UNIVERZITETE CRNE GORE

Elektrotehnički fakultet, Podgorica

Materijal sa četvrtog termina predavanja iz

EKSPERTNIH SISTEMA

Vrednovanje heurističkih funkcija

Lokalne strategije pretraživanja

Prof. dr Vesna Popović-Bugarin

Podgorica, 2013.

4.5. Vrednovanje heurističkih funkcija

Očigledno je da će kvalitet pojedine strategije usmjerenog pretraživanja zavisiti od kvaliteta heurističke funkcije koju ta strategija koristi. Što je heuristička funkcija bolja procjena stvarne udaljenosti do ciljnog čvora, to će manji biti broj čvorova generisanih u toku usmjerenih pretraživanja. Za ocjenu kvaliteta heurističkih funkcija se stoga koristi **broj generisanih čvorova** uz upotrebu posmatrane heurističke funkcije.

Heurističke funkcije se zapravo najčešće porede na osnovu **efektivnog faktora grananja**. Ukoliko se nekom strategijom pretraživanja generiše N čvorova, a dubina rješenja je *d*, efektivni faktor grananja se određuje kao faktor grananja *b** koji bi moralo imati drvo sa uniformnim grananjem da bi na dubini *d* imalo *N* generisanih čvorova (plus korijen):

$$N+1 = 1 + b^* + (b^*)^2 + ... + (b^*)^d$$

Efektivni faktor grananja može varirati od jedne do druge realizacije nekog problema, ali se pokazuje da je ipak konstantan za dovoljno teške probleme [1]. Efektivni faktor grananja *b** se, imajući ovo na umu, može izmjeriti eksperimentalno na malom skupu problema i koristiti za ocjenu upotrijebljene heuristike. Pokazuje se da dobre heuristike imaju *b** približno jednak jedinici, što ukazuje da su dobro usmjerene i da je njima moguće riješiti znatno velike probleme [1].

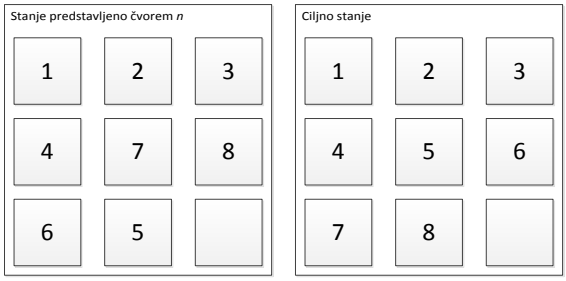
Da bi se dobilo optimalno rješenje korišćenjem A* strategije pretraživanja potrebno je da odabrana heuristika bude optimistična, odnosno da nikada ne precijeni stvarnu udaljenost od rješenja. S druge strane, veća vrijednost za heurističku funkciju daje usmjerenije pretraživanje, što dovodi do generisanja manjeg broja čvorova. Ovo se može koristiti za poređenje dvije heuristike $h_1(n)$ i $h_2(n)$. Ukoliko važi da je $h_2(n) \geq h_1(n)$ za svaki čvor *n* i obije heuristike su optimistične, kaže se da je heuristika ***$h_2(n)$ dominantna u odnosu na $h_1(n)$.*** Heuristika $h_2(n)$ je bolja od heuristike $h_1(n)$ jer strategija pretraživanja A* neće nikada generisati veći broj čvorova korišćenjem $h_2(n)$, nego što bi učinila korišćenjem $h_1(n)$. A* proširuje sve čvorove koji imaju manju vrijednost funkcije procjene od funkcije procjene optimalnog rješenja $f(C^*)$, ukoliko koristi optimističnu heurističku funkciju. Ovo znači da će proširivati sve čvorove koji imaju $h_2(n) < f(C^*) - g(n)$, a znajući da je $h_1(n) \leq h_2(n)$, jasno je da će svaki čvor koji se proširuje korišćenjem heuristike $h_2(n)$ biti sigurno proširen i korišćenjem heuristike $h_1(n)$, dok se korišćenjem heuristike $h_1(n)$ može proširiti i još dodatnih čvorova. Kada su ove dvije heuristike jednake, ukoliko se proširi čvor sa jednom heuristikom sigurno će se proširiti i sa drugom. S druge strane, može se desiti, kada je $h_2(n) > h_1(n)$ da se neki čvor ne proširi heuristikom $h_2(n)$, a da se proširi heuristikom $h_1(n)$.

Jedan od problema koji se može koristiti za ilustraciju poređenja više heuristika jeste slagalica sa 9 polja, Slika 1. To je jedan od najranijih problema za čije rješavanje se koristilo heurističko pretraživanje. Predložen je veliki broj heurističkih funkcija za ovaj problem, ali su najčešće korišćene dvije [1]:

1. $h_1(n)$ – broj pločica koje nijesu na traženom mjestu (u ciljnoj raspodjeli). Ukoliko se posmatra Slika 1 vidi se da, za čvor *n*, četiri pločice nijesu na mjestu na kojem trebale biti u ciljnom stanju. Ova heursitika jeste optimistična

jer se svaka pločica mora pomjeriti barem za jedno mjesto da bi bila postavljena na željenu-ciljnu poziciju.

2. $h_2(n)$ – suma rastojanja pločica do njihovih ciljnih pozicija. Prilikom računanja ovog rastojanja ima se na umu da pločica može biti u jednom koraku pomjerena za jedno mjesto po horizintali, ili po vertikali, pa se rastojanje računa kao zbir horizontalnih i vertikalnih rastojanja. Ovo rastojanje se naziva često Menhetn rastojanjem i za naš slučaj bi bilo $h_2(n)=1+3+2+2=8$. $h_2(n)$ je optimistična jer najbolje što se može desiti je da je u svakom željenom prelazu slobodno polje na koje se treba premjestiti odgovarajuća pločica, odnosno da se samo uračunatim potezima (bez dodatnih poteza za oslobađanje susjednih polja) može doći do cilja.



Slika 1 Slagalica sa 9 polja

Za svaki čvor će biti $h_2(n) \geq h_1(n)$, odnosno da $h_2(n)$ dominira nad $h_1(n)$. Ukoliko postoji dominantna heuristička funkcija i može se relativno jednostavno izračunati, ta se heuritička funkcija treba koristiti.

4.6. Pronalaženje heurističkih funkcija

Glavno pitanje koje se postavlja, kada su u pitanju heurističke funkcije, jeste *da li je moguće obučiti računar da sam definiše heurističke funkcije za neki problem*. Ne manje bitno pitanje, na koje je svakako potrebno prvo odgovoriti, jeste – kako je neko došao na ideju da koristi neku heurističku funkciju za konkretan problem. Ponovo se može poći od ranije spomenutih heurističkih funkcija za slagalicu. Ukoliko se bolje analiziraju, pokazuje se da su ove dvije heurističke funkcije zapravo tačna vrijednost udaljenosti (heurističke funkcije) nekog čvora od rješenja za problem sa *opuštenim ograničenjima*.

Za funkciju $h_1(n)$ se moraju ograničenja opustiti u smislu da se dozvoljava premještanje pločice na bilo koje mjesto, ne samo na susjedno i slobodno polje. Kada je u pitanju $h_2(n)$, ona predstavlja tačnu heurističku funkciju za problem slagalice sa opuštenim ograničenjima, tako što se dozvoljava pomjeranje pločice na sva susjedna mjesta, čak i preko zauzetih polja. Sada se može zaključiti da je cijena optimalnog rješenja od nekog čvora (najjeftinije putanje od nekog čvora do cilja) za problem sa opuštenim ograničenjima sigurno optimistična heuristička funkcija za originalni problem.

Treba voditi računa da se problem sa opuštenim ograničenjima definiše tako da se njegovo optimalno rješenje može odrediti lako, bez uključivanja strategija pretraživanja. Ukoliko je teško riješiti problem sa opuštenim ograničenjima, onda je teško i odrediti njegovu heurističku funkciju, koja će se koristiti za rješavanje originalnog problema. Samim tim se ne dobija mnogo u odnosu na originalni

problem, jer bi se i njegova heuristička funkcija mogla tačno odrediti pokretanjem kompletnog pretraživanja u širinu i pamćenjem cijene puta od pojedinih čvorova do cilja. Dakle, *problem sa opuštenim ograničenjima mora biti takav da se njegovo rješenje lako nalazi i odgovarajuća heuristička funkcija jednostavno računa.*

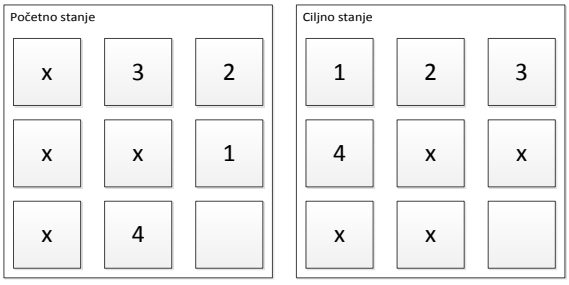
U primjeru sa slagalicom smo imali dvije heurističke funkcije, pri čemu je važilo da je $h_2(n)$ dominantna u odnosu na $h_1(n)$, odnosno da je za svaki čvor veća ili jednaka. Jasno je bilo da je bolje koristiti $h_2(n)$. Međutim, može se desiti da za neki problem imamo više heurističkih funkcija i da nijedna od njih nije dominantna nad ostalima. U tom slučaju se za neki konkretan čvor koristi ona heuristička funkcija koja ima najveću vrijednost:

$$h(n) = \max\{h_1(n), h_2(n), ..., h_k(n)\} \quad (1.1)$$

Na ovaj način se i dalje održava svojstvo optimalnosti, dok se izvlači najbolje iz dostupnih heurističkih funkcija. Odnosno $h(n)$ dominira nad $h_i(n)$, za $i=1,...,k$.

Drugi način za određivanje heurističkih funkcija jeste **rješavanje potproblema**. Naravno, potproblem se bira tako da ga je lako riješiti. Kada je u pitanju slagalica, jedan potproblem bi se svodio na postavljanje pločica sa brojevima 1, 2, 3, 4 na poziciju koju imaju u ciljnom stanju, uz poštovanje svih pravila originalne slagalice. Pri rješavanju ovog potproblema se ne bi vodilo računa o tome na kojim mjestima se nalaze preostale pločice, ali bi se uzimala u obzir cijena njihovog pomjeranja, Slika 2. Jasno je da je dobijena heuristička funkcija optimistična, jer najoptimalnija stvar koja se može desiti je da nakon dovođenja četiri odabrane pločice do ciljnog stanja, imamo i preostale četiri pločice u ciljnom stanju – kada budemo koristili dobijenu heurističku funkciju za rješavanje problema sa osam pločica. U ovom slučaju bi procijenjena heuristička funkcija bila jednaka tačnoj. U svim ostalim slučajevima bi bila manja.

Heuristička funkcija se dobija na osnovu rješavanja potproblema. Heuristička funkcija smještena u bazi je tačna cijena putanje od bilo kojeg stanja četiri pločice do cilja i samim tim tačna heuristička funkcija za svaki čvor u putanji za posmatrani potproblem. Kada se posjeduje ovakva baza, heuristička funkcija za neko kompletno stanje u originalnom problemu (stanje u kojem nam je poznata i bitna pozicija svih osam pločica) dobija se pretraživanjem baze i nalaženjem heurističke funkcije potproblema kod kojeg imamo poklapanje položaja četiri posmatrane pločice sa odgovarajućim pločicama u kompletnom problemu. Baza se popunjava računanjem heurističke funkcije rezonovanjem unazad, odnosno polazi se od cilja i računa se cijena (udaljenost od cilja) svakog šablona-kombinacije četiri pločice na koji se naiđe.



Slika 2 Potproblem slagalice sa osam polja. Cilj je postaviti pločice *1, 2, 3, 4* na željenu poziciju, bez obzira na poziciju preostalih pločica.

Kada je u pitanju izbor četiri pločice koje će se pratiti za formiranje baze, sasvim je proizvoljan. Mogle su se odabrati preostale četiri pločice i dobijene heurističke funkcije kombinovati korišćenjem (1.1). Kombinovanjem heurističkih funkcija na ovaj način dobija se mnogo tačnija heuristička funkcija u odnosu na Menhetn rastojanje, što vodi ka smanjenju generisanih čvorova za faktor 1000 prilikom rješavanja slučajne slagalice sa 16 polja [1].

Bitno je shvatiti da se ne mogu heurističke funkcije prosto sabirati, čak ni kada su dobijene rješavanjem potproblema koji uključuje različite pločice, jer će sigurno u ovim potproblemima biti koraka – stanja koja će se poklapati, jer je nemoguće pomjeriti četiri posmatrane pločice na željeno mjesto, a da se pri tome preostale četiri ne pomjeraju. Prostim sabiranjem se dakle ne bi održala optimističnost dobijene heurističke funkcije. Ono što se može učiniti, jeste koristiti samo ona stanja u kojima su pomjerane samo četiri posmatrane ploče i onda vršiti sabiranje. Korišćenjem ovako generisane baze može se riješiti slučajna slagalica sa 16 polja sa smanjenjem generisanih čvorova za faktor 10000. Računar ovu slagalicu rješava za svega nekoliko milisekundi, [1].

Jasno je da se računar može osposobiti da pravi bazu potproblema i određuje heurističku funkciju na gore opisan način. Ovakav način učenja se naziva *učenjem na osnovu iskustva*. Međutim, korišćenjem formalnog jezika, koji će biti obrađivan na nekim od narednih termina, može se računar naučiti i da automatski generiše probleme sa opuštenim ograničenjima, na osnovu definicije originalnog problema.

Ukoliko bi se problem slagalice predstavio kao:

Pločica se može pomjeriti iz kvadrata A u kvadrat B, ako kvadrat A je susjedan kvadratu B (ima zajedničku horizontalnu ili vertikalnu ivicu) i B je praznina.

Računar bi mogao automatski generisati tri problema sa olakšanim ograničenjima

1. *Pločica se može pomjeriti iz kvadrata A u kvadrat B (bezuslovno).*
2. *Pločica se može pomjeriti iz kvadrata A u kvadrat B ako kvadrat A je susjedan kvadratu B.*
3. *Pločica se može pomjeriti iz kvadrata A u kvadrat B ako kvadrat B je praznina.*

Na ovaj način se dobijaju tri problema sa opuštenim ograničenjima, prvi i drugi odgovaraju problemima na osnovu kojih je dobijena heuristička funkcija $h_1(n)$ i $h_2(n)$, respektivno.

Osim gore navedenih načina automatskog određivanja heurističke funkcije, računar može koristiti i učenje na osnovu iskustva, gdje mu se daju čvorovi sa optimalne putanje i tačna cijena putanje od tog čvora do rješenja. Računar bi trebalo da ***induktivnim učenjem*** iskoristi ove primjere kako bi generisao heurističku funkciju i za druga stanja koja se mogu pojaviti tokom pretraživanja. Jedna od tehnika za ovu vrstu učenja su neuralne mreže.

5. Algoritmi za rješavanje problema iterativnim pretraživanjem - heuristike iz prirode

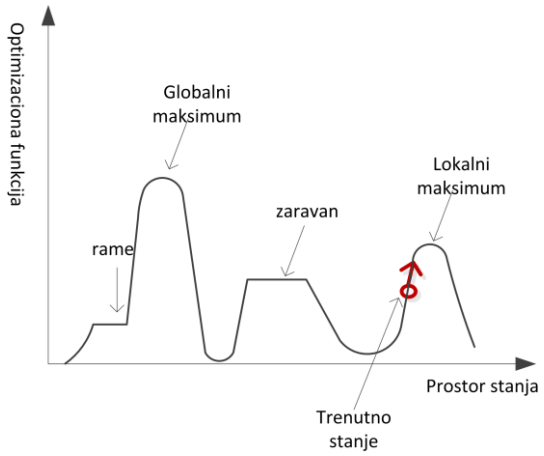
U problemima koji su do sada razmatrani bila je bitna putanja do cilja. U skladu sa tim su i razvijani algoritmi koji sistematski pretražuju prostor stanja, kako bi odredili optimalnu putanju do cilja. Međutim, veliki je broj realnih problema u kojima

nije bitna putanja do cilja, odnosno cilj se ne definiše kroz neko konkretno stanje već kroz niz ograničenja koja moraju biti ispunjena da bi se smatralo da je cilj dostignut. Već smo pomenuli problem osam kraljica u kojem se cilj definiše kao stanje u kojem su sve kraljice raspoređene na šahovskoj tabli, ali tako da se međusobno ne napadaju. Drugi problem ovog tipa je problem putujućeg trgovca. Svi su gradovi međusobno direktno povezani, trgovac ima zadatak da svaki obiđe tačno po jedan put i da to učini uz najmanji pređeni put. Problemi ovog tipa se srijeću i prilikom dizajna integrisanih kola, optimizacije telekomunikacionih mreža, optimizacije potrošnje električne energije i slično.

Kada nije bitna putanja do cilja može se razmatrati klasa algoritama koja ne vodi računa o putanji, odnosno tzv. *algoritmi sa lokalnim pretraživanjem*. Jasno je da je prednost ovih algoritama činjenica da se u memoriji treba držati samo tekuće stanje. Ovdje se ipak stanje može razlikovati od onog što podrazumijevamo pod stanjem u klasičnim algoritmima pretraživanja. Ovdje je stanje zapravo potencijalno rješenje (ukoliko je ono definisano skupom uslova koja moraju biti zadovoljeni), a pretražuju se okolna stanja, ne bi li se popravio kvalitet rješenja.

Ukoliko bismo posmatrali problem putujućeg trgovca, kod klasičnog algoritma pretraživanja stanje bi bilo grad u kojem se nalazi trenutno trgovac i redosljed gradova koje je posjetio, dok će u algoritmima sa lokalnim pretraživanjem stanja biti zapravo jedno moguće rješenje. U slučaju putujućeg trgovca to bi bio određeni redosljed obilaženja svih gradova – odnosno potencijalno rješenje. Sljedeće stanje se obično bira tako da bude neko susjedno (samo se jedan par gradova izmijeni), ali da se tom izmjenom poboljša **cijena rješenja.** Otuda i naziv *iterativno pretraživanje* koji možda i više odgovara, jer neće sva pretraživanja razmatrati samo susjedna stanja. Kada se na ovaj način predstave, vidi se da algoritmi lokalnog pretraživanja mogu vršiti, ne samo nalaženje ciljnog stanja, već i rješavati *optimizacione probleme,* u kojima je zadatak naći stanje koje daje najbolju *funkciju cilja*. Naravno ova funkcija je vezana za konkretan problem. Kod putujućeg trgovca to je dužina pređenog puta.

Lokalno pretraživanje se može vršiti i od nekog stanja ka cilju ali se češće vrši od lošijeg ka boljem rješenju, odnosno ka poboljšanju optimizacione funkcije. Da bi se razumio ovaj način pretraživanja, i razvili odgovarajući algoritmi, potrebno je razmotriti opšti oblik optimizacione funkcije, u zavisnosti od pojedinih stanja (koja najčešće odgovaraju potencijalnim rješenjima), Slika 3. Cilj lokalnog algoritma pretraživanja može biti pronalaženje ili globalnog maksimuma, ili globalnog minimuma, u zavisnosti od načina zadavanja optimizacione funkcije. Slika 3 ilustruje optimizacionu funkciju u slučaju nalaženja globalnog maksimuma. Strelica obilježava način izbora sljedećeg stanja – u smjeru rasta optimizacione funkcije. Jasno je da se nalaženje globalnog minimuma može posmatrati kao nalaženje globalnog maksimuma optimizacione funkcije pomnožene sa minus jedan.



Slika 3 Optimizaciona funkcija u zavisnosti od prostora stanja za jednodimenzioni proctor stanja

Algoritmi lokalnog pretraživanja imaju prednost u tome što ne pamte sve putanje, već samo trenutno rješenje koje se želi poboljšati, pa se najčešće i ne govori i njihovoj memorijskoj zahtijevnosti. Kada se vrši njihovo poređenje ima se na umu definicija optimalnosti i kompletnosti. Algoritam je optimalan ukoliko uvijek nađe stanje koje daje globalni maksimum optimizacione funkcije, Slika 3. Algoritam je kompletan ukoliko uvijek pronađe cilj ukoliko on postoji. Kada su u pitanju algoritmi sa iterativnim pretraživanjem, najčešće se ne insistira na optimalnosti jer se oni primjenjuju u velikim prostorima pretraživanja, kada standardni algoritmi ne mogu naći rješenje, ili im je potrebno previše vremena ili memorije da to učine, dok algoritmi iterativnog pretraživanja u ovakvim prostorima stanja imaju veliku šansu da daju rješenje sa razumnom cijenom.

5.1. Pretraživanje planinarenjem

Pretraživanje planinarenjem se vrši tako što se pođe od nekog stanja (najčešće rješenja) i vrši se iterativno pretraživanje biranjem susjednih stanja (rješenja) koja imaju bolju optimizacionu funkciju. Algoritam se završava kada se dođe do stanja, čija sva susjedna stanja imaju manju vrijednost optimizacione funkcije. U svakom trenutku se čuva samo vrijednost za trenutno stanje i vrijednost njegove optimizacione funkcije, ukoliko stanje predstavlja potencijalno rješenje, odnosno o trenutnoj putanji i otvorenim čvorovima ukoliko se želi primijeniti na pretraživanje optimalne putanje do cilja.

Ovo pretraživanje se često naziva *pohlepnim slijepim pretraživanjem* jer se u svakom trenutku bira susjedno bolje stanje, a ne vodi se računa kuda to stanje dalje vodi. Iako se veliki broj problema može pojaviti kod ovakvog pretraživanja, pokazuje se da ono ipak dosta dobro funkcioniše. Pretraživanje planinarenjem dosta brzo daje velika poboljšanja početnog stanja (najčešće predloženog rješenja). Nedostatak je činjenica da se može zaglaviti kada naiđe na lokalni maksimum, vrat ili zaravan, Slika 3.

- **lokalni maksimum** (brežuljak) optimizacione funkcije će dovesti do prekida daljeg planinarenja i dati neoptimalno rješenje. Naime, naići će se na stanje koje ima veću vrijednost optimizacione funkcije u odnosu na sva susjedna stanja i smatraće se da je ovo rješenje i da nema potrebe za daljim pretraživanjem.

- **vrat** optimizacione funkcije će dovesti do prekida daljeg planinarenja i dati neoptimalno rješenje. Nailazi se na stanje koje ima istu vrijednost optimizacione funkcije kao sva susjedna stanja, ne može se naći pravac poboljšanja optimizacione funkcije i prestaje se sa daljim pretraživanjem.
- **zaravan** optimizacione funkcije će dovesti do prekida daljeg planinarenja i dati neoptimalno rješenje, na isti način kao i vrat.

Glavna razlika između vrata i zaravni jeste što bi omogućavanje odabira bliskih stanja koja nijesu susjedna u slučaju nailaženja na predio gdje je optimizaciona funkcija konstantna doveo do poboljšanja rješenja kod vrata (na kraju bi se stiglo do planine čiji je ovo vrat), dok bi se kod zaravni ušlo u beskonačnu petlju (na kraju bi se došlo do manjih vrijednosti, i kretalo bi se od početka do kraja zaravni u krug). Uobičajeno se uzima alternativno rješenje, odnosno, dozvoljava se konačan broj koraka pri kojima se ne mijenja vrijednost optimizacione funkcije.

Pretraživanje planinarenjem je neoptimalno, ali se rezultati koji se njime dobijaju mogu poboljšati restartovanjem sa slučajno generisanim početnim stanjem-rješenjem i pamćenjem najboljeg rješenja.

Pretraživanje planinarenjem se može ilustrovati kroz primjer osam kraljica gdje se svako stanje sastoji od određenog rasporeda osam kraljica, tako da se po jedna nalazi i jednoj koloni. Susjedna stanja su sva stanja koja se dobiju pomjeranjem samo jedne kraljice na neko od preostalih polja u njenoj koloni, dok ostale zadržavaju svoja mjesta. S obzirom da se jedna kraljica može pomjeriti na sedam preostalih pozicija u njenoj koloni, i da se može odabrati pomjeranje jedne od osam kraljica, broj susjednih stanja je *7x8=56*. Heuristička funkcija koja se nameće je broj parova kraljica koje se međusobno napadaju. Kada se dođe do rješenja, vrijednost heurističke funkcije je *0* (ne postoje dvije kraljice koje se međusobno napadaju). Ova vrijednost je globalni minimum odabrane heurističke funkcije. Za jedno od mogućih početnih stanja, kao u [1], heurističke funkcije za svako moguće susjedno stanje su date tabelom1. Najmanja vrijednost je *h=12* i ima više takvih susjednih stanja. Kada se naiđe na ovakvu situacije, susjedno stanje se odabira slučajno između najboljih sljedbenika. Razmisliti kakvo opuštanje ograničenja je učinjeno prilikom računanja ove heurističke funkcije.

18	12	14	13	13	13	14	13
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♛	13	16	13	16
♛	14	17	15	♛	14	16	16
17	♛	18	18	15	♛	15	♛
18	14	♛	15	15	14	♛	16
14	14	13	17	12	14	12	18

Tabela 1 Početno rješenje problema osam kraljica i heurističke funkcije za naredno stanje kod planinarenja.

Ovakvim načinom rješavanja problema osam kraljica pošavši od slučajnog početnog rasporeda dolazi se do rješenja u *14%* slučajeva, dok se zaglavi u nekom od neoptimalnih rješenja *86%* puta [1]. Međutim, u svakom od ovih slučajeva planinarenje daje brzo rješenje – uobičajeno nakon četiri iteracije ukoliko je rješenje optimalno, odnosno tri kada zaglavi sa neoptimalnim rješenjem [1]. Ovo je veliki napredak kada se ima na umu da se radi sa prostorom od $8^8 \approx 17$ miliona stanja.

Kada se u slučaju nailaska na konstantnu vrijednost optimizacione funkcije dozvoli *100* sporednih poteza za problem osam kraljica, povećava se procenat dobijanja optimalnog rješenja sa *14%* na *84%*!!! Naravno, uspjeh dolazi na uštrb povećane vremenske složenosti. Srednja vrijednost broja koraka nakon kojih se dobija dobro rješenje je *21*, a nakon kojih se dobija loše je 64, [1].

Rečeno je da su iterativna pretraživanja u principu neoptimalna, ali da se dobijeno rješenje može poboljšati restartovanjem sa slučajno generisanim početnim stanjem. Postavlja se pitanje da li se može unaprijed odrediti koliki je broj restartovanja potreban da bi se dobilo optimalno rješenje. Broj restartovanja koji je potreban da bi se dobilo optimalno rješenje nekog problema se može estimirati kao *1/p*, gdje je *p* vjerovatnoća da se tim algoritmom nađe rješenje. Kada je u pitanju rješavanje problema osam kraljica, $p \approx 0.14$, pa je okvirno potrebno sedam restartovanja da bi se našlo optimalno rješenje. Očekivani broj koraka se dobija kao broj koraka jedne uspješne iteracije plus *(1-p)/p* puta broj koraka u neuspjeloj iteraciji, dakle *22*. Kada se dozvole sporedni potezi dobija se da je potrebno $1/0.94 \approx 1.06$ iteracija i da je prosječan broj koraka $(1x21)+(0.06/0.94)x64 \approx 25$. Dakle, za slučaj problema osam kraljica slučajno restartovanje je veoma efikasno. Pokazuje se da se čak i za problem od tri miliona kraljica na ovaj način može naći rješenje za manje od minut [1].

Iako rijetko, i samo u odsustvu globalnih informacija, može se planinarenje primijeniti i na traženje optimalne putanje kroz stablo pretraživanja. U ovom slučaju se uvijek razvija najbolje dijete čvora koji je posljednji razvijen, odnosno sljedeći čvor koji će biti ispitivan i proširivan se bira samo među djecom trenutno ispitivanog čvora. Ukoliko se planinarenjem dođe do čvora čija su sva djeca listovi, ispitaju se sva njegova djeca i ukoliko ne predstavljaju rješenje, ispituje se i proširuje najbliži otvoreni čvor. Najbliži otvoreni čvor je dijete nekog od čvorova iz putanje kojom se došlo do lista.

Algoritam za pretraživanje planinarenjem bi bio:

1. Formirati listu čvorova koja inicijalno sadrži samo startni čvor.
2. Dok se lista čvorova ne isprazni provjeriti da li je prvi element liste ciljni čvor.
   a. Ako je prvi element ciljni čvor, vratiti uspješno nađeno rješenje i prekinuti dalje pretraživanje.
   b. Ako prvi element nije ciljni čvor, ukloniti ga iz liste i sortirati njegove sljedbenike iz stabla pretrage (ako ih ima). Dodati sortiranje sljedbenike na početak liste, tako da sljedbenik sa najmanjom funkcijom procjene bude njen prvi element.

3. Ako je pronađen ciljni čvor, pretraga je uspješno završena; u suprotnom pretraga je neuspješna.

Minimum zahtijevanog znanja sa petog termina

1. Pretražiti neko stablo planinarenjem, pri čemu se može desiti da se zada konkretan problem koji je potrebno predstaviti u prostoru stanja, a nakon toga izvršiti pretraživanje.
2. Dati ideju kako bi se predstavilo rješenje nekog konkretnog problema i na koji način bi se to rješenje poboljšavalo planinarenjem. Šta bi bila heuristička funkcija, šta susjedna stanja (rješenja)

LITERATURA

[1] Russell S., Norvig P.: *Artificial Intelligence: A Modern Approach*, Prentice Hall, NJ, 1995.

[2] *http://www.zemris.fer.hr/predmeti/tes/nastava.html* - zadnji put pristupano, 10. 02. 2010. godine.