Algoritmi za traženje maksimalnog toka minimalne cijene u mreži

Bujanović, Tomislav

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Tomislav Bujanović

ALGORITMI ZA TRAŽENJE MAKSIMALNOG TOKA MINIMALNE CIJENE U MREŽI

Diplomski rad

Voditelj rada:
prof. dr. sc. Robert Manger

Zagreb, rujan, 2017.

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

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

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

U ovom radu donosimo pregled nekoliko algoritama koji rješavaju jedan od poznatijih problema u teoriji grafova, problem pronalaska maksimalnog toka minimalne cijene u mreži.

U prvom poglavlju ukratko će se ponoviti nekoliko pojmova koji su ključni za sam problem, kao što su pojam grafa (tj. mreže), puta i ciklusa. U drugom dijelu poglavlja opisat ćemo i problem nalaska maksimalnog toka u težinskom grafu, što predstavlja bitan dio rješavanja početnog problema.

U drugom poglavlju, polazeći od definicije funkcije cilja koju je potrebno minimizirati, uvest ćemo nekoliko pojmova koji će biti temelj za predložene algoritme, a neki od njih su pojam potencijala i reduciranih cijena. Uz nekoliko tvrdnji koje ćemo pritom dokazati dobit ćemo i teoretsku podlogu koja će jamčiti ispravnost algoritama. Konačno, u ovom poglavlju predstavit ćemo i tri algoritma koji na različite načine rješavaju problem maksimalnog toka minimalne cijene.

Zadnje poglavlje bit će usmjereno k usporedbi performansi prikazanih algoritama. Osim uspješnosti pronalaska točnog rješenja, zanimat će nas i neka druga svojstva, kao što su vrijeme izvođenja i broj iteracija svakog od algoritama.

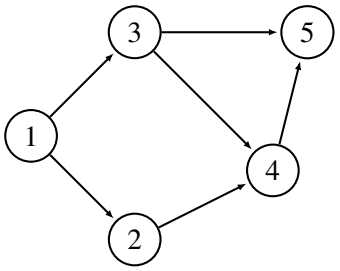
Poglavlje 1

Osnovni pojmovi teorije grafova. Maksimalni tok u mreži

1.1 Uvodni pojmovi

Na početku ovog poglavlja ponovit ćemo neke osnovne definicije i rezultate koji su poznati u teoriji grafova.

U matematici razlikujemo usmjerene i neusmjerene grafove. Usmjereni graf G se najčešće definira kao uređeni par (V, E) gdje je V skup vrhova, a E skup bridova, pri čemu je $E \subseteq V \times V$. Neusmjereni graf tada se definira kao usmjereni graf kod kojeg za svaki brid (i, j) postoji i obratni brid (j, i). U ovom radu prvenstveno ćemo promatrati usmjerene grafove, a smatrat ćemo i da između svaka dva para vrhova (i, j) postoji najviše jedan brid koji odgovara tom paru vrhova. Preciznije, najviše jedan brid spaja neka dva vrha. Na sljedećoj slici dan je primjer usmjerenog grafa koji se sastoji od pet vrhova i šest bridova.



Slika 1.1: Primjer usmjerenog grafa

Stupanj nekog vrha definira se kao broj svih bridova koji ulaze i izlaze iz tog vrha. Primjerice, za graf sa slike 1.1 vrijedi da je stupanj vrha 4 jednak 3. Nadalje, kao konkretni zapis

grafa često se koristi matrica susjedstva ukoliko broj vrhova nije prevelik. Ako broj vrhova V grafa G označimo s n, onda se radi o matrici dimenzije $n \times n$, pri čemu u retku i i stupcu j piše 1 ako postoji brid od vrha i do j, a 0 inače. Za prethodni graf matrica izgleda ovako:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

Šetnja u usmjerenom grafu $G = (V, E)$ je niz $v_1 - v_2 - \ldots - v_k$, gdje za sve $1 \leq i \leq k-1$ vrijedi da postoji $e \in E$ takav da je $e = (v_i, v_{i+1})$. Put je šetnja u kojoj se vrhovi ne ponavljaju. Put se najčešće pamti pomoću liste prethodnika, gdje se za svaki vrh pamti tko je njegov prethodnik u tom putu. Primjerice, za put $1 - 2 - 4 - 5$ u prethodnom grafu, vrh 1 nema prethodnika, dok su za vrhove $2, 4$ i 5 prethodnici $1, 2, 4$, redom. Ciklus se definira kao šetnja u kojoj su prvi i zadnji vrh jednaki.

U teoriji grafova javlja se i podjela na težinske i grafove bez težina. Težinski grafovi najčešće se mogu dijeliti na one koji imaju težine na bridovima i/ili vrhovima. Kažemo da graf ima težine na bridovima ako je zadana funkcija $w : E \to \mathbb{R}$, a da ima težine na vrhovima ako je zadana i funkcija $h : V \to \mathbb{R}$. U ovom radu javljat će se grafovi koji će imati težine i na bridovima i na vrhovima. Težine na bridovima imat će značenje maksimalne količine toka koji je moguće prenijeti tim bridom (odnosno, radi se o kapacitetu toka) te uvjetno i cijene koju pritom plaćamo, dok težine na vrhovima označavaju razliku u količini toka koju taj vrh prima i šalje dalje.

Kapacitet i cijena mogu se definirati kao funkcije $U : E \to \mathbb{R}_+$, $C : E \to \mathbb{R}$, dok se težine, tj. zahtjevi pojedinih vrhova definiraju funkcijom $b : V \to \mathbb{R}$.

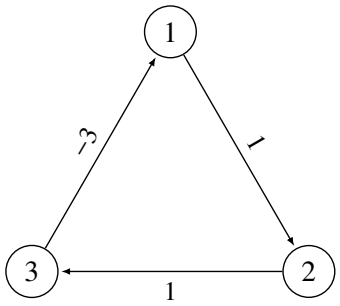
1.2 Algoritmi za traženje najkraćih puteva

Mnogi algoritmi za pronalazak najkraćih puteva koriste se upravo u slučaju težinskih grafova. Neki od najpoznatijih su breath-first search (BFS), Dijkstrin, Floyd-Warshallov te Bellman-Fordov algoritam, a neke od njih koristit ćemo i u ovome radu. Ovdje ćemo opisati Dijkstrin i Bellman-Fordov algoritam.

Dijkstrin algoritam vjerojatno je najpoznatiji algoritam za pronalazak najkraćih puteva u grafu. Za dani početni vrh s, algoritam pronalazi najkraću udaljenost od vrha s do svakog drugog vrha u grafu. Osim najkraće udaljenosti, za svaki vrh v najčešće se pamti i prethodnik tog vrha u najkraćem putu što kasnije omogućuje da se za svaki vrh pronađe najkraći put do tog vrha.

Opišimo ukratko rad algoritma. Struktura podataka koja se najčešće koristi u algoritmu je prioritetni red ili neka druga struktura koja ima logaritamsko vrijeme dohvaćanja najmanjeg elementa u skupu, pri čemu je kriterij minimalnosti udaljenost od početnog vrha. U početku ta struktura sadrži samo početni vrh s s udaljenošću nula. U svakoj iteraciji iz strukture se dohvaća vrh v koji ima trenutno najkraću udaljenost od početnog vrha te se briše iz strukture. Za svakog od njegovih susjeda u uspoređuje se njegova najkraća udaljenost od vrha s s udaljenošću vrha v od s zbrojenom s duljinom brida (v, u). Ako prolaskom kroz vrh v dobivamo kraću udaljenost, najkraća udaljenost za vrh u se ažurira i kao prethodnik vrha u se postavlja vrh v te se vrh u dodaje u strukturu podataka ako put do njega do sada nije postojao ili se ažurira, inače. Kada u strukturi ne ostane niti jedan vrh, algoritam staje. Ako je broj vrhova u grafu n, a broj bridova m, složenost Dijkstrinog algoritma uporabom efikasne strukture podataka je $O(m \log n)$.

Kako bi algoritam radio ispravno, nužno je da sve duljine bridova budu nenegativne. Primjer kada algoritam ne bi dao točan rezultat u slučaju da je neki brid negativne duljine vidljiv je na slici 1.2. Budući da je duljina brida $(3, 1)$ negativna i po apsolutnoj vrijednosti veća od zbroja duljina preostala dva brida, algoritam će u svakom prolasku ciklusa smanjivati duljinu najkraćeg puta od, primjerice, vrha 1 do vrha 3, tj. algoritam će raditi beskonačno.



Slika 1.2: Primjer grafa za koji Dijkstrin algoritam ne radi ispravno

Bellman-Fordov algoritam također pronalazi najkraće udaljenosti i puteve od nekog početnog vrha do svih drugih vrhova u težinskom grafu. Za razliku od Dijkstrinog algoritma, točna rješenja mogu se pronaći i u slučaju da neki od bridova imaju negativnu duljinu (cijenu). Ipak, algoritam ne radi ispravno ako se u grafu pojavi negativni ciklus (tj. ciklus u kojem je zbroj težina bridova negativna).

Kao i Dijkstrin algoritam, i Bellman-Fordov algoritam kroz iteracije pokušava smanjiti cijenu kojom je moguće doći od početnog do svakog drugog vrha u grafu. Međutim, pri tome se ne koristi prioritet u odnosu na cijenu koji bi neki vrh mogao imati, nego se kroz iteracije prolaze svi bridovi grafa i pokušava smanjiti cijena najkraćeg puta do pojedi-

nog vrha. Po završetku i-te iteracije algoritam će pronaći sve najkraće puteve koji koriste najviše i bridova. Budući da najdulji put koji nije ciklus može imati $n - 1$ brid, ukupan broj iteracija mora biti $n - 1$. Ako se u n-toj iteraciji cijena puta do nekog vrha promijeni, tada se pojavio najkraći put koji prolazi kroz n bridova, što je jedino moguće ako postoji negativan ciklus u grafu.

Bellman-Fordov algoritam očito ima složenost $O(mn)$, što je lošije nego u slučaju Dijkstrinog algoritma. Ipak, prednost Bellman-Fordovog algoritma je što je upotrebljiv i na grafovima koji imaju neke bridove negativne težine. Također, algoritam ćemo koristiti i u slučaju kada budemo htjeli provjeriti postojanje negativnog ciklusa u grafu.

1.3 Maksimalni tok kroz mrežu

Prvi problem koji ćemo rješavati u ovom radu je problem pronalaska maksimalnog toka kroz graf, koji u kontekstu ovog problema češće nazivamo mrežom. Ovaj problem donekle je sličan problemu pronalaska najkraćeg puta u mreži i oba se javljaju u problemu pronalaska maksimalnog toka minimalne cijene. Ipak, bitna razlika je ta što u problemu nalaska maksimalnog toka moramo voditi računa o gornjoj granici toka koju pojedini brid u mreži dopušta, dok kod traženja najkraćeg puta minimiziramo cijenu kojom možemo doći do pojedinog vrha.

Definirajmo problem maksimalnog toka: u povezanoj mreži u kojoj bridovi imaju kapacitete, potrebno je poslati maksimalnu količinu toka između dva istaknuta vrha, izvora i ponora, tako da niti kod jednog brida kapacitet nije premašen. Preciznije: Neka je dana mreža $G = (V, E)$ s nenegativnim kapacitetima u_{ij}, za svaki brid $e_{ij} = (i, j) \in E$. Neka je $U = max\{u_{ij} : (i, j) \in E\}$. Tok definiramo kao broj x_{ij}, pri čemu je $0 \leq x_{ij} \leq u_{ij}$. Dalje, pretpostavimo da postoje dva vrha koje zovemo izvor s i ponor t. Cilj je pronaći maksimalan mogući tok od s do t tako da su zadovoljeni svi kapaciteti i zahtjevi u pojedinim vrhovima, tj. da vrijedi Kirchoffov zakon o održanju toka u mreži. Za svaki vrh promatramo razliku toka koji izlazi i ulazi u taj vrh. Izvor s ima višak toka od f jedinica toka, dok ponor t ima manjak toka od f jedinica; za sve ostale vrhove količina toka koji izlazi iz vrha mora biti jednaka količini toka koji ulazi u taj vrh. Dakle, zadatak je pronaći vrijednosti x_{ij} kojima se maksimizira vrijednost f uz sljedeće uvjete:

$$\sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = \begin{cases} f & \text{za } i = s \\ 0 & \text{za sve } i \in V \setminus \{s,t\} \\ -f & \text{za } i = t \end{cases} \tag{1.1}$$
$$0 \leq x_{ij} \leq u_{ij}, \text{ za sve } (i,j) \in E.$$

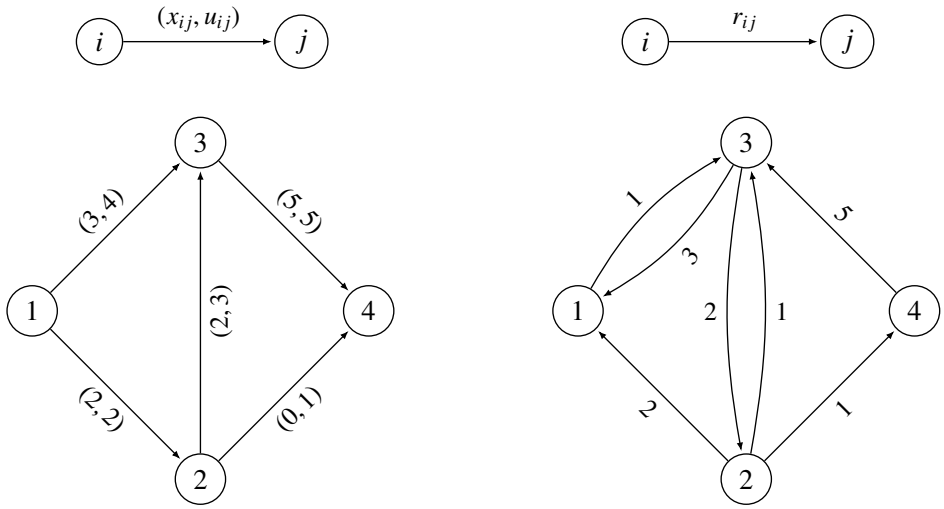
Skup $x = \{x_{ij}\}$ zvat ćemo tok u mreži dok ćemo broj f zvati vrijednost toka u mreži. Lijeva strana izraza (1.1) jednaka je razlici toka koji izlazi i ulazi u vrh i.

Pretpostavke koje vrijede za mrežu:

- mreža je usmjerena;
- svi kapaciteti su prirodni brojevi. Uočimo da ako i nije tako, budući da računala u memoriji pamte brojeve kao racionalne, množenjem s dovoljno velikim brojem (najmanjim zajedničkim višekratnikom nazivnika svih brojeva) možemo postići da su svi kapaciteti zaista prirodni brojevi;
- u mreži ne postoji put od izvora do ponora koji prolazi samo lukovima koji imaju beskonačan kapacitet. Ova pretpostavka je bitna jer onda očito vrijednost maksimalnog toka nije odozgo omeđena;
- mreža ne sadrži paralelne lukove, odnosno, ne postoji više lukova koji direktno povezuju neka dva vrha.

1.4 Rezidualna mreža i rezovi

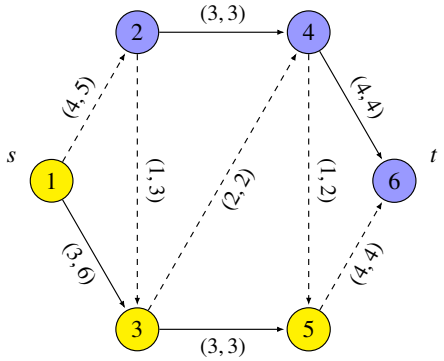
Rezidualna mreža jedan je od ključnih pojmova koji se javljaju u slučaju traženja maksimalnog toka. Drugi naziv koji bi dobro opisivao ovu mrežu je i mreža preostalog toka. Rezidualna mreža sastoji se od istog skupa vrhova kao i originalna mreža, dok se za svaki brid originalne mreže u rezidualnoj pojavljuje jedan ili dva brida. Pretpostavimo da brid (i, j) ima trenutni tok od x_{ij} jedinica. Tada je moguće poslati još $u_{ij} - x_{ij}$ jedinica toka od vrha i do vrha j. U rezidualnoj mreži tada se definira da brid (i, j) ima tzv. *rezidualni kapacitet* jednak $u_{ij} - x_{ij}$. Ako je tok x_{ij} pozitivan, tada definiramo da u rezidualnoj mreži postoji i obrnuti brid (j, i) rezidualnog kapaciteta x_{ij}. Možemo zaključiti da rezidualni kapacitet r_{ij} nekog brida $(i, j) \in E$ ima dvije komponente: $u_{ij} - x_{ij}$, što označava neiskorišteni tok na bridu (i, j) te trenutni tok x_{ij} na bridu (j, i). Rezidualnu mrežu definiranu nekim tokom x ćemo odsad nadalje označavati s $R(x)$. Na sljedećoj slici vidljiva je konstrukcija rezidualne mreže.



(a) originalna mreža (b) rezidualna mreža

Slika 1.3: Primjer konstrukcije rezidualne mreže

Rez je particija skupa vrhova V na dva skupa S i $T = V \setminus S$. Za takav rez koristit ćemo oznaku $[S, T]$. Za rez kažemo da je $s - t$ rez ako je $s \in S$ i $t \in T$. Za brid $(i, j) \in E$ za koji je $i \in S$ i $j \in T$ kažemo da je izlazni brid reza $[S, T]$, a ako je $i \in T$ i $j \in S$ govorimo o povratnom bridu reza $[S, T]$. Neka (S, T) označava skup svih izlaznih, a (T, S) skup svih povratnih bridova reza $[S, T]$. Na slici 1.4 prikazan je jedan $s - t$ rez dane mreže. Isprekidanim linijama prikazani su izlazni i povratni bridovi. Za konstruirani rez vrijedi $(S, T) = \{(1, 2), (3, 4), (5, 6)\}$, te $(T, S) = \{(2, 3), (4, 5)\}$.



Slika 1.4: Prikaz reza $[S, T]$ za $S = \{1, 3, 5\}$, $T = \{2, 4, 6\}$

Kapacitet $s-t$ reza $[S,T]$, u oznaci $u[S,T]$ je suma kapaciteta svih izlaznih bridova, tj.

$$u[S,T] = \sum_{(i,j)\in(S,T)} u_{ij}. \tag{1.2}$$

Minimalni rez definira se kao $s-t$ rez čiji je kapacitet najmanji.

Pokažimo sada neka svojstva minimalnog reza. Sumiranjem jednadžbi (1.1) po svim vrhovima u S dobivamo da vrijedi:

$$f = \sum_{i\in S}\left(\sum_{\{j:(i,j)\in E\}} x_{ij} - \sum_{\{j:(j,i)\in E\}} x_{ji}\right). \tag{1.3}$$

Uočimo da se prethodni izraz može pojednostaviti. Ako za neka dva vrha p i q vrijedi da se oba nalaze u skupu S te postoji brid $(p,q)\in E$, očito se pojavljuje izraz x_{pq} za $i=p, j=q$ te izraz $-x_{pq}$ za $i=q, j=p$, pa se ta dva izraza ponište u gornjoj sumi. Također, ako i p i q pripadaju skupu T, tada se izraz x_{pq} ne pojavljuje u sumi. Ovime zaključujemo da je jednakost (1.3) ekvivalentna s:

$$f = \sum_{(i,j)\in(S,T)} x_{ij} - \sum_{(i,j)\in(T,S)} x_{ij}. \tag{1.4}$$

Prva suma u prethodnoj jednakosti odgovara količini toka iz vrhova skupa S u vrhove skupa T, dok druga suma odgovara toku iz vrhova skupa T u vrhove skupa S. Dakle, iz gornje jednadžbe zaključujemo da je tok u bilo kojem $s-t$ rezu $[S,T]$ jednak f. Budući je $0\le x_{ij}\le u_{ij}$ te $\sum_{(i,j)\in(T,S)} x_{ij}\ge 0$, vrijedi:

$$f \le \sum_{(i,j)\in(S,T)} u_{ij} = u[S,T]. \tag{1.5}$$

Time smo dokazali da je vrijednost proizvoljnog toka manja ili jednaka od kapaciteta bilo kojeg $s-t$ reza u mreži.

Sada nam je cilj pokazati na koji način je moguće povećati trenutni tok u mreži ukoliko on nije maksimalan. Definirajmo pojam *puta povećanja toka* kao usmjereni put od izvora do ponora u rezidualnoj mreži. Nadalje, definiramo kapacitet puta povećanja toka kao minimalni rezidualni kapacitet bilo kojeg brida na tom putu. Primjerice, rezidualna mreža na slici 1.3b sadrži samo jedan put povećanja toka: $1-3-2-4$. Njegov rezidualni kapacitet je $\delta = min\{r_{13}, r_{32}, r_{24}\} = min\{1,2,1\} = 1$. Uočimo da je po definiciji kapacitet puta povećanja toka δ uvijek nenegativan. Upravo taj tok δ možemo iskoristiti kako bismo povećali trenutni tok u mreži. Time dobivamo i ideju za algoritam: sve dok postoji neki put povećanja toka u rezidualnoj mreži, identificiraj kapacitet δ tog puta i povećaj tok u mreži.

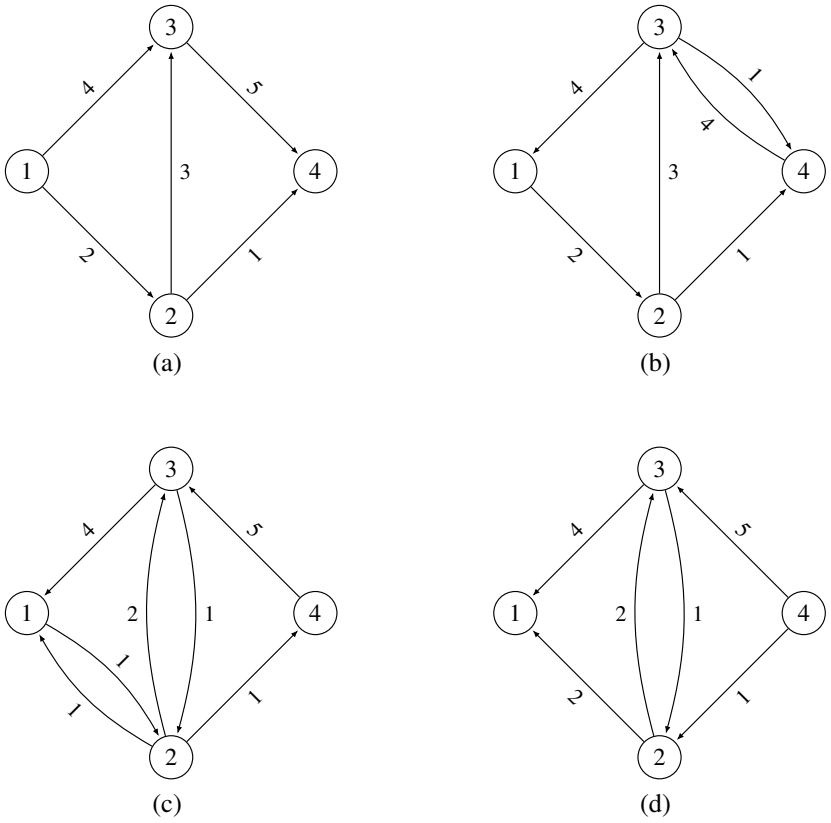
Algorithm 1 Korištenje puta povećanja toka

1: $x = 0$
2: **while** $R(x)$ sadrži usmjereni put od s prema t **do**
3: pronađi usmjereni put P od s prema t u $R(x)$
4: $\delta = \min\{r_{ij} : (i, j) \in P\}$
5: dodaj δ jedinica toka na putu P u $R(x)$, tj. za svaki brid $(i, j) \in P$:
$\cdot\ r_{ij} = r_{ij} - \delta$
$\cdot\ r_{ji} = r_{ji} + \delta$
6: ažuriraj rezidualnu mrežu $R(x)$
7: **end while**

Pokažimo sada na jednom primjeru kako je moguće pronaći maksimalan tok u mreži. Na slici 1.5a prikazana je originalna mreža koja je ujedno i rezidualna na početku traženja toka. Pretpostavimo da algoritam najprije pronađe put $1 - 3 - 4$. Rezidualni kapacitet tog puta je $\delta = min\{r_{13}, r_{34}\} = min\{4, 5\} = 4$. Nakon povećanja toka, rezidualni kapacitet brida $(1, 3)$ postaje nula, kapacitet brida $(3, 4)$ pada na 1, dok se kapaciteti bridova $(4, 3)$ i $(3, 1)$ postavljaju na 4. Na slici 1.5b prikazano je stanje rezidualne mreže nakon povećanja toka. Neka je kao sljedeći put od izvora do ponora pronađen put $1 - 2 - 3 - 4$. Rezidualni kapacitet tog puta je $\delta = min\{r_{12}, r_{23}, r_{34}\} = min\{2, 3, 1\} = 1$. Nakon povećanja toka za 1 duž tog puta, kapaciteti bridova $(1, 2), (2, 3), (3, 4)$ smanjeni su za 1, dok se kapaciteti bridova $(4, 3), (3, 2)$ i $(2, 1)$ povećavaju za 1, kako je i prikazano na slici 1.5c. Konačno, preostao je još jedan put od izvora do ponora i to je put $1 - 2 - 4$ rezidualnog kapaciteta $\delta = min\{r_{12}, r_{24}\} = min\{1, 1\} = 1$. Nakon povećanja puta, dolazimo do konačnog stanja rezidualne mreže koje je prikazano na slici 1.5d.

1.5 Ford-Fulkersonov algoritam

U ovom odlomku opisat ćemo detaljnije korake u pronalaženju maksimalnog toka u mreži, tj. zanimat će nas prvenstveno dva pitanja. Prvo, kako identificirati postoji li u mreži put povećanja toka te drugo, da li algoritam staje u konačno mnogo vremena i daje li na kraju maksimalan mogući tok. Najpoznatiji algoritam za pronalaženje maksimalnog toka je Ford-Fulkersonov algoritam kojeg su 1956. godine objavili L. R. Ford i D. R. Fulkerson. Ford-Fulkersonov algoritam koristi tehniku pretraživanja u širinu (eng. *breath-first search, BFS*) kako bi pronašao put od izvora do ponora. Za svaki vrh u mreži pamti se da li je bio posjećen na nekom putu u rezidualnoj mreži krenuvši od izvora. U trenutku kada algoritam dođe do nekog vrha i, on pokušava doći do svakog od njegovih susjeda u rezidualnoj mreži koji još nisu bili posjećeni. Ako je j jedan od takvih susjeda, tada se bilježi da se može stići do vrha j te j bilježi da je njegov prethodnik vrh i - na taj način će na kraju pretrage



Slika 1.5: Primjer traženja maksimalnog toka. Težine bridova predstavljaju rezidualne kapacitete.

vrhova biti moguće odrediti put od izvora do ponora ukoliko on postoji. Ako se na kraju pretraživanja pokaže da se do ponora može doći, dobili smo jedan put povećanja toka. Nakon toga algoritam šalje maksimalnu količinu toka koji je moguć po tom putu, te se ponovno vraća na pronalazak novog puta od izvora do ponora u modificiranoj rezidualnoj mreži. Algoritam završava u trenutku kada se pretragom u širinu ispostavi da nije moguće doći do ponora, čime se zapravo implicira da ne postoji put povećanja toka u mreži.

Pokažimo ispravnost Ford-Fulkersonovog algoritma, tj. da prilikom završetka algoritma zaista dobivamo maksimalni tok u mreži. U svakom koraku postoje dvije mogućnosti: put od izvora do ponora postoji pa je moguće povećati tok ili put ne postoji pa ponor nije označen te algoritam završava. Potrebno je pokazati da u drugom slučaju mreža ima maksimalni tok. Neka skupu S pripadaju svi vrhovi koji su označeni te skupu T vrhovi koji su

Algorithm 2 Ford-Fulkersonov algoritam

```
1:  while true do
2:      postavi POSJEĆEN[v] = false za sve vrhove v
3:      postavi RODITELJ[v] = −1 za sve vrhove v
4:      inicijaliziraj prazni red Q
5:      postavi POSJEĆEN[s] na true i stavi s na kraj reda Q
6:      while red Q nije prazan i POSJEĆEN[t] == false do
7:          ukloni vrh i sa početka reda Q
8:          for za svaki brid (i, j) u rezidualnoj mreži do
9:              if r_ij > 0 i vrh j još nije posjećen then
10:                 postavi RODITELJ[j] = i
11:                 postavi POSJEĆEN[j] = true
12:                 dodaj vrh j na kraj reda Q
13:             end if
14:         end for
15:     end while
16:     if POSJEĆEN[t] == false then
17:         break
18:     else
19:         POVEĆAJ_TOK
20:     end if
21: end while
22: procedure POVEĆAJ_TOK
23:     pomoću oznaka roditelja iz niza RODITELJ pronađi put P od izvora s do ponora t
24:     δ = min{r_ij : (i, j) ∈ P}
25:     povećaj tok na putu P za δ jedinica toka
26: end procedure
```

ostali neoznačeni. Očito, $S \cup T = V$. Također, uočimo da je $s \in S$ te $t \in T$. Budući da algoritam nije mogao označiti niti jedan vrh iz T, zaključujemo da je $r_{ij} = 0$ za sve bridove $(i, j) \in (S, T)$. Kako je $r_{ij} = u_{ij} - x_{ij}$, dobivamo da je $x_{ij} = u_{ij}$ za svaki brid $(i, j) \in (S, T)$, pa je nužno $x_{ij} = 0$, za svaki brid $(i, j) \in (T, S)$ u rezidualnoj mreži. Sada je:

$$f = \sum_{(i,j)\in(S,T)} x_{ij} - \sum_{(i,j)\in(T,S)} x_{ij} = \sum_{(i,j)\in(S,T)} u_{ij} = u[S, T]. \tag{1.6}$$

Dakle, vrijednost toka jednaka je kapacitetu $s - t$ reza $[S, T]$. U jednakosti (1.5) imali smo da je vrijednost toka uvijek manja ili jednaka od vrijednosti svakog $s - t$ reza. Dakle, sada smo dobili da je f poprimio gornju granicu, a kapacitet reza svoju donju granicu,

odnosno, f je maksimalan tok, a $u[S,T]$ je minimalan rez. Time smo dokazali i sljedeći bitan teorem, u literaturi često poznat kao *Max-Flow Min-Cut Theorem*.

Teorem 1.5.1. *Maksimalni tok u mreži od izvora s do ponora t jednak je minimalnom kapacitetu svih mogućih $s-t$ rezova.*

Zapravo smo usput dokazali još jedan teorem koji je usko povezan s ispravnošću Ford-Fulkersonovog algoritma.

Teorem 1.5.2. *Tok x^* je maksimalan ako i samo ako rezidualna mreža $R(x^*)$ ne sadrži niti jedan put povećanja toka.*

Iskažimo još jedan rezultat opisanog algoritma.

Propozicija 1.5.3. *Ako su kapaciteti svih bridova prirodni brojevi, onda i dobiveni maksimalni tok kao rješenje ima samo prirodne brojeve.*

Dokaz. Dokaz možemo provesti matematičkom indukcijom po broju koraka u pronalaženju novih puteva povećanja toka. Budući da algoritam započinje s nul tokom, odnosno tokom koji je na svim bridovima jednak nula, baza indukcije je zadovoljena. Kada se pronađe neki put povećanja toka, tok kroz mrežu se povećava za minimum od svih kapaciteta bridova na tom putu, što je po pretpostavci indukcije prirodan broj (uočimo da su svi kapaciteti na tom putu barem jedan). Dakle, novi rezidualni kapaciteti na bridovima na kojima postoji tok će ponovno biti prirodni brojevi. Budući da se tok u svakom koraku povećava barem za jedan, u konačno mnogo koraka doći ćemo do maksimalnog toka u mreži koja će, induktivno, tada imati također količine toka po bridovima s vrijednostima u prirodnim brojevima. Time je dokaz dovršen. □

Složenost Ford-Fulkersonovog algoritma

Ocijenimo složenost Ford-Fulkersonovog algoritma. Neka je $|V| = n$, $|E| = m$, te neka je U najveća vrijednost kapaciteta kojeg neki brid ima u mreži. Uočimo da svaka iteracija algoritma započinje označavanjem vrhova do kojih možemo doći nekim putem u rezidualnoj mreži ako krenemo od izvora s. Budući da prilikom posjete nekog vrha algoritam pokušava doći do svakog od njegovih susjeda, očito složenost ovog pretraživanja metodom BFS-a iznosi $O(m)$ jer je moguće da se pokušaju ispitati svi postojeći bridovi u mreži. U drugom dijelu svake iteracije u kojoj se pronađe put povećanja toka potrebno je na tom putu povećati tok u rezidualnoj mreži, što zahtjeva $O(n)$ koraka. Potrebno je još odrediti koliko je puta moguće pronaći neki put povećanja toka. Kako su svi kapaciteti odozgo omeđeni s U, očito kapacitet bilo kojeg $s-t$ reza može biti najviše nU, pa je i maksimalni tok, zbog (1.5), omeđen odozgo s nU. U svakoj iteraciji Ford-Fulkersonovog algoritma u kojoj ponor bude označen tok će se povećati barem za jedan. Iz svega navedenog slijedi da je složenost Ford-Fulkersonovog algoritma $O(nmU)$.

1.6 Primjene maksimalnog toka

Na kraju ovog poglavlja ukratko ćemo opisati neke situacije gdje se ideja maksimalnog toka može primijeniti.

Traženje dopustivog toka

Problem dopustivog toka zahtijeva da u mreži $G = (V, E)$ pronađemo tok x koji zadovoljava sljedeće uvjete:

$$\begin{gathered}\sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = b(i), \text{ za } i \in V \\ 0 \leq x_{ij} \leq u_{ij}, \text{ za sve } (i, j) \in E.\end{gathered} \tag{1.7}$$

Kao što smo i prije naveli, vrijednost $b(i)$ predstavlja zahtjev pojedinog vrha u mreži, a pretpostavljamo da vrijedi $\sum_{i\in V} b(i) = 0$.

Primjer gdje se problem dopustivog toka može primjeniti u praksi je problem transporta robe između proizvođača i potrošača. U tom problemu, poznata je količina robe koju određeni proizvođač može proizvesti, količina robe koju neki potrošač može zahtijevati i maksimalna količina robe koja se može prevesti određenim putem između proizvođača i potrošača. Potrebno je odrediti može li se roba prevesti na način da svi uvjeti budu zadovoljeni.

Kako bismo problem transporta modelirali pomoću problema traženja dopustivog toka, proizvođače i potrošače prikažimo kao vrhove u mreži. Svaki proizvođač i ima suficit robe; njegov odgovarajući $b(i)$ je veći od nule i po vrijednosti odgovara količini robe koju taj proizvođač može proizvesti. S druge strane, svaki potrošač j ima deficit robe; njegov $b(j)$ je manji od nule i po vrijednosti odgovara količini robe koju taj potrošač zahtijeva. Brid između proizvođača i i potrošača j postoji ako postoji transportni put između tog proizvođača i potrošača i ima kapacitet u_{ij}. Cilj je pronaći tok $x = \{x_{ij}\}$ u mreži tako da su zadovoljeni uvjeti (1.7).

Rješavanje problema dopustivog toka može se svesti na problem traženja maksimalnog toka u mreži. U odnosu na početnu mrežu, uvedimo dva dodatna vrha koje zovemo izvor s i ponor t. Za svaki vrh i za koji je $b(i) > 0$ mreži dodajmo bridove (s, i) kapaciteta $b(i)$. Nadalje, za svaki vrh i za koji je $b(i) < 0$ mreži dodajmo bridove (i, t) kapaciteta $-b(i)$. Ovako definirana mreža često se naziva i *transformirana mreža*. Ukoliko rješavanjem problema maksimalnog toka od izvora s do ponora t u transformiranoj mreži dobijemo rješenje u kojem su bridovi od izvora te bridovi prema ponoru zasićeni (tj. vrijednost toka u njima jednaka je njihovom kapacitetu), onda i problem traženja dopustivog toka ima rješenje. U suprotnom, rješenje ne postoji.

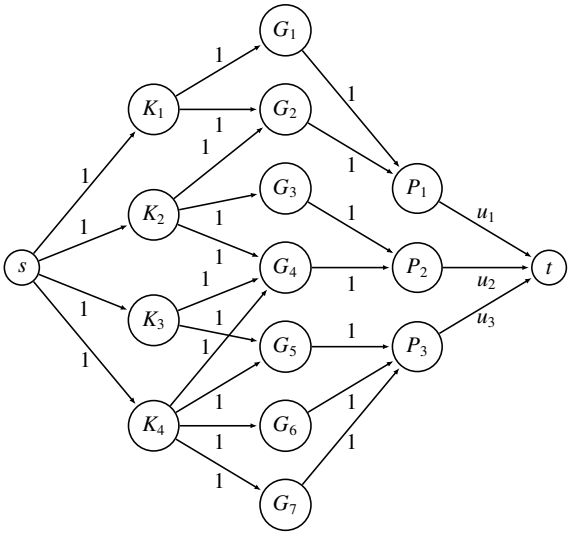
Pokažimo ukratko da su ta dva problema zaista ekvivalentna. Ako tok x zadovoljava (1.7), onda je isti tok uz $x_{si} = b(i)$ za svaki brid od izvora s do vrha i za koji je $b(i) > 0$ te $x_{it} = -b(i)$ za svaki brid od vrha i za koji je $b(i) < 0$ do ponora t, maksimalan u transformiranoj mreži (jer su svi bridovi koji izlaze iz izvora zasićeni). S druge strane, neka je x maksimalan tok u transformiranoj mreži gdje su svi bridovi od izvora i prema ponoru punog kapaciteta. Uočimo da tada za svaki vrh $i \in V$ vrijedi da je lijeva strana jednadžbe (1.7) jednaka nula. Maknemo li vrhove izvora i ponora te bridove koji iz njih izlaze i ulaze, očito vrijednost izraza $\sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji}$ poraste za $b(i)$ ako je izvor bio povezan s i, a smanji se za $b(i)$ ako je i bio povezan s ponorom. Zaključujemo da je tada zadovoljena jednadžba (1.7).

Problem gradskog vijeća

Pretpostavimo da u nekom gradu ima g građana $G_1, G_2, \ldots, G_g$, k klubova $K_1, K_2, \ldots, K_k$ i p političkih stranaka $P_1, P_2, \ldots, P_p$. Svaki građanin je član najmanje jednog kluba i može pripadati najviše jednoj stranci. Svaki klub nominira jednog od svojih članova u gradsko vijeće, ali tako da je broj članova u vijeću koji pripadaju stranci P_i najviše u_i. Problem je odrediti može li se vijeće sastaviti tako da budu zadovoljeni svi uvjeti.

Prikažimo primjerom kako se ovaj problem može svesti na problem određivanja maksimalnog toka. Neka je $g = 7, k = 4, p = 3$ i neka su sve pripadnosti građana klubovima i strankama prikazane na sljedećem grafu. Graf sadrži dva specijalna vrha, izvor s i ponor t, te čvorove $G_1, \ldots, G_7, K_1, \ldots, K_4, P_1, \ldots, P_3$. Također, pojavljuju se bridovi (s, K_i) za svaki klub K_i, (K_i, G_j) ako građanin G_j pripada klubu K_i, (G_j, P_k) ako građanin G_j pripada političkoj stranci P_k te bridovi (P_k, t) kapaciteta u_k za svaku političku stranku P_k. Svi ostali bridovi koje smo naveli imaju kapacitet 1.

Sada je potrebno pronaći maksimalan tok u grafu. Ako on iznosi k, tada je moguće sastaviti vijeće; u suprotnom to nije moguće. Dokaz ove tvrdnje može se provesti tako da se pokaže da svaki tok vrijednosti k u ovom grafu daje mogućnost sastavljanja vijeća te tako da se pokaže da ako je moguće sastaviti vijeće, onda postoji tok vrijednosti k.



Slika 1.6: Primjer mreže za problem gradskog vijeća

Poglavlje 2

Maksimalni tok minimalne cijene u mreži

2.1 Uvod

U ovom poglavlju obradit ćemo glavnu temu ovoga rada, pronalazak maksimalnog toka minimalne cijene u mreži. Ova tema kombinira poznate rezultate iz područja najkraćih puteva u mreži, kao i rezultate i metode iz prethodnog poglavlja, gdje smo dali osnovne pojmove i algoritme vezane za problem nalaska maksimalnog toka u mreži. Na početku, razmotrit ćemo nekoliko pitanja koja se prirodno javljaju.

1. Koliko je teže riješiti problem maksimalnog toka minimalne cijene od problema maksimalnog toka?

2. Mogu li se metode koje smo iskoristili kod problema maksimalnog toka primijeniti i na ovaj problem?

3. Mogu li se neki od poznatih algoritama za pronalaženje najkraćih puteva u mreži jednostavno prilagoditi ovome problemu?

U sljedećih nekoliko odlomaka uvest ćemo neke pojmove koje će pojedini algoritmi koristiti, kao što su pojam potencijala vrha, reducirane cijene bridova, negativan ciklus u mreži i sl.

Definirajmo sada precizno problem maksimalnog toka minimalne cijene. Neka je ponovno $G = (V, E)$ usmjereni graf, takav da svaki brid $(i, j) \in E$ ima pridružen kapacitet u_{ij}, ali i cijenu prolaska jedne jedinice toka kroz taj brid c_{ij}. Također, za svaki vrh v zadan je iznos $b(v)$ koji predstavlja razliku toka koji se nalazi u vrhu v. Ako je $b(v) > 0$, onda u vrhu v imamo višak (suficit) pa količina toka koji izlazi iz v treba za $b(v)$ biti veća od količine toka koja ulazi. Ako je $b(v) < 0$, onda u vrhu v imamo manjak (deficit) i količina toka koja

ulazi u vrh v treba za $|b(v)|$ biti veća od količine koja izlazi. (Možemo zamišljati da vrhovi v sa suficitom proizvode $b(v)$ jedinica toka, dok vrhovi v s deficitom potražuju $|b(v)|$ jedinica toka.). Potrebno je odrediti tok minimalne cijene tako da zahtjevi vrhova budu balansirani na opisani način.

Problem maksimalnog toka minimalne cijene stoga možemo definirati ovako: potrebno je odrediti tok $x = \{x_{ij}\}$ tako da vrijedi:

$$\min z(x) = \sum_{(i,j)\in E} c_{ij}x_{ij}, \tag{2.1}$$

uz uvjete:

$$\begin{gathered}\sum_{\{j:(i,j)\in E\}} x_{ij} - \sum_{\{j:(j,i)\in E\}} x_{ji} = b(i), \text{ za sve } i \in V, \\ 0 \le x_{ij} \le u_{ij}, \text{za sve } (i,j) \in E.\end{gathered} \tag{2.2}$$

Za tok x koji zadovoljava uvjete (2.2) kažemo da je *dopustivi* tok. Nadalje, za proizvoljni tok x definiramo vrijednosti:

$$\sum_{\{j:(i,j)\in E\}} x_{ij} - \sum_{\{j:(j,i)\in E\}} x_{ji} = b_x(i), \text{ za sve } i \in V. \tag{2.3}$$

Vrijednost $b_x(i)$ možemo promatrati kao trenutnu razliku u toku u vrhu i. Ako je $b_x(i) = b(i)$ za svaki vrh i, onda je tok x dopustiv.

Ponovno, neka postoje neke pretpostavke na mrežu:

- sve vrijednosti (zahtjevi na vrhovima, kapaciteti i cijene) su prirodni brojevi;
- mreža je usmjerena;
- zahtjevi na vrhovima mreže, dakle deficiti i suficiti zadovoljavaju uvjet $\sum_{v\in V} b(v) = 0$;
- mreža ne sadrži paralelne bridove, odnosno, ne postoji više bridova koji direktno povezuju neka dva vrha.

2.2 Potencijali i reducirane cijene

Neki od algoritama koje ćemo kasnije analizirati koriste pojmove potencijala i reduciranih cijena. Svakom vrhu $v \in V$ pridružimo broj $\pi(v)$ koji se zove *potencijal* vrha v. Na temelju potencijala $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, definira se i pojam *reduciranih cijena* bridova kao:

$$c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j). \tag{2.4}$$

Utvrdimo vezu između reduciranih cijena c_{ij}^{π} i stvarnih cijena bridova c_{ij}. Neka je:

$$\begin{aligned} z(\pi) &= \sum_{(i,j)\in E} c_{ij}^{\pi} x_{ij} \\ z(0) &= \sum_{(i,j)\in E} c_{ij} x_{ij}. \end{aligned} \tag{2.5}$$

Pretpostavimo da je $\pi = \vec{0}$ i da se potencijal vrha v postavi na $\pi(v)$. Po definiciji (2.4) očito se reducirana cijena brida koji izlazi iz vrha v smanjuje za $\pi(v)$, dok se reducirana cijena svakog brida koji ulazi u vrh v povećava za $\pi(v)$. Dakle, smanjenje vrijednosti funkcije z iznosi $\pi(v)$ pomnoženo s razlikom toka koji izlazi i ulazi u vrh v.
Nadalje, po definiciji (2.2) znamo da razlika toka koji izlazi i ulazi u vrh v mora biti upravo $b(v)$. Time zaključujemo da porastom potencijala vrha v za neki iznos $\pi(v)$ dolazi do smanjenja vrijednosti minimizacijske funkcije za $\pi(v)b(v)$. Ukoliko ovaj postupak ponovimo za svaki vrh $v \in V$, dobivamo da vrijedi:

$$z(0) - z(\pi) = \sum_{v\in V} \pi(v)b(v) = \pi b. \tag{2.6}$$

Sa πb označili smo skalarni produkt vektora π i b. Za dani potencijal, πb je konstanta, tj. ne ovisi o toku x. Stoga, ako neki tok minimizira $z(\pi)$, on minimizira i $z(0)$. Time smo dokazali sljedeću propoziciju.

Propozicija 2.2.1. *Problemi maksimalnog toka minimalne cijene s cijenama c_{ij}^{π} i c_{ij} imaju isto optimalno rješenje. Pritom vrijedi:*

$$z(\pi) = z(0) - \pi b. \tag{2.7}$$

Dokažimo još sljedeća dva bitna svojstva vezana za potencijale i reducirane cijene.

Propozicija 2.2.2.

(a) Za proizvoljni usmjereni ciklus C i potencijale π vrijedi:

$$\sum_{(i,j)\in C} c_{ij}^{\pi} = \sum_{(i,j)\in C} c_{ij}.$$

(b) Za proizvoljni usmjereni put P od vrha k do l i potencijale π vrijedi:

$$\sum_{(i,j)\in P} c_{ij}^{\pi} = \sum_{(i,j)\in P} c_{ij} - \pi(k) + \pi(l).$$

Dokaz.

(a) Koristeći definiciju (2.4) imamo redom:

$$\begin{aligned}\sum_{(i,j)\in C} c_{ij}^{\pi} &= \sum_{(i,j)\in C} (c_{ij} - \pi(i) + \pi(j)) \\ &= \sum_{(i,j)\in C} c_{ij} + \sum_{(i,j)\in C} (\pi(j) - \pi(i))) \\ &= \sum_{(i,j)\in C} c_{ij}.\end{aligned}$$

Posljednja jednakost vrijedi zbog činjenice da se u ciklusu svaki vrh pojavljuje jednom kao početni vrh te jednom kao završni vrh u bridu, pa se svaki vrh pojavi s pozitivnim i negativnim predznakom u sumi.

(b) Slično kao u (a) dijelu imamo:

$$\begin{aligned}\sum_{(i,j)\in P} c_{ij}^{\pi} &= \sum_{(i,j)\in P} (c_{ij} - \pi(i) + \pi(j)) \\ &= \sum_{(i,j)\in P} c_{ij} - \sum_{(i,j)\in C} (\pi(i) - \pi(j))) \\ &= \sum_{(i,j)\in P} c_{ij} - \pi(k) + \pi(l).\end{aligned}$$

Uočimo da za razliku od izraza za ciklus ovdje dolazi do poništavanja vrijednosti potencijala π za sve vrhove u putu osim za početni i završni vrh.

□

2.3 Dekompozicija toka i uvjeti optimalnosti

Kao i u slučaju problema maksimalnog toka, i ovdje ćemo problem rješavati pomoću rezidualne mreže. Dakle, za svaki brid $(i, j) \in E$ rezidualnog kapaciteta $r_{ij} = u_{ij} - x_{ij}$ postoji i brid (j, i) kapaciteta $r_{ji} = x_{ij}$. Za svaki brid koji postoji u originalnoj mreži, njegova cijena u rezidualnoj mreži je također c_{ij}. Budući se u rezidualnoj mreži slanjem toka obrnutim bridom (j, i) tok u originalnoj mreži poništava, definira se da je cijena tog brida $-c_{ij}$ u rezidualnoj mreži.

Promotrimo sljedeći način definiranja toka u mreži: neka je P bilo koji put u rezidualnoj mreži, te pretpostavimo da svakim bridom tog puta pustimo tok $f(P)$. Slično, neka je C

bilo koji ciklus u rezidualnoj mreži, te neka svakim bridom tog ciklusa prolazi tok $f(C)$. Sada za bilo koji brid (i, j) možemo odrediti tok koji njime prolazi - to je jednostavno suma tokova svih puteva i svih ciklusa koji sadrže taj brid.
Dakle, ako znamo tokove na putevima i ciklusima, znamo i tok na pojedinom bridu. Postavlja se pitanje može li se napraviti i obratno, tj. da iz tokova na bridovima rekonstruiramo tokove na putevima i ciklusima. Ponovimo ukratko da za vrh i kažemo da ima višak u toku x ako je $b_x(i) > 0$ (jer iz njega više toka izlazi nego što ulazi), dok za vrh kažemo da ima manjak u toku x ako je $b_x(i) < 0$.
Dokažimo sljedeći teorem.

Teorem 2.3.1. *Svaki tok x može se prikazati pomoću tokova na putevima i ciklusima tako da:*

(a) Svaki usmjereni put s pozitivnim tokom povezuje vrh s viškom toka s vrhom s manjkom toka.

(b) Najviše $n + m$ puteva i ciklusa ima pozitivan tok, te najviše m ciklusa ima pozitivan tok.

Dokaz. Pokažimo da se svaki tok x može podijeliti na tokove na putevima i ciklusima. Neka je v_0 suficitni vrh, tj. vrh koji ima višak u toku. Tada postoji neki brid (v_0, v_1) koji ima pozitivan tok. Ako je v_1 deficitni vrh, pronašli smo put s početkom u suficitnom, a krajem u deficitnom vrhu, pa smo prvi dio teorema dokazali. U suprotnom, iz (2.3) slijedi da postoji neki vrh v_2 tako da brid (v_1, v_2) ima pozitivan tok. Analogno nastavimo dalje. U nekom trenutku ili ćemo pronaći deficitni vrh ili ćemo zatvoriti ciklus. Budući da je broj vrhova n, ovaj postupak će sigurno završiti u n koraka. U slučaju da se radi o nekom putu P gdje je završni vrh v_k, definiramo $f(P) = min\{b_x(v_0), -b_x(v_k), min\{x_{ij} : (i, j) \in P\}\}$, te postavimo $b_x(v_0) = b_x(v_0) - f(P)$, $b_x(v_k) = b_x(v_k) + f(P)$, $x_{ij} = x_{ij} - f(P)$. Ako se radilo o ciklusu C, onda definiramo $f(C) = min\{x_{ij} : (i, j) \in C\}$ te postavimo $x_{ij} = x_{ij} - f(C)$.

Ovaj postupak ponavljamo dok zahtjevi svih vrhova ne postanu nula. Nakon toga, pronalazimo vrh koji ima barem jedan izlazeći brid s pozitivnim tokom, te nastavljajući dalje sa sljedećim vrhom, zatvorimo ciklus te na tom ciklusu ponovnom poništimo tok. S postupkom prestajemo kada vektor x postane nula. Na ovaj način očito cijeli tok x možemo prikazati pomoću tokova na putevima i ciklusima. Uočimo nadalje da smo prilikom pronalaska puteva svaki puta zahtjev nekog vrha sveli na nulu ili smo tok na nekom bridu sveli na nulu. Slično, prilikom pronalaska ciklusa tok na barem jednom bridu je postavljen na nulu. Time smo dokazali i da je moguće pronaći najviše $n + m$ usmjerenih puteva i ciklusa te najviše m ciklusa jer je broj bridova u mreži m. □

Uočimo bitnu posljedicu ovog teorema: ako u nekoj mreži svi vrhovi imaju zahtjeve nula, tj. nemaju niti deficit niti suficit, tada se tok na toj mreži može prikazati samo pomoću najviše m usmjerenih ciklusa.

Sada smo spremni dokazati važan rezultat za problem maksimalnog toka minimalne cijene.

Teorem 2.3.2. *Dopustivo rješenje x^* je optimalno rješenje problema maksimalnog toka minimalne cijene ako i samo ako rezidualna mreža $R(x^*)$ ne sadrži ciklus negativne cijene (odnosno, ciklus u kojem je suma cijena svih bridova negativna).*

Dokaz. Pretpostavimo da je x dopustivi tok i neka $R(x)$ sadrži negativni ciklus. Tada očito x ne može biti optimalno rješenje jer slanjem dodatnog toka kroz taj ciklus funkcija z poprima manju vrijednost. Dakle, ako je x^* optimalan tok, $R(x^*)$ ne sadrži negativni ciklus. Pretpostavimo sada da je x dopustivi tok i neka $R(x)$ ne sadrži negativni ciklus. Također, neka je x^* optimalan tok takav da je $x \neq x^*$. Prikažimo tok x^* u rezidualnoj mreži $R(x)$ dodavanjem toka x' . Definirajmo tok x' na sljedeći način:

$$\begin{aligned} x'_{ij} - x'_{ji} &= x^*_{ij} - x_{ij} \\ x'_{ij}x'_{ji} &= 0. \end{aligned} \tag{2.8}$$

Iz drugog uvjeta (2.8) slijedi da x'_{ij} i x'_{ji} ne mogu oba biti pozitivni u istom trenutku. Ako je $x^*_{ij} \geq x_{ij}$ definiramo $x'_{ij} = x^*_{ij} - x_{ij}$, $x'_{ji} = 0$. Uočimo, budući je $x^*_{ij} \leq u_{ij}$, to je $x'_{ij} \leq u_{ij} - x_{ij} = r_{ij}$. Zaključujemo da se količina toka x'_{ij} može dodati toku x_{ij}, a da kapacitet brida ne bude premašen. S druge strane, ako je $x^*_{ij} < x_{ij}$, definiramo $x'_{ij} = 0$, $x'_{ji} = x_{ij} - x^*_{ij}$. Zbog $x'_{ji} \leq x_{ij} = r_{ji}$, količina toka x'_{ji} se može dodati toku x_{ji}. Budući su i tok x i tok x^* dopustivi, tj. zadovoljavaju uvjet (2.1), iz konstrukcije toka x' vrijedi da, ako bismo promatrali samo taj tok u mreži, svi vrhovi imaju balans u toku, tj. količina toka koji izlazi iz svakog vrha jednaka je količini toka koji ulazi u vrh. Nakon dokaza teorema 2.3.1 zaključili smo da se takav tok može prikazati pomoću najviše m usmjerenih ciklusa. Budući da $R(x)$ ne sadrži negativne cikluse, a cijena toka x' je $cx^* - cx$, očito je $cx^* - cx \geq 0$, tj. $cx^* \geq cx$. S druge strane, kako je x^* optimalan tok, $cx^* \leq cx$. Dakle, $cx^* = cx$. Iz ovoga slijedi da je i x optimalan tok. Ovime smo dokazali da ako je x dopustiv tok i $R(x)$ ne sadrži negativni ciklus, onda je x optimalan.

□

Sljedeći teorem daje još jedan uvjet po kojem možemo prepoznati optimalnost rješenja našeg problema.

Teorem 2.3.3. *Dopustivo rješenje x^* je optimalno rješenje problema maksimalnog toka minimalne cijene u mreži ako i samo ako za neki niz potencijala π vrijedi sljedeći uvjet:*

$$c^\pi_{ij} \geq 0, \text{ za svaki brid } (i, j) \in R(x^*). \tag{2.9}$$

Dokaz. Teorem ćemo dokazati tako da pokažemo ekvivalentnost s teoremom 2.3.2. Neka je x^* tok koji zadovoljava (2.9). Pokažimo da tada ne postoji negativni ciklus u mreži.

Zaista, po pretpostavci je tada $\sum_{(i,j)\in C} c_{ij}^{\pi} \geq 0$, za svaki usmjereni ciklus $C \in R(x^*)$. Po (a) dijelu propozicije 2.2.2, $\sum_{(i,j)\in C} c_{ij}^{\pi} = \sum_{(i,j)\in C} c_{ij} \geq 0$, pa $R(x^*)$ ne sadrži negativne cikluse.

Pretpostavimo sada da je dopustivi tok x^* ujedno i optimalan, tj. $R(x^*)$ ne sadrži negativne cikluse te neka $d(v)$ predstavlja najkraću udaljenost od nekog početnog čvora do čvora v u $R(x^*)$, pri čemu udaljenost vrhova mjerimo koristeći cijene c_{ij}. Budući da mreža ne sadrži negativne cikluse, to očito za svaki vrh vrijedi $d(j) \leq d(i) + c_{ij}$, za svaki brid $(i, j) \in R(x^*)$. Stoga je $c_{ij} - (-d(i)) + (-d(j)) \geq 0$, tj. $c_{ij}^{\pi} \geq 0$, za $\pi = -d$. Dakle, za tok x^* svojstvo (2.9) vrijedi. □

Rezultat prethodnog teorema najčešće ćemo spominjati kao *uvjet optimalnosti reduciranih cijena u odnosu na potencijale* π. Teorem koji slijedi pokazat će povezanost reduciranih cijena i količine toka na bridovima.

Teorem 2.3.4. *Dopustivo rješenje x^* je optimalno rješenje problema maksimalnog toka minimalne cijene u mreži ako i samo ako postoje potencijali π takvi da vrijede sljedeći uvjeti na tok i reducirane cijene:*

(a) Ako je $c_{ij}^{\pi} > 0$, onda je $x_{ij}^* = 0$.

(b) Ako je $0 < x_{ij}^* < u_{ij}$, onda je $c_{ij}^{\pi} = 0$.

(c) Ako je $c_{ij}^{\pi} < 0$, onda je $x_{ij}^* = u_{ij}$.

Dokaz.

(a) Najprije se podsjetimo činjenice koju smo ranije naveli: ako se u originalnoj mreži pojavljuje brid (i, j) koji ima cijenu c_{ij} te se u rezidualnoj mreži pojavi brid (j, i) sa tokom $x_{ji} = x_{ij} > 0$, tada je njegova cijena $-c_{ij}$. Pretpostavimo sada da je $c_{ij}^{\pi} > 0$. Ako bi se u rezidualnoj mreži pojavio brid (j, i) s tokom $x_{ji}^* > 0$, tada bi njegova cijena bila $c_{ji}^{\pi} < 0$. Međutim, to je kontradikcija s teoremom 2.3.3 i pretpostavkom da je x^* optimalan tok. Dakle, tok na bridu (i, j) se ne može poništiti, tj. $x_{ij}^* = 0$.

(b) U slučaju da je $0 < x_{ij}^* < u_{ij}$, rezidualna mreža sadrži bridove (i, j) i (j, i) i tokovi na oba brida su pozitivni. Po teoremu 2.3.3, $c_{ij}^{\pi} \geq 0$ i $c_{ji}^{\pi} \geq 0$. No, $c_{ij}^{\pi} = -c_{ji}^{\pi}$. To je jedino moguće ako je $c_{ij}^{\pi} = 0$.

(c) Uočimo da u slučaju $c_{ij}^{\pi} < 0$ i postojanju brida (i, j) u rezidualnoj mreži dobivamo kontradikciju s teoremom 2.3.3. Dakle, taj brid se ne pojavljuje u rezidualnoj mreži, iz čega zaključujemo da tok na tom bridu mora biti maksimalan, tj. $x_{ij}^* = u_{ij}$.

□

2.4 Algoritmi

U ovom odlomku prikazat ćemo tri različita algoritma koji rješavaju problem maksimalnog toka minimalne cijene. Za svakog od njih bit će opisana ideja, primjer na kojem se može vidjeti uporaba tog algoritma te analiza složenosti.

Algoritam s poništavanjem negativnih ciklusa

Teorem 2.3.2 poslužit će kao temelj za prvi algoritam koji rješava dani problem, a koji smo nazvali algoritam s poništavanjem negativnih ciklusa. Algoritam je osmislio Morton Klein i opisao ga u radu [11]. Ideja algoritma je da, krenuvši od nekog dopustivog rješenja, u svakoj iteraciji pokuša popraviti funkciju cilja z. Algoritam najprije pronađe neki dopustivi tok pomoću proizvoljnog algoritma za pronalazak maksimalnog toka u mreži, primjerice, Ford-Fulkersonovim algoritmom primjenjenim na transformiranu mrežu (dakle mrežu s dodanim izvorom i ponorom) kako je objašnjeno u odlomku 1.6. U svakoj iteraciji cilj je pronaći neki negativni ciklus u rezidualnoj mreži. Ako on postoji, rješenje nije optimalno po prethodno navedenom teoremu. Tada je moguće poslati određenu količinu toka kroz taj ciklus i na taj način popraviti trenutnu vrijednost funkcije z. Nakon toga postupak pronalaženja novog negativnog ciklusa se ponavlja. U slučaju da nije moguće pronaći novi negativni ciklus, teorem 2.3.2 jamči da smo pronašli optimalni tok i algoritam staje.

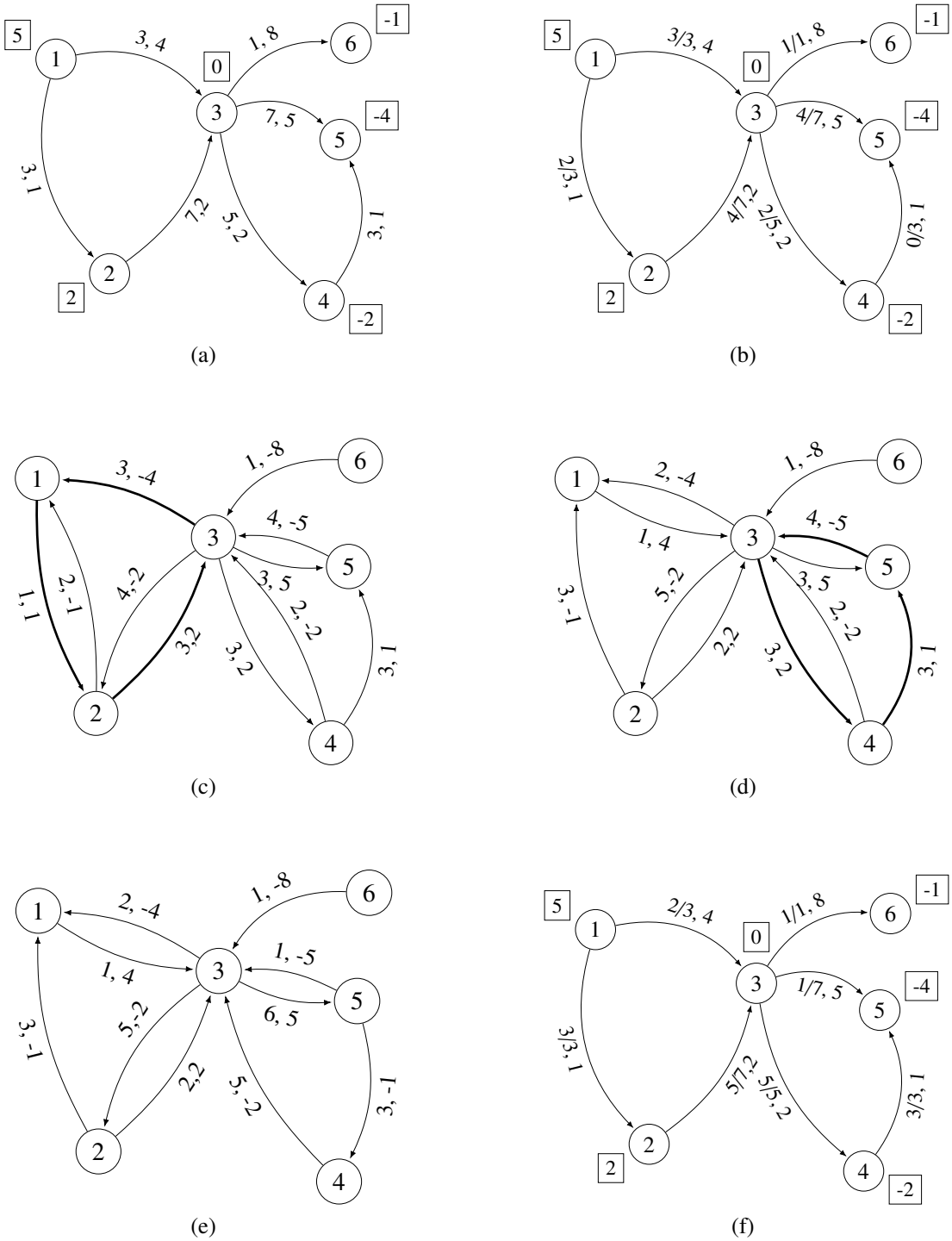
Algorithm 3 Algoritam s poništavanjem negativnih ciklusa

1: pronađi dopustivi tok x u mreži i na temelju njega izgradi rezidualnu mrežu $R(x)$
2: **while** $R(x)$ sadrži negativni ciklus **do**
3: identificiraj negativni ciklus C u mreži $R(x)$
4: $\delta = \min\{r_{ij} : (i, j) \in C\}$
5: dodaj δ jedinica toka u ciklusu C
6: ažuriraj rezidualnu mrežu $R(x)$
7: **end while**

Jedna od pretpostavki koje koristimo pri ovom algoritmu je da su svi kapaciteti i zahtjevi na vrhovima cijeli brojevi (u slučaju racionalnih brojeva možemo ih svesti na cijele, kao što smo napravili i u razmatranju kod maksimalnog toka).

Pokažimo na jednostavnom primjeru rad ovog algoritma. Na slici 2.1a prikazana je originalna mreža. Pokraj svakog vrha v napisana je vrijednost $b(v)$, odnosno, razlika toka koji treba izlaziti i ulaziti iz tog vrha, dok su kapacitet i cijena slanja jedinice toka napisani na svakom od bridova na mreži.
Prvi korak ovog algoritma zahtjeva pronalazak nekog dopustivog toka u mreži, što se može postići pronalaskom maksimalnog toka pomoću Ford-Fulkersonovog algoritma. Tokovi na

pojedinim bridovima koji se na taj način pronađu prikazani su na slici 2.1b Uočimo da smo trenutno postigli rješenje vrijednosti $\sum_{(i,j)\in E} c_{ij}x_{ij} = 1 \cdot 2 + 4 \cdot 3 + 2 \cdot 4 + \ldots + 1 \cdot 0 = 54$. Na slici 2.1c prikazana je pripadna rezidualna mreža. U svakoj iteraciji algoritma pokušava se pronaći negativan ciklus u rezidualnoj mreži. Uporabom, primjerice, Bellman-Fordovog algoritma moguće je pronaći ciklus $1-2-3-1$ cijene $1+2+(-4) = -1$. Njegov kapacitet je $min\{r_{12}, r_{23}, r_{31}\} = min\{1, 3, 3\} = 1$. Rezidualna mreža nakon povećanja toka za jednu jedinicu na tom ciklusu prikazana je na slici 2.1d. Trenutna vrijednost funkcije z je 53.
U drugoj iteraciji pronalazi se negativan ciklus $3-4-5-3$ cijene $2+1+(-5) = -2$ čiji je kapacitet $min\{r_{34}, r_{45}, r_{53}\} = min\{3, 3, 4\} = 3$. Povećanjem toka za tri jedinice toka u tom ciklusu dobivamo stanje rezidualne mreže kao što je vidljivo na slici 2.1e.
Nakon toga, nije moguće pronaći sljedeći negativan ciklus u rezidualnoj mreži te algoritam završava. Rekonstrukcijom iz rezidualne mreže dobivamo tokove na originalnoj mreži prikazane na slici 2.1f. Zaključujemo da je optimalna vrijednost funkcije cilja z jednaka $1 \cdot 3 + 4 \cdot 2 + 2 \cdot 5 + \ldots + 1 \cdot 3 = 47$.



Slika 2.1: Primjer uporabe algoritma s poništavanjem negativnih ciklusa

Analizirajmo složenost algoritma s poništavanjem negativnih ciklusa. Kao i do sada, neka n označava broj vrhova, a m broj bridova u originalnoj mreži te neka je sa U označen maksimalan kapacitet od svih bridova i sa C najveća cijena koja se pojavljuje za prijenos jedinice toka na nekom od bridova. Uočimo da je funkcija cilja z odozgo omeđena s mUC. Dopustivi tok koji se zahtjeva na početku algoritma može se pronaći u vremenu $O(nmU)$ pomoću Ford-Fulkersonovog algoritma. U svakoj iteraciji u kojoj se pronađe negativan ciklus D, njegov kapacitet δ bit će barem 1, pa će promjena funkcije cilja biti za $(\sum_{(i,j)\in D} c_{ij})\delta$. Zaključujemo da se vrijednost funkcije uvijek smanjuje jer se radi o negativnom ciklusu. Budući su po pretpostavci sve vrijednosti cijeli brojevi, algoritam će završiti u $O(mUC)$ iteracija. U svakoj od iteracija potrebno je i utvrditi postojanje negativnog ciklusa, što se Bellman-Fordovim algoritmom može napraviti u vremenskoj složenosti $O(nm)$. Dakle, složenost ovog algoritma je $O(nm^2UC)$.

Algoritam s traženjem najkraćih puteva

Za razliku od algoritma s poništavanjem negativnih ciklusa koji je u svakoj iteraciji održavao dopustivost rješenja i težio k optimalnom, algoritam s traženjem najkraćih puteva odmah će pronaći uvjete optimalnosti kako je opisano u teoremu 2.3.3, ali će u svakom koraku popravljati neispravne tokove kako bi se zadovoljili zahtjevi na vrhovima. Algoritam su nezavisno pronašli autori Masao Iri [8], William S. Jewell [9] te Robert G. Busacker i Paul J. Gowen [5].

Definirajmo pojam *balansa* vrha i kao:

$$e(i) = b(i) - b_x(i) \tag{2.10}$$

Ako je $e(i) > 0$ kažemo da je $e(i)$ višak vrha i, a ako je $e(i) < 0$ kažemo da je manjak tog vrha $|e(i)|$. Ako je, pak, $e(i) = 0$, kažemo da je vrh i balansiran. Neka je S skup svih vrhova s viškom te D skup svih vrhova s manjkom toka. Očito mora vrijediti $\sum_{i\in S} e(i) = -\sum_{i\in D} e(i)$ kako bi problem imao rješenje. Također, jasno je da ako postoji vrh s viškom mora postojati i vrh s manjkom toka. Ideja algoritma je u svakoj iteraciji pronaći vrh i s viškom, vrh j s manjkom u toku te poslati maksimalni mogući tok po najkraćem putu od i do j. Iteracije se ponavljaju dok svi vrhovi ne zadovolje svoje zahtjeve, tj. dok god postoji vrh s viškom (a onda i vrh s manjkom) toka.
Dokažimo sada neke bitne rezultate koje ćemo koristiti prilikom ovog algoritma. Pritom napomenimo da će važnu ulogu imati pojam reduciranih cijena kojeg smo definirali ovako (2.4).

Propozicija 2.4.1. *Pretpostavimo da za neki tok x vrijedi da reducirane cijene c^{π}_{ij} zadovoljavaju uvjet* (2.9) *u odnosu na potencijale π. Neka vektor d označava vektor najkraćih*

udaljenosti od nekog vrha s do svakog drugog vrha u rezidualnoj mreži $R(x)$ u odnosu na reducirane cijene c^{π}_{ij}. Tada vrijedi:

(a) Za tok x zadovoljen je uvjet (2.9) *i u odnosu na potencijale $\pi' = \pi - d$.*

(b) Reducirane cijene $c^{\pi'}_{ij}$ su jednake nula na svim bridovima (i, j) na najkraćem putu od vrha s do nekog vrha t.

Dokaz.

(a) Kako vektor d predstavlja vektor najkraćih udaljenosti od vrha s do svakog drugog vrha u rezidualnoj mreži u odnosu na reducirane cijene c^{π}_{ij}, vrijedi:

$$d(j) \leq d(i) + c^{\pi}_{ij}, \text{ za sve } (i, j) \in R(x).$$

Kako je $c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j)$, to je $d(j) \leq d(i) + c_{ij} - \pi(i) + \pi(j)$. Taj izraz možemo zapisati i u obliku $c_{ij} - (\pi(i) - d(i)) + (\pi(j) - d(j)) \geq 0$, tj. $c^{\pi'}_{ij} \geq 0$. Dakle, i uz potencijale π' zadovoljen je uvjet (2.9).

(b) Za svaki brid (i, j) na nekom najkraćem putu vrijedi $d(j) = d(i) + c^{\pi}_{ij}$. Tada uvrštavanjem u jednadžbu $c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j)$ dobivamo da je $c^{\pi'}_{ij} = 0$.

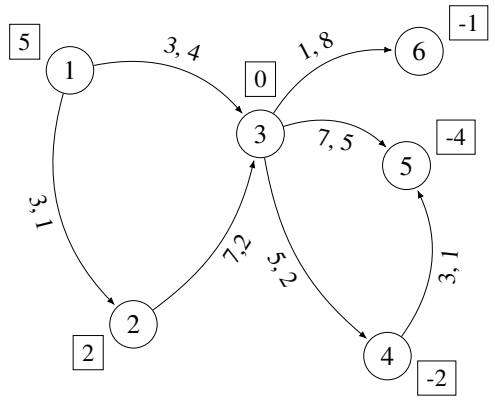
□

Opišimo sada ukratko algoritam. Kako ne bismo morali u svakom koraku tražiti vrhove s viškom ili manjkom toka, dodat ćemo dva posebna vrha u mrežu: vrh s kojeg ćemo zvati *izvor* te vrh t kojeg ćemo zvati *ponor*. Sve vrhove i kod kojih je u početku algoritma $e(i) > 0$ povezat ćemo s izvorom bridom (s, i) čiji je kapacitet $e(i)$, a cijena nula, dok ćemo vrhove i kod kojih je $e(i) < 0$ povezati s ponorom bridom (i, t) kapaciteta $-e(i)$ i cijene nula. Sada ćemo traženje puta od vrha s viškom do vrha s manjkom ekvivalentno zamijeniti s traženjem puta od izvora do ponora. Propozicija 2.4.1 omogućuje da u svakoj iteraciji od vektora potencijala π oduzmemo vektor najkraćih udaljenosti te da je za taj novi vektor potencijala π' zadovoljen uvjet (2.9) (uz pretpostavku da je isti zadovoljen i za potencijale π). Kada put od izvora do ponora više ne bude postojao, iz odlomka 1.6 znamo da će tok biti dopustiv. Budući da će uvjet (2.9) biti stalno zadovoljen, po teoremu 2.3.3 tok će tada biti optimalan.

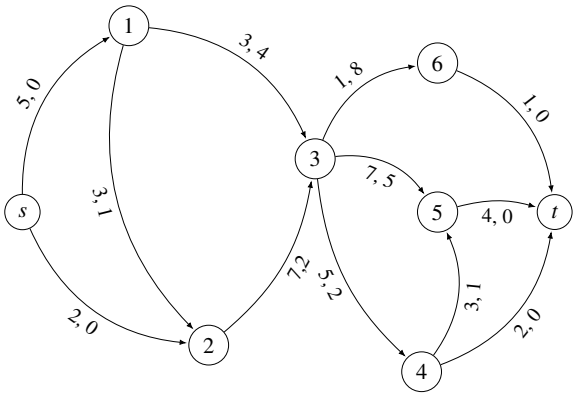
Algorithm 4 Algoritam s traženjem najkraćih puteva

1: $x = 0, \pi = \vec{0}$
2: $e(i) = b(i)$, za svaki $i \in V$
3: uvedi dva nova vrha: izvor s i ponor t
4: vrh s poveži s vrhovima i za koje je $e(i) > 0$ bridom (s, i) kapaciteta $e(i)$ i cijene 0
5: vrh t poveži s vrhovima i za koje je $e(i) < 0$ bridom (i, t) kapaciteta $-e(i)$ i cijene 0
6: $e(s) = \sum_{\{i:e(i)>0\}} e(i)$
7: **while** $e(s) > 0$ **do**
8: Dijkstrinim algoritmom pronađi najkraće udaljenosti d od vrha s do svakog vrha u rezidualnoj mreži u odnosu na reducirane cijene c_{ij}^{π}
9: neka je P najkraći put od s do t
10: $\pi = \pi - d$
11: $\delta = min\{e(s), min\{r_{ij} : (i, j) \in P\}\}$
12: dodaj δ jedinica toka na putu P
13: ažuriraj rezidualnu mrežu i vektor e
14: **end while**

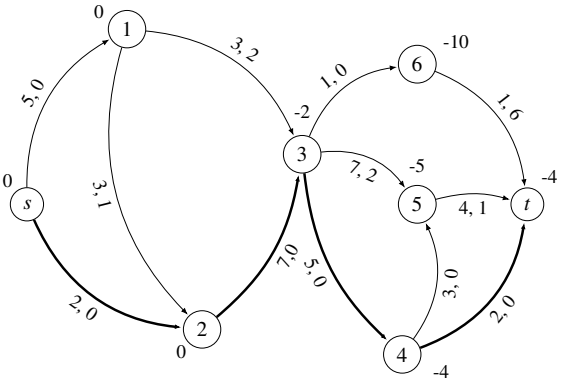
Pogledajmo sada rad algoritma na primjeru mreže koja je bila i u slučaju algoritma s poništavanjem negativnih ciklusa. Na slici 2.2a prikazana je originalna mreža, pri čemu je pokraj svakog vrha napisana tražena razlika u toku koji izlazi i ulazi u vrh, dok su na svakom od bridova zarezom odvojeni kapacitet i cijena slanja jedinice toka po tom bridu. Na slici 2.2b dodani su vrhovi s i t te dodatni bridovi kako smo opisali ranije. Svaki vrh na početku ima potencijal nula. Dijkstrinim algoritmom pronalazimo duljine najkraćih puteva od s do svakog od preostalih vrhova, te dobivamo da je vektor d jednak $(0, 0, 0, 2, 4, 5, 10, 4)$, pri čemu prvi i zadnji element u tom vektoru označavaju duljinu puta do vrhova s i t. Novi potencijali (naznačeni pokraj vrhova), reducirane cijene bridova te pronađeni najkraći put P koji prolazi vrhovima $s, 2, 3, 4, t$ prikazani su na slici 2.2c. Na tom putu P moguće je poslati $\delta = min\{e(s), min\{r_{s2}, r_{23}, r_{34}, r_{4t}\}\} = min\{7, min\{2, 7, 5, 2\}\} = 2$ jedinice toka. Stanje rezidualne mreže nakon povećanja toka prikazano je na slici 2.2d. Budući je $e(s) = 5$, nastavljamo dalje. Novi vektor najkraćih udaljenosti u mreži je $d = (0, 0, 1, 1, 1, 1, 1, 2)$. Promijenjeni potencijali, reducirane cijene, kao i najkraći put od s do t prikazani su na slici 2.2e. Rezidualna mreža nakon povećanja puta od 3 jedinice toka vidljiva je na slici 2.2f. $e(s) > 0$ pa postoji sljedeći najkraći put od s do t. Vektor d sada iznosi: $d = (0, 0, 1, 1, 3, 3, 1, 3)$. Potencijali nakon ovog koraka, kao i reducirane cijene te pronađeni najkraći put prikazani su na slici 2.1g. Na putu P koji prolazi vrhovima $s, 1, 3, 5, t$ moguće je povećati tok za 2 jedinice toka, te je mreža nakon povećanja toka prikazana na slici 2.1h. Uočimo da je sada $e(s) = 1$, te će nakon pronalaska idućeg puta algoritam završiti.



(a)

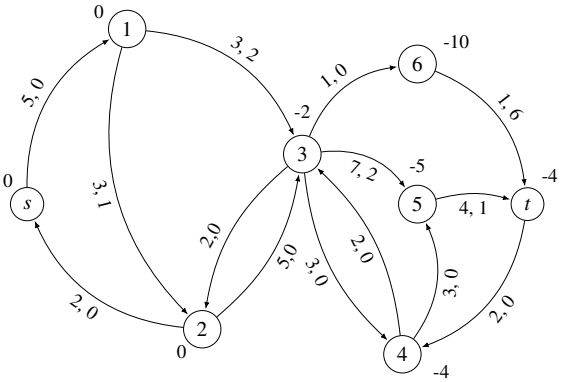


(b)



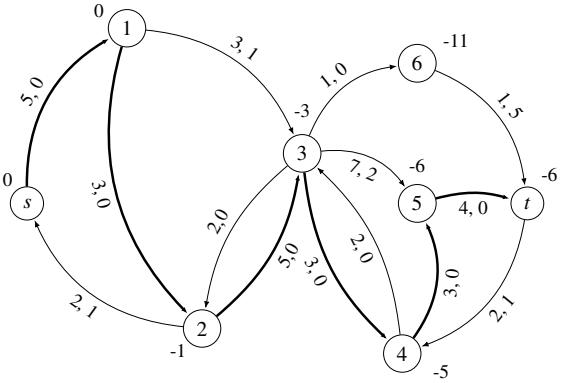
$e(s) = 7$

(c)



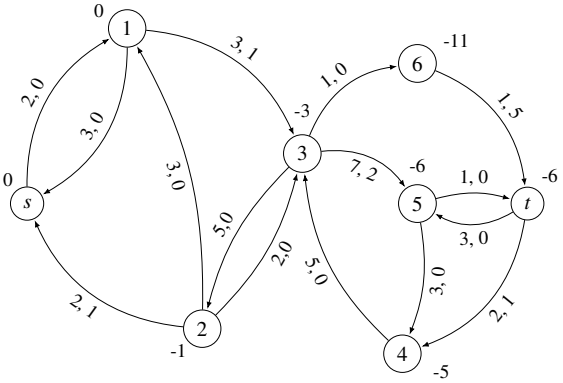
$e(s) = 5$

(d)



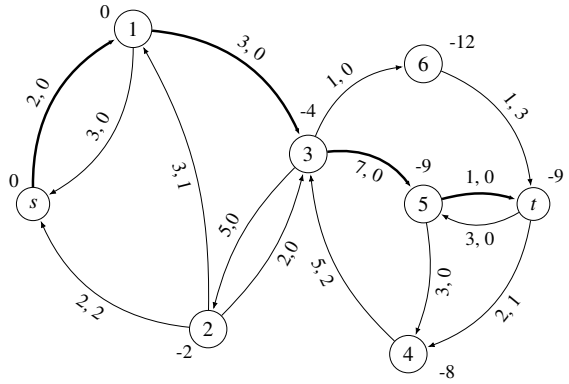
$e(s) = 5$

(e)



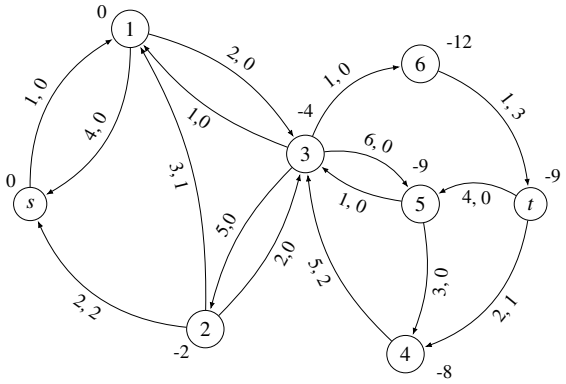
$e(s) = 2$

(f)



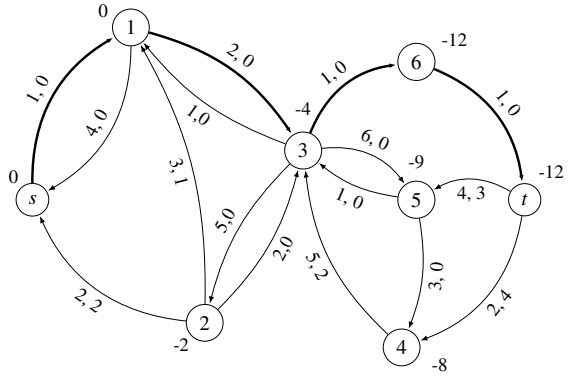
$e(s) = 2$

(g)



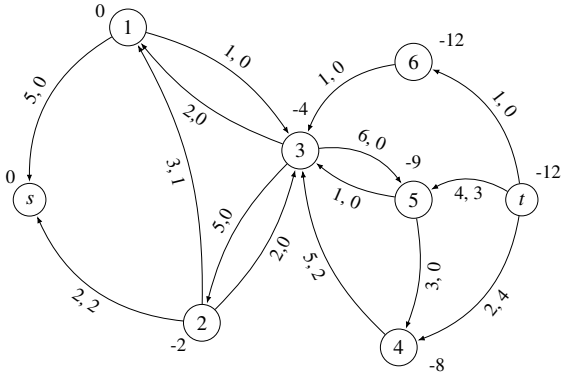
$e(s) = 1$

(h)



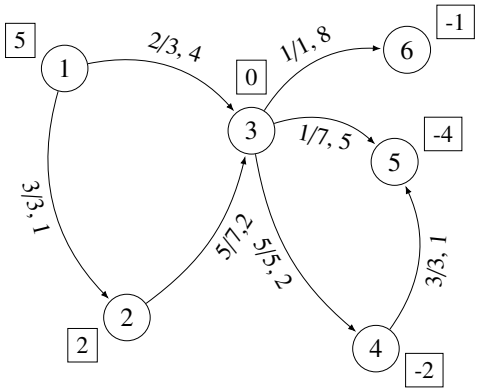
$e(s) = 1$

(i)



$e(s) = 0$

(j)



(k)

Slika 2.1: Primjer uporabe algoritma s traženjem najkraćih puteva

Taj put P sada prolazi vrhovima $s, 1, 3, 6, t$, a pripadni δ je 1. Konačno, nakon slanja toka od δ jedinica toka dolazimo do stanja kakvo je prikazano na slici 2.1j. Zaključujemo da je, budući je $e(s) = 0$, algoritam došao do dopustivog toka i da su svi zahtjevi na vrhovima zadovoljeni. Originalna mreža s pronađenim tokom prikazana je na slici 2.1k i vrijednost funkcije cilja je 47, kao što je bilo i prilikom rješavanja istog problema s algoritmom s poništavanjem negativnih ciklusa.

Analizirajmo još ukratko i složenost algoritma. Svaka iteracija algoritma pronalazi najkraći put od izvora s do ponora t, te na tom putu pronalazi povećanje toka od barem jedne jedinice toka. Ako sa K označimo najveći zahtjev (suficit ili deficit) nekog vrha, n broj vrhova te m broj bridova u mreži, onda zaključujemo da je maksimalan mogući broj iteracija algoritma jednak nK. Nadalje, u svakoj iteraciji potrebno je i pronaći najkraći put od izvora do ponora, što je moguće u složenosti $O(m \log n)$ uz uporabu Dijkstrinog algoritma koji koristi neku strukturu koja ima logaritamsko vrijeme dohvaćanja najmanjeg elementa, kao što je, primjerice, struktura *set* u standardnoj biblioteci programskog jezika C++. Time zaključujemo da je vremenska složenost ovog algoritma $O(nmK \log n)$.

Primal-dual algoritam

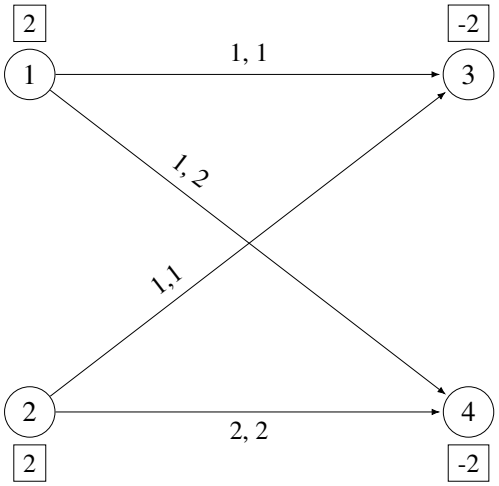
Primal-dual algoritam sličan je algoritmu s traženjem najkraćih puteva po tome što također održava zadovoljenost uvjeta (2.9) s tokom kojeg postepeno dovodi do dopustivog toka. Međutim, umjesto slanja maksimalnog toka po jednom pronađenom najkraćem putu, tok se šalje po svim najkraćim putevima koji trenutno postoje od izvora do ponora. Algoritam su prvi opisali L. R. Ford i D. R. Fulkerson u radu [6].

Kao i u prethodnom algoritmu, uvest ćemo dva dodatna vrha, izvor s i ponor t te dodatne bridove kako je prije bilo opisano. Uz rezidualnu mrežu, promatrat ćemo i tzv. *nul-mrežu* koja je podmreža rezidualne mreže i koju označavamo s $R_0(x)$ u odnosu na tok x. Nul-mreža sadrži samo one bridove u rezidualnoj mreži koji imaju reducirane cijene jednake nula u odnosu na neki potencijal π. Rezidualni kapaciteti u nul-mreži su jednaki kao u rezidualnoj mreži. U svakoj iteraciji algoritma, nakon pronalaska najkraćih udaljenosti u rezidualnoj mreži ažuriraju se potencijali i reducirane cijene te se tada konstruira nul-mreža. Prema propoziciji 2.4.1, u $R_0(x)$ postoji barem jedan put cijene nula od s do t. Međutim, tih puteva može biti i više. Zato se može odjednom preko više različitih nul-puteva poslati nova količina toka. Ford-Fulkersonovim algoritmom u nul-mreži se pronalaze putevi povećanja toka te će tok kasnije biti dodan u rezidualnoj mreži. Svaki nul-put odgovara jednom putu preko kojeg možemo povećati tok u mreži. Nakon toga, rezidualna mreža se ažurira te se cijeli algoritam ponavlja dok višak toka na izvoru na postane nula. Zaista, ako algoritam ne pronađe neki put od izvora do ponora u rezidualnoj mreži, tok je maksimalan pa i dopustiv prema pokazanom u odlomku 1.6. Zbog ispunjenosti uvjeta (2.9), taj tok je i optimalan.

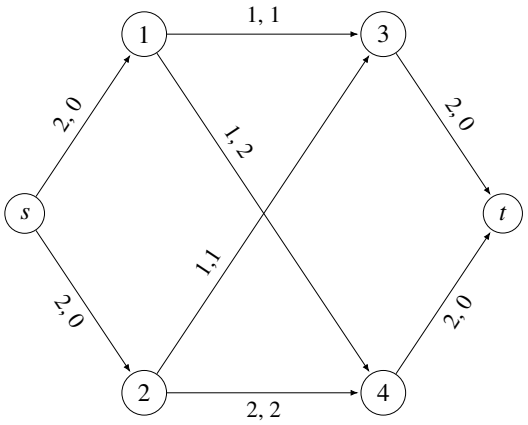
Algorithm 5 Primal-dual algoritam

1: $x = 0, \pi = \vec{0}$
2: $e(i) = b(i)$, za svaki $i \in V$
3: uvedi dva nova vrha: izvor s i ponor t
4: vrh s poveži s vrhovima i za koje je $e(i) > 0$ bridom (s, i) kapaciteta $e(i)$ i cijene 0
5: vrh t poveži s vrhovima i za koje je $e(i) < 0$ bridom (i, t) kapaciteta $-e(i)$ i cijene 0
6: $e(s) = \sum_{\{i:e(i)>0\}} e(i)$
7: **while** $e(s) > 0$ **do**
8: Dijkstrinim algoritmom pronađi najkraće udaljenosti d od vrha s do svakog vrha u rezidualnoj mreži u odnosu na reducirane cijene c^{π}_{ij}
9: $\pi = \pi - d$, ažuriraj reducirane cijene
10: pronađi nul-mrežu $R_0(x)$ u odnosu na nove reducirane cijene c^{π}_{ij}
11: pošalji maksimalni tok od s do t u $R_0(x)$
12: ažuriraj rezidualnu mrežu $R(x)$ i vektor e
13: **end while**

Budući da bi iteracije ovog algoritma na primjeru kojeg smo do sada koristili izgledale potpuno isto kao i prilikom korištenja algoritma s traženjem najkraćih puteva, rad primal-dual algoritma pokazat ćemo na drugom primjeru. Neka je dana mreža koja je prikazana na slici 2.2a. Kao i do sada, uz vrhove su prikazane razlike u toku koji treba izlaziti i ulaziti u vrh, a na bridovima su zarezom odvojeni kapacitet i cijena slanja jedinice toka po tom bridu. Dodavanjem vrhova izvora i ponora te dodatnih bridova koji ih povezuju s mrežom dobivamo situaciju kao na slici 2.2b. Algoritmom za pronalaženje najkraćih udaljenosti od s do svakog od vrhova u mreži dobivamo da je vektor udaljenosti d jednak $(0, 0, 0, 1, 2, 1)$, pri čemu je poredak vrhova $s, 1, 2, 3, 4, t$. Nove reducirane cijene prikazane su uz vrhove na slici 2.2c, dok se pripadna nul-mreža može vidjeti na slici 2.2d. Pritom su na nul-mreži prikazani samo kapaciteti jer su reducirane cijene svih prikazanih bridova nula. Primjetimo da sada postoje dva puta kojima se može povećati tok u mreži: $s - 1 - 3 - t$ te nakon toga $s - 2 - 3 - t$. Time u jednoj iteraciji kroz dva različita puta u nul-mreži možemo ukupni tok povećati za dvije jedinice toka. Na kraju prve iteracije stanje rezidualne mreže prikazano je na slici 2.2e. U sljedećoj iteraciji pronalazimo da je vektor najkraćih udaljenosti $d = (0, 0, 0, 1, 0, 1)$. Nakon ažuriranja potencijala i reduciranih cijena, stanje mreže je kao na slici 2.2f. Nul-mreža trenutne rezidualne mreže vidljiva je na slici 2.1g. Ponovno postoje dva puta povećanja toka: $s-1-4-t$ te $s-2-4-t$; oba dopuštaju povećanje za jednu jedinicu toka. Nakon povećanja toka stanje rezidualne mreže izgleda kao na slici 2.1h. Uočimo da je sada $e(s) = 0$, čime su zahtjevi svih vrhova zadovoljeni pa algoritam staje. Optimalni tok za originalnu mrežu prikazan je na slici 2.1i te je minimalna vrijednost funkcije z jednaka 6.

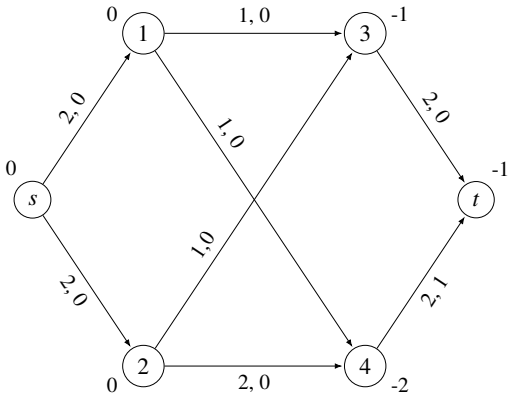


(a)



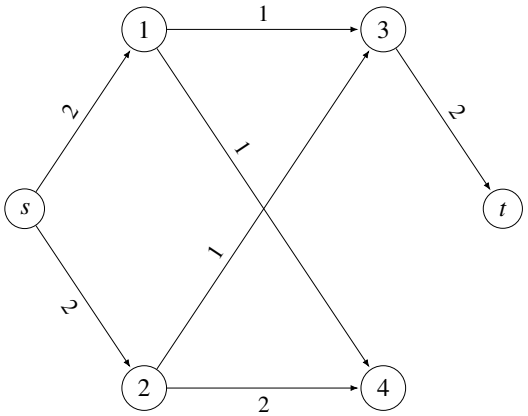
$e(s) = 4$

(b)

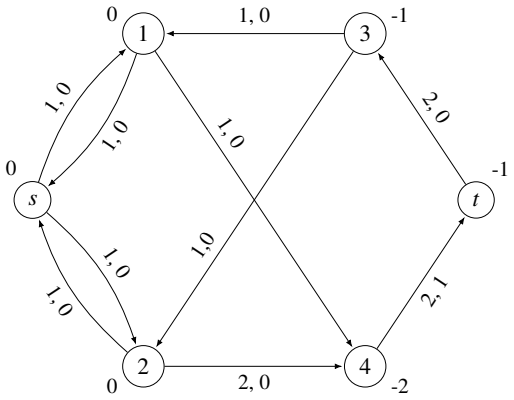


$e(s) = 4$

(c)

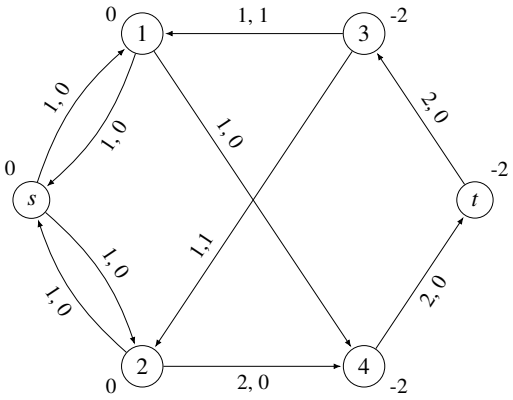


(d)



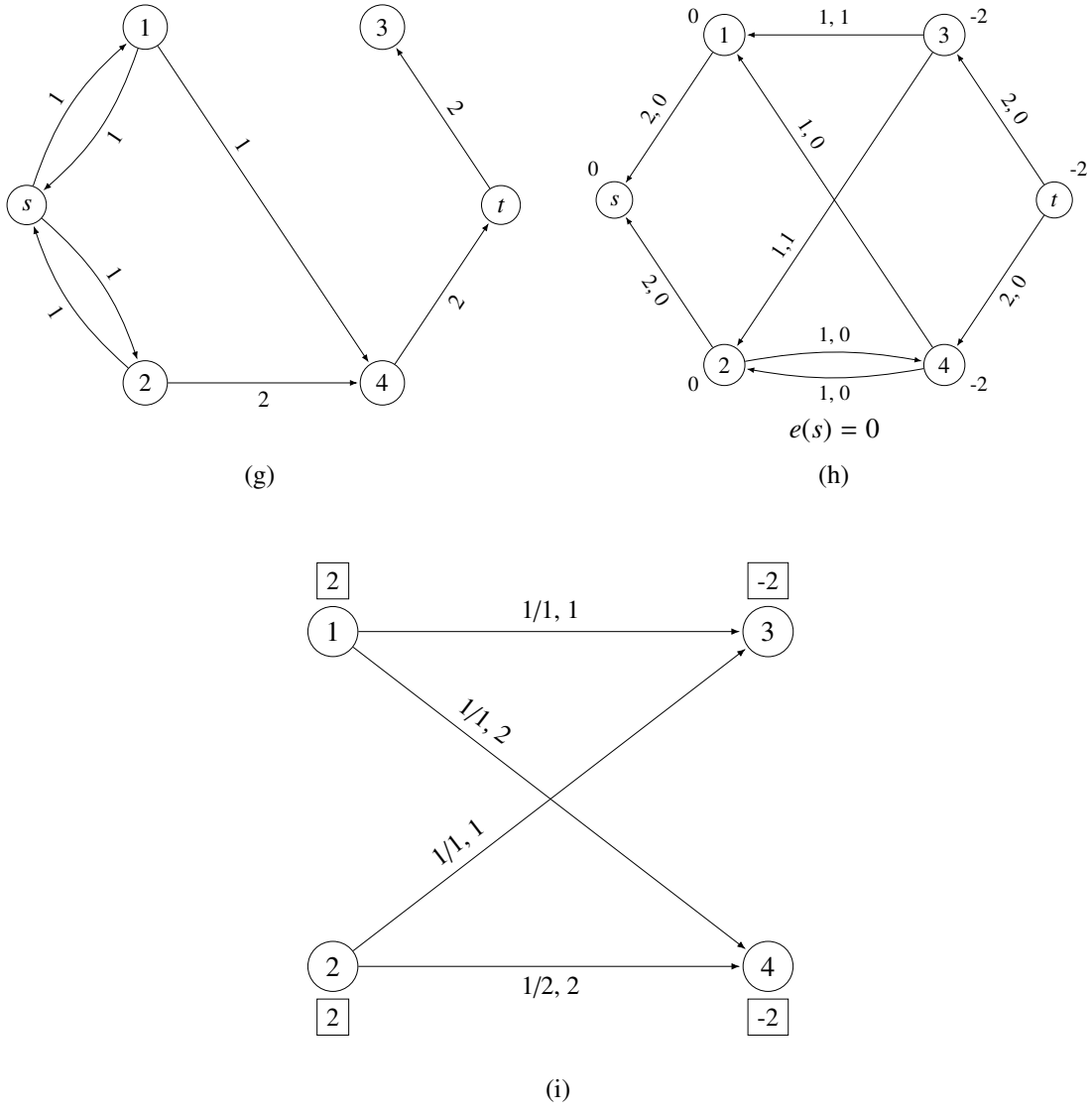
$e(s) = 2$

(e)



$e(s) = 2$

(f)



Slika 2.1: Primjer uporabe primal-dual algoritma

Uočimo da se u svakoj iteraciji algoritma višak izvora smanji barem za jednu jedinicu toka jer se uspije pronaći barem jedan put od izvora do ponora. Također, primjetimo da nakon što se pošalje maksimalna količina toka kroz bridove iz nul-mreže, rezidualna mreža više ne sadrži put od s do t koji bi prolazio isključivo kroz bridove s cijenom nula. To znači da se potencijal ponora u svakom koraku smanji barem za jedan. Ako ponovno sa K označimo po

apsolutnoj vrijednosti najveći zahtjev nekog vrha u originalnoj mreži, sa C najveću cijenu slanja jedinice toka nekim od bridova, zaključujemo da potencijal ponora ne može biti niži od $-nC$. Uz to, višak izvora u svakoj iteraciji smanji se barem za 1, a u početku je ograničen s nK. Time zaključujemo da je broj iteracija ograničen s $min\{nK, nC\}$. U svakoj iteraciji potrebno je pronaći i najkraće puteve od izvora do svakog od vrhova, što je moguće u vremenu $O(m \log n)$. Također, maksimalan tok u nul-mreži može se pronaći Ford-Fulkersonovim algoritmom u vremenskoj složenosti $O(nmU)$, gdje je U najveći kapacitet nekog od bridova. Dakle, složenost primal-dual algoritma je $O(min\{nK, nC\} \cdot (m \log n + nmU))$.

Uočimo na kraju da niti jedan od predložena tri algoritma nema polinomijalnu složenost, tj. složenost ovih algoritama ne ovisi samo o broju vrhova i bridova, već i o kapacitetima i cijenama bridova te zahtjevima vrhova. Ipak, postoje složeniji algoritmi koji imaju polinomijalnu složenost te se mogu pronaći u knjizi [4].

2.5 Primjene problema maksimalnog toka minimalne cijene

U ovom odjeljku opisat ćemo nekoliko situacija gdje se ideja problema maksimalnog toka minimalne cijene može primijeniti.

Problem transporta

Pretpostavimo da u nekoj zemlji postoji n gradova $G_1, \ldots, G_n$, a između nekih od gradova postoje cijevi za transport određene tekućine. Svaka od m postojećih cijevi ima neki kapacitet i cijenu slanja jedne litre tekućine kroz tu cijev. Iz grada G_1 želi se poslati l litara tekućine kroz dane cijevi do grada G_n, ali tako da cijena bude minimalna moguća te da kapacitet niti jedne cijevi ne bude premašen. Poznato je da se između svaka dva para gradova u toj zemlji tekućina može prevesti preko dane mreže cijevi.
Ovo je jedna od najosnovnijih i najpoznatijih primjena problema maksimalnog toka minimalne cijene koja predstavlja tip problema kada imamo samo jedan vrh s viškom, tj. manjkom toka.

Problem dodjeljivanja poslova

Neka je dano n radnika i n poslova koje je potrebno obaviti. Svaki od radnika može obaviti bilo koji posao i pri tome tražiti određenu plaću za taj posao. Svi poslovi moraju biti obavljeni, pri čemu svaki od radnika mora obaviti točno jedan posao i svaki posao smije biti obavljen od strane točno jednog radnika. Zadatak je minimizirati ukupnu cijenu koju

je potrebno izdvojiti za plaće radnika.

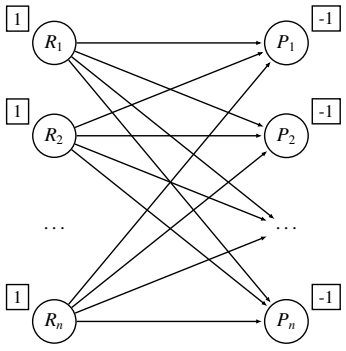
Problem možemo i ovako formulirati:
Neka je dana matrica C dimenzije $n \times n$. Svaki element matrice sadrži jedan prirodan broj, te neka je c_{ij} element na presjeku i-tog retka i j-tog stupca. Taj element predstavlja plaću koju i-ti radnik traži za obavljanje j-tog posla. Zadatak je odabrati točno po jedan broj iz svakog retka i stupca matrice tako da je suma odabranih brojeva minimalna. Točnije, potrebno je riješiti:

$$\min z(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}, \tag{2.11}$$

uz uvjete:

$$\begin{aligned} &\sum_{j=1}^{n} x_{ij} = 1, \text{ za sve } i = 1, \ldots, n, \\ &\sum_{i=1}^{n} x_{ij} = 1, \text{ za sve } j = 1, \ldots, n, \\ &x_{ij} \in \{0, 1\}, \text{ za sve } i, j = 1, \ldots, n. \end{aligned} \tag{2.12}$$

Dopustivo rješenje očito postoji; primjerice, uvijek je moguće odabrati dijagonalne elemente matrice C i svi uvjeti će biti zadovoljeni. Kako bismo pronašli optimalno rješenje, matricu ćemo transformirati u bipartitnu mrežu, kao što je prikazano na slici 2.2. Svaki brid (R_i, P_j) ima kapacitet 1 i cijenu c_{ij}, a zahtjevi pojedinih vrhova su 1, odnosno -1. Budući su sve veze u mreži tipa $R_i - P_j$, nema negativnih ciklusa pa rješenje sigurno postoji i može se pronaći u složenosti $O(n^3 \log n)$ pomoću, primjerice, algoritma s traženjem najkraćih puteva.



Slika 2.2: Mreža za problem dodjeljivanja

Primjerice, za matricu:

$$\begin{pmatrix} 1 & 3 & 2 \\ 2 & 6 & 4 \\ 3 & 7 & 6 \end{pmatrix}$$

potrebno je odabrati broj 3 u prvom retku, 4 u drugom i 3 u trećem retku kako bi suma bila minimalna (10).

Poglavlje 3

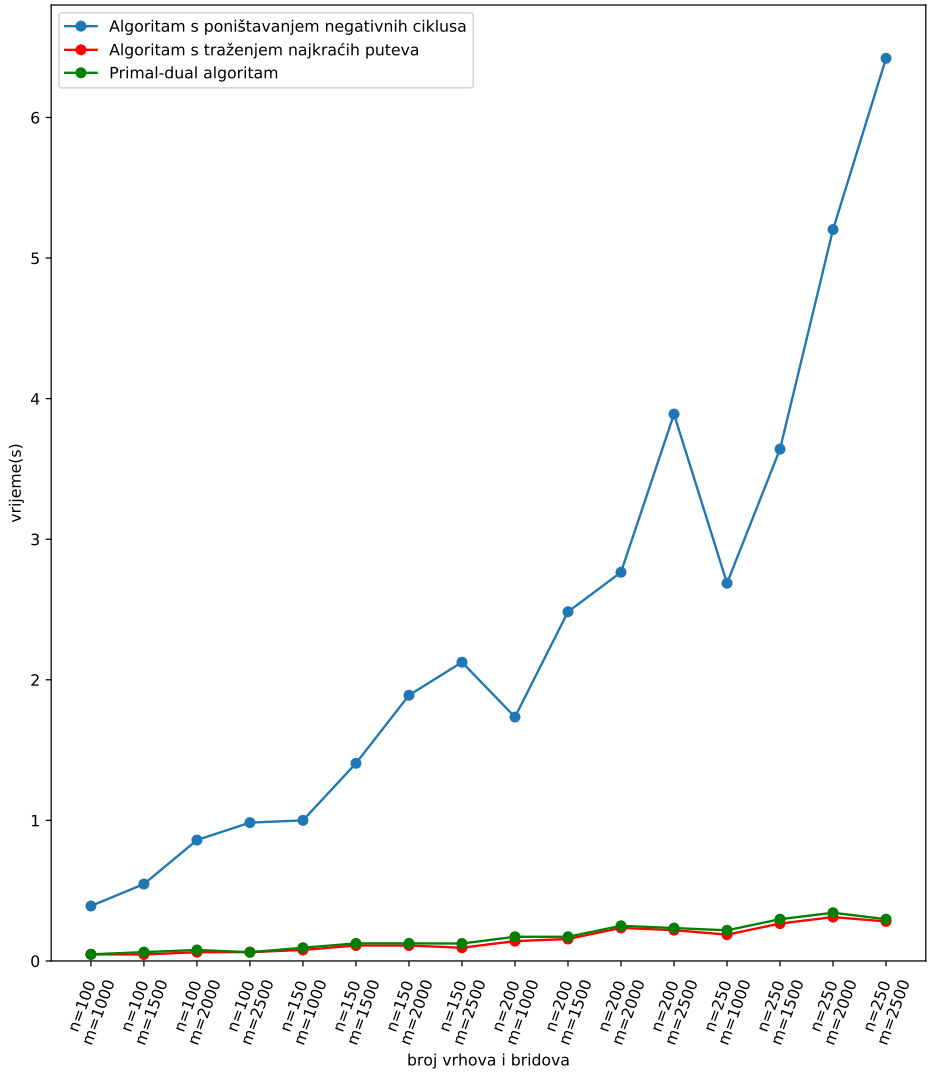
Usporedba performansi algoritama

U ovom poglavlju bit će predstavljeni rezultati testiranja tri prikazana algoritma. Uspješnost algoritama testirali smo na primjerima mreža koje su generirane programom *NETGEN*, a koji je javno dostupan na adresi [1]. Taj program ima mogućnost generiranja različitih tipova mreža u ovisnosti o broju vrhova, bridova, kapacitetima, cijenama, broju vrhova sa suficitom ili deficitom toka te još mnogo toga.

Sve prethodno navedene algoritme implementirali smo u programskom jeziku C++. Glavni razlog odabira ovog jezika bila je uporaba raznih struktura podataka koje su efikasno implementirane u standardnoj biblioteci jezika, kao što su red, skup, vektor i druge. Rezidualna mreža prikazana je na način da za svaki vrh postoji vektor objekata u kojima se čuva oznaka susjeda te udaljenost i cijena pripadnog brida. Također, struktura skup (eng. *set*) omogućila je da Dijkstrin algoritam bude efikasno implementiran, što je dovelo do boljih performansi nekih od algoritama. Implementacije algoritama dostupne su na CD-u priloženom uz ovaj rad.

Prvi kriterij uspješnosti kojeg su na svim generiranim primjerima ispunila sva tri algoritma bila je točnost rješenja. Kako bismo zaista utvrdili da smo dobili ispravno rješenje, tj. vrijednost funkcije cilja z, kao referentno rješenje koristili smo rješenje koje smo dobili prilikom pokretanja test primjera u programskom jeziku *Python*. U njemu postoji modul *networkx* [2], koji ima ugrađenu funkciju za rješavanje problema maksimalnog toka minimalne cijene.

Nadalje, jedno od svojstava koje smo htjeli provjeriti je i vrijeme izvršavanja programa. Sva testiranja izvršena su na računalu sa sljedećim specifikacijama: HP ProBook 4710s, Intel Core2Duo T5870 (2GHz), 4GB RAM, Windows 10 x64. U tu svrhu generirali smo 16 različitih test primjera pomoću programa NETGEN. Dobiveni su sljedeći rezultati:

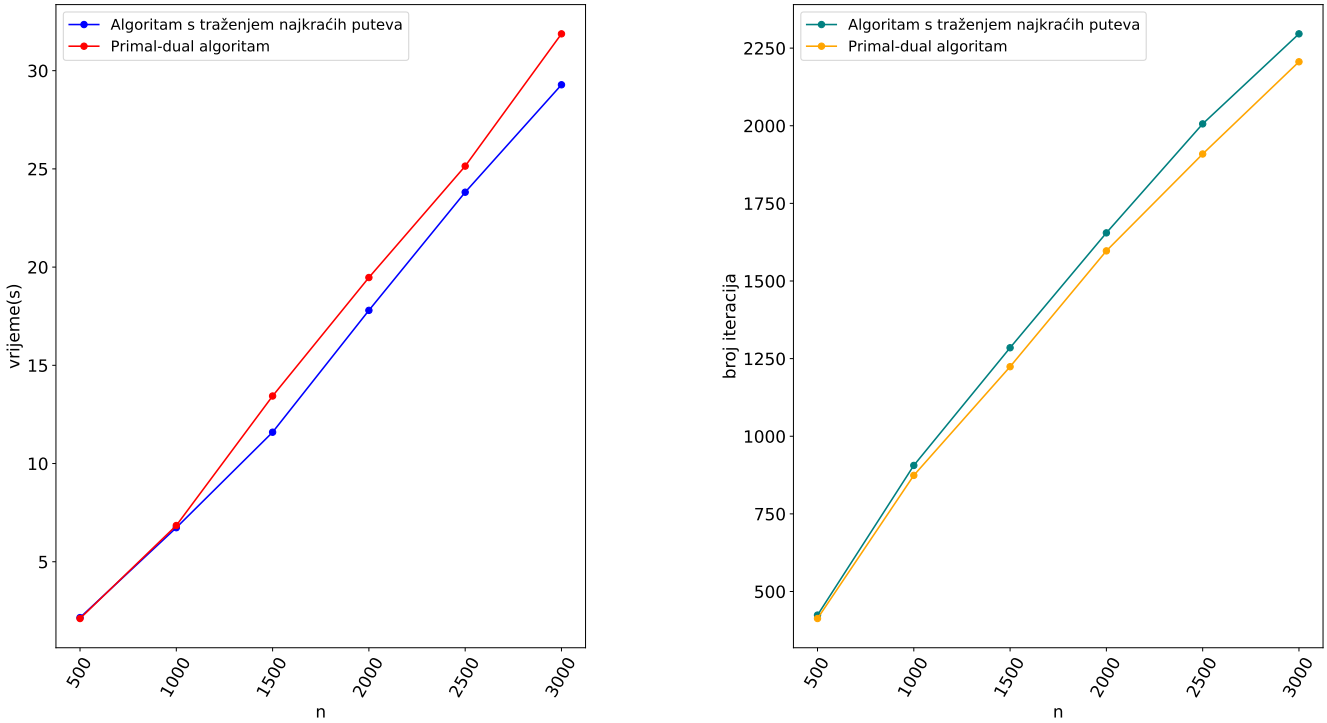


Slika 3.1: Usporedba sva tri algoritma po vremenu izvršavanja

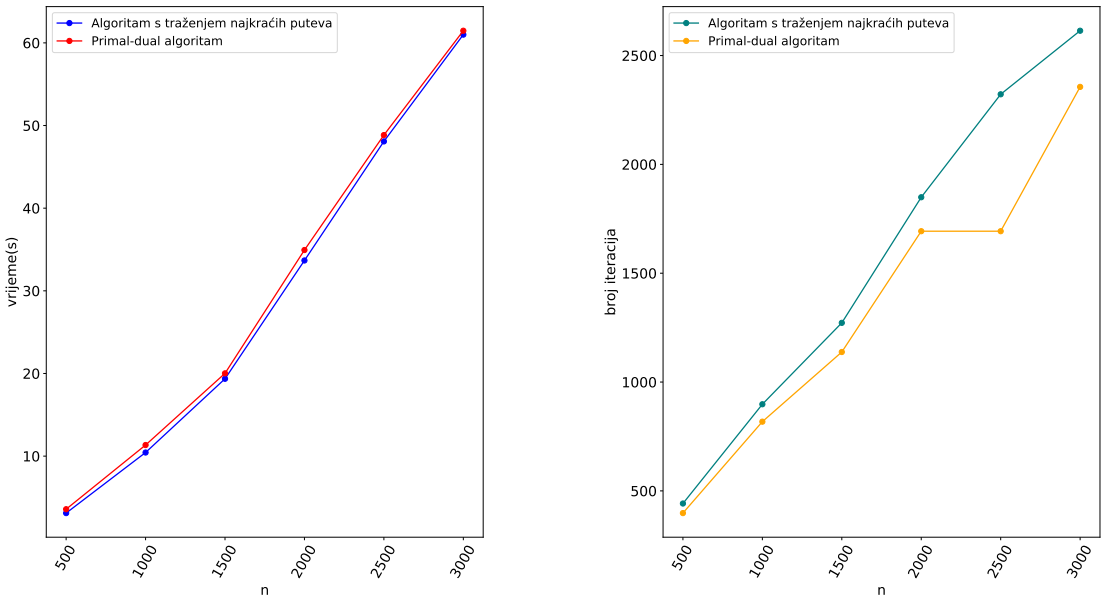
S prethodnog grafa jasno uočavamo da algoritam s poništavanjem negativnih ciklusa ima mnogo dulje vrijeme izvršavanja na danim test-primjerima od preostala dva algoritma. Isto tako, na ovim primjerima vidljivo je i to da algoritam s traženjem najkraćeg puta i

primal-dual algoritam imaju podjednako vrijeme potrebno za izvršavanje. Budući se radilo o primjerima mreža s dosta malim brojem bridova, preostala testiranja na većim mrežama provodit ćemo samo na ta dva brža algoritma.

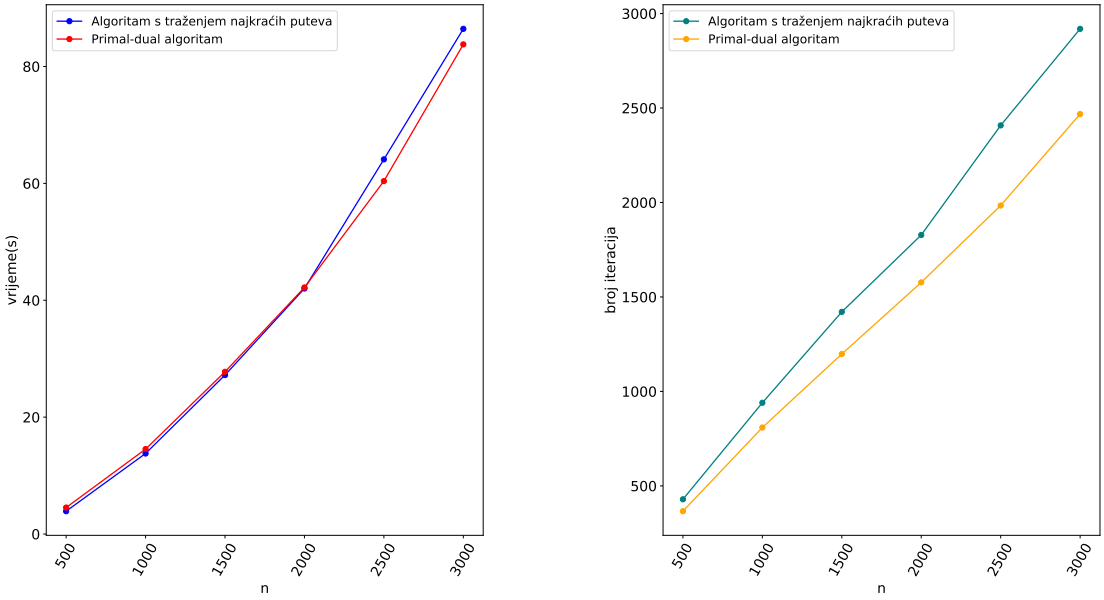
Kako bismo provjerili odnose vremena izvršavanja i broja iteracija dva algoritma, generirali smo primjere koji imaju veći broj vrhova i bridova. Broj vrhova bio je u intervalu od 500 do 3000, dok je broj bridova rastao do 56000. Polovica vrhova u svakoj mreži nije imala niti suficit niti deficit, dok je od preostalih vrhova polovica imala suficit, a polovica deficit. Algoritam s pronalaženjem najkraćih puteva i primal-dual algoritam usporedili smo po vremenu i broju iteracija za različite veličine broja vrhova i broja bridova. Pritom su dobiveni sljedeći rezultati:



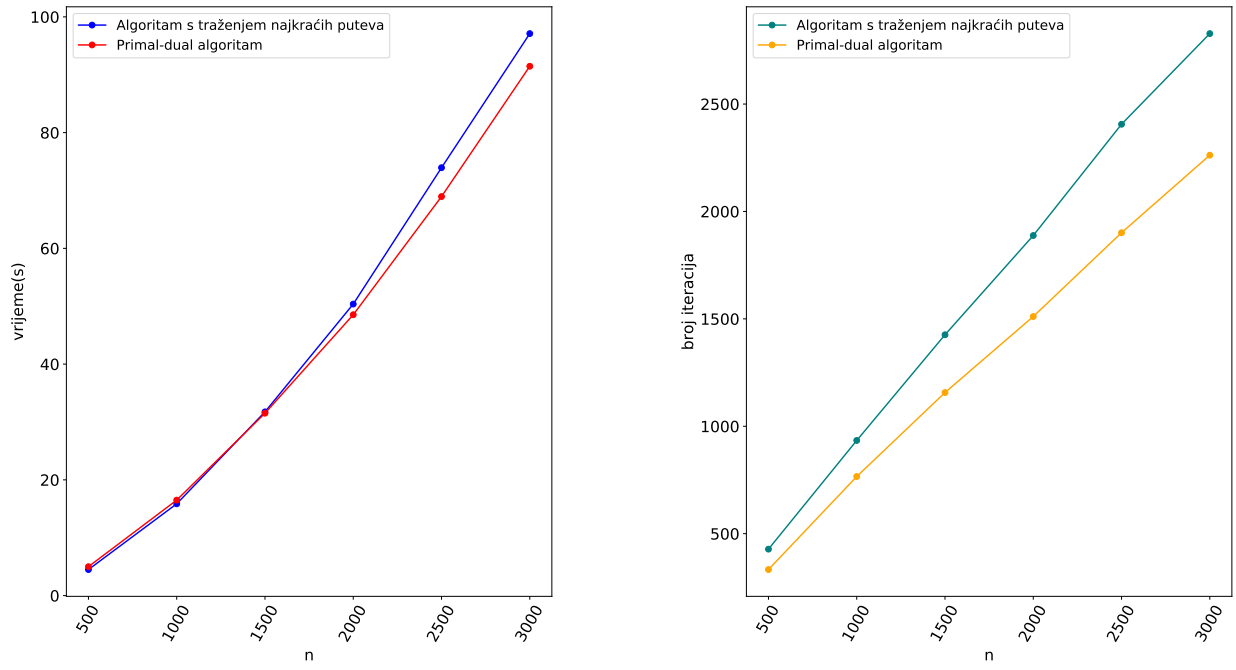
Slika 3.2: Usporedba performansi algoritama za mreže s $m = 8000$ bridova.



Slika 3.3: Usporedba performansi algoritama za mreže s $m = 24000$ bridova.



Slika 3.4: Usporedba performansi algoritama za mreže s $m = 40000$ bridova.



Slika 3.5: Usporedba performansi algoritama za mreže s $m = 56000$ bridova.

Analiziranjem vremena izvršavanja ova dva algoritma, uočavamo da se odnos algoritama po vremenu trajanja mijenja kako broj bridova raste. Zaista, dok je broj bridova bio 8000, primal-dual algoritam imao je nešto veće vrijeme izvršavanja od algoritma s traženjem najkraćih puteva. Porastom broja bridova, primal-dual algoritam postupno je imao kraće vrijeme, pogotovo s porastom broja vrhova u mreži.

Primal-dual algoritam je kroz sve testove imao jednak ili manji broj iteracija od drugog algoritma. Za fiksni broj bridova, razlika u broju iteracija je rasla kako je broj vrhova rastao. Također, jasno se vidi da porastom broja bridova razlika u broju iteracija dostiže i nekoliko stotina, pa primjerice, za $n = 3000, m = 56000$ taj broj iznosi 566.

Analiziranjem složenosti svakog od prikazana tri algoritma jasno je da je algoritam s poništavanjem negativnih ciklusa imao najveće vrijeme izvršavanja. Testiranjem smo uspješno potvrdili činjenicu koju smo pokazali u prethodnom poglavlju da gornja granica u broju iteracija za primal-dual algoritam sigurno nije veća od gornje granice za algoritam s traženjem najkraćih puteva. Složenost pojedine iteracije primal-dual algoritma nešto je lošija od složenosti iteracije za algoritam s traženjem najkraćih puteva, pa je stoga i

vrijeme izvršavanja bilo nešto dulje u odnosu na drugi algoritam kod primjera gdje razlika u broju iteracija nije bila velika. Ipak, kada je razlika u broju iteracija porasla, primal-dual algoritam imao je za koju sekundu brže vrijeme izvršavanja.

Zaključak

Glavna tema ovog rada bio je problem nalaženja maksimalnog toka minimalne cijene u mreži. U prvom dijelu rada naveli smo neke poznate rezultate iz teorije grafova te definirali pojam maksimalnog toka u mreži i način njegova nalaska, što predstavlja bitan dio početnog problema. Osim toga, naveli smo i neke primjere iz svakodnevnog života gdje se problem maksimalnog toka može pojaviti. U drugom dijelu definirali smo problem maksimalnog toka minimalne cijene te uveli nove pojmove kao što su pojam potencijala i reduciranih cijena. Oni su omogućili da razvijemo teoriju i dokažemo neke rezultate koji daju optimalne uvjete za rješenje problema. Središnji dio rada predstavlja opis tri algoritma koji rješavaju početni problem i čije smo performanse kasnije testirali na različitim tipovima mreža. Svojstvo da je dopustivo rješenje ujedno i optimalno ako rezidualna mreža ne sadrži negativan ciklus dalo je uvjete za prvi od algoritama koji smo opisali, algoritam s poništavanjem negativnih ciklusa. Unatoč jednostavnoj implementaciji, uporaba Bellman-Fordovog algoritma za pronalazak negativnog ciklusa u svakoj iteraciji dovela je to toga da se ovaj algoritam pokazao kao najlošiji od opisana tri algoritma. Preostala dva algoritma, algoritam s traženjem najkraćih puteva i primal-dual algoritam bazirala su se na posljedicama teorema koji su povezivali pojmove potencijala, reduciranih cijena i kapaciteta bridova. Oba algoritma imala su dosta slične performanse prilikom testiranja na mrežama s većim brojem vrhova i bridova. Složenost jedne iteracije primal-dual algoritma sugerira da bi taj algoritam mogao imati lošije vrijeme izvršavanja od algoritma s traženjem najkraćih puteva. Međutim, uočili smo da porastom broja vrhova i bridova primal-dual algoritam ima dosta manji broj iteracija od drugog algoritma pa je to utjecalo na činjenicu da je primal-dual algoritam imao nešto kraće vrijeme izvršavanja od algoritma s traženjem najkraćih puteva. Stoga možemo zaključiti da se primal-dual algoritam pokazao kao najbolji od ovdje opisana tri algoritama.

Na kraju još napomenimo da opisana tri algoritma imaju pseudopolinomijalnu složenost, tj. ne ovise samo o broju vrhova i bridova u mreži, već i o kapacitetima i cijenama bridova kao i o zahtjevima pojedinih vrhova. U literaturi se mogu naći i neki polinomijalni algoritmi koji su predstavljeni osamdesetih godina prošlog stoljeća. Ti algoritmi su dosta složeniji od ovdje opisanih algoritama i koriste se u slučaju vrlo velikih mreža koje se javljaju u svakodnevnom životu.

Bibliografija

[1] *NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow networks*, http://elib.zib.de/pub/mp-testdata/mincost/netg/index.html, Pristupljeno: srpanj, 2017.

[2] *Python modul networkx*, https://networkx.readthedocs.io/en/stable/, Pristupljeno: kolovoz, 2017.

[3] *TopCoder tutorial*, https://www.topcoder.com/community/data-science/data-science-tutorials/minimum-cost-flow-part-one-key-concepts/#4, Pristupljeno: srpanj, 2017.

[4] R. K. Ahuja, T. L. Magnanti i J. B. Orlin, *Network flows: theory, algorithms, and applications*, Prentice hall, 1993.

[5] R. G. Busacker i P. J. Gowen, *A procedure for determining a family of miminum-cost network flow patterns*, (1960).

[6] L. R. Ford i D. R. Fulkerson, *Flows in networks*, Princeton University Press (1962).

[7] M. Goetschalckx, *Supply Chain Engineering*, International Series in Operations Research And Management Science, Springer, 2011.

[8] M. Iri, *A new method for solving transportationnetwork problems*, Journal of the Operations Research Society of Japan (1960), 27–87.

[9] W. S. Jewell, *Optimal flow through networks*, Operations Research (1962), 476–499.

[10] D. Jungnickel, *Graphs, Networks and Algorithms*, Algorithms and Computation in Mathematics, Springer Berlin Heidelberg, 2012.

[11] M. Klein, *A primal method for minimal cost flows, with applications to the assignment and transportation problems*, 1967.

Sažetak

U ovom radu proučavali smo problem maksimalnog toka minimalne cijene u mreži. Radi se o poznatom problemu u području teorije grafova, a kao što smo i pokazali, ima široku primjenu i u svakodnevnom životu. Kako bismo istražili teoretsku pozadinu ovog problema, uveli smo pojmove potencijala i reduciranih cijena. To nam je omogućilo da dokažemo nekoliko rezultata u pogledu optimalnih uvjeta za rješenje problema, a koji su nam poslužili kao osnova za dokazivanje točnosti danih algoritama. Pokazali smo da teorija maksimalnog toka minimalne cijene povezuje neke poznate rezultate iz teorije maksimalnog toka i pronalaska najkraćih puteva u mreži. Najveći dio ovog rada odnosi se na tri algoritma: algoritam s poništavanjem negativnih ciklusa, algoritam s traženjem najkraćih puteva i primal-dual algoritam. Pokazali smo nekoliko svojstava tih algoritama te analizirali njihove vremenske složenosti. Testiranjem na skupu podataka koji je uključivao mreže različitih broja vrhova i bridova potvrdili smo pretpostavku da primal-dual algoritam ima najbolje performanse od sva tri prikazana algoritma.

Summary

In this work we study the minimum cost flow problem. This problem is well known in graph theory and has wide applications in everyday life. In order to explore the theoretical background of the problem, we introduced the notions of potentials and reduced costs. This enabled us to prove several results regarding optimality conditions for the problem solution, which served as a basis for showing the correctness of algorithms that solve the problem. The theory of the minimum cost flow problem combines several well known facts from the theory of the maximum flow and the shortest path problems. Large part of this work is focused on three algorithms: the cycle-canceling algorithm, the successive shortest path algorithm and the primal-dual algorithm. We presented a number of properties of these algorithms and analyzed their time complexities. Testing on a dataset that included graphs of varying sizes and densities has confirmed our assumption that the primal-dual algorithm has the best performance among the presented algorithms for solving the minimum cost flow problem.

Životopis

Rođen sam 1. listopada 1993. godine u Zagrebu gdje sam završio i osnovnu školu. Tijekom srednjoškolskog obrazovanja u XV. gimnaziji uspješno sam sudjelovao na državnim natjecanjima iz matematike i informatike. Akademske godine 2012/2013. upisao sam preddiplomski studij Matematika koji sam završio s prosjekom ocjena 5.00, a 2015/2016. godine diplomski studij Računarstvo i matematika. Dobitnik sam Stipendije Grada Zagreba za najbolje studente te dviju nagrada Prirodoslovno-matematičkog fakulteta za najboljeg studenta na preddiplomskom studiju Matematika i diplomskom studiju Računarstvo i matematika.