Brzi algoritmi za problem najkraćeg puta

Mandić, Andrija

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Andrija Mandić

BRZI ALGORITMI ZA PROBLEM NAJKRAĆEG PUTA

Diplomski rad

Voditelj rada:
doc. dr. sc. Goranka Nogo

Zagreb, rujan, 2021.

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

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

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Posvetio bih ovaj rad svim profesorima, mentorima i kolegama koji su uvelike doprinijeli mom matematičkom obrazovanju

Sadržaj

Uvod

Problem pronalaska najkraćeg puta između početnog i krajnjeg vrha u grafu bitan je problem koji ima široku upotrebu u stvarnom svijetu. Modeliranjem stvari pomoću grafova, primjena se proteže od navigacijskih sustava, prometnih simulacija pa do *web* pretraživanja, usmjeravanja podataka na internetu i upotrebe kod baza podataka. Iako je potreba za efikasnim algoritmima velika, u primjeni se često upotrebljavaju heurističke metode koje na brz način daju približno dobra rješenja. Ukoliko je potrebna preciznost, problemu se pristupa egzaktnim algoritmima od kojih ćemo dio obraditi u ovom radu.

Problem možemo podijeliti u dvije faze: faza *predprocesiranja* i faza *upita*. Faza upita sastoji se od pronalaska najkraćeg puta u grafu za zadana dva vrha. Glavna motivacija kod nastajanja algoritama je smanjiti vrijeme potrebno za izvršavanje te faze. Kako bi se to postiglo, u novije doba razvijaju se razni algoritmi koji koriste obradu podataka i spremanje bitnih informacija o grafu prije same faze upita – to nazivamo fazom predprocesiranja. Ovisno o pozadini iz koje dolazi problem, zahtjevi na fazu predprocesiranja mogu biti razni. Na primjer, ukoliko se radi o prometnoj mreži koja se u pravilu ne mijenja često, može se dopustiti veće vrijeme potrebno za obradu podataka koje bi kasnije rezultiralo kraćom fazom upita. Ukoliko je predprocesiranje potrebno češće provoditi, npr. kod dinamičnih prometnica gdje u obzir uzimamo i gužve na njima, preveliko vrijeme izvršavanja pripremne faze bi bilo nedopustivo. Naravno, tu su i memorijska ograničenja na količinu informacija koju se može spremiti i koja bi služila ubrzanju faze upita.

U poglavlju 1 donosimo osnovne pojmove iz teorije grafova potrebne za razumijevanje ovog rada. Poglavlje 2 opisuje problem najkraćeg puta u grafu, njegove varijante i primjene. Odlučujemo se za varijantu pronalaska najkraćeg puta za zadani par vrhova, dok ćemo sve promatrati iz perspektive grafova nastalih iz prometnih mreža. Vrhovi grafa predstavljaju važna prometna čvorišta, a bridovi među njima daju informaciju o potrebnom vremenu prelaska. Poglavlje 3 opisuje klasične algoritme za problem najkraćih putova koji ne koriste predprocesiranje. Radi se o Dijkstrinom i A^* algoritmu koji će se koristiti u pojedinim dijelovima naprednijih algoritama. Jedan takav naziva se *Contraction Hierarchies* – CH koji je predstavljen u poglavlju 4. Algoritam se temelji na hijerarhiji vrhova koja se uspostavi u fazi predprocesiranja tokom koje se u graf dodaju novi bridovi zvani *prečaci*. Koristeći dvosmjerno Dijkstrino pretraživanje na modificiranom grafu, uzimajući u obzir

hijerarhiju vrhova, CH algoritam efikasno realizira fazu upita. Poglavlje 5 donosi još jedan algoritam koji koristi predprocesiranje. Radi se o *Landmark* A^* – ALT algoritmu koji koristi ciljno orijentirani pristup, za razliku od hijerarhijskog CH algoritma. ALT algoritam uz pomoć vrhova *orijentacije* za koje je poznata udaljenost od i do svih ostalih vrhova, te modifikacije nejednakosti trokuta daje kvalitetnu procjenu ciljne udaljenosti do krajnjeg vrha. Ovaj algoritam koristi A^* pretraživanje s navedenom heuristikom procjene udaljenosti. U poglavlju 6 donosimo neke implementacijske detalje i usporedbu promatranih algoritama. Predstavljeni napredni algoritmi CH i ALT usporedit će se s Dijkstrinim algoritmom na podacima cestovnih mreža dobivenih iz američkih saveznih država. U završnom poglavlju 7 donosimo zaključak rada uz prijedloge daljnjeg poboljšanja i nastavka istraživanja.

Poglavlje 1

Teorija grafova

Problem najkraćeg puta matematički se modelira pomoću grafova. Ovisno o prirodi problema, često je reprezentacija intuitivna. Primjerice, proučavamo li prometnu mrežu, za svaku cestu uvest ćemo poseban brid grafa. Sjecišta prometnica predstavljat će vrhove. Kako bi se lakše razumio problem najkraćeg puta u grafu potrebno je određeno predznanje teorije grafova. U ovom odjeljku uvest ćemo osnovne pojmove potrebne za praćenje ostatka rada, koji se koriste u promatranom problemu. Detaljniji uvod u teoriju grafova nalazi se u [15].

1.1 Osnovni pojmovi

Definicija 1.1.1. *Usmjereni težinski graf G je uređena trojka $G = (V, E, \omega)$, gdje je V skup vrhova, $E \subseteq V \times V$ skup bridova grafa G, te $\omega\colon E \to R$ funkcija koja svakom bridu pridružuje njegovu težinu.*

Svaki brid grafa povezuje točno dva vrha. Dva vrha su povezana ako postoji brid koji ih spaja. Brid $e = (u, v) \in E$ označava usmjerenu poveznicu od vrha u prema vrhu v. Neusmjerene grafove možemo gledati kao posebne vrste usmjerenih, s tim da vrijedi: $(u, v) \in E$ ako i samo ako $(v, u) \in E$. Težinska funkcija kod netežinskih grafova bila bi jednaka jediničnoj funkciji, tj. $\omega(u, v) = 1$ ako je $(u, v) \in E$.

Definicija 1.1.2. *Put $P = \langle v_0, v_1, v_2, \ldots, v_k \rangle$ od vrha v_0 do vrha v_k je niz vrhova redom povezanih bridovima. Težina puta iznosi:*

$$\omega(P) = \sum_{i=0}^{i=k-1} \omega(v_i, v_{i+1}). \tag{1.1}$$

Napomena 1.1.3. *Ovisno o literaturi, definicija 1.1.2 se nekad naziva šetnja u grafu, a put se definira kao šetnja kod koje su svi vrhovi različiti (osim eventualno prvog i zadnjeg).*

U primjeni ovog rada, razlika u definicijama neće igrati ulogu. Međutim, potrebno se ograničiti na određenu kodomenu težina bridova, o čemu će biti riječ u odjeljku 2.

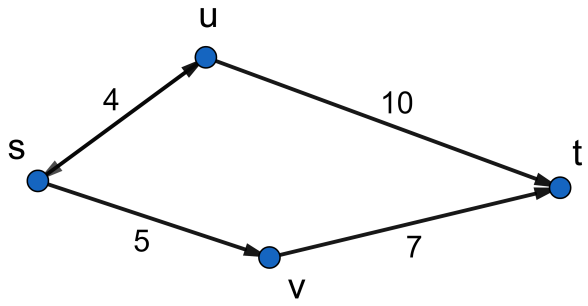
Definicija 1.1.4. *Najkraći put, tj. minimalnu težinsku udaljenost između vrhova $u, v \in V$ označavamo s $d_G(u, v)$ te ona iznosi:*

$$d_G(u,v) := \begin{cases} \min\{\, \omega(P) \mid P \text{ je put od } u \text{ do vrha } v \,\} & \text{ako postoji put } P \text{ od } u \text{ do vrha } v \\ +\infty & \text{inače} \end{cases}. \tag{1.2}$$

Definicija 1.1.5. *Gustoća $\rho(G)$ usmjerenog grafa $G = (V, E, \omega)$ računa se izrazom:*

$$\rho(G) = \frac{|E(G)|}{|V(G)| \cdot (|V(G)| - 1)}. \tag{1.3}$$

Ovisno o broju vrhova, graf s velikim brojem bridova, čija je gustoća blizu 1 nazivamo *gustim*. Suprotno, ako je gustoća grafa bliže 0, možemo reći ga se radi o *rijetkom* grafu. Većina uvedenih pojmova odnosi se na usmjerene težinske grafove, no pošto su oni poopćenja neusmjerenih netežinskih, na očit način dobijemo odgovarajuće definicije i za takve vrste grafova.



Slika 1.1: Primjer usmjerenog grafa

Primjer 1.1.6. *Na slici 1.1 prikazan je jedan usmjereni graf $G = (V, E, \omega)$, pri čemu je skup vrhova $V = \{s, u, v, t\}$ i skup bridova $E = \{(s, u), (u, s), (s, v), (u, t), (v, t)\}$. Vrijednosti težinske funkcije mogu se lako očitati sa slike, gdje je vrijednost funkcije zapisana iznad pojedinog brida grafa. Npr. $\omega(u, s) = \omega(s, u) = 4$ i $\omega(v, t) = 7$. Put $P_1 = \langle s, u, t\rangle$ primjer je puta od vrha s do vrha t, težine $\omega(P_1) = \omega(s, u) + \omega(u, t) = 4 + 10 = 14$. Lako je uočiti da je $P = \langle s, v, t\rangle$ najkraći put u grafu G od početnog s do krajnjeg vrha t. Vrijedi $d_G(s, t) = 12$. Gustoća grafa G iznosi: $\rho(G) = \frac{5}{4 \cdot 3} \approx 0.4167$.*

1.2 Zapis grafa

Graf se može na različite načine zapisati u računalu. Kod odabira nekog od načina, uzimamo u obzir procijenjenu gustoću grafa, složenosti operacija poput provjere postojanja brida i težine između dva vrha te veličine memorije koju bi određena reprezentacija grafa zauzela.

Matrica susjedstva

Matrica susjedstva (eng. *adjacency matrix*) je matrica dimenzija $|V| \times |V|$ koja na lokaciji (i, j) sadrži informaciju o postojanju/težini brida od čvora i do čvora j. Nepopunjena mjesta u tablici se dopunjuju shodno promatranom problemu. Za problem pronalaska najkraćeg puta, prikladno je na mjestu (i, j) ako ne postoji brid $(i, j) \in E$ postaviti vrijednost $+\infty$ tj. neku jako veliku konstantu. Naravno, na mjestima (i, i) pisat će 0. Prednost matrice susjedstva je vremenska složenost $O(1)$ provjere postojanja i težine brida - ako je graf težinski. Nedostatak je poprilično veliko zauzeće memorije, pogotovo ako radimo s grafovima koji su uglavnom rijetki (def. 1.1.5). Matrica susjedstva netežinskog grafa (1 oznaka da postoji brid, 0 u suprotnom) potencirana na k zapisuje broj staza duljine k, na mjestu (i, j), od vrha i prema j. Više detalja može se pronaći u [11].

Popis susjeda

Zapis grafa kao popis susjedstva (eng. *adjacency list*) sastoji se od popisivanja susjeda svakog pojedinog vrha. Prednost je velika ušteda prostora kod grafova s manjim brojem bridova, no ovim pristupom ne možemo u konstantnom vremenu provjeriti postoji li veza između neka dva proizvoljna čvora. Naime, potrebno je provjeriti u popisu susjedstva početnog čvora nalazi li se krajnji čvor u njemu. Dodatna prednost je mogućnost iterativnog obilaska po postojećim bridovima zadanog vrha (imamo ih u listi susjeda), što nije bilo moguće kod matrice susjedstva.

Poglavlje 2

Problem najkraćeg puta u grafu

Postoje razne inačice problema pronalaska najkraćeg puta u grafu. Prije odabira algoritma i pristupanja rješavanju problema, bitno je uzeti u obzir o kakvom se grafu radi. Koje sve vrijednosti mogu poprimati težine bridova? Mogu li one biti negativne? Jesu li cjelobrojne? Kolika je očekivana gustoća grafa? Smijemo li dopustiti aproksimativno ili tražimo egzaktno rješenje? Uz to, postoje razne varijante problema ovisno o broju izvora i broju traženih putova. Više detalja može se pronaći u [3] i [9].

2.1 Varijante problema

U ovom radu ograničit ćemo se na traženje egzaktnog rješenja problema najkraćeg puta. Brzi algoritmi koji daju približna rješenja nisu u domeni razmatranja.

Ovisno o kodomeni funkcije težina bridova, imamo različite pristupe problemu. Ukoliko su dozvoljene negativne vrijednosti težina, postoji opasnost egzistencije negativnog ciklusa u grafu. U tom slučaju bi najkraći put mogao iznositi $-\infty$, pošto svaki put koji obiđe cijeli negativni ciklus više puta, ima manju ukupnu duljinu te stoga ne postoji najkraći put. Jedan način uklanjanja tog problema je zabraniti ponavljanje istih bridova u potencijalnom rješenju. Takav uvjet se svede na restrikciju postojanja dijela puta koji je ciklus negativne težine. Međutim, problem najkraćeg puta tada postaje NP-težak. Dokaz te tvrdnje nalazi se u dodatku rada [9]. Kod takvih problema nemamo poznate algoritme primjenjive na većim, stvarnim podacima te se njima pristupa aproksimacijskim ili metaheurističkim metodama. Druga opcija je detekcija negativnih ciklusa, koja je izvediva koristeći *Bellman-Fordov* algoritam. Pošto se ovaj rad orijentira na primjere i testiranja nad podacima iz cestovnih mreža, fokusirat ćemo se samo na nenegativne realne težine bridova kod kojih ne postoji opisani problem. Promatramo li samo prirodne brojeve kao potencijalne težine bridova, postoje razna poboljšanja *Dijkstrinog* algoritma iz odlomka 3.1. Detaljnije se može pronaći u [1].

Za dani graf $G = (V, E, \omega)$ (usmjereni ili neusmjereni, težinski ili netežinski) postoji više vrsta problema najkraćeg puta:

- **Najkraći put iz jednog vrha** (eng. *Single-Source Shortest-Path Problem*): za zadani vrh $s \in V$ tražimo najkraći put u grafu do svakog od preostalih vrhova $t \in V$.
- **Najkraći put do jednog odredišta** (eng. *Single-Destination Shortest-Path Problem*): zadan je ciljni vrh $t \in V$, potrebno je pronaći najkraći put od s do t, za svaki vrh $s \in V$. Invertiranjem smjerova bridova u polaznom grafu možemo ovu varijantu svesti na problem najkraćeg puta iz jednog vrha.
- **Najkraći put između dva vrha** (eng. *Single-Pair Shortest-Path problem*): potrebno je pronaći najkraći put između početnog $s \in V$ i završnog vrha $t \in V$. Ova varijanta nastoji ubrzati problem izbjegavanjem nepotrebnog računanja najkraćih putova do svih odredišnih vrhova. Međutim, za sad poznati algoritmi imaju isto asimptotsko vrijeme izvršavanja u najgorem slučaju kao algoritmi najkraćeg puta iz jednog vrha.
- **Najkraći put između svaka dva vrha** (eng. *All-Pairs Shortest-Paths Problem*): zadatak je pronaći najkraći put između svaka dva vrha s i t. Ovaj problem moguće je riješiti pokrećući algoritam najkraćeg puta iz svakog vrha grafa posebno, no postoje prikladniji algoritmi specificirani za ovu vrstu problema.

Algoritmi namijenjeni za jednu varijantu problema najkraćeg puta u grafu se lako mogu primijeniti i na ostalim verzijama. Naravno, algoritmi se mogu kreirati i modificirati kako bi pružili što bolje rješenje određene varijante. Primjeri algoritama za problem najkraćeg puta između svaka dva vrha je *Floyd-Warshallov* algoritam. Od bržih algoritama ističe se primjena *Highway hierarchies* algoritma, vidljiva u [10]. U daljnjem dijelu orijentirat ćemo se na varijantu problema najkraćeg puta između dva vrha, primijenjenu na težinskom usmjerenom grafu s nenegativnim težinama.

2.2 Primjene algoritma

Modeliranjem raznih problema grafovima često se javlja potreba za računanjem najkraćih putova. Primjena u prometnoj mreži (cestovna, željeznička, zrakoplovna) je jedan od primjera problema, koji će služiti i kod uspoređivanja obrađenih metoda. Primjene možemo pronaći i u: bazama podataka koje koriste strukturu grafa za semantičke upite i pohranu podataka, web pretraživanju i usmjeravanju podataka na internetu, prepoznavanju govora (eng. *speech recognition*) [13], segmentaciji slika (eng. *image segmentation*), raznim primjenama kod društvenih mreža poput ciljanog marketinga.

Poglavlje 3

Klasični algoritmi

U ovom odjeljku opisujemo najpoznatiji algoritam za pronalazak najkraćeg puta u grafu – *Dijkstrin* algoritam. Donosimo prikaz poboljšanja vremenske složenosti korištenjem prikladnijih struktura podataka, dvosmjernog pretraživanja i heurističkih varijanti koje usmjeravaju traženje te tako smanjuju područje pretrage. Zajedničko svim ovim algoritmima je to što ne zahtijevaju predprocesiranje podataka, već direktno rješavaju problem najkraćeg puta.

3.1 Dijkstrin algoritam

Edsger W. Dijkstra osmislio je algoritam za pronalazak najkraćeg puta od nekog početnog čvora do svih ostalih u težinskom grafu s nenegativnim težinama bridova. Izvorni algoritam može se pronaći u [5]. Algoritam krenuvši od početnog vrha, u svakoj iteraciji pronađe najkraći put do najbližeg vrha koji do tada nije pronađen. Prekinemo li pretragu čim dođemo do traženog ciljnog vrha, Dijkstrin algoritam služi za rješavanje problema najkraćeg puta između zadana dva vrha.

Neka je $s \in V$ zadani početni, a $t \in V$ završni čvor. Želimo pronaći najkraći put od s do t, $d_G(s,t)$. Algoritam postupno stvara rješenje koristeći posebno polje D, veličine broja vrhova $|V|$. Vrijednost $D[v]$ predstavlja određenu gornju ogradu na udaljenost od s do v, tj. vrijedi

$$D[v] \geq d_G(s,v). \tag{3.1}$$

Ideja je iterativno obnavljati polje D tako da uvijek vrijedi zadana nejednakost, s tim da za svaki vrh v u jednom trenutku postane $D[v] = d_G(s,v)$ i ta jednakost ostaje do kraja izvršavanja. Algoritam staje kada spomenuta jednakost vrijedi za ciljni vrh t. Kako promatramo problem najkraćeg puta između zadana dva vrha i prekidamo pretraživanje čim ga pronađemo, u tom slučaju će vrijediti $D[v] = d_G(s,v)$ za sve vrhove v takve da je

$d_G(s,v) < d_G(s,t)$ ili $d_G(s,v) = d_G(s,t)$ uz uvjet $v < t$. Ovdje pretpostavljamo neki fiksni uređaj vrhova, koji interpretiramo redoslijedom kojim ih raspoređujemo u polje D.

Obnavljanje polja D svodi se na primjene jednostavne ideje, nazvane *relaksacija brida*. Ako u nekom trenutku postane $D[v] = d_G(s,v)$, za neki $v \in V$, tada za svaki njemu susjedni vrh w znamo da postoji put $P_w = \langle s, \ldots, v, w\rangle$ težine $D[v] + \omega(v,w)$. Ako je pronađena gornja ograda $D[w]$ veća od težine puta P_w tada postavljamo $D[w] = \omega(P_w) = D[v] + \omega(v,w)$. Kako je pronađeni put P_w u tom slučaju samo jedan od raznih mogućih putova, sigurno je njegova težina gornja granica za najkraći traženi put, tj. vrijedi nejednakost (3.1).

Definicija 3.1.1. *Skup svih vrhova idejno se dijeli na tri dijela: neposjećene, posjećene i obrađene vrhove. Za vrh $v \in V$ kažemo da je: neposjećen ako vrijedi $D[v] = +\infty$, posjećen ako $D[v] < +\infty$ i $D[v] > d_G(s,v)$, a obrađen ako smo pronašli najkraći put, kada vrijedi $D[v] = d_G(s,v)$.*

Kako bi se mogao rekonstruirati najkraći put, u posebnom polju P pamti se za svaki vrh w, indeks vrha v koji je prethodnik vrha w u trenutnom pronađenom najboljem rješenju. Pseudokod Dijkstrinog algoritma nalazi se u algoritmu 1.

Teorem 3.1.2. *Dijkstrin algoritam proveden na težinskom, usmjerenom grafu $G = (V, E, \omega)$ s nenegativnom težinskom funkcijom ω i početnim vrhom s, pri završetku daje vrijednosti najkraćih putova, $D[v] = d_G(s,v)$ za svaki $v \in V$.*

Dokaz. Verzija koja pronalazi najkraće putove do svih vrhova dobije se sitnom izmjenom u *while* dijelu algoritma 1, uklanjanjem uvjeta $c \neq t$. Ideja dokaza je pokazati da se u svakom koraku iteracije pronađe najkraći put pojedinog vrha, i to u redoslijedu od najbližeg do najudaljenijeg vrha od s. Detalji se mogu pronaći u [3]. □

Iz redoslijeda *obrađivanja* vrhova, od najbližeg do najudaljenijeg, iz dokaza teorema 3.1.2 slijedi ispravnost Dijkstrinog algoritma koji rješava problem najkraćeg puta između zadana dva vrha.

Za daljnju analizu vremenske složenosti Dijkstrinog algoritma uvodimo oznake: $n := |V|$ broj vrhova i $m := |E|$ broj bridova u grafu. Inicijalizacijski dio je reda veličine $O(n)$. Relaksacija brida izvodi se u konstantnoj složenosti. Pošto će svaki brid (u,v) biti relaksiran maksimalno jednom – prije jedinstvenog označavanja vrha v oznakom *obrađen*, provodimo postupak relaksacije najviše m puta. Direktan pristup susjedima pojedinog vrha, potreban za istinitost prethodne tvrdnje, ostvariv je npr. korištenjem popisa susjeda obrađenog u 1.2. U originalnoj verziji algoritma, postupak pronalaska novog čvora c najbližeg početnom, koji nije *obrađen*, ostvaruje se linearnim pretraživanjem polja D, složenosti $O(n)$. Taj će se postupak u najgorem slučaju ponoviti $O(n)$ (točnije, $n-1$) puta. Ukupna složenost algoritma u ovakvoj implementaciji iznosi $O(n)+m+(n-1)O(n) = O(n^2)$.

Data: Graf $G = (V, E, \omega)$, početni $s \in V$ i krajnji vrh $t \in V$.
Result: Polje D udaljenosti od vrha s, polje P za rekonstrukciju najkraćeg puta.

---

forall $v \in V$ **do**
  $D[v] = 0$ ▷ Inicijalizacija.
  $P[v] = -1$
end
$D[s] = 0$ ▷ Udaljenost do početnog vrha je 0.
$c = s$ ▷ Vrh c je trenutni, neobrađeni, najbliži vrh vrhu s.
$i = 0$
while $c \neq t$ ***and*** $i < |V| - 1$ **do**
  ▷ Pokušaj popraviti rješenje putom do preostalih vrhova preko vrha c.
  forall *w povezan s c,* $(c, w) \in E$ ***and*** *w nije obrađen* **do**
    if $D[w] > D[c] + \omega(c, w)$ **then**
      $D[w] = D[c] + \omega(c, w)$ ▷ Relaksacija brida.
      $P[w] = c$
    end
  end
  označi c oznakom *obrađen* ▷ Obrađen čvor c.
  $c \leftarrow \text{argmin}\{D[x] \mid D[x] < +\infty \text{ i } x \text{ nije obrađen}\}$ ▷ Neobrađen, najbliži čvor od s.
  ++i
end

Algoritam 1: Dijkstrin algoritam

Kako bi se izbjeglo pretraživanje cijelog polja D kod pronalaska novog vrha c u algoritmu 1, uvodi se korištenje *prioritetnog reda*. Takva struktura podataka služi za pohranjivanje *posjećenih* vrhova, što će dati značajno ubrzanje za *rijetke* grafove.

Postoje razni načini implementacije prioritetnog reda. Od svake se zahtijeva implementacija funkcija:

- **Ubaci s prioritetom**: struktura se sortira uzimajući u obzir prioritet vrha. Ubacivanje novog vrha obavlja se tako da se očuva sortiranost strukture u nepadajućem poretku. Prioritet u slučaju Dijkstrinog algoritma bi bila najmanja pronađena udaljenost ($D[v]$ za vrh v) od početnog vrha s.
- **Obnovi prioritet**: potrebno je za zadani vrh obnoviti njegov prioritet u prioritetnom redu te ispravno ga smjestiti na novo mjesto u strukturi.
- **Pronađi minimalni element**: operacija koja vraća element s najmanjim prioritetom, u ovom slučaju element s najmanjom udaljenosti od početnog vrha.

- **Obriši minimalni element**: operacija uklanja minimalni element iz prioritetnog reda – vrh s minimalnim prioritetom, najmanjom pronađenom udaljenosti od početnog vrha.

Implementacija Dijkstrinog algoritma koja koristi prioritetni red za pronalaženje najbližeg *posjećenog* vrha u novoj iteraciji ima asimptotski drugačiju vremensku složenost. Kod binarne hrpe (eng. *binary heap*), pomoću koje se može implementirati prioritetni red, funkcije ubacivanja, obnavljanja i brisanja minimalnog elementa realizirane su u složenosti $O(\log n)$. Pronalazak minimalnog elementa je konstantne vremenske složenosti, $O(1)$. Ukupna složenost iznosi $O(m \log n)$. Koristeći Fibonaccijevu hrpu (eng. *Fibonacci heap*) vrijeme izvršavanja svodi se na $O(n \log n + m)$.

Napomena 3.1.3. *Navedene modifikacije daju asimptotski bolja rješenja ukoliko se algoritam izvršava na grafu s relativno manjim brojem vrhova. Za gust graf vrijedi da je m reda veličine $O(n^2)$ te se u tom slučaju ukupna složenost opet svede na $O(n^2)$.*

Detaljnija analiza i daljnja razmatranja mogu se pronaći u [1]. Implementacija u jeziku *C++* koja koristi iz STL biblioteke klasu *set* za potrebe prioritetnog reda vidljiva je u [11].

3.2 A^* pretraživanje

Algoritam A^* koristi heurističku funkciju kako bi usmjerio pretraživanje prema ciljnom vrhu. Ovisno o informacijama o domeni problema, konstruiraju se prigodne heuristike za procjenu udaljenosti do odredišta. Općenito heuristički pristup sam za sebe neće dati uvijek optimalno rješenje, no ovdje se kombinira s Dijkstrinim algoritmom koji osigurava ispravnost rješenja. Točnije, algoritam A^* radi slično kao Dijkstrin algoritam, samo što pri novom odabiru najbližeg vrha v koji još nije *obrađen* ne uzima takav v koji ima najmanju vrijednost $D[v]$ već onaj vrh s najmanjom vrijednosti $k(v) = D[v] + \pi_t(v)$. Funkcija $\pi_t \colon V \to \mathbb{R}$ pridružuje svakom vrhu procjenu njegove udaljenosti do krajnjeg vrha t. Za ispravnost algoritma bitno je da funkcija π_t nikad ne precjenjuje udaljenost, tj. da vrijedi $\pi_t(v) \leq d_G(v, t)$.

Uvodimo nekoliko definicija i rezultata vezanih za izmijenjene težinske funkcije grafa iz kojih će lako slijediti ispravnost A^* algoritma. Funkcija π pridaje realne vrijednosti vrhovima grafa, što u primjeru najkraćeg puta možemo promatrati kao procjenu udaljenosti do ciljnog vrha.

Definicija 3.2.1. *Neka su zadani graf $G = (V, E, \omega)$ i funkcija $\pi \colon V \to \mathbb{R}$. Reducirana funkcija težine bridova s obzirom na funkciju π definira se kao:*

$$\omega_\pi(v, w) = \omega(v, w) - \pi(v) + \pi(w). \tag{3.2}$$

Pretpostavimo da u grafu G zamijenimo težinsku funkciju bridova ω s ω_π. Proizvoljan put od vrha v do vrha w tada se promijeni za fiksnu vrijednost: $\pi(w) - \pi(v)$, za sve $v, w \in V$.

Lema 3.2.2. *Neka je zadan graf $G = (V, E, \omega)$, funkcija $\pi\colon V \to \mathbb{R}$ i pripadna reducirana funkcija težine bridova ω_π. Za proizvoljan put $P = \langle v, x_1, \ldots, x_k, w\rangle$ vrijedi:*

$$\omega_\pi(P) = \omega(P) + \pi(w) - \pi(v). \tag{3.3}$$

Dokaz. Tvrdnja slijedi korištenjem definicija 1.1.2 i 3.2.1:

$$\begin{aligned}\omega_\pi(P) &= \omega_\pi(v, x_1) + \omega_\pi(x_1, x_2) + \cdots + \omega_\pi(x_k, w)\\ &= \big[\omega(v, x_1) - \pi(v) + \pi(x_1)\big] + \big[\omega(x_1, x_2) - \pi(x_1) + \pi(x_2)\big] + \cdots + \big[\omega(x_k, w) - \pi(x_k) + \pi(w)\big]\\ &= \omega(v, x_1) + \omega(x_1, x_2) + \cdots + \omega(x_k, w) - \pi(v) + \pi(w) = \omega(P) - \pi(v) + \pi(w) \quad \square\end{aligned}$$

Korolar 3.2.3. *Promjena duljine puta $\omega_\pi(P)$ ovisi o vrijednostima funkcije π samo u početnoj i krajnjoj točki puta. Prema tome, zaključujemo da je put P najkraći put od v do w s obzirom na težinsku funkciju ω ako i samo ako je takav i s obzirom na težinsku funkciju ω_π.*

Definicija 3.2.4. *Funkciju $\pi\colon V \to \mathbb{R}$ nazivamo izvedivom ako je njena reducirana funkcija težine nenegativna za svaki brid grafa G, tj. $\omega_\pi(v, w) \geq 0$ za svaki $(v, w) \in E$.*

Napomena 3.2.5. *A^* pretraživanje koje odabire idući vrh u pretrazi uzimajući u obzir vrh s najmanjom vrijednosti $D[v] + \pi_t(v)$ ekvivalentno je Dijkstrinom algoritmu na grafu s izmijenjenom funkcijom težine ω_{π_t}. Originalna težina brida $\omega(u, v)$ zamjenjuje se $\omega(u, v) - \pi_t(u) + \pi_t(v)$, za svaki brid $(u, v) \in E$. Ako je takva funkcija težine izvediva, izmijenjene težine bridova su nenegativne te po teoremu 3.1.2 vrijedi ispravnost algoritma. Naglasimo da takva pretraga obrađuje vrhove od najbližih prema najdaljima od početnog vrha i staje kada se pronađe udaljenost do ciljnog vrha t.*

Lema 3.2.6. *Neka je funkcija $\pi\colon V \to \mathbb{R}$ izvediva takva da za neki vrh $t \in V$ vrijedi $\pi(t) \leq 0$. Tada za svaki vrh $v \in V$ vrijedi nejednakost $\pi(v) \leq d_G(v, t)$.*

Dokaz. Neka je $v \in V$ fiksiran vrh. Ukoliko ne postoji put od v do vrha t, tvrdnja trivijalno vrijedi pošto je tada $d_G(v, t) = +\infty$. Pretpostavimo stoga da postoji put između v i t, te neka je P najkraći takav put. Vrijedi:

$$0 \leq \omega_\pi(P) = \omega(P) + \pi(t) - \pi(v) \leq d_G(v, t) - \pi(v) \tag{3.4}$$

iz čega slijedi tvrdnja leme. Prva nejednakost posljedica je izvedivosti funkcije π, iz koje slijedi da je težina svakog brida, pa i svakog puta nenegativna. Druga jednakost vrijedi iz leme 3.2.2. Zadnja nejednakost je definicija najkraćeg puta i pretpostavka $\pi(t) \leq 0$. $\square$

Lema 3.2.7. *Neka su π_1 i π_2 dvije izvedive funkcije. Tada je funkcija $p = \max(\pi_1, \pi_2)$ također izvediva.*

Dokaz. Uzmimo proizvoljan brid $(v, w) \in E$. Želimo dokazati $\omega_p(v, w) \geq 0$. Bez smanjenja općenitosti možemo pretpostaviti da je $\pi_1(v) \geq \pi_2(v)$. U suprotnom dokaz provodimo sasvim analogno. Nadalje, ako je $\pi_1(w) \geq \pi_2(w)$ tada vrijedi:

$$\omega_p(v, w) = \omega(v, w) - p(v) + p(w) = \omega(v, w) - \pi_1(v) + \pi_1(w) \geq 0. \tag{3.5}$$

Preostaje još promotriti slučaj kada je $\pi_1(w) \leq \pi_2(w)$. Imamo:

$$\omega_p(v, w) = \omega(v, w) - p(v) + p(w) = \omega(v, w) - \pi_1(v) + \pi_2(w) \geq \omega(v, w) - \pi_1(v) + \pi_1(w) \geq 0. \tag{3.6}$$

Posljednje nejednakosti u (3.5) i (3.6) slijede iz definicije izvedivosti funkcije π_1. □

Iz prethodnih lema zaključujemo da na svaku *izvedivu* funkciju π_t možemo gledati kao na aproksimaciju, donju ogradu na vrijednost najkraće udaljenosti do ciljnog vrha t. Također, ako imamo dvije takve funkcije, uzimajući po točkama maksimume, dobijemo opet *izvedivu funkciju*. Pretraživanje A^* radi logičnu stvar – u svakom koraku bira vrh preko kojeg put od vrha s do vrha t ima najmanju procijenjenu udaljenost. Slijedi teorem koji potvrđuje intuiciju da bolja procjena udaljenosti rezultira obrađivanjem manjeg skupa vrhova.

Teorem 3.2.8. *Neka su π'_t i π_t dvije izvedive funkcije takve da vrijedi $\pi'_t(t) = \pi_t(t) = 0$ i za svaki vrh $v \in V$ vrijedi nejednakost $\pi'_t(v) \geq \pi_t(v)$. Tada je skup obrađenih vrhova prilikom A^* pretraživanja koristeći funkciju π'_t podskup skupa vrhova obrađenih A^* pretragom uz funkciju π_t.*

Dokaz. Pretpostavimo da pretraga koristeći funkciju π_t ne *obradi* neki vrh $v \in V$. Potrebno je pokazati da tada niti pretraga uz π'_t ne *obradi* vrh v.

Ukoliko je vrh v nedohvatljiv iz početnog vrha s, tj. ako ne postoji put od s do v koristeći funkciju π_t tada ne postoji takav put niti kod pretrage s funkcijom π'_t. Dodatne funkcije usmjeravaju pretragu mijenjajući težine bridova, dok postojanje bridova ostaje nepromijenjeno. U ovom slučaju očito onda niti A^* pretraga s π'_t ne *obradi* vrh v.

U daljnjem dijelu pretpostavljamo da su vrhovi s i v povezani. Kako se kod Dijkstrinog algoritma utvrđuju najkraće udaljenosti od najmanjih prema najvećima, tj. tim se redom *obrađuju* vrhovi, imamo dva slučaja: vrh v je udaljeniji od vrha s nego vrh t i stoga je pretraga stala nakon vrha t ili im je udaljenost od vrha s jednaka no vrh t ima manji redni broj od vrha s, stoga prije dolazi na obradu. Tada uzimajući u obzir korolar 3.2.3 vrijedi:

$$d_G(s, v) - \pi_t(s) + \pi_t(v) > d_G(s, t) - \pi_t(s) + \pi_t(t) \quad \text{ili} \tag{3.7}$$

$$d_G(s, v) - \pi_t(s) + \pi_t(v) = d_G(s, t) - \pi_t(s) + \pi_t(t) \quad \text{i } v > t. \tag{3.8}$$

Kako je $\pi_t(t) = 0$, nejednakost (3.7) ekvivalentna je nejednakosti $d_G(s, v) + \pi_t(v) > d_G(s, t)$, dok je u drugom slučaju, jednakost (3.8) ekvivalentna $d_G(s, v) + \pi_t(v) > d_G(s, t)$ uz $v > t$. Iskoristimo li $\pi'_t(v) \geq \pi_t(v)$ imamo:

$$d_G(s, v) + \pi'_t(v) > d_G(s, t) \quad \text{ili} \tag{3.9}$$
$$d_G(s, v) + \pi'_t(v) \geq d_G(s, t) \quad \text{i } v > t. \tag{3.10}$$

Iz toga zaključujemo:

$$d_G(s, v) - \pi'_t(s) + \pi'_t(v) > d_G(s, t) - \pi'_t(s) + \pi'_t(t) \quad \text{ili} \tag{3.11}$$
$$d_G(s, v) - \pi'_t(s) + \pi'_t(v) = d_G(s, t) - \pi'_t(s) + \pi'_t(t) \quad \text{i } v > t. \tag{3.12}$$

Prvi slučaj znači da je u grafu G s reduciranom funkcijom težine π'_t vrh t bliži vrhu s nego što je to vrh v. U tom slučaju ćemo algoritmom prije pronaći najkraću udaljenost do vrha t, od početnog vrha s, te tada algoritam staje i vrh v ostaje *neobrađen*. U drugom slučaju se dogodi ista stvar: udaljenosti vrhova v i t do vrha s su jednake, no vrh t ima prednost pri obradi i tako ostavlja vrh v *neobrađenim*. □

Primijetimo da je Dijkstrin algoritam zapravo izvođenje A^* algoritma s funkcijom $\pi(v) = 0$ za svaki vrh $v \in V$. Iz teorema 3.2.8 zaključujemo da svaka usmjerena pretraga algoritmom A^* do ciljnog vrha t uz procjenu udaljenosti *izvedivom* funkcijom π nikad ne *obradi* više vrhova nego Dijkstrin algoritam pronalaska najkraćeg puta.

Primjer *izvedive* funkcije

Promatrajući prometnu mrežu, lako možemo dobiti *izvedivu* funkciju koja se može upotrijebiti u A^* algoritmu i tako smanjiti prostor pretrage kod najkraćeg puta. Ukoliko znamo koordinate sjecišta prometnica ili gradova, tj. vrhova u grafu, kao procjenu udaljenosti do ciljnog vrha možemo uzeti *euklidsku* udaljenost. Npr. za dvije točke $x = (x_1, x_2)$, $y = (y_1, y_2)$ u dvodimenzionalnom prostoru, euklidska udaljenost računa se izrazom:

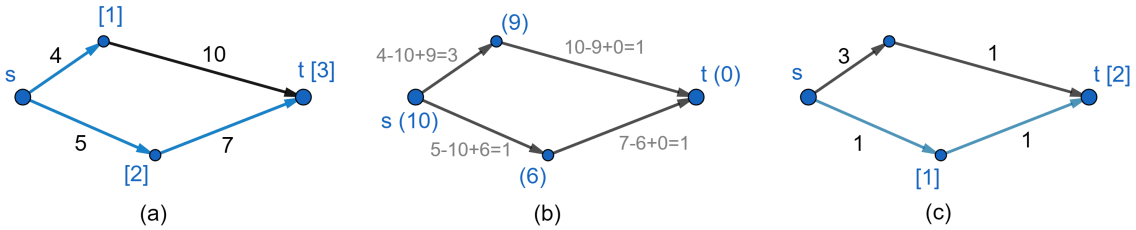
$$\pi_E(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}. \tag{3.13}$$

Lema 3.2.9. *Neka je zadan ciljni vrh $t \in V$. Definiramo funkciju $\pi_{E,t} \colon V \to \mathbb{R}$ kao euklidsku udaljenost vrha do ciljnog vrha t. Točnije, za svaki $v \in V$ definiramo $\pi_{E,t}(v) := \pi_E(v, t)$. Tako definirana funkcija je izvediva.*

Dokaz. Kako je riječ o prometnoj mreži, euklidsku udaljenost možemo promatrati kao zračnu udaljenost dvaju točaka te je ona kao takva sigurno donja ograda za prometnu udaljenost, tj. vrijedi $\pi_E(u, v) \leq d_G(u, v)$ za sve vrhove $u, v \in V$. Posebno, za $(u, v) \in E$ vrijedi $\pi_E(u, v) \leq \omega(u, v)$. Zaključujemo:

$$\omega(u, v) + \pi_{E,t}(v) \geq \pi_E(u, v) + \pi_{E,t}(v) \geq \pi_{E,t}(u). \tag{3.14}$$

Zadnja nejednakost je zapravo $\pi_E(u,v) + \pi_E(v,t) \geq \pi_E(u,t)$ što je istina po nejednakosti trokuta. Iz (3.14) zaključujemo $\omega_{E,t}(u,v) \geq 0$ za svaki brid $(u,v) \in E$, tj. funkcija $\pi_{E,t}$ je izvediva. □



Slika 3.1: Usporedba *obrađenih* vrhova - Dijkstrin (a) i A^* algoritam (c).

Primjer 3.2.10. *Jednostavan primjer grafa sa slike 3.1 prikazuje situaciju u kojoj A^* algoritam ubrza pretragu najkraćeg puta, pretražujući manje vrhova od Dijkstrinog algoritma. Cilj je pronaći najkraći put od vrha s do vrha t. Slika 3.1 (a) prikazuje pretragu koristeći Dijkstrin algoritam. Unutar uglatih zagrada označeni su redni brojevi vrhova redom kojim su obrađeni u pretrazi. Plava boja označava relaksirane bridove, tj. one bridove koje je algoritam pretražio. Na slici 3.1 (b) unutar okruglih zagrada prikazana je euklidsku udaljenost $\pi_{E,t}$ za svaki vrh grafa. Također, prikazana je i transformacija težina bridova – računaju se vrijednosti reducirane funkcije težine $\omega_{\pi_{E,t}}$. Slika 3.1 (c) prikazuje obrađene vrhove i bridove kod A^* pretrage koristeći euklidsku funkciju za procjenu udaljenosti do cilja.*

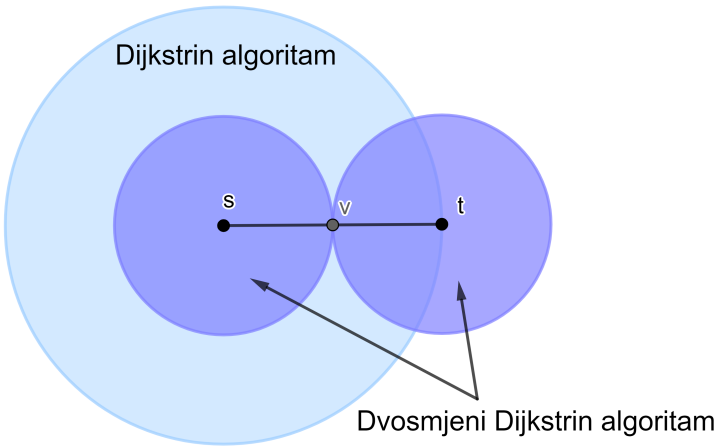
3.3 Dvosmjerno pretraživanje

Dvosmjerno pretraživanje uz pretragu od početnog prema odredišnom vrhu uvodi i pretragu u suprotnom smjeru. Provodi se pretraga od završnog prema početnom vrhu koju nazivamo *pretraga unazad*. Pretraga od početnog vrha s prema ciljnom vrhu t naziva se *pretraga unaprijed*. U *pretrazi unazad* pamtimo za svaki vrh pronađenu udaljenost do krajnjeg vrha t. Kod ovakve vrste pretrage koristi se izmijenjeni početni graf, kod kojeg su orijentacije bridova obrnute.

Pošto se obavljaju dva pretraživanja, *unaprijed* i *unazad*, moguće su razne opcije kojim ih redom provoditi. Jedna od njih je držati pretrage razdvojeno i alternirati njihove korake izvođenja. Druga opcija je, npr. kod Dijkstrinog algoritma izvedenog uz pomoć strukture podataka prioritetnog reda, *posjećene* vrhove oba pretraživanja čuvati u istom prioritetnom redu. U svakom koraku tada biramo vrh s najmanjom vrijednosti (udaljenosti od ciljnog ili

početnog vrha), i nastavljamo odgovarajuće pretraživanje. Oba pristupa rezultiraju korektnim algoritmima. Za više detalja pogledati [14] i [8].

Kriterij zaustavljanja razlikuje se ovisno o algoritmu kojim provodimo obje pretrage. Ukoliko je Dijkstrin algoritam izbor, prilikom inicijalizacije *obrade* se početni s i krajnji vrh t. Algoritam pamti najkraći pronađeni put od vrha s do vrha t, duljine μ. Odgovarajući put rekonstruira se slično kao kod Dijkstrinog algoritma, pamteći za svaki vrh njegovog prethodnika na tom putu. Početno, μ iznosi $+\infty$. U trenutku kada se odradi *relaksacija* brida $(u, v) \in E$ u pretraživanju *unaprijed*, znamo da vrh u postaje *obrađen*, tj. pronađen je najkraći put od početnog vrha s. Ukoliko je vrh v u tom trenutku već *obrađen* u pretrazi *unazad*, znamo najkraći put od v do t. Tada imamo novog kandidata za rješenje: put od s do u koji se nastavlja bridom (u, v) te nadalje putem od v do t. Ukoliko je $\mu > D_s[u] + \omega(u, v) + D_t[v]$, pronađen je kraći put od s do t. Obnavlja se vrijednost μ i pripadni prethodnici korišteni za rekonstrukciju puta. Oznake su slične kao kod Dijkstrinog algoritma: polje D_s pamti pronađene najkraće putove od početnog vrha s, dok u polju D_t na mjestu vrha v piše duljina pronađenog najkraćeg puta od v do krajnjeg vrha t. Sasvim analogno se postupa ukoliko se vrh v *obradi* pretraživanjem *unazad*, kada je vrh u već *obrađen* pretragom *unaprijed*. Algoritam završava kada oba pretraživanja *obrade* isti vrh $v \in V$. Najkraći traženi put ne mora nužno prolazit kroz taj vrh v, već kroz onaj koji je zadnji promijenio vrijednost μ.



Slika 3.2: Usporedba prostora pretrage klasičnog i dvosmjernog Dijkstrinog pretraživanja

Što se tiče prostora pretrage, možemo zamišljati da Dijkstrin algoritam pretražuje prostor kugle s centrom u početnom vrhu, radijusa duljine najkraćeg puta do ciljnog vrha. Dvosmjerni Dijkstrin algoritam opisan u ovom poglavlju pretražuje dvije kugle, dok se pretrage ne susretnu. Radijusi tih kugli približno su pola vrijednosti traženog najkraćeg puta, dok

su centri u početnom i krajnjem vrhu. Promatramo li dvodimenzionalnu primjenu kod prometnih mreža, dvosmjerno pretraživanje *obradi* upola manji broj vrhova od klasičnog Dijkstrinog algoritma. Više detalja moguće je pronaći u [8]. Ilustracija približnog prostora pretrage vidljiva je na slici 3.2.

Kod dvosmjernog A^* pretraživanja uvjet zaustavljanja postaje malo kompliciraniji. Neka je π_t funkcija korištena u pretrazi *unaprijed*, dok se kod obrnute pretrage koristi π_s. Funkcije π_t i π_s nazvat ćemo *konzistentnima* ukoliko vrijedi $\pi_t + \pi_s = c$ za neku konstantu $c \in \mathbb{R}^+$. Korištenje *konzistentnih* funkcija ostavlja isti kriterij zaustavljanja kao kod Dijkstrinog algoritma, čim obje pretrage *obrade* isti vrh grafa. Ipak, ovakav uvjet je poprilično restriktivan kod odabira heurističkih funkcija. *Simetrični* pristup dozvoljava slobodu odabira funkcija π_s i π_t, samim time i prikladnijih kod procjene udaljenosti do ciljnog/početnog vrha. Nedostatak je što takav algoritam ne pronalazi rješenje u trenutku kada se obje pretrage susretnu, već je potrebna daljnja obrada. Pretraga staje kada *unaprijed* ili *unazad* pretraživanje obrade vrh v za koji vrijedi $D_s[v] + \pi_t(v) \geq \mu$ ili kada niti jedno pretraživanje nema *posjećenih* vrhova za obrađivanje. Opisani *simetrični* pristup osigurava korektnost algoritma koja se vidi iz redoslijeda kojim se u pretragama *obrađuju vrhovi* – od najbližih prema udaljenijima. U trenutku zaustavljanja, najkraći traženi put je već morao biti pronađen. Više detalja o dvosmjernom A^* pretraživanju, *konzistentnom* i *simetričnom* pristupu dostupno je u [8].

Poglavlje 4

Contraction Hierarchies – CH

U mnogim primjerima upotrebe najkraćeg puta, poput primjene u cestovnim mrežama, neke bridove možemo proglasiti „značajnijima" od ostalih. Konkretno, prometnice se mogu podijeliti na lokalne, županijske, državne ceste i autoceste. Očekivani najbrži put od mjesta A do mjesta B bi koristio prometnice niže razine do najbližeg spajanja na autocestu kojom bi se što bliže približili odredišnom mjestu B. Nakon toga, potrebno je napustiti autocestu te manjim cestama doći do odredišta. Iz primjera možemo uočiti moguće prednosti klasifikacije bridova u grafu u kojemu tražimo najkraći put. Ukoliko postoji određena hijerarhija među bridovima, hijerarhijski algoritmi provode obostranu pretragu, iz početnog i završnog vrha, krenuvši od bridova s nižom prema bridovima s višom hijerarhijom. Motivirani hijerarhijom bridova, u ovom dijelu ćemo opisati algoritam koji radi sličnu stvar. Kako vrhovi predstavljaju prometna čvorišta, umjesto bridova određivati će se hijerarhija vrhova. Da bi ovakav pristup bio efikasan, potrebni su dodatni podaci o vrhovima i bridovima grafa, tj. potrebna je njihova smislena hijerarhija. Kako u općenitom slučaju za zadani graf nemamo takve podatke, oni se posebno računaju u fazi koju nazivamo *predprocesiranje*. Rezultat predprocesiranja su dodatne informacije o grafu koje će ubrzati vrijeme izvršavanja upita, pronalazak najkraćeg puta između dva vrha.

Contraction Hierarchies algoritam (vidjeti [7]) koristi hijerarhijski pristup problemu pronalaska najkraćeg puta u grafu. Kako bi se ubrzalo vrijeme izvršavanja pronalaska najkraćeg puta između dva vrha, procedure zvane *faza upita*, ideja je u početni graf dodati nove bridove zvane *prečaci* (eng. *shortcuts*). Uz to, svakom vrhu grafa pridruži se jedinstveni prirodni broj, iz čega slijedi hijerarhija vrhova. U modificiranom početnom grafu vrijeme pronalaska najkraćeg puta upotrebom dvosmjernog Dijkstrinog pretraživanja bit će značajno kraće u odnosu na klasične algoritme opisane u odjeljku 3, koji ne koriste predprocesiranje.

Formalno, za zadani usmjereni graf $G = (V, E, \omega)$ predprocesiranje definira dodatni skup bridova prečaca E' te pridružuje svakom vrhu $v \in V$ prirodan broj $rank(v)$. Funkcija

$rank\colon V \to \mathbb{N}$ svakom vrhu pridružuje jedinstveni prirodni broj u rasponu $[0, n]$, gdje je $n = |V|$. Rezultantni graf označavamo s $G^* = (V, E^*, \omega^*)$, pri čemu je $E^* = E \cup E'$, te za $e \in E^*$ vrijedi $\omega^*(e) = \omega(e)$ ukoliko je $e \in E$. U suprotnom je $\omega^*(e)$ duljina prečaca koji je ubačen predprocesiranjem. Iznimka je moguća radi izbjegavanja višestrukih bridova između dva vrha, kada se težina postojećeg brida smanji kao rezultat prečaca, opisano u uvjetu 2 kontrakcije vrhova.

Hijerarhijski pristup opisan na početku poglavlja očitava se u *fazi upita*, prilikom računanja najkraćeg puta između zadanih vrhova. Vrhovi većeg značaja su vrhovi s većom *rank* vrijednosti. Ideja je obostranom pretragom tražiti put koji koristi vrhove uzlazno po hijerarhiji, od početnog i krajnjeg prema sredini puta koja ima najveći prioritet. Kako bi se to ostvarilo, faza upita izvršavat će dvosmjernim Dijkstrinim pretraživanjem (odjeljak 3.3) na modificiranom grafu G^*. Ključno poboljšanje u odnosu na klasični Dijkstrin algoritam leži baš u hijerarhijskom odabiru kod pretraživanja, čime će se u pravilu *obraditi* značajno manji broj vrhova.

4.1 Predprocesiranje

Ključna operacija predprocesiranja u *Contraction Hierarchies* algoritmu naziva se *kontrakcija vrhova*. Potrebno je ukloniti vrh iz grafa tako da se očuvaju najkraći putovi između preostalih vrhova u tako dobivenom izvedenom grafu. Kako bi se to ostvarilo, u nekim slučajevima potrebno je dodati nove bridove - prečace. Ovisno kojim redom se biraju vrhovi kod kontrakcije, broj dodanih prečaca značajno varira. Cilj je dodati što manje takvih bridova, kako se ne bi bespotrebno previše povećala gustoća novog grafa s dodanim bridovima te tako usporila dvosmjerna pretraga u fazi upita.

Kontrakcija vrhova

Kontrakcija vrhova je proces izbacivanja vrha i njemu incidentnih bridova iz grafa, čuvajući vrijednosti najkraćih putova između preostalih vrhova u grafu.

Neka je zadan graf $G = (V, E, \omega)$ iz kojeg želimo izbaciti vrh $v \in E$. Neka su $\{u_1, u_2, \ldots, u_l\}$ vrhovi ulaznih bridova vrha v, tj. vrhovi za koje vrijedi $(u_i, v) \in E$. Slično, vrhove izlaznih bridova vrha v označimo s $\{w_1, w_2, \ldots, w_k\}$, za njih vrijedi $(v, w_j) \in E$. Ukoliko je najkraći put između proizvoljnih vrhova $s, t \in V \setminus \{v\}$ prekinut zbog izbacivanja vrha v i njegovih bridova, tada je takav put oblika $\langle s, \ldots, u_i, v, w_j, \ldots, t\rangle$, za neke $i \in \{1, \ldots l\}, j \in \{1, \ldots k\}$. Stoga, kako se izbacuju bridovi (u_i, v) i (v, w_j), potrebno je razmotriti jedino dodavanje prečaca, novog brida (u_i, w_j) težine $\omega(u_i, v) + \omega(v, w_j)$.

Kako bi se ustanovila potreba za dodavanjem prečaca, potrebno je provesti lokalnu pretragu najkraćeg puta od svakog vrha u, gdje je $(u, v) \in E$, prema svakom vrhu w, takvom

da je $(v, w) \in E$. U graf dodajemo brid (u, w) težine $\omega(u_i, v) + \omega(v, w_j)$ ukoliko su ispunjena sljedeća dva uvjeta:

1. put $\langle u, v, w \rangle$ je jedinstveni najkraći put od vrha u do vrha w. U suprotnom. egzistencija takvog puta P za kojeg vrijedi $\omega(P) \leq \omega(\langle u, v, w \rangle)$ čuva vrijednost najkraćeg puta te čini nepotrebnim dodatno ubacivanje brida prečaca.

2. brid $(u, w) \notin E$. U suprotnom, ako već postoji brid (u, w) te je ispunjen prvi uvjet, težina brida se postavlja na vrijednost najkraćeg puta, tj. $\omega(u, w) = \omega(u, v) + \omega(v, w)$.

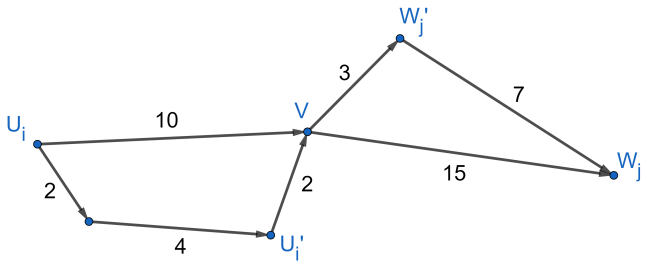
Uvjet 2 bi se mogao izostaviti uz potrebnu modifikaciju Dijkstrinog algoritma koji dopušta višestruke bridove između dva vrha. Uz pristup izmjene težine postojećeg brida (u, w) moguće je izgubiti originalnu vrijednost udaljenosti, no ona za potrebe najkraćeg puta nije bitna pošto postoji kraći put čijom smo vrijednosti i zamijenili težinu promatranog brida. Naravno, uvijek se mogu sačuvati dodatno informacije o polaznom grafu ukoliko ne želimo gubitak početnih vrijednosti.

Određivanje novih prečaca može se obaviti direktno provodeći Dijkstrin algoritam za svaki od ulaznih vrhova u_i, računajući najkraći put do svih izlaznih vrhova w_j te uspoređujući taj put s putom $\langle u_i, v, w_j \rangle$. Takvo pretraživanje može se prekinuti kada se dosegne vrijednost puta veća od $\omega(u_i, v) + \max\{\omega(v, w_j) \colon (v, w_j) \in E\}$, što direktno slijedi iz svojstva Dijkstrinog algoritma koji pronalazi vrhove redom od najbližih prema najudaljenijima.

Kako je pri svakoj kontrakciji te kod određivanja redoslijeda vrhova (4.1) potrebno izvršiti Dijkstrinu pretragu krenuvši iz svakog susjednog ulaznog vrha, cjelokupni proces predprocesiranja može biti izuzetno vremenski zahtjevan. Primijetimo da uvođenje ne nužno potrebnih prečaca ne kvari korektnost kontrakcije – ostaje osigurano svojstvo postojanja najkraćeg puta između svaka dva vrha nakon izbacivanja pojedinog vrha v. Stoga, procjenu treba li dodati prečac možemo malo pojednostavniti što će uštedjeti značajno na vremenu, riskirajući da možda u graf ubacimo neke nepotrebne bridove. Cilj je postići ravnotežu u omjeru vremena predprocesiranja i faze upita, dodati što manji broj prečaca kako krajnji graf G^* sa svim dodanim bridovima ne bi bio značajno gušći od početnog G, što bi utjecalo na brzinu pronalaska najkraćih putova u fazi upita.

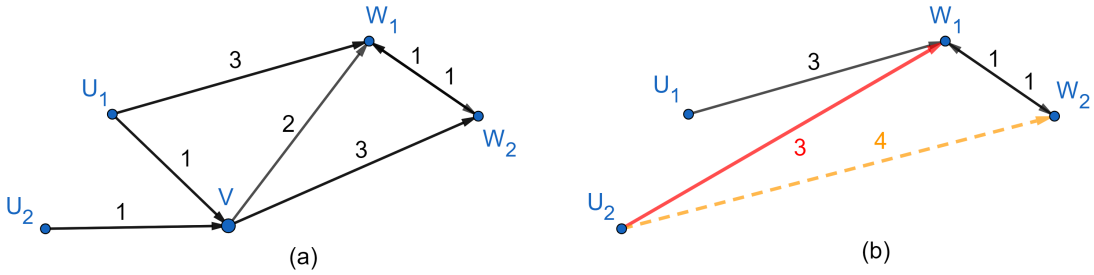
Kao prvo pojednostavljenje, moguće je provesti Dijkstrino pretraživanje iz svakog vrha u_i za koji vrijedi $(u_i, v) \in E$, ignorirajući vrh v i sve bridove koji ulaze ili izlaze iz njega. Slično kao i prije, pretragu prekinemo kada na *obradu* dođe vrh udaljenosti veće od $\omega(u_i, v) + \max\{\omega(v, w_j) \colon (v, w_j) \in E\}$. Prečac (u_i, w_j) dodajemo ako i samo ako vrijedi $D_i[w_j] > \omega(u_i, v) + \omega(v, w_j)$.

Ukoliko postoji najkraći put P od vrha u_i do vrha w_j koji ne prolazi kroz v, tada očito vrijedi $P \leq \omega(\langle u_i, v, w_j \rangle) = \omega(u_i, v) + \omega(v, w_j)$ te opisani Dijkstrin algoritam neće stati dok ne pronađe takav put. Na kraju vrijedi $D_i[w_j] \leq \omega(u_i, v) + \omega(v, w_j)$ te se ne dodaje nepotrebni prečac. Jedini slučaj kada može nastati problem prikazan je na slici 4.1, gdje postoji



Slika 4.1: Primjer najkraćeg puta od u_i do vrha w_j, koji prolazi vrhom v.

najkraći put P duljine 18 što je strogo manje od $\omega(u_i, v) + \omega(v, w_j) = 25$. Općenito, ako postoji najkraći put P koji u sebi sadrži v te je strogo manji od $\omega(\langle u_i, v, w_j \rangle)$, takav put neće biti pronađen te će biti dodan brid (u_i, w_j) težine $\omega(u_i, v) + \omega(v, w_j)$. Međutim, u takvom putu P postoje vrhovi u_i' i w_j' takvi da je $P = \langle u_i, \ldots, u_i', v, w_j', \ldots, w_j \rangle$ te će Dijkstrina pretraga iz vrha u_i' pravilno očuvati duljinu puta $\langle u_i', v, w_j' \rangle$. Konkretno, na grafu sa slike 4.1 bio bi dodan prečac $\langle u_i', v, w_j' \rangle$ duljine 5, te bi nakon cjelokupnog procesa kontrakcije i dalje bio očuvan najkraći put od vrha u_i do vrha w_j. Primijetimo još kako redoslijed izvođenja Dijkstrinih algoritama igra ulogu u dodavanju bridova. Na primjer, da je vrh u_i' bio prije vrha u_i, prečac (u_i, w_j) ne bi bio dodan. Sličan slučaj mogućeg dodavanja viška prečaca vidljiv je i u primjeru 4.1.1, na slici 4.2.



Slika 4.2: Primjer grafa prije (a) i poslije (b) kontrakcije vrha v. Crvenom bojom označen je prečac koji se ubacuje u skup bridova, dok je isprekidanom narančastom linijom označen brid koji se može a i ne mora ubaciti u izmijenjeni graf.

Primjer 4.1.1. *Slika 4.2 prikazuje jedan primjer kontrakcije vrha. Neka je zadan graf G kao sa slike 4.2 (a). Potrebno je izbaciti vrh v te njegove bridove $(u_1, v), (u_2, v), (v, w_1)$ i (v, w_2). Promotrimo prvo vrh u_1 iz kojeg postoji ulazni brid prema vrhu v. Iz primjera se lako vidi da je $\omega(u_1, w_1) = 3$ i $\omega(u_1, w_2) = 4$. Kako nakon izbacivanja vrha v i dalje*

postoje putovi $P_1 = \langle u_1, w_1 \rangle$ *i* $P_2 = \langle u_1, w_1, w_2 \rangle$*, duljina redom* 3 *i* 4*, nije potrebno dodavati prečace iz vrha* u_1*. Promotrimo li nadalje vrh* u_2*, u polaznom grafu vrijedi* $\omega(u_2, w_1) = 3$ *te se izbacivanjem vrha v gubi takav najkraći put (zapravo nisu ni povezani nakon izbacivanja). Stoga, potrebno je dodati brid* (u_2, w_1) *duljine* 3*. Kako je duljina* $\omega(u_2, w_2) = 4$*, te je* $\omega(\langle u_2, w_1, w_2 \rangle) = 4$*, nije potrebno dodati prečac* (u_2, w_2)*. Primijetimo da je taj brid potrebno dodati ukoliko prvo pretražujemo u grafu bez vrha v najkraći put od* u_2 *do vrha* w_2*, umjesto do* w_1*. Takav brid je označen isprekidanom narančastom bojom, te ga nije nužno dodati u skup bridova. Iz predložene realizacije kontrakcije vrha, pokreće se Dijkstrino pretraživanje iz vrha* u_2 *do svih preostalih, te se iz dobivenih informacija zaključuje potreba dodavanja prečaca. Takvom implementacijom narančasti brid će uvijek biti dodan u krajnji graf.*

Moguće je koristiti razne heuristike koje bi dodatno ubrzale procjenu nužnosti dodavanja prečaca. Niti jedna od njih ne narušava valjanost kontrakcije – uvijek se dodaju svi nužni bridovi. Kako se radi o uštedi vremena, pretraga nije potpuna te u nekim situacijama ne uspijeva pronaći zaobilazni put kojim bi se opovrgnula nužnost ubacivanja prečaca. U tim slučajevima dolazi do kreiranja prečaca koji su se mogli izbjeći. Jedan način na koji se može ubrzati pretraga je uvesti ograničenje na broj *obrađenih* vrhova u Dijkstrinom algoritmu. Pretraga prestaje kada se dostigne zadani broj, žrtvujući mogući nepronalazak najkraćeg puta od vrha u_i do vrha w_j. Moguće je ograničiti i broj bridova koji se pretraže na udaljenosti od početnog vrha u_i. Tim pristupom je svaki pronađeni put ograničen po broju vrhova sadržanih u sebi, zvanim *hop limit*.

Raspored vrhova

U prethodnom odjeljku objašnjen je proces kontrakcije jednog vrha iz zadanog grafa. U *Contraction Hierarchies* algoritmu potrebno je provesti kontrakciju svih vrhova, čime se nameće pitanje kojim redom je to najbolje obaviti. U odjeljku 4.3 dokazano je da proizvoljan poredak vrhova rezultira ispravnim algoritmom, no neće svaki poredak dati jednako brze rezultate u fazi upita. Razlike u broju dodanih prečaca kod raznih poredaka vrhova mogu biti poprilično velike, što rezultira u razlici vremena izvođenja pronalaska najkraćih putova u fazi upita. Stoga, potrebno je što prikladnije odrediti poredak kojim će se izvršavati kontrakcije vrhova. U nastavku navodimo neke od najznačajnijih svojstava koja služe pri određivanju poretka:

- **Razlika bridova** (eng. *edge difference*) – brojčana vrijednost dobivena razlikom broja prečaca koji bi bili dodani kontrakcijom promatranog vrha s brojem njemu incidentnih bridova.
- **Uniformnost** (eng. *uniformity*) – odabir novog vrha pri kontrakciji trebao bi biti uniformno distribuiran u grafu, tj. trebalo bi izbjegavati biranje vrhova uvijek iz iste ma-

nje okoline. Jednostavna heuristika koja se pokazuje kvalitetnom je *broj obrisanih susjeda* (eng. *deleted neighbors*). Za svaki vrh se pamti broj njemu obrisanih (kontrahiranih) susjednih vrhova, uzimajući u obzir i susjede povezane nekim od prethodno dodanih prečaca. *Voronoijev parametar* (eng. *Voronoi-Region parameter*) također je prikladna heuristika pri odabiru vrhova. Vrijednost parametra je kvadratni korijen veličine Voronoijeve regije pojedinog vrha. Za svaki vrh v, Voronoijeva regija $R(v)$ trenutnog grafa G' (nastao od početnog kontrakcijom nekih vrhova) definira se kao skup vrhova kojima je vrh v najbliži od preostalih vrhova u grafu G'. Ideja je prioritet pri izboru idućeg vrha dati čvorovima s manjom vrijednosti Voronoijevog parametra, s ciljem da se tako odabir vrhova uniformno distribuira po početnom grafu.

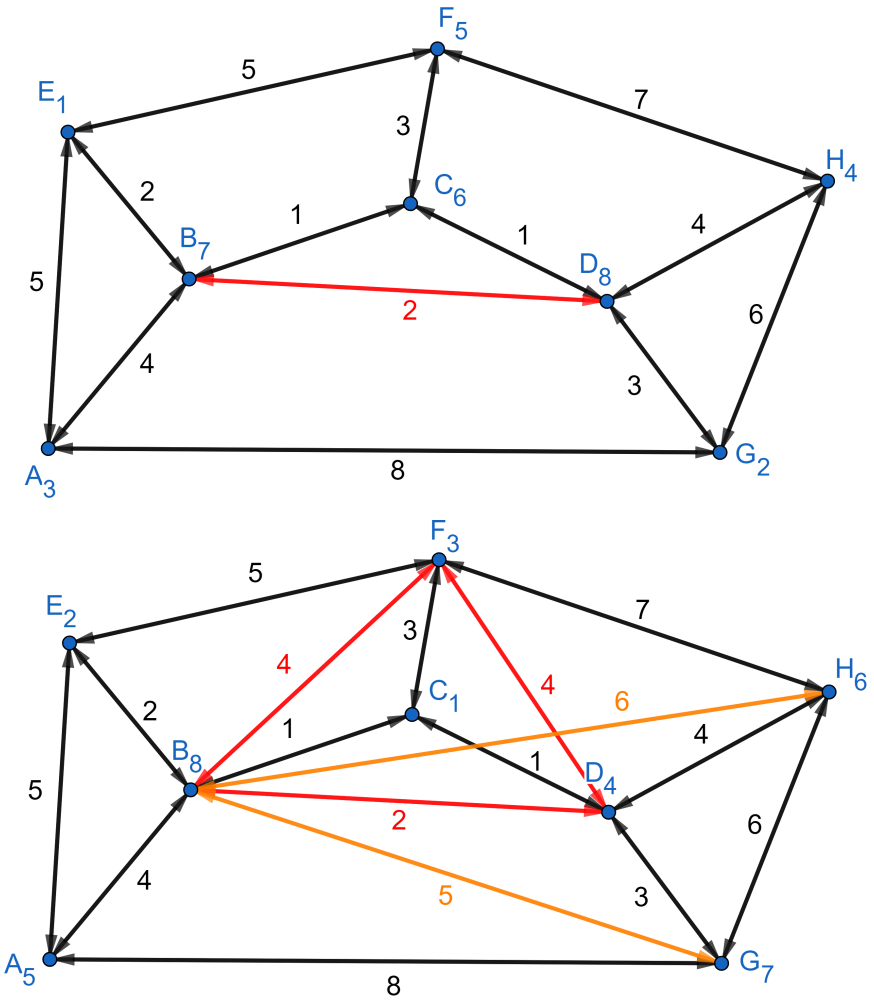
Postoje još razni parametri po kojima bi se mogao pridijeliti prioritet odabira vrha. Neki od njih su *cijene kontrakcije* (eng. *cost of contraction*) i *cijene upita* (eng. *cost of queries*). Detaljnije o pojedinim pristupima može se pronaći u [7].

Primjer 4.1.2. *Slika 4.3 prikazuje graf G i dodane bridove prečace u ovisnosti o poretku kontrakcije vrhova. Nad skupom vrhova $V = \{A, B, C, D, E, F, G, H\}$ označen je poredak vrhova indeksom u njihovom nazivu. Npr. E_1 znači da se prvo nad vrhom E provodi kontrakcija, C_6 – da je vrh C šesti po redu. U primjeru se mogao promatrati i neusmjeren graf, no ostavljamo bridove u oba smjera pošto općenito promatramo usmjerene grafove.*

Gornja slika prikazuje postupak u kojemu je prvi na redu za kontrakciju vrh E. Skup ulaznih i izlaznih bridova je $\{A, B, F\}$. Lako se vidi da se izbacivanjem vrha E ne gubi najkraći put između nijedna dva susjedna vrha. Na primjer, $\omega(\langle A, E, F\rangle) = 10 > 8 = \omega(\langle A, B, C, F\rangle)$ pa nije potrebno dodati brid (A, F). Laganom provjerom ustanovi se da nije potrebno dodavati prečace sve do trenutka kontrakcije vrha C, kada je put $\langle B, C, D\rangle$ jedinstveni najkraći put od vrha B do vrha D. Također, isti argument vrijedi i u suprotnom smjeru, od vrha D do vrha B. Iz navedenog razloga dodaju se bridovi (B, D) i (D, B).

Donja slika prikazuje drugačiji poredak kojim se kontrahiraju vrhovi. Crvenom bojom označeni su dodani prečaci nakon kontrakcije vrha C, koji je prvi na redu. Vidimo da je brid (B, D) nužno dodati pošto je $\langle B, C, D\rangle$ jedinstveni najkraći put od vrha B do vrha D. Analogno za ostale bridove. Daljnjom kontrakcijom vrhova E i F nije potrebno dodavati nove bridove. Narančastom bojom označeni su dodani bridovi u procesu kontrakcije vrha D. Na primjer, nužno je dodati brid (B, G) težine 5, jer je $\langle B, D, G\rangle$ jedinstven najkraći put u grafu koji prethodi kontrakciji vrha D – modificirani početni graf G nastao prethodnim kontrakcijama vrhova.

Svi vrhovi grafa čuvaju se u strukturi prioritetnog reda, gdje je prvi po redu vrh s najmanjom vrijednosti linearne kombinacije razlike vrhova i uniformnosti. U svakom koraku bira se vrh s najmanjom vrijednosti te se provodi proces kontrakcije nad njim. Zatim se takav vrh izbacuje iz prioritetnog reda, u kojemu se uvijek nalaze isključivo preostali vrhovi



Slika 4.3: Primjer utjecaja redoslijeda kontrakcija vrhova na broj dodanih bridova prečaca. Crnom bojom označeni su početni bridovi, dok su bridom u boji označeni dodani prečaci.

nad kojima još nije provedena kontrakcija. Redoslijed vrhova koji se uspostavlja u predprocesiranju definira se redoslijedom provođenja kontrakcije nad svim čvorovima grafa.

Primijetimo da se izbacivanjem pojedinog vrha vrijednosti razlike vrhova i uniformnosti kod preostalih čvorova mijenjaju. Stoga, moguće je nakon svake kontrakcije ponovno izračunati vrijednosti linearne kombinacije promatranih parametara, no to je izuzetno vremenski zahtijevno i narušava performanse algoritma. Kako bi se riješio navedeni problem ažuriranja prioritetnog reda, moguće je primijeniti neke od sljedećih pristupa:

– *Lijeno ažuriranje* (eng. *lazy update*) – prije kontrakcije vrha v najmanjeg u prioritetnom redu, ponovno se računa vrijednost njegove linearne kombinacije parametara. Ukoliko je novoizračunata vrijednost i dalje manja od vrijednosti drugog najmanjeg vrha u prioritetnom redu, provodi se kontrakcija vrha v. U suprotnom, vrh v se ponovno ubacuje u red sa svojim ažuriranim prioritetom te se proces nastavlja s novim vrhom v' najmanjeg prioriteta.

– *Ažuriranje susjeda* (eng. *neighbors recomputing*) – ažurira prioritete svih čvorova povezanih s vrhom nad kojim je proveden postupak kontrakcije.

– *Periodično ažuriranje* (eng. *periodically rebuilding*) – nakon određenog broja kontrakcija, provodi se ponovno računanje prioriteta svih vrhova u prioritetnom redu.

Pojedine metode obnavljanja prioriteta vrhova pri novom odabiru za kontrakciju ne daju nužno u svim slučajevima ispravan poredak. Međutim, kombinirajući neke od navedenih pristupa moguće je dobiti poprilično dobre rezultate uz veliku uštedu vremena predprocesiranja u odnosu na potpuno obnavljanje prioriteta svih vrhova nakon svake kontrakcije.

4.2 Faza *upita*

Za zadani graf G provođenjem predprocesiranja dobijemo graf $G^* = (V, E^*, \omega^*)$, opisan u uvodu ovog poglavlja. Graf G^* nastaje dodavanjem u graf G svih prečaca i njihovih težina određenih u fazama kontrakcije vrhova.

Definicija 4.2.1. *Za zadani graf $G = (V, E, \omega)$ i funkciju rank, $rank\colon E \to \mathbb{N}$, definiramo rastući podgraf $G_\uparrow = (V, E_\uparrow, \omega_\uparrow)$ gdje je $E_\uparrow = \{(u, v) \in E\colon rank(u) < rank(v)\}$, te $\omega_\uparrow$ restrikcija težinske funkcije ω na skup bridova $E_\uparrow$. Analogno, definiramo padajući podgraf $G_\downarrow = (V, E_\downarrow, \omega_\downarrow)$, uz $E_\downarrow = \{(u, v) \in E\colon rank(u) > rank(v)\}$ i $\omega_\downarrow$ restrikcija od ω nad skupom $E_\downarrow$.*

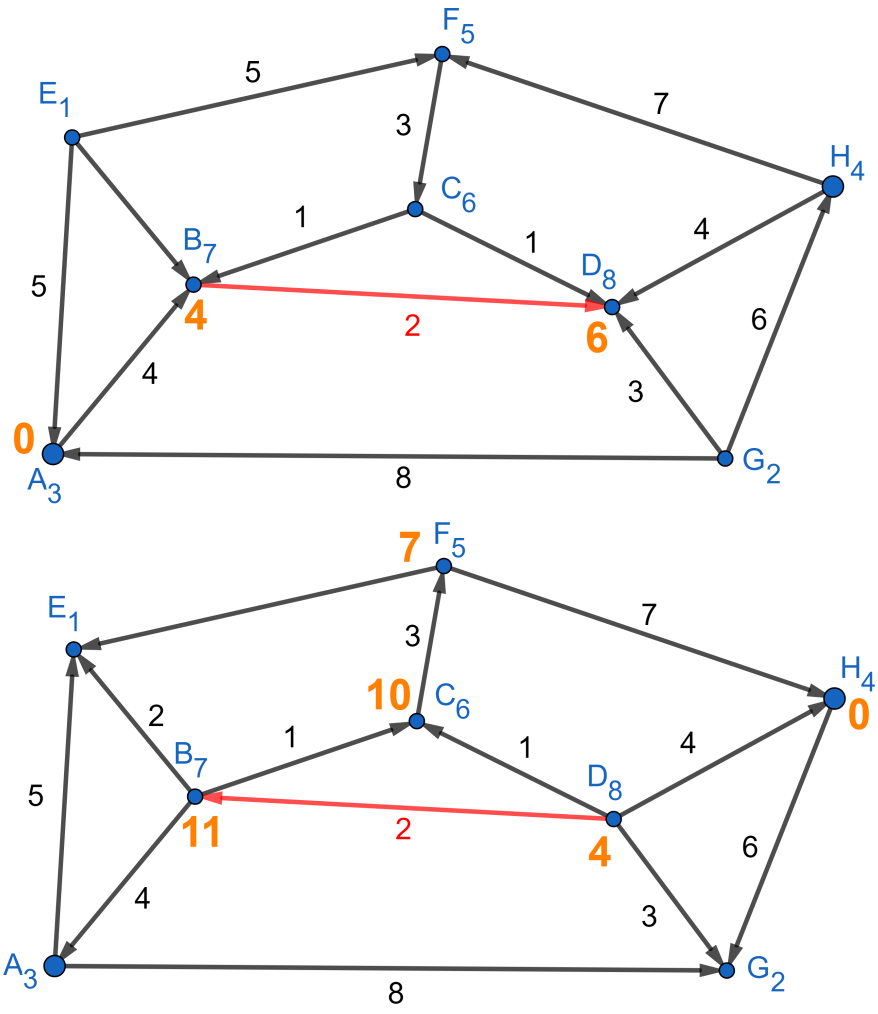
Neka su zadani graf $G = (V, E, \omega)$, početni i krajnji vrhovi, redom $s, t \in V$. Faza upita sastoji se od pronalaska najkraćeg puta između vrhova s i t, te se može ostvariti jednostavnim Dijkstrinim pretraživanjima najkraćih putova u grafovima $G^*_\uparrow$ i $G^*_\downarrow$. Potrebno je:

1. nad grafom $G^*_\uparrow$ provesti Dijkstrin algoritam s početnim vrhom s, čime dobijemo polje udaljenosti $D_s[v]$ – duljina pronađenog najkraćeg puta u grafu $G^*_\uparrow$ od s do v, za sve $v \in V$.

2. provesti *pretraživanje unazad*, opisano u odjeljku 3.3, nad grafom $G^*_\downarrow$, počevši u vrhu t. Kako bi se provelo *pretraživanje unazad*, unutar Dijkstrinog algoritma promatraju

se obrnute orijentacije bridova. Rezultat je polje D_t koje na mjestu indeksa vrha v, u varijablu $D_t[v]$, pohranjuje vrijednost najkraćeg puta od v do t unutar grafa $G^*_{\downarrow}$.

3. ako s I označimo skup svih *obrađenih* vrhova u obje Dijkstrine pretrage, tj. skup onih vrhova do kojih je pronađen najkraći put u oba grafa $G^*_{\uparrow}$ i $G^*_{\downarrow}$, redom od vrha s i do vrha t, tada je $d_G(s,t) = \min\{D_s[v] + D_t[v] : v \in I\}$.



Slika 4.4: Primjer pronalaska najkraćeg puta *Contraction Hierarchies* algoritmom – *faza upita.*

Primjer 4.2.2. *Na slici 4.4 nalazi se primjer izvrednjavanja opisanog algoritma kod faze upita. Zadani graf G identičan je grafu iz primjera 4.1.2. Crvenom bojom označen je jedini potrebni brid prečac koji se dodaje prilikom procesa kontrakcije vrhova, kao kod gornjeg dijela sa slike 4.3. Na gornjem dijelu slike 4.4 prikazan je $G^*_\uparrow$, dok je ispod njega prikazan padajući podgraf $G^*_\downarrow$. Faza upita se sastoji od sljedećeg: potrebno je pronaći najkraći put od vrha A do vrha H. Indeksom ispod imena vrha označen je njegov redoslijed pri kontrakciji, tj. vrijednost funkcije rank koja opisuje prioritet vrha. U rastućem podgrafu $G^*_\uparrow$ provodimo Dijkstrinu pretragu unaprijed, s početkom u vrhu A. Podebljanim narančastim brojevima označene su vrijednosti $D_s[v]$, za one vrhove v za koje je pronađen najkraći put. Točnije, ti vrhovi su obrađeni u Dijkstrinoj pretrazi te vrijedi $v \in I$. Na isti način, na donjem dijelu slike označeni su i vrhovi koji su obrađeni u pretrazi unazad, prilikom provođenja dijela 2 algoritma faze upita, kod pretrage grafa $G^*_\downarrow$ iz vrha H. Iz grafova vidimo kako je $I = \{B, D\}$, te je po 3 dijelu algoritma vrijednost $d_G(A, H)$ jednaka $\min\{D_s[B] + D_t[B], D_s[D] + D_t[D]\} = \min\{4 + 11, 6 + 4\} = 10$. Vrijednost minimuma se postigla za vrh D, te se iz dodatnih podataka o roditeljima/prethodnicima unutar Dijkstrine pretrage lako može odrediti najkraći put: $\langle A, B, D, H\rangle$. Vidimo da vrijednost duljine najkraćeg puta u grafu G uistinu odgovara vrijednosti 10, no ovdje je potrebno još „raspakirati" bridove prečace iz razloga što smo u određivanju puta gledali podgrafove grafa G^*, koji ima dodane prečace. Uzimajući u obzir koje smo prečace dodali prilikom kontrakcije određenih vrhova, dobijemo najkraći put u grafu G: $\langle A, B, C, D, H\rangle$.*

Primjer 4.2.2 prikazuje provođenje faze upita u *Contraction Hierarchies* algoritmu. Malo složeniji primjeri mogu se pronaći u [2] i [14]. U nastavku navodimo dva načina kako se može modificirati osnovni algoritam faze upita dan na početku ovog odjeljka, zadržavajući korektnost algoritma uz smanjeni prostor pretrage.

Kao što je opisano u dijelu 3.3, koraka 1 i 2 algoritma faze upita, pretraživanja *unaprijed* grafa $G^*_\uparrow$ i *unazad* grafa $G^*_\downarrow$ mogu se obavljati simultano. Dvije su opcije: u zajedničkom prioritetnom redu pamtiti sve vrhove i uzimati uvijek vrh s najmanjom vrijednosti – udaljenosti, ili imati dva odvojena prioritetna reda te uvijek uzimati najmanji iz oba kao idući vrh za *obradu*. Pronalazak najkraćeg puta odvija se isprepletući pretragu *unaprijed* i *unazad*. Kada na red dođe *obrada* vrha *v* koji je već *obrađen* u suprotnoj pretrazi – pronađen najkraći put, dobijemo novog kandidata za rješenje: $D_s[v] + D_t[v]$. Pretragu u jednom smjeru možemo obustaviti ukoliko je sljedeći na redu vrh koji ima pronađenu vrijednost najkraćeg puta veću od najbolje do tada pronađene vrijednosti. Kako je redoslijed *obrade* vrhova od najbližih prema najudaljenijima, ovakva modifikacija ne utječe na korektnost rješenja.

Faza upita pronalazi ispravan najkraći put u grafu G između zadanih vrhova s i t, kao što će biti dokazano u odjeljku 4.3. Međutim, kako se odvija pretraga u grafovima $G^*_\uparrow$ i $G^*_\downarrow$, vrijednosti $D_s[v]$ i $D_t[v]$ neće uvijek nužno biti vrijednosti najkraćih putova između vrha v i vrhova s, t redom, gledano u grafu G^*. Naravno, razlog je računanje najkraćih putova Dijkstrinim algoritmom redom u grafovima $G^*_\uparrow$ i $G^*_\downarrow$, a ne u cijelom grafu G^*. Stoga, ukoliko

možemo lako primijetiti da neka od vrijednosti $D_s[v]$ ili $D_t[v]$ nisu vrijednosti najkraćih putova od/do odgovarajućih vrhova, bez narušavanja korektnosti algoritma možemo prekinuti daljnju pretragu iz takvog vrha v. Jedna takva metoda naziva se *stall-on-demand* tehnika – prilikom *obrade* vrha v kod *pretraživanja unaprijed* u grafu $G^*_\uparrow$, provjerava se može li se ustanoviti kako $D_s[v]$ nije najkraći put u grafu G od vrha s, tj. $d(s, v) \neq D_s[v]$. Za takvu provjeru koriste se vrhovi x grafa $G^*_\downarrow$, takvi da postoji brid (x, v) te $rank(x) > rank(v)$. Ukoliko je $D_s[x] + \omega(x, v) < D_s[v]$ tada sigurno postoji put u grafu G^* od s do v koji nije pronađen pretragom *unaprijed* u podgrafu $G^*_\uparrow$. U tom slučaju nije potrebno obaviti *relaksaciju bridova* (vidjeti 3.1) te se pretraga može „podrezati" u grani koja bi slijedila iz vrha v. Pritom se ne može ugroziti pronalazak najkraćeg puta iz razloga što niti jedan put preko vrha v ne bi bio optimalan jer nije najkraći put u grafu G^*, pa pritom i u G (slijedi iz leme 4.3.1). *Stall-on-demand* se analogno primjenjuje na pretragu *unazad* kod grafa $G^*_\downarrow$. Naravno, „podrezivanje" se neće uvijek primijetiti, te je stoga moguće provesti intenzivniju pretragu krenuvši na susjede susjeda i tako dalje. Više detalja moguće je pronaći u [7]. Kako povećana pretraga iziskuje više vremena u fazi upita, potrebno je pronaći pravi balans kojim bi se smanjio prostor pretrage i ne previše povećalo vrijeme izvršavanja. Opisana *stall-on-demand* metoda u primjeni pokazuje zapažena ubrzanja (vidjeti [7]).

Rekonstrukcija najkraćeg puta: primijetimo da algoritam pronalaska najkraćeg puta može kao rješenje dati put koji u sebi ima neke od dodanih bridova prečaca, kao što je to slučaj i u primjeru 4.2.2. Ukoliko je cilj odrediti i najkraći put između dva zadana vrha, a ne samo njegovu vrijednost, potrebno je za svaki dodani prečac (u, w) pamtiti vrh v čijom je kontrakcijom dodan u G^*. Ključno je primijetiti kako svaki prečac „zaobilazi" točno jedan vrh, tj. dodan je kontrakcijom jednog vrha. Rekurzivnim pristupom lako se rekonstruira najkraći put: iz najkraćeg puta P dobivenog algoritmom faze upita, izbacujemo sve dok postoji prečac (u, w) te ga zamjenjujemo dijelom puta $\langle u, v, w\rangle$, gdje je prečac (u, w) dodan kontrakcijom vrha v.

4.3 Dokaz korektnosti CH algoritma

Neka je funkcija *rank*, $rank\colon E \to \mathbb{N}$, dobivena postupkom predprocesiranja opisanog u pododjeljku 4.1. Kako je dobivena funkcija injekcija, implicitno imamo uređaj vrhova $V = \{u_1, u_2, \ldots, u_n\}$, gdje vrijedi $rank(u_i) < rank(u_j)$ za sve $1 \leq i < j \leq n$. U fazi predprocesiranja takav poredak je bio bitan kako bi dao što kraće vrijeme izvršavanja faze upita *Contraction Hierarchies* algoritma. Što se tiče ispravnosti algoritma, dokazat ćemo da proizvoljni poredak daje točne rezultate, najkraće putove u fazi upita.

Fiksirajmo neki proizvoljan poredak vrhova $\{u_1, u_2, \ldots, u_n\}$. Nadalje, neka je $G_0 = G$ početni graf. Graf G_i je graf koji nastaje iz grafa G_{i-1} postupkom kontrakcije vrha u_i. Točnije, izbacivanjem vrha u_i, njegovih incidentnih bridova te dodavanjem svih potreb-

nih bridova prečaca iz faze kontrakcije. Time dobijemo niz grafova $G_0, G_1, \ldots, G_n$, gdje graf G_i ima $|V_i| = n - i$ vrhova, za svaki $i \in \{1, 2, \ldots, n\}$. Na kraju, graf G^* definira se preko početnog grafa G proširenjem skupa bridova, dodavanjem svih prečaca iz grafova $G_1, G_2, \ldots, G_{n-1}$.

Lema 4.3.1. *Kontrakcijom vrhova očuvaju se najkraći putovi. Točnije, za svaki $i = 1, \ldots, n$ i za sve $s, t \in V_i$ vrijedi*

$$d_{G_i}(s, t) = d_{G_{i-1}}(s, t). \tag{4.1}$$

Dokaz. Neka je $i \in \{1, 2, \ldots, n\}$ proizvoljan, te neka su takvi i vrhovi $s, t \in V_i$. Označimo s P neki najkraći put od vrha s do vrha t u grafu G_i. Ukoliko put P ne sadrži bridove prečace dodane kontrakcijom vrha u_i, put P se također nalazi i u grafu G_{i-1}. U suprotnom, takav put sadržava neki prečac (u, w) dodan postupkom kontrakcije vrha u_i iz grafa G_{i-1}. To znači da graf G_{i-1} sadrži put $\langle u, u_i, w\rangle$ iste težine kao prečac (u, w). Zamijenimo li brid (u, w) s dijelom puta $\langle u, u_i, w\rangle$, dobijemo put P' u grafu G_{i-1} jednake težine kao početni put P. Ovime smo dokazali da za najkraći put P iz grafa G_i postoji put jednake težine u grafu G_{i-1}. Stoga, vrijedi nejednakost $d_{G_{i-1}}(s, t) \leq d_{G_i}(s, t)$.

Kako bi dokazali suprotnu nejednakost, promotrimo najkraće putove u grafu G_{i-1}. Ukoliko postoji najkraći put P koji u sebi ne sadrži vrh u_i, tada takav put postoji i nakon kontrakcije tog vrha, tj. postoji put u grafu G_i duljine $\omega(P)$. Ukoliko to nije slučaj, svaki najkraći put u grafu G_{i-1} prolazi vrhom u_i. Kako najkraćih putova ima konačno mnogo, označimo s $P = \langle s, \ldots, u, u_i, w, \ldots, t\rangle$ najkraći put koji prolazi kroz najveći broj vrhova grafa G_{i-1}. Tada je put $\langle u, u_i, w\rangle$ najkraći put od vrha u do vrha w, te je svaki drugi put od u do w koji prolazi kroz u_i strogo veći od njega. U suprotnom bi imali put $\langle u, \ldots, u_i, \ldots, w\rangle \neq \langle u, u_i, w\rangle$ koji bi mogli zamijeniti s dijelom puta $\langle u, u_i, w\rangle$ iz P te tako dobiti najkraći put u grafu G_{i-1} koji ima strogo više vrhova nego put P. Iz činjenice da ne postoji najkraći put od vrha s do vrha t koji ne prolazi vrhom u_i, zaključujemo da je $\langle u, u_i, w\rangle$ jedinstveni najkraći put od u do w. Stoga, kontrakcijom vrha u_i, u skup bridova grafa G_i bit će dodan prečac (u, w) težine $\omega(\langle u, u_i, w\rangle)$. Time smo dobili put $P' = \langle s, \ldots u, w, \ldots, t\rangle$ u grafu G_i jednake težine kao put P, tj. vrijedi $d_{G_i}(s, t) \leq d_{G_{i-1}}(s, t)$. □

Korolar 4.3.2. *Za svaki $i \in \{1, \ldots, n\}$ te za sve $s, t \in V_i$ vrijedi $d_{G_i}(s, t) = d_G(s, t)$.*

Dokaz. Primijetimo kako vrijedi $V_n \subset V_{n-1} \subset \cdots \subset V_0 = V$, pa za proizvoljne vrhove $s, t \in V_i$ vrijedi $s, t \in V_j$, za sve i, j takve da je $j < i$. Tvrdnja korolara slijedi primjenom matematičke indukcije koristeći lemu 4.3.1. □

Lema 4.3.3. *Uz oznake kao u odjeljku 4.2, faza upita Contraction Hierarchies algoritma računa ispravnu vrijednost najkraćeg puta u zadanom grafu G:*

$$d_G(s, t) = \min\{D_s[v] + D_t[v] : v \in I\}, \tag{4.2}$$

gdje su $s, t \in V$ te je s I označen skup obrađenih vrhova u obje pretrage, pretrazi unaprijed grafa $G^*_\uparrow$ te pretrazi unazad grafa $G^*_\downarrow$.

Dokaz. Dokaz leme u jednostavnijem slučaju, uz pretpostavku da je svaki najkraći put u grafu jedinstven, može se pronaći u [7] i [6]. Pretpostavka olakšava argumentaciju kod kontrakcije vrhova i nužnosti dodavanja brida prečaca. U nastavku slijedi dokaz u potpunoj općenitosti, bez navedene pretpostavke.

Za vrh v koji se nalazi na nekom putu P reći ćemo da je *lokalno mali* ako nije krajnji vrh u putu te postoji vrh prethodnik w i vrh sljedbenik u takvi da vrijedi $rank(w) > rank(v) < rank(u)$. Za vrhove w, u vrijedi da je $\langle w, v, u\rangle$ dio puta P.

Dokažimo za početak sljedeću tvrdnju: postoji najkraći put P od vrha s do vrha t u grafu G^* koji ne sadrži niti jedan *lokalno mali* vrh. Pretpostavimo suprotno, za svaki najkraći put od s do t u grafu G^* postoji *lokalno mali* vrh v kroz koji prolazi navedeni put. Stoga, za svaki najkraći put P od s do vrha t u grafu G^* dobro je definirano preslikavanje f, $f(P) = \min\{rank(v)\colon v$ lokalno mali vrh na putu P$\}$. Ideja je uzeti najkraći put P' s maksimalnom vrijednosti preslikavanja f, te iz procesa kontrakcije najmanjeg *lokalno malog* vrha u tom putu dobiti kontradikciju. Definiramo $P' = arg\max\{f(P)\colon P$ najkraći put u grafu G^* od vrha s do vrha $t\}$. Iz definicije slijedi postojanje *lokalno malog* vrha v na putu P' takvog da vrijedi $f(P') = rank(v)$ i $rank(v) = \max\{f(P)\colon P$ najkraći put u grafu G^* od vrha s do vrha $t\}$. Također, postoji dio puta $\langle w, v, u\rangle$ na putu P' za koji vrijedi $rank(w) > rank(v) < rank(u)$. Označimo s G' graf u procesu kontrakcije svih vrhova, neposredno prije kontrakcije vrha v. Kako vrhovi w, u imaju veću *rank* vrijednost, oni su u skupu vrhova grafa G'. Prilikom kontrakcije vrha v iz G' dodaje se brid (w, u) ili postoji neki drugi najkraći put od w do u koji koristi samo vrhove čije su *rank* vrijednosti strogo veće od $rank(v)$. Ovdje koristimo svojstvo injektivnosti funkcije *rank*, koja svakom vrhu pridruži jedinstveni prirodni broj. Kako se prilikom kontrakcije očuvaju vrijednosti najkraćih putova, iz korolara 4.3.2 zaključujemo da nakon izbacivanja vrha v iz G' postoji najkraći put od vrha w do vrha u duljine $\omega(\langle w, v, u\rangle)$. Kako je graf G^* nastao dodavanjem svih bridova prečaca, zaključujemo da u G^* postoji najkraći put između w i u koji ne sadrži niti jedan vrh prioriteta manjeg ili jednakog $rank(v)$. U slučaju da postoji direktan brid (w, u) tvrdnja i dalje vrijedi. Zamijenimo li dio puta $\langle w, v, u\rangle$ iz puta P' opisanim najkraćim putem od w do u koji ima vrhove prioriteta barem $rank(v) + 1$, dobivamo put P'' za koji vrijedi $f(P'') > f(P')$. Naime, put P'' je i dalje najkraći put od s do t u grafu G^*, te smo ga dobili zamjenom dijela puta $\langle w, v, u\rangle$ s putom od w do u koji ne sadrži v. Dodali smo eventualne vrhove prioriteta barem $rank(v) + 1$ iz čega slijedi da je najmanji *lokalno mali* vrh na putu P'' prioriteta strogo većeg od $rank(v)$. Dobili smo $f(P'') > f(P')$ što je direktno kontradikcija s definicijom puta P'. Uzimanje maksimuma iz definicije puta P' je dobro definirano pošto postoji samo konačno mnogo najkraćih putova u grafu, za zadani početni i krajnji vrh puta.

Ovime smo dokazali postojanje puta $P = \langle s = v_0, v_1 \dots v_{k-1}, v_k = t \rangle$ u grafu G^*, za koji ne postoji indeks $i \in \{1, 2, \dots, k-1\}$ takav da je $rank(v_{i-1}) > rank(v_i) < rank(v_{i+1})$. Označimo s j, ako postoji, prvi indeks za koji vrijedi $rank(v_0) < rank(v_1) < \cdots < rank(v_j) > rank(v_{j+1})$, $j \in \{0, 1, \dots, n-1\}$. Vrh v_{j+1} je ili krajnji, ako je $j + 1 = n$, ili mora vrijediti $rank(v_{j+1}) > rank(v_{j+2})$. To vrijedi iz činjenice da u putu P ne postoji *lokalno mali vrh*. Slično, zaključujemo da vrijedi $rank(v_0) < \cdots < rank(v_j) > rank(v_{j+1}) > \cdots > rank(v_n)$. Ukoliko ne postoji takav j, onda su u putu P vrhovi poredani u strogo rastućem poretku s obzirom na vrijednost funkcije *rank*. Radi jednostavnosti možemo uzeti $j = n$. U oba slučaja *pretraga unaprijed* u grafu $G^*_\uparrow$ *obradi* vrh v_j. Također vrijedi i za *pretragu unazad* kod grafa $G^*_\downarrow$. Zaključujemo da vrijedi:

$$\min\{D_s[v] + D_t[v] \colon v \in I\} = D_s[v_j] + D_t[v_j] = d_{G^*}(s, t), \tag{4.3}$$

gdje je vrh v_j dobiven na opisani način iz vrhova na putu P. Jednakost 4.3 vrijedi iz činjenice da je vrh v_j obrađen u obje pretrage algoritma faze upita, te je pronađen put $\langle s = v_0, \dots v_j, \dots, v_n = t \rangle = P$ koji je najkraći put od s do vrha t u grafu G^*. Vrijedi: $D_s[v_j] = d_{G^*}(s, v_j)$ i $D_t[v_j] = d_{G^*}(v_j, t)$. Konačno, iz leme 4.3.1 i načina dodavanja bridova vrijedi jednakost $d_G(s, t) = d_{G^*}(s, t)$, tj. dodavanjem bridova prečaca nikada ne smanjimo vrijednost najkraćeg puta, već on ostaje očuvan. Iz 4.3 zaključujemo

$$\min\{D_s[v] + D_t[v] \colon v \in I\} = d_{G^*}(s, t) = d_G(s, t) \tag{4.4}$$

što je i trebalo dokazati. □

Poglavlje 5

Landmark A^* – ALT

U prethodnom poglavlju predstavljen je *Contraction Hierarchies* algoritam koji koristi proces predprocesiranja kako bi odredio hijerarhijski odnos vrhova grafa te pomoću tih podataka ubrzao pronalazak najkraćeg puta – fazu upita. U ovom poglavlju opisujemo malo drugačiju svrhu predprocesiranja, čiji se algoritmi nazivaju *ciljno usmjereni* algoritmi (eng. *goal directed*). Takvi algoritmi usmjeruju pretragu prema ciljnom vrhu, smanjujući ukupan broj vrhova koje je potrebno pretražiti kako bi se pronašao najkraći put.

Algoritam A^* pretraživanja spada u ciljno usmjerena pretraživanja. Međutim, on ne koristi predprocesiranje, čijim bi dodavanjem mogao imati bitne informacije uz pomoć kojih bi se dobilo ubrzanje pronalaska najkraćeg puta. Jedan takav algoritam je *Landmark* A^*, kraće zvani ALT algoritam (eng. A^* *Landmarks Triangle Inequality Algorithm*), predstavljen u radu [8]. Glavni cilj ALT algoritma je korištenjem A^* pretraživanja, opisanog u odjeljku 3.2, dobiti što bolje rezultate uz pomoć što preciznije heurističke funkcije. Naime, iz teorema 3.2.8 slijedi da preciznija *izvediva* funkcija (vidjeti definiciju 3.2.4) procjene udaljenosti do ciljnog vrha *obradi* podskup vrhova u odnosu na neku u usporedbi s njom neprecizniju heuristiku. Ukoliko se radi o prometnoj mreži, u odjeljku 3.2 pokazali smo kako je euklidska udaljenost jedan ispravan primjer izvedive funkcije. ALT algoritam koristeći skup vrhova zvanih *orijentiri* i nejednakosti trokuta daje novi pristup određivanju procjene udaljenosti do ciljnog vrha.

Definicija 5.0.1. *Vrh $v \in V$ nazvat ćemo orijentirom ukoliko su poznate vrijednosti najkraćih putova do i od svih preostalih vrhova u grafu G.*

Za vrh v koji ima ulogu *orijentira*, u procesu predprocesiranja izračunaju se potrebni najkraći putovi $d_G(v, u)$ i $d_G(u, v)$, za svaki vrh $u \in V$. Klasična nejednakost trokuta kaže da je duljina svake stranice trokuta kraća od zbroja duljina preostalih dviju stranica. Pristup je motiviran nejednakošću trokuta, te se temelji na činjenici da je duljina najkraćeg puta od vrha a do vrha b uvijek manja ili jednaka duljini najkraćeg puta od a do vrha c, zbrojenog s

duljinom najkraćeg puta od c do ciljnog vrha b. Dokaz je očit, put sastavljen nadogradnjom najkraćeg puta od vrha a do c s najkraćim putom od c do b, jedan je od mogućih putova od vrha a do ciljnog b te kao takav sigurno nije kraći od najkraćeg puta od a do b. Formalno, za sve vrhove $a, b, c \in V$ vrijedi:

$$d_G(a,b) \leq d_G(a,c) + d_G(c,b). \tag{5.1}$$

Potrebno je procijeniti duljinu najkraćeg puta za zadane vrhove $u, t \in V$, pri čemu će se koristiti vrhovi *orijentiri*. Promotrimo sliku 5.1. Poznate su vrijednosti $d_G(l,u), d_G(l,t), d_G(u,l)$ i $d_G(t,l)$ dobivene predprocesiranjem. Koristeći 5.1 vrijedi:

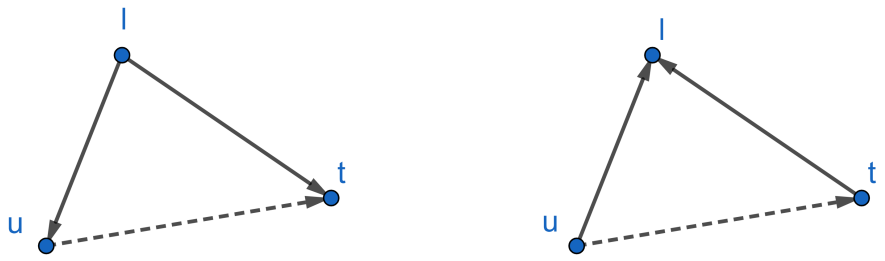
$$d_G(l,t) \leq d_G(l,u) + d_G(u,t) \quad \Longrightarrow \quad d_G(u,t) \geq d_G(l,t) - d_G(l,u) \tag{5.2}$$
$$d_G(u,l) \leq d_G(u,t) + d_G(t,l) \quad \Longrightarrow \quad d_G(u,t) \geq d_G(u,l) - d_G(t,l). \tag{5.3}$$

Stoga, ukoliko imamo izračunate vrijednosti za vrh *orijentir* l, procjenu najkraćeg puta do ciljnog vrha t možemo definirati izrazom:

$$\pi_{t,l}(u) := \max(d_G(l,t) - d_G(l,u),\ d_G(u,l) - d_G(t,l)) \leq d_G(u,t), \tag{5.4}$$

za svaki vrh $u \in V$. Kod neusmjerenih grafova bi izraz bio jednostavniji:

$$\pi_{t,l}(u) := |\, d_G(l,u) - d_G(l,t)\,|. \tag{5.5}$$



Slika 5.1: Načini procjene najkraćeg puta koristeći udaljenosti *od* i *do* vrha *orijentira* l.

Postavlja se pitanje koliko je funkcija $\pi_{t,l}$ procjene udaljenosti do ciljnog vrha t uistinu dobra aproksimacija funkcije $d_G(\,\cdot\,, t)$. Za proizvoljan $u \in V$, vrijednost $\pi_{t,l}(u)$ je „blizu" vrijednosti $d_G(u,t)$ ukoliko je neka od nejednakosti (5.2), (5.3) „blizu" jednakosti. Nejednakost (5.2) je „blizu" jednakosti ukoliko se vrh u nalazi „blizu" najkraćeg puta od l do vrha t. Tada možemo zamišljati kao da se *orijentir* t nalazi iza vrha u. Slično, nejednakost (5.3) je „blizu" jednakosti ako se t nalazi skoro na najkraćem putu od vrha u do vrha

l. Promatrajući sliku 5.1, možemo zamišljati da bi vrhovi u, t i l trebali biti skoro pa kolinearni, uz to da se *orijentir* l ne nalazi između vrhova u i t. Naravno, ovo je samo intuicija kojom zaključujemo da jedan *orijentir* ne može dati dobru donju ogradu na duljinu najkraćeg puta, za sve vrhove u i t. Iz tog razloga se na prikladan način odabire skup vrhova L, koji predstavlja vrhove *orijentire* uz pomoć kojih bi se trebala dobiti preciznija procjena $\pi_{t,L}$, za sve ciljne vrhove $t \in V$.

5.1 Odabir vrhova *orijentacije*

Prikladan skup vrhova *orijentacije* ključan je faktor koji utječe na brzinu pronalaska najkraćeg puta u fazi upita. Kod *Landmark A** algoritma, odabir skupa *orijentacije* jedini je korak kod kojeg dodatno znanje o domeni problema može donijeti značajno poboljšanje. Cijeli postupak određivanja takvih vrhova obavlja se u fazi predprocesiranja, u nadi da će tako dobiveni skup dati što veće ubrzanje pronalaska najkraćeg puta između zadanih vrhova. Neka je k broj vrhova koje želimo promatrati kao *orijentire*, tj. za njih je u fazi predprocesiranja potrebno izračunati najkraće putove *od* i *do* preostalih vrhova u grafu. U nastavku opisujemo tri načina odabira vrhova *orijentacije*:

- *slučajni odabir* vrhova
- *pohlepni najudaljeniji odabir* vrhova
- *planarni odabir* vrhova,

te način optimiziranja svakog od njih modifikacijama trenutno odabranog skupa *orijentira* kako bi se dobila bolja donja ograda na udaljenosti vrhova grafa.

Najjednostavniji način odabira skupa L svih vrhova *orijentacije* je *slučajni odabir* (eng. *random landmark selection*). Kao što mu i ime govori, na slučajan način bira se k različitih vrhova koji predstavljaju skup L.

Pohlepni najudaljeniji odabir (eng. *greedy farthest selection*) na brz način, pohlepnim pristupom pokušava odrediti što bolju aproksimaciju skupa L s k elemenata za koji je minimalna udaljenost između svih parova vrhova iz L maksimalna. Na početku se na slučajan način odabire jedan vrh iz skupa vrhova grafa G te se od njega izračuna najudaljeniji vrh v_1 koji se zatim ubaci u skup L. Zatim slijedi $k - 1$ iteracija u kojima se odvija sljedeće: odabire se i dodaje u skup L vrh u koji maksimizira vrijednost $\min_{l \in L'} d_G(l, u)$, gdje je L' skup prethodno odabranih vrhova *orijentacije*. Time se u svakoj iteraciji pronalazi vrh najudaljeniji od trenutnog skupa odabranih vrhova, te se takav dodaje u traženi skup L.

Ukoliko je zadan graf koji ima dobru geometrijsku interpretaciju, npr. kod prometne mreže su udaljenosti ili vrijeme najkraćeg puta često korelirane s geometrijskom udaljenosti između vrhova, pokazuje se dobrim kod takvih slučajeva odabrati ekvidistantan skup

vrhova *orijentacije*. Motivacija je ista kao u uvodu ovog poglavlja, intuitivno je jasno da bi procjena udaljenosti moga biti točnija ako vrh *orijentir* leži na pravcu zadanih vrhova, prije početnog ili iza ciljnog vrha. Takav pristup naziva se *planarni odabir* (eng. *simple planar landmark selection*). Napomenimo da graf ne mora nužno biti planaran da bi ovakav odabir dao dobre rezultate, kao što prometna mreža najčešće ni nije. Ideja određivanja *orijentacijskog* skupa je sljedeća: odabere se centralni vrh c, koji na neki način predstavlja centar mreže. Zatim se skup vrhova podijeli u k dijelova, s približno jednakim brojem vrhova. Možemo zamišljati kao da je vrh c centar kružnice, a svaki dio jedan kružni isječak s centrom u vrhu c. Zatim se iz svakog od k dijelova odabire po jedan vrh *orijentacije*. Takav vrh je najudaljeniji od centra c u usporedbi sa svim ostalim vrhovima u toj cjelini. Kako bi se izbjegao odabir bliskih vrhova iz susjednih cjelina, ukoliko je iz jedne cjeline odabran vrh blizu njihove granice, iz susjedne cjeline se pri odabiru zanemaruju relativno bliski vrhovi zajedničkoj granici.

Opisani načini odabira *orijentacijskih* vrhova mogu se dodatno optimizirati na razne načine. Glavna ideja je uz iterativno dodavanje novih vrhova u skup L, povremeno izbaciti neki od vrhova *orijentira* te ga pokušati zamijeniti prikladnijim kandidatom. Jedan pristup k tome je imati skup kandidata, koji se može odabrati na slučajan način odabirom međusobno najudaljenijih vrhova ili nadskupom trenutno odabranih vrhova *orijentacije*. Uz to, na slučajan način odabire se i uzorak parova vrhova koji predstavljaju upite – pronalazak najkraćeg puta od prvog do drugog vrha. Za svaki vrh l iz skupa kandidata računa se njegova vrijednost na način da se za svaki par vrhova (u, v) iz uzorka odredi procjena udaljenosti $\pi_{v,l}(u)$. Zatim, iz trenutno određenog skupa *orijentira* L' izračuna se procjena udaljenosti $\pi_{v,L'}(u) = \max_{h \in L'} \pi_{v,h}(u)$. Iznos $\pi_{v,l}(u) - \pi_{v,L'}(u)$, ukoliko je pozitivan, pridodaje se vrijednosti kandidata l. Na taj način dobijemo informaciju koliko pojedini kandidat poboljšava procjenu udaljenosti u odnosu na trenutni skup *orijentira* L'. Prilikom izbacivanja nekog *orijentira* iz skupa L', dodaje se vrh kandidat s najvećom opisanom vrijednosti. Drugi način optimizacije provodi se npr. kod *pohlepnog najudaljenijeg odabira*, gdje se također periodično izbacuje vrh te se zamjenjuje s novim *orijentirom* koji je najudaljeniji od trenutnog odabranog skupa.

Navedene optimizacije mogu pridonijeti boljim procjenama udaljenosti, no iziskuju i znatno veća vremena kod predprocesiranja. Optimizirani *pohlepni najudaljeniji odabir* ne daje značajno bolje donje ograde, stoga je češće bolje koristiti jednostavniju verziju. Optimizirani *slučajni odabir* ima bolje rezultate od jednostavnog *slučajnog odabira*, no u usporedbi s ostalim pristupima daje najlošije rezultate kod prometnih mreža. Optimizirani *planarni odabir* se pokazuje kao dobar izbor ukoliko postoji geometrijska interpretacija zadanog grafa, kao npr. poznate koordinate vrhova u prostoru. Međutim, velika joj je mana izuzetno dugotrajno predprocesiranje. Za više detalja pogledati [8].

5.2 Faza *upita*

Rezultat faze predprocesiranja je skup vrhova *orijentacije* L te sve udaljenosti $d_G(l, u)$ i $d_G(u, l)$, za sve $u \in V$, $l \in L$. Faza *upita* sastoji se od računanja najkraćeg puta između zadana dva vrha: početnog s i krajnjeg vrha t. U fazi upita računa se heuristička funkcija procjene udaljenosti svih vrhova do ciljnog vrha. Takva funkcija se dobije koristeći informacije skupa *orijentira* i modifikacije nejednakosti trokuta, opisane u uvodnom dijelu ovog poglavlja. Heuristička funkcija za jedan fiksni *orijentacijski* vrh l definirana je jednakošću (5.4). Za skup vrhova L definirajmo $\pi_{t,L}$, ili kraće π_t, uzimajući najbolju donju ogradu po svim vrhovima $l \in L$. Za $u \in V$ neka je:

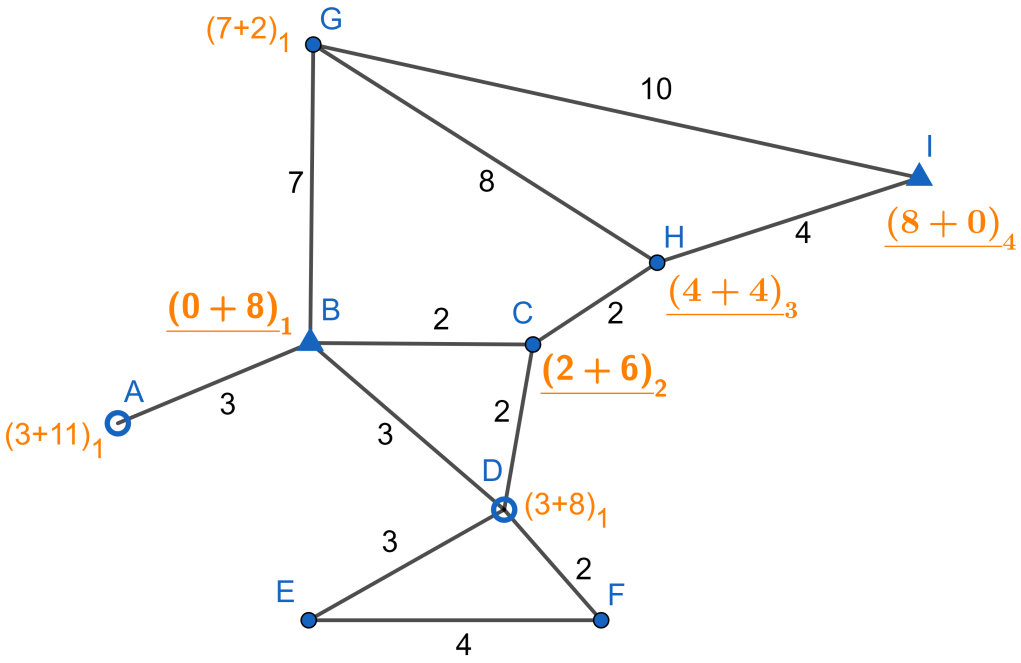
$$\pi_t(u) := \max_{l \in L} \pi_{t,l}(u) = \max_{l \in L} \{ \max(d_G(l, t) - d_G(l, u), d_G(u, l) - d_G(t, l)) \}. \tag{5.6}$$

Kod neusmjerenih grafova funkcija π_t poprima jednostavniji oblik:

$$\pi_t(u) = \max_{l \in L} |d_G(l, u) - d_G(l, t)|. \tag{5.7}$$

Faza upita dalje se odvija u potpunosti jednako kao kod standardnog A^* algoritma, obrađenog u odjeljku 3.2. Pretraga može biti klasična jednosmjerna, ili dvosmjerna A^* pretraga opisana u odjeljku 3.3. Kako su grafovi iz prometne mreže s jako velikim brojem vrhova najčešće *rijetki*, često se koristi popis susjeda kao reprezentacija grafa na računalu (vidjeti 1.2). Ukoliko se radi o usmjerenom grafu, kod dvosmjerne pretrage potrebno je dodatno čuvati liste bridova obrnutih orijentacija, koje se koriste u pretraživanju *unazad*. Stoga, takav pristup bi zauzeo dodatnu memoriju reda veličine $|E|$, što jednosmjerna pretraga može iskoristi za dodavanje većeg broja vrhova *orijentira* i čuvanja potrebnih informacija iz predprocesiranja. Ako memorijski prostor nije problem, dvosmjerno pretraživanje daje kraće vrijeme faze upita. Međutim, uspoređujemo li pretraživanja uz istu memorijsku potrošnju, tada je bolje koristiti jednosmjernu pretragu ukoliko imamo manji broj vrhova *orijentacije*. Naime, za razliku memorije koje je potrebno dvosmjernom pretraživanju, jednosmjernom pretraživanju bi se moglo dodati još nekoliko vrhova *orijentira*, što bi značajnije ubrzalo fazu upita. U suprotnom, ako se radi o većem broju *orijentira*, dodavanje par novih vrhova *orijentira* ne bi radilo značajnu razliku kod određivanja procjene udaljenosti. U tom slučaju se dvosmjerno pretraživanje pokazuje kao prikladniji odabir. Za više detalja pogledati [8].

Primjer 5.2.1. *Neka je neusmjereni graf G zadan kao na slici 5.2. Neusmjereni graf možemo promatrati kao usmjereni, gdje za svaki brid postoji po jedan brid u oba smjera te se na isti način provodi algoritam. Potrebno je odrediti najkraći put od vrha B do krajnjeg vrha I. Skup orijentira zadan je s $L = \{A, D\}$ te je stoga u fazi predprocesiranja potrebno izračunati $d_G(A, u)$ $(= d_G(u, A))$ i $d_G(D, u)$ $(= d_G(u, D))$, za sve vrhove u. U tu svrhu provodi se Dijkstrino pretraživanje iz svakog od navedenih vrhova orijentacije. Rezultati su zapisani u prva dva retka tablice 5.1.*



Slika 5.2: Primjer faze upita ALT algoritma. Vrhovi *orijentiri* su A i D te se traži $d_G(B, I)$.

U fazi upita potrebno je znati vrijednosti funkcije π_I. Radi jednostavnosti izračunati ćemo $\pi_I(u)$ prije početka izvršavanja A algoritma, za sve vrhove $u \in V$. U tablici 5.1 su od trećeg do petog retka prikazane vrijednosti funkcija $\pi_{I,A}$, $\pi_{I,D}$ i π_I. Prve dvije vrijednosti su potrebne pošto je $\pi_I(u) = \max(\pi_{I,A}(u), \pi_{I,D}(u))$. Iz 5.5 vrijedi: $\pi_{I,A}(u) = |d_G(A,u) - d_G(A,I)|$, pa iz prvog retka tablice lako izračunamo treći redak. Analogno za drugi i četvrti redak tablice 5.1. Konačnu procjenu udaljenosti π_I dobijemo uzimajući maksimume iz trećeg i četvrtog retka, tj. po vrijednostima $\pi_{I,A}$ i $\pi_{I,D}$.*

Uz izračunate vrijednosti potrebno je provesti postupak pronalaska najkraćeg puta s početkom u B i krajem u vrhu I. U tu svrhu provodi se jednosmjerno A pretraživanje s heurističkom funkcijom π_I. Postupak je isti kao kod Dijkstrinog algoritma, jedino što se prioritet vrha modificira. Prioritet je zadan izrazom $D[u] + \pi_I(u)$, gdje je $D[u]$ gornja ograda za najkraći put od B do u, kao kod Dijkstrinog pretraživanja. Na slici 5.2 narančastom bojom pokraj oznake vrha prikazana je vrijednost $D[u] + \pi_I(u)$. Ukoliko pretraživanje nije posjetilo neki od vrhova tada izostavljamo oznaku kraj imena vrha, s tim da imamo na umu da je gornja ograda za najkraći put do takvog vrha ostaje početna vrijednost $+\infty$. Podcrtane narančaste oznake označuju obrađene vrhove, one do kojih je ustanovljena točna vrijednost najkraćeg puta do vrha I. Indeks kraj takve oznake ukazuje na poredak kojim su obrađeni vrhovi. Iz primjera, redom su obrađeni vrhovi B, C, H i I. Kod posjećenih a*

neobrađenih vrhova, narančasta oznaka nije podcrtana te indeks kraj nje označava indeks rednog broja obrađenog vrha kod čije je obrade relaksacijom bridova promijenjen prioritet promatranog vrha. Tako na primjer oznaka $(3 + 11)_1$ *kraj vrha A znači da je prioritet vrha A postavljen na* $14 = D[A] + \pi_I(A)$ *kod obrade prvog obrađenog vrha. Radi se početnom vrhu B, čijom je relaksacijom bridova postavljena vrijednost vrha A.*

Pretraživanje A krenulo je obradom vrha B te su relaksacijom bridova postavljene vrijednosti kod vrhova* A, D, C *i* G*. Vrh C od posjećenih a neobrađenih vrhova ima najmanji prioritet, stoga je on drugi na redu za obradu. Relaksira se samo brid* (C, H) *te u sljedećem koraku na obradu dolazi na red vrh H najmanjeg prioriteta (procjene ukupne duljine puta od B do I prolazeći vrhom H). Relaksacijom bridova postavlja se vrijednost vrha I, koji u idućem koraku postaje obrađen čime pretraga prestaje pošto je pronađen završni vrh upita. Pamteći prethodnika kod svakog posjećenog vrha, lako se rekonstruira najkraći put* $\langle B, C, H, I\rangle$ *težine* 8*.*

Za kraj, promotrimo još posljednja dva retka tablice 5.1 gdje su prikazane vrijednosti $d_G(u, I)$ *i* $d_G(B, u)$ *za zadani graf G i sve vrhove u. Znamo da je* $\pi_I(u)$ *procjena udaljenosti do vrha I takva da vrijedi* $\pi_I(u) \leq d_G(u, I)$*, te da je cilj imati što bolju gornju ogradu. Vidimo da se za vrhove* A, B, C, D, H *i I postiže baš jednakost, što daje dobro usmjerenje A* pretraživanju. Intuicija koja kaže da će procjena biti dobra ukoliko se orijentir nalazi prije početnog ili iza završnog vrha na najkraćem putu pokazuje se dobra. Tako se npr. orijentir A nalazi prije početnog vrha B, te su* A, B, I *„kolinearne", tj. najkraći put od A do I i najkraći put od B do I se većim dijelom preklapaju. Orijentir D pomaže kod usmjerenja pretrage tako da pretraživanje ne posjećuje vrhove E i F koji sigurno nisu na najkraćem putu. Iz zadnjeg retka tablice vidimo kako je vrh I najudaljeniji od vrha B, iz čega slijedi da bi Dijkstrin algoritam obradio sve preostale vrhove prije nego ustanovi najkraći pod do vrha I. Vidimo da ALT algoritmom uz prikladan odabir vrhova orijentira dobijemo osjetno poboljšanje: obrađena su u primjeru samo četiri nužna vrha koja su na najkraćem putu, od njih ukupno devet koji bi bili obrađeni Dijkstrinim algoritmom.*

$u \rightarrow$	A	B	C	D	E	F	G	H	I
$d_G(A, u)$, $d_G(u, A)$	0	3	5	6	9	8	10	7	11
$d_G(D, u)$, $d_G(u, D)$	6	3	2	0	3	2	10	4	8
$\pi_{I,A}(u)$	11	8	6	5	2	3	1	4	0
$\pi_{I,D}(u)$	2	5	6	8	5	6	2	4	0
$\pi_I(u)$	11	8	6	8	5	6	2	4	0
$d_G(u, I)$	11	8	6	8	11	10	10	4	0
$d_G(B, u)$	3	0	2	3	6	5	7	4	8

Tablica 5.1: Prikaz vrijednosti najkraćih putova i njihovih procjena, izračunatih na grafu sa slike 5.2.

5.3 Dokaz ispravnosti ALT algoritma

Za početak dokažimo jednu vrlo jednostavnu lemu koja će se koristiti kod dokaza *izvedivosti* heurističke funkcije π_t.

Lema 5.3.1. *Neka su x_i, y_i, $i = 1, \ldots, n$ realni brojevi te neka je $I \subset \{1, \ldots, n\}$. Ukoliko vrijedi $x_i \leq y_i$ za svaki $i \in I$, tada vrijedi:* $\max_{i \in I} x_i \leq \max_{i \in I} y_i$.

Dokaz. Neka je indeks $k \in I$ takav da je $\max_{i \in I} x_i = x_k$. Tada vrijedi:

$$\max_{i \in I} x_i = x_k \leq y_k \leq \max_{i \in I} y_i. \tag{5.8}$$

□

Lema 5.3.2. *Za zadani skup vrhova orijentacije L, ciljni vrh $t \in V$, funkcija π_t je izvediva, gdje je*

$$\pi_t(u) = \max_{l \in L} \pi_{t,l}(u) = \max_{l \in L} \{\max(d_G(l,t) - d_G(l,u), d_G(u,l) - d_G(t,l))\},$$

za $u \in V$. Odnosno, vrijedi $\omega(u,v) - \pi_t(u) + \pi_t(v) \geq 0$, za sve bridove $(u,v) \in E$.

Dokaz. Neka je $l \in L$ proizvoljan fiksan vrh *orijentacije*. Sličnom argumentacijom kao kod dokaza nejednakosti (5.1), modificirana „nejednakost trokuta" nam daje: $d_G(u,l) \leq \omega(u,v) + d_G(v,l)$. To je jasno iz činjenice da najkraći put od u do vrha l ne može biti dulji od konkretnog puta između navedenih vrhova, koji koristi brid (u,v) te nastavlja najkraćim putom od v do krajnjeg vrha l. Oduzimanjem s obje strane vrijednosti $d_G(t,l)$ dobijemo:

$$d_G(u,l) - d_G(t,l) \leq \omega(u,v) + d_G(v,l) - d_G(t,l). \tag{5.9}$$

Analogno vrijedi i $d_G(l,v) \leq d_G(l,u) + \omega(u,v)$, iz čega zaključujemo:

$$d_G(l,t) - d_G(l,u) \leq \omega(u,v) + d_G(l,t) - d_G(l,v). \tag{5.10}$$

Uzimajući redom lijeve strane nejednakosti (5.10) i (5.9) za vrijednosti x_i, te desne strane kao niz y_i, koristeći lemu 5.3.1 vrijedi:

$$\begin{aligned} \pi_{t,l}(u) &= \max(d_G(l,t) - d_G(l,u),\ d_G(u,l) - d_G(t,l)) \\ &\leq \omega(u,v) + \max(d_G(l,t) - d_G(l,v),\ d_G(v,l) - d_G(t,l)) \\ &= \omega(u,v) + \pi_{t,l}(v) \end{aligned} \tag{5.11}$$

Dobili smo nejednakost $\pi_{t,l}(u) \leq \omega(u,v) + \pi_{t,l}(v)$, za svaki $l \in L$. Primjenom leme 5.3.1 zaključujemo $\max_{l \in L} \pi_{t,l}(u) \leq \omega(u,v) + \max_{l \in L} \pi_{t,l}(v)$, što po definiciji znači $\pi_t(u) \leq \omega(u,v) + \pi_t(v)$. Drugi način kako dovršiti dokaz nakon dobivene nejednakosti (5.11) je primijetiti da ona zapravo do definiciji daje izvedivost funkcije $\pi_{t,l}$, za svaki $l \in L$. Primjenjujući lemu 3.2.7 dobijemo izvedivost tražene funkcije π_t. □

Uz dokazanu *izvedivost* funkcije π_I, ispravnost ALT algoritma slijedi iz korektnosti A^* pretraživanja uz reduciranu funkciju težine π_I (vidjeti korolar 3.2.3 i napomenu 3.2.5). Drugi dokaz ispravnosti A^* algoritma uz *izvedivu* funkciju težine može se pronaći u [2].

Poglavlje 6

Implementacija i testiranje algoritama

U ovom poglavlju najprije opisujemo skupove podataka nad kojima ćemo testirati obrađene algoritme za problem najkraćeg puta u grafu. Nakon toga slijede pojedini implementacijski detalji i analiza bitnih parametara kod *Contraction Hierarchies* i *Landmark A** algoritma. U zadnjem dijelu nalazi se usporedba promatranih algoritma. Kako je Dijkstrin algoritam ogledni primjer za pronalazak najkraćeg puta, u svrhu usporedbe prikazujemo i rezultate dobivene takvim pretraživanjem. Kako je moguće postojanje više različitih najkraćih putova između promatrana dva vrha, tj. takav put ne mora nužno biti jedinstven, algoritmi neće vraćati najkraći put – niz vrhova. Točnije, zadatak faze *upita* bit će pronalazak vrijednosti najkraćeg puta za zadani početni i krajnji vrh u grafu. Svi algoritmi biti će testirani na istom skupu proizvoljno odabranih 1000 upita, te će se kao usporedba gledati prosječno vrijeme upita t_{pUpit}. Dodatno, kao mjeru koja ne ovisi o performansama računala i kvaliteti implementacije, promatrat ćemo prosječan broj *obrađenih* vrhova $N_{pObradeno}$. Vrijeme predprocesiranja označavat ćemo s $t_{predproc}$.

Za implementaciju i testiranje algoritama odabran je programski jezik *Java*. Računalo na kojemu su se izvršavali programi sadrži procesor AMD Ryzen 5 4600H, frekvencije radnog takta 3.00 GHz i 16.0 GB RAM memorije. Korišten je 64-bitni operacijski sustav Windows 10 Pro. Cjelokupni programski kod može se pronaći u [12].

6.1 Skup podataka

Algoritme za pronalazak najkraćeg puta u grafu testirat ćemo na stvarnim podacima cestovnih mreža, koje se mogu pronaći u [4]. Konkretno, za skup podataka uzimamo četiri grafa dobivena obradom informacija s prometnica iz američkih saveznih država: New Yorka, Colorada, Floride, Kalifornije i Nevade. Podaci iz Kalifornije i Nevade objedinjeni su u jedan graf. Redom ćemo grafove označavati kraćim imenima: NY, COL, FLA i CAL. Detalji o broju vrhova i bridova pojedinih grafova nalaze se u tablici 6.1. Važna prometna

čvorišta reprezentirana su vrhom, a cestovne poveznice između dva takva mjesta prikazana su usmjerenim bridovima grafa. U svakom grafu, svaki vrh označen je jedinstvenim rednim brojem te su poznati podaci geografske širine i dužine gdje se nalaze pojedini vrhovi. Takve informacije bile bi korisne npr. kod A^* algoritma s euklidskom udaljenosti kao procjenom duljine puta do ciljnog vrha. Kod Dijkstrinog algoritma, *Landmark* A^* i *Contraction Hierarchies* algoritma ti podaci nisu potrebni pa ih zanemarujemo. Dostupni su podaci o težinama bridova s obzirom na fizičku udaljenost vrhova i s obzirom na prosječno vrijeme putovanja između promatranih vrhova. Testiranja ćemo temeljiti na procijeni vremena putovanja između dva vrha, tj. cilj je pronaći najkraće vrijeme koje je potrebno za putovanje od početnog do ciljnog vrha. Prilikom obrade podataka potrebno je pripaziti na višestruke usmjerene bridove između istog para vrhova. U takvim situacijama pamtimo samo jedan brid u svakom smjeru, onaj s najmanjom vrijednosti. Naime, pošto tražimo najkraći put, zanemarivanjem višestrukih bridova koji imaju veću cijenu prelaska nećemo izgubiti potrebne informacije pošto najkraći put sigurno neće prolaziti njima.

	Savezna država	Broj vrhova	Broj bridova
NY	New York	264 346	730 100
COL	Colorado	435 666	1 042 400
FLA	Florida	1 070 376	2 687 902
CAL	Kalifonija i Nevada	1 890 815	4 630 444

Tablica 6.1: Prikaz informacija o skupovima podataka cestovnih mreža.

Svi promatrani algoritmi koristit će istu klasu *RoadNetwork* koja predstavlja graf nad kojim će se izvršavati upiti pronalaska vrijednosti najkraćeg puta. Takva klasa je prilagođena za potrebe svih algoritama, tako da ne mora nužno svaki od njih koristiti sve elemente. Kako su općenito grafovi dobiveni iz cestovnih mreža *rijetki*, kao što se to može vidjeti i iz podataka iz tablice 6.1, logičan je izbor podatke o bridovima spremiti kao listu susjeda (vidjeti 1.2). Naime, prosječan stupanj vrha je značajno manji od ukupnog broja vrhova u grafu, pa bi korištenje matrice susjedstva zauzimalo puno redundantne memorije. Kako se u dvosmjernom Dijkstrinom pretraživanju te i u *Contraction Hierarchies* algoritmu koristi pretraga unazad, za svaki vrh spremat ćemo listu ulaznih i listu izlaznih bridova. Ukoliko bi se ograničili na neusmjerene grafove, tada bi te liste bile identične te bi se moglo uštedjeti na potrošnji memorije. Metode klase *RoadNetwork* su standardne: čitanje podataka grafa iz tekstualne datoteke, dodavanje vrha, dodavanje usmjerenog brida te vraćanje podataka o broju bridova i vrhova. Napomenimo da je za ispravnost nekih algoritama bitno paziti da se ne dodaju višestruki usmjereni bridovi između dva vrha, već se uzima manja vrijednost težine brida. Klasa *Arc* predstavlja usmjereni brid grafa. Kako će liste susjeda određenog vrha sadržavati objekte te klase, u njoj je dovoljno pamtiti samo drugi vrh tog brida. Kao dodatak, klasa *Arc* sadrži jednu logičku varijablu koja, ukoliko je postavljena na

false, označava da se taj brid zanemaruje. Potreba za tim dolazi u *Contraction Hierarchies* algoritmu, o čemu se više nalazi u odjeljku 6.2.

6.2 *Contraction Hierarchies*

Contraction Hierarchies spada u hijerarhijske algoritme koji na predprocesiranju temelje ubrzanje faze upita. Proučavajući literaturu očekujemo kako će ovaj algoritam dati najkraća vremena kod računanja vrijednosti najkraćeg puta, no tu će cijenu „platiti" dužim vremenom predprocesiranja. Algoritam je implementiran klasom *ContractionHierarchies* koja temeljni rad prije faze upita bazira na metodama *preprocess*, *contractNode* i *calculateNodePriority*. Metoda *preprocess* objedinjuje fazu predprocesiranja: na temelju prioriteta vrhova određuje redoslijed kojim se provodi kontrakcija cijelog skupa vrhova. Provođenjem kontrakcija iz početnog grafa G dobijemo rezultantni graf G^* u čiji se skup bridova dodaju potrebni bridovi prečaci. Takav redoslijed vrhova predstavlja *rank* funkciju, čije će vrijednosti biti značajne kod odabira bridova u pronalasku najkraćeg puta između zadanih vrhova.

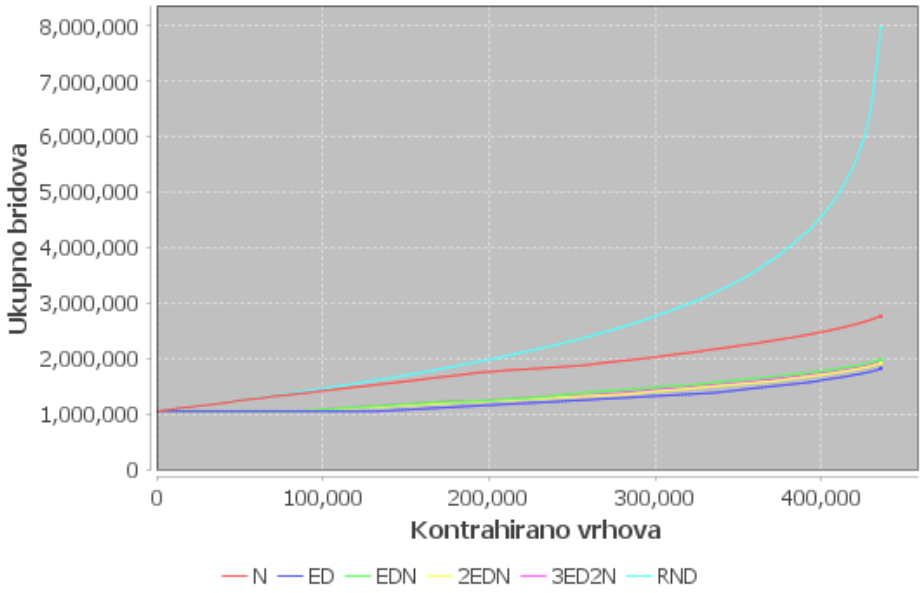
Kontrakcijom vrhova cilj je dodati što manji broj dodatnih bridova prečaca koji će se koristiti u fazi upita. Stoga, izuzetno je bitan redoslijed kojim se provode kontrakcije vrhova. U suprotnom bi u početni *rijedak* graf mogli dodati previše bridova što bi utjecalo na brzinu pronalaska najkraćeg puta. Naravno, broj dodanih prečaca utječe i na vrijeme izvođenja faze predprocesiranja, koje može biti značajno sporije ukoliko se uvede puno novih bridova. Prioritete i odgovarajuće vrhove pamtimo u prioritetnom redu iz kojeg se za novu kontrakciju bira vrh najmanje vrijednosti. Za ažuriranje reda koristimo metode *lijenog ažuriranja* i *ažuriranja susjeda* (vidjeti 4.1).

Autori u radu [7] navode *razliku bridova* kao jednu od najvažnijih vrijednosti koja opisuje prioritet vrha pri odabiru za kontrakciju. Kao što je to opisano u dijelu 4.1, razlika vrhova pojedinog brida može se izračunati simulirajući kontrakciju promatranog vrha. Naravno, tada bi metoda ustanovila broj potencijalno dodanih prečaca no ne bi ih stvarno dodala u rezultantni graf G^*. Takav način računanja razlike bridova pokazao se izuzetno vremenski zahtjevan. Čak se kod najmanjeg grafa, NY, s 260 tisuća vrhova faza predprocesiranja odvijala satima. Kako graf CAL sadrži skoro 1.9 milijuna vrhova, potrebno je pronaći drugačije rješenje ukoliko želimo dobiti razumna vremena predprocesiranja. Ključna stvar je sljedeća: pri kontrakciji vrha, pa time i računanju razlike bridova, potrebno je provoditi Dijkstrino pretraživanje iz svakog od ulaznih vrhova do svakog izlaznog vrha kontrahiranog čvora. Time bi se točno odredilo koliko bi bilo potrebno dodati bridova prečaca, te bi se dobila razlika bridova. Kako bi ubrzali sam proces, uvodimo procjenu razlike bridova koju ćemo označavati s ED. Ključna stvar je njena konstantna vremenska složenost. Za proizvoljan vrh v, označimo s $|V_{in,v}|$ broj ulaznih te $|V_{out,v}|$ broj izlaznih vrhova. Ulazni vrh vrha v je vrh u ukoliko postoji brid (u, v). Analogno značenje ima izlazni vrh. Vrijednost

procjene razlike bridova dana je izrazom:

$$ED_v = |V_{in,v}| \cdot |V_{out,v}| - |V_{in,v}| - |V_{out,v}|. \tag{6.1}$$

Takva procjena je zapravo razlika bridova u najgorem slučaju. Ukupno je moguće dodati najviše $|V_{in,v}| \cdot |V_{out,v}|$ bridova ukoliko je potreban prečac između svakog para ulaznog i izlaznog vrha. Broj bridova incidentnih vrhu v koji se uklanjaju njegovom kontrakcijom iznosi $|V_{in,v}| + |V_{out,v}|$. Uniformnost je drugi važan parametar koji je potrebno promotriti pri određivanju prioriteta vrha. Za svaki vrh promatrat ćemo broj njemu susjednih vrhova nad kojima je provedena kontrakcija. Taj faktor označavamo s N. Konačno, za vrijednost prioriteta vrha uzet ćemo linearnu kombinaciju vrijednosti ED i N. Slika 6.1 prikazuje ukupan broj bridova rezultantnog grafa u ovisnosti o broju vrhova nad kojima je provedena kontrakcija. Prikazana je usporedba za više opisanih linearnih kombinacija. Odabrani skup podataka nad kojim se provelo testiranje je COL, s ukupno 435 666 vrhova i 1 042 400 bridova.



Slika 6.1: Usporedba ukupnog broja bridova rezultantnog grafa G^* dobivenog različitim vrijednostima prioriteta vrhova.

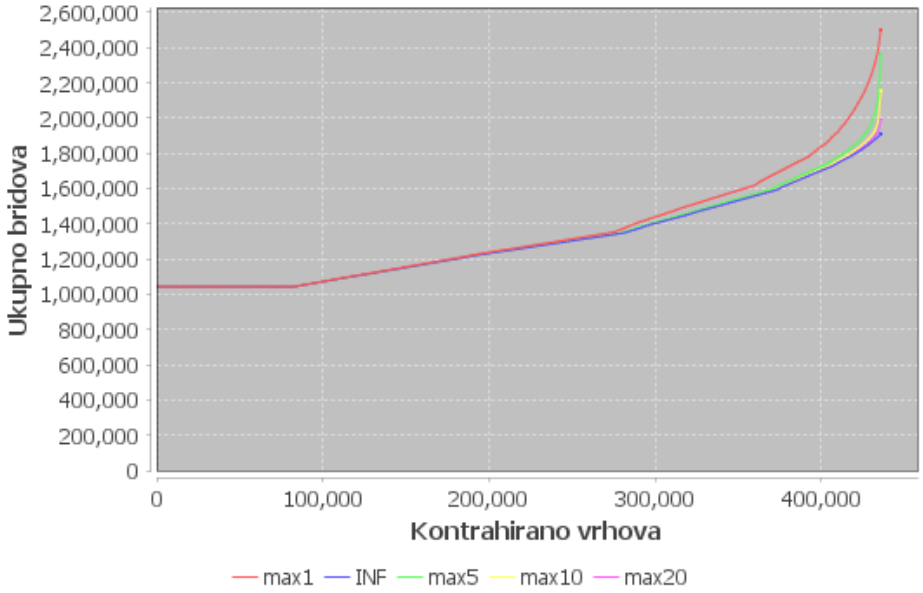
Usporedimo li vrijednosti kad uzmemo samo u obzir uniformnost N (crvena krivulja) i procjenu razlike bridova ED (tamnoplava krivulja), vidimo koliko se razlikuju krajnji grafovi G^* po broju bridova. Tablica 6.2 donosi detaljniju usporedbu raznih linearnih kombinacija prioriteta prikazanih na slici 6.1. Vidimo kako prioritet N doda preko duplo

više bridova od prioriteta ED, što se očituje i kroz duplo duže vrijeme predprocesiranja. Međutim, uniformnost se pokazuje kao bitan faktor kod prosječnog broja obrađenih vrhova u fazi upita te na kraju i kod prosječnog vremena izvršavanja jednog upita. Iako razlike nisu prevelike, lako je uočiti da dodavanjem faktora N kod prioriteta vrha dobivamo blaga ubrzanja prosječnog računanja najkraćeg puta. Također, uzimajući netrivijalnu linearnu kombinaciju vrijednosti ED i N dobivamo prihvatljivije vrijeme predprocesiranja i manji broj dodanih bridova prečaca. Kako bi pokazali važnost „pametnog" dodjeljivanja redoslijeda vrhova pokrenuli smo proces predprocesiranja uz proizvoljan poredak, bez računanja prioriteta kod redoslijeda kontrakcije. Takvo izvršavanje daleko premašuje sve ostale pristupe po broju dodanih prečaca. Faza predprocesiranja traje značajno duže, preko 12 sati, što na kraju ne pridonosi niti brzini faze upita. Krajnji graf G^* sadrži puno bridova te je dvosmjerna pretraga u fazi upita osjetno sporija. Uspoređujući promatrane načine dodjeljivanja prioriteta, u daljnjim testiranjima i usporedbama uzimamo vrijednost $2ED + N$.

	Prioritet	Dodano bridova	$t_{predproc}$	t_{pUpit}	$N_{pObradeno}$
N	N	1 722 113	29 min 10.35 s	3.407 ms	246.11
ED	ED	767 267	14 min 31.47 s	3.716 ms	672.46
EDN	$ED + N$	929 317	16 min 36.30 s	3.430 ms	402.42
2EDN	$2ED + N$	866 272	16 min 5.85 s	3.429 ms	444.11
3ED2N	$3ED + 2N$	888 866	16 min 27.91 s	3.458 ms	459.94
RND	nasumično	6 921 552	12 h 35 min 2.06 s	58.991 ms	2761.80

Tablica 6.2: Prikaz informacija o predprocesiranju i fazi upita za različite načine određivanja prioriteta vrhova pri odabiru za kontrakciju.

Nadalje, nakon što fiksiramo prioritet vrha na vrijednost $2ED + N$, na istom grafu COL prikazat ćemo rezultate još jednog načina za potencijalnim ubrzanjem algoritma. Kod svake kontrakcije vrha v se za svaki ulazni vrh u, takav da postoji brid (u, v), provodi posebno Dijkstrino pretraživanje. Njega smo ograničili da stane ukoliko na obradu dođe vrh vrijednosti veće od $\omega(u, v) + \max\{\omega(v, w) \colon (v, w) \in E^*\}$. Naravno, vrijednosti *rank* funkcije u vrhovima u i w moraju biti veće od v, što znači da ti vrhovi još nisu kontrahirani. Kao dodatni način ubrzanja uvodimo gornju ogradu na broj obrađenih vrhova u svakom od takvih pretraživanja. Ovdje moramo biti oprezni – smanjili smo broj obrađenih vrhova u pretraživanjima kod procesa kontrakcije te time očekujemo potencijalnu uštedu na vremenu predprocesiranja. Međutim, kako u tom slučaju Dijkstrino pretraživanje nije potpuno, moguće je nekad dodati brid prečac iako takav nije potreban. Time bi se moglo dogoditi ubacivanje prevelikog broja bridova što bi negativno utjecalo na promatranu brzinu obrade. Slika 6.2 prikazuje ovisnost ukupnog broja bridova grafa G^* s obzirom na broj vrhova nad kojima je provedena kontrakcija, za razne vrijednosti ograničenja broja obrađenih vrhova.



Slika 6.2: Usporedba ukupnog broja bridova rezultantnog grafa G^* u ovisnosti o gornjoj granici za svako Dijkstrino pretraživanje koje se obavlja u fazi kontrakcije vrha.

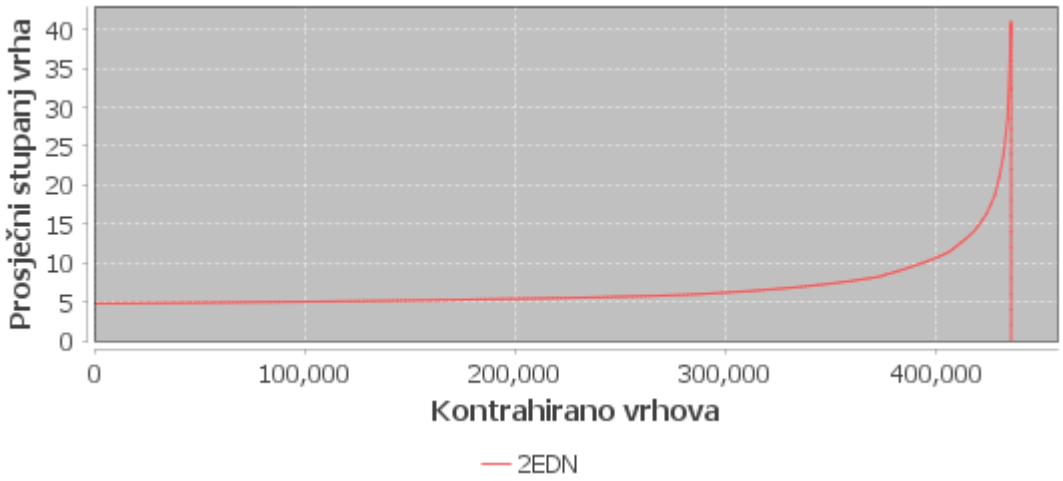
Kao i očekivano, ukoliko postavimo ograničenje na maksimalan prikaziv cijeli broj (čime efektivno dobijemo izvršavanje bez ograničenja) ukupan broj dodanih prečaca je najmanji. Druga krajnost je ograničenje pretrage na jedan obrađeni vrh – $max1$ (crvena krivulja). Time se *relaksiraju* bridovi incidentni vrhu v nad kojim se provodi kontrakcija te se uvodi prečac (u, w) ukoliko već ne postoji takav brid težine barem $\omega(u, v) + \omega(v, w)$. Očekivano, takvim pristupom dodali smo najviše bridova prečaca. Tablica 6.3 prikazuje detalje o fazama predprocesiranja i upita za sva promatrana ograničenja. Vidimo kako npr. $max1$ ograničenje iziskuje kraće vrijeme predprocesiranja no ujedno daje i bolje vrijeme faze upita nego ograničenje $max5$. Razlog leži u tome kako svaka pojedina kontrakcija vrha uštedi na vremenu kod Dijkstrinih pretraživanja iz ulaznih vrhova, no ipak se ne doda značajno više prečaca nego kod $max5$, što bi usporilo predprocesiranje. Međutim, ti dodani prečaci čine razliku kod faze upita, stoga vidimo da je takvo vrijeme kod ograničenja $max1$ kraće nego kod $max5$. Povećavajući gornju ogradu na broj obrađenih vrhova, za očekivati je da se uvodi manje bridova pošto su pretraživanja točnija. To potvrđuju i podaci iz tablice 6.3. Za daljnja testiranja odabrat ćemo granicu INF koja uvodi najmanji broj dodatnih bridova. Rezultantni graf G^* *rjeđi* je nego za preostale granične vrijednosti, što daje i najkraće vrijeme faze upita. Kako je broj vrhova u grafu uvijek daleko manji od maksimalnog prikazivog cijelog broja INF, zapravo svaku procjenu nužnosti dodavanja prečaca puštamo na obradu u potpunosti, kao što je to bilo opisano u dijelu 4.1.

	Max. obrađenih	Dodano bridova	$t_{predproc}$	t_{pUpit}	$N_{pObradeno}$
max1	1	1 460 966	17 min 34.54 s	4.037 ms	335.58
max5	5	1 315 575	18 min 8.8 s	6.374 ms	609.83
max10	10	1 113 814	16 min 48.48 s	6.626 ms	624.71
max20	20	951 026	15 min 8.86 s	4.411 ms	420.52
INF	2 147 483 647	860 272	16 min 3.85 s	3.455 ms	444.11

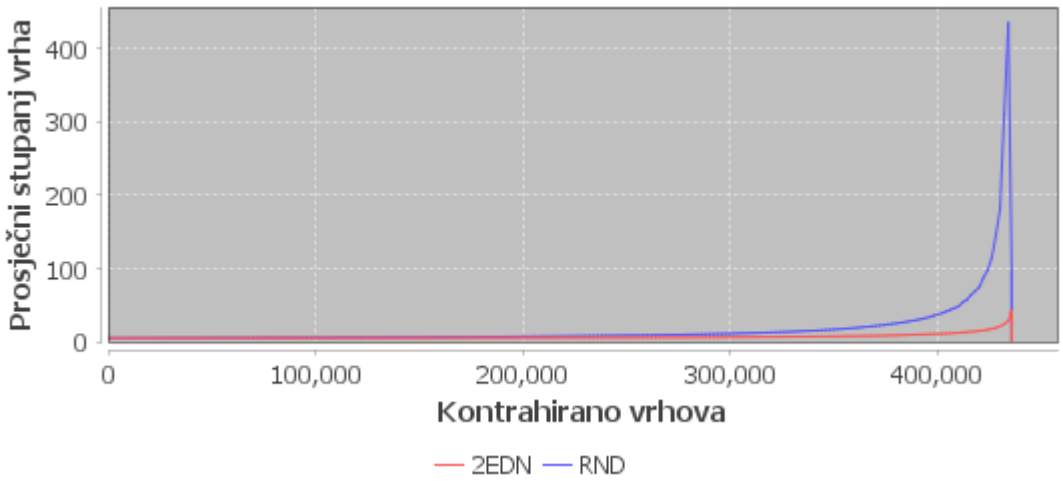
Tablica 6.3: Prikaz informacija o predprocesiranju i fazi upita za različite vrijednosti ograničenja na broj obrađenih vrhova pri procjeni dodavanja brida prečaca.

Primijetimo još jedan zanimljiv fenomen kod faze predprocesiranja *Contraction Hierarchies* algoritma. Vrijeme provođenja kontrakcija vrhova ne ponaša se linearno. Točnije, na početku se vrhovi kontrahiraju u puno kraćem vremenskom roku u usporedbi sa zadnjim dijelom vrhova koji dolaze na red. Razlog tome je taj što je početni graf u pravilu *rijedak*, te je kod kontrakcije vrha potrebno provesti svega nekoliko Dijkstrinih pretraživanja. Međutim, kako se određeni broj vrhova kontrahira, trenutni graf s ubačenim prečacima i izbačenim kontrahiranim vrhovima postaje sve gušći. Stupanj vrha promatrat ćemo kao zbroj ulaznog i izlaznog stupnja, tj. broj ulaznih i izlaznih incidentnih bridova. Slika 6.3 prikazuje za odabrane parametre $2ED + N$ i INF rast prosječnog stupnja vrha u trenutnom grafu koji se sastoji od dodanih prečaca i bez izbačenih vrhova. Radi usporedbe, prikazujemo i sliku 6.4 koja prikazuje odnos prosječnog stupnja vrha kod proizvoljnog poretka kontrakcije bridova i onog odabranog analizom bitnih parametara. Kako se na kraju izbace svi vrhovi, prosječni stupanj mora pasti na nulu, što je i vidljivo iz priloženih grafova.

Potrebno je još pojasniti kako se izvodi brisanje vrha prilikom njegove kontrakcije. To se jednostavno simulira pamteći vrijednosti *rank* funkcije vrhova. Ukoliko vrijednost za pojedini vrh nije postavljena, to znači da će taj vrh biti kasnije kontrahiran te je još „aktivan" u trenutnom grafu. Rezultat toga je zanemarivanje bridova i vrhova koji već imaju postavljenu vrijednost *rank* funkcije. Naime, u trenutnom grafu G_i nakon i provedenih kontrakcija, promatramo samo vrhove i bridove među vrhovima koji do tad nisu kontrahirani. Takvi će imati veću vrijednost *rank* funkcije od one trenutnog vrha u fazi kontrakcije. Izbacivanje vrhova i njima pripadajućih bridova moglo se ostvariti i stvarnim izbacivanjem iz strukture grafa. No, u tom slučaju bi morali u memoriji čuvati rezultantni graf G^* te trenutni graf G_i, dok opisanim pristupom nema potrebe za time. Također, izbacivanje brida iz liste susjedstva ne može se obaviti u konstantnom vremenu, što bi utjecalo na vrijeme predprocesiranja. U drugu ruku, simulirajući graf G_i dok u stvarnosti radimo sa cijelim grafom G^* zanemarujući kontrahirane bridove, može također igrati ulogu kod vremena izvršavanja obrade. Liste susjedstva imaju više elemenata od kojih se dobar dio zanemaruje. Usporedba na manjim skupovima podataka nije pokazala bitniju razliku između ova dva pristupa, što može biti daljnji prostor za napredak i ubrzanje.



Slika 6.3: Primjer ovisnosti prosječnog stupnja vrha o broju vrhova nad kojima je provedena kontrakcija. Promatra se privremeni graf bez izbačenih vrhova, u kojem su dodani bridovi prečaci.



Slika 6.4: Usporedba prosječnog stupnja vrha trenutnog grafa dobivenog kontrakcijom vrhova. Prikazani su rezultati s optimalnim parametrima – 2EDN te oni dobiveni kontrakcijom slučajnog poretka – RND.

Što se tiče faze upita, provodi se dvosmjerno Dijkstrino pretraživanje nad grafovima $G^*_\uparrow$ i $G^*_\downarrow$ kao što je to opisano u dijelu 4.2. Kao dodatno ubrzanje koristimo *stall-on-demand* tehniku koja prekine pretragu u određenom smjeru ukoliko ustanovi da ne vodi ka krajnjem

rješenju. Grafovi $G^*_\uparrow$ i $G^*_\downarrow$ ne zauzimaju novu memoriju, već se implicitno kreiraju iz grafa G^* dobivenog dodavanjem bridova prečaca u početni graf G. U tu svrhu koristimo logičku varijablu unutar objekta klase *Arc*, čime je realiziran usmjereni brid. U listi izlaznih bridova kod vrha v, bridu (v, w) postavimo varijablu na *true* ako i samo ako je $rank(v) < rank(w)$. Time jednoznačno određujemo graf $G^*_\uparrow$. Kod popisa ulaznih bridova vrha v, brid (u, v) ima postavljenu logičku varijablu na *true* ako i samo ako je $rank(u) > rank(v)$. Promatrajući samo bridove sa postavljenom „zastavicom" *true* kod liste ulaznih bridova realiziramo graf $G^*_\downarrow$.

6.3 *Landmark* A^*

Algoritam *Landmark* A^* implementiran je klasom *LandmarkAlgorithm* koja kao i *ContractionHierarchies* u sebi sadrži objekt klase *DijkstrasAlgorithm* koji realizira Dijkstrin algoritam. Faza upita razlikuje se od Dijkstrinog pretraživanja samo u definiciji prioriteta vrha. Naime, kod Dijkstrinog pretraživanja uvijek na obradu dolazi vrh v s najmanjom trenutnom pronađenom udaljenosti $D[v]$. ALT algoritam bira novi vrh s najmanjom vrijednosti $D[v] + \pi_t(v)$, gdje je $\pi_t(v)$ procjena udaljenosti do ciljnog vrha t. U tu svrhu kod implementacije Dijkstrinog algoritma postavljamo logičku zastavicu koja označava uzimamo li u obzir heuristiku. Ukoliko je postavljena na *true*, tada imamo zapravo A^* pretraživanje. Implementirali smo dva načina odabira vrhova orijentacije: na slučajan način i pohlepnim najudaljenijim odabirom, opisanim u dijelu 5.1. Kod najudaljenijeg pohlepnog pristupa javlja se problem pronalaska vrha koji je najudaljeniji od trenutnog odabranog skupa. Taj problem može se efikasno riješiti jednim modificiranim Dijkstrinim pretraživanjem. Naime, algoritam je isti samo što na početku imamo više početnih vrhova. Nazovimo taj skup sa S. Tada je izračunata vrijednost za vrh v jednaka $D[v] = \min_{s \in S} d_G(s, v)$, što daje točno traženu vrijednost. Za više detalja pogledati [7].

Nakon odabira vrhova orijentacije, potrebno je za svaki takav vrh izračunati duljine najkraćih putova *od* i *do* svih preostalih vrhova grafa. Za fiksni vrh koji ima ulogu orijentira, Dijkstrin algoritam pronalazi udaljenosti do svih preostalih čvorova. Ovdje se zapravo koristi verzija koja nema zadan ciljni vrh u pretraživanju, već pronalazi najkraće putove do svih povezanih vrhova. Problem nastaje kod određivanja vrijednosti $d_G(u, l)$, gdje je $u \in V$ proizvoljan te l vrh orijentir. To je moguće riješiti pokretanjem zasebnog pretraživanja za svaki vrh u, no to nikako ne bi bilo efikasno rješenje. Ideja leži u sljedećem: najkraći put od u do l jednak je najkraćem putu od l do u u grafu s bridovima obrnutih orijentacija. Stoga, dovoljno je pokrenuti za svaki orijentir l još jedno Dijkstrino pretraživanje, ovoga puta *unazad*, koje izračuna potrebne vrijednosti.

Što se tiče računanja konkretnih vrijednosti procjene udaljenosti, iznimno je bitno da se taj proces izvršava samo kad je to uistinu potrebno. Računanje vrijednosti $\pi_t(v)$ za svaki vrh v dodatno usporava algoritam. ALT pretraživanje uz dobru heurističku funkciju

značajno smanji prostor pretrage, samim time i broj posjećenih vrhova. Stoga, za posjećene vrhove pamtimo izračunate vrijednosti heuristike, koju ne računamo za u prosjeku velik broj preostalih vrhova.

Parametri kod *Landmark* A^* algoritma su odabir broja vrhova orijentacije te način zadavanja skupa orijentacije. Usporedbu ćemo prikazati na dva skupa podataka: NY s ukupno 264 346 vrhova i 730 100 bridova, te COL skupom podataka koji sadrži 435 666 vrhova i 1 042 100 bridova. Tablice 6.4, 6.5, 6.6 i 6.7 prikazuju dobivene rezultate. Oznaka ALT-n predstavlja ALT algoritam s ukupno n vrhova orijentacije. Kako procjena udaljenosti nikad ne nadmašuje stvarnu najkraću udaljenost, možemo promatrati omjer tih dviju vrijednosti. Vrijednost omjera bila bi 1 ukoliko bi heuristika vraćala egzaktnu vrijednost najkraćeg puta. Očekujemo da bi preciznija heuristika trebala dati bolja rješenja. Prosječna vrijednost takvog omjera prikazana je u zadnjem stupcu navedenih tablica.

	$t_{predproc}$	t_{pUpit}	$N_{pObradeno}$	Procjena
ALT-1	0.59 s	47.899 ms	60 952.57	42.43 %
ALT-2	1.01 s	41.769 ms	52 728.40	49.22 %
ALT-4	1.95 s	22.767 ms	27 293.71	77.26 %
ALT-8	3.61 s	21.545 ms	20 675.66	84.23 %
ALT-16	6.82 s	17.552 ms	11 665.91	86.58 %
ALT-24	10.30 s	18.692 ms	9 824.49	87.20 %
ALT-32	13.51 s	20.671 ms	8 676.92	87.76 %

Tablica 6.4: Usporedba ALT algoritama s obzirom na broj vrhova orijentacije, čiji se odabir izvršava na slučajan način. Skup podataka nad kojim je provedena usporedba je NY.

	$t_{predproc}$	t_{pUpit}	$N_{pObradeno}$	Procjena
ALT-1	0.82 s	46.403 ms	61 794.15	56.42 %
ALT-2	1.57 s	26.681 ms	34 370.61	71.23 %
ALT-4	2.72 s	14.813 ms	17 714.79	82.97 %
ALT-8	5.11 s	13.575 ms	12 989.65	86.02 %
ALT-16	10.44 s	14.849 ms	9 522.91	89.09 %
ALT-24	15.47 s	14.827 ms	7 330.60	91.27 %
ALT-32	20.53 s	16.269 ms	6 471.39	91.66 %

Tablica 6.5: Usporedba ALT algoritama s obzirom na broj vrhova orijentacije, čiji se odabir određuje pohlepnim najudaljenijim pristupom. Skup podataka nad kojim je provedena usporedba je NY.

Kod proučavanja rezultata moramo imati na umu da faktor slučajnosti igra ulogu u oba načina odabira skupa orijentacije. Kod slučajnog odabira skupa očito, no i kod pohlepnog

	$t_{predproc}$	t_{pUpit}	$N_{pObradeno}$	Procjena
ALT-1	0.85 s	72.801 ms	108 954.04	54.63 %
ALT-2	1.44 s	68.549 ms	100 044.24	68.73 %
ALT-4	2.78 s	49.895 ms	64 907.34	74.73 %
ALT-8	5.38 s	31.819 ms	32 381.95	86.64 %
ALT-16	10.12 s	34.121 ms	23 657.20	88.13 %
ALT-24	18.95 s	37.683 ms	18 309.01	90.73 %
ALT-32	24.02 s	42.102 ms	16 888.85	91.41 %

Tablica 6.6: Usporedba ALT algoritama s obzirom na broj vrhova orijentacije, čiji se odabir izvršava na slučajan način. Skup podataka nad kojim je provedena usporedba je COL.

	$t_{predproc}$	t_{pUpit}	$N_{pObradeno}$	Procjena
ALT-1	1.88 s	84.902 ms	107 504.52	64.38 %
ALT-2	3.43 s	49.380 ms	66 754.56	74.93 %
ALT-4	6.74 s	31.579 ms	37 339.80	86.50 %
ALT-8	10.84 s	27.876 ms	27 601.40	90.44 %
ALT-16	19.48 s	34.789 ms	22 308.64	92.13 %
ALT-24	24.02 s	37.632 ms	18 563.00	93.25 %
ALT-32	35.81 s	43.861 ms	17 637.13	93.80 %

Tablica 6.7: Usporedba ALT algoritama s obzirom na broj vrhova orijentacije, čiji se odabir određuje pohlepnim najudaljenijim pristupom. Skup podataka nad kojim je provedena usporedba je COL.

najudaljenijeg odabira se na slučajan način odabire početni vrh. Iz tog razloga se ponovnim pokretanjem opisanih algoritama najvjerojatnije ne bi dobili identični rezultati. Stoga, usporedbu promatramo na dva različita skupa podataka NY i CAL kako bi donijeli što bolju odluku o izboru parametara. Iz tablica primjećujemo kako prosječno vrijeme faze upita pada s povećanjem broja orijentira. Razlog leži u tome što na taj način dobijemo precizniju procjenu ciljne udaljenosti. Međutim, kada se broj orijentira previše poveća, dobijemo suprotan učinak. Naime, za svako računanje heurističke vrijednosti (5.6) potrebno je $O(l)$ vremena, gdje je l broj vrhova orijentacije. Iako se preciznost procjene uvećava, razlika nije toliko značajna da bi ubrzala fazu upita, već „skuplje" računanje heuristike dolazi do izražaja. Zanimljivo je primijetiti kako instance algoritma s većim brojem orijentira uistinu daju bolju procjenu udaljenosti, što se očituje i u prosječnom broju obrađenih vrhova prilikom računanja vrijednosti najkraćeg puta. Za daljnju usporedbu odabiremo 8 vrhova orijentacije i pohlepni najudaljeniji način odabira takvih vrhova. Time dobijemo malo lošiju heurističku procjenu od algoritama s većim brojem orijentira, no značajno je

kraće vrijeme predprocesiranja te se pokazuje najboljim i vrijeme faze upita.

6.4 Usporedba rezultata

U ovom odjeljku donosimo usporedbu obrađenih algoritama. Pojedini parametri opisani su u prethodnim odgovarajućim poglavljima. Prikazat ćemo vrijeme potrebno za predprocesiranje $t_{predproc}$, prosječno vrijeme faze upita t_{pUpit} te prosječan broj obrađenih vrhova $N_{pObradeno}$. Tablice 6.8, 6.9, 6.10 i 6.11 donose redom rezultate izvršavanja algoritama na skupovima podataka NY, COL, FLA i CAL. Kako bi se provjerila točnost rješenja, povratne vrijednosti CH i ALT algoritma uspoređuju se s vrijednosti najkraćeg puta dobivenog Dijkstrinim algoritmom. Naglasimo kako su sva tri algoritma testirana na istom skupu od 1000 slučajno odabranih parova vrhova, gdje fazu upita predstavlja pronalazak vrijednosti najkraćeg puta. Važno je imati na umu da u ovoj usporedbi nije potrebna rekonstrukcija takvog puta. Kako *Contraction Hierarchies* algoritam uvodi bridove prečace koji su sastavni dijelovi pronađenog puta, da se tražio konkretan niz vrhova takvog puta bilo bi potrebno provesti postupak opisan u dijelu 4.2. Samim time bi broj obrađenih vrhova CH algoritma bio blago veći.

	$t_{predproc}$	t_{pUpit}	$N_{pObradeno}$
Dijkstra	0 s	94.460 ms	132 288.81
CH	13 min 58.97 s	4.348 ms	664.06
ALT	5.11 s	13.575 ms	12 989.65

Tablica 6.8: Usporedba algoritama na skupu podataka NY koji sadrži 264 346 vrhova i 730 100 bridova.

	$t_{predproc}$	t_{pUpit}	$N_{pObradeno}$
Dijkstra	0 s	140.809 ms	222 503.23
CH	15 min 56.85 s	3.429 ms	444.11
ALT	10.84 s	27.876 ms	27 601.40

Tablica 6.9: Usporedba algoritama na skupu podataka COL koji sadrži 435 666 vrhova i 1 042 400 bridova.

Uz vremensku usporedbu možemo promatrati koliko dodatne memorije zahtijeva pojedini algoritam. Kako se CH i ALT algoritmi zasnivaju na modifikacijama Dijkstrinog pretraživanja, faze upita zahtijevaju podjednako mnogo memorije. Algoritam CH u fazi predprocesiranja okvirno udvostruči broj bridova dodavanjem potrebnih prečaca, kao što je to vidljivo sa slike 6.1 i tablice 6.2. Time smo ugrubo udvostručili potrebnu memoriju za

	$t_{predproc}$	t_{pUpit}	$N_{pObradeno}$
Dijkstra	0 s	375.670 ms	530 997.30
CH	2 h 39 min 17.98 s	8.842 ms	251.83
ALT	20.53 s	66.844 ms	62 278.29

Tablica 6.10: Usporedba algoritama na skupu podataka FLA koji sadrži 1 070 376 vrhova i 2 687 902 bridova.

	$t_{predproc}$	t_{pUpit}	$N_{pObradeno}$
Dijkstra	0 s	741.203 ms	918 261.76
CH	4 h 50 min 28.70 s	10.59 ms	686.33
ALT	42.11 s	124.364 ms	103 718.06

Tablica 6.11: Usporedba algoritama na skupu podataka CAL koji sadrži 1 890 815 vrhova i 4 630 444 bridova.

zapis originalnog grafa. *Landmark* A^* algoritam iziskuje $O(l \cdot n)$ dodatne memorije, gdje je $l = 8$ broj vrhova orijentacije a $n = |V|$ broj vrhova polaznog grafa. Naime, za svaki vrh orijentacije potrebno je pamtiti odgovarajuće vrijednosti *od* i *do* preostalih vrhova.

Iz dobivenih podataka vidimo kako oba algoritma, CH i ALT, značajno ubrzavaju vrijeme faze upita u usporedbi s Dijkstrinim pretraživanjem. Kod grafova s većim brojem vrhova i bridova takva su poboljšanja još očitija. Tako CH algoritam daje ubrzanje od otprilike 70 puta, dok ALT ubrza fazu upita približno 6 puta, uspoređujući s Dijkstrinim algoritmom. Ukoliko fazu predprocesiranja nije potrebno često ponovno izvoditi, *Contraction Hierarchies* algoritam se čini kao jako dobro rješenje za brz pronalazak najkraćeg puta. Algoritam *Landmark* A^* daje pravi balans između dvije faze problema. Uz značajno kraću fazu predprocesiranja postiže se poprilično dobro ubrzanje faze upita.

Poglavlje 7

Zaključak

U ovom radu detaljno smo opisali i testirali dva algoritma koja ubrzanje pronalaska najkraćeg puta temelje na prethodnoj obradi podataka. Iz skupine algoritama koja uvodi hijerarhiju na skupu vrhova ili bridova predstavili smo *Contraction Hierarchies* – CH algoritam. Druga skupina algoritama koji se temelje na predprocesiranju su ciljno orijentirani algoritmi od kojih smo se upoznali s *Landmark* A^* – ALT algoritmom. CH algoritam iziskuje značajno više vremena za fazu predprocesiranja, što rezultira daleko najkraćom fazom upita, uspoređujući ju s ALT i Dijkstrinim pretraživanjem. ALT pretraga također daje zamjetno ubrzanje te je njena implementacija jednostavnija u odnosu na CH algoritam. Uz to, vrijeme predprocesiranja se uvelike smanji, što je pogodnije za probleme kod kojih je potrebno često ponovno provođenje te faze. Uz jače računalne resurse bilo bi zanimljivo napraviti usporedbu rezultata dobivenih na grafovima s još većim brojem vrhova i bridova. Naime, iz dijela 6.4 vidimo kako su na većim grafovima ubrzanja još značajnija.

Kao daljnji prostor za napredak moglo bi se pokušati dodatno ubrzati algoritme modificirajući njihove implementacije. Iako se na manjim skupovima podataka nije pokazao kao bolji pristup, kod CH algoritma mogla bi se kontrakcija vrha realizirati stvarnim izbacivanjem vrha iz grafa. U tom slučaju bi bilo potrebno pamtiti trenutni i rezultantni graf G^*, što bi zahtijevalo dodatnu memoriju. Takvo izbacivanje uz korištenje liste susjedstva nije najoptimalnije pošto se ne može obaviti u konstantnom vremenu. Međutim, u svim daljnjim procesima kontrakcije vrhova ne bi više bilo potrebno pretraživati cijelu listu kako bi se našli preostali neizbačeni vrhovi, što bi moglo ubrzati predprocesiranje.

Zanimljivo je pitanje mogu li se CH i ALT algoritmi kombinirati zajedno producirajući bolje rezultate. Tematikom spajanja hijerarhijskih i ciljno orijentiranih algoritama bave se pojedini znanstveni članci, te bi takav pristup mogao donijeti još zapaženija ubrzanja. Stalnim izmjenama rezultiranim prometnim gužvama moglo bi se promatrati i dinamičke grafove kod kojih se težine bridova često mijenjaju. Kod takvih grafova vrijeme predprocesiranja bi trebalo biti što kraće te bi veliko ubrzanje faze upita bio popriličan izazov.

Bibliografija

[1] R. K. Ahuja, K. Mehlhorn, J. B Orlin i R. E. Tarjan, *Faster Algorithms for the Shortest Path Problem*, MIT Operations Research Center, 1988.

[2] H. Bast, *Efficient Route Planning*, Chair for Algorithms and Data Structures, Department of Computer Science, University of Freiburg (2012), https://ad-wiki.informatik.uni-freiburg.de/teaching/EfficientRoutePlanningSS2012.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest i C. Stein, *Introduction to Algorithms*, sv. 3, MIT Press and McGraw-Hill, 2009.

[4] C. Demetrescu, A. Goldberg i D. Johnson, *9th DIMACS Implementation Challenge - Shortest Paths*, 2006.

[5] E. W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*, 1959.

[6] S. Funke, *Contraction Hierarchies briefly explained*, https://fmi.uni-stuttgart.de/files/alg/teaching/s15/alg/CH.pdf, siječanj 2017.

[7] R. Geisberger, P. Sanders, D Schultes i D. Delling, *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks*, WEA 2008 (2008), 319–333.

[8] A. V. Goldberg i C. Harrelson, *Computing the Shortest Path: A* Search Meets Graph Theory*, Technical Report MSR-TR-2004-24, Microsoft Research (2004), https://www.microsoft.com/en-us/research/wp-content/uploads/2004/07/tr-2004-24.pdf.

[9] K.S Jørgensen, *Shortest paths in Directed Graphs*, Magistarska radnja, Department of Computer Science, University of Aarhus, svibanj 2004, https://www.karstenstrandgaard.dk/thesis/ShortestPathsJorgensen.pdf, Theory and Practice of a New Hierarchy-Based Algorithm.

[10] S. Knopp, P. Sanders, D. Schultes, F. Schulz i D. Wagner, *Computing Many-to-Many Shortest Paths Using Highway Hierarchies*, Workshop on Algorithm Engineering and Experiments (ALENEX) (2007), 36–45.

[11] D. Kusalić, *Napredno programiranje i algoritmi u C-u i C-++-u*, Element, 2014.

[12] A. Mandić, *Brzi algoritmi za problem najkraćeg puta*, rujan 2021, `https://github.com/andrijaMandic/BrziAlgoritmiZaProblemNajkracegPuta`.

[13] M. Mohri, *Finite-State Transducers in Language and Speech Processing*, (1997), `https://cs.nyu.edu/~mohri/pub/cl1.pdf`.

[14] H. Ortega-Arranz, D. R. Llanos i A. Gonzalez-Escribano, *The Shortest-Path Problem: Analysis and Comparison of Methods*, Morgan & Claypool, 2014.

[15] D. B. West, *Introduction to Graph Theory*, sv. 2, Pearson Education, 2001, `https://ia801204.us.archive.org/35/items/igt_west/igt_west_text.pdf`.

Sažetak

Problem pronalaska najkraćeg puta u grafu široko je primjenjiv u stvarnom svijetu. Mnoge se stvari matematički modeliraju pomoću grafova čime se stvara potreba za što boljim i bržim algoritmima za rješenje ovog problema. Primjeri su razne vrste prometnica i situacija u prometu, internetske mreže, posebne vrste baza podataka te razni drugi. Prvi dio rada posvećen je klasičnim algoritmima koji ne zahtijevaju prethodnu obradu podataka koju nazivamo predprocesiranje. Opisuju se Dijkstrin algoritam i A^* algoritam te njihove dvosmjerne verzije. U novije vrijeme ključ ubrzanja pronalaska najkraćeg puta u algoritmima leži u fazi predprocesiranja. Algoritam provodeći potrebne postupke prikupi informacije koje koristi pri ubrzanju faze traženja puta. Detaljno opisujemo dva takva algoritma: *Contraction Hierarchies* i *Landmark* A^*.

Contraction Hierarchies algoritam na prikladan način određuje hijerarhiju vrhova, te u početni graf dodaje nove bridove zvane *prečaci*. Pronalazak najkraćeg puta provodi se dvosmjernim Dijkstrinim pretraživanjem u modificiranom grafu gdje obje pretrage pretražuju samo bridove koje vode do vrhova veće hijerarhije. *Landmark* A^* algoritam temelji se na A^* pretraživanju uz heurističku funkciju procjene udaljenosti dobivenu fazom predprocesiranja. Uvode se vrhovi *orijentiri* za koje je poznata udaljenost *od* i *do* svih preostalih vrhova. Procjena udaljenosti dobije se po uzoru na nejednakost trokuta.

U zadnjem dijelu provodimo usporedbu promatranih algoritama te iznosimo neke implementacijske pojedinosti. Rezultati su dobiveni na podacima stvarnih cestovnih prometnica. *Contraction Hierarchies* nadmašuje *Landmark* A^* algoritam u brzini pronalaska najkraćeg puta, no uz veće vrijeme predprocesiranja. Ovisno o dozvoljenom vremenu i raspoloživoj memoriji kod pripremne faze, bira se prikladniji algoritam. Oba algoritma su značajno brža u usporedbi s klasičnim Dijkstrinim pretraživanjem.

Summary

The problem of finding the shortest path in a graph is widely applicable in the real world. Many things are mathematically modeled using graphs, which creates the need for better and faster algorithms to solve this problem. Examples are various types of roads and traffic situations, Internet networks, special types of databases, and various others. The first part of this thesis is dedicated to classical algorithms that do not require preprocessing. The Dijkstra's algorithm and the A^* algorithm, and their two-way versions are described. More recently, the key to accelerating finding the shortest path in algorithms lies in the preprocessing phase. By performing the necessary procedures, the algorithm collects the information it uses to speed up the path search phase. We describe two such algorithms in detail: *Contraction Hierarchies* and *Landmark* A^*.

The Contraction Hierarchies algorithm appropriately determines the hierarchy of vertices and adds new edges called shortcuts to the initial graph. The shortest path finding is performed by a two-way Dijkstra's search in a modified graph where both searches search only the edges leading to the tops of the larger hierarchy. The Landmark A^* algorithm is based on A^* search with a heuristic distance estimation function obtained by the preprocessing phase. We introduce the landmark nodes for which the distance *from* and *to* all remaining nodes is known. The estimate of the distance is obtained based on the triangle inequality.

In the last part, we compare the observed algorithms and present some implementation details. The results were obtained on the basis of actual road network data. Contraction Hierarchies outperforms the Landmark A^* algorithm in the speed of finding the shortest path, but with a longer preprocessing time. Depending on the time allowed and the available memory during the preparation phase, a more suitable algorithm is chosen. Both algorithms are significantly faster compared to the classic Dijkstra's search.

Životopis

Rođen sam u Zagrebu 20. 9. 1997. godine. Pohađao sam Osnovnu školu Dragutina Domjanića u zagrebačkim Gajnicama. U višim razredima osnovne škole počela me intenzivnije zanimati matematika što je rezultiralo plasmanom na državno natjecanje u sedmom i osmom razredu.

Nakon osnovnoškolskog obrazovanja upisujem XV. gimnaziju u Zagrebu te nastavljam sudjelovati na natjecanjima iz matematike i srodnih područja. Među rezultatima s državnih natjecanja iz matematike ističu se treće mjesto u prvom te druga nagrada u četvrtom razredu. Od međunarodnih natjecanja najistaknutija postignuća su mi pohvala na Međunarodnoj matematičkoj olimpijadi (IMO) održanoj u Hong Kongu 2016. godine, te brončana medalja u pojedinačnoj i zlatna u ekipnoj kategoriji na Srednjoeuropskoj matematičkoj olimpijadi (MEMO) održanoj u Kopru 2015. godine.

Na jesen 2016. godine upisujem preddiplomski sveučilišni studij Matematika na Matematičkom odjeku Prirodoslovno-matematičkog fakulteta u Zagrebu. Obrazovanje nastavljam na istom fakultetu 2019. godine upisom diplomskog studija Računarstvo i matematika. Za vrijeme studiranja bio sam demonstrator iz kolegija Matematička analiza 1 i 2 te iz Diskretne matematike. Sudjelujući na studentskim natjecanjima ostvario sam jednu srebrnu i dvije brončane medalje na International Mathematical Competition (IMC) natjecanju u Blagoevgradu, uz dvije pohvale na međunarodnom matematičkom natjecanju Vojtech Jarnik, održanom u Ostravi. Bio sam aktivan član udruge Mladi nadareni matematičari „Marin Getaldić", gdje sam uz mentorstvo učenicima XV. gimnazije, pripremao učenike za razna srednjoškolska matematička natjecanja. Tijekom ljeta 2020. godine obavljao sam stručnu praksu u računalnoj firmi HashCode iz Zagreba.

Tijekom cijelog razdoblja osnovne i srednje škole trenirao sam nogomet u zagrebačkom klubu NP Ponikve. Nakon upisa na fakultet i prestanka igranja u klubu nastavio sam se aktivno baviti rekreacijskim sportom.