SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Pronalazak najkraćeg puta algoritmom A*

Marko Lazarić
Voditelj: *Doc. dr. sc. Marko Čupić*

Zagreb, svibanj 2019.

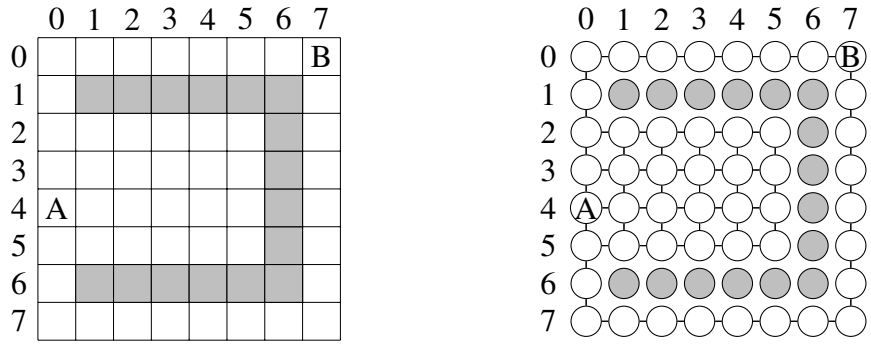
SADRŽAJ

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 široke 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 rešetci.

U svrhu jednostavne vizualizacije rada algoritama, koristit će se cjelobrojna rešetka u kojoj sivi kvadratići predstavljaju neprolazna polja, dok bijeli predstavljaju polja kroz koja se može proći. Moguće je 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 što svaka akcija znači.
4. Provjera riješenosti (engl. *goal test*) – provjera je li neko stanje rješenje.
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 uspoređivanja algoritama

Algoritme za rješavanje navedenog problema možemo uspoređivati 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 dovođenja do najkraćeg puta. Njihove prednosti su jednostavna implementacija i generalnost, a mana im je velika memorijska i vremenska složenost.

2.1. Pretraživanje u širinu

Pretraživanje u širinu (engl. *breadth-first search*) je naivni algoritam za pronalazak najkraćeg puta u grafu s uniformnim težinama bridova. Algoritam je potpun i optimalan, a njegova vremenska i memorijska složenost je $O(b^d)$. [3]

Za pohranu vrhova koje treba obraditi koristi red (engl. *queue*), dok za pohranu već pronađenih vrhova koristi skup (engl. *set*). U svakom koraku širi se u svim mogućim smjerovima, što dovodi do velike memorijske i vremenske složenosti. Implementiran je metodom `bfs` u razredu `SearchAlgorithms`.

Na slici 2.1 je prikazano izvođenje algoritma. Crvenom bojom su označena polja koje je algoritam obradio, ali nisu dovela do najkraćeg puta, dok je zelenom označen najkraći put. Brojevi u poljima označavaju udaljenost od početnog polja. Pretraga u širinu je obradila 41 polja, dok bi optimalno rješenje 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 pretraživanja u širinu i optimalnog rješenja.

```
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 pretraživanja u širinu.

2.2. Pretraživanje s jednolikom cijenom

Pretraživanje s jednolikom cijenom (engl. *uniform-cost search*) je naivni algoritam za pronalazak najkraćeg puta u grafu s pozitivnim težinama bridova. Algoritam je potpun i optimalan, a njegova vremenska i memorijska složenost je $O\left(b^{1+\lfloor C^*/\epsilon \rfloor}\right)$ gdje C^* predstavlja cijenu optimalnog rješenja, a ϵ minimalnu težinu svih bridova. [3]

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

Prioritetni red koristi evaluacijsku funkciju (engl. *evaluation function*) za određivanje prioriteta, pri čemu manji broj ima veći prioritet. Evaluacijska funkcija se označava s $f(n)$. Kod pretraživanja s jednolikom cijenom evaluacijska funkcija je jednaka cijeni puta od početnog stanja do stanja n. Oznaka za cijenu puta od početnog stanja do stanja n je $g(n)$, pa za pretraživanje s jednolikom cijenom vrijedi $f(n) = g(n)$.

Za grafove s uniformnim težinama bridova, pretraživanje s jednolikom cijenom obradi strogo više vrhova od pretraživanja u širinu jer provjerava sve vrhove na istoj dubini kao i rješenje kako bi provjerio postoji li bolji put do rješenja.

	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 pretraživanja u širinu i pretraživanja s jednolikom cijenom.

```
fronta ← prioritetni red vrhova koristeci f(stanje) za odredivanje prioriteta s
  jednim elementom: (pocetnoStanje, null, 0);
obradeni ← prazan skup stanja;
dok fronta nije prazna čini
    trenutni ← prvi element iz fronte;
    ako trenutni.stanje je rjesenje onda
        vrati trenutni;
    inače
        dodaj trenutni.stanje u obradeni;
        prijelazi ← 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 ← 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
vrati null;
```

Algoritam 2: Pseudokod pretraživanja 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 heurističku 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. Heuristička 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]

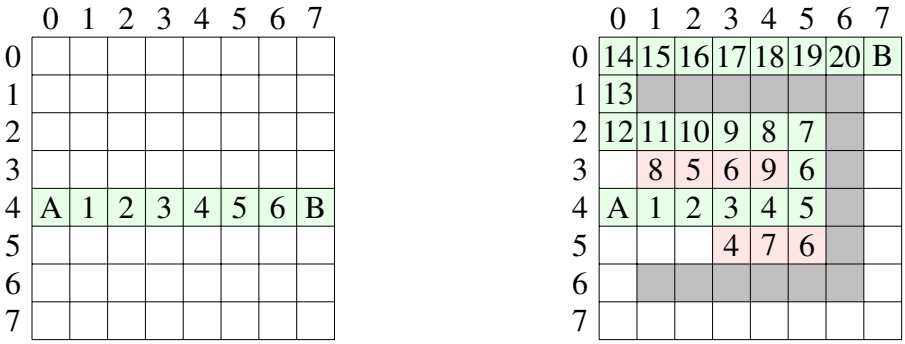
Heuristika $h(n)$ je konzistentna ili monotona ako i samo ako za svako stanje n i za svaki njegov sljedbenik n' vrijedi: $h(n) \leq c(n, n') + h(n')$ gdje $c(n, n')$ predstavlja trošak prijelaza iz stanja n u stanje n'. [3]

Stanja u cjelobrojnoj rešetci označena su s dva cijela broja x i y, pa se heuristička 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 heuristička funkcija može definirati kao Manhattan udaljenost između trenutnog stanja i završnog, odnosno $h(x, y) = |x - x_B| + |y - y_B|$. Takva heuristička 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 informiran algoritam koji nije optimalan, a potpun je jedino ako koristimo listu posjećenih stanja. Vremenska i prostorna složenost mu je $O(b^m)$. [3] [1]

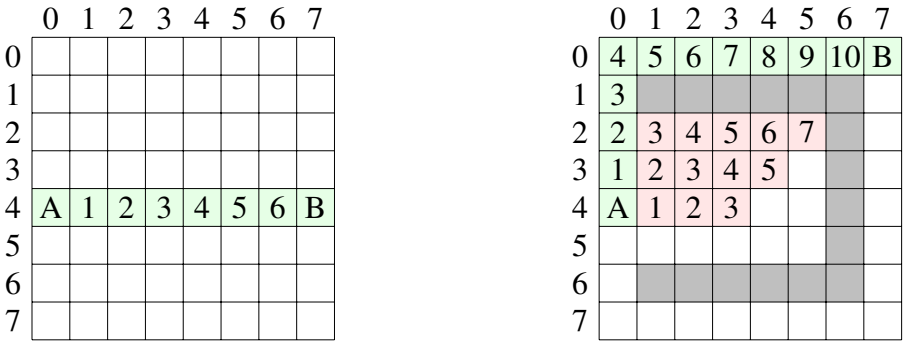
Algoritam je ekvivalentan pretraživanju s jednolikom cijenom, osim što za evaluacijsku funkciju koristi heurističku funkciju $f(n) = h(n)$. To znači da će prvo evaluirati stanja koja su bliža rješenju, no taj lokalni optimum neće nužno dovesti do optimalnog rješenja, kao što je prikazano na slici 3.1.



Slika 3.1: Rezultat izvođenja pretraživanja "najbolji prvi".

3.3. Algoritam A*

Algoritam A* je ekvivalentan pretraživanju s jednolikom cijenom, osim što za evaluacijsku funkciju koristi sumu cijene puta od početnog stanja do stanja n i heurističke 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 složenost mu je $O\left(\min(b^{d+1}, b|S|)\right)$ gdje $|S|$ predstavlja ukupan broj stanja. [3] [1]



Slika 3.2: Rezultat izvođenja algoritma A*.

Takva evaluacijska funkcija kombinira prednosti pretraživanja s jednolikom cijenom i pretraživanja "najbolji prvi". Zbog heurističke funkcije algoritam preferira stanja koja se približavaju cilju, a zbog funkcije cijene algoritam je optimalan (uz konzis-

tentnu 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 pretraživanju u dubinu s iterativnim povećavanjem 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 ponaša 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 pretražuje prostor stanja uz dobro definiranu heurističku funkciju. Njegova prednost je smanjenje prostora stanja koje treba pretražiti pomoću heurističke funkcije, no njegova mana je velika memorijska složenost, pa je običan A* primjenjiv prostore stanja. Za istraživanje prostora stanja s velikim brojem stanja, prikladniji su algoritmi IDA* i SMA*.

5. Literatura

[1] Bojana Dalbelo Bašić i Jan Šnajder. 3. heurističko pretraživanje, 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. Sažetak

Brojni problemi se mogu prikazati skupom stanja i prijelazima između njih, odnosno prostorom stanja. Rješavanje problema se svodi na istraživanje prostora stanja, te pronalazak najkraćeg puta iz nekog početnog stanja, do stanja koje predstavlja rješenje.

Postoje različiti algoritmi za istraživanje prostora stanja. Naivni algoritmi istražuju prostor stanja naivno, bez dodatnih informacija o problemu, što dovodi do velike vremenske i memorijske složenosti.

Informirani algoritmi koriste dodatne informacije kako bi smanjili prostor stanja koji trebaju istražiti. Te dodatne informacije se ugrađuju u heurističku funkciju koja predstavlja aproksimaciju najmanje cijene prijelaza od stanja n do stanja koje predstavlja rješenje. Prednost informiranih algoritama je manja vremenska i memorijska složenost zato što ne moraju istražiti cijeli prostor stanja.

Algoritam A* je informirani algoritam za pretraživanje prostora stanja koji kombinira heurističku funkciju i trenutnu cijenu kako bi smanjio prostor stanja kojeg treba istražiti, ali svejedno našao optimalno rješenje. Postoje varijante algoritma A* koje nastoje smanjiti memorijsku složenost kao što 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 složenosti. Osim same analize algoritama, vizualizacijama je pobliže objašnjen način rada, kao i slučajevi u kojima opisani algoritmi ne daju dobre rezultate.