

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Pronalazak najkraeg puta algoritmom A^*

Marko Lazari

Voditelj: *Doc. dr. sc. Marko upi*

Zagreb, svibanj 2019.

SADRŽAJ

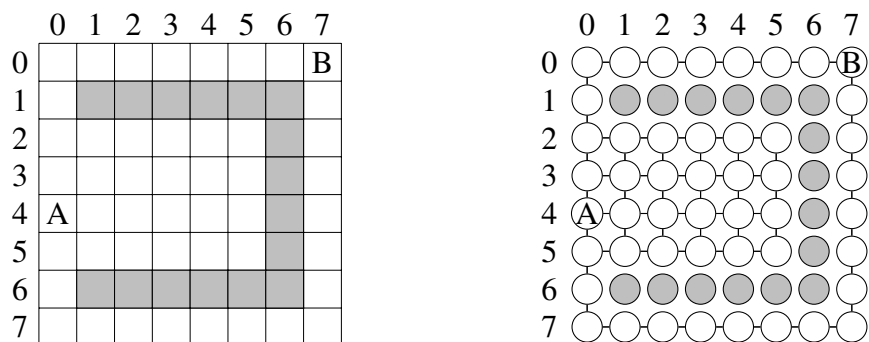
1. Uvod	1
1.1. Definicija problema	2
1.2. Kriteriji uspoređivanja algoritama	2
2. Naivni algoritmi	3
2.1. Pretraživanje u širinu	3
2.2. Pretraživanje s jednolikom cijenom	5
3. Informirani algoritmi	7
3.1. Heuristička funkcija	7
3.2. Pretraživanje "najbolji prvi"	7
3.3. Algoritam A*	8
4. Zaključak	10
5. Literatura	11
6. Sažetak	12

1. Uvod

Velik broj problema može se modelirati grafom u kojem vrhovi predstavljaju stanja, a bridovi prijelaze između tih stanja, takav graf se naziva prostor stanja (engl. *state space*). Rješavanje problema se onda svodi na pretraživanje prostora stanja, odnosno pronalazak najkraćeg puta između početnog stanja i stanja koje predstavlja rješenje problema.

Zbog svoje iroke primjenjivosti, osmišljeni su razni algoritmi za efikasno rješavanje tog problema. U ovom seminaru razmatrat će se prednosti i mane različitih algoritama za pronalazak najkraćeg puta između dva vrha u grafu, počevši od jednostavnijih naivnih algoritama, do složenijih za implementaciju, ali efikasnijih informiranih algoritama s naglaskom na algoritam A* i njegovu primjenu na pronalazak najkraćeg puta u cjelobrojnoj rešetki.

U svrhu jednostavne vizualizacije rada algoritama, koristit će se cjelobrojna rešetka u kojoj sivi kvadrati predstavljaju neprolazna polja, dok bijeli predstavljaju polja kroz koja se može proći. Može se kretati se u 4 osnovna smjera. "A" predstavlja početno stanje (vrh), a "B" završno.



Slika 1.1: Primjer jednostavne cjelobrojne rešetke i njezinog prostora stanja.

1.1. Definicija problema

Opisani problem definiran je s pet komponenti [3]:

1. Početno stanje (engl. *initial state*)
2. Moguće akcije (engl. *actions*) – opis mogućih akcija u nekom stanju.
3. Model prijelaza (engl. *transition model*) opis to svaka akcija zna.
4. Provjera riješenosti (engl. *goal test*) – provjera je li neko stanje riješeno.
5. Funkcija troška prijelaza između stanja (engl. *step cost*)

Za problem cjelobrojne rešetke, početno stanje je stanje sa slovom "A". Moguće akcije su: gore, dolje, lijevo i desno. Model prijelaza je intuitivan: gore povećava y koordinatu za jedan, dolje ju smanjuje, a lijevo i desno povećavaju, odnosno smanjuju x koordinatu. Provjera riješenosti provjerava sadrži li stanje slovo "B". Trošak neposrednog prijelaza između stanja je uvijek jednak 1. Definicija tog problema implementirana je razredom `RectangularGrid`.

1.2. Kriteriji usporeivanja algoritama

Algoritme za rješavanje navedenog problema možemo usporeivati po 4 kriterija [3]:

1. Potpunost – potpuni algoritam će uvijek pronaći rješenje, ako rješenje postoji.
2. Optimalnost – optimalni algoritam će uvijek pronaći optimalno rješenje, ako rješenje postoji.
3. Vremenska složenost – koliko dugo traje izvođenje algoritma u ovisnosti o veličinom ulaza.
4. Memorijska složenost – koliko memorije koristi algoritam pri izvođenju u ovisnosti o veličinom ulaza.

Pri analizi vremenske i memorijske složenosti korisne su sljedeće oznake: b faktor grananja (najveći broj sljedbenika iz nekog stanja), d dubina rješenja s najmanjom dubinom i m najveća duljina bilo kojeg puta u prostoru stanja.

2. Naivni algoritmi

Naivni algoritmi (engl. *uninformed algorithms*) nemaju dodatne informacije o stanjima, pa smatraju da svaki prijelaz ima jednaku vjerojatnost dovoenja do najkraeg puta. Njihove prednosti su jednostavna implementacija i generalnost, a mana im je velika memorijska i vremenska sloenost.

2.1. Pretraivanje u irinu

Pretraivanje u irinu (engl. *breadth-first search*) je naivni algoritam za pronalazak najkraeg puta u grafu s uniformnim teinama bridova. Algoritam je potpun i optimalan, a njegova vremenska i memorijska sloenost je $O(b^d)$. [3]

Za pohranu vrhova koje treba obraditi koristi red (engl. *queue*), dok za pohranu ve pronaenih vrhova koristi skup (engl. *set*). U svakom koraku iri se u svim moguim smjerovima, to dovodi do velike memorijske i vremenske sloenosti. Implementiran je metodom `bfs` u razredu `SearchAlgorithms`.

Na slici 2.1 je prikazano izvoenje algoritma. Crvenom bojom su oznaena polja koje je algoritam obradio, ali nisu dovela do najkraeg puta, dok je zelenom oznaen najkrai put. Brojevi u poljima oznaavaju udaljenost od poetnog polja. Pretraga u irinu je obradila 41 polja, dok bi optimalno rjeenje obradilo samo 8.

	0	1	2	3	4	5	6	7
0	4	5	6					
1	3	4	5	6				
2	2	3	4	5	6			
3	1	2	3	4	5	6		
4	A	1	2	3	4	5	6	B
5	1	2	3	4	5	6		
6	2	3	4	5	6			
7	3	4	5	6				

	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4	A	1	2	3	4	5	6	B
5								
6								
7								

Slika 2.1: Usporedba pretraivanja u irinu i optimalnog rjeenja.

```

fronta ← FIFO red vrhova s jednim elementom: (pocetnoStanje, null, 0);
pronadeni ← skup s jednim elementom: pocetnoStanje;
dok fronta nije prazna čini
    trenutni ← prvi element iz fronte;
    ako trenutni.stanje je rjesenje onda
        vrati trenutni;
    inače
        prijelazi ← prijelazi iz trenutni.stanje;
        za svaki prijelaz iz prijelazi čini
            ako prijelaz.stanje nije u pronadeni onda
                dodaj prijelaz.stanje u pronadeni;
                dodaj (prijelaz.stanje, trenutni, trenutni.cijena + prijelaz.cijena)
                    u frontu;
            kraj
        kraj
    kraj
kraj
vrati null;

```

Algoritam 1: Pseudokod pretraivanja u irinu.

2.2. Pretraivanje s jednolikom cijenom

Pretraivanje s jednolikom cijenom (engl. *uniform-cost search*) je naivni algoritam za pronalazak najkraeg puta u grafu s pozitivnim teinama bridova. Algoritam je potpun i optimalan, a njegova vremenska i memorijska sloenost je $O(b^{1+\lceil C^*/\epsilon \rceil})$ gdje C^* predstavlja cijenu optimalnog rjeenja, a ϵ minimalnu teinu svih bridova. [3]

Za pohranu vrhova koristi prioritetni red (engl. *priority queue*), te hash tablicu radi efikasne provjere je li pronaao bolji put do nekog vrha koji se ve nalazi u prioritetnom redu. Implementiran je metodom `ucs` u razredu `SearchAlgorithms`.

Prioritetni red koristi evaluacijsku funkciju (engl. *evaluation function*) za odreivanje prioriteta, pri emu manji broj ima vei prioritet. Evaluacijska funkcija se oznaava s $f(n)$. Kod pretraivanja s jednolikom cijenom evaluacijska funkcija je jednaka cijeni puta od poetnog stanja do stanja n . Oznaka za cijenu puta od poetnog stanja do stanja n je $g(n)$, pa za pretraivanje s jednolikom cijenom vrijedi $f(n) = g(n)$.

Za grafove s uniformnim teinama bridova, pretraivanje s jednolikom cijenom obradi strogo vie vrhova od pretraivanja u irinu jer provjerava sve vrhove na istoj dubini kao i rjeenje kako bi provjerio postoji li bolji put do rjeenja.

	0	1	2	3	4	5	6	7
0	4	5	6					
1	3	4	5	6				
2	2	3	4	5	6			
3	1	2	3	4	5	6		
4	A	1	2	3	4	5	6	B
5	1	2	3	4	5	6		
6	2	3	4	5	6			
7	3	4	5	6				

	0	1	2	3	4	5	6	7
0	4	5	6	7				
1	3	4	5	6	7			
2	2	3	4	5	6	7		
3	1	2	3	4	5	6	7	
4	A	1	2	3	4	5	6	B
5	1	2	3	4	5	6	7	
6	2	3	4	5	6	7		
7	3	4	5	6	7			

Slika 2.2: Usporedba pretraivanja u irinu i pretraivanja s jednolikom cijenom.

fronta \leftarrow prioritetni red vrhova koristeći $f(\text{stanje})$ za određivanje prioriteta s
 jednim elementom: (pocetnoStanje, null, 0);
 obradeni \leftarrow prazan skup stanja;

dok fronta nije prazna **čini**

 trenutni \leftarrow prvi element iz fronte;

ako trenutni.stanje je rjesenje **onda**

 vрати trenutni;

inače

 dodaj trenutni.stanje u obradeni;

 prijelazi \leftarrow prijelazi iz trenutni.stanje;

za svaki prijelaz iz prijelazi **čini**

ako prijelaz.stanje nije u obradeni **onda**

ako prijelaz.stanje je u fronti **onda**

 stari \leftarrow element iz fronte koji ima stanje = prijelaz.stanje;

ako stari.cijena > trenutni.cijena + prijelaz.cijena **onda**

 izbrisi stari iz fronte;

 dodaj (prijelaz.stanje, trenutni, trenutni.cijena +
 prijelaz.cijena) u frontu;

kraj

inače

 dodaj (prijelaz.stanje, trenutni, trenutni.cijena +
 prijelaz.cijena) u frontu;

kraj

kraj

kraj

kraj

kraj

vрати null;

Algoritam 2: Pseudokod pretraivanja s jednolikom cijenom.

3. Informirani algoritmi

Informirani algoritmi (engl. *informed algorithms*) koriste dodatne informacije o problemu kako bi smanjili prostor stanja kojeg trebaju pretražiti. Te informacije se najčešće ugrađuju u heuristiku funkciju (engl. *heuristic function*) koja se označava s $h(n)$, a predstavlja aproksimaciju najmanje cijene prijelaza od stanja n do stanja koje predstavlja rješenje. Njihova prednost je manja memorijska i vremenska složenost, a mana im je specijalizacija za pojedini problem.

3.1. Heuristika funkcija

Heuristika $h(n)$ je optimistična ili dopustiva ako i samo ako nikada ne precjenjuje cijenu puta od stanja n do cilja. Odnosno za svako stanje n vrijedi da je $h(n)$ manje ili jednako od cijene optimalnog puta od stanja n do cilja. [1]

Heuristika $h(n)$ je konzistentna ili monotona ako i samo ako za svako stanje n i za svaki njegov sljedbenik n' vrijedi: $h(n) \leq c(n, n') + h(n')$ gdje $c(n, n')$ predstavlja trošak prijelaza iz stanja n u stanje n' . [3]

Stanja u cjelobrojnoj rešetci označena su s dva cijela broja x i y , pa se heuristika funkcija može zapisati kao $h(x, y)$. Neka je početno stanje $A(x_A, y_A)$, a završno stanje $B(x_B, y_B)$. Onda se heuristika funkcija može definirati kao Manhattan udaljenost između trenutnog stanja i završnog, odnosno $h(x, y) = |x - x_B| + |y - y_B|$. Takva heuristika funkcija predstavlja najmanji broj prijelaza potrebnih za prijelaz iz trenutnog stanja u završno stanje, pa je nužno optimistična, a također je konzistentna.

3.2. Pretraživanje "najbolji prvi"

Pretraživanje "najbolji prvi" (engl. *greedy best-first-search*) je pohlepan informirani algoritam koji nije optimalan, a potpun je jedino ako koristimo listu posjećenih stanja. Vremenska i prostorna složenost mu je $O(b^m)$. [3] [1]

Algoritam je ekvivalentan pretraivanju s jednolikom cijenom, osim to za evaluacijsku funkciju koristi heuristiku funkciju $f(n) = h(n)$. To znači da e prvo evaluirati stanja koja su blia rjeenju, no taj lokalni optimum nee nuno dovesti do optimalnog rjeenja, kao to je prikazano na slici 3.1.

	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4	A	1	2	3	4	5	6	B
5								
6								
7								

	0	1	2	3	4	5	6	7
0	14	15	16	17	18	19	20	B
1	13							
2	12	11	10	9	8	7		
3		8	5	6	9	6		
4	A	1	2	3	4	5		
5				4	7	6		
6								
7								

Slika 3.1: Rezultat izvoenja pretraivanja "najbolji prvi".

3.3. Algoritam A*

Algoritam A* je ekvivalentan pretraivanju s jednolikom cijenom, osim to za evaluacijsku funkciju koristi sumu cijene puta od poetnog stanja do stanja n i heuristike funkcije $f(n) = g(n) + h(n)$, odnosno aproksimaciju najboljeg puta kroz stanje n . A* je uvijek potpun, a optimalan je pod uvjetom da je heuristika $h(n)$ konzistentna. Vremenska i memorijska sloenost mu je $O(\min(b^{d+1}, b|S|))$ gdje $|S|$ predstavlja ukupan broj stanja. [3] [1]

	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4	A	1	2	3	4	5	6	B
5								
6								
7								

	0	1	2	3	4	5	6	7
0	4	5	6	7	8	9	10	B
1	3							
2	2	3	4	5	6	7		
3	1	2	3	4	5			
4	A	1	2	3				
5								
6								
7								

Slika 3.2: Rezultat izvoenja algoritma A*.

Takva evaluacijska funkcija kombinira prednosti pretraivanja s jednolikom cijenom i pretraivanja "najbolji prvi". Zbog heuristike funkcije algoritam preferira stanja koja se pribliavaju cilju, a zbog funkcije cijene algoritam je optimalan (uz konzistentnu

heuristiku) jer preferira stanja koja imaju najjeftiniji put od početnog stanja do cilja kroz to stanje.

Postoje brojne varijante algoritma A*. Algoritam A* s iterativnim povećanjem dubine (engl. *iterative-deepening A**, *IDA**) je ekvivalentan pretraivanju u dubinu s iterativnim povećanjem dubine, no umjesto dubine, IDA* koristi evaluacijsku funkciju $f(n)$. Prednost algoritma IDA* je manja memorijska složenost, a posebno je praktičan za grafove s uniformnim težinama bridova. [3]

Pojednostavljeni algoritam A* s ograničenom memorijom (engl. *simplified memory-bound A**, *SMA**) se ponavlja kao običan A* sve dok ne napuni ograničenu memoriju. Kada ju napuni, algoritam izbacuje list s najvećom vrijednosti evaluacijske funkcije. Vrijednost izbačenog vrha se zapisuje u njegovog roditelja, kako bi znao je li potrebno ponovno obrađivati taj vrh. SMA* je optimalan ako optimalno rješenje stane u ograničenu memoriju. [3]

4. Zaključak

Algoritam A^* je jednostavan informirani algoritam koji efikasno pretrauje prostor stanja uz dobro definiranu heuristiku funkciju. Njegova prednost je smanjenje prostora stanja koje treba pretraiti pomoću heuristike funkcije, no njegova mana je velika memorijska složenost, pa je običan A^* primjenjiv prostoru stanja. Za istraživanje prostora stanja s velikim brojem stanja, prikladniji su algoritmi IDA^* i SMA^* .

5. Literatura

- [1] Bojana Dalbelo Bai i Jan najder. 3. heuristiko pretraivanje, 2018.
URL https://www.fer.unizg.hr/_download/repository/UI-3-HeuristickoPretrazivanje.pdf.
- [2] Amit Patel. Introduction to a*. URL <https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html>.
- [3] Stuart J. Russell i Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002. ISBN 0137903952.

6. Saetak

Brojni problemi se mogu prikazati skupom stanja i prijelazima izmeu njih, odnosno prostorom stanja. Rjeavanje problema se svodi na istraivanje prostora stanja, te pronalazak najkraeg puta iz nekog poetnog stanja, do stanja koje predstavlja rjeenje.

Postoje razliiti algoritmi za istraivanje prostora stanja. Naivni algoritmi istrauju prostor stanja naivno, bez dodatnih informacija o problemu, to dovodi do velike vremenske i memorijske sloenosti.

Informirani algoritmi koriste dodatne informacije kako bi smanjili prostor stanja koji trebaju istraititi. Te dodatne informacije se ugrauju u heuristiku funkciju koja predstavlja aproksimaciju najmanje cijene prijelaza od stanja n do stanja koje predstavlja rjeenje. Prednost informiranih algoritama je manja vremenska i memorijska sloenost zato to ne moraju istraititi cijeli prostor stanja.

Algoritam A^* je informirani algoritam za pretraivanje prostora stanja koji kombinira heuristiku funkciju i trenutnu cijenu kako bi smanjio prostor stanja kojeg treba istraititi, ali svejedno naao optimalno rjeenje. Postoje varijante algoritma A^* koje nastoje smanjiti memorijsku sloenost kao to su IDA^* i SMA^* .

U ovom seminaru opisane su obje kategorije algoritama, i naivni i informirani, prikazane su njihove razlike, kako u implementaciji, tako i u memorijskoj i vremenskoj sloenosti. Osim same analize algoritama, vizualizacijama je poblie objanjen nain rada, kao i sluajevi u kojima opisani algoritmi ne daju dobre rezultate.