Sveučilište u Splitu
Prirodoslovno-matematički fakultet

UVOD U UMJETNU INTELIGENCIJU – PROJEKT

8-SLAGALICA (THE 8-PUZZLE)

STUDENTICA:

Lucija Bročić

Split, 20. veljače 2018.

SADRŽAJ

1. UVOD

Što je 8-slagalica (8-puzzle)?

8-slagalica (The 8-puzzle) je 3x3 slagalica koja se sastoji od nasumično poslaganih 8 pločica numeriranih brojevima od 1 do 8 te jednog praznog mjesta. Slagalica postoji i u drugim veličinama, od kojih je poznatija 15-slagalica (4x4). Slagalice se još zovu u 9-slagalica, odnosno 16-slagalica, ako se računa ukupan broj mjesta, a ne pločica. Cilj slagalice je posložiti pločice po redu, od 1 do 8 s praznim mjestom na kraju, pomičući ih koristeći prazno mjesto.

n-slagalica je klasičan problem za modeliranje algoritama koji koristi heuristiku. Često korištene heuristike uključuju brojanje pločica koje nisu na ciljnom mjestu (Hamming distance / misplaced tiles) i računanje Manhattan udaljenosti između trenutnog mjesta pločice i njenog ciljnog mjesta. Obje heuristike su optimalne (eng. admissible) tj. nikad ne precjenjuju broj preostalih pokreta, što osigurava optimalnost za neke algoritme, primjerice A*.

Rješivost

Nije svaki raspored pločica u početnom stanju rješiv. Općenito, rješivost slagalice sa brojem redaka i stupaca N (tj. n- slagalica gdje je n= N – 1) su računa po ovim jednostavnim pravilima:

1. Ako je N neparan broj, onda je slagalica rješiva ako je broj inverzija početnog stanja paran.
2. Ako je N paran, slagalica je rješiva ako
    a. Prazno mjesto je u parnom retku brojeći od kraja (drugi od kraja, četvrti od kraja itd.)
    b. Prazno mjesto je u neparnom stupcu brojeći od kraja (zadnji, treći od kraja, peti od kraja, itd.) i broj inverzija je paran.
3. Za sve druge slučajeve, slagalica nije rješiva.

Što je inverzija?
Ako stanje slagalice zapišemo u obliku jednodimenzionalne liste umjesto u obliku dvodimenzionalne N x N matrice, par pločica (a, b) tvori inverziju ako se a u nizu pojavljuje prije b, a vrijedi a > b (to je zapravo inverzija u permutaciji). Primjerice, 13245 ima jednu inverziju, (3,2), dok 42351 ima inverzije (4,2), (4,3), (4,1), (2,1), (5,1), (3,1).

Radi jednostavnosti, za testiranje programa je uzeto 10 rješivih početnih stanja (pronađenih na stranici https://www.cs.cmu.edu/~adamchik/15-121/labs/HW-7%20Slide%20Puzzle/lab.html)

2. OPIS PROBLEMA I SIMULACIJE

Problem: Napravite program koji rješava zadanu simulaciju jednim od odabranih algoritama. Usporedite algoritme po parametrima u BehaviourSpace-u.

Definicija problema preko 4 komponente:

1. Početno stanje – zadano rješivo početno stanje
2. Skup dozvoljenih akcija – pomicanje jedne od pločica oko praznog mjesta na prazno mjesto. Prostor stanja je organiziran u strukturu stabla, čvorove predstavljaju liste u koje su spremljeni: lista s trenutnim rasporedom pločica i lista poteza koje je trebalo napraviti da se dođe do tog stanja, te ovisno o algoritmu još i: 1 ili 0, ovisno je li čvor posjećen ili ne i broj koji predstavlja vrijednost heurističke funkcije
3. Provjera cilja – postoji, a ciljno stanje je u obliku liste zapisano s [1 2 3 4 5 6 7 8 0]
4. Cijena puta - u nekim algoritmima broj koraka, a u nekima heuristička funkcija

Izgled simulacije

Simulacija se sastoji od svijeta veličine 3 x 3 (rub je u gornjem lijevom kutu i max-pxcor je 2, a min-pycor - 2). Takav svijet je odabran zbog jednostavnosti prikaza iz matrice.

Na simulaciji se nalaze tipka **setup**, chooser za odabrati vrstu-slagalice gdje brojevi 1 – 10 označavaju pojedino početno stanje, od jednostavnijih prema težima; chooser za odabir algoritma i tipka korak koja izvršava odabrani algoritam; monitori za put do ciljnog stanja i za broj poteza potrebnih za doći do ciljnog stanja te slider za odabir dubine kod nekih algoritama. Kada je put pronađen, tipkom **prikaži rješenje** se vizualno prikažu potezi korišteni pri rješavanju slagalice.

setup - služi za postavljanje svijeta tj. generira odabrano početno stanje (moguć odabir među 10 već određenih rješivih početnih stanja) s procedurom **generiraj-rjesiva-pocetna-stanja** u obliku matrice 3x3; prikazuje početno stanje na simulaciji procedurom stvori-stanja (stvorili smo novi breed – tiles, i odgovarajuće početno stanje očitava pri čemu su tiles 0 – 9 oznake za pločice s brojevima 1 - 8); u listu svih stanja se dodaje početno stanje i to u obliku [lista] [potezi] gdje [lista] predstavlja trenutno stanje u slagalici zapisano u obliku liste, a ne matrice, a [potezi] kojima smo došli do tog rasporeda (oznake su u koju stranu se 0 pomaknula; L lijevo, R desno, U gore, D dolje), a pozivom find-moves se u listu oko-nule spremaju brojevi koji se nalaze oko praznog mjesta tj. elementi koje je moguće pomaknuti.

vrsta-slagalice – brojevima su označena sljedeća početna stanja:
(lista u kojoj je prikazano stanja / broj poteza)

1. 123405786 / 2
2. 123745086 / 4

3. 123480765 / 5
4. 413726580 / 8
5. 162530478 / 9
6. 512630478 / 11
7. 126350478 / 13
8. 356148072 / 16
9. 436871052 / 18
10. 302651478 / 21

algoritam – odabir između (ograničene) pretrage po dubini, pretrage po širini, pohlepne pretrage, A* algoritma i IDA* algoritma

3. PRETRAŽIVANJE

Kako je skup stanja prikazan preko strukture stabla, pretraživanje je pronalaženje puta kroz to stablo, tj. kretanje od čvora do čvora nakon širenja.
Jedna od mogućih podjela postupaka pretraživanja je:

1. Slijepo pretraživanje
2. Usmjereno pretraživanje

1. Slijepo pretraživanje

U ovoj vrsti pretraživanja, jedino raspoložive informacije su početno stanje, dozvoljene akcije nad stanjima i test na rješenje problema. Algoritmi koji pripadaju metodama slijepog pretraživanja, a implementirani su u ovom projektu su pretraživanje u širinu i ograničeno pretraživanje u dubinu.

Pretraživanje u širinu (eng. breadth-first search, BFS) je pretraživanje u kojem se otvaraju svi čvorovi na određenoj dubini (razini) u stablu prije odlaska na sljedeću razinu. U projektu je implementirano na sljedeći način:

Za svako stanje na određenoj razini (koja se prati varijablom broj-koraka) pronađe moguće poteze i brojeve koji su oko praznog mjesta sprema u listu oko-nule (u proceduri find-moves) i zatim u proceduri sirina-korak za svaki broj iz oko-nule u listi stanja zamijeni 0 i taj broj i takvu promijenjenu listu doda na listu stanja.

Algoritam je potpun jer pronađe rješenje (algoritam se i izvršava dok rješenje nije pronađeno) i optimalan, jer će ga pronaći u najmanjem broju koraka, no općenito je najveći nedostatak eksponencijalna kompleksnost, međutim u ovom problemu, najviše čvorova kojih može biti na jednoj razini je 4 (kada je prazno mjesto u sredini i kada je to stanje početno).

Pretraživanje u dubinu (eng. depth-first search, DFS) obavlja pretraživanje otvarajući čvorove zadnje proširenog čvora dok ne dođe do rješenja ili dok ne dođe do

zadane granice dubine. Zadnje otvoreni čvorovi se spremaju na početak tako da budu prvi ispitani. Zbog prirode problema (vrlo duge grane), uzeta je ograničena pretraga po dubini. U projektu je implementirano na sljedeći način:

Uzima prvi element sa liste stanja i ako nije na dubini koja je stavljena za granicu, napravi dubina-korak, a ako je na dubini koja je stavljen za granicu, stavi da je zadnji čvor ispitan, ali ga nemoj širiti dalje. Procedura dubina-korak prima trenutno stanje, pronalazimo moguće sljedeće poteze za to stanje, prolazimo kroz ta stanja na siti način kao i kod širine, samo sada sljedeća stanja stavljamo na početak liste stanja s oznakom posjećenosti 0, a stanje koje smo širili stavljamo na kraj s oznakom posjećenosti 1.

Potpunost algoritma ovisi o odabranoj dubini, ali općenito nije potpun, jer ne mora pronaći rješenje, i općenito nije optimalan jer ne nalazi najbolje rješenje.

2. Usmjereno pretraživanje

Postupci usmjerenog pretraživanja se dijele ovisno o tome koji tip informacija se koristi kod određivanja sljedećeg koraka, pa tako ih možemo podijeliti na heuristička pretraživanja i optimalna pretraživanja.

Heuristički algoritmi koriste heurističku evaluacijsku funkciju tj. funkciju koja daje informacije koje nisu eksplicitno dokazive, a to su informacije o prirodi stanja, o cijeni prijelaza iz jednog u drugo stanje itd. Heuristika je tehnika evaluacije koja može pomoći, ali ne osigurava pronalaženje cilja, ali pomaže u smanjenju dimenzije problema od eksponencijalne na polinomijalnu. U ovom projektu je implementiran jedan heuristički algoritam – pohlepni algoritam.

Pohlepno pretraživanje (eng. greedy search) je poput metode pretraživanja u dubinu s tim da se širi onaj čvor koji ima najmanju vrijednost heurističke funkcije. Vrijednost se računa procedurom racunaj-heuristiku, a heuristička funkcija je jedna od popularnijih za ovaj problem – za stanje izračunamo Manhattan udaljenost svakog broja od mjesta gdje bi trebao biti i kao vrijednost heurističke funkcije uzmemo zbroj svih Manhattan udaljenosti za brojeve u tom stanju. U projektu je algoritam implementiran na sljedeći način:

Prvo početnom stanju dodamo oznaku posjećenosti (0). To se napravi samo jednom, jer kad postavimo stanje ne stavimo i oznaku posjećenosti.

Uzmemo prvi element sa liste stanja (zadnji otvoreni čvor) i onda prolazimo kroz sva stanja i tražimo neposjećena stanja koja su na istoj dubini kao i prvi element. Među tim stanjima pronađemo koja je najmanja vrijednost heurističke funkcije. Zatim opet prolazimo kroz sva stanja i sva ona stanja koja su na istoj dubini kao i prvi element, a imaju minimalnu vrijednost heurističke funkcije stavljamo u listu trenutna-stanja. Zatim provjeravamo ima li neposjećenih čvorova; ako nema, zaustavljamo pretragu. Promatramo prvi element s liste trenutna-stanja (zadnji otvorni čvor s najmanjom vrijednosti heurističke funkcije na toj razini). Provjeravamo je li promatrana razina jednaka granici dubine. Ako nije, obavljamo pohlepna-korak, a ako je, promatrani element označavamo kao posjećenog i stavljamo na kraj liste stanja.

Procedura pohlepna-korak funkcionira isto kao i dubina-korak, samo u stanje dodajemo još i vrijednost heurističke funkcije.

Od postupaka optimalnog pretraživanja u projektu su implementirani A* algoritam i IDA* algoritam.

A* algoritam se često koristi pri pronalasku puteva i pretrazi stabla zbog svoje točnosti i izvedbe. Pri pronalasku puta koristi i heurističku funkciju koja je obično zbroj funkcije koja računa cijenu već obavljenog puta od početnog stanja do trenutnog čvora te funkcije koja procjenjuje cijenu puta od trenutnog čvora do cilja. U projektu je algoritam implementiran na sljedeći način:

Koristimo open i closed liste za otvorene, odnosno zatvorene čvorove. Najprije za prvo stanje izračunamo vrijednost heurističke funkcije (zbroj racunaj-heuristiku i racunaj-udaljenost, koja računa koliko je poteza napravljeno do trenutnog stanja) i pridružimo prvom stanju. Nadalje, sve dok ima otvorenih čvorova sortiramo listu otvorenih čvorova, od najmanje do najveće vrijednosti funkcije; uzimamo prvi otvoreni čvor te provjeravamo je li ciljno stanje. Zatim mičemo trenutni čvor s liste open i stavljamo na listu closed te obavljamo astar-korak. S procedurom a-star-korak punimo listu neighbour s čvorovima koje možemo dobiti iz trenutnog čvora. Zatim prolazimo kroz sve elemente liste neighbour i ako nije ni na open ni na closed listi, dodajemo ih na početak open liste te trenutno stanje stavljamo na kraj liste stanja.

Heuristička funkcija je optimalna (admissible) jer, kako se pločice ne mogu pomicati dijagonalno, ne precjenjuje procjenu puta do ciljnog stanja, pa je i algoritam optimalan.

IDA* algoritam (eng. iterative deepening A*) bi se najbolje mogao opisati kao iterativna pretraga po dubini, međutim granica dubine se računa preko vrijednosti heurističke funkcije. U svakoj iteraciji izvršava pretragu po dubini, i prestaje u grani kada vrijednost heurističke funkcije prekorači vrijednost granice. Granica se prvo postavi na vrijednost heurističke funkcije od početnog stanja, a u svakoj iteraciji se granica za sljedeću iteraciju postavlja na najmanju vrijednost heurističke funkcije čvorova čija je vrijednost prekoračila granicu. U projektu je algoritam implementiran na sljedeći način:

Pri setup-u granicu postavljamo na vrijednost heuristike funkcije početnog stanja (racunaj-udaljenost + racunaj-heuristiku), a u ida-pretraga sve dok na listi stanja ima neposjećenih čvorova, uzimamo prvi element koji je zadnji otvoreni čvor i ako njegova vrijednost heuristike nije strogo veća od granice, obavljamo ida-dubina-korak (što je praktički dubina-korak, ali stanjima dodajemo i heuristiku), a ako je veća onda sa stanja mičemo taj element te njegovu vrijednost dodajemo na listu udaljenosti (gdje se spremaju vrijednosti čvorova koji su prekoračili granicu). Ako lista udaljenosti nije prazna, za granicu postavljamo najmanji element iz te liste, a ako je prazna, brišemo sa liste stanja sve osim početnog stanja.

Kao i kod A*, IDA* je optimalan ako je heuristička funkcija optimalna, a kako je funkcija ista kao i u A*, IDA* je optimalan.

3. ANALIZA ALGORITAMA

U Behaviour Space-u su provedena 2 eksperimenta: prvi prolazi kroz sve algoritme i slagalice 1 – 9 (slagalica 10 je izostavljena zbog vremenske kompleksnosti) s fiksnom dubinom 15 (14 ako gledamo broj poteza) te drugi koji prolazi kroz sve algoritme osim pretrage po širini (jer bi se isti algoritam ponavljao za 10 dubina, pa je velike vremenske kompleksnosti), slagalice 1 – 9 i dubine 11 -20 (po broju poteza, 10 - 19). Orange izvještaj je priložen u obliku pdf datoteke, gdje se prva tri grafa odnose na prvi eksperiment, a druga tri grafa na drugi eksperiment.

Na prvom grafu možemo primijetiti da A* i IDA* algoritmi općenito trebaju manje vremena od pohlepne pretrage i pretrage po dubini, dok pretraga po širini ima tako male vrijednosti zbog toga što su se tickovi računali nakon svake razine.
Još možemo primijetiti da što je kompleksnija slagalica (što je više poteza potrebno za njeno rješavanje) to je vremenska kompleksnost veća.

Na drugom grafu možemo primijetiti da A* i IDA* algoritmi ne zauzimaju mnogo memorije kao što je npr. slučaj s pretragom po širini za kompleksnije slučajeve, do pohlepna pretraga i pretraga po dubini, ovisno o složenosti slagalice, ne zauzimaju previše memorije.

Na trećem grafu možemo primijetiti da su algoritmi A*, IDA* i pretraga po širini uvijek pronašli rješenje, dok pohlepna i pretraga po dubini nisu, međutim, kako je dubina fiksirana, nije ni očekivano.

Na grafovima drugog eksperimenta možemo izvući iste zaključke, ali na posljednjem grafu možemo primijetiti da je za slagalice manjeg broja uglavnom pronađen put pohlepnim algoritmom i pretragom po dubini, dok za slagalice s većim brojem (kojima je i broj poteza veći, pa je time potrebno i povećati dubinu pri izvršavanju algoritama) .

5. IZVORI

- Materijali s predavanja i vježbi
- **https://www.cs.cmu.edu/~adamchik/15-121/labs/HW-7%20Slide%20Puzzle/lab.html**
- **https://en.wikipedia.org/wiki/15_puzzle**
- **http://www.d.umn.edu/~jrichar4/8puz.html**
- **https://en.wikipedia.org/wiki/Iterative_deepening_A***
- **https://en.wikipedia.org/wiki/A*_search_algorithm**