Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tomaž Šetina

Gomory-Hu drevesa

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Gašper Fijavž

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Gomory-Hu drevo uteženega G je drevesna podatkovna struktura $T(G)$, s katero učinkovito shranjujemo minimalne prereze med točkami grafa G. V diplomskem delu preučite algoritme za izračun Gomory-Hu dreves, jih implementirajte in časovno ovrednotite. Ravno tako preverite morebitne učinkovitejše algoritme za izračun Gomory-Hu dreves v primeru dinamičnega spreminjanja grafa G.

Kazalo

Povzetek

Naslov: Gomory-Hu drevesa

Avtor: Tomaž Šetina

Klasični Ford-Fulkersonov rezultat zlepi problema maksimalnega u-v pretoka in minimalnega u-v prereza med izbranima vozliščema u in v v omrežju — uteženem grafu. V diplomski nalogi se ukvarjamo s problemom Gomory-Hu drevesa, ki v eni sami drevesni strukturi hrani informacijo o vseh minimalnih prerezih v grafu. Natančneje, iskanje minimalnega prereza med poljubnima vozliščema u in v v omrežju lahko predstavimo z iskanjem drevesne povezave z najmanjšo vrednostjo prepustnosti na edini poti med istima vozliščema v Gomory-Hu drevesu. V delu implementiramo Gusfieldov algoritem za izračun Gomory-Hu drevesa in ga časovno ovrednotimo.

Zaradi velike časovne zahtevnosti izračuna Gomory-Hu drevesa implementiramo algoritem za dinamičen izračun novega drevesa iz obstoječega drevesa pri spremembi prepustnosti posamezne povezave v grafu G. Izkaže se, da je algoritem za izračun drevesa pri povečanju prepustnosti povezave v grafu G hitrejši v primerjavi z osnovnim algoritmom. V primeru zmanjšanja prepustnosti povezave ne pride do kvalitativnih razlik.

Ključne besede: Gomory-Hu drevo, maksimalni pretok, minimalni prerez, minimalni k-prerez, dinamični grafi.

Abstract

Title: Gomory-Hu trees

Author: Tomaž Šetina

The classical Ford-Fulkerson algorithm computes a maximum u-v flow and a minimum u-v cut between two selected nodes u, v from flow network — weighted graph. In this thesis we study Gomory-Hu trees which in one tree structure include information about all minimum in the graph. More precisely computing a minimum cut between a pair of nodes u and v nodes in flow network can be reduced to searching for an edge with smallest capacity in the unique u-v path in the Gomory-Hu tree. We implement and evaluate Gusfield algorithm for computing Gomory-Hu tree.

The presented algorithm for computing Gomory-Hu trees has relatively high time complexity, so we also implement an algorithm for dinamically computing Gomory-Hu trees following a capacity change in the graph. It turns out that in the case of increasing capacity the dynamic approach outperforms the basic algorithm. However, we measure no substantial improvement in the case of reducing capacity of an edge.

Keywords: Gomory-Hu tree, max flow, min cut, min k-cut, dynamic graphs.

Poglavje 1

Uvod

Za reševanje marsikaterih problemov v realnem življenju velikokrat naletimo na probleme kombinatorične optimizacije. Eden izmed teh je problem maksimalnega pretoka in minimalnega prereza, pri katerem nas za dano tokovno omrežje zanima največja pretočnost med izbranima vozliščema v njem. To pomeni, koliko enot (električnega toka, vode, materiala, ...) lahko naenkrat dostavimo med izbranima vozliščema. Spoznali bomo drevesno strukturo Gomory-Hu drevo s katero poiščemo optimalno rešitev za vse možne pare vozlišč v tokovnem omrežju in nam omogoča hiter izračun pretočnosti med izbranima vozliščema.

V grafu G z n točkami obstaja 2^{n-1} prerezov, med katerimi pa niso vsi minimalni. Z uporabo Gomory-Hu dreves minimalne prereze stisnemo v drevesno strukturo tako, da se ne prepletajo med seboj, in v tej strukturi hranimo natanko $n-1$ različnih minimalnih prerezov za graf G.

V diplomskem delu bomo najprej v poglavju 2 obravnavali postopek reševanja problema maksimalnega pretoka in minimalnega prereza med izbranima vozliščema v grafu G. Spoznali bomo Edmonds-Karpov algoritem za reševanje problema. V poglavju 3 uvedemo Gomory-Hu drevesa. Opisali bomo njihove bistvene značilnosti in spoznali prvotni Gomory-Hu algoritem in izboljšani Gusfieldov algoritem za njihov izračun. Opisali bomo tudi, na kakšen način bomo testirali algoritme na realnih podatkih in kako lahko

na splošno izračunamo časovno zahtevnost algoritma iz rezultatov meritev. Ocenili bomo vpliv izbire različnih naborov prepustnosti povezav v grafu G na čas izvajanja algoritma za izračun Gomory-Hu drevesa, nato pa izbrali časovno najzahtevnejšo možnost za nadaljnje meritve.

V poglavju 4 bomo navedli primer uporabe Gomory-Hu dreves za reševanje problema minimalnega k-prereza. Analizirali bomo algoritem s katerim rešitev problema aproksimiramo glede na optimalno rešitev z uporabo predhodno izračunanega Gomory-Hu drevesa. Pri tem bomo poleg izpeljave dokaza dodali tudi zgled s katerim dokažemo tesnost aproksimacijskega faktorja.

Govorili bomo tudi o tem, kako lahko učinkovito vzdržujemo izračunano Gomory-Hu drevo v primeru naključnih elementarnih sprememb v grafu G. Spoznali bomo dva algoritma za izračun Gomory-Hu drevesa za graf v katerem smo spremenili prepustnost na eni izmed povezav v grafu G. Ocenili bomo tudi časovni prihranek.

Vse opisane algoritme za izračun Gomory-Hu dreves bomo v diplomskem delu časovno ovrednotili. Pri tem bomo izmerjeno časovno zahtevnost algoritma na realnih podatkih primerjali s teoretično ocenjeno časovno zahtevnostjo.

1.1 Osnovni pojmi

V tem razdelku bomo opisali nekaj osnovnih pojmov, ki jih je potrebno razumeti pri obravnavi diplomskega dela. Definirali bomo pojem grafa, grafovske matrike, nekaterih vrst grafov, proti koncu pa še nekaj struktur v zvezi s sprehodi v grafih.

Graf je urejen par $G = (V, E)$ kjer je V neprazna končna množica točk (vozlišč) grafa G in E množica povezav grafa, pri čemer vsaka povezava povezuje par vozlišč $\{u, v\}$ in ima vozlišče u za soseda vozlišče v in obratno. Število vozlišč v grafu bomo označili z n, velja $n = |V(G)|$. Vozlišča označimo z oznakami od 0 do $n - 1$ ali pa s črkami slovenske abecede, kjer črka a predstavlja oznako 0, črka b oznako 1 in tako dalje. *Stopnja vozlišča* $deg(u)$

je število vozlišč, ki jih ima izbrano vozlišče u za soseda. Vozliščem stopnje 0 pravimo izolirana vozlišča in vozliščem s stopnjo 1 pravimo *listi*. Graf $G = (V, E, w)$ z utežmi na povezavah imenujemo *utežen graf*. Pri tem je $w : E \to \mathbb{R}^+$, za povezavo e je $w(e)$ vrednost uteži na povezavi e.

Matrika sosednosti $A_G = (a_{i,j})_{i,j \in V(G)}$ grafa G je matrika velikosti $n \times n$. Če sta vozlišči i in j sosedi in je $e = ij$, potem je $a_{i,j} = w(e)$. Če vozlišči i in j nista sosedi, je $a_{i,j}$ enako 0. Koeficienti $a_{i,j}$ predstavljajo vrednosti prepustnosti na posamezni povezavi med vozliščema i in j.

V usmerjenem grafu so vozlišča med seboj povezana z usmerjenimi povezavami. Povezave v usmerjenem grafu so urejeni pari vozlišč, matrika sosednosti v takem primeru ni nujno simetrična. Povezava kaže iz prvega vozlišča para v drugo vozlišče para. V primerih, kadar je matrika simetrična, imamo v grafu G za vsako povezavo e njej nasprotno povezavo $\overline{e}$ in lahko povezavi združimo v eno neusmerjeno povezavo med vozliščema. Dobljen graf imenujemo *neusmerjen graf*.

Graf G je *polni graf*, če je vsako vozlišče sosedno z vsakim drugim vozliščem. Poln graf z $n \geq 1$ točkami označimo s K_n in zanj velja: vsako vozlišče v polnem grafu K_n je stopnje $n - 1$ in število povezav grafa K_n je enako $\binom{n}{2} = \frac{n \cdot (n-1)}{2}$.

Grafu G pravimo *mrežni graf* oziroma $n \times n$ *mreža*, kadar so njegova vozlišča števila od 0 do $n^2 - 1$. Robna vozlišča sestavljajo stranice mreže. Za posamezno robno vozlišče i velja:

- vozlišča med 0 in $n - 1$ so zgornji rob mreže
- vozlišča med $n^2 - n$ in $n^2 - 1$ so spodnji rob mreže
- če velja $i \pmod n = 0$, potem je to vozlišče na levem robu mreže
- če velja $i \pmod n = n - 1$, potem je to vozlišče na desnem robu mreže

Posamezno vozlišče i, ki ni robno vozlišče je sosedno z vozlišči $i + 1$, $i - 1$, $i + n$ in $i - n$.

Slučajne grafe v tem delu generiramo v skladu z modelom $\mathcal{G}(n,p)$ [3], kjer je $p \in (0,1)$. Graf $G \in \mathcal{G}(n,p)$ generiramo tako, da med točkama $i,j \in \{0,\ldots,n-1\}$ povezavo ustvarimo z verjetnostjo p. Povezave med pari točk generiramo neodvisno.

Graf G je *ravninski graf*, kadar ga lahko v ravnini narišemo tako, da se povezave sekajo samo v vozliščih grafa.

Sprehod S v grafu $G = (V,E)$ je zaporedje vozlišč in povezav

$$u_0, e_0, u_1, e_1, \ldots, u_{k-1}, e_{k-1}, u_k,$$

pri čemer sta zaporedni vozlišči u_i in u_{i+1} krajišči povezave e_{i+1}. Dolžina sprehoda S, označimo jo z $|S|$, je enaka številu povezav na sprehodu, $|S| = k$. Vozlišče u_0 imenujemo *začetno vozlišče* in vozlišče u_n *končno vozlišče* sprehoda. Sprehod $S = u_0,\ldots,u_n$ je *pot*, če $u_i \neq u_j$ za vse $0 \leq i < j \leq n$. Najkrajši u-v sprehod v grafu je *pot*.

Podgraf grafa G dobimo tako, da iz grafa G odstranimo izbrana vozlišča in povezave. Podgraf H grafa G je *povezan podgraf*, kadar med vsakima dvema vozliščema $i,j \in V(H)$ obstaja pot s krajiščema i in j, ki v celoti leži v H. *Povezana komponenta grafa* v grafu G predstavlja maksimalen povezan podgraf. Če vozlišči u in v pripadata različnima komponentama grafa G, potem v grafu G ne obstaja pot P_{uv} s krajiščema v u in v.

Povezavi $e_p \in E$ pravimo *prerezna povezava* ali *most* v grafu G, kadar ima $G-e$ več komponent, kot jih ima G. Ločene komponente grafa vsebuje samo *nepovezan graf*. Če je povezava e_p most v komponenti K grafa G, potem ima $K - e_p$ natanko dve komponenti.

Poglavje 2

Maksimalni pretok in minimalni s-t prerez

V tem poglavju si bomo pogledali problem maksimalnega pretoka med izbranima vozliščema s in t v grafu G. Omenili bomo pravila, ki jih moramo upoštevati pri izračunu pretoka in predstavili znamenit dokaz o enakosti maksimalnega pretoka in minimalnega s-t prereza. Na koncu bomo opisali Edmonds-Karpov algoritem za reševanje problema maksimalnega pretoka. Navedli bomo psevdokodo ter analizirali prostorsko in časovno zahtevnost za polne, slučajne in grafe v obliki mrež ter jo primerjali z ocenjeno časovno zahtevnostjo. Snov v poglavju je delno povzeta iz spletnih virov [13, 10].

2.1 Problem maksimalnega pretoka

Problem maksimalnega pretoka je eden najpomembnejših problemov kombinatorične optimizacije. Še posebej se pojavlja v transportu, kjer nas zanima količina materiala, ki jo lahko dostavimo med dvema izbranima lokacijama preko vmesnih povezav z določeno vrednostjo prepustnosti. Prepustnosti bomo pravili tudi kapaciteta. V tem primeru imamo namesto uteži na povezavah prepustnosti zato bomo tak graf označili z $G = (V, E, c)$ in mu pravili *tokovno omrežje*. Preslikava c nam torej za posamezno povezavo hrani njeno

vrednost prepustnosti $c(e)$, ki prikazuje koliko enot lahko naenkrat pošljemo skozi izbrano povezavo e.

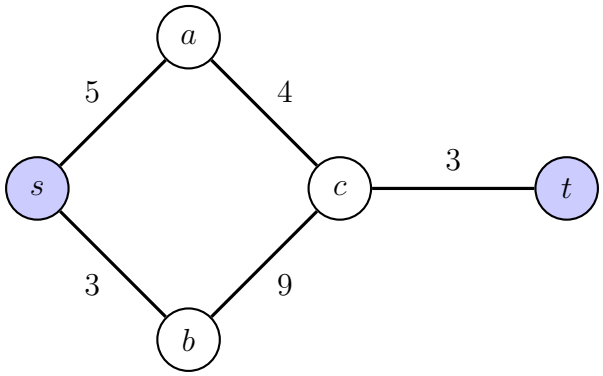
Vhodni podatki za reševanje problema maksimalnega pretoka so graf $G = (V, E, c)$ in izbrani vozlišči s in t v grafu G. V podanem grafu iščemo pretok $f : E \to \mathbb{R}^+$, pri katerem je njegova vrednost $|f|$ kar se da velika. Slika 2.1 prikazuje primer grafa, na katerem nas zanima največja možna vrednost kapacitete pretoka $|f|$ med izbranima vozliščema s in t. To pomeni koliko enot (vode, materiala,...) lahko naenkrat prenesemo iz vozlišča s do vozlišča t z upoštevanjem pravil omejitve kapacitet pretokov skozi posamezno povezavo. Številu enot, ki jih trenutno prenašamo skozi povezavo e med računanjem maksimalnega pretoka, bomo pravili *vrednost pretoka* in označili z $f(e)$. Pravila so naslednja:

1. Vrednost pretoka $f(e)$ skozi povezavo e ne sme preseči njene kapacitete $c(e)$, velja
$$0 \leq f(e) \leq c(e). \tag{2.1}$$

2. Za vsako izbrano vozlišče v v grafu G_f, razen za vozlišči s in t, mora biti vsota vrednosti pretokov skozi povezave ki gredo proti v, enaka vsoti vrednosti pretokov skozi povezave, ki gredo iz v. Pri tem množico povezav, ki imajo začetno vozlišče u označimo z $\delta^+(u)$ in množico povezav s končnim vozliščem u z $\delta^-(u)$. V formalnem zapisu torej dobimo
$$\sum_{e \in \delta^+(u)} f(e) - \sum_{e \in \delta^-(u)} f(e) = 0. \tag{2.2}$$

2.1.1 Pridruženi graf

Za izračun maksimalnega pretoka med vozliščema s in t si pomagamo z grafom G_f, ki ga bomo poimenovali *pridruženi graf* $G_f = (V_f, E_f, c_f)$. Njegova oblika je odvisna od izbranega tokovnega omrežja G in vrednosti pretokov $f(e)$, saj ga zgradimo iz tokovnega omrežja G. Pri ničelnem pretoku skozi



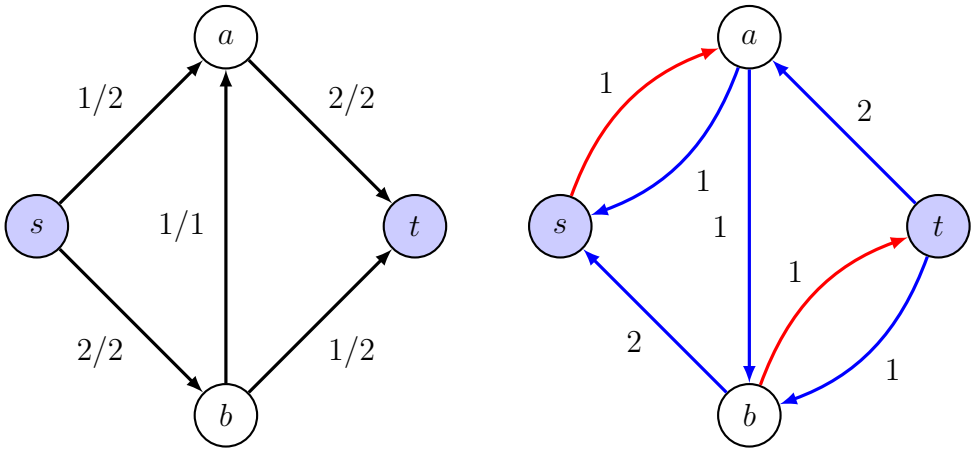
Slika 2.1: Primer grafa $G = (V, E, c)$ z izbranima vozliščema s in t za reševanje problema maksimalnega pretoka.

vse povezave je enak grafu G. Lahko vsebuje dvosmerne povezave, tako kot graf G. Za vsako povezavo e v grafu G_f, ki je vsebovana v grafu G imamo lahko njej nasprotno povezavo, ki jo bomo označili z $\overline{e}$. Graf G_f vsebuje samo povezave skozi katere lahko povečamo pretok. Vrednostim na teh povezavah pravimo *pridružena kapaciteta* in jo označimo z $c_f(e)$. Za graf G_f velja:

1. Množica vozlišč V_f je enaka množici vozlišč V v grafu G.
2. Vrednosti $c_f(e)$ na povezavah $e \in E_f$ v grafu G_f, ki ga dobimo iz tokovnega omrežja G znašajo:

$$c_f(e) = \begin{cases} c(e) - f(e), & \text{če } e \in E_f, \\ f(e), & \text{če } \overline{e} \in E_f, \\ 0, & \text{sicer.} \end{cases}$$

Za lažjo interpretacijo si poglejmo grafa, ki ju prikazuje slika 2.2. Na levi strani imamo graf G, ki predstavlja tokovno omrežje. Vsaka povezava v grafu vsebuje dve vrednosti ločeni z znakom /. Prva vrednost prikazuje vrednost trenutnega pretoka f skozi povezavo e, druga vrednost pa kapaciteto povezave $c(e)$. Vrednost $f(e)$ je vedno manjša od vrednosti $c(e)$. Na desni strani slike imamo graf G_f glede na s-t pretok v prvem grafu.



Slika 2.2: Na levi strani prikaz tokovnega omrežja in na desni strani pridruženega grafa G_f. Povezave označene z rdečo barvo v G_f so vsebovane v tokovnem omrežju G. Njim nasprotne povezave, kjer so vrednosti $c_f(e)$ večje od 0, so označene z modro barvo.

Preden si pogledamo značilnosti za vrednosti $c_f(e)$ v grafu G_f po najdeni poti P_{st}, bomo razložili kaj pomeni *f-povečujoča pot*. Vsaka s-t pot P_{st} v grafu G_f predstavlja f-povečujočo pot. To pomeni, da lahko na tej poti po povezavah $e \in P_{st}$ povečamo pretok za nekaj enot. Vrednost pretoka f, ki ga lahko povečamo skozi povezave na poti, je odvisna od najmanjše vrednosti pridružene kapacitete $c_f(e)$ na povezavi $e = (u, v)$. Pri tem velja:

$$\Delta_f P_{st} = \min\{c_f(u, v) \mid (u, v) \in P_{st}\} > 0 \tag{2.3}$$

Vrednosti $c_f(e)$ na povezavah v G_f so torej po posodobitvi po poti P_{st} v G_f definirane takole:

$$c_f(e) = \begin{cases} c_f(e) - \Delta_f P_{st}, & \text{če } e \in P_{st}, \\ c_f(e) + \Delta_f P_{st}, & \text{če } \overline{e} \in P_{st}, \\ c_f(e), & \text{sicer.} \end{cases}$$

2.2 Prerez

Prerez ali *razdelitev* vozlišč grafa G je par $(U, \overline{U})$, za katerega je $U \cup \overline{U} = V$, $U \cap \overline{U} = \emptyset$ in sta U in $\overline{U}$ neprazni. *s-t prerez* je razdelitev (S, T) vozlišč grafa G, za katero velja $s \in S$ in $t \in T$.

Prerez v usmerjenem grafu razdeli povezave na 4 kategorije: povezave $E(S, S)$ z začetnim in končnim vozliščem v množici S, povezave $E(T, T)$ z začetnim in končnim vozliščem v množici T, povezave $E(S, T)$ z začetnim vozliščem v množici S in končnim vozliščem v množici T ter njim nasprotne $E(T, S)$.

Prepustnost s-t prereza (S, T) označimo s $c(S, T)$ in izračunamo iz povezav $E(S, T)$ takole:

$$c(S, T) = \sum_{e \in E(S,T)} c(e) \tag{2.4}$$

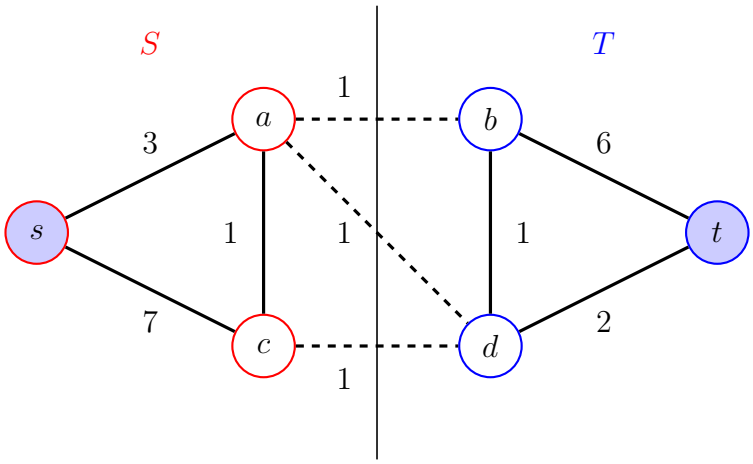
V primeru da iz grafa G odstranimo množico povezav $E(S, T)$, pot med vozliščema s in t ne obstaja več. V grafu G lahko obstaja več s-t prerezov z različnimi množicami povezav in prepustnostjo, med katerimi je (S^*, T^*) *minimalni s-t prerez*, če ima med vsemi s-t prerezi najmanjšo prepustnost. Vsi minimalni s-t prerezi imajo isto prepustnost. Slika 2.3 prikazuje primer minimalnega s-t prereza v grafu G.

2.3 Izrek o maksimalnem pretoku in minimalnem s-t prerezu

V razdelku bomo govorili o izreku maksimalnega pretoka in minimalnega s-t prereza. Napisali bomo trditve, ki veljajo za maksimalni pretok in minimalni prerez in jih dokazali. Pri tem bomo uporabljali tudi zapise v obliki formul.

Izrek 2.1. *Naj bo f pretok v grafu G. Glede na f so naslednje trditve enakovredne v smislu, da držijo bodisi vse bodisi pa nobena.*

(1) f je maksimalen pretok v grafu G,



Slika 2.3: Minimalni s-t prerez v grafu G. Ob odstranitvi črtkanih povezav, ki predstavljajo množico povezav prereza, razdelimo vozlišča grafa na množico $S = \{s, a, c\}$ in $T = \{b, d, t\}$. Barva obrobe vsakega vozlišča ponazarja pripadnost množici vozlišč S ali T. Prepustnost $c(S, T)$ prereza je sestavljena iz vsote prepustnosti črtkanih povezav in znaša 3.

(2) obstaja s-t prerez s kapaciteto $c(S, T)$, kjer je pretok $|f|$ enak kapaciteti tega prereza,

(3) v grafu G_f ne obstaja s-t pot.

Izrek bomo dokazali po korakih, pri čemer iz vsake trditve sledi naslednja trditev in skupaj dobimo cikel. V prvem koraku dokazujemo (2) $\Rightarrow$ (1), v drugem (1) $\Rightarrow$ (3) in v tretjem (3) $\Rightarrow$ (2).

(2) $\Rightarrow$ (1):

V pravilih zapisanih v razdelku 2.1, ki jih moramo upoštevati pri izračunu maksimalnega pretoka in minimalnega prereza med izbranima vozliščema s in t, enačba (2.2) predstavlja tokovno omejitev. Za določen pretok f je njegova vrednost $|f|$ vsota oddanega pretoka iz vozlišča s:

$$|f| := \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e) \tag{2.5}$$

V zapisu enačbe (2.5) velja, da je vrednost $|f|$ enaka vsoti pretoka v vozlišče t ali negativni vrednosti vsote pretoka iz vozlišča t. Z združitvijo

enačbe (2.2) za vsa vozlišča $v \in S \setminus \{s\}$ z enačbo (2.5), dobimo enačbo (2.6), kjer vidimo prispevek posamezne povezave k skupnemu pretoku med vozliščema.

$$|f| = \sum_{v \in S} \left(\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) \right) = \sum_{e \in \delta^-(t)} f(e) - \sum_{e \in \delta^+(t)} f(e) \tag{2.6}$$

V opisu s-t prereza smo omenili, da povezave grafa G razdeli na 4 kategorije. Prispevek posamezne povezave je odvisen od tega, v kateri kategoriji se nahaja. Povezave, ki povezujejo množico S z množico T, prispevajo vrednost $f(e)$, nasprotne povezave pa $-f(e)$ k skupnemu pretoku f. Povezave, ki se nahajajo samo v eni izmed množic S ali T ne prispevajo ničesar.

Z upoštevanjem pravil omejitve kapacitete in predpostavko, da so vse vrednosti pretokov pozitivne, dobimo enačbo (2.7). S tem smo dokazali prvi korak.

$$|f| := \sum_{e \in \delta^+(S)} f(e) - \sum_{e \in \delta^-(S)} f(e) \leq \sum_{e \in \delta^+(S)} c(e) \tag{2.7}$$

$$= c(S, T) \tag{2.8}$$

(1) $\Rightarrow$ (3):

V tem koraku bomo dokazali nasprotje. Naj bo f pretok. V grafu G_f obstaja f-povečujoča pot S med vozliščema s in t. Potem velja, da f ni maksimalen, saj lahko vzdolž take poti v G_f strogo povečamo vrednost f.

(3) $\Rightarrow$ (2):

Za dokaz moramo poiskati množico vozlišč S, pri čemer velja, da za posamezno vozlišče $v \in S$ obstaja pot P_{sv} v grafu G_f.

Za reševanje tega problema lahko uporabimo pregled v širino. Pregled izvajamo iz točke s. Množica S predstavlja množico obiskanih vozlišč v. Velja, da $T = (S, V - S)$ predstavlja s-t prerez. Za izbrano vozlišče $s \in S$ obiščemo samega sebe v G_f. Ob predpostavki, da graf G_f ne vsebuje s-t poti, velja $t \notin S$. s-t prerez bo moral vsebovati množico povezav, ki povezujejo množico T z S. V primeru, da bi obstajala povezava (v, x) iz S v T bi iskalni algoritem obiskal tudi $x \notin S$.

V nadaljevanju si oglejmo, kaj se dogaja z vrednostmi $c_f(e)$ na povezavah v grafu G_f ob s-t prerezu, kjer povezave minimalnega prereza potekajo iz množice T v množico S.

1. Vsaka povezava, ki gre iz množice S v množico T v grafu G ima zapolnjeno kapaciteto pretoka. Vrednost $f(e)$ je enaka $c(e)$. Če bi veljalo $f(e) < c(e)$ za $e \in \delta^+(S)$, bi graf G_f vseboval povezave e, ki bi potekale iz množice S v T, kar je v nasprotju z grafom G_f, ki ga dobimo po izračunanem s-t pretoku.

2. Vsaka povezava, ki gre iz množice T v množico S v grafu G je prazna. Pretok skozi te povezave $f(e)$ je enak 0. V primeru, da bi veljalo $(f(e) < c(e))$ za $e \in \delta^+(S)$, potem bi G_f vseboval povezave, ki gredo iz množice S v T, kar je v nasprotju z grafom G_f, ki ga dobimo po izračunanem s-t pretoku.

Zato za neenakost v enačbi (2.7) velja enakost

$$|f| = c(S, V - S)$$

in tako smo končali z dokazom.

2.4 Edmonds-Karpov algoritem

Opisali bomo Edmonds-Karpov algoritem za izračun maksimalnega pretoka in minimalnega prereza. Pri tem se bomo nanašali na Algoritem 1. Algoritem prejme graf $G = (V, E, c)$ ter izbrani vozlišči s in t kot vhodne podatke. Na izhodu dobimo vrednost maksimalnega pretoka $|f|$, množico vozlišč S, ki se nahajajo na isti strani prereza kot izbrano vozlišče s in množico vozlišč T, ki se nahajajo na isti strani prereza kot izbrano vozlišče t. Na začetku inicializiramo vrednost skupnega pretoka $|f|$ in vrednost pretoka f, ki ga lahko povečamo po poti P_{st} v grafu G_f in prekopiramo matriko sosednosti A_G v matriko sosednosti A_{G_f} pridruženega grafa. Tabela S hrani logično vrednost $S[q]$ za posamezno vozlišče $q \in V(G)$ in nam pove ali se vozlišče po

izračunu maksimalnega pretoka in minimalnega prereza nahaja na isti strani kot s. Algoritem se nato izvaja dokler obstaja pot P_{st} med vozliščema s in t v grafu G_f. V vsaki iteraciji iskanja nove poti uporabimo BFS algoritem z razširitvijo hranjenja poteka poti. Potek poti P_{st} hranimo v obliki tabele v obratni smeri, kot poteka pot v G_f in nam za vsako vozlišče v G_f hrani predhodnika izbranega vozlišča na poti P_{st}.

Po pregledu poti v grafu G_f na koncu vrnemo logično spremenljivko, katere vrednost ponazarja ali med izbranima vozliščema s in t obstaja pot. V primeru, da pot P_{st} obstaja, se moramo sprehoditi preko nje in si zapomniti povezavo z najmanjšo prepustnostjo $c_f(e)$, katere vrednost predstavlja za koliko enot bomo povečali skupno vrednost trenutnega pretoka med vozliščema s in t. V naslednjem koraku je potrebno posodobiti graf G_f, kjer veljajo naslednji primeri:

- če povezava e pripada poti P_{st}, potem bo ta povezava v primeru, da povečanje pretoka skozi to povezavo zapolni njeno kapaciteto $c(e)$, odstranjena iz grafa G_f. Nasprotna povezava $\overline{e}$ bo v primeru, da še ni bila dodana v graf G_f imela vrednost $c_f(e)$ enako vrednosti $\Delta_f P_{st}$ povečanju pretoka skozi pot P_{st}

- če pot P_{st} vsebuje povezavo $\overline{e}$, potem bo ta povezava po povečanju pretoka odstranjena iz grafa G_f, v primeru da povečanje pretoka povzroči ničelni pretok skozi povezavo e, $f(e) = 0$. Povezava e bo dodana v graf G_f, če je bila pred povečanjem pretoka zasičena, $f(e) = c(e)$

- nobena druga povezava ali vozlišče ni dodano v graf G_f

- vsaka nova povezava, ki jo ustvarimo ob povečanju pretoka skozi pot P_{st}, je nasprotna povezava povezavi, ki pripada poti P_{st}.

Po posodobitvi povezav v grafu G_f prištejemo vrednost $\Delta_f P_{st}$ skupni vrednosti pretoka f. Ko med vozliščema s in t v grafu G_f ne obstaja več pot P_{st}, smo končali z računanjem maksimalnega pretoka. Za izračun vozlišč, ki

pripadajo množici S minimalnega s-t prereza v grafu G_f, ponovno uporabimo BFS algoritem. Med tistimi pari, za katere obstaja pot P_{sx}, v tabeli S vozlišču x iz tega para nastavimo logično vrednost $S[x]$ na pozitivno, sicer pa v tabeli T nastavimo vrednost $T[x]$ na pozitivno in tako končamo z izračunom. Na koncu vrnemo maksimalno vrednost pretoka $|f|$ in tabeli S ter T.

Algoritem 1 Edmonds-Karp

Input: $G = (V, E, c), s, t$

Output: $|f|, S, T$

1: $|f| = 0, f = 0, G_f = G, S = [], T = [], S[s] = true, T[t] = true$
2: **while** $BFS(G_f, s, t)$ **do**
3: $\quad |f| = |f| + \Delta_f P_{st}$
4: **for** $x \in V(G) \setminus \{s, t\}$ **do**
5: $\quad$ **if** $BFS(G_f, s, x)$ **then**
6: $\quad\quad S[x] = true$
7: $\quad$ **else**
8: $\quad\quad T[x] = true$
9: **return** $|f|, S, T$

2.4.1 Časovna zahtevnost algoritma

Na začetku prekopiramo matriko sosednosti A_G grafa G v graf G_f, za kar potrebujemo $O(n^2)$ operacij. Za iskanje f-povečujoče poti v vrstici 2 v grafu G_f z minimalnim številom povezav z uporabo BFS algoritma potrebujemo $O(n + m)$ operacij. Po odkriti poti P_{st} je potrebno poiskati povezavo m z minimalno vrednostjo $c_f(e)$, za kar potrebujemo največ $O(n)$ operacij preverjanj vrednosti prepustnosti in to vrednost prišteti skupni trenutni vrednosti pretoka $|f|$. Nato pa je potrebno posodobiti vrednosti prepustnosti v grafu G_f. Za posodobitev vrednosti potrebujemo $O(n)$ operacij. Skupno število operacij enega izvajanja znotraj zanke v vrstici 2 bo torej $O(2 \cdot n)$.

Zanima pa nas število f-povečujočih poti v grafu G_f v najslabšem primeru. Za njihov izračun si predstavljajmo BFS iskalno drevo grafa G_f, kjer začnemo z iskanjem poti v vozlišču s. Vozlišča, ki predstavljajo BFS iskano drevo so urejena po nivojih $L_0, L_1, \ldots, L_k$. Začetni nivo vedno vsebuje samo začetno vozlišče s. Ostali nivoji, kjer je $i > 0$, pa vsebujejo vsa vozlišča v, ki so od vozlišča s oddaljena natanko i. Za BFS algoritem velja še pomembna lastnost, da za vsako vozlišče $v \in L_j$ vsaka najkrajša pot med paroma vozlišč s in t vsebuje natanko eno vozlišče iz vsakega nivoja po vrsti $L_0, L_1, \ldots, L_j$. f-povečujoča pot v grafu G_f je sestavljena iz vozlišč $s = v_0, v_1, \ldots, v_j = t$ in velja $v_i \in L_i$ za $0 \leq i \leq j$. Za izbrano določeno f-povečujočo pot kapaciteta povezave $(u, v) \in E_f$ predstavlja kapaciteto pretoka, ki jo lahko prištejemo k vrednosti skupnega pretoka. Ob povečanju pretoka skozi pot P_{st}, bomo to povezavo odstranili iz grafa G_f. Naj bosta $u \in L_i$ in $v \in L_{i+1}$ ob odstranitvi te povezave. Da bo povezava kasneje ponovno dodana v G_f, se mora vozlišče u nahajati v višjem nivoju v BFS iskanem drevesu kot vozlišče v. Dolžina poti med vozliščema s in v se nikoli ne zmanjša, zato vozlišče v ostaja na nivoju L_{i+1} ali višjem in vozlišče u se mora nahajati na nivoju L_{i+2} ali višjem, preden lahko povezavo e ponovno dodamo v graf G_f. Graf G_f vsebuje dvosmerne povezave, zato lahko vsebuje največ $2 \cdot m$ število povezav in vsaka od njih se lahko pojavi z najmanjšo vrednostjo prepustnosti na poti P_{st} $\frac{n}{2}$-krat. Iz tega sledi, da je največje število f-povečujočih poti enako $\frac{n}{2} \cdot 2 \cdot m = n \cdot m$. V vsaki iteraciji zanke v vrstici 2 najdemo f-povečujočo pot, zato je največje število f-povečujočih poti enako $O(n \cdot m)$. Skupno število operacij zanke v vrstici 2 bo v najslabšem primeru znašalo $O((n \cdot m) \cdot ((n + m) \cdot (2 \cdot n)) = O((n \cdot m) \cdot (2 \cdot n^2 + 2 \cdot n \cdot m) = O((2 \cdot n^3 \cdot m) + (2 \cdot n^2 \cdot m^2)) = O((n^3 \cdot m) + (n^2 \cdot m^2)) = O(n^2 \cdot m^2)$.

Za izračun množice vozlišč, ki sestavljajo tabelo vozlišč S minimalnega s-t prereza (S, T) moramo med vozliščem s in ostalimi vozlišči v G razen s, preveriti obstoj poti med njima v G_f. Uporabljamo BFS algoritem, uporabimo pa ga $O(n)$-krat. Časovna zahtevnost BFS algoritma znaša $O(n + m)$. Skupna časovna zahtevnost za izračun pripadnosti posameznega vozlišča $x \in V \backslash \{s, t\}$

tabeli S ali T je enaka $O(n \cdot (n + m)) = O(n^2 + (n \cdot m))$.

Časovna zahtevnost Edmonds-Karp algoritma je sestavljena iz kopiranja matrike sosednosti grafa G, številom f-povečujočih in časom iskanja povezave z najmanjšo vrednostjo $c_f(e)$ f-povečujoče poti skupaj s posodobitvijo grafa G_f in vsote časovne zahtevnosti za izračun vozlišč, ki se nahajajo na strani vozlišča s ali t s-t prereza. Če zapišemo v obliki enačbe dobimo: $O(n^2 + (n^2 \cdot m^2) + n^2 + (n \cdot m)) = O(2 \cdot n^2 + (n^2 \cdot m^2) + (n \cdot m))$. Zanima nas najhitreje naraščajoči člen glede na števili n in m in dobimo končen rezultat časovne zahtevnosti Edmonds-Karp algoritma $O(n^2 \cdot m^2)$.

2.4.2 Prostorska zahtevnost algoritma

Za hranjenje matrik sosednosti za grafa G in G_f potrebujemo $O(n^2)$ prostora. Med računanjem maksimalnega pretoka za iskanje nove poti uporabimo BFS algoritem. Algoritem uporablja tabelo za hranjenje poteka P_{st} poti s prostorsko zahtevnostjo $O(n)$ in podatkovno strukturo vrsto za hranjenje še neobiskanih vozlišč s prostorsko zahtevnostjo $O(n)$. Prav tako je prostorska zahtevnost $O(n)$ za hranjenje logične vrednosti o pripadnosti posameznega vozlišča grafa G v tabeli S na levi ali T na desni strani minimalnega prereza. Za hranjenje ostalih podatkov bomo predpostavili, da je prostorska zahtevnost $O(1)$. Končna ocena prostorske zahtevnosti algoritma tako znaša $O(n^2 + 3 \cdot n) = O(n^2)$.

Poglavje 3

Gomory-Hu drevo

V tem poglavju bomo predstavili Gomory-Hu drevesa za neusmerjene grafe. Pogledali si bomo kaj predstavljajo, njihove lastnosti ter opisali postopek za izračun drevesa po prvotnem Gomory-Hu in kasnejšem Gusfieldovem algoritmu. Zaradi lažje interpretacije bomo govorili predvsem o slednjem. Kasneje bomo analizirali tudi časovno zahtevnost in ocenili prostorsko zahtevnost. Časovno zahtevnost bomo primerjali z meritvami v praksi. Na koncu si bomo pogledali različne oblike dreves, ki jih dobimo glede na izbor minimalnega prereza v primeru obstoja večih minimalnih prerezov med izbranima vozliščema. Snov v poglavju je delno povzeta iz spletnih virov [7, 12].

V grafu G imamo 2^{n-1} parov komplementarnih podmnožic vozlišč in tudi toliko možnih prerezov, izmed katerih jih je največ $\binom{n}{2}$ minimalnih. S pomočjo Gomory-Hu drevesa te minimalne prereze zapakiramo v strukturo, v kateri hranimo natanko $n-1$ različnih minimalnih prerezov med vsemi pari vozlišč v grafu G. Za Gomory-Hu drevo $T(G) = (V_T, E_T, c_T)$ grafa $G = (V, E, c)$ velja naslednje:

- množica vozlišč v drevesu $T(G)$ je enaka množici vozlišč grafa G,
- vsaka povezava $e \in E_T$ ustreza minimalnemu prerezu v grafu G z množicama vozlišč S in T, ki sta množici vozlišč komponent $T(G) - e$, ta prerez označimo s $C(e)$

- vrednost prepustnosti posamezne povezave e v $T(G)$ predstavlja prepustnost prereza (S, T). Velja $c_T(e) = c(S, T)$
- v primeru, da je dolžina poti P_{st} v drevesu $T(G)$ večja od 1, potem je minimalen s-t prerez v grafu G predstavljen s povezavo z najmanjšo vrednostjo prepustnosti na poti P_{st} v drevesu $T(G)$.

Pravimo, da se prereza $(U,\ \overline{U})$ in $(W,\ \overline{W})$ *prepletata*, če so vse množice $U \cap W$, $U \cap \overline{W}$, $\overline{U} \cap W$ in $\overline{U} \cap \overline{W}$ neprazne. Sicer se prereza *ne prepletata*. V postopku računanja drevesa $T(G)$ je zelo pomembno, da se trenutno izračunan minimalni prerez v grafu G ne prepleta s katerim od predhodno izračunanih minimalnih prerezov. Lema 3.1 opisuje na kakšen način moramo preoblikovati izbran minimalni prerez, da odstranimo prepletanje prerezov.

Lema 3.1. *(Neprepletanje prerezov) Naj bosta* $(U,\ \overline{U})$ *minimalni* u-u' *prerez in* $(W,\ \overline{W})$ *minimalni* w-w' *prerez, ki se prepletata. Potem lahko poiščemo minimalni* w-w' *prerez* $(W^*,\ \overline{W^*})$*, ki se ne prepleta s prerezom* $(U,\ \overline{U})$*.*

Dokaz. Če je $(U,\ \overline{U})$ tudi w-w' prerez, potem lahko za $(W^*,\ \overline{W^*})$ vzamemo kar $(U,\ \overline{U})$. Sicer lahko brez škode za splošnost privzamemo, da $w, w' \in U$ in da $u \in W$. Ločimo dve možnosti glede na lokacijo vozlišča u'. Ta lahko pripada $\overline{U} \cap \overline{W}$ ali pa pripada $\overline{U} \cap W$. Obe možnosti prikazuje slika 3.1.

Če je $u' \in \overline{U} \cap \overline{W}$, potem je $(U \cap W,\ \overline{U \cap W})$ minimalni w-w' prerez in če je $u' \in \overline{U} \cap W$, potem je $(U \cap \overline{W},\ \overline{U \cap \overline{W}})$ minimalni w-w' prerez.

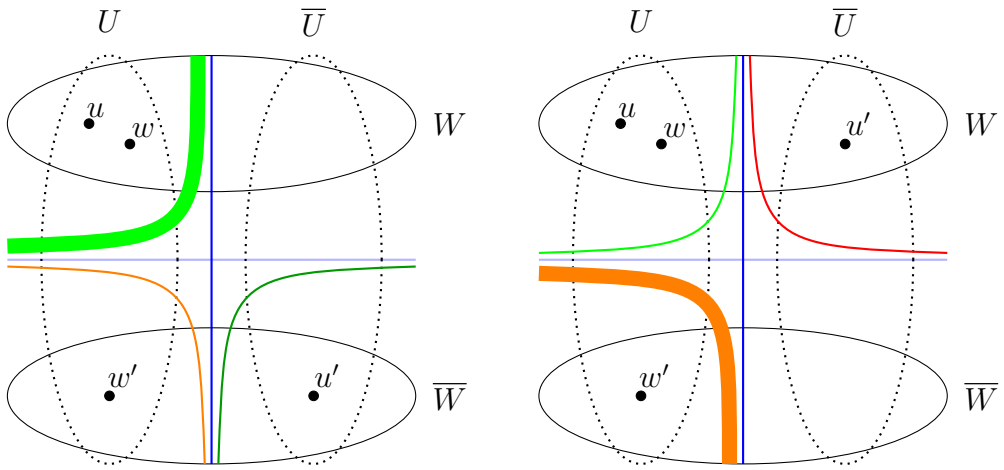
V nadaljevanju bomo dokazali ustreznost zgornje izbire.

Za poljubna minimalna $(U,\ \overline{U})$ in $(W,\ \overline{W})$ prereza v neusmerjenem grafu G velja

$$c(U) + c(W) \geq c(U \cup W) + c(U \cap W). \tag{3.1}$$

Pravimo tudi, da je funkcija za prepustnost prereza c *submodularna*.

Iz (3.1) glede na $u' \in \overline{U} \cap \overline{W}$ zaradi upoštevanja dvakratne vrednosti prepustnosti prepletanja povezav po diagonali $U \cap W$, $\overline{U} \cap \overline{W}$ in $U \cap \overline{W}$, $\overline{U} \cap W$ na levi strani neenakosti, dobimo glede na prepletanje dveh prerezov na obeh straneh slike 3.1



Slika 3.1: Na sliki prikaz prepletanja $(U,\ \overline{U})$ minimalnega u-u' prereza in $(W,\ \overline{W})$ minimalnega w-w' prereza glede na lokacijo vozlišča u'. V prvem primeru imamo $u' \in \overline{U} \cap\ \overline{W}$ in v drugem $u' \in \overline{U} \cap W$. Poleg tega imamo v obeh primerih z različnimi barvami črt označene vse možne prereze, ki jih tvorita skupaj minimalna prereza z medsebojnim prepletanjem. Prereza označena z odebeljeno zeleno črto na prvi sliki in z oranžno črto na drugi sliki glede na lego vozlišča u' predstavljata minimalen w-w' prerez, ki se ne prepleta z $(U,\ \overline{U})$ minimalnim prerezom.

$$c(U \cap W, \overline{U \cap W}) + c(\overline{U} \cap \overline{W},\ \overline{\overline{U} \cap \overline{W}}) \leq c(U,\ \overline{U}) + c(W,\ \overline{W}). \tag{3.2}$$

V primeru, da $(U \cap W,\ \overline{U \cap W})$ ni minimalni w-w' prerez, potem je

$$c(U \cap W,\ \overline{U \cap W}) > c(W,\ \overline{W}) \tag{3.3}$$

in dobimo protislovje. Vemo pa tudi, da je

$$c(\overline{U} \cap \overline{W},\ \overline{\overline{U} \cap \overline{W}}) \geq c(U,\ \overline{U}), \tag{3.4}$$

ker je $(U,\ \overline{U})$ minimalni u-u' prerez. S seštevanjem (3.3) in (3.4) dobimo

$$c(U \cap W, \overline{U \cap W}) + c(\overline{U} \cap \overline{W},\ \overline{\overline{U} \cap \overline{W}}) > c(W,\ \overline{W}) + c(U,\ \overline{U}), \tag{3.5}$$

kar je v nasprotju s submodularnostjo in s tem končamo z dokazom za prvi primer.

V drugem primeru, kadar je $(U \cap \overline{W},\ \overline{U \cap \overline{W}})$ minimalni w-w' prerez, zaradi neupoštevanja dvakratne vrednosti prepustnosti povezav po diagonalah prerezov, dobimo

$$c(U \cap \overline{W},\ \overline{U \cap \overline{W}}) + c(\overline{U} \cap W, \overline{\overline{U} \cap W}) \leq c(W,\ \overline{W}) + c(U,\ \overline{U}). \tag{3.6}$$

Če $(U \cap \overline{W},\ \overline{U \cap \overline{W}})$ ni minimalni w-w' prerez, potem je

$$c(U \cap \overline{W},\ \overline{U \cap \overline{W}}) > c(W,\ \overline{W}) \tag{3.7}$$

protislovje. Vemo pa tudi, da je

$$c(\overline{U} \cap W,\ \overline{\overline{U} \cap W}) \geq c(U,\ \overline{U}), \tag{3.8}$$

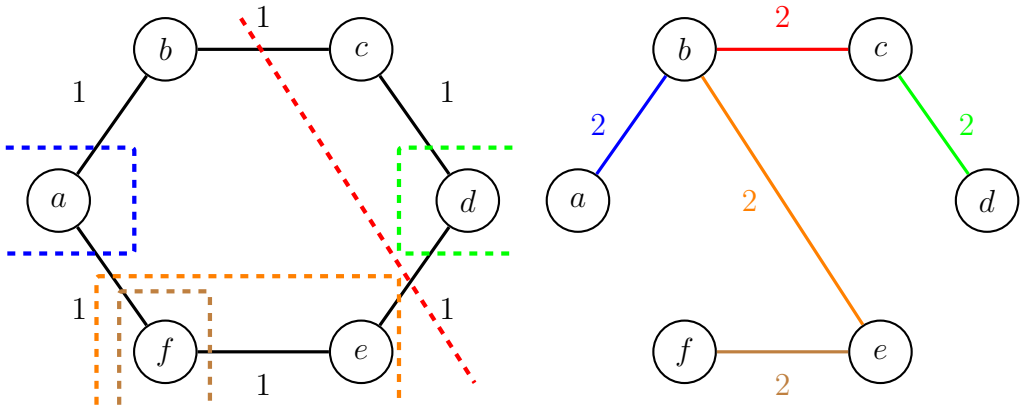
ker je $(U,\ \overline{U})$ minimalni u-u' prerez. S seštevanjem (3.7) in (3.8) dobimo

$$c(U \cap \overline{W}, \overline{U \cap \overline{W}}) + c(\overline{U} \cap W, \overline{\overline{U} \cap W}) > c(W,\ \overline{W}) + c(U,\ \overline{U}) \tag{3.9}$$

kar je v protislovju z submodularnostjo in končamo z dokazom za drugi primer.

□

V primeru drugačnih izbir nepreplетajočih se minimalnih prerezov prepustnosti 2 v ciklu, bi Gomory-Hu drevo lahko izgledalo tudi drugače. Lahko celo pot ali zvezda.

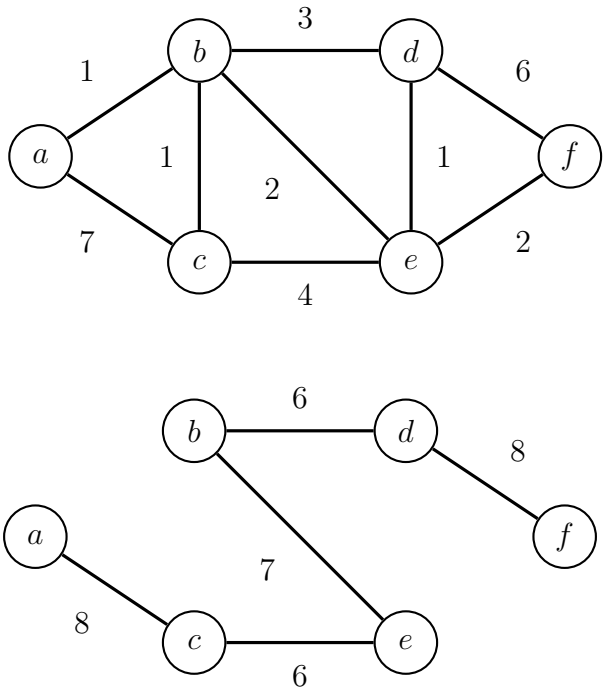


Slika 3.2: Prikaz grafa G in drevo $T(G)$ glede na množico izbranih $n-1$ minimalnih prerezov v grafu G. Posamezen prerez v grafu G je predstavljen z isto barvo povezave v drevesu $T(G)$.

Na sliki 3.3 imamo prikazan graf G in njemu pripadajoče Gomory-Hu drevo $T(G)$. Za primer vzemimo, da nas zanima maksimalni pretok in minimalni prerez v grafu G med vozliščema a in d. Minimalni prerez med vozliščema a in d v grafu G iz drevesa $T(G)$ preberemo na naslednji način:

1. v $T(G)$ poiščemo enolično določeno $a-d$ pot P_{ad}
2. na poti P_{ad} poiščemo povezavo e z najmanjšo prepustnostjo $c_T(e)$ in jo odstranimo iz $T(G)$
3. iz $T(G)-e$ dobimo dve komponenti z množicami vozlišč A in D, pri čemer je $a \in A$ in $d \in D$

4. vrednost prepustnosti $c(A, D)$ odstranjene povezave (A, D) predstavlja vrednost maksimalnega pretoka med vozliščema a in d v grafu G.



Slika 3.3: Zgoraj prikaz grafa G in spodaj Gomory-Hu drevesa $T(G)$.

3.1 Implementacija algoritma

Implementirali bomo Gomory-Hu in Gusfieldov algoritem za izračun drevesa $T(G)$ za graf G. Oba temeljita na problemu maksimalnega pretoka in minimalnega prereza. Algoritem za reševanje tega problema smo spoznali v razdelku 2.4.

Vozlišča grafa G med izračunom drevesa $T(G)$ hranimo v množicah, ki jim pravimo *vreče*. Posamezno vrečo bomo označili z B in predstavlja podgraf grafa G. Vsebuje vozlišča med katerimi minimalnega prereza še nismo izračunali. Na začetku so vozlišča v eni sami vreči B in po vsakem koraku

izračuna maksimalnega pretoka in minimalnega prereza med vozliščema znotraj vreče, dobimo dodatno vrečo. Algoritem se konča, ko imamo natanko $n-1$ vreč in vsaka vsebuje natanko eno vozlišče.

3.1.1 Gomory-Hu algoritem

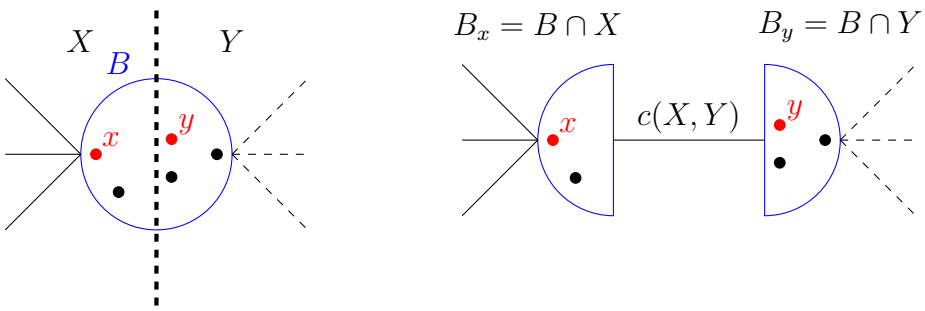
Vhodni podatki: graf $G = (V, E, c)$, pri čemer je n število točk, $V(G) = \{v_1, v_2, \ldots, v_n\}$ množica točk, $E(G)$ množica povezav in c prepustnost povezav $c : E \to \mathbb{R}^+$. Izhodni podatki: drevo $T(G)$ za graf G.

Algoritem izračuna zaporedje uteženih dreves T_1, T_2,..., T_n, za katera velja:

- vsaka povezava v T_i določa prerez v grafu G
- vozlišča drevesa T_i so členi razbitja $V(G)$ na natanko i členov − $V(T_1)$ = $\{V(G)\}$ in $V(T_n)$ = $\{\{v_1\}, \{v_2\}, \ldots, \{v_n\}\}$. Množico točk $V(T_n)$ smiselno enačimo kar z $V(G)$.
- drevo T_{i-1} dobimo iz drevesa T_i tako, da v drevesu T_i stisnemo povezavo B_i, k_1 B_i, k_2 v eno samo vozlišče $B_i, k_1 \cup B_i, k_2$ − particijo $V(T_{i-1})$ iz $V(T_i)$ pridelamo tako, da dva izmed členov nadomestimo z njuno unijo

Za opis algoritma za izračun Gomory-Hu drevesa je potrebno samo še razložiti, kako iz drevesa T_{i-1} pridelamo drevo T_i. Novo drevo pridelamo po naslednjih korakih:

1. izberemo vrečo $B \in V(T_{i-1})$, ki vsebuje vsaj dve točki x in y grafa G, $\{x, y\} \subseteq B$,
2. poiščemo minimalni x, y prerez (X, Y) v grafu G, za katerega velja $x \in X$ in $y \in Y$,
3. vrečo $B \in V(T_{i-1})$ v drevesu T_i nadomestimo s parom sosednjih vreč $B_x = B \cap X$ in $B_y = B \cap Y$ − člena B_x in B_y sta neprazna, saj vsebujeta po vrsti x oziroma y,



Slika 3.4: Na levi strani prikaz vreče B, ki vsebuje izbrani x in y vozlišči označeni z rdečo barvo. Na desni strani razbitje izbrane vreče B na vreči B_x in B_y glede na izračunan minimalni prerez med izbranima vozliščema v grafu G. Med dobljenima vrečama dodamo povezavo z vrednostjo prepustnosti minimalnega (X, Y) prereza. Polne povezave na levi in črtkane povezave na desni strani povezujejo vreče, ki so sosedne vreči B na levi ali desni strani prereza in se lahko po izračunanem minimalnem prerezu nahajajo na nasprotni strani. V tem primeru je vozlišče x ostalo povezano z vrečami na levi strani prereza in vozlišče y z vrečami na desni strani prereza ob odstranitvi povezav, ki predstavljajo minimalni prerez.

4. prerez (X, Y) v drevesu utežimo s $c(X, Y)$,

5. za vsakega soseda B' vreče B v drevesu T_{i-1} povezavo $B'B$ v drevesu T_i nadomestimo z eno od povezav med B' in $B \cap X$ oziroma med B' in $B \cap Y$. Povezavo $B'B_x$ dodamo, če $B' \subseteq X$, če pa je $B' \subseteq Y$, potem dodamo povezavo $B'B_y$. V vsakem primeru se zgodi natanko ena od možnosti $B' \subseteq X$ ali $B' \subseteq Y$.

Slika 3.4 vizualno prikazuje kako iz drevesa T_{i-1} pridelamo drevo T_i.

3.1.2 Gusfieldov algoritem

Gusfieldov algoritem je izboljšava Gomory-Hu algoritma za izračun Gomory-Hu drevesa. Razlika je v tem, da

1. izberemo najmanjšo vrečo B z vsaj dvema elementoma, pri čemer naj bo indeks vreče B točka $s \in V(G)$,

2. za t izberemo najmanjšo točko iz $B \setminus \{s\}$.

To lahko implementiramo s pomočjo tabel. V nadaljevanju bomo opisali Gusfieldov algoritem za izračun Gomory-Hu drevesa, glej Algoritem 2.

Vhodni podatek je graf $G = (V, E, c)$ in izhodni podatek drevo $T(G)$. Pri razlagi si bomo pomagali s psevdokodo algoritma in tabelama *drevo* ter *pretoki*. Vozlišča grafa G so števila od 0 do $n-1$. Vse tabele so velikosti n in na začetku vsebujejo samo vrednosti 0. Tabela *drevo* hrani strukturo drevesa in nam za vsako vozlišče i v grafu G pove, s katerim vozliščem v drevesu bo vozlišče i sosedno. Povezave drevesa bodo pari $(i, drevo[i])$. Tabela *pretoki* nam za vsako vozlišče grafa G hrani vrednost maksimalnega pretoka.

Na začetku imamo drevo $T(G)$ grafa G v obliki zvezde, v katerem je vozlišče 0 sredinsko vozlišče in so vsa ostala vozlišča sosednja vozlišču 0. V vsaki iteraciji izberemo različno vozlišče $s \in V_G$, za katero mora veljati $s \geq 1$. Takšna izbira izvornih vozlišč nam določi ponorno vozlišče $t = drevo[s]$, ki je trenutno sosednje vozlišče vozlišču s v drevesu $T(G)$. Nato je potrebno izračunati maksimalni pretok in minimalni s-t prerez (S, T) v grafu G. Za izračun maksimalnega pretoka in minimalnega s-t prereza (S, T) uporabimo Edmonds-Karpov algoritem. Njegovo delovanje smo opisali v podpoglavju 2.4. Po izračunu maksimalnega pretoka $|f|$ in množice vozlišč S posodobimo vrednosti v tabelah. V tabeli *pretoki* si za izbrano izvorno vozlišče s zapomnimo vrednost maksimalnega pretoka ($pretoki[s] = |f|$). V vrstici 7 vsem vozliščem, ki so sosednja vozlišču t v drevesu $T(G)$ in se nahajajo v tabeli S izračunanega minimalnega s-t prereza ter niso enaka s, dodamo vozlišču s za sosedna in jih tako odcepimo od vozlišča t. Na koncu preverimo ali množica vozlišč S vsebuje vozlišče $drevo[t]$. V primeru da ga vsebuje, vsem vozliščem v $T(G)$ dodelimo vozlišče $drevo[s]$ za soseda oziroma jih preusmerimo k tem vozlišču. Potem pa vsem vozliščem v $T(G)$, ki so imela vozišče $drevo[t]$ za sosednjo, dodelimo vozlišče s za sosednjo. Potrebno je še posoditi vrednosti pretokov. Vozlišču s v tabeli *pretoki* nastavimo vrednost

$pretoki[t]$ in $pretoki[t]$ nastavimo na vrednost trenutno izračunanega maksimalnega pretoka $|f|$ in tako končamo z iteracijo izračuna.

Algoritem 2 Gusfield

Input: $G = (V, E, c)$

Output: $drevo, pretoki$

```
1: drevo = [], pretoki = [], S = [], T = []
2: for s = 1 to |V(G)| − 1 do
3:     t = drevo[s]
4:     |f|, S, T = Edmonds-Karp(G, s, t)
5:     pretoki[s] = |f|
6:     for i ∈ V(G) do
7:         if i ≠ s && S[i] == true && drevo[i] == t then
8:             drevo[i] = s
9:     if S[drevo[t]] == true then
10:        drevo[s] = drevo[t]
11:        drevo[t] = s
12:        pretoki[s] = pretoki[t]
13:        pretoki[t] = |f|
14: return drevo, pretoki
```

Ocena praktične časovne zahtevnosti algoritma

V tem podpoglavju bomo opisali, na kakšen način ocenimo časovno zahtevnost algoritma iz tabele, v kateri hranimo za določeno število vozlišč n_i grafa povprečno vrednost časa izvajanja algoritma v sekundah za izračun Gomory-Hu drevesa $\overline{t_{n_i}}$. Opisali bomo tudi postopek izračuna premera drevesa $T(G)$ in na koncu definirali interval prepustnosti povezav $e \in E_T$.

Časovne zahtevnosti algoritmov, ki jih bomo obravnavali v diplomskem delu so polinomske in zato predstavljajo funkcije oblike $f(n) = (b \cdot n^a) + o(n^a)$. Naloga je torej poiskati funkcijo $f(n)$, za katero koeficienta a in b določimo tako, da se regresijska premica oblike $a \cdot \log(n) + \log(b)$ izračunana po metodi

najmanjših kvadratov najbolje prilega logaritmiranim meritvam $(n_i, t_{n_i}) = (\log(n_i), \log(t_{n_i}))$.

Poleg izračuna časovne zahtevnosti smo računali tudi povprečno vrednost premera dreves, izračunanih v posameznem koraku. Premer posameznega drevesa izračunamo z uporabo BFS algoritma tako, da izračunamo iz izbranega vozlišča $p \in V(G)$ razdaljo do preostalih vozlišč v G. Iz dobljenih izračunanih razdalj si zapomnimo vozlišče o z največjo oddaljenostjo od vozlišča p. Nato ponovno izračunamo razdaljo od vozlišča o do preostalih vozlišč v grafu G in si zapomnimo največjo vrednost razdalje in ta vrednost predstavlja premer drevesa $T(G)$.

Izbor vrednosti prepustnosti na povezavah

Pri analizi časovne zahtevnosti Gusfieldovega algoritma moramo določiti način izbire vrednosti prepustnosti na povezavah in tipe grafov G, saj vplivajo na čas izračuna drevesa. Algoritem smo testirali na treh različnih tipih grafov G. Na polnih grafih $\mathcal{K}_n$, slučajnih grafih $\mathcal{R}_n$ in grafih v obliki kvadratnih mrež $\mathcal{M}_n$ na n vozliščih. Za posamezno število vozlišč grafa smo stokrat generirali graf in izračunali pripadajoče drevo $T(G)$. Pri tem smo za vsak tip grafa vzeli dve različni vrednosti števila vozlišč. Za grafe $\mathcal{R}_n$ smo izbrali verjetnost posamezne povezave $p = \frac{5}{n}$. Za izbor vrednosti prepustnosti smo uporabili enakomerno izbiro s celoštevilskega intervala. V prvem primeru vrednosti od 1 do n. V drugem primeru smo vzeli vrednosti v intervalu od 10^6 do $2 \cdot 10^6$. S tem smo aproksimirali enakomerno zvezno porazdelitev na intervalu od 1 do 2. Nazadnje smo vzeli prepustnosti v vrednosti od 1 do 5. Grafi na sliki 3.5, ki jim pravimo *škatle z brki*, prikazujejo čase izračuna drevesa za grafe $\mathcal{K}_n$, $\mathcal{R}_n$ in $\mathcal{M}_n$ glede na izbran interval prepustnosti. Znotraj posameznega grafa imamo prikazane tri škatle z brki, kjer vsaka prikazuje čase izračunov dreves $T(G)$ za določen interval vrednosti prepustnosti na povezavah. Posamezno škatlo z brki dobimo tako, da čase meritev izračuna dreves uredimo po vrednostih naraščajoče in jih razdelimo na kvartile. Prvi kvartil, pravimo mu tudi *spodnji kvartil*, predstavlja vrednost časa izračuna

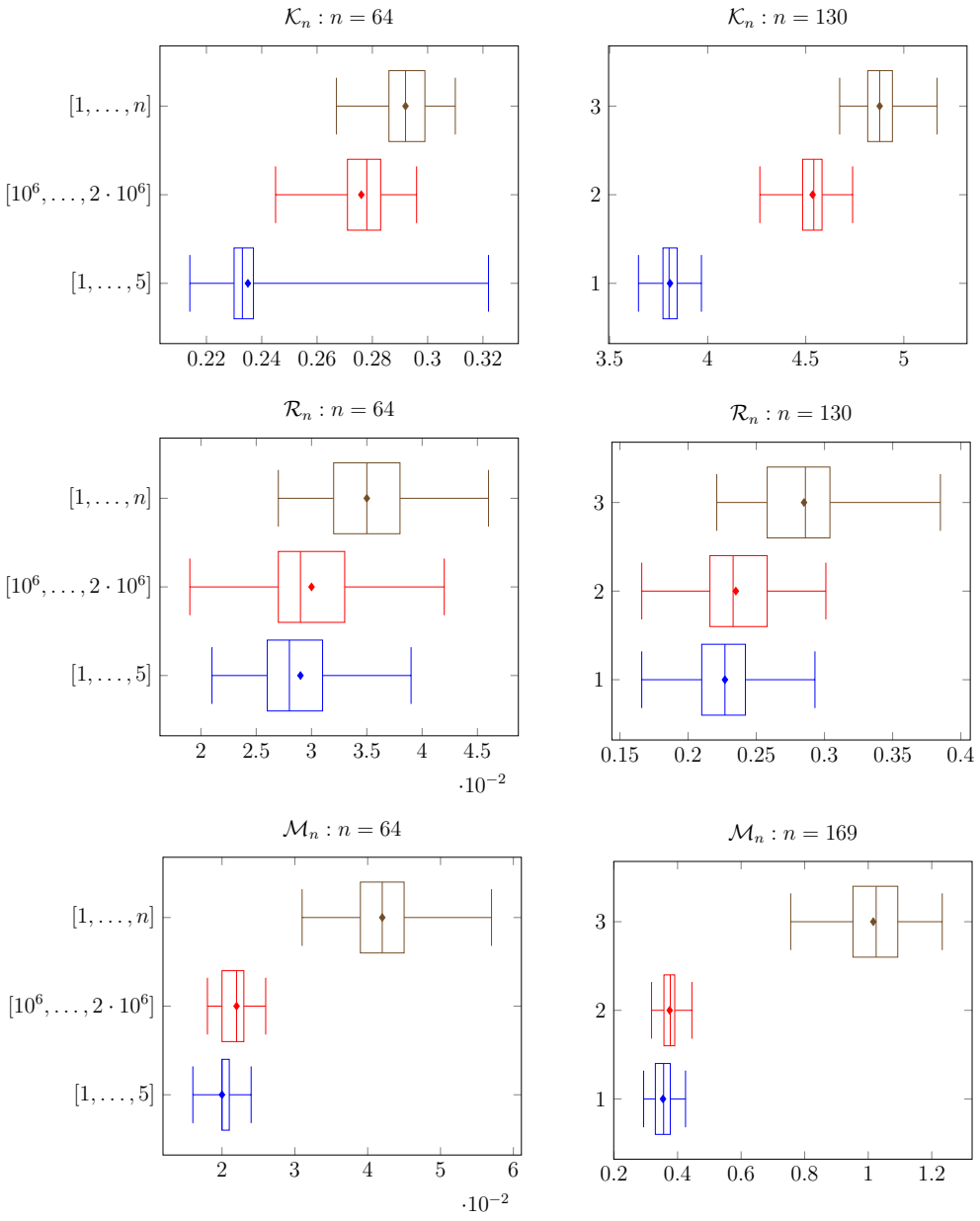
za katerega velja, da ima četrtina meritev manjšo vrednost, ostale tri četrtine pa večjo vrednost. Tretji kvartil, pravimo mu tudi *zgornji kvartil*, pa predstavlja vrednost za katero velja ravno obratno kot za prvi kvartil. Navpična črta skrajno levo predstavlja minimalni čas izračuna in skrajno desna navpična črta maksimalni čas izračuna. Levi rob škatle predstavlja vrednost prvega kvartila in desni rob škatle vrednost desnega kvartila. Znotraj nje navpična črta predstavlja vrednost mediano in romb predstavlja povprečno vrednost meritev. Ožja škatla z brki pomeni bolj koncentrirane čase meritev.

Iz škatel z brki je razvidno, da z izbiro vrednosti prepustnosti povezav med 1 in n dobimo najdaljše čase izvajanja algoritma, zato bomo ta interval prepustnosti uporabili na vseh treh tipih grafov pri analizi časovne zahtevnosti posameznega algoritma v diplomi. Iz tega lahko sklepamo, da je časovna zahtevnost izračuna drevesa odvisna od razmerja med najmanjšo in najvišjo vrednostjo prepustnosti. Čim višja je ta vrednost, večji bodo tudi časi izračuna drevesa.

Časovna zahtevnost algoritma

V tem podrazdelku bomo teoretično analizirali časovno zahtevnost Gusfieldovega algoritma. Ocenjeno časovno zahtevnost bomo primerjali s časovno zahtevnostjo algoritma izračunano iz rezultatov meritev in komentirali razlike med njima. Na koncu bomo grafično za posamezne tipe grafov G prikazali rezultate meritev in regresijske premice.

Za izračun Gomory-Hu drevesa je potrebno izračunati natanko $n - 1$ maksimalnih pretokov in minimalnih prerezov. Iz tega sledi da je ocenjena časovna zahtevnost algoritma približno enaka produktu $O(n)$ in času izvajanja Edmonds-Karp algoritma, katerega smo v podpoglavju 2.4.1 analizirali in znaša $O(n^2 \cdot m^2)$. Po izračunu maksimalnega pretoka in minimalnega prereza na vsakem koraku ustrezno posodobimo strukturo drevesa, za kar potrebujemo $O(n)$ operacij. Časovna zahtevnost preostalega dela algoritma je enaka $O(1)$, saj samo posodobimo posamezne vrednosti v tabelah. Skupna časovna zahtevnost algoritma tako znaša $O(n \cdot (n^2 \cdot m^2) \cdot n) = O(n^4 \cdot m^2)$.



Slika 3.5: Prikaz škatel z brki za čase izračuna dreves za grafe $\mathcal{K}_n$, $\mathcal{R}_n$ in $\mathcal{M}_n$ glede na posamezne intervale vrednosti prepustnosti povezav.

V nadaljevanju bomo primerjali teoretično časovno zahtevnost z meritvami v praksi. Algoritem smo testirali in določili njegovo časovno zahtevnost v praksi po postopku opisanem v podrazdelku 3.1.2.

Za grafe $\mathcal{K}_n$ je število povezav m enako $O(n^2)$, zato bo teoretična časovna zahtevnost za izračun maksimalnega pretoka in minimalnega prereza v tem primeru znašala namesto $O(n^2 \cdot m^2)$, $O(n^2 \cdot n^4) = O(n^6)$ in skupna časovna zahtevnost Gusfieldovega algoritma bo $O(n^8)$. Za grafe $\mathcal{R}_n$ in $\mathcal{M}_n$ bomo predpostavili enako časovno zahtevnost kot v osnovni izračunani, le da bomo vzeli za $m = O(n)$ in dobimo torej $O(n^4 \cdot n^2) = O(n^6)$.

V nadaljevanju bomo opisali opisali postopek analiziranja časovne zahtevnosti algoritma in prikazali rezultate meritev.

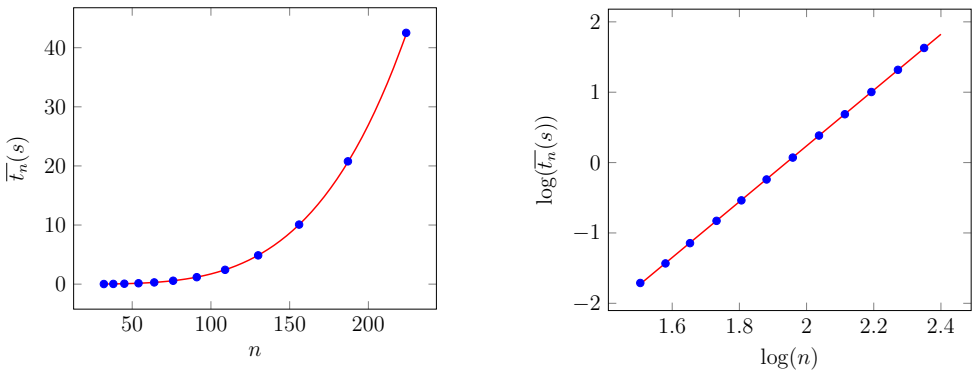
Pri grafih $\mathcal{K}_n$ in $\mathcal{R}_n$ smo vzeli za začetno število vozlišč $n = 32$ in jih nato v vsakem naslednjem koraku povečali približno za faktor 1.2. Skupno smo izvedli 12 korakov meritev. Za grafe $\mathcal{M}_n$ smo vzeli kvadratna števila z začetno vrednostjo $n = 16$ in z enakim številom korakov izračunov. V posameznem koraku smo stokrat generirali graf z različnimi vrednostmi prepustnosti povezav in izračunali pripadajoča drevesa $T(G)$.

Izračunana regresijska premica za logaritmirane rezultate meritev v praksi na grafih $\mathcal{K}_n$ znaša $3.966 \cdot n - 7.695$. Dobimo funkcijo za časovno zahtevnost $f_1(n) = 10^{-7.695} \cdot n^{3.966}$ in iz tega dobimo ocenjeno časovno zahtevnost algoritma $O(10^{-7.695} \cdot n^{3.966}) = O(n^4)$. Govorimo o kvartični časovni zahtevnosti. Pri tem je potrebno omeniti, da je vrednost člena b zelo majhna. Do razlike med teoretično in ocenjeno časovno zahtevnostjo v praksi za grafe $\mathcal{K}_n$ pride zaradi manjše časovne zahtevnosti algoritma za izračun maksimalnega pretoka in minimalnega prereza med izbranima vozliščema. Predpostavili smo, da je ob vsaki uporabi Edmonds-Karp algoritma f-povečujočih poti v grafu G_f enako $n \cdot m$. Glede na časovno zahtevnost je f-povečujočih poti precej manj in tudi njihove dolžine so krajše ter tako hitreje posodobimo vrednosti v grafu G_f.

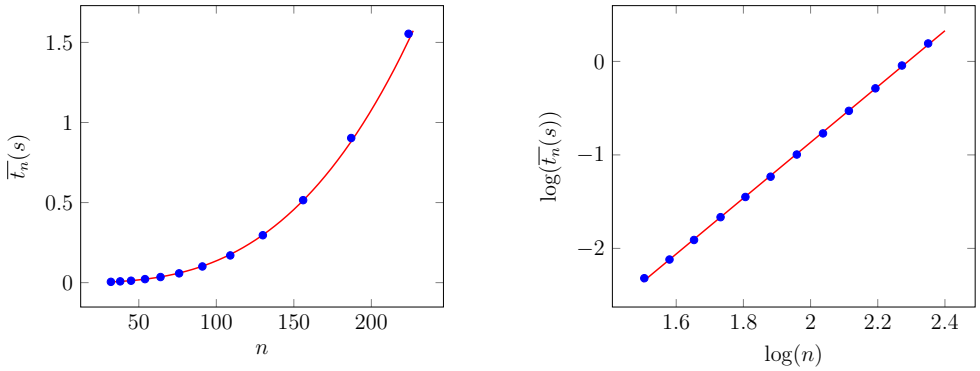
Izračunana regresijska premica za grafe $\mathcal{R}_n$ znaša $2.985 \cdot n - 6.836$. Dobimo funkcijo $f_2(n) = 10^{-6.836} \cdot n^{2.985}$. Ocenjena časovna zahtevnost torej znaša

$O(10^{-6.836} \cdot n^{3.223}) = O(n^3)$. Za grafe $\mathcal{M}_n$ smo dobili regresijsko premico $3.223 \cdot n - 7.180$. Funkcija bo torej $f_3(n) = 10^{-7.180} \cdot n^{3.223}$ in ocenjena časovna zahtevnost algoritma $O(10^{-7.180} \cdot n^{3.223}) = O(n^3)$. Ob tem je potrebno ponovno omeniti, da smo za grafe $\mathcal{R}_n$ določili verjetnost povezave $p = \frac{5}{n}$. To pomeni, da je pričakovana stopnja posameznega vozlišča enaka 5. Za grafe $\mathcal{M}_n$ je pričakovana stopnja posameznega vozlišča enaka 4. Pričakovani vrednosti stopnje posameznega vozlišča sta za oba tipa grafov skoraj enaki, torej imata oba tipa grafov približno enako število povezav. Vrednosti eksponenta polinomske ocene v zapisu regresijskih premic in s tem časovni zahtevnosti za izračun dreves za grafe $\mathcal{R}_n$ in grafe $\mathcal{M}_n$ sta zato tudi blizu skupaj.

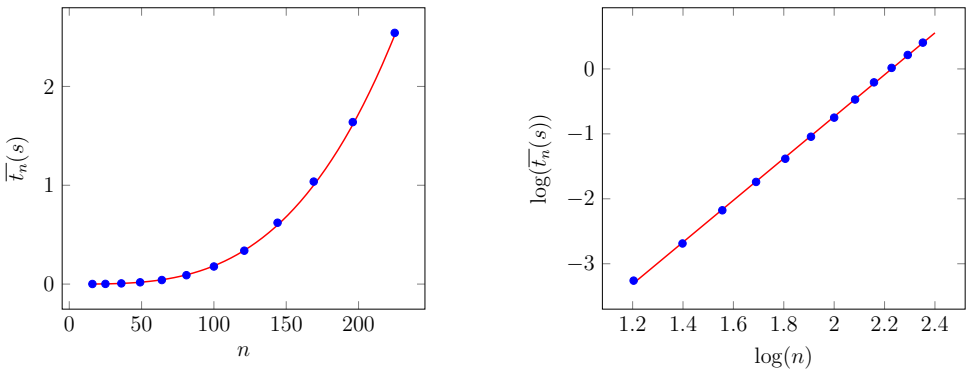
Slike 3.6, 3.7 in 3.8 grafično prikazujejo rezultate meritev za posamezne tipe grafov. Na posamezni sliki imamo dva grafa. V prvem grafu točke označene z modro barvo predstavljajo čase meritev izvajanj algoritma v sekundah glede na število vozlišč n. V drugem grafu točke označene z isto barvo predstavljajo logaritmirane vrednosti iz prvega grafa in na podlagi teh točk izračunano regresijsko premico označeno z rdečo barvo. Iz dobljene regresijske premice dobimo funkcijo za časovno zahtevnost algoritma, prikazano z rdečo barvo v prvem grafu.



Slika 3.6: Prikaz rezultatov meritev izračuna dreves $T(G)$ z uporabo Gusfieldovega algoritma za grafe $\mathcal{K}_n$.



Slika 3.7: Prikaz rezultatov meritev izračuna dreves $T(G)$ z uporabo Gusfieldovega algoritma za grafe $\mathcal{R}_n$.



Slika 3.8: Prikaz rezultatov meritev Gusfieldovega algoritma za izračun dreves $T(G)$ za grafe $\mathcal{M}_n$.

Prostorska zahtevnost algoritma

Prostorska zahtevnost algoritma predstavlja število podatkov, ki jih algoritem potrebuje za izračun Gomory-Hu drevesa. Za hranjenje posameznih vrednosti je prostorska zahtevnost $O(1)$, za tabele velikosti n znaša $O(n)$ in za predstavitev grafov v obliki matrike sosedov velikosti $n \times n$ znaša $O(n^2)$.

Prostorska zahtevnost grafa G predstavljenega v obliki matrike sosednosti je $O(n^2)$. Za hranjenje vrednosti izbranih vozlišč s in t, število vozlišč n ter vrednosti k za razdelitev drevesa na k povezanih komponent, potrebujemo $O(1)$ prostora. Tabelo *drevo* uporabimo za hranjenje strukture drevesa in tabelo *pretoki* za hranjenje vrednosti pretoka za posamezno vozlišče, njuni prostorski zahtevnosti znašata $O(2 \cdot n) = O(n)$. Po izračunu maksimalnega pretoka si moramo to vrednost zapomniti in množico vozlišč S in T v obliki tabele, za kar potrebujemo $O(2 \cdot n) = O(n)$ prostora. Prostorsko zahtevnost algoritma za izračun maksimalnega pretoka in minimalnega prereza smo obravnavali v podpoglavju 2.4.2 in znaša $O(n^2)$.

Skupna prostorska zahtevnost algoritma znaša vsota zgoraj opisanih vrednosti: $O((2 \cdot n^2) + (2 \cdot n)) = O(n^2)$. Pri tem nismo upoštevali hranjenja posameznih vrednosti, saj smo za te podatke predpostavili prostorsko zahtevnost enako $O(1)$. V zapisu smo odstranili konstante, saj nas pri prostorski zahtevnosti podobno kot pri časovni zanima le najhitreje naraščajoči člen.

3.2 Gomory-Hu drevo ni enolično

V opisu izračuna drevesa $T(G)$ z uporabo Gusfieldovega algoritma smo omenili, da vozlišča grafa označimo z vrednostmi na intervalu od 0 do $n-1$. Za zgled vzemimo grafa G_1 in G_2. Graf G_2 dobimo iz grafa G_1 s preimenovanjem oznak vozlišč na istem intervalu. To pomeni da bomo za določen graf G_2 tekom izračuna drevesa $T(G_2)$ izračunali maksimalni pretok in minimalni prerez med različnimi pari vozlišč kot v grafu G_1 in za drevesi $T(G_1)$ in $T(G_2)$ ni nujno, da bosta enaki.

Do raznolikosti dreves $T(G)$ za graf G lahko pride tudi zaradi obstoja

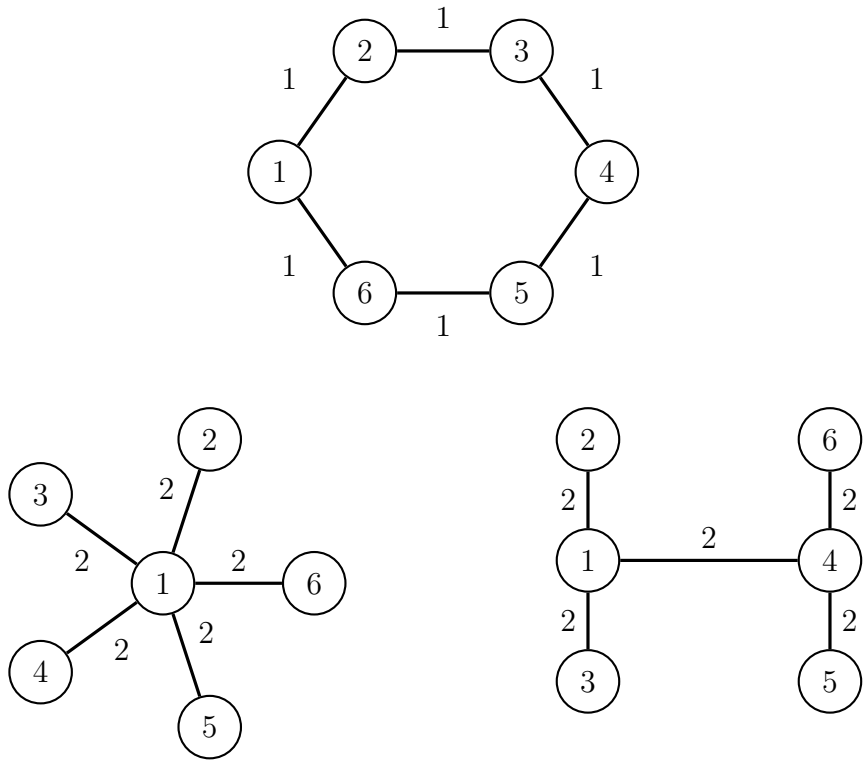
večih minimalnih prerezov med izbranima vozliščema s in t iz vreče B v grafu G, ki različno razdelijo vozlišča v vreče B_s in B_t. Glede na število vozlišč v vrečah B_s in B_t bomo definirali naslednja minimalna prereza:

1. Minimalni (S, T) prerez, ki razdeli vozlišča iz vreče B tako, da je število vozlišč v vreči B_s kar se da majhno — takšen prerez bomo imenovali *neuravnotežen minimalni prerez*,

2. Minimalni (S, T) prerez, ki razdeli vozlišča iz vreče B v vreči B_s in B_t tako, da je razlika v številu vozlišč med vrečama čim manjša — takšen prerez bomo imenovali *uravnotežen minimalni prerez*.

Na začetku izračuna imamo drevo T v obliki zvezde, v katerem ima vozlišče 0 stopnjo $n - 1$ in predstavlja *sredinsko vozlišče*. Vsa vozlišča grafa G se nahajajo v vreči z indeksom 0. Najprej si poglejmo obliko drevesa T tekom izvajanja algoritma pri izbiri neuravnoteženih minimalnih prerezov. Po izračunu vsakega minimalnega (S, T) prereza med vozliščema iz vreče B, bo množica vozlišč S vsebovala samo izbrano vozlišče s. Iz tega sledi, da se bo oblika drevesa ohranjala in dobimo drevo $T_1(G)$ v obliki zvezde z istim sredinskim vozliščem kot na začetku izračuna drevesa T. Sedaj si poglejmo kaj se dogaja z obliko drevesa T tekom izvajanja algoritma za izbiro minimalnih prerezov med katerimi je prvi minimalni prerez uravnotežen in ostalih $n-2$ minimalnih prerezov neuravnoteženih. Po izračunu minimalnega prereza bomo iz začetnega drevesa T dobili drevo, v katerem ima približno polovica vozlišč iz grafa G za soseda izbrano vozlišče s in preostala vozlišča za soseda vozlišče t. Za vsak naslednji izračunan minimalni prerez med izbranima vozliščema s in t se bo oblika drevesa ohranjala. Na koncu dobimo drevo $T_2(G)$ z dvema podgrafoma v obliki zvezd, ki sta med seboj povezana s prerezno povezavo med sredinskima vozliščema podgrafa.

Slika 3.9 prikazuje za graf 6-cikla pripadajoči drevesi $T_1(G)$ in $T_2(G)$, ki ju dobimo glede na omenjene izbire minimalnih prerezov tekom izračuna. V drevesu $T_2(G)$ ob odstranitvi prerezne povezave med sredinskima vozliščema $(1, 4)$ razdelimo drevo na dva podgrafa. Vozlišče 1 predstavlja sredinsko vo-



Slika 3.9: Za graf G na zgornji strani slike prikaz pripadajoči drevesi $T_1(G)$ in $T_2(G)$ glede na izbiro uravnoteženosti minimalnih prerezov tekom izračuna.

zlišče za podgraf na levi strani drevesa in vozlišče 4 za desni podgraf drevesa. Oznaki sredinskih vozlišč za posamezen podgraf sta najmanjši izmed preostalih vozlišč.

Grafa G_1 in G_2 sta izomorfna kadar obstaja preslikava $p : V(G_1) \to V(G_2)$, ki je bijektivna in $u\ G_1 v \Leftrightarrow p(u)\ G_2 p(v)$. V tem primeru se ohranja število povezav, stopnje vozlišč in število vozlišč. Kot vidimo na sliki 3.9 se stopnja vozlišč različnih dreves $T(G)$ za graf G ne ohranja, zato sta drevesi med seboj neizomorfni.

Poglavje 4

Uporaba Gomory-Hu dreves

V poglavju bomo obravnavali problem minimalnega k-prereza grafa G in omenili nekaj algoritmov za ta problem. Še posebej pa se bomo osredotočili na algoritem s katerim rešitev zgolj aproksimiramo in si pri reševanju pomagamo z izračunanim Gomory-Hu drevesom $T(G)$ ter s tem pridobimo na časovni zahtevnosti. Snov v poglavju je delno povzeta iz [8].

Minimalni k-prerez je kombinatorično optimizacijski problem, pri katerem iščemo množico povezav $E' \subseteq E$ grafa $G = (V, E, c)$, za katere ima graf $G - E'$ (vsaj) k povezanih komponent. Pri tem mora biti vsota prepustnosti povezav v množici E' kar se da majhna. Za vrednost $k = 1$ in kadar graf G vsebuje več kot k komponent je problem trivialno rešen, zato se bomo omejili na primere za $k \geq 2$ in problem obravnavali samo na grafih G, ki vsebujejo največ $k - 1$ komponent.

Za vrednost $k = 2$ govorimo o izračunu globalnega minimalnega prereza. To pomeni, da iščemo prerez predstavljen z množico povezav E', katerih vsota prepustnosti je najmanjša izmed vseh možnih minimalnih prerezov v grafu G. Za izračun globalnega minimalnega prereza lahko uporabimo Kargerjev algoritem, pri čemer obstaja verjetnost $p \geq \frac{2}{n^2}$, da bo izračunan prerez tudi globalni minimalni prerez. Omenili bomo samo, da je časovna zahtevnost algoritma $O(n^4 \cdot \log(n))$. Psevdokoda in analiza časovne zahtevnosti algoritma ter algoritem Karger in Stein z večjo verjetnostjo izračuna globalnega

minimalnega prereza in manjšo časovno zahtevnostjo so podrobneje opisani v [1].

Problem minimalnega k-prereza za vrednosti $k \geq 3$ spada med NP težke probleme. Prvotni algoritem za iskanje minimalnega k-prereza ima časovno zahtevnost $O(n^{k^2})$. Obstaja tudi algoritem z nekaj izboljšavami s časovno zahtevnostjo $O(n^{2 \cdot k})$. Takšni časovni zahtevnosti nista polinomski, saj je stopnja odvisna od k. Oba algoritma in bistvene razlike med njima so opisane v [9].

Psevdokoda 3 prikazuje aproksimacijski algoritem za reševanje minimalnega k-prereza grafa G z uporabo drevesa $T(G)$. Vhodni podatki so: graf $G = (V, E, c)$ in vrednost k. Izhodna podatka sta množica povezav Z ter vsota prepustnosti povezav w iz množice Z. Na začetku z uporabo Gusfieldovega algoritma opisanega v podrazdelku 3.1.2 izračunamo drevo $T(G) = (V_T, E_T, c_T)$. Po izračunu drevesa množico njegovih povezav E_T uredimo po prepustnostih povezav naraščajoče in izberemo $k - 1$ povezav z najmanjšo vrednostjo prepustnosti iz množice E_T. Posamezna izbrana povezava e_i, $i = 1, \ldots, k - 1$ predstavlja množico povezav $C(e_i)$ v grafu G. Množico povezav Z dobimo z unijo množic povezav $C(e_1), C(e_2), \ldots, C(e_k)$ in vrednost w kot vsoto prepustnosti povezav iz dobljene množice Z ter končamo z izračunom.

Časovna zahtevnost aproksimacijskega algoritma je v glavnem odvisna od časovne zahtevnosti Gusfieldovega algoritma za izračun drevesa $T(G)$. Vse ostalo je torej zanemarljivo in dobimo $O(n^4 \cdot m^2)$. Za polne grafe s številom povezav $m = O(n^2)$ dobimo časovno zahtevnost izračuna minimalnega k-prereza $O(n^4 \cdot n^4) = O(n^8)$.

Trditev 4.1. *V drevesu $T(G)$ posamezna povezava $e \in E_T$ predstavlja minimalni prerez z množico povezav $C(e)$ v grafu G. Če iz $T(G)$ odstranimo $k - 1$ povezav $e_1, e_2, \ldots, e_k$, v grafu G odstranimo izbranih $k - 1$ prerezov $C(e_1), C(e_2), \ldots, C(e_{k-1})$ in graf G razdelimo na vsaj k komponent.*

Dokaz. $V_1, V_2, \ldots, V_k$ so komponente, ki jih dobimo iz $T(G) - \{e_1, e_2, \ldots, e_k\}$. Če sta v_i in v_j poljubni vozlišči iz V_i in V_j, v grafu $G' = G - C(e_1) - C(e_2) -$

Algoritem 3 Minimalni k-prerez

Input: $G = (V, E, c), k$

Output: Z, w

1: $Z = \bigcup_{i=1}^{k-1} C(e_i)$

2: $w = \sum_{e \in Z} c(e)$

3: **return** Z, w

$\ldots - C(e_{k-1})$ ležita v različnih komponentah. Morebitna v_i-v_j pot v G' bi morala vsebovati vsaj eno izmed povezav $C(e_k)$, kjer je e_k povezava na v_i- v_j poti v $T(G)$. □

Izrek 4.1. *Naj bo G graf in Z^* optimalni k-prerez v grafu G z vrednostjo w^*. Nadalje, naj bosta Z, w rezultata Algoritma 3. Potem je*

$$w \leq (2 - \frac{2}{k}) \cdot w^*.$$

Z drugimi besedami, Algoritem 3 ima aproksimacijski faktor $2 - \frac{2}{k}$.

Dokaz. Množica povezav Z je po Trditvi 4.1 k-prerez grafa G. Potrebno je še dokazati, da je $c(Z) \leq (2 - \frac{2}{k}) \cdot c(Z^*)$.

Vemo, da $G - Z^*$ razpade na k komponent z množicami vozlišč $V_1, V_2, \ldots, V_k$. Z oznako Z_i označimo množico povezav med V_i in $V \setminus V_i$ v grafu G, z $c(Z_i)$ pa vsoto njihovih prepustnosti. Pri tem številčenje izberemo tako, da velja

$$c(Z_1) \leq c(Z_2) \leq \ldots \leq c(Z_k). \tag{4.1}$$

Posamezna povezava $e \in Z^*$ pripada natanko dvema izmed množic $Z_1, Z_2, \ldots, Z_k$, zato za vsoto prepustnosti povezav $c(Z_i)$, za $i = 1, \ldots, k$, velja zveza

$$c(Z_1) + c(Z_2) + \ldots + c(Z_k) = 2 \cdot c(Z^*). \tag{4.2}$$

Za vsako množico $V_1, V_2, \ldots, V_k$ z f_i označimo začetno povezavo na (eni od) najkrajših $V_i - V_k$ poti v drevesu $T(G)$. Krajišči povezave označimo z v_i, u_i, pri čemer je $v_i \in V_i$ in $u_i \notin V_i$. Povezave $f_1, f_2, \ldots, f_{k-1}$ zaradi lege na

najkrajših poteh med seboj ne sovpadajo. Ker prerez $C(f_i)$ loči vozlišči v_i in u_i in $f_i \in E_T$ velja

$$c(f_i) \leq c(Z_i). \tag{4.3}$$

Zdaj bomo ocenjevali vrednost izraza $w = c(Z)$.

Vsaka povezava $e \in Z$ pripada vsaj enemu od prerezov $C(e_i)$ in dobimo

$$w = c(Z) \leq \sum_{i=1}^{k-1} c(e_i),$$

povezave $e_1, e_2, \ldots, e_k$ imajo najmanjšo vrednost prepustnosti v $T(G)$, zato velja

$$\sum_{i=1}^{k-1} c(e_i) \leq \sum_{i=1}^{k-1} c(f_i).$$

Po neenačbi (4.3) dobimo

$$\sum_{i=1}^{k-1} c(f_i) \leq \sum_{i=1}^{k-1} c(Z_i)$$

in na koncu iz (4.2) in (4.1) izpeljemo oceno aproksimacijske konstante, torej velja

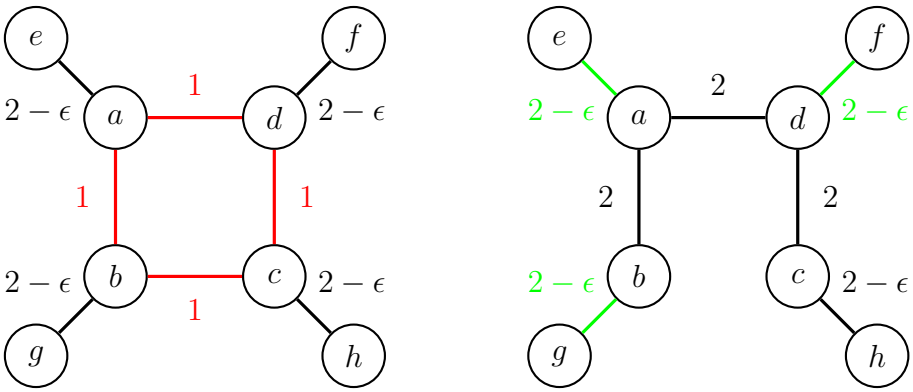
$$\sum_{i=1}^{k-1} c(Z_i) \leq 2 \cdot (1 - \frac{1}{k}) \cdot c(Z^*) = (2 - \frac{2}{k}) \cdot c(Z^*)$$

in tako končamo z dokazom. □

V nadaljevanju bomo opisali značilnosti grafa G za dokaz tesnosti aproksimacijske konstante. Pri tem dobimo približno rešitev, ki se približa zgornji meji aproksimacijske konstante. Za izbrano vrednost k definiramo graf G_k kot k-cikel z dodanimi k povezavami. Prepustnosti povezav na k-ciklu so vse enake 1, dodatne povezave pa so prepustnosti $2 - \epsilon$. Slika 4.1 na levi strani

prikazuje primer grafa G_k za $k = 4$. Drevo $T(G_k)$ bo torej vsebovalo vpeto poddrevo na vozliščih cikla grafa G_k s prepustnostmi povezav 2 in dodatne povezave s prepustnostmi $2 - \epsilon$. Primer drevesa $T(G_k)$ za $k = 4$ je prikazan na desni strani slike 4.1.

Aproksimacijski algoritem izbere $k - 1$ povezav z najmanjšo vrednostjo prepustnosti. V tem primeru imajo vse povezave vrednost prepustnosti enako $2 - \epsilon$ enot in dobimo $c(Z) = (k - 1) \cdot (2 - \epsilon)$. Optimalni k-prerez pa vsebuje izključno k povezav cikla v grafu G in dobimo $c(Z^*) = k$. Razmerje med izračunanima vrednostma se tako razlikuje za faktor $2 - \frac{2-\epsilon+\epsilon \cdot k}{k}$. Čim manjša je vrednost ϵ tem bližje smo aproksimacijski konstanti $2 - \frac{2}{k}$.



Slika 4.1: Prikaz grafa G_4 in pripadajočega drevesa $T(G_4)$. Povezave z rdečo barvo predstavljajo množico Z^* optimalnega 4-prereza grafa G in povezave z zeleno barvo množico Z 4-prereza. Čim manjša je vrednost ϵ, tem bolj se razmerje med $\frac{c(Z)}{c(Z^*)}$ približuje aproksimacijski konstanti $2 - \frac{2}{k}$.

Poglavje 5

Gomory-Hu drevesa za dinamične grafe

Do sedaj smo obravnavali izračun drevesa $T(G)$ za statičen graf G. V tem poglavju si bomo pogledali na kakšen način lahko učinkovito vzdržujemo že izračunano drevesno strukturo v primeru majhnih sprememb grafa G. Spremembe vključujejo spremembo prepustnosti na povezavi in dodajanje ali brisanje vozlišča. Graf G v katerem izvedemo minimalno spremembo bomo označevali z G^U. Pri spremembi prepustnosti lahko izbrano povezavo odstranimo ali dodamo. Zmanjšanje prepustnosti izbrane povezave $e \in E$ na vrednost 0 smatramo kot odstranitev povezave in povečanje prepustnosti med dvema izbranima vozliščema iz vrednosti 0 na določeno pozitivno vrednost kot dodajanje povezave. Snov v poglavju je deloma povzeta iz virov [7, 6].

Graf G v katerem smo povečali prepustnost na povezavi ali dodali novo povezavo ali vozlišče bomo označevali z $G^+ = (V^+, E^+, c^+)$, v nasprotnem primeru pa z $G^- = (V^-, E^-, c^-)$. Proti koncu poglavja bomo spoznali algoritme za izračun drevesa $T(G^U)$, jih časovno in prostorsko analizirali ter teoretično časovno zahtevnost primerjali z ocenjeno.

Operacijo odstranitve vozlišča p lahko modeliramo z zmanjšanjem vrednosti prepustnosti povezav, ki povezujejo vozlišče p s preostalimi vozlišči v grafu G na vrednost 0 in na koncu vozlišče p odstranimo kot izolirano

vozlišče. V primeru dodajanja vozlišča v graf G naredimo ravno obratno, postopoma dodajamo željene povezave z izbranimi vrednostmi prepustnosti med novo vstavljenim vozliščem in ostalimi vozlišči v grafu G. V grafu G in drevesu $T(G)$ so izolirana vozlišča povezana preko povezav s kapaciteto 0 s preostalimi vozlišči. Tako lahko enostavno dodamo ali odstranimo izolirano vozlišče iz grafa G in drevesa $T(G)$ ter dobimo graf G^U in pripadajoče drevo $T(G^U)$. Pri teh dveh operacijah torej nimamo opravka z računanjem maksimalnega pretoka in minimalnega prereza v grafu G^U. V nadaljevanju si bomo pogledali operacijo spreminjanja prepustnosti na izbrani povezavi v grafu G, za katero so potrebni ponovni izračuni maksimalnih pretokov in minimalnih prerezov v grafu G^U za izračun drevesa $T(G^U)$.

Glavna skrb dinamičnega izračuna Gomory-Hu drevesa je v tem, da želimo učinkovito odločiti, katere povezave v drevesu $T(G)$, minimalni prerezi grafa G, pripadajo novemu drevesu $T(G^U)$ in so torej tudi minimalni prerezi v spremenjenem grafu G^U.

V nadaljevanju si bomo pogledali katere prereze v drevesu $T(G)$ lahko zagotovo uporabimo za izgradnjo drevesa $T(G^U)$. Na povezavi $e = (b, d)$ v grafu G spremenimo vrednost prepustnosti. Lema 5.1 opisuje, katere povezave drevesa $T(G)$ pripadajo drevesu $T(G^U)$.

Lema 5.1. *Naj bo graf G, (b, d) njegova povezava s prepustnostjo c in P_{bd} pot med vozliščem b in d v drevesu $T(G)$. Denimo, da prepustnost povezave (b, d) spremenimo za vrednost $\Delta > 0$, potem z G^+ označimo graf, v katerem smo prepustnost povezave (b, d) povečali za Δ in z G^- graf, v katerem smo prepustnost povezave (b, d) zmanjšali za vrednost Δ. Različne povezave v drevesu $T(G)$ predstavljajo minimalne prereze za graf G^U:*

1. *Povezave v drevesu $T(G)$, ki ne ležijo na poti P_{bd} predstavljajo minimalne prereze za graf G^+.*

2. *Povezavam v drevesu $T(G)$, ki ležijo na poti P_{bd}, zmanjšamo prepustnost za vrednost Δ in tako predstavljajo minimalne prereze za graf G^-.*

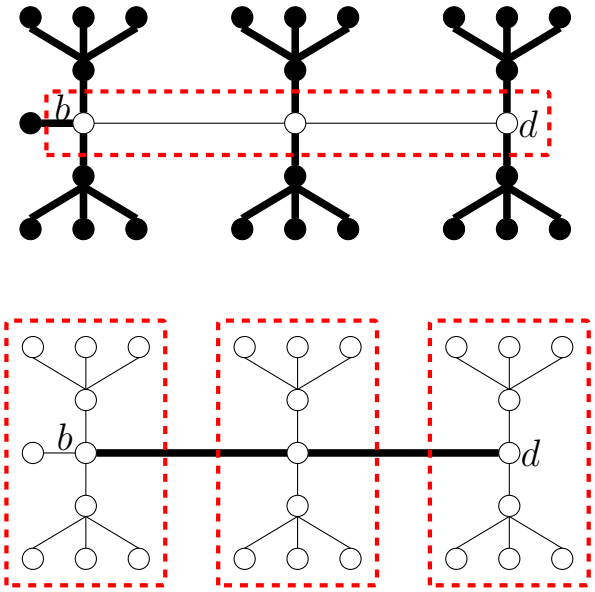
Dokaz. Povezave, ki sestavljajo pot P_{bd} v drevesu $T(G)$ so edine povezave, ki porodijo prereze grafa G za ločitev vozlišč b in d. Vsak izmed teh prerezov vsebuje povezavo (b, d). Torej te povezave predstavljajo prereze med izbranima vozliščema, za katere se spremeni prepustnost za vrednost Δ v grafu $T(G)$. V primeru, da kapaciteto povezave (b, d) v grafu G zmanjšamo, potem se kapaciteta posamezne povezave $(u, v) \in P_{bd}$ zmanjša za največ Δ. Zato je (u, v) minimalni u-v prerez v grafu G^-, za katerega velja $c^-(u, v) = c_T(u, v) - \Delta = c_*(u, v) - \Delta$. Prerezi katerih vrednost se ne spremeni po povečanju kapacitete na povezavi (b, d) ostanejo minimalni prerezi za graf G^+. □

Pri gradnji drevesa $T(G^U)$ si bomo pomagali z *vmesnim drevesom* $T_* = (V_*, E_*, c_*)$. Začetno vmesno drevo je kar enako $T(G)$. Povezave vmesnega drevesa razdelimo na dve skupini, E_*^F in E_*^T. V množici E_*^F so tiste povezave drevesa T_*, ki predstavljajo minimalne prereze v grafu G^U, povezave iz te množice bomo imenovali *debele*, tako jih bomo tudi prikazovali. Posamezna debela povezava v drevesu T_* predstavlja prerez, ki mu bomo pravili *veljaven minimalni prerez*. V množici E_*^T pa so tiste povezave drevesa T_*, za katere še ne vemo, ali predstavljajo minimalne prereze v grafu G^U. Imenovali jih bomo *tanke* povezave.

Algoritem za dinamičen izračun Gomory-Hu drevesa $T(G^U)$ bo korakoma večal število debelih povezav, dokler ne bodo vse povezave vmesnega drevesa T_* debele — takrat z algoritmom končamo in drevo $T(G^U)$ je zadnje izračunano vmesno drevo T_*.

Slika 5.1 prikazuje vmesno drevo T_{*1} na zgornji in T_{*2} na spodnji strani slike, ki ju dobimo na začetku postopka izračuna drevesa $T(G^U)$. Drevo T_{*1} predstavlja začetno drevo T_* za graf G^+ in drevo T_{*2} začetno drevo T_* za graf G^-. Posamezni drevesi vsebujeta veljavne minimalne prereze glede na Lemo 5.1 in jih lahko uporabimo za izgradnjo drevesa $T(G^U)$. Drevo T_{*1} predstavlja množico prerezov za ponovno uporabo v primeru povečanja kapacitete na povezavi (b, d) v grafu G. V tem primeru lahko vse povezave, ki ne sestavljajo poti P_{bd} ponovno uporabimo, zato so označene kot debele.

Ostale povezave so označene kot tanke in minimalne prereze med temi vozlišči moramo ponovno izračunati. Drevo T_{*2} predstavlja množico prerezov za ponovno uporabo v primeru zmanjšanja kapacitete na povezavi (b, d) v grafu G. V tem primeru so povezave na poti P_{bd} debele in kapaciteta vsake debele povezave bo glede na začetno drevo T_{*2} v končno izračunanem drevesu $T_{*2} = T(G^-)$ zmanjšana za vrednost Δ.



Slika 5.1: Prikaz dveh začetnih vmesnih dreves izračuna drevesa $T(G^U)$. Drevo zgoraj za graf G^+ in spodaj za graf G^-. Debele povezave v obeh drevesih predstavljajo množico veljavnih minimalnih prerezov — povezav ki pripadajo $T(G^U)$. Prekinjena rdeča črta opisuje množico tankih povezav.

Sedaj se bomo usmerili na veljavne minimalne prereze, ki so predstavljeni z debelimi povezavami na sliki 5.1. Napisali bomo pravila, s katerimi si pomagamo pri iskanju teh prerezov in dokazali njihovo pravilnost.

Lema 5.2. *Če je povezava (b, d) na novo vstavljena v graf G s prepustnostjo $c(b, d) = \Delta$ ali se prepustnost te povezave poveča za vrednost Δ, potem vsak b-d prerez v grafu G postane veljaven v grafu G^+ z vrednostjo prepustnosti $c_T(b, d)$.*

Dokaz. Ob ponovni uporabi veljavnega minimalnega b-d prereza v postopku izračuna drevesa $T(G^+)$, kot smo opisali pri Gomory-Hu algoritmu za izračun drevesa, ločimo vozlišči b in d. Tako para vozlišč b, d ne moremo več uporabiti za izračun prereza v eni izmed naslednjih iteracij izračuna. To pomeni, da lahko ponovno uporabimo le en minimalni b-d prerez, kljub temu da lahko med vozliščema b in d v drevesu $T(G)$ obstaja več minimalnih prerezov na poti P_{bd}. Skupaj z lemo 5.2 in 5.3 lahko ponovno uporabimo celotno drevo $T(G)$ v primeru obstoječega mostu (b, d) v grafu G s povečanjem kapacitete na tej povezavi za $\Delta = c^U(b, d) - c(b, d)$. □

Lema 5.3. *Za prerezne povezave $e_p \in E$ v grafu G velja, da se ista povezava z enako vrednostjo prepustnosti nahaja tudi v drevesu $T(G)$, dobimo $c(e) = c_T(e) > 0$.*

Dokaz. Da se bo povezava (u, v) nahajala v grafu G in $T(G)$ mora veljati naslednje. Minimalni u-v prerez v grafu G je samo eden (razen če je G nepovezan), zato nobeden izmed minimalnih prerezov med vozliščema x in y, pri katerem vozlišči nista para u in v, ne bo ločil vozlišč u in v.

V nasprotnem primeru, če povezave (u, v) ni v drevesu $T(G)$, potem minimalni u-v prerez predstavlja druga povezava (w, z), kjer se vozlišče w nahaja na poti P_{uz}. Zato vsaka povezava z minimalno vrednostjo kapacitete pretoka na poti P_{uw} v $T(G)$ predstavlja minimalni u-w prerez, ki bo ločil vozlišči u in v, kljub temu da par vozlišč u in w ni enak paru u in v. S tem končamo s protislovjem in z dokazom. □

Lema 5.4. *Kadar se vozlišči b in d nahajata v različnih povezanih komponentah grafa G in dodana povezava (b, d) predstavlja nov most v grafu G^+, potem dobimo drevo $T(G^+)$ tako, da povezavo (b, d) s prepustnostjo Δ dodamo v drevo $T(G)$.*

Dokaz. Naj bo povezava (b, d) nov most, potem se vozlišči b in d nahajata v različnih povezanih komponentah v grafu G in pot P_{bd} v drevesu $T(G)$ vsebuje vsaj eno povezavo katere vrednost prepustnosti je enaka 0. Nadomestitev te povezave z drugo povezavo lahko izvedemo, saj ne pride do cikla v drevesu

in dobimo drevo $T(G^+)$. Po spremembi prepustnosti na povezavi (b, d) ali odstranitvi iz grafa G je vzdrževanje drevesne strukture enostavno, saj na isti povezavi v $T(G)$ spremenimo vrednost prepustnosti. □

Lema 5.5. *Naj bo $(U, V \setminus U)$ minimalni u-v prerez v grafu G^-, pri čemer za vsako vozlišče x na poti P_{bd} velja $x \in V \setminus U$ in naj bo povezava $(g, h) \in E_T$ z $g, h \in U$. Potem povezava (g, h) predstavlja minimalni prerez v grafu G^- za prereze med vsemi pari vozlišč, ki se nahajajo v množici U prereza $(U, V \setminus U)$.*

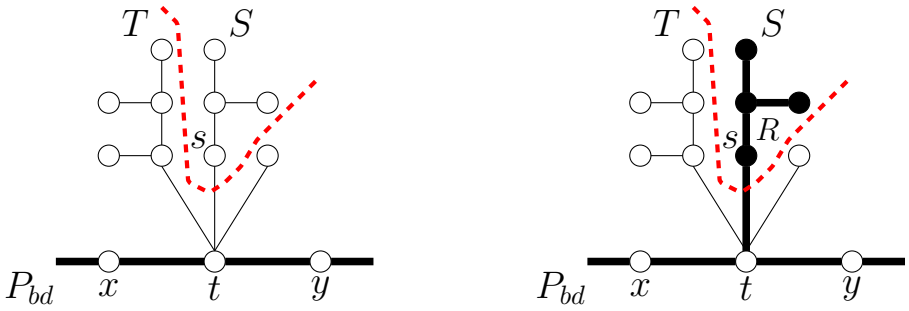
Dokaz. Za poljuben minimalni $(U, V \setminus U)$ prerez vozlišča v množici U ločimo od ostalih vozlišč grafa in vsak minimalni g-h prerez, $g, h \notin U$ lahko preoblikujemo tako, da vozlišč v množici U ne ločimo med seboj. Vzemimo, da v grafu G^- obstaja minimalni g-h prerez, katerega prepustnost je manjša od prepustnosti na povezavi (g, h) v drevesu $T(G)$. Tak minimalni g-h prerez bo ločil vozlišči b in d v množici $V \setminus U$ in bo prišlo do križanja med prerezi, zato ga moramo preoblikovati tako, da vozlišči b in d ne ločimo. □

Lema (5.6) opisuje, kdaj med postopkom izračuna drevesa $T(G^-)$ v drevesu T_* pride do križanja med prerezi.

Lema 5.6. *Naj bo T_* drevo za graf G^-, v katerem debele povezave na poti P_{bd} predstavljajo veljavne minimalne prereze, in je povezava (s, t) tanka z vozliščem t na poti P_{bd}. Naj bosta x in y sosedi vozlišča t vzdolž poti P_{bd} in e_x, e_y povezavi med t in x oziroma t in y. Povezavi e_x in e_y sta v drevesu T_* in predstavljata minimalna prereza v grafu G^-. Naj L označuje manjšo od prepustnosti povezav e_x in e_y. V primeru, da je vrednost L manjša ali enaka vrednosti prepustnosti povezave (s, t), potem povezava (s, t) predstavlja veljaven minimalni prerez za vozlišči s in t v grafu G^-.*

Dokaz. Povezave na poti P_{bd} predstavljajo podmnožico S' veljavnih minimalnih prerezov za graf G^-. Po lemi 3.1 nobeden naslednje izračunan minimalni u-v prerez v grafu G^- z manjšo vrednostjo kapacitete v primerjavi z vrednostjo $c_T(u, v)$ ne more ločiti vozlišči b in d. V nasprotnem primeru bi prišlo do križanja minimalnih prerezov v G^-. □

Sliki 5.2 in 5.3 prikazujeta spremembe v drevesu T_* med izračunom drevesa $T(G^-)$, glede na vrednost prepustnosti minimalnega s-t prereza v grafu G^- v primerjavi z L.



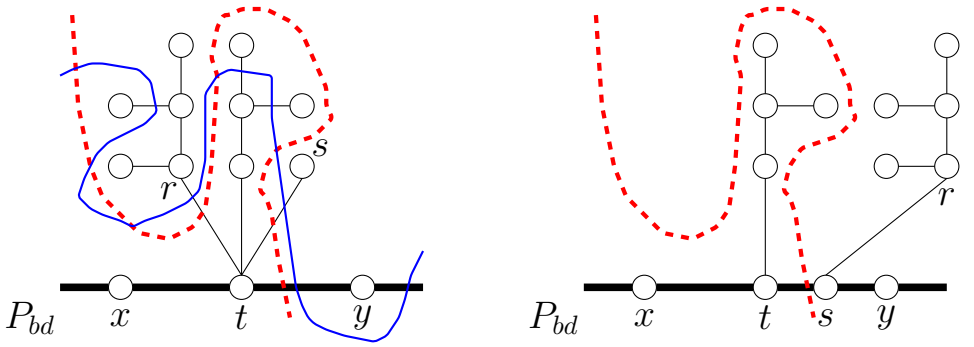
Slika 5.2: Prikaz dveh vmesnih dreves T_* za graf G^-. Na sliki levo drevo prikazuje veljaven minimalni s-t prerez v grafu G^-, zato lahko celotno poddrevo R s korenom vozlišča s, vključno s povezavo (s, t), odebelimo. Horizontalne debele povezave v obeh drevesih ležijo na poti P_{bd}.

5.1 Algoritmi za izračun Gomory-Hu dreves za dinamične grafe

5.1.1 Zmanjšanje prepustnosti na povezavi

Opisali bomo delovanje algoritma za izračun drevesa $T(G^-)$ za graf G^- po zmanjšanju vrednosti prepustnosti na poljubni povezavi (b, d) v grafu G, glej psevdokodo Algoritma 4 in 5.

Vhodni podatki za izračun Gomory-Hu drevesa za graf G^- po zmanjšanju prepustnosti na izbrani povezavi (b, d) v grafu G so naslednji: graf $G = (V, E, c)$, drevo $T(G) = (V_T, E_T, c_T)$, vozlišči b in d povezave s spremembo prepustnosti in za izbrano povezavo (b, d) vrednost nove prepustnosti $c^-(b, d)$. Zadnji vhodni podatek je struktura *neighbours* za hranjenje sosedov za posamezno vozlišče v $T(G)$. Izhodni podatek je drevo $T(G^-)$.



Slika 5.3: Prikaz vmesni drevesi T_{*1} in T_{*2} med postopkom računanja drevesa za graf G^-. V drevesu T_{*1} imamo prereza označena z rdečo črtkasto in modro črto. Modra črta prikazuje razdelitev vozlišč glede na minimalni s-t prerez, ki smo ga izračunali v grafu G^-. Z rdečo črtkasto črto je v obeh drevesih prikazan prerez, v katerega preoblikujemo minimalni prerez označen z modro črto. Pri tem poddrevo s korenskim vozliščem r v celoti ohranimo, ga odcepimo od vozlišča t in dodamo vozlišču s za soseda ter na poti P_{bd} vstavimo vozlišče s za soseda vozlišču t. Dobljeno povezavo (s,t) označimo debelo in dobimo drevo T_{*2}. Debele povezave v obeh drevesih predstavljajo pot P_{bd}.

Z Δ označimo razliko med staro in novo vrednostjo prepustnosti na (b,d), pri tem velja $\Delta = c_T(b,d) - c^-(b,d)$. V vrstici 2 preverimo ali povezava (b,d) predstavlja most v grafu. V tem primeru zmanjšamo prepustnost na izbrani povezavi v drevesu $T(G)$ za vrednost Δ in dobimo drevo $T(G^-)$ ter končamo z izračunom. V nasprotnem primeru zgradimo vmesno drevo T_* prikazano na sliki 5.1 zgoraj.

Drevo T_* izračunamo v dveh korakih po naslednjem postopku. Najprej v vrstici 5 z uporabo BFS algoritma izračunamo vozlišča v drevesu $T(G)$, ki sestavljajo pot P_{bd}. Nato v vrsticah 6–10 za povezave iz množice E_T preverimo ali se izbrana povezava nahaja na poti P_{bd}. Če se nahaja jo dodamo v množico debelih povezav E_*^F, sicer pa jo dodamo v množico tankih povezav E_*^T. V vrstici 11 z uporabo algoritma urejanja s kopico uredimo povezave

v E_*^T po vrednostih prepustnosti padajoče. Algoritem se nato izvaja dokler množica povezav E_*^T ni prazna.

V vsaki iteraciji v vrsticah 13–16 izberemo iz množice E_*^T povezavo (s, t) z največjo vrednostjo prepustnosti, pri čemer mora veljati da se vozlišče t izbrane povezave nahaja na poti P_{bd}. V vrstici 17 definiramo množico vozlišč N_t za hranjenje trenutno sosednih vozlišč vozlišču t na poti P_{bd}. Izračunamo jo tako, da za vozlišče $x \in neighbours[t]$ preverimo ali je vsebovano v množici vozlišč P_{bd}. V primeru da je vsebovano, dodamo x v množico N_t. V vrstici 21 izračunamo vrednost L tako, da za posamezne pare vozlišč $x \in N_t$ in v preverimo vrednost prepustnosti dobljene (x, v) povezave in izberemo najmanjšo vrednost prepustnosti.

Če je vrednost prepustnosti L večja ali enaka od vrednosti prepustnosti izbrane povezave (s, t) ali se ista povezava z enako vrednostjo prepustnosti nahaja v grafu G, v vrsticah 23–27 dodamo povezavo (s, t) v množico E_*^F in obhodimo poddrevo R s korenom vozlišča s, kot ga prikazuje slika 5.2. Vse povezave iz poddrevesa R dodamo v množico E_*^F, jih odstranimo iz množice E_*^T in se vrnemo na vrstico 12 ter začnemo z novo iteracijo izračuna. V nasprotnem primeru v vrsticah 28–38 izračunamo maksimalni pretok in minimalni prerez med vozliščema s in t z uporabo Edmonds-Karpovega algoritma. Povezavo (s, t) odebelimo.

Nato preverimo ali je vrednost prepustnosti $c_T(s, t)$ v drevesu $T(G)$ enaka izračunani vrednosti maksimalnega pretoka $|f|$ med vozliščema s in t v grafu G^-. Če sta vrednosti enaki, ponovno obhodimo poddrevo R s korenskim vozliščem s. Vse povezave poddrevesa odebelimo in začnemo z novo iteracijo izračuna.

V primeru, da je vrednost prepustnosti na povezavi (s, t) v drevesu $T(G)$ različna od vrednosti $|f|$, preoblikujemo drevo T_* kot je prikazano na sliki 5.3. To pomeni, da v drevesu T_* dodamo vozlišče s za soseda vozlišču t in dobimo povezavo (s, t) na poti P_{bd}. Povezavo dodamo v množico debelih povezav E_*^F in ji nastavimo prepustnost na vrednost $|f|$.

V množici N_t hranimo vsa trenutno sosedna vozlišča vozlišču t v drevesu

T_* in nato za vozlišča x, ki so vsebovana v množici N_t in niso vsebovana v P_{bd} ter ležijo na isti strani izračunanega minimalnega (S, T) prereza kot vozlišče s, odcepimo od vozlišča t in jih dodamo vozlišču s za sosedna ter končamo z iteracijo izračuna.

Časovna zahtevnost algoritma

Vsaka povezava, ki ni na poti P_{bd} v drevesu T_* na zgornji strani slike 5.1, je v začetku tanka in za vsako takšno povezavo bomo morali izračunati maksimalni pretok in minimalni prerez. Število izračunov maksimalnih pretokov in minimalnih prerezov za izračun drevesa $T(G^-)$ torej znaša $n - 1 - |P_{bd}|$.

Pot P_{bd} poiščemo v linearnem času $O(n+m)$, ki pa je lahko dolžine zgolj 1. Hkrati povezave na poti označimo za debele, tanke povezave pa v času $O(n \cdot \log(n))$ uredimo.

Dokler množica E_*^T ni prazna, v vsaki iteraciji iz nje izberemo povezavo (s, t) z največjo vrednostjo prepustnosti in vozliščem t na poti P_{bd}. V najslabšem primeru bo po vsaki izbrani povezavi izračunana vrednost prepustnosti L na povezavah iz množice E_*^F s skupnim vozliščem t na poti P_{bd}, manjša od vrednosti prepustnosti izbrane povezave. Torej bomo morali v vsaki iteraciji izračunati maksimalni pretok in minimalni prerez v grafu G^-. Časovno zahtevnost algoritma za izračun maksimalnega pretoka in minimalnega prereza smo analizirali v podpoglavju 2.4 in znaša $O(n^2 \cdot m^2)$. Skupna časovna zahtevnost algoritma torej znaša $O((n+m) + (2 \cdot n) + (n \cdot \log(n)) + (n \cdot (n^2 \cdot m^2))) = O(n^3 \cdot m^2)$.

V grafih $\mathcal{K}_n$ je število povezav $m = O(n^2)$, zato časovno zahtevnost algoritma ocenimo z $O(n^3 \cdot n^4) = O(n^7)$ in za grafe $\mathcal{R}_n$ ter $\mathcal{M}_n$ število povezav $m = O(n)$ in dobimo $O(n^3 \cdot n^2) = O(n^5)$.

V nadaljevanju bomo opisali postopek testiranja algoritma v praksi. Dobljene časovne zahtevnosti bomo primerjali s časovnimi zahtevnostmi Gusfieldovega algoritma za posamezne tipe grafov, opisanih v podrazdelku 3.1.2.

Algoritem smo testirali na enak način kot Gusfieldov algoritem. Razlika je bila le v tem, da smo na vsakem koraku za posamezen tip grafa na n

Algoritem 4 Zmanjšanje prepustnosti na povezavi 1. del

Input: $G = (V, E, c), T(G) = (V_T, E_T, c_T), b, d, c(b, d), c^-(b, d)$, $neighbours[\{\}]$

Output: $T(G^-)$

1: $\Delta = c(b, d) - c^-(b, d), T_* = (V_*, E_*^F, E_*^T, c_*)$
2: **if** $c(b, d) == c_*(b, d)$ **then**
3: $\quad c_T(b, d) = c_T(b, d) - \Delta$
4: $\quad$ **return** $T(G)$
5: $P_{bd} = BFS(T(G), b, d)$
6: **for** $(u, v) \in E_T$ **do**
7: $\quad$ **if** $u, v \in P_{bd}$ **then**
8: $\quad\quad E_*^F = E_*^F \cup (u, v)$
9: $\quad$ **else**
10: $\quad\quad E_*^T = E_*^T \cup (u, v)$
11: $HeapSort(E_*^T)$
12: **while** $E_*^T \neq \emptyset$ **do**
13: $\quad$ **for** $(u, v) \in E_*^T$ **do**
14: $\quad\quad$ **if** $v \in P_{bd}$ **then**
15: $\quad\quad\quad (s, t) = (u, v)$
16: $\quad\quad\quad$ **break**
17: $\quad N_t = \{\}$
18: $\quad$ **for** $x \in neighbours[t]$ **do**
19: $\quad\quad$ **if** $x, t \in P_{bd}$ **then**
20: $\quad\quad\quad N_t = N_t \cup \{x\}$
21: $\quad L = \min_{x \in N_t}\{c_*(x, t)\}$
22: $\quad$ **if** $L \geq c_T(s, t)$ || $(s, t) \in E$ && $c_T(s, t) == c(s, t)$ **then**
23: $\quad\quad E_*^F = E_*^F \cup \{(s, t)\}, E_*^T = E_*^T \setminus \{(s, t)\}$
24: $\quad\quad R = SUBTREE(S \setminus t, root = s)$
25: $\quad\quad$ **for** $(o, p) \in R$ **do**
26: $\quad\quad\quad E_*^F = E_*^F \cup \{(o, p)\}, E_*^T = E_*^T \setminus \{(o, p)\}$

Algoritem 5 Zmanjšanje prepustnosti na povezavi 2. del

Continue
$|f|, S, T = Edmonds - Karp(G^-, s, t)$
$E_*^F = E_*^F \cup \{(s,t)\}, E_*^T = E_*^T \setminus \{(s,t)\}$
if $c_T(s,t) == |f|$ **then**
Go to line 13.
$P_{bd} = P_{bd} \cup (u,v), E_*^F = E_*^F \cup \{(u,v)\}$
$c_*(s,t) = |f|$
$N_t = neighbours[t]$
for $x \in N_t$ **do**
if $S[x] == true$ && $x \neq s$ && $(x,t) \in E_*^T$ **then**
$temp = c_*(x,t), c_*(x,t) = 0, c_*(x,s) = temp$
$E_*^F = E_*^F \setminus \{(x,t)\}, E_*^T = E_*^T \cup \{(x,s)\}$
return T_*

vozliščih stokrat izbrali naključno povezavo in ji zmanjšali prepustnost na naključno vrednost med 1 in $c(e)$. Za dobljen graf G^- smo nato izračunali pripadajoče drevo $T(G^-)$.

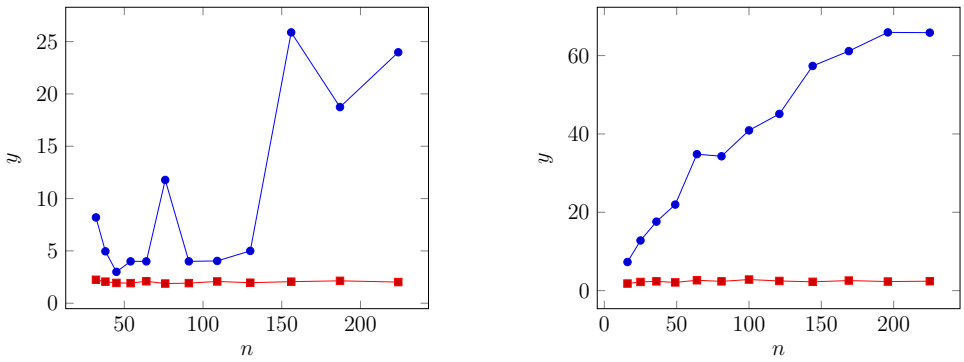
Izračunana regresijska premica iz logaritmiranih časov izvajanja algoritma v sekundah $\log(\overline{t_n(s)})$ glede na logaritmirano vrednost števila vozlišč $\log(n)$ za grafe $\mathcal{K}_n$ je $3.944 \cdot n - 7.831$. Dobimo funkcijo za časovno zahtevnost $f_1(n) = 10^{-7.831} \cdot n^{3.944}$. Časovno zahtevnost algoritma v praksi lahko ocenimo kot kvartično. Stopnja ocenjene časovne zahtevnosti je enaka kot pri analizi Gusfieldovega algoritma za polne grafe G, za katera so drevesa $T(G)$ tipično zvezde. V takih drevesih je pot P_{bd} dolžine 1 ali 2, zato pri številu izračunov maksimalnih pretokov in minimalnih prerezov prihranimo kvečjemu 1 izračun.

Za zmanjšanje prepustnosti na povezavah v grafih $\mathcal{R}_n$ smo iz dobljenih rezultatov meritev dobili izračunano regresijsko premico $3.051 \cdot n - 7.284$. Dobimo funkcijo za časovno zahtevnost $f_2(n) = 10^{-7.284} \cdot n^{3.051}$. Za grafe $\mathcal{M}_n$ smo dobili regresijsko premico $3.320 \cdot n - 8.129$ in funkcijo za časovno

zahtevnost algoritma $f_3(n) = 10^{-8.129} \cdot n^{3.320}$. Za obe vrsti grafov znaša ocenjena časovna zahtevnost algoritma $O(n^4)$. Slika 5.4 za oba tipa grafov na n vozliščih prikazuje povprečno vrednost premera drevesa $T(G)$ in povprečno dolžino poti P_{bd}. Povprečne dolžine poti P_{bd} neodvisno od premerov dreves $T(G)$ nihajo okoli vrednosti 2 in iz tega smo za izračun drevesa $T(G^-)$ prihranili na izračunu enega ali dveh maksimalnih pretokov in minimalnih prerezov. Število potrebnih izračunov pretokov še zmeraj znaša $O(n)$. V izračunanih regresijskih premicah za posamezne tipe grafov algoritma za zmanjšanje prepustnosti na povezavi so vrednosti ocene polinomske stopnje višje od vrednosti Gusfieldovega algoritma. Izkaže se torej da je algoritem za zmanjšanje prepustnosti v povprečju časovno počasnejši od Gusfieldovega algoritma. Do tega pride tudi zaradi izračuna drevesa T_* na začetku in urejanja povezav v E_*^T s kopico po prepustnostih padajoče. Glede na povprečne vrednosti poti P_{bd} v T_* in premerov samih dreves $T(G)$ lahko povemo še nekaj o njihovi obliki.

V podpoglavju 3.2 smo glede na izbiro uravnoteženosti minimalnih prerezov dobili dve različni drevesi $T(G)$ za graf v obliki d-cikla. Iz tega lahko trdimo, da smo tekom računanja drevesa $T(G)$ z uporabo Gusfieldovega algoritma večinoma dobili neuravnotežene minimalne prereze. V drevesu imamo torej večino vozlišč grafa G povezanih v zvezdo in preostanek vozlišč med seboj povezanih verižno, pri čemer je eno izmed vozlišč v verigi sosedno z vozliščem iz zvezde.

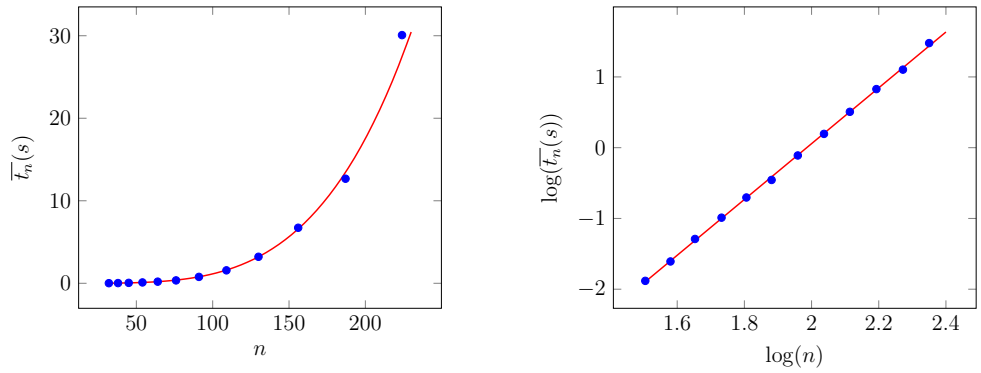
Slike 5.5, 5.6 in 5.7 prikazujejo rezultate meritev za algoritem zmanjšanje prepustnosti na povezavi za posamezne tipe grafov. Na posamezni sliki imamo dva grafa. V levem grafu imamo z modro prikazane merilne točke za povprečni čas izračuna drevesa $T(G^-)$ za določen tip grafa na n vozliščih. Na desnem grafu imamo iz merilnih točk označenih z isto barvo, iz levega grafa izračunane logaritmirane vrednosti in iz teh vrednosti dobljeno regresijsko premico označeno z rdečo barvo. Iz regresijske premice dobimo funkcijo prikazano na levem grafu, označeno z rdečo barvo.



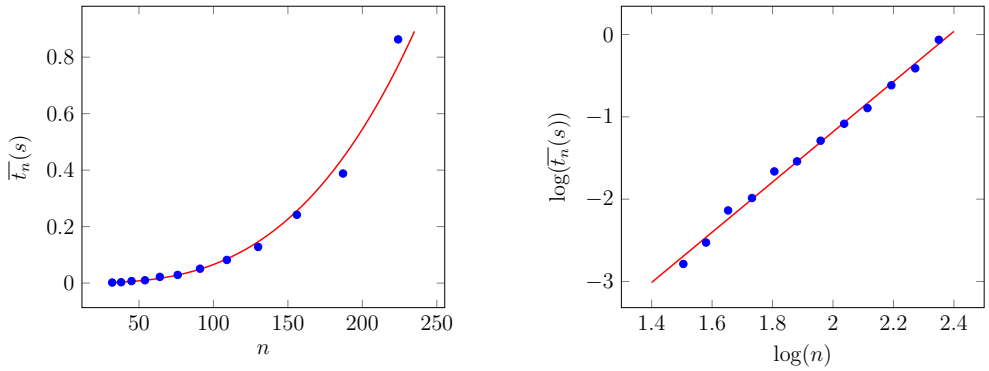
Slika 5.4: Funkcija označena z rdečo barvo prikazuje povprečno dolžino poti $\overline{P_{bd}}$ v izračunanih drevesih $T(G^U)$ na n vozliščih v posameznih korakih. Funkcija označena z modro barvo za izračunana drevesa v vsakem koraku prikazuje povprečno vrednost njihovega premera. Na levi strani imamo prikaz meritev za grafe $\mathcal{R}_n$ in na desni za grafe $\mathcal{M}_n$.

Prostorska zahtevnost algoritma

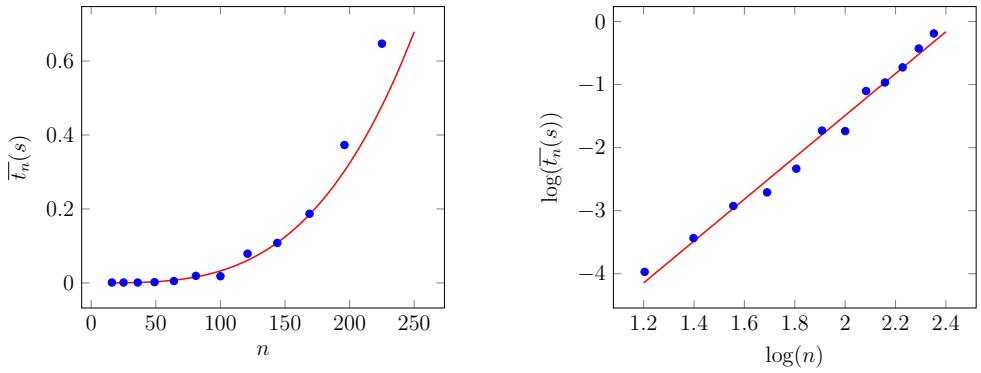
Za izračun drevesa $T(G^-)$ hranimo drevo $T(G)$ v obliki matrike sosednosti s prostorsko zahtevnostjo $O(n^2)$. Poleg te matrike uporabimo dodatno matriko za hranjenje logične vrednosti o debelini posamezne povezave v drevesu, katere prostorska zahtevnost znaša $O(n^2)$. Sosede posameznega vozlišča v T_* hranimo v dvodimenzionalni tabeli s prostorsko zahtevnostjo $O(n^2)$. Množica povezav E_*^T bo vsebovala $O(n)$ povezav in za ureditev po vrednostih prepustnosti padajoče uporabimo algoritem urejanje s kopico s prostorsko zahtevnostjo $O(1)$. Dokler množica povezav E_*^T ni prazna, v vsaki iteraciji glede na vrednost prepustnosti med trenutno izbrano povezavo in vrednostjo L, se sprehodimo po poddrevesu R in dodamo povezave v množico povezav E_*^F in jih odstranimo iz množice E_*^T ali pa izračunamo maksimalni pretok in minimalni prerez med izbranima vozliščema v grafu G^-. V prvem primeru za obhod po poddrevesu uporabimo BFS algoritem s prostorsko zahtevnostjo $O(n)$ za hranjenje vozlišč grafa G v vrsti za obisk. V drugem primeru za izračun maksimalnega pretoka in minimalnega prereza uporabimo



Slika 5.5: Prikaz rezultatov meritev izračuna dreves $T(G^-)$ za grafe $\mathcal{K}_n$.



Slika 5.6: Prikaz rezultatov meritev izračuna dreves $T(G^-)$ za grafe $\mathcal{R}_n$.



Slika 5.7: Prikaz rezultatov meritev izračuna dreves $T(G^-)$ za grafe $\mathcal{M}_n$.

Edmonds-Karpov algoritem, katerega prostorsko zahtevnost smo obravnavali v podpoglavju (2.4.2) in znaša $O(n^2)$. Prostorska zahtevnost algoritma za izračun drevesa $T(G^-)$ je torej vsota omenjenih vrednostih in dobimo $O(n^2 + n^2 + n^2 + n + n^2) = O((4 \cdot n^2) + n) = O(n^2)$.

5.1.2 Povečanje prepustnosti na povezavi

Algoritem za izračun drevesa $T(G^+)$ deluje na podoben princip kot Gomory-Hu algoritem opisan v razdelku 3.1.1. Razlika je le v tem, da si pomagamo z drevesom T_*, ki vsebuje tanke in debele povezave. Debele povezave povezujejo vreče med seboj in predstavljajo veljavne minimalne prereze. Tanke povezave povezujejo vozlišča znotraj posamezne vreče B in med temi vozlišči je potrebno izračunati maksimalne pretoke in minimalne prereze za graf G^+.

Vhodni podatki algoritma so: izračunano drevo $T(G)$, struktura za hranjenje sosedov za posamezno vozlišče v $T(G)$ označena z $neighbours$, vozlišči b in d povezave na kateri smo povečali prepustnost v grafu G in vrednost nove prepustnosti $c^+(b, d)$. Izhodni podatek je drevo $T(G^+)$. Tekom izvajanja algoritma bomo v strukturi $neighbours$ hranili za posamezno vozlišče sosede v trenutnem drevesu T_*.

V strukturi $bags$ hranimo vreče z vozlišči grafa G. Na začetku se vsa vozlišča nahajajo v eni vreči, $bags = [\{V\}]$. Vrednost $index$ predstavlja zaporedno številko vreče v strukturi $bags$. Nato zgradimo drevo T_* kot je prikazano na zgornji strani slike 5.1 v dveh korakih. Najprej s pregledom določimo pot P_{bd} in s tem množico tankih povezav povezav E_*^T. Tanke povezave bomo korakoma spreminjali v debele. Dokler množica tankih povezav E_*^T ni prazna ponavljamo:

Izberemo poljubno povezavo (s, t) iz množice E_*^T in v vrsticah 10–14 izberemo vrečo B, ki vsebuje vozlišči s in t izbrane povezave. Nato v vrstici 15 izračunamo maksimalni pretok in minimalni prerez med izbranima vozliščema s in t. Algoritem za izračun maksimalnega pretoka in minimalnega prereza smo opisali v podpoglavju (2.4). V vrsticah 16–19 iz množice E_*^T izbrišemo povezave, ki povezujejo vozlišča znotraj izbrane vreče B. Vozlišča iz vreče B bomo razdelili v vreči B_s, ta vsebuje s, in B_t, ta vsebuje t, takole: V vrsticah 22–30 za posamezno vozlišče $x \in B \setminus \{s, t\}$ preverimo ali se nahaja v isti množici izračunanega minimalnega prereza kot vozlišče s ali t. V primeru da se nahaja na isti strani kot vozlišče s, dodamo vozlišče x v vrečo B_s in ga dodamo za soseda vozlišču s ter v množico E_*^T

dodamo dobljeno povezavo (s, x), sicer pa vozlišče x dodamo v vrečo B_t in ga dodamo za soseda vozlišču t ter v množico E_*^T dodamo dobljeno povezavo (t, x). V vrstici 31 definiramo množico N za hranjenje debelih povezav, ki povezujejo vozlišča iz preostalih vreč z vozlišči v vreči B. Množico povezav N izračunamo v vrsticah 32–38 tako, da za posamezno vozlišče x v vrečah iz strukture *bags*, ki niso enake trenutno izbrani vreči B preverimo, ali ima za soseda katero izmed vozlišč t_x, ki se nahaja v vreči B. Za vsa vozlišča t_x v B dobimo debelo povezavo (t_x, x) in jo dodamo v množico N.

V vrsticah 39–45 nato za posamezno povezavo $(t_x, x) \in N$ preverimo, na kateri strani prereza se nahaja vozlišče x izbrane povezave. V primeru da se nahaja na isti strani minimalnega prereza kot vozlišče s, vozlišču x dodamo vozlišče s za soseda. V množico E_*^F dodamo povezavo (x, s) in odstranimo povezavo (x, t_x). Odstranjeno povezavo dodamo v množico E_*^T. V nasprotnem primeru, kadar se vozlišče x nahaja na isti strani minimalnega prereza kot vozlišče t, dodamo vozlišču x vozlišče t za soseda. V množico povezav E_*^F dodamo povezavo (t, x) in odstranimo povezavo (t_x, x). Odstranjeno povezavo dodamo v množico povezavo E_*^T. Na ta način zagotovimo, da so vozlišča znotraj posamezne vreče B' povezana s tankimi povezavami. Vozlišča vsebovana v vreči B', ki so povezana z vozlišči iz preostalih vreč, pa z debelimi povezavami. V vrstici 46 vrečo B odstranimo iz strukture *bags* in jo nadomestimo z vrečama B_s in B_t. Na koncu v vrstici 47 dodamo v množico E_*^F izbrano povezavo (s, t) in jo odstranimo iz množice E_*^T. Dodani debeli povezavi nastavimo prepustnost na vrednost izračunanega maksimalnega pretoka in minimalnega $s-t$ prereza $c_*(s, t) = |f|$ in končamo z iteracijo izračuna.

Časovna zahtevnost algoritma

Povezave na poti P_{bd} v drevesu T_* na zgornji strani slike 5.1 so na začetku tanke in lahko v najslabšem primeru predstavljajo znaten delež drevesa, zato glede na teoretično časovno zahtevnost prihranka na izračunu drevesa $T(G^+)$ nimamo. Torej bo teoretična časovna zahtevnost algoritma enaka teoretičnim

Algoritem 6 Povečanje prepustnosti povezave: 1. del

Input: $G = (V, E, c), T(G) = (V_T, E_T, c_T), b, d, c^+(b, d)$, $neighbours[\{\}]$

Output: T_*

1: $T_* = (V_*, E_*^F, E_*^T, c_*), bags = [\{V\}], index = 0$
2: $P_{bd} = BFS(T(G), b, d)$
3: **for** $(u, v) \in E_T$ **do**
4: **if** $(u, v) \in P_{bd}$ **then**
5: $E_*^T = E_*^T \cup (u, v)$
6: **else**
7: $E_*^F = E_*^F \cup (u, v)$
8: **while** $E_*^T \neq \emptyset$ **do**
9: $(s, t) = (u, v) \in E_*^T$
10: **for** $a = 0; a < bags.length; a++$ **do**
11: **if** $s, t \in bags[a]$ **then**
12: $B = bags[a]$
13: $index = a$
14: **break**
15: $|f|, S, T = Edmonds - Karp(G^+, s, t)$
16: **for** $x \in B$ **do**
17: $N_x = neighbours[x]$
18: **for** $z \in N_x$ **do**
19: $E_*^T = E_*^T \setminus \{\{(x, z)\}\}$
20: $B_s = B_s \cup \{s\}$
21: $B_t = B_t \cup \{t\}$

Algoritem 7 Povečanje prepustnosti povezave: 2. del

22: **for** $x \in B \setminus \{s, t\}$ **do**
23: **if** $S[x] == true$ **then**
24: $E_*^T = E_*^T \cup \{(s, x)\}, E_*^F = E_*^T \setminus \{(s, x)\}$
25: $temp = c_*(x, t), c_*(x, t) = 0, c_*(x, s) = temp$
26: $B_s = B_s \cup \{x\}$
27: **if** $T[x] == true$ **then**
28: $E_*^T = E_*^T \cup \{(t, x)\}$
29: $temp = c_T(x, s), c_T(x, s) = 0, c_*(x, t) = temp$
30: $B_t = B_t \cup \{x\}$
31: $N = \{\}$
32: **for** $a = 0; a < bags.size; a++$ **do**
33: **if** $a \neq index$ **then**
34: **for** $x \in bags[a]$ **do**
35: $N_p = neighbours[x]$
36: **for** $t_x \in N_p$ **do**
37: **if** $t_x \in B$ **then**
38: $N = N \cup \{(x, t_x)\}$
39: **for** $(t_x, x) \in N$ **do**
40: **if** $S[x] == true$ **then**
41: $E_*^F = E_*^F \setminus \{(t_x, x)\}) \cup \{(s, x)\}$
42: $temp = c_*(x, t), c_*(x, t) = 0, c_*(s, x) = temp$
43: **if** $T[x] == true$ **then**
44: $E_*^F = (E_*^F \setminus \{(t_x, x)\}) \cup \{(t, x)\}$
45: $temp = c_*(x, s), c_*(x, s) = 0, c_*(t, x) = temp$
46: $bags[index] = B_s, bags[index++] = B_t$
47: $E_*^F = E_*^F \cup \{(s, t)\}, E_*^T = E_*^T \setminus \{(s, t)\}, c_*(s, t) = |f|$
48: **return** T_*

časovnim zahtevnostim algoritma za izračun drevesa $T(G^-)$ za vse tri tipe grafov G.

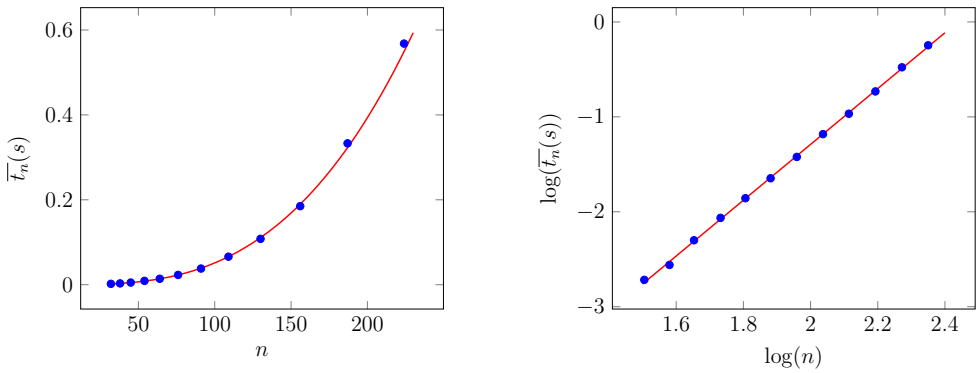
V nadaljevanju bomo opisali rezultate meritev v praksi za posamezne tipe grafov. Postopek izračuna regresijskih premic in časovne zahtevnosti algoritma smo opisali v podrazdelku 3.1.2. Algoritem smo testirali za posamezne tipe grafov na enak način kot pri algoritmu za izračun drevesa $T(G^-)$, opisanem v podpoglavju 5.1.1. Razlika je bila le, da smo na naključni povezavi (b, d) v grafu G povečali prepustnost. Vrednost prepustnosti $c^+(b, d)$ smo izračunali tako, da smo vrednosti $c(b, d)$ prišteli naključno vrednost med 1 in $(n - c(b, d))$.

Za grafe $\mathcal{K}_n$ smo izračunali regresijsko premico $2.940 \cdot x - 7.170$ iz katere dobimo funkcijo $f_1(n) = 10^{-7.170} \cdot n^{2.940}$. Časovna zahtevnost algoritma znaša $O(10^{-7.170} \cdot n^{2.940})$ in ocenjena časovna zahtevnost algoritma bo torej $O(10^{-7.170} \cdot n^{2.940}) = O(n^3)$. Kot smo že omenili za grafe $\mathcal{K}_n$ so drevesa $T(G)$ v obliki zvezde, kjer je dolžina poti P_{bd} 1 ali 2. V tem primeru smo za posamezen graf G^+ morali izračunati 1 ali 2 minimalna prereza v grafu G^+, da smo iz drevesa $T(G)$ dobili drevo $T(G^+)$. Tako smo prihranili na izračunu $O(n)$ minimalnih prerezov zato se ocenjena časovna zahtevnost v primerjavi z Gusfieldovim in algoritmom za izračun drevesa $T(G^-)$ razlikuje za faktor $O(n)$.

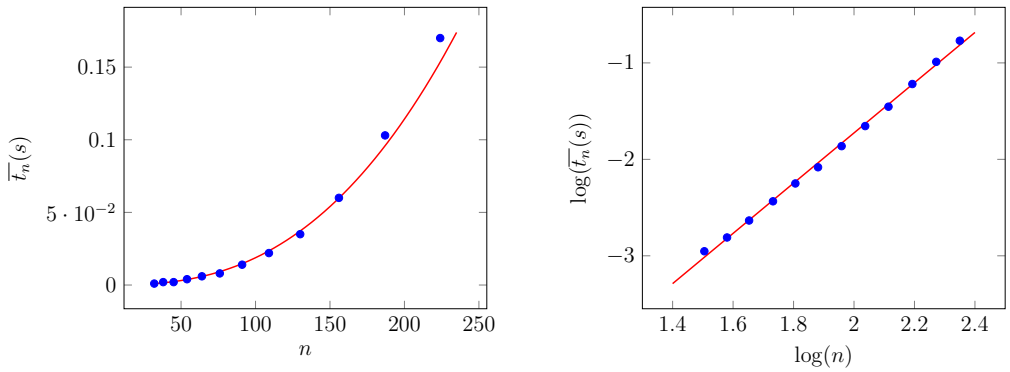
Izračunana regresijska premice za grafe $\mathcal{R}_n$ je znašala $2.604 \cdot x - 6.934$. Dobimo funkcijo $f_2(n) = 10^{-6.934} \cdot n^{2.604}$ in časovno zahtevnost algoritma $O(10^{-6934} \cdot n^{2.604})$. Ocenjena časovna zahtevnost algoritma bo znašala $O(10^{-6.934} \cdot n^{2.604}) = O(n^3)$. Za grafe $\mathcal{M}_n$ smo dobili regresijsko premico $2.685 \cdot x - 7.202$. Dobimo funkcijo $f_3(n) = 10^{-7.202} \cdot n^{2.685}$ in časovno zahtevnost algoritma $O(10^{-7.202} \cdot n^{2.685})$. Ocenjena časovna zahtevnost algoritma na mrežnih grafih je $O(10^{-7.202} \cdot n^{2.685}) = O(n^3)$. Glede na oblike dreves $T(G)$ za oba tipa grafov je povprečna dolžina poti P_{bd} za graf na n vozliščih približno 2, kot prikazuje slika 5.4. Torej za izračun drevesa $T(G)$ potrebujemo $\overline{P_{bd}}$ izračunov maksimalnih pretokov in minimalnih prerezov. Ocenjena časovna zahtevnost algoritma je torej za oba tipa grafov enaka ocenjeni časovni zah-

tevnosti algoritma za izračun drevesa $T(G^-)$ in za faktor $O(n)$ manjša od ocenjene časovne zahtevnosti Gusfieldovega algoritma.

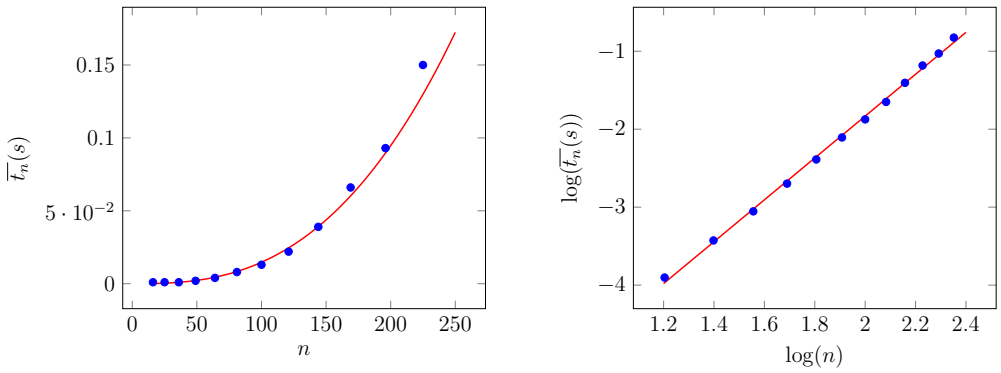
Slike 5.8, 5.9 in 5.10 prikazujejo rezultate meritev za posamezen tip grafa G. Prikaz rezultatov in izračun časovnih zahtevnosti je narejen na enak način kot pri algoritmu za izračun drevesa $T(G^-)$.



Slika 5.8: Prikaz rezultatov meritev algoritma za izračun dreves $T(G^+)$ za grafe $\mathcal{K}_n$.



Slika 5.9: Prikaz rezultatov meritev algoritma za izračun dreves $T(G^+)$ za grafe $\mathcal{R}_n$.



Slika 5.10: Prikaz rezultatov meritev algoritma za izračun dreves $T(G^+)$ za grafe $\mathcal{M}_n$.

Prostorska zahtevnost algoritma

Za hranjenje grafa G in drevesa T_* uporabimo matriki sosednosti, pri čemer potrebujemo $O(2 \cdot n^2)$ prostora. Poleg tega imamo tabelo za hranjenje vreč v katerih se nahajajo vozlišča grafa G in prostorska zahtevnost te tabele znaša $O(n)$. Za hranjenje logične vrednosti o debelini posamezne povezave v T_* uporabimo matriko sosednosti s prostorsko zahtevnostjo $O(n^2)$.

Za drevo T_* tekom izračuna potrebujemo strukturo za hranjenje trenutno sosednih vozlišč za posamezno vozlišče. To lahko implementiramo z uporabo dvodimenzionalne tabele, v kateri za izbrano vozlišče hranimo tabelo sosedov. Prostorska zahtevnost dvodimenzionalne tabele znaša $O(n^2)$.

Prostorsko zahtevnost algoritma za izračun maksimalnega pretoka in minimalnega prereza smo omenili v podpoglavju 2.4.1 in znaša $O(n^2)$. Po izračunu hranimo vozlišča na posamezni strani minimalnega u-v prereza v tabeli vozlišč U in V s prostorsko zahtevnostjo $O(2 \cdot n)$. Nato trenutno izbrano vrečo B razdelimo na dva dela, na vreči B_u in B_v. Prostorska zahtevnost dveh na novo ustvarjenih vreč znaša toliko kolikor vozlišč hranimo v vreči B, torej $O(n)$.

Pri analizi prostorske zahtevnosti algoritma smo za hranjenje posameznih vrednosti predpostavili prostorsko zahtevnost $O(1)$. Skupna prostorska

zahtevnost algoritma je vsota omenjenih prostorskih zahtevnosti in znaša $O((2 \cdot n^2) + n + n^2 + n^2 + n^2 + (2 \cdot n) + n) = O((5 \cdot n^2) + (4 \cdot n)) = O(n^2)$.

5.1.3 Slučajna sprememba prepustnosti

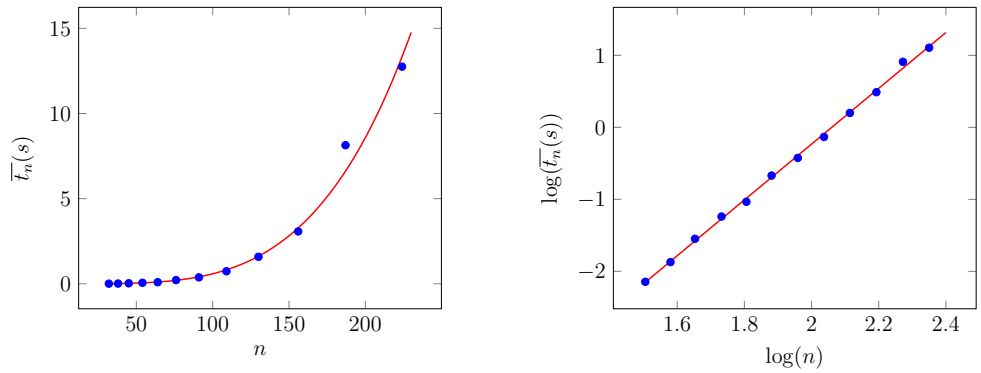
Na koncu smo uporabili algoritem za izračun drevesa $T(G^U)$ z uporabo algoritma za izračun drevesa $T(G^+)$ in $T(G^-)$. Postopek analize časovne zahtevnosti je potekal na enak način kot pri analizi časovne zahtevnosti pri obeh algoritmih, le da smo na naključno izbrani povezavi (b, d) v grafu G izvedli naključno operacijo zmanjšanja ali povečanja prepustnosti. Glede na izbrano operacijo smo uporabili enega izmed omenjenih algoritmov za izračun drevesa.

Sedaj bomo opisali rezultate meritev za posamezne tipe grafov. Za grafe $\mathcal{K}_n$ smo dobili regresijsko premico oblike $3.876 \cdot n - 7.985$. Iz izračunane premice dobimo funkcijo za časovno zahtevnost algoritma $f_1(n) = 10^{-7.985} \cdot n^{3.876}$. Za grafe $\mathcal{R}_n$ smo dobili regresijsko premico oblike $2.857 \cdot n - 7.066$ in dobljeno funkcijo $f_2(n) = 10^{-7.066} \cdot n^{2.857}$. Za zadnji tip grafov $\mathcal{M}_n$ smo dobili regresijsko premico $3.166 \cdot n - 7.938$ in funkcijo $f_3(n) = 10^{-7.938} \cdot n^{3.166}$. Za grafe $\mathcal{K}_n$ dobimo torej ocenjeno časovno zahtevnost algoritma $O(n^4)$ in za preostala dva tipa grafov časovno zahtevnost $O(n^3)$.

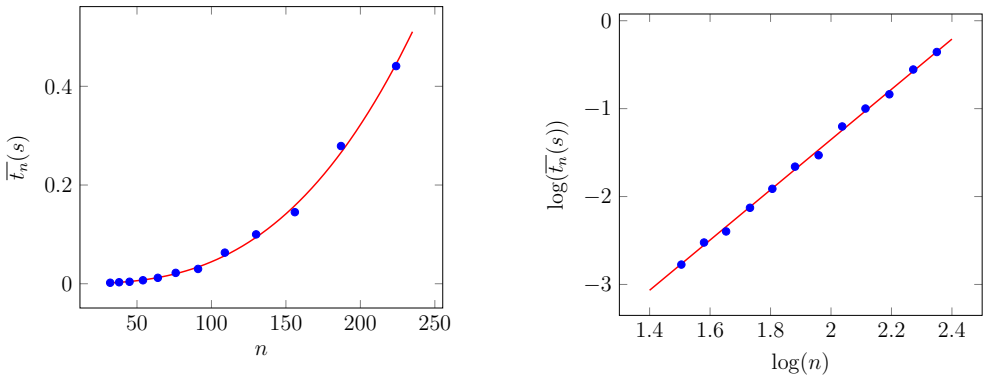
Slike 5.11, 5.12 in 5.13 prikazujejo rezultate meritev za posamezne tipe grafov. Sam prikaz in izračun časovnih zahtevnosti je narejen na enak način kot pri algoritmih za izračun drevesa po zmanjšanju ali povečanju prepustnosti na povezavi.

Časovna zahtevnost algoritma je v glavnem odvisna od deleža operacij zmanjšanja in povečanja prepustnosti. V primeru, da predstavlja večino operacij zmanjšanje prepustnosti na povezavi, je glede na analizirano časovno zahtevnost algoritma za posamezne tipe grafov bolje uporabiti Gusfieldov algoritem za izračun posameznega drevesa $T(G^-)$ od začetka. V nasprotnem primeru se izplača uporabiti algoritem za izračun drevesa $T(G^+)$ ter tako prihranimo glede na analizo oblike izračunanih dreves približno $O(n)$ izračunov maksimalnih pretokov in minimalnih prerezov za posamezen tip grafa na n

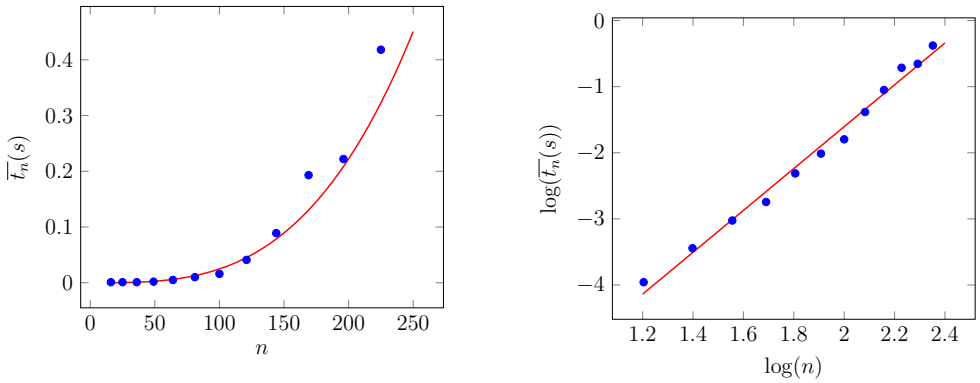
vozliščih.



Slika 5.11: Prikaz rezultatov meritev algoritma za izračun dreves $T(G^U)$ za grafe $\mathcal{K}_n$.



Slika 5.12: Prikaz rezultatov meritev algoritma za izračun drevesa $T(G^U)$ za grafe $\mathcal{R}_n$.



Slika 5.13: Prikaz rezultatov meritev algoritma za izračun drevesa $T(G^U)$ za grafe $\mathcal{M}_n$.

Poglavje 6

Zaključek

V diplomskem delu smo spoznali Gomory-Hu problem, Gomory-Hu drevo in algoritem za izračun Gomory-Hu drevesa, ki kot bistveno sestavino uporablja rutino za izračun maksimalnega pretoka in minimalnega prereza s pomočjo Edmonds-Karpovega algoritma. Algoritem za izračun Gomory-Hu drevesa smo tudi implementirali in ocenili njegovo časovno zahtevnost na treh različnih tipih grafov.

V osnovi ima Gusfieldov algoritem relativno veliko časovno zahtevnost, zato smo se lotili tudi dinamičnega izračuna Gomory-Hu drevesa, pri katerem s pomočjo obstoječega Gomory-Hu drevesa za graf G, izračunamo novo Gomory-Hu drevo v spremenjenem grafu G^U. Ugotovili smo, da je v praksi prihranek pri dinamičnem izračunu Gomory-Hu drevesa bistveno večji v primeru povečanja prepustnosti na eni povezavi v primerjavi z zmanjšanjem prepustnosti. Tudi algoritme za izračun dinamičnih Gomory-Hu dreves smo implementirali in eksperimentalno ovrednotili na istih treh tipih grafov.

Za konec bomo na kratko omenili algoritme, s pomočjo katerih v krajšem času izračunamo maksimalni pretok in minimalni prerez ter s tem drevo $T(G)$.

Za izračun maksimalnega pretoka in minimalnega prereza smo uporabili Edmonds-Karpov algoritem. Poleg tega algoritma obstaja še veliko hitrejših, vendar za implementacijo bolj zapletenih algoritmov. Nekateri izmed

njih so: Dinicov [2], Rao-Tarjanov [11] ter Goldberg-Raov [4]. Slednji je izmed vseh obstoječih algoritmov najhitrejši, njegova časovna zahtevnost znaša $O(\min(n^{\frac{3}{2}}, m) \cdot m^{\frac{1}{2}}))$.

Na hitro opišimo idejo Dinicovega algoritma. Pri izračunu maksimalnega pretoka in minimalnega prereza med izbranima vozliščema s in t si pomagamo z *nivojskim grafom* $G_L = (V_L, E_L, c_L)$. Vsebuje enako število vozlišč kot graf G, le da za označevanje vozlišč uporabljamo celoštevilske vrednosti. Zgradimo ga iz grafa G_f in posamezno vozlišče $z \in V_L$ predstavlja dolžino najkrajše poti P_{sz} v grafu G_f, velja $dist(z) = |P_{sz}|$. Za vozlišče s imamo v G_L vedno $dist(s) = 0$ in za vozlišča, ki niso dosegljiva iz vozlišča s, vrednost ∞. Algoritem izračuna maksimalni pretok in minimalni prerez na naslednji način:

V vsaki iteraciji izračunamo iz grafa G_f graf G_L. Če velja $dist(t) = \infty$ končamo z izračunom, sicer pa poiščemo množico F f-povečujočih poti z enakimi dolžinami med vozliščema s in t, ki predstavljajo *blokirni tok*. Blokirni tok pomeni, da po povečanju pretoka skozi posamezno f-povečujočo pot P_{st} v grafu G_f vsaj eno povezavo $e \in P_{st}$ zasičimo, velja $c_f(e) = 0$. In tako končamo z iteracijo izračuna. Časovna zahtevnost algoritma znaša $O(n^2 \cdot m)$. Algoritem je podrobneje opisan v [2].

Najhitrejši algoritem za izračun drevesa $T(G)$ z uporabo Golderg-Rao algoritma za izračun maksimalnega pretoka in minimalnega prereza ter s prepustnostmi na povezavah v grafu G enako eni enoti, ima časovno zahtevnost $O(\min(n^{\frac{3}{2}}, m) \cdot m^{\frac{1}{2}} \cdot n)$. Obstaja tudi algoritem [5], pri katerem za izračun drevesa $T(G)$ ni potrebno izračunati maksimalnega pretoka in minimalnega prereza med izbranima vozliščema v grafu G. Časovna zahtevnost algoritma znaša $O(m \cdot n)$ in je v primerjavi s predhodno omenjenim algoritmom za izračun drevesa $T(G)$ za enak graf G, hitrejši za faktor $\Omega(\sqrt{n})$.

So pa omenjeni algoritmi precej kompleksnejši in bistveno presegajo okvir diplomskega dela.

Literatura

[1] Pan An, Philipp Keck in Taehoon Kim. Min-cut algorithms. `http://www.comp.nus.edu.sg/~rahul/allfiles/cs6234-16-mincuts.pdf`, 2013. [Dostopano: 12. 2. 2018].

[2] Dinics's algorithm. Dosegljivo: `https://en.wikipedia.org/wiki/Dinic%27s_algorithm`. [Dostopano: 8. 3. 2018].

[3] E. N. Gilbert. Random graphs. *Ann. Math. Statist.*, 30(4):1141–1144, 12 1959.

[4] Andrew V. Goldberg in Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.

[5] Ramesh Hariharan, Telikepalli Kavitha, Debmalya Panigrahi in Anand Bhalgat. An õ(mn) gomory-hu tree construction algorithm for unweighted graphs. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 605–614, 2007.

[6] Tanja Hartmann. *Algorithms for Graph Connectivity and Cut Problems - Connectivity Augmentation, All-Pairs Minimum Cut, and Cut-Based Clustering.* Disertacija, Karlsruhe Institute of Technology, 2014.

[7] Tanja Hartmann in Dorothea Wagner. Dynamic gomory-hu tree construction - fast and simple. *CoRR*, abs/1310.0178, 2013.

[8] Brian Jacokes. Lecture notes on multiway cuts and k-cuts. Dosegljivo: http://math.mit.edu/~goemans/18434S06/multicuts-brian.pdf, 2006. [Dostopano: 9. 1. 2018].

[9] David R Karger in Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996.

[10] Robert Kleinberg. Introduction to Algorithms: The Edmonds-Karp Max-Flow Algorithm. Dosegljivo: http://www.cs.cornell.edu/courses/cs4820/2011sp/handouts/edmondskarp.pdf, 2010. [Dostopano: 21. 1. 2018].

[11] James B. Orlin. Max flows in õ(nm) time, or better. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 765–774, New York, NY, USA, 2013. ACM.

[12] Yaniv Ovaida. Computational Feasibility of Increasing the Visibility of Vertices in Covert Networks. Dosegljivo: http://scholarship.claremont.edu/cgi/viewcontent.cgi?article=1015&context=hmc_theses, 2010. [Dostopano: 8. 2. 2018].

[13] Tim Roughgarden. CS261: A Second Course in Algorithms: Augmenting Path Algorithms for Maximum Flow. Dosegljivo: http://theory.stanford.edu/~tim/w16/l/l2.pdf, 2016. [Dostopano: 21. 1. 2018].