Dokumentacija za algoritam najkraćeg puta sa preprekama

Seminarski rad u okviru kursa
Geometrijski algoritmi
Matematički fakultet

Dara Milojković, 1037/2020
milojkovic.dara@gmail.com

14. januar 2021

Sažetak

Ovaj rad predstavlja dve implementacije nalaženja najkraćeg puta između prepreka. Zadatak je da se na osnovu dve zadate tačke, početka i kraja, pronađe put koji je najkraći i pri tom zaobilazi prepreke, odnosno poligone u Euklidskoj ravni. Naivni algoritam i optimalni algoritam su implementirani uz pomoć DCEL strukture.

Sadržaj

1 Uvod

Zadatak ovog rada je da predstavi algoritam za nalaženje najkraćeg puta pritom izbegavajući prepreke, odnosno poligone. Urađena su dva algoritma, naivni i optimalni algoritam. Algoritmi imaju slične pristupe problemu, razlika je u implementaciji i načinu čuvanja podataka.

Velika atrakcija u skorije vreme je izrada robota koji mogu da oponašaju ljudsko razmišljanje i ljudske pokrete. Robotika teži da kretanje robora bude što sličnije ljudskom kretanju, tj. da robot zna kako prepreke da izbegne i da stigne do određenog cilja. Kako bi se kretanje isplaniralo, robot mora da ima predznanje o njegovoj okolini. Ako se robotu programira predznanje tada je veoma lako naći rešenje problema. Međutim, u velikom broju slučajeva to predznanje neće biti programirano već robot mora da "uči". Učenje se ogleda u upotrebi senzora i proceni razmera. Ako se robot nalazi u fabrici njegovi senzori mogu da "videšamo vidljive delove mašina i zidova, tada robot može da putem svojih senzora izračuna najkraći put do tog mesta otvarajući time mogućnost novog vidljivog prostora. Model kretanja robota je kompleksne prirode međutim, model kretanja se može prebaciti na 2D prostor, uz par promena.

Neka postoji skup P čiji su elementi prosti poligoni koji međusobno nemaju dodirnih tačaka. Zadatak je da se pronađe najkraći put koji spaja dve tačke, $p_{pocetak}$ i p_{kraj}. Pri tome, mora da važi da dobijeni put ne seče ni jedan poligon iz skupa P. Poligoni skupa P predstavljaju prepreke koje robot mora da zaobiđe, dok tačke predstavljaju početak i kraj kretanja robota.

2 Traženje najkraćeg puta

Postavlja se pitanje kako naći najkraći put. Jedna ideja je da se skup poligona i tačaka preslika u poseban graf i izračuna najkraći put između dva čvora uz pomoć Dijkstrinog algoritma ili neke druge pogodne heuristike. Nije svaki graf pogodan za računanje najkraćeg puta, ali možemo biti sigurni da će put nađen algoritmom biti najkraći. Ovo tvrđenje je potpomognuto posebnom lemom:

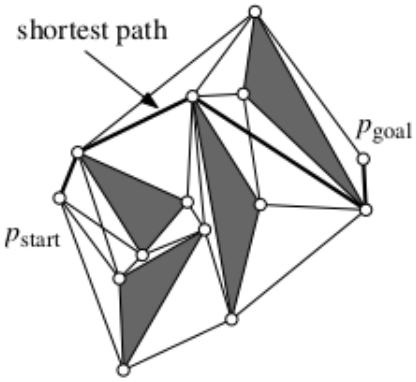
Lema 2.1 *Svaki najkraći put između $p_{pocetak}$ i p_{kraj} između skupa poligona P je poligonijalni put čije su unutrašnji čvorovi temena skupa P.*

Lema se dokazuje uvođenjem kontradikcije. Pretpostavlja se da najkraći put τ nije poligonijalna linija. Pošto su prepreke poligoni, znači da postoji tačka p na putu τ koja se nalazi u unutrašnjosti slobodnog prostora i ne postoji duž koja sadrži p a pripada τ. Pošto se nalazi u prostoru moguće je pravljenje diska sa centrom u p. Međutim, taj disk seče put te će postojati tetiva koja će spajati dve tačke preseka. Tetiva je lokalno kraći put od puta τ, što je kontradikcija, τ mora da bude najkraći put kako globalno tako i lokalno.

Lema je karakterizacija najkraćeg puta, to nam omogućava pravljenje grafa sa takvim putem. Taj graf je *graf vidljivosti*. Njegovi čvorovi su temena iz skupa P, grana postoji ako su dva temena jedna drugom vidljiva. Takva grana se naziva *vidljiva grana*. Kako bi se kompletirala putanja, u skup P se dodaju i $p_{pocetak}$ i p_{kraj}.

Posledica leme (ta) je sledeća:

Posledica 2.1 *Najkraći put koji spaja $p_{pocetak}$ i p_{kraj} i koji nema preseka sa skupom odvojenih poligona P, se sastoji od grana vidljivog grafa $G(P^*)$ gde je $P^* := P \cup \{p_{pocetak}, p_{kraj}\}$.*



Slika 1: Graf vidljivosti i najkraći put

3 Graf vidljivosti

Graf vidljivosti je graf obostrano vidljivih lokacija, obično se koristi za probleme nalaženja puta između prepreka. Svaki čvor je lokacija a grana predstavlja prohodan put između dve lokacije. Problem najkraćeg puta se svodi na efikasno pronalaženje grafa vidljivosti. Nakon nalaženja grafa, u zavisnosti od veličine grafa, može se upotrebiti ili algoritam $A*$ ili Dijkstrin algoritam Skup P, koji sadrži razdvojene poligone koji predstavljaju prepreke u ravni, sadrži ukupno n grana. Dodavanje tačaka $p_{pocetak}$ i p_{kraj} neće uticati na krajnji rezultat. Kako bi se napravio graf vidljivosti, mora da bude poznato koji čvorovi, tj. temena mogu da se vide. Jedan način je da se provere svi parovi temena i ako ne seče neki poligon, taj par temena se ubacuje u graf. Vreme koje je potrebno za nalaženje grafa je $O(n^3)$.

Vreme nalaženja grafa može da se skrati ako ne razmatranjem parove čvorova, već samo čvor i njihove neposredne susede. Grana $\overline{pw}$ će pripadati grafu $G(P^*)$ ukoliko ne seče nijednu liniju poligona iz skupa S, samim tim čvor w je vidljiv iz p i obrnuto. Štaviše moguće je iskoristiti informacije jednog čvora za drugi. Pretpostavimo da postoji poluprava ρ sa početkom u tački p i neka prolazi w. Ako je w nije vidljivo onda poluprava seče neki poligon iz skupa S. Međutim, čvorovi se ne vide ako pripadaju različitim stranama istog poligona. To se rešava proverom da li je čvor u unutrašnjosti poligona. Kako se ne bi ponavljale grane, svaka pređena grana tj. koja ima presek sa ρ, se smešta u balansirano stablo pretrage τ. Listovi τ čuvaju redosled presečenih grana po redu koji su presečeni. Način biranja grana koje se smeštaju u τ je cikličnog oblika, zato ρ treba da bude rotirajuća prava. Prava rotira u smeru kazaljke na satu i skuplja sve one grane koje preseče. Ovako konstruisan algoritam u najgorem slučaju iznosi $O(n^2 \log(n))$.

4 Implementacija

Implementiran je naivni algoritam i optimalniji algoritam za nalaženje najkraćeg puta uz pomoć literature [1]. Optimalni algoritam nije sasvim tačno implementiran i ima svojih mana koje će biti iznete u dokumentaciji.

4.1 Strukture

Struktura koja je ovde iskorišćena je izmenjena DCEL (Double Connected Edge List). Ova struktura je poznata i kao half-edge struktura podataka. Ova struktura daje efikasnu manipulaciju topoloških informacija. Osnove ove strukture su upotrebljene i u ovoj implementaciji, konkretno definicije polu-ivica i čvorova koji čine poligone. Za čuvanje čvorova tj. temena grafa iskorišćena je klasa *Cvor* koja sadrži:

- identifikator poligona kojem čvor pripada
- pokazivač na susednu ivicu poligona
- koordinate na mapi
- tip čvora

Za čuvanje ivica poligona, a kasnije i grane grafa je iskorišćena klasa *PoluIvica* koja sadrži:

- čvor iz kog polazi jedna poluivica
- pokazivač na poluivicu koja predstavlja istu ivicu ali u suprotnom smeru
- pokazivač na sledeću poluivicu
- pokazivač na prethodnu poluivicu

Ovako definisane strukture lako prave lanac ivica povezane čvorovima. Niz takvih ivica i čvorova se čuva u klasi *Poligon*. DCEL struktura čuva i lice poligona međutim ovde to nije potrebno.

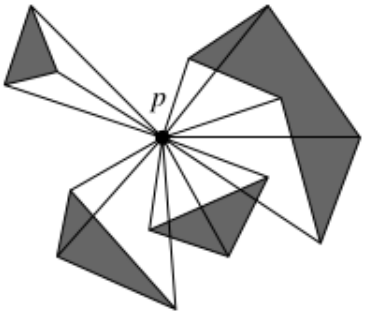
4.2 Naivni algoritam

Ideja ovog algoritma je da se ispita da li neka nova ivica seče postojeće ivice poligona. Prolazi se kroz skup čvorova i proverava se odnos svaka dva čvora i poligona. Za svakih n ivica se proverava odnos sa ostalim $n–1$ ivica u $O(n^2)$ vremenu. Onda se proverava odnos sa ivicama poligona što ukupno daje $O(n^3)$. Pseudo kod je dat u primeru koda 5.

```
1000 NajkraciPut(P*){
       Neka je G = (V, E) graf vidljivosti.
1002   Dodaj sve cvorove iz P* u skup V.
       E = {}
1004   Za svaki cvor v iz P*:
         Za svaki cvor w iz P*:
1006        Za sve poligone P:
                ako g=(v, w) ne sece P onda dodaj g u E
1008
       dodeli svakom cvoru tezinu
1010   Dijkstrin algoritam(G)

1012 }
```

Listing 1: Pseudo kod naivnog algoritma



Slika 2: Presek polu-ivice i poligona

Uslovi kada polu-ivica seče poligon su:

- polu-ivica je deo poligona, ovaj slučaj se prihvata jer tačka može da putuje duž zida prepreke
- polu-ivica nije deo poligona i seče neku ivicu, ovaj slučaj se ne prihvata jer ne sme da prelazi unutrašnjost prepreke. Ne računa se ukoliko je presek polu-ivice i neke druge polu-ivice teme tih polu-ivica, jer će ti čvorovi da budu u grafu.

4.3 Optimalni algoritam

Ideja ovog algoritma je da se na pametan način nađu vidljivi čvorovi. Umesto provere da li neka ivica seče sve poligone uvodi se koncept rotirajuće prave. Rotirajuća prava pokuplja sve ivice sa kojima ima presek. Time se dobija podatak da li postoje neke prepreke između. Ivice se smeštaju u balansirano stablo, razlog za to što su operacije pretrage i ubacivanja svedene na $O(\log(n))$.

```
1000 NajkraciPut(P, p_pocetak, p_kraj){
       Nadji graf Vidljivosti za P*
1002   dodeli svakom cvoru tezinu
       Dijkstrin algoritam(G)
1004 }
```

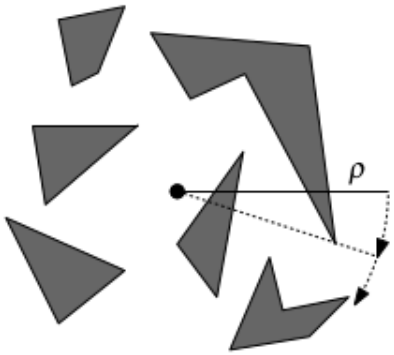
Listing 2: Pseudo kod optimalnog algoritma

U najgorem slučaju ovaj algoritam se izvršava u vremenu $O(n^2 \log(n))$. A najbitniji deo je kako se pravi graf vidljivosti. Graf vidljivosti se računa na sledeći način:

```
1000 Graf_Vidljivost(P*){
       Inicijalizuje se graf G = (V, E) gde je V skup svih cvorova
         poligona, E = {}
1002   Za sve cvorove v
         nadji vidljive cvorove za v i staviti u skup W
1004       za svaki par cvorova (v, w) napraviti granu u E, gde je w iz
           skupa W
       vrati G;
1006 }
```

Listing 3: Pseudo kod graf vidljivosti

Funkcija nađi vidljive čvorove je srž algoritma jer se u toj funkciji pravi rotirajuća prava i proveravaju preseci sa poligonima. Ažuriranje se vrši ubacivanjem i izbacivanjem određenih ivica.



Slika 3: Poluprava sa početkom u tački p

```
Vidljivi_Cvorovi(p, P*){
  Sortiraj sva temena poligona u smeru kazaljke na casovniku po
     uglu koji pravi poluprava iz p i sadrzi teme sa pozitivnim
     delom x-ose
  Pravi se poluprava sa pocetkom u p i koja je paralelna sa x-osom
  Pokupe se sve ivice poligona koje se seku sa tom polupravom i
     cuvaju se u balansiranom stablu
  W = {}
  od i=0 do n
    ako je cvor vidljiv staviti cvor u W
    u stablo se ubacuje ona susedna ivica koja se nalazi ispod
     poluprave p i koja sadrzi w
    iz stabla se izbacuje ona susedna ivica koja se nalazi iznad
     poluprave p i koja sadrzi w
  vrati W
}
```

Listing 4: Pseudo kod za pronalazenje vidljivih cvorova

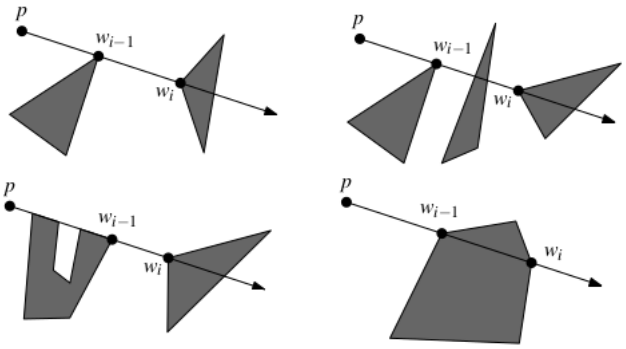
U ovoj funkciji mogu da nastanu problemi. Iz funkcije se vidi da rotirajuća poluprava prolazi kroz sve čvorove, gde se i vidi njena rotirajuća priroda. Međutim, paralelna poluprava možda ne pokupi nijednu ivicu što pravi problem za sledeću iteraciju. Takođe nije jasno šta se radi kada su obe ivice koje koje su susedne posmatranom čvoru ispod te poluprave ili obrnuto. Jer u tom slučaju u sledećoj iteraciji obe prave mogu biti presečene ukoliko se jedna ubaci a druga izbaci jedan slučaj preseka ostaje nepokriven.

```
Da_li_je_vidjliv_cvor(wi){
Ukoliko duz pwi sece poligon u kome se nalazi wi
  vrati netacno
Ukoliko je i=0 ili pwi ne sadrzi wi_1:
  potrazi u stablu pretrage ivicu e koja je najbliza pwi
  Ukoliko postoji e i sece pwi
    onda  vrati netacno
  u suprotnom vrati tacno
Ukoliko wi_1 nije vidljivo
  onda  vrati netacno
U suprotnom
  Potrazi se ivica e u stablu pretrage koja sece duz wiw_1
  Ukoliko postoji takva ivica
    vrati netacno
  U suprotnom vrati tacno
}
```

Listing 5: Pseudo kod za ispitivanje vidljivih cvorova

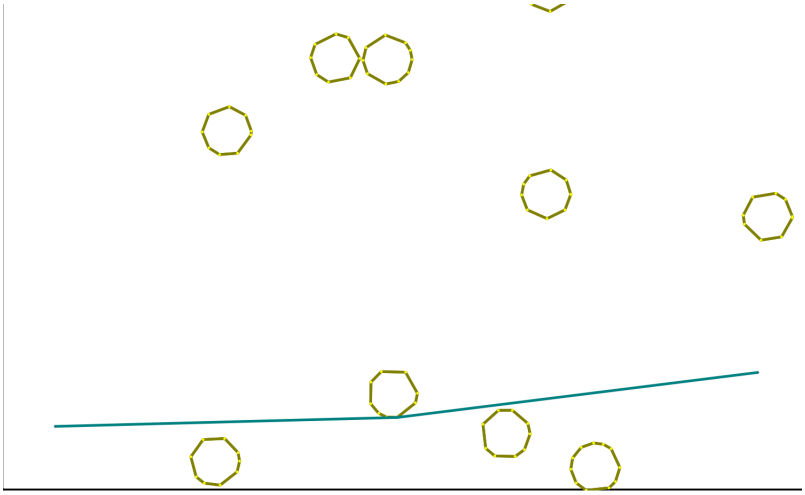
Ova funckija je poprilično jednostavna funkcija i pokriva sve slučajeve koje mogu da se dogode. Ukoliko duž $\tilde{p}w_i$ seče poligon u kome se nalazi tačka w_i tada taj čvor nije vidljiv iz p. Ako to nije slučaj onda se proverava da li je prethodno pređeni čvor na duži ili ne. Ukoliko nije onda se proverava presek duži i najbliže ivice iz stabla, ako postoji presek onda taj čvor nije vidljiv iz p ako ne postoji presek tada je čvor vidljiv.

Ukoliko se w_{i-1} nalazi na duži $\tilde{p}w_i$ i nije vidljiva iz p onda ni w_i nije vidljivo. Ukoliko je w_{i-1} vidljivo tada se pronalazi ivica koja koja seče tu ivicu, ukoliko postoji onda između postoji neka prepreka pa je w_i nevidljiva.

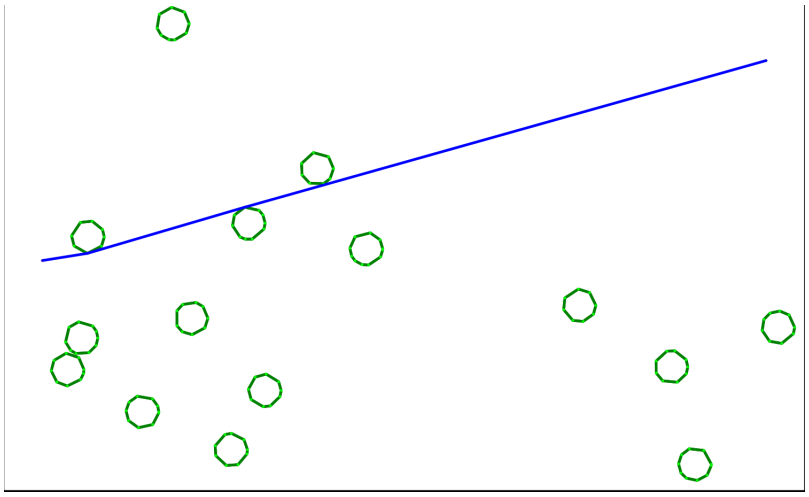


Slika 4: Mogući preseci

Kada se pronađe graf vidljivosti potrebno je da se pronađe najkraći put od $p_pocetak$ do p_kraj. To je moguće dijkstrinim algoritmom. Dijkstrin algoritam potencijalno može da dovede do vremena $O(n^2)$, međutim to neće da utiče na ukupno vreme izvršavanja algoritma.



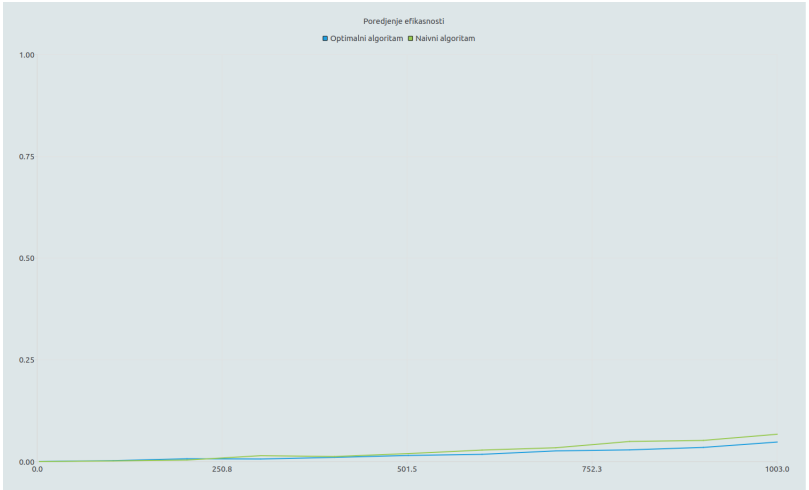
Slika 5: Najkraći put – Naivni algoritam



Slika 6: Najkraći put – Optimalni algoritam

5 Zaključak

Optimalni algoritam se pokazao kao bolji algoritam. Ipak nije optimalniji od naivnog algoritma jer se podaci čuvaju povoljnom strukturom. Međutim algoritam ima i svoje mane. Ako se ne implementira pogodnom strukturom neke preseke neće ni da registruje a koji bi možda bili bitni za proveru u iteracijama koje slede. Postoje i bolji algoritmi koji su navedeni u literaturi [1].



Slika 7: Provera efikasnosti

Literatura

[1] M.T. Berg, M.J. Kreveld, and M.H. Overmars. *Computational Geometry: Algorithms and Applications*. 01 2008.