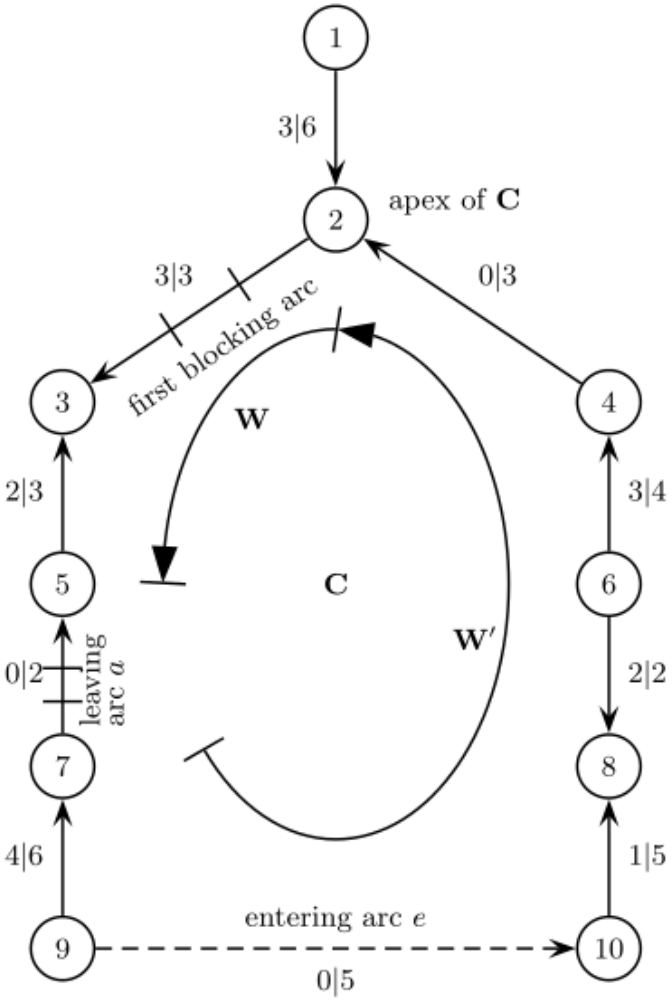
$$Y(i, 0) = u'(i, 0) - 1 < u'(i, 0), (i, 0) \in E'$$

Te je stablo T nedegenerirano.

Sada razmotrimo iteracije koje se odvijaju tijekom algoritma. Neka je s ulazni luk izabran u 3. koraku i neka je $\gamma \subset T \cup \{s\}$ pivotni ciklus. Odredit ćemo orijentaciju ciklusa γ kako je definirano u algoritmu. Luk iz $\gamma \setminus \{s\}$ ćemo nazvati *blokirajući luk* ako nakon izračuna novog toka (1.8) vrijedi jednakost (1.9) ili (1.10). Jedinstveni vrh iz γ koji je najbliži korijenu w stabla T ćemo nazvati početni vrh ciklusa γ. Pokazat ćemo da iduće pravilo za izlazni luk čuva jaku dopustivost za sve strukture bazičnog toka koje se pojavljuju tijekom algoritma.
Pravilo zadnjeg blokirajućeg luka: *Počevši od početnog vrha pivotnog ciklusa* γ *i prolazeći kroz* γ *prateći orijentaciju, biramo zadnji blokirajući luk koji se našao na putu kao izlazni luk.*
Na Slici 1.2 je prikazana ilustracija navedenoga pravila zadnjeg blokirajućeg luka. Pritom smo koristili sljedeću terminologiju:

- W: put od početnog vrha ciklusa γ do zadnjeg blokirajućeg luka, prateći orijentaciju pivotnog ciklusa γ. Na Slici 1.2 početni vrh pivotnog ciklusa je označen s 2 i W = 2–3–5.

- W': put od početnog vrha do zadnjeg blokirajućeg luka prateći suprotnu orijentaciju γ. Na Slici 1.2 W = 2–4–6–8–10–9–7.



Slika 1.2: Pravilo zadnjeg blokirajućeg luka gdje je $s = e$ i $o = a$.

- Lukovi imaju oznaku u formi $x|u$ i označava trenutnu vrijednost toka i gornji kapacitet promatranoga luka. Vrijednost donjeg kapaciteta je 0 za sve lukove.

Teorem 1.4.3. *Neka je (T, L, U) jako dopustiva struktura bazičnog toka, i neka je $s \notin T$. Ako je izlazni luk o izabran u 4. koraku algoritma 1.4.1 primjenjujući pravilo zadnjeg blokirajućeg luka, tada je ažurirana struktura stabla iz 5. koraka i dalje jako dopustiva.*

Dokaz. Neka je x bazičan tok strukture (T, L, U), i neka je x' dopustivi tok nastao kao rezultat 4. koraka algoritma. Ostaje nam pokazati da je u dobivenom razapinjućem stablu

$T' = (T \cup \{s\}) \setminus \{o\}$ put koji vodi od i do w put povećanja toka x' (za svaki vrh $i \neq w$). Razlikujemo 4 slučaja.

Slučaj 1. i je početni vrh od C. Tada se put od i do w u stablu T' i T slažu, i vrijednost toka neće biti promijenjena na lukovima toga zajedničkog puta. Označimo taj put s Q. Kako je (T, L, U) jako dopustiv, Q je put povećanja toka x, a i time i toka x'.

Slučaj 2. i je na W'. Kako je o zadnji blokirajući luk koji se našao na putu ciklusa γ od početnog vrha prateći orijentaciju, ni jedan luk u W' ne može biti blokirajući. Stoga, put od i do početnog vrha ciklusa γ u T' je put povećanja toka x'. Iz 1. slučaja slijedi da je put od i do w također put povećanja toka x'.

Slučaj 3. i je na W. Neka je $\delta \geq 0$ iz 4. koraka algoritma. U slučaju $\delta > 0$ vrijednost toka x je na lukovima puta W povećana za δ. Stoga put suprotan od W (koji završava u početnom vrhu od γ) je put povećanja toka x'. Iz 1. slučaja, put od i do w u T' je također put povećanja toka x'. Sada pretpostavimo da je $\delta = 0$. Kako o nije u W, put od i do w u T i T' je isti i vrijednost toka se nije promijenila na tim lukovima zajedničkog puta Q (zbog toga što je $\delta = 0$).Kako je (T, L, U) jako dopustiv, Q je put povećanja toka x, a time i toka x'.

Slučaj 4. i nije u γ. Kako je (T, L, U) jako dopustiv, put u oznaci Z od i do w u T je put povećanja toka x. Prvo pretpostavimo da su Z i Q disjunktni. Tada je Z također put u T' koji je put povećanja toka x', jer se vrijednost toka nije promijenila na Z. Zatim, neka je v prvi vrh u Z koji također pripada γ-u. Tada je put od v do w u T' put povećanja toka x' (kao prethodno). Iz prethodna tri slučaja isti zaključak vrijedi za put od v do w u T'. Stoga je put od i do w u T' također put povećanja toka x'. □

Treba nam još jedan pomoćni rezultat da dokažemo da algoritam za mrežnu simpleks metodu završava u konačno mnogo koraka kada se koristi pravilo zadnjeg blokirajućeg luka.

Lema 1.4.4. *Neka je (T, L, U) struktura bazičnog toka koja se javlja u izvođenju algoritma 1.4.1, i neka je π pridružen potencijal. Osim toga, neka je o izlani luk i neka je $s = (i, j)$ ulazni luk. Označimo novi potencijal (nakon ažuriranja u 5. koraku algoritma) s π'. Na kraju, neka je T_1 povezana komponenta od $T \setminus \{o\}$ koja sadrži w i $T_2 = V \setminus T_1$. Tada je:*

$$\pi'(v) = \begin{cases} \pi(v) & \text{ako je } v \in T_1 \\ \pi(v) - c^{\pi}(s) & \text{ako je } v \in T_2 \text{ i } i \in T_1 \\ \pi(v) + c^{\pi}(s) & \text{ako je } v \in T_2 \text{ i } i \in T_2 \end{cases}$$

Dokaz. Zapišimo $T' = (T \cup \{s\}) \setminus \{o\}$ i

$$c(u, v) - \pi'(u) + \pi'(v) = 0 \text{ za sve } (u, v) \in T'. \tag{1.11}$$

Neka je v bilo koji vrh iz T_1. Tada je jedinstveni put od v do w u $T1'$ isti kao i u stablu T. Zbog $\pi(w) = \pi'(w) = 0$ slijedi da je $\pi'(v) = \pi(v)$, $v \in T_1$.

Sada pretpostavimo da je $i \in T_1$ i $j \in T_2$. Tada je $\pi'(i) = \pi(i)$, te $\pi'(j) = \pi(i) - c(i,j) = \pi(j) - c^\pi(i,j)$ po (1.11). Put od j do v u T' i T se preklapaju za svaki vrh $v \in T_2$. Prema tome, $\pi'(v) = \pi(v) - c^\pi(i,j)$ za svaki $v \in T_2$.

Na kraju, pretpostavimo da je $i \in T_2$ i $j \in T_1$. Tada je $\pi'(j) = \pi(j)$ i prema tome $\pi'(i) = -c(i,j) + \pi'(j) = c^\pi(i,j) + \pi(i)$ po (1.11). Put od i do v u T' i T se preklapaju za svaki vrh $v \in T_2$. Prema tome, $\pi'(v) = \pi(v) + c^\pi(i,j)$ za svaki $v \in T_2$. □

U idućem teoremu pretpostavljamo da su svi podaci u danom problemu (P) racionalni. Naravno, to nije stvarno ograničenje s praktične točke gledišta, no jedino racionalne brojeve možemo reprezentirati u računalu.

Teorem 1.4.5. *Neka je dan problem* (P) *s usmjerenim grafom G kao iz Definicije 1.1.1 i pretpostavimo da su svi podaci l, u, b, c racionalni. Tada algoritam za mrežnu simpleks metodu 1.4.1 staje u konačno mnogo koraka pod uvjetom da se koristi pravilo zadnjeg blokirajućeg luka za biranje izlaznog luka.*

Dokaz. Dokaz pogledati u [1]. □

1.5 Efikasna implementacija

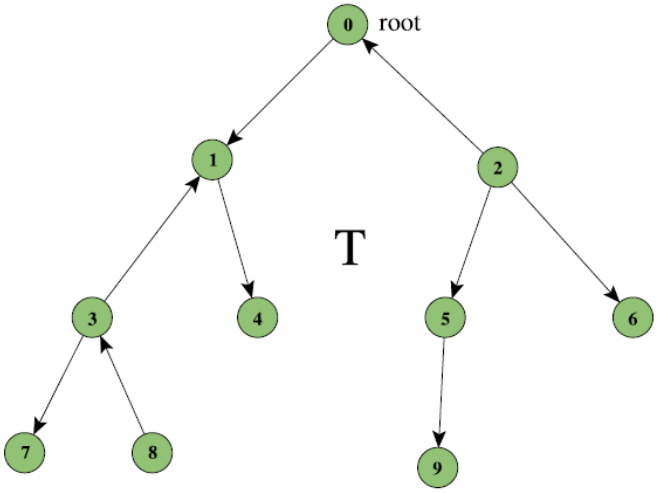
Pri izvođenju Algoritma 1.4.1 moramo odrediti pivotni ciklus γ, povećati tok na γ i ažurirati potencijal. Kako bi te operacije obavljali što efikasnije trebaju nam informacije o razapinjućem stablu. Jedna mogućnost, koju ćemo mi primijeniti, za efikasnu implementaciju tih operacija je korištenje indeksa stabla kojeg ćemo sada uvesti.

Prisjetimo se da smo izabrali fiksni vrh w koji nam služi kao korijen za sva razapinjuća stabla koja se javljaju prilikom izvođenja algoritma. U našem slučaju smo izabrali vrh 0. Neka je T jedno takvo razapinjuće stablo. Definiramo indekse stabla za T:

- **indeks prethodnik**: Za svaki vrh $v \neq w$, $p(v)$ je prethodnik od v na putu od w do v u T.
- **indeks dubine**: Za svaki vrh v, $d(v)$ udaljenost između v i w, tj. $d(v)$ je broj bridova na putu od w do v u T. Posebno, $d(w) = 0$
- **indeks obilaska**: Neka je $w, v_1, v_2, \ldots, v_n$ DFS[5] pre-order obilazak stabla T s početkom u vrhu w. Stavimo:

  $$th(w) = v_1, th(v_n) = w \text{ i } th(v_i) = v_{i+1} \text{ za svaki } i = 1, 2, \ldots, n-1.$$

  Indeks obilaska se koristi za opisivanje mogućeg DFS pre-order obilaska stabla od vrha. Naravno, indeksi nisu generalno jedinstveno određeni s T.



v	0	1	2	3	4	5	6	7	8	9
$p(v)$	–	0	0	1	1	2	2	3	3	5
$d(v)$	0	1	1	2	2	2	2	3	3	3
$th(v)$	1	3	5	7	2	9	0	8	4	6

Slika 1.3: Indeksno stablo

Na Slici 1.3 smo prikazali razapinjuće stablo T s korijenom $w = 0$. Ispod stabla su prikazane vrijednosti indeksa stabla $p(v)$, $d(v)$ i $th(v)$.

Kada izaberemo ulazni luk s za T, dobivamo pivotni ciklus γ. S naivnom implementacijom, pronalazak γ će zahtijevati pretraživanje cijelog stabla T stoga mu je složenost $O(|V|)$. Korištenjem indeksa prethodnika i indeksa dubine složenost se smanjuje na $O(|\gamma|)$. U praksi se pokazalo da je faktor ubrzanja $n/\log n$. Tako da možemo očekivati da će pronalazak γ biti 1000 puta brže za $n = 10000$ (v. [3]). Slično, ubrzanje ćemo postići za ažuriranje potencijala. No, o tome malo kasnije.

Sljedeća procedura će odrediti početni vrh ciklusa γ, vrijednost δ i također izlazni luk o primjenjujući spomenuto pravilo zadnjeg blokirajućeg luka.

Algoritam 1.5.1. *Neka je (T, L, U) trenutna jako dopustiva struktura bazičnog toka, s indeksima stabla p, d, th, ulaznim lukom $s = (i, j)$ i pivotnim ciklusom γ. Označimo s apex početni vrh pivotnog ciklusa γ. Za $v \neq w$, označimo luk iz T pridružen vrhovima v i $p(v)$*

[5]DFS = depth first search

sa $s(v)$ i neka je s $r(v)$ označena količina toka koja se još može poslati kroz $s(v)$ prateći orijentaciju ciklusa γ.

```
1:  procedure PIVOTCYCLE(G, l, u, b, c, T, L, U, s; apex, o, δ)
2:      δ ← ∞;
3:      if s ∈ L then
4:          u ← i; v ← j
5:      else
6:          u ← j; v ← i
7:      end if
8:      while u ≠ v do
9:          if d(u) > d(v) then
10:             if r(u) < δ then
11:                 δ ← r(u); o ← s(i)
12:             end if
13:             u ← p(u)
14:         else
15:             if r(v) ≤ δ then
16:                 δ ← r(v); o ← s(v)
17:             end if
18:             v ← p(v)
19:         end if
20:     end while
21:     apex ← u
22: end procedure
```

Primijetimo da su u i v definirani na takav način da u stigne prije v kada se putuje u skladu orijentacije. Vrhovi u i v putuju tijekom procedure kroz dva disjunktna podskupa od T koji se susreću u početnom vrhu od γ. Štoviše, doseći ćemo početni vrh od γ onog trena kada je $u = v$. Tada procedura staje i u je početni vrh ciklusa γ, luk o je zadnji blokirajući luk, i δ je maksimalna vrijednost toka koju možemo dodati odnosno oduzeti na lukovima ciklusa γ.

Kako bi povećali trenutni bazični tok x strukture (T, L, U) potrebno je ponovno proći kroz γ. Novi prolaz ćemo iskoristiti kako bi ažurirali potencijal. Možemo odlučiti, ako je polazni vrh i ulaznog luka $s = (i, j)$, leži li vrh u podstablu T_1 definiran u lemi 1.4.4 pomoću varijable *subtree*.
Slijedi procedura koja će proći kroz γ i povećati trenutni bazični tok x. Pri tome će odrediti vrijednost varijable *subtree*.

Algoritam 1.5.2. *Neka je (T, L, U) trenutna jako dopustiva struktura stabla, s indeksima stabla p, d, th, ulaznim lukom $s = (i, j)$ i pivotnim ciklusom γ. Povrh svega, neka je početni*

vrh od γ, vrijednost δ i izlani luk o izračunat pomoću Algoritma 1.5.1.

```
1: procedure AUGMENT(G, l, u, b, c, T, L, U, s, apex, o, δ; subtree)
2:     u ← i; v ← j;
3:     while u ≠ apex do
4:         izračunaj vrijednost toka na s(u) s δ (u smjeru γ);
5:         if s(u) = o then
6:             subtree ← T_2
7:         end if
8:         u ← p(u)
9:     end while
10:    while v ≠ apex do
11:        izračunaj vrijednost toka na e(v) s δ (u smjeru γ);
12:        if s(v) = o then
13:            subtree ← T_1
14:        end if
15:        v ← p(v)
16:    end while
17: end procedure
```

Naravno, kada kažemo povećanje vrijednosti toka na pivotnom ciklusu γ, onda mislimo oduzeti ili dodati vrijednost δ na svaki luk pivotnog ciklusa, ovisno o orijentaciji luka u γ.

Kada uklonimo luk $o = (u, v)$ iz T, stablo T se dijeli na dvije povezane komponente, T_1 i T_2. Po Lemi 1.4.4 trenutni potencijal π se ne mijenja na T_1, kada pređemo u novu strukturu stabla dodavanjem novog ulaznog luka. Na T_2 novi potencijal se razlikuje od π samo za konstantu. Primijetimo da je u u T_1 ako i samo ako je $d(u) < d(v)$. Po Lemi 1.4.4 ažuriramo potencijal na definirani način. Slijedi procedura koja ažurira potencijal.

Algoritam 1.5.3. *Neka je (T, L, U) trenutno jako dopustiva struktura bazičnog toka, s odgovarajućim potencijalom π i indeksima stabla p,d, th. Neka je ulazni luk $s = (i, j)$ i izlazni luk $o = (u, v)$. Povrh svega, neka je vrijednost subtree izračunata s Algoritmom 1.5.2.*

```
1: procedure PIUPDATE(G, l, u, b, c, T, L, U, s, subtree; π)
2:     if subtree = T_1 then
3:         y ← v
4:     else
5:         y ← u
6:     end if
7:     if d(u) < d(v) then
8:         ϵ ← −c^π(s)
9:     else
10:        ϵ ← c^π(s)
```

11: ***end if***
12: $\pi(y) \leftarrow \pi(y) + \epsilon$, $z \leftarrow th(y)$
13: ***while*** $d(z) > d(y)$ ***do***
14: $\pi(z) \leftarrow \pi(z) + \epsilon$, $z \leftarrow th(y)$
15: ***end while***
16: ***end procedure***

Na kraju, ostaje nam opisati kako efikasno ažurirati indekse stabla. No, to je više tehničko pitanje, tako da nećemo o tome raspravljati u ovom diplomskom radu.

Također smo izostavili jedan veliki korak u mrežnoj simpleks metodi, a to je izbor ulaznog luka. Izabrana strategija za tu operaciju igra odlučivu ulogu u performansi algoritma. Uobičajeno se koristi heuristika. Mi ulazni luk biramo po principu "tko prvi djevojci, njegova djevojka." Naime, prvi luk koji narušava uvjete optimalnosti biva izabran kao ulazni luk.

Sada se okrećemo implementaciji samog algoritma u programskom jeziku Python.

Poglavlje 2

PyQt5 aplikacija za mrežnu simpleks metodu

2.1 Python, Qt5, PyQt5 i biblioteka NetworkX

Python je programski jezik opće namjene, interpretiran i visoke razine kojeg je stvorio Guido van Rossum 1990. godine. Python je ime dobio po televizijskoj seriji Monty Python's Flying Circus. Po automatskoj memorijskoj alokaciji, Python je sličan programskim jezicima kao što su Perl, Ruby, Smalltalk itd. Python je programski jezik koji postaje sve popularniji, a glavna prednost je dopuštanje programerima korištenje nekoliko stilova programiranja. Dopušteni stilovi su objektno orijentirano, strukturno i aspektno orijentirano programiranje. Python se najviše koristi na Linuxu, no postoje i inačice za druge operacijske sustave.

Jedna od mana Pythona je njegova sporost. Pošto je Python interpreterski jezik, programi napisani u njemu izvršavaju se malo sporije za usporedbu od kompajlerskih jezika, kao što su C, C++ i slični. Međutim, usprkos njegovoj sporosti on se poprilično koristi, ponajviše kao pozadinski programski jezik.

Python sadrži dosta implementiranih programskih modula i biblioteka za razna područja primjene. Neke od standardnih Python biblioteka su Numpy, Scipy, Matplotlib, Pandas, Sympy, Sage i mnoge druge. Sa stajališta Pythona kao znastvenog alata, skoro za svaki problem postoji implementirano rješenje u Pythonu. Tako i za naš promatrani problem (P) postoji biblioteka koja ga rješava. Konkretno radi se o biblioteci NetworkX.

Programska biblioteka NetworkX se koristi za kreiranje, manipuliranje i proučavanje struktura, funkcionalnosti i dinamičnosti složene mreže. Više o NetworkX biblioteci se može naći na [5].
Neke značajke NetworkX-a su:

- Python programska biblioteka za grafove, usmjerene grafove i multigrafove.

- Sadrži mnogo standardnih algoritama za grafove.
- Mrežna struktura i analize mjere.
- Generatori za klasične grafove, slučajne grafove i sintetičke mreže.
- Čvorovi mogu biti bilo što (npr. tekst, slika, XML zapis itd.).
- Lukovi mogu sadržavati proizvoljne podatke (npr. težine, kapacitete itd.)

Sada ćemo proći primjerom i pokazati kako riješiti jedan proizvoljan problem (P) u Pythonu. Prije samog korištenja potrebno je uključiti biblioteku u naš program.

```
import networkx as nx
```

Sljedećom naredbom ćemo stvoriti prazni usmjereni graf.

```
G=nx.DiGraph()
```

Svaki graf se sastoji od čvorova i lukova. Čvorovi grafa G mogu sadržavati neke podatke. Naš graf iz Definicije 1.1.1 za svaki čvor pamti napajanje. NetworkX nam daje mogućnost da mu eksplicitno napišemo napajanje. Napajanje je zadano s ključnom riječi *demand*.

Napomena 2.1.1. *NetworkX je definirao napajanje suprotno od naše definicije napajanja. Naime, čvor s negativnim napajanjem je čvor ponude, a čvor s nenegativnim napajanjem je čvor potražnje. Samo u idućem kodu primjenjujemo definiciju od NetworkX-a.*

```
G.add_node(4, demand= 7)
G.add_node(1, demand= -5)
G.add_node(2, demand= -1)
G.add_node(3, demand = -1)
```

Za svaki luk su nam bitni donji kapacitet, gornji kapacitet i jedinična cijena transporta robe na tom luku. U NetworkX-u zadajemo te podatke na sljedeći način: *capacity* nam predstavlja gornji kapacitet i *weight* jediničnu cijenu transporta na tom promatranom luku. Jedina mana je što NetworkX pretpostavlja da je donji kapacitet za svaki luk u grafu 0. To ne znači da su neki problemi (P) nerješivi u NetworkX-u, nego da je potrebno transformirati promatrani problem (P) u njegov ekvivalentan problem u kojem su mu donji kapaciteti jednaki 0. Ukoliko *capacity* izostavimo, tada se podrazumijeva da je gornji kapacitet neograničen.

```
# add arcs to the graph: fromNode, toNode, capacity, cost (=weight)
G.add_edge(1, 2, capacity = 6, weight = 1 )
G.add_edge(1, 3, capacity = 3, weight = 4 )
G.add_edge(2, 4, capacity = 4, weight = 2 )
G.add_edge(3, 2, capacity = 7, weight = 1 )
G.add_edge(3, 4, capacity = 4, weight = 6 )
```

Graf G sadrži sve potrebne informacije za rješavanje problema (P). Sljedeći dio koda rješava (P).

```
flowCost, flowDict = nx.network_simplex(G)
```

Ukoliko tok nije dopustiv ili ne postoji optimalni tok, funkcija vraća odgovarajuću poruku. Varijabla *flowCost* je cijena minimalnog transporta koja zadovoljava sva ograničenja koja su navedena, dok *flowDict* rječnik s ključem po čvorovima tako da je *flowDict[u][v]* tok na luku (u, v).

Postavlja se pitanje što je onda PyQt5 i Qt5 i čemu to? Slijedi vrlo jednostavan i kratak odgovor. Qt je skup C++ biblioteka za razne platforme koje implementiraju API na visokoj razini pružajući mnoge mogućnosti stolnim i mobilnim sustavima. To uključuje servise za pronalaženje lokacije, multimedije, NFC i Bluetooth povezivanje, kao i tradicionalni razvoj UI. Možemo reći da su Qt biblioteke jedne od najmoćnijih GUI biblioteka.

PyQt5 je sveobuhvatan skup Python poveznica za Qt5. Dostupan je za Python 2.x i 3.x. Implementiran je kao skup Python modula. Sadrži više od 620 klasa i 6000 funkcija i metoda. Omogućava da Python bude korišten kao alternativni aplikacijski razvojni jezik C++ na svim podržanim platformama, uključujući iOS i Android. PyQt5 također može biti ugrađen u aplikacije razvijene u C++ kako bi omogućio korisnicima tih aplikacija konfiguriranje ili kako bi povećao funkcionalnost same aplikacije.
Navedimo neke module PyQt5: QtCore, QtGui, QtWidgets, QtMultimedia, QtBluetooth, QtNetwork, QtPositioning, Enginio, QtWebSockets, QtWebKit, QtWebKitWidgets, QtXml, QtSvg, QtSql, QtTest.

Sada slijedi prikaz implementiranih funkcija koje smo naveli u prethodnom poglavlju. Prvo smo napisali proceduru *PIVOTCYCLE* 1.5.1 za pronalazak početnog vrha pivotnog ciklusa γ, te vrijednost δ i izlazni luk o.

```
def pivot_cycle(self, r, s):
 tmp_u = {}
 self.delta = -1

 for e1, e2, c in self._G.edges(data='capacity'):
  tmp_u[(e1, e2)] = c['capacity']

 E = self._G.edges()

 if (r, s) in self.L:
  i = r
  j = s
 else:
  i = s
  j = r

 while i != j:
  if self.d[i] > self.d[j]:
```

```
   z = self.p[i]
  if (z, i) in E:
   tmp_delta = tmp_u[(z, i)] - self.x[(z, i)]
   if tmp_delta < self.delta or self.delta == -1:
    self.delta = tmp_delta
    l = (z, i)
   else:
    tmp_delta = self.x[(i, z)] - self._l[(i,z)]
    if tmp_delta < self.delta or self.delta == -1:
     self.delta = tmp_delta
     l = (i, z)
   i = z
  else:
   z = self.p[j]
   if (j, z) in E:
    tmp_delta = tmp_u[(j, z)] - self.x[(j, z)]
    if tmp_delta <= self.delta or self.delta == -1:
     self.delta = tmp_delta
     l = (j, z)
   else:
    tmp_delta = self.x[(z, j)] - self._l[(z, j)]
    if tmp_delta <= self.delta or self.delta == -1:
     self.delta = tmp_delta
     l = (z, j)
   j = z
 self.apex = i
 self.l = l
```

Izlazni luk je označen s *self.l*, delta *self.delta* i sa *self.apex* je označen početni vrh ciklusa γ.
Sada slijedi implementacija procedure *AUGUMENT* 1.5.2.

```
def augment(self, r, s):

        tmpT = nx.Graph(self.T)
        tmpT.remove_edge(self.a[0], self.a[1])
        self.T1 = nx.node_connected_component(tmpT, self.w)
        self.T2 = [e for e in self._G.nodes() if e not in self.T1]

        E = self._G.edges()
        if (r, s) in self.L:
                i = r
                j = s
        self.x[(i, j)] += self.delta
        else:
                self.x[(r, s)] -= self.delta
                i = s
                j = r
```

```
                ok = False

        while i != self.apex:
                z = self.p[i]
                if (z, i) in E:
                        self.x[(z, i)] += self.delta
                        if self.a == (z, i):
                                self.subtree = self.T2
                else:
                        self.x[(i, z)] -= self.delta
                        if self.a == (i, z):
                                self.subtree = self.T2
                i = z

        while j != self.apex:
                z = self.p[j]
                if (j, z) in E:
                        self.x[(j, z)] += self.delta
                        if self.a == (j, z):
                                self.subtree = self.T1
                else:
                        self.x[(z, j)] -= self.delta
                        if self.a == (z, j):
                                self.subtree = self.T1
                j = z

         if ok == False:

                if self.subtree == self.T1:
                         self.subtree = self.T2
                else:
                         self.subtree = self.T1
```

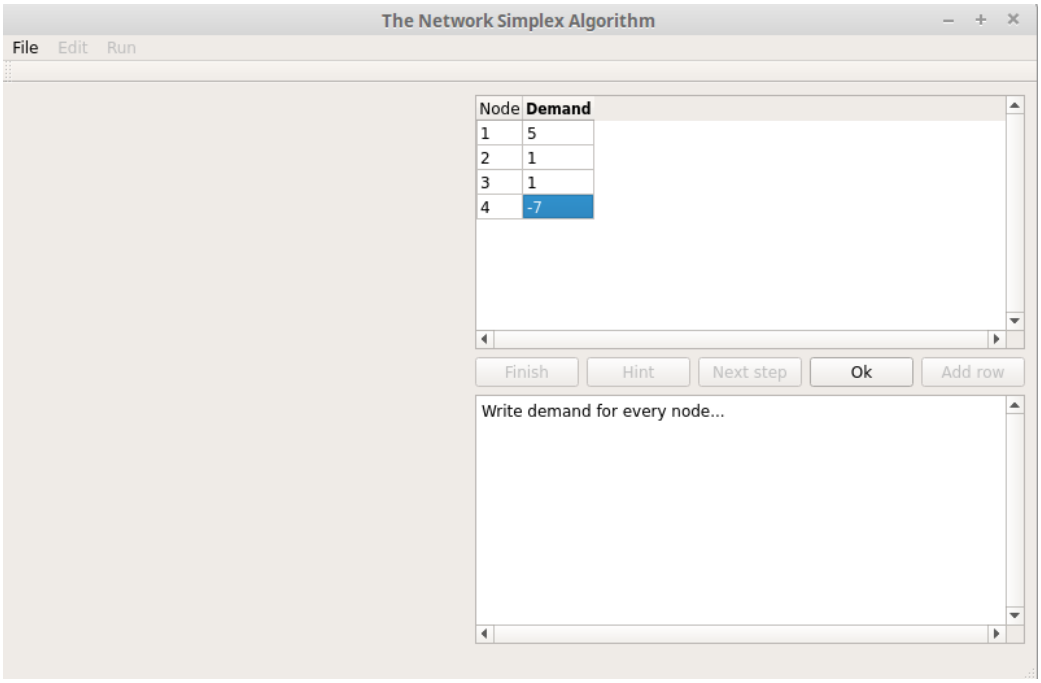
Na kraju slijedi implementacija *PIUPDATE* 1.5.3 koja ažurira vrijednost π na definirani način u lemi 1.4.4.

```
def pi_update(self, s):
        if self.subtree == self.T1:
                y = self.l[1]
        else:
                y = self.l[0]

        if self.d[self.l[0]] < self.d[self.l[1]]:
                eps = -self.redCost[s]
        else:
                eps = self.redCost[s]

        self.Pi[y] = self.Pi[y] + eps
```

```
z = self.th[y]

while self.d[z] > self.d[y] and z in self.T2:
        self.Pi[z] = self.Pi[z] + eps
        z = self.th[z]
```

2.2 Aplikacija

Opisani algoritam je implementiran u programskom jeziku Python, koristeći PyQt5, framework Qt5 i biblioteku Networkx. Aplikacija nudi rješavanje problema (P) korak po korak u interakciji s korisnikom aplikacije. Na kraju prethodnog koraka smo naglasili da se luk, koji narušava uvjete optimalnosti, bira po pravilu "tko prvi djevojci, njegova djevojka". No, tu korisnik igra ključnu ulogu. Korisnik može kliknuti na gumb *Next step* i tada će se izabrati ulazni luk po spomenutom pravilu ili može izabrati ulazni luk, između lukova koji narušavaju uvjete optimalnosti, tako da klikne gumb *Add*. Također korisnik može zaobići pravilo po kojem se bira izlazni luk. Korisnik može birati koji će luk napustiti stablo T.
Sada ćemo riješiti problem sa Slike 1.1 i tom prilikom demonstrirati rad aplikacije. Umjesto strelicom, završetak luka simboliziran je zadebljanjem linije.
Nakon pokretanja aplikacije, omogućava se unos grafa čitanjem iz datoteke ili unosom ručno. Unosom ručno korisnik prvo unese broj čvorova, te zatim napajanje. Kako to izgleda u aplikaciji može se vidjeti na Slici 2.1 koja prikazuje unos napajanja. Nakon što korisnik klikne na gumb *Ok*, otvara mu se tablica s lukovima u kojem može dodati proizvoljan broj lukova s donjim i gornjim kapacitetom i cijenom. Nakon što klikne gumb *Add row* stvara mu se novi redak u tablici (Slika 2.2). Nakon što korisnik klikne na gumb *Ok*, završava unos lukova. Zatim se u sučelju aplikacije crta graf kojeg je zadao (Slika 2.3). Ukoliko korisnik želi završiti s radom i dobiti krajnji rezultat, tada klikne na gumb *Finish*. Korisnik može završiti s radom u bilo kojem trenutku. To nam trenutno nije toliko zanimljivo. Pretpostavimo da korisnik želi korak po korak izvoditi algoritam. Tada klikne na gumb *Next step*. Nakon klika, inicijalizira se pomoćni problem koji ćemo rješavati: dodaje se pomoćni vrh 0 i spaja se lukovima s ostalim vrhovima na način da je početni bazični tok prirodno definiran. Program sam određuju početnu bazičnu strukturu (T, L, U) toka x'. Lukovi stabla T su označeni s crvenom bojom na Slici 2.4. Vezani lukovi iz U s plavom bojom, a iz lukovi L s cijan bojom. U nastavku, nakon inicijalizacije problema (P′), koji uključuje i definiranje početne strukture bazičnog rješenja (T, L, U), korisnik u interakciji s aplikacijom rješava problem (P′). Prvo slijedi pronalazak luka koji narušava uvjete optimalnosti, te ga ubacujemo u generirajuće stablo T. Aplikacija u pozadini odradi provjeru koji lukovi narušavaju uvjete optimalnosti, te uz odgovarajući luk stavi gumb *Add*. Pretpostavimo da korisnik želi ubaciti luk $(3, 4)$ u stablo T. Tada treba kliknuti na odgo-

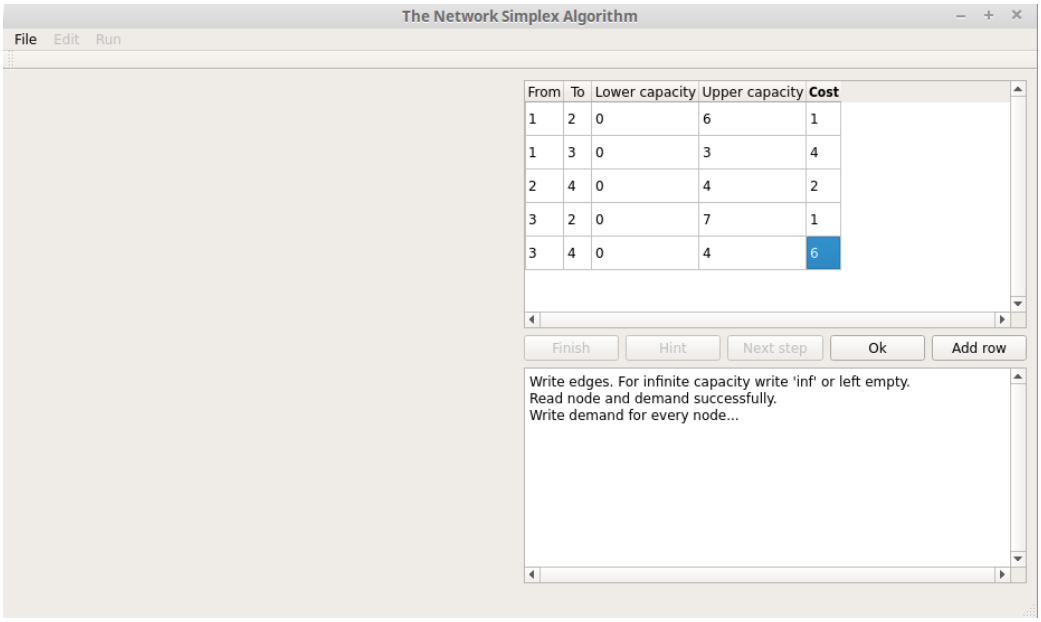


Slika 2.1: Unos napajanja.

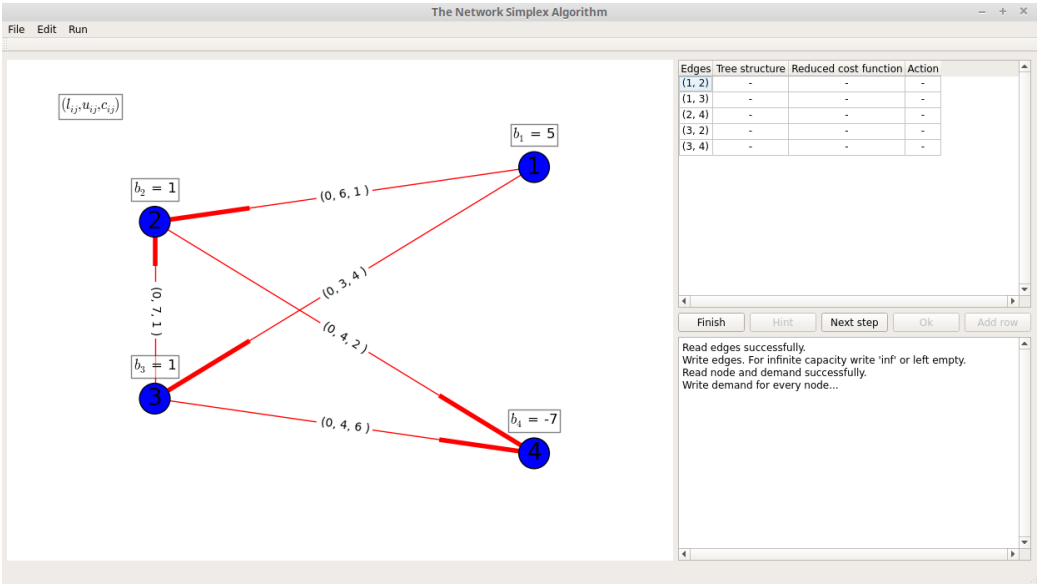
varajući gumb *Add*. Prilikom klika stvara se ciklus u generirajućem stablu T. Aplikacija će korisniku taj pivotni ciklus prikazati i omogućiti izbacivanje luka kojeg želi(Slika 2.5). Također mu je označena i orijentacija pivotnog ciklusa. Sada slijedi odabir izlaznog luka u pivotnom ciklusu. Korisnik može kliknuti na gumb *Hint* kako bi dobio pomoć oko odabira izlaznog luka. Također, može kliknuti na gumb *Next step* i aplikacija će sama odraditi izbacivanje uzimajući u obzir odabrani ulazni luk. Pretpostavimo da je korisnik kliknuo na *Hint*. Tada dobije pomoć u obliku poruke u kojoj je napisan luk kojeg bi trebao izabrati kao izlazni luk(Slika 2.6). Izlazni luk je odabran po napisanom pravilu za odabir izlaznog luka. Klikom na *Throw* uz luk $(3, 0)$, luk se izbaci iz T i dodamo ga L ili U ovisno o novom ažuriranom toku (Slika 2.7). U našem slučaju, dodan je u L. Nastavimo tako i dolazimo do optimalnog bazičnog toka kojem je cijena minimizirana. Slika 2.8 prikazuje optimalni bazični tok problema (P) koji je dobiven primjenjujući definirane operacije u aplikaciji.

Primijetimo da se svaki korak bilježi i ispisuje korisniku aplikacije. Na taj način korisnik u svakom trenutku zna što je učinio u dosadašnjem radu.

Aplikacija se može podijeliti na tri dijela. Dio za prikaz grafova, dio za prikaz tablice s podacima i dio za prikaz poruka. U dijelu namijenjenoga za poruke, aplikacija ispisuje povijest radnji koje je izvršio. Također, ispisuje i poruke vezane za pomoć prilikom odabira izlaznog luka. Dio za prikaz tablice je dio u kojem aplikacija komunicira s korisnikom. Dio



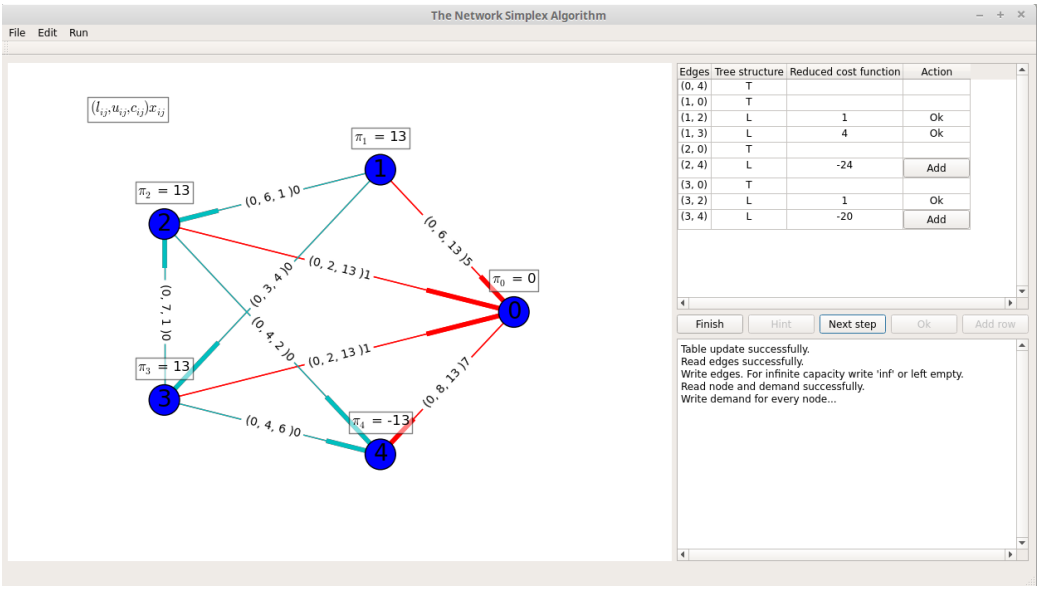
Slika 2.2: Unos lukova.



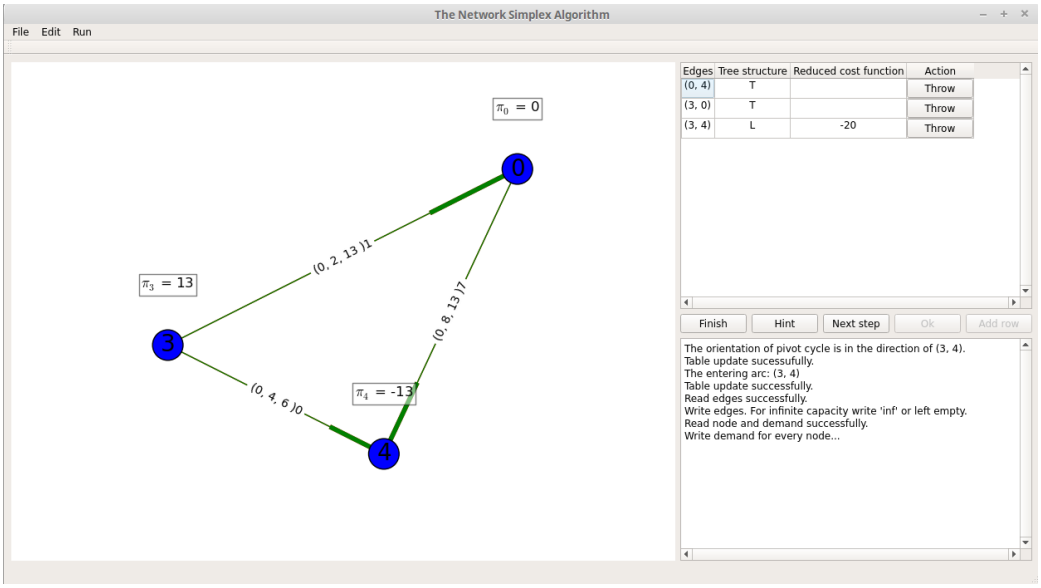
Slika 2.3: Zadani početni graf

za prikaz grafova služi za prikaz grafova, te također odgovarajućih ciklusa.

Navedimo neke mogućnosti same aplikacije:

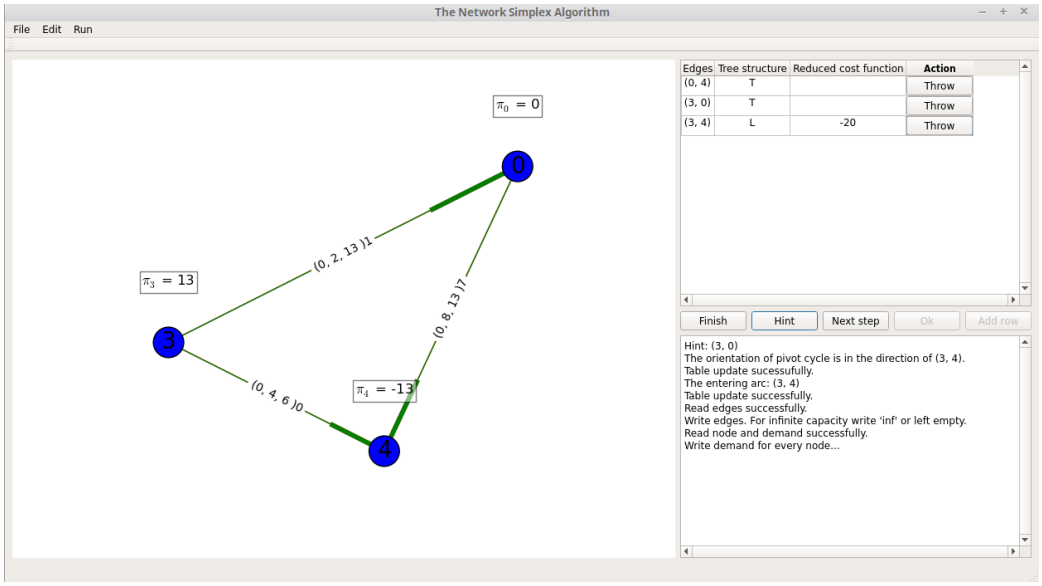


Slika 2.4: Pomoćni problem

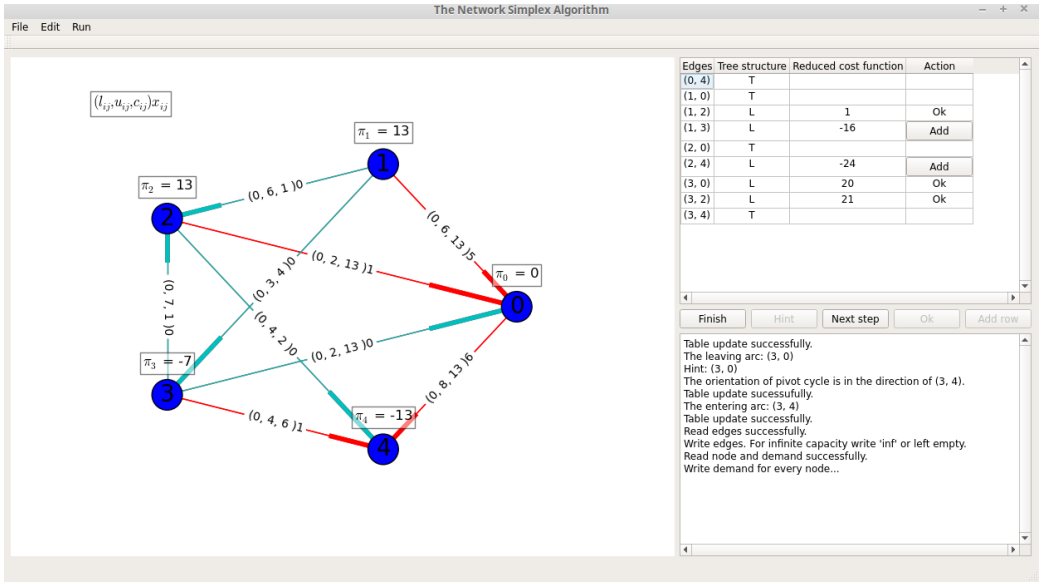


Slika 2.5: Ciklus koji je nastao dodavanjem luka u T.

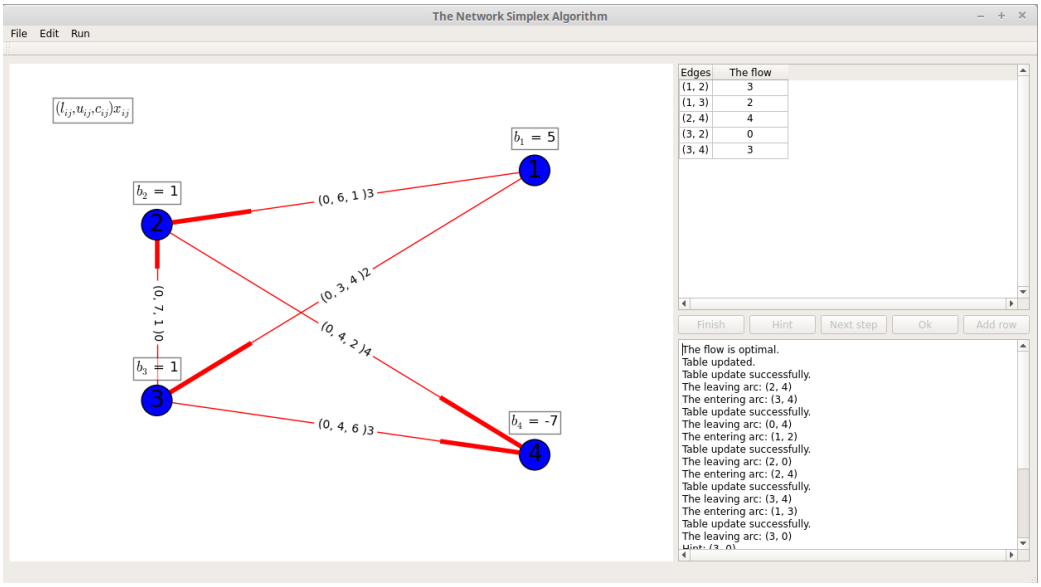
- Unos željenog grafa kroz aplikaciju.
- Učitavanje grafa iz lokalne datoteke.



Slika 2.6: Pomoć za odabir izlaznog luka.



Slika 2.7: Stablo nakon izbacivanja luka.



Slika 2.8: Optimalni bazični tok problema (P).

- Spremanje grafa na željenu lokaciju. Prilikom spremanja vrijednost donjeg kapaciteta ne mora biti 0, za razliku od pretpostavke Networkx-a za vrijednost donjeg kapaciteta.
- U svakom koraku algoritma, graf prikazan u dijelu za prikaz grafova, se može spremiti na željenu lokaciju u obliku slike (npr. png format).
- Završiti algoritam u jednom kliku. Aplikacija će u pozadini sama izvršiti algoritam do kraja od koraka do kojeg je došao.
- Izvršavati algoritam korak po korak.
- Odabir izlaznog i ulaznog luka.

Za više informacija je najbolje pogledati aplikaciju. Ovom demonstracijom došli smo do kraja diplomskoga rada. Aplikacija trenutno u verziji 1.0. Aplikacija će se dalje razvijati. Cilj je učiniti unos grafa što učinkovitije i lakše. Kao i mnoge druge funkcionalnosti. Spomenute nadogradnje se očekuju u verziji 2.0.

Bibliografija

[1] M. Jungnickel, *Graphs, Networks and Algorithms*, Fourth Edition, Springer, 2013.

[2] M.S. Bazaraa, J.J. Jarvis, *Linear programming and network flows*, Wiley, 2010.

[3] J.B. Orlin, *A polynomial time primal network simplex algorithm for minimum cost flows*, Cambridge, 1996.

[4] *L. Čaklović*,*Predavanja o Pythonu, Qt5 i Pyqt5*, `http://viveka.math.hr/nastava/Qt5/`.

[5] *NetworkX*, `https://networkx.github.io/`.

Sažetak

U ovom radu je opisan problem minimizacije troškova toka na mreži u terminima teorije grafova. Definiran je konkretan algoritam za mrežnu simpleks metodu. Zatim je dana efikasna implementacija algoritama. Na kraju, dana je implementacija u konkretnom programskom jeziku Python. Implementirani algoritam je integriran u PyQt5 aplikaciju koja koristi Qt5 framework. Aplikacija u interakciji s korisnikom rješava zadani problem minimizacije troškova toka na mreži. Aplikacija je nastala u edukacijske svrhe i samo u te svrhe će se i koristi.

Summary

In this thesis the minimum cost flow problem is described in terms of the Graph theory. A particular algorithm for the network simplex method is defined. An efficient application of the algorithm is described. Lastly, implementation in the programming language Python is given. The implemented algorithm is integrated in PyQt5 application which is using the Qt5 framework. This application, interacting with the user, solves the given minimum cost flow problem. The application is made for educational purposes and will be used for those purposes only.

Životopis

Moje ime je Jure Milašinović. Rođen sam 13.04.1992. u selu Kaniži pokraj Slavonskoga Broda. Godine 1998. sam upisao osnovnu školu Antun Matija Reljković u Bebrini. Nakon što sam završio osnovnu školu, 2007. godine sam upisao matematičku gimnaziju Matija Mesić u Slavonskom Brodu. U školi sam najviše volio matematiku. Iza matematike slijedila je informatika i fizika. U srednjoj školi smo radili u programskom jeziku Pascal. Općenito sam volio sve predmete osim jezika. Bio sam na raznim natjecanjima i ostvario solidne rezultate. Moj hobi u to vrijeme je bio nogomet. Igrao sam u 2. ŽNL za juniorski i seniorski sastav. Nakon srednje, 14. 07. 2011. godine upisao sam se na preddiplomski studij na Prirodoslovni-matematički fakultet, smjer Matematika u Zagrebu. Tada kreće ozbiljno bavljenje matematikom, te polako i s računarstvom. Bavljenje nogometom je polako slabilo, tako da sam karijeru nastavio u 3.ŽNL. Osim nogometa, davao sam instrukcije iz matematike te sam također volontirao u župi Marije Pomoćnice - Knežija kao instruktor matematike. Do kraja preddiplomskog studija odlučio sam upisati diplomski studij računarstvo i matematika na istom fakultetu. 2014. godine sam postao sveučilišni prvostupnik(baccalaureus) matematike(univ.bacc.math). Diplomski studij sam upisao 2014. godine. Počeo sam ozbiljnije raditi na svojim programerskim vještinama. Radio sam razne web aplikacije, android aplikacije, web stranice i mnoge druge aplikacije. 2015. sam uključen u projekt kao student teorijskog računarstva za potrebe EU-projekta: HR.3.1.15-0017 Razvoj studija ekologije, računarstva i matematike uz primjenu Hrvatskog kvalifikacijskog okvira- EkoRaMa. Projekt je još u tijeku. Na kraju bih istaknuo osvojenu drugu nagradu za natječaj za najbolji studentski rad iz područja primijenjene matematike, statistike i računarstva koji je provodilo Hrvatsko matematičko društvo, PMF-MO i Udruženja društava za upravljanje mirovinskim fondovima i mirovinskih osiguravajućih društava, Zagreb (Hrvatska).