UNIVERZITET CRNE GORE

Elektrotehnički fakultet, Podgorica

Materijal sa četvrtog termina predavanja iz

EKSPERTNIH SISTEMA

Informisana pretraživanja

Pretraživanja sa ograničenom memorijom

Prof. dr Vesna Popović-Bugarin

Podgorica, 2013.

4.3 Informisano-usmjereno pretraživanje

Informisano, ili kako se češće naziva, usmjereno pretraživanje koristi dodatne informacije o problemu čije se rješenje traži, za razliku od neusmjerenog pretraživanja koje koristi samo formulaciju problema, a pretražuje po nekom unaprijed definisanom redoslijedu. Generalan pristup koji se koristi kod usmjerenog pretraživanja je pristup ***prvo-najbolji***. Ovim pristupom se prilikom odabira čvora koji će se sledeći proširiti koristi *funkcija procjene f(n)* koja predstavlja mjeru udaljenosti od ciljnog čvora. Proširuje se onaj čvor za koji se smatra, na osnovu dostupnih informacija, da ima najmanju udaljenost od ciljnog.

Specijalni slučajevi informisanog pretraživanja su:

- pohlepno pretraživanje (engl. greedy best-first searh);
- A*.

U principu, postoji cijela familija različitih algoritama za usmjereno pretraživanje po principu prvo-najbolji, a specificira ih *heuristička funkcija*, koja se obično obilježava sa *h(n),* i koja je jednaka estimiranoj vrijednosti najmanje udaljenosti *n*-tog čvora do ciljnog čvora. U nekim slučajevima je *f(n)=h(n).*

4.3.1 Pohlepno pretraživanje

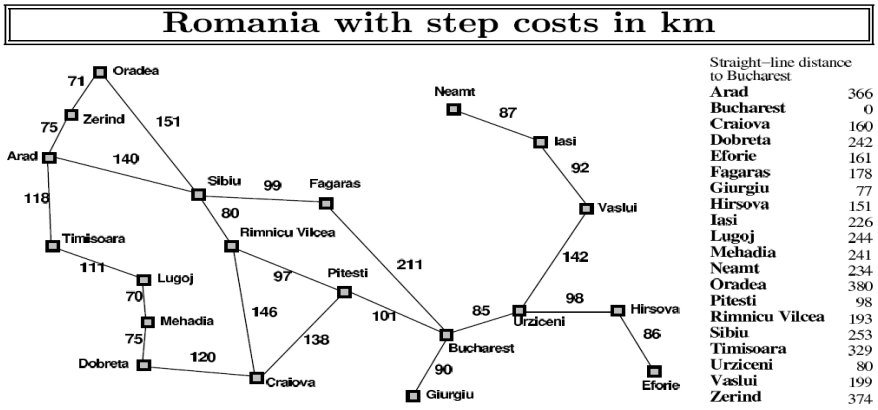
Ovim algoritmom pretraživanja se u svakoj iteraciji proširuje onaj čvor za koji se smatra da je najbliži cilju *f(n)=h(n).* Može se smatrati direktnim proširenjem neusmjerenih heuristika, sa izmjenom da se prilikom proširivanja nekog čvora, njegova djeca ne stavljaju ni na početak, ni na kraj liste koju čine otvoreni a neispitani čvorovi, već se nakon svakog dodavanja novootvorenih čvorova, ova lista sortira po heurističkoj funkciji *h(n)* u rastućem redosljedu, i čvor sa najmanjom vrijednošću se stavlja na prvo mjesto u listi.

Algoritam za pohlepno pretraživanje 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 dodati njegove sledbenike (ako ih ima) iz stabla pretrage u listu, a nakon toga sortirati listu u rastućem poredku, tako da na početku liste bude čvor sa najmanjom funkcijom procjene *f(n)*.
3. Ako je pronađen ciljni čvor, pretraga je uspješno završena; u suprotnom pretraga je neuspješna.

Kod problema pronalaženja najkraće putanje, kao najlogičniji izbor za heurističku funkciju se uzima vazdušna udaljenost dva grada.

Pohlepno pretraživanje se najčešće ilustruje kroz putnu mrežu Rumunije, [1],[2], Slika 1. Pohlepno pretraživanje je ilustrovano kroz primjer traženja puta od Arada do Bukurešta, Slika 2. Ovdje je nađena putanja bez ispitivanja ijednog čvora koji nije dio putanje do cilja, pa se može reći da je pohlepno pretraživanje dalo rezultat sa minimalnom cijenom pretraživanja. Ipak, rješenje koje se dobilo nije optimalno.



Slika 1 Putna mreža Ruminije. Sa desne strane su date udaljenosti gradova do Bukurešta. Slika je preuzeta iz [3]

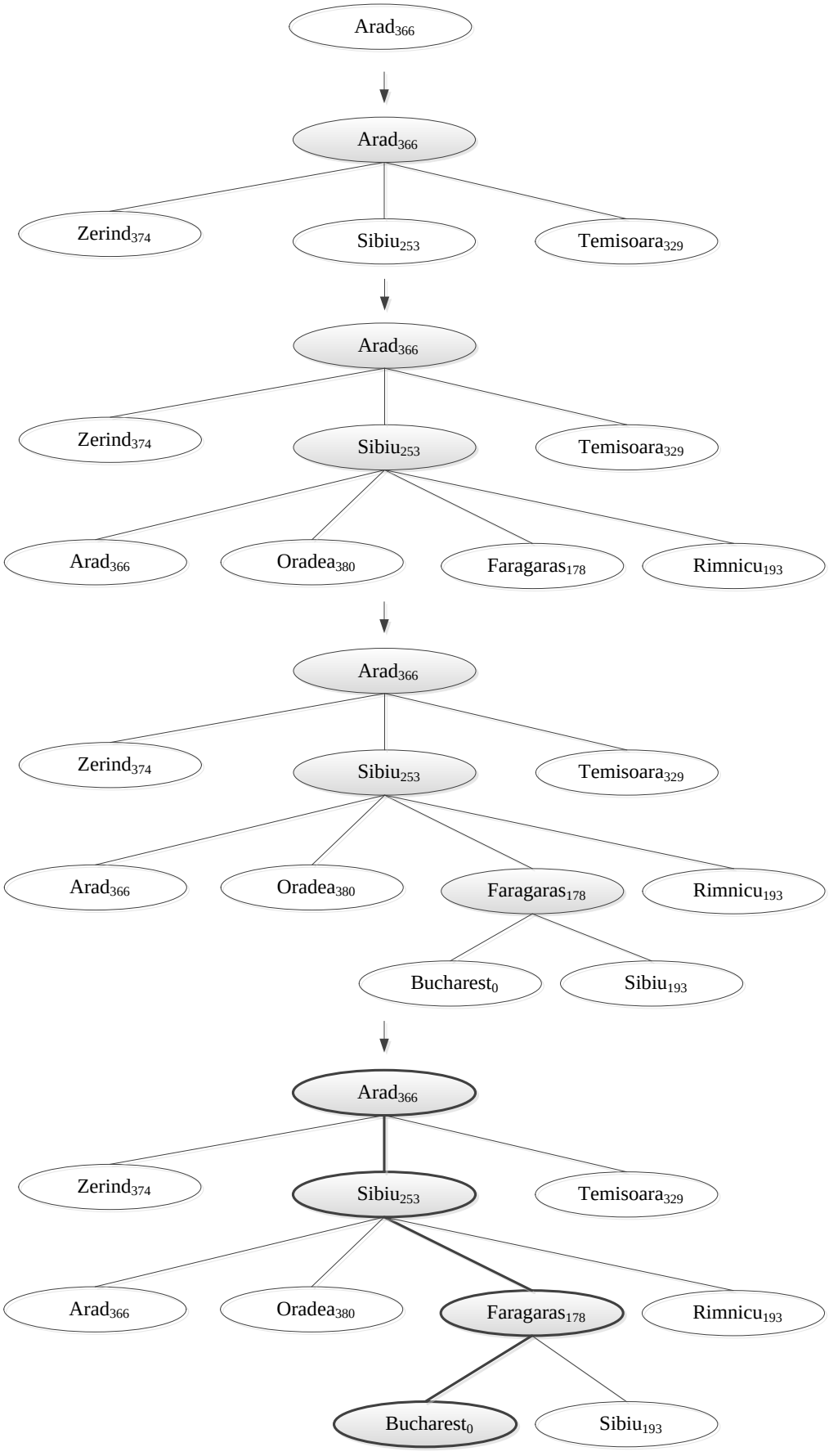
Kao funkcija procjene se koristi udaljenost po vazdušnoj liniji nekog grada od cilja, Slika 1. Naravno, ovaj vid pretraživanja se ne može primijeniti ukoliko je dostupno samo predstavljanje problema u prostoru stanja, već je potrebno dodatno iskustvo (informacije) vezano za udaljenost pojedinih gradova od cilja**. Obratiti pažnju da se u svakom trenutku prilikom odabira sljedećeg čvora koji će biti ispitivan i proširivan uzima u obzir svaki otvoren čvor, a ne samo djeca čvora koji je poslednji zatvoren, te da se za naredni bira otvoreni čvor sa najmanjom funkcijom procjene, bez obzira na njegovu dubinu.**

Međutim, ovaj način pretraživanja i ima ime pohlepan, jer će odabirati čvor sa najmanjom vazdušnom udaljenošću od ciljnog, što ga može dovesti do toga da u nekom slučaju bira grad koji nema vezu sa ciljnim, pa se mora vraćati nazad i vrši bespotrebno generisanje čvorova. Ukoliko bi tražili putanju Iasi-Faragas, iz grada Iasi bi se krenulo u Neamt, jer je njegova udaljenost po vazdušnoj liniji od Faragas-a manja nego za drugi grad Vaskli koji je susjedan početnom gradu Iasi, Slika 1. Kada se dođe u Neamt zaključilo bi se da nema dalje veze sa nekim gradom i morali bi se vratiti u početni grad, pa tražiti put preko grada Vaslui. NA ovaj način je bespotrebno generisan čvor Neamt. Šta ako se ne vodi računa o ponavljenom stanju - Iasi?

Posebna se pažnja mora voditi o stanjima koja se ponavljaju kako bi se obezbijedila potpunost algoritma i kako se ne bi ulazilo u beskonačnu petlju.

Kako, u principu, ova strategija pretraživanja prati jednu putanju, birajući za naredni čvor onaj koji ima najmanju vrijednost *f(n)*, a alternativne čvorove uzima u obzir tek ako dođe do kraja trenutne putanje i ne nađe cilj, jako je slična pretraživanju u dubinu. S toga, pohlepno pretraživanje ima iste mane kao pretraživanje u dubinu: nije optimalno i ne mora biti konačno (ukoliko imamo neku beskonačno dugu putanju). Vremenska i prostorna složenost u najgorem slučaju je $O(b^m)$.

Ključna stvar kod ove vrste pretraživanja je dobar odabir heurističke funkcije jer ona može dovesti do znatnog smanjenja vremenske i prostorne složenosti.



Slika 2 Pohlepno pretraživanje kroz primjer traženja puta od Arada do Bukurešta

4.3.2. A* pretraživanje

Pohlepno pretraživanje ima manu što za sljedeći čvor koji će se proširivati bira čvor koji je najbliži cilju, ne vodeći računa o tome da li će cijena kompletnog puta do cilja biti na taj način uvećana. Stoga, ova vrsta pretraživanja često daje neoptimalna rješenja. A* pretraživanje bira čvor koji će proširiti, ne samo na osnovu njegove udaljenosti od cilja, već na osnovu ukupne cijene putanje koja bi se dobila tim proširivanjem, pa je:

$$f(n) = g(n) + h(n)$$

gdje je *g(n)* udaljenost od startnog čvora do *n*-tog čvora, a *h(n)* udaljenost *n*-tog čvora od cilja. Na ovaj način nam funkcija procjene predstavlja procjenu cijene najjeftinije putanje do rješenja; a koja bi prolazila kroz čvor *n*. *g(n)* znamo jer je taj dio puta već pređen, ali se *h(n)* najčešće može samo procijeniti, i od njenog kvaliteta zavise performanse algoritma. Ipak, pokazuje se da biranjem heurističke funkcije *h(n)* tako da posjeduje određena svojstva, možemo postići i kompletnost i optimalnost ovog algoritma. Ukoliko bismo znali tačnu vrijednost heurističke funkcije, zapravo ne bismo ni imali pretraživanje, već usmjereno kretanje do cilja najkraćom putanjom.

Optimalnost A* algoritma se postiže ukoliko se heuristička funkcija bira tako da nikada nije veća od stvarne udaljenosti tog čvora do cilja. Drugim riječima, trebalo bi birati heurističku funkciju tako da nikada ne precjenjuje udaljenost od nekog čvora do cilja.

Jedna od funkcija koja zadovoljava ovo svojstvo koje vodi do optimalnosti jeste procjena udaljenosti dva grada njihovim rastojanjem po vazdušnoj liniji, pa se ponovo može posmatrati slučaj određivanja putanje od Arada do Bukurešta, Slika 3. Vidi se da nakon proširivanja čvora Fagaras, iako među otvorenim čvorovima postoji ciljni čvor, on se ipak ne bira kao naredni čvor putanje, jer je njegova funkcija procjene veća od funkcije procjene za čvor Pitesti.

Ukoliko bi se ciljni čvor (C2) našao među otvorenim čvorovima, a njegovim biranjem se ne bi dobila optimalna putanja, ova strategija pretraživanja ga ne bi birala za ispitivanje i proširivanje, jer bi pronašla neki drugi čvor (C1) preko kojeg bi se dobila bolja putanja do ciljnog čvora, odnosno koji bi davao *f(n)* (cijenu putanje do rješenja) manju nego što je daje taj ciljni čvor. Imali bi:

$$f(C2) = g(C2) + h(C2) = g(C2) > f^*(C),$$

gdje je sa $f^*(C)$ obilježena cijena optimalne putanje i

$$f(C1) = g(C1) + h(C1) \leq f^*(C)$$

ukoliko $h(C1)$ nije precijenjeno. Dakle, A* neće ispitivati ciljni čvor, bez obzira što se ciljni čvor nalazi među otvorenim čvorovima, ukoliko on ne čini dio optimalne putanje jer bi se time završila pretragu dajući neoptimalno rješenje.

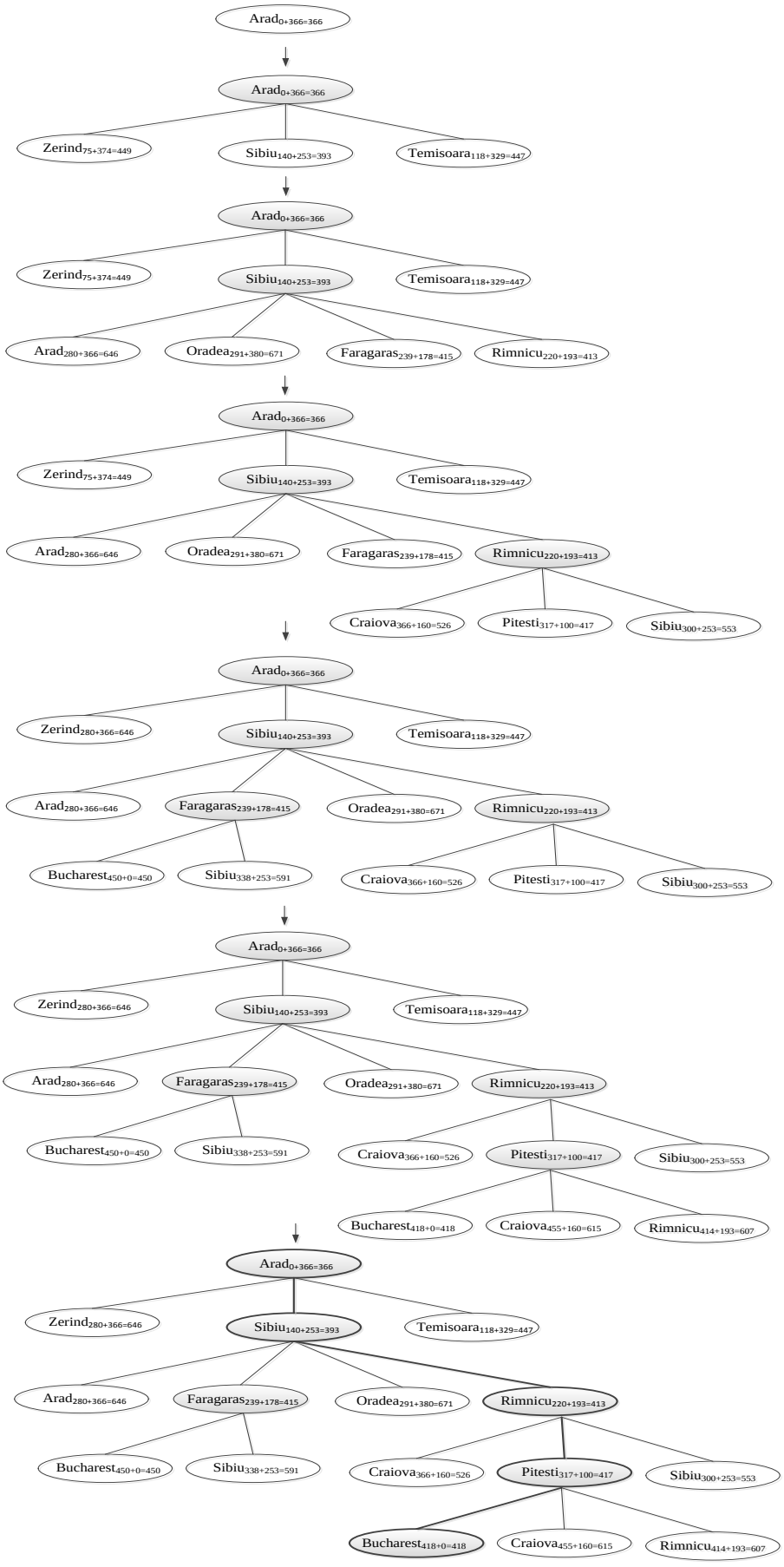
I kod ove strategije pretraživanja ciklusi mogu praviti problem. Ukoliko imamo više putanja koje vode do istog čvora rješenje da se zadrži optimalnost i dalje leži u odbacivanju onog čvora koji ima veću funkciju procjene, što se zapravo svodi na poređenje cijene putanje do tog čvora. Alternativno rješenje je biranje heurističke funkcije tako da se uvijek prva prati optimalna putanja do stanja koje se ponavlja. Da bi se ovo postiglo, heuristička funkcija mora biti takva da za svaki čvor *n* i svako njegovo dijete *n'* koje se generiše nekim prelazom (operatorom) *op* važi da

estimirana udaljenost do cilja od čvora *n* nije veća od zbira cijene prelaza na čvor *n'* i udaljenosti čvora *n'* od cilja*:*

$$h(n) \leq cijena(n, op, n') + h(n')$$

Heurističke funkcije koje zadovoljavaju gornje svojstvo se nazivaju konzistentnim. Biranjem heurističke funkcije tako da zadovoljava konzistentnost se postiže da rezultujuća funkcija procjene bude neopadajuća duž svake putanje u stablu. Ovo dalje potvrđuje optimalnost dobijenog rješenja. Naime, prvi ciljni čvor koji se bira za proširivanje je na optimalnoj putanji, jer svi ciljni čvorovi koji se kasnije biraju imaju makar istu, ako ne i veću funkciju procjene. S druge strane, A* je kompletan (ako svaki čvor ima konačan broj grana) jer će se povećavanjem *f(n)* u jednom trenutku stići do vrijednosti koja odgovara optimalnoj putanji.

Za A* algoritam se kaže da je *optimalno efikasan*, što znači da će doći do optimalnog rješenja razvijanjem najmanjeg broja čvorova poredeći sa ostalim algoritmima koji kreću od korijena. Ovo važi iz razloga što A* algoritam razvija sve čvorove kojima je funkcija procjene manja od cijene optimalne putanje. S druge strane, upravo ovo vodi ka glavnoj mani A* algoritma, a to je velika prostorna i vremenska kompleksnost. Vremenska složenost je eksponencijalna, a prostorna složenost je $O(b^d)$. Uz male modifikacije heurističke funkcije ili žrtvovanje kompletnosti ili optimalnosti, mogu se prevazići problem prostorne i vremenske složenosti.



Slika 3 A* pretraživanje na primjeru nalaženja puta od Arada do Bukurešta.

4.4. Pretraživanja sa ograničenom memorijom

Glavni nedostatak A* pretraživanja, bez obzira na svojstvo optimalne efikasnosti, jeste velika memorijska zahtijevnost. On pamti sve isprobane putanje (zatvorene čvorove) i sve otvorene čvorove, odnosno sve čvorove generisane u toku pretraživanja. S obzirom da A* pretraživanje proširuje uvijek čvor sa najmanjom funkcijom procjene, svi čvorovi koji imaju funkciju procjene manju od cijene optimalne putanje će biti prošireni u toku pretrage za ciljem (čime se i postiže optimalnost). Broj čvorova koji imaju funkciju procjene manju od cijene optimalne putanje u većini problema raste eksponencijalno sa dubinom rješenja, ukoliko se ne koristi adekvatna heuristička funkcija sa malom greškom u poređenju sa tačnom cijenom putanje [1]. Dakle, bez obzira na veliku vremensku složenost, glavni problem ove strategije pretraživanja je velika memorijska kompleksnost. Zapravo se može reći da će A* pretraživanju prije ponestati memorije, nego vremena. Zbog toga je razvijen veliki broj algoritama kojima se pokušava prevazići problem prostorne kompleksnosti A* pretraživanja, a na način povećavanja vremenske složenosti.

4.4.1 IDA* - iterativno produbljivanje A*

Postoji više strategija za smanjivanje memorijske zahtijevnosti, a najjednostavnija je proširenje iterativnog pretraživanja u dubinu. Uzimajući u obzir informacije o razmatranom problemu, u svakoj iteraciji se pretraživanje vrši u dubinu, ali ne do dubine zadate za tekuću iteraciju, već dok se ne dođe do otvorenog čvora sa funkcijom procjene većom od zadate za tekući iteraciju. On se proglašava listom i izbacuje iz memorije i ispituju se ostali potomci njegovog roditelja. Ukoliko svi potomci imaju funkciju procjene veću od zadate vraća se na najbliži zatvoreni plići čvor koji ima neispitane djece. Zatim se vrši njihovo ispitivanje i proširivanje u dubinu po istoj proceduri. Tekuća iteracija dostiže kraj kada se nađe rješenje ili nema više čvorova za ispitivanje - svi listovi imaju veću vrijednost funkcije procjene od zadate i izbačeni su iz memorije. Minimalna funkcija procjene *f(x)* svih čvorova koji su u tekućoj iteraciji proglašeni za listove se uzima kao kontrolna-zadata za sljedeću iteraciju. Modifikacija u odnosu na iterativno pretraživanje u dubinu se sastoji u tome da se u sljedećoj iteraciji pretraživanje vrši, ne do za jedan veće dubine od trenutne, već dok se ne prošire svi čvorovi koji imaju funkciju procjene ne veću od kontrolne. Ostali čvorovi se ne ispituju i proširuju. Kada se zaključi da im je funkcija procjene veća od zadate, jednostavno se izbace iz memorije. Važno je samo voditi računa da se uvijek ima podatak o najmanjoj vrijednosti funkcije procjene svih čvorova koji su proglašeni listovima i izbačeni iz memorije. Rezultujući algoritam se naziva iterativno-produbljivanje A* (engl. iterative-deepening A* - IDA*).

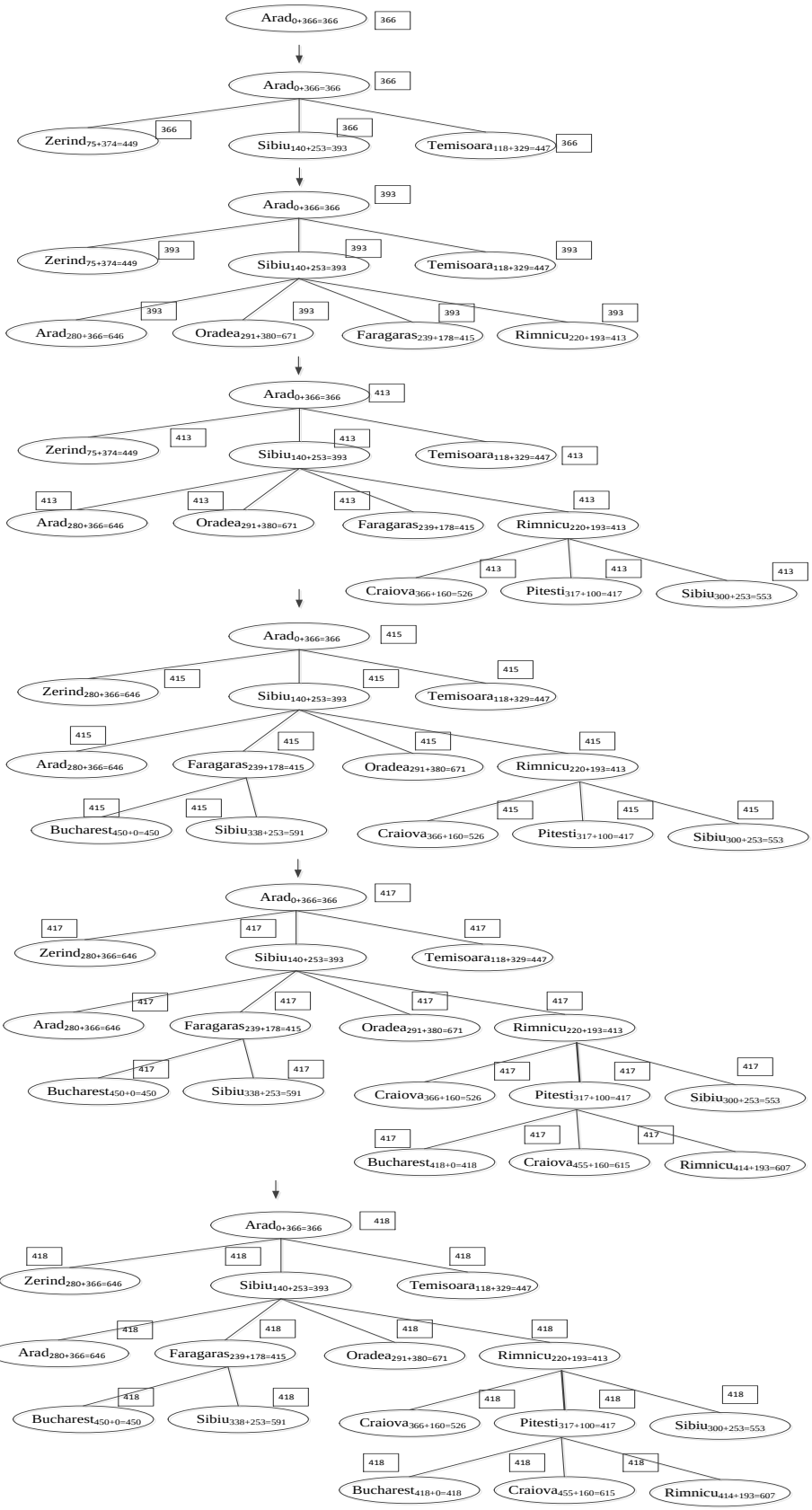
Ilustracija IDA* algoritma je data za primjer traženja najkraće putanje od Arada do Bukurešta, Slika 4. Za svaku iteraciju prikazano je stablo po kojem se vrši pretraživanje u dubinu. Kontrolna funkcija procjene koja određuje koji će se čvorovi proširivati je prikazana u pravougaonicima. **Treba imati na umu da je dato čitavo stablo po kojem se vrši pretraživanje u dubinu, dok se u svakom trenutku u memoriji nalaze samo otvoreni a neispitani čvorovi i putanja koja se trenutno ispituje.** Takođe se nalazi i vrijednost minimalne funkcije procjene listova.

U prvoj iteraciji je kontrolna funkcija procjene jednaka *f(x)* korijena, dakle *366* i ispituju se samo čvorovi sa funkcijom procjene manjom od *366*, t.j samo korijen u ovom slučaju. Kontrolna funkcija procjene za sljedeću iteraciju se uzima kao minimalna vrijednost svih čvorova koji su u tekućoj iteraciji proglašeni listovima

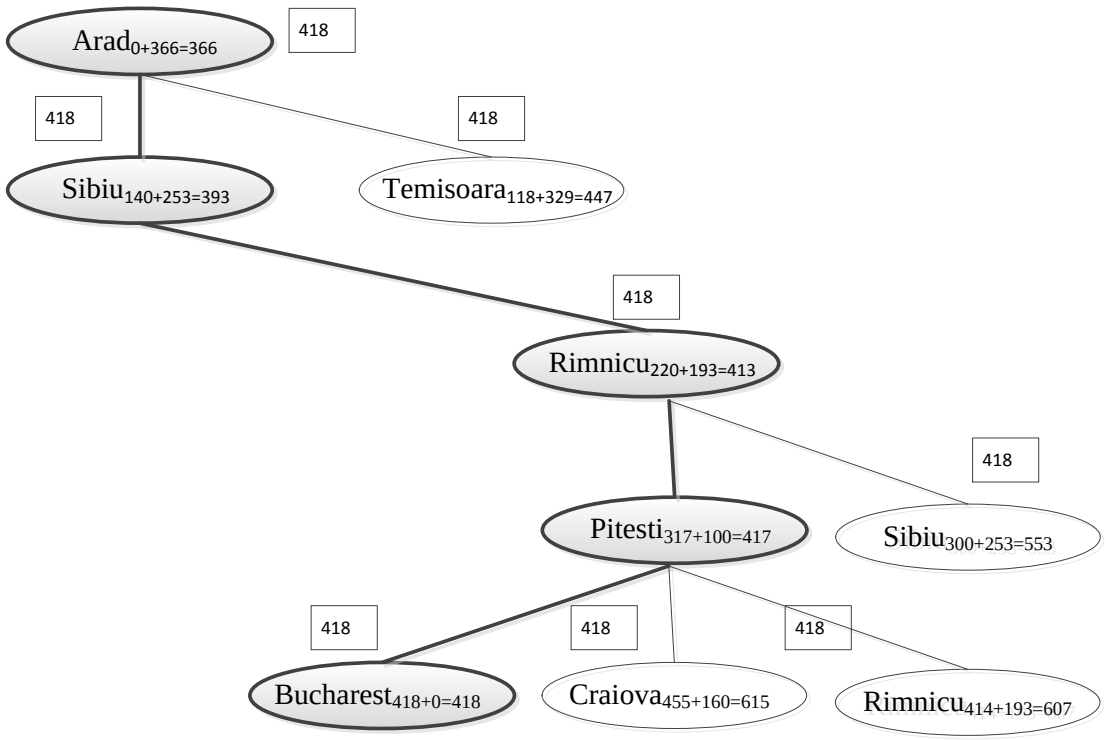
(Zerind, Sibiu i Temisoara) i iznosi *393*. Koristeći ovu kontrolnu funkciju procjene, granica do koje će se ispitivati i proširivati čvorovi u drugoj iteraciji je, ne dubina čvorova, već uslov da im je $f(x) \leq 393$. Kako nijedan od čvorova koji zadovoljavaju ovaj uslov ne predstavlja cilj, mora se ići na treću iteraciju uz povećavanje *f(x)* na minimalnu vrijednost svih listova stabla koje se ispitivalo u drugoj iteraciji. Funkcija procjene se za sljedeću iteraciju povećava na *f(x)=413*. Ni u ovoj iteraciji se ne pronalazi cilj, pa se na njenom kraju funkcija procjene za sljedeću iteraciju povećava na *f(x)=415* i postupak se ponavlja. U iteraciji sa *f(x)=415* među otvorenim čvorovima se pojavljuje i ciljni – Bukurešt. Međutim, ne završava se pretraživanje jer ovaj čvor ima funkciju procjene (*450)* veću od kontrolne, proglašava se listom i izbacuje iz memorije, bez ispitivanja da li je ciljni čvor. Samo se koristi njegova funkcija procjene za poređenje sa minimalnom funkcijom procjene listova koju u tekućoj iteraciji detektovani prije njega pretraživanjem u dubinu. I u narednoj iteraciji sa *f(x)=417* imamo dva lista Bukurešt, ali se ni jedan od njih ne proglašava ciljnim, jer oba imaju funkciju procjene veću od kontrolne, već se ide na sljedeću iteraciju uz *f(x)=418*. Kontrolna funkcija procjene je cijena ciljnog čvora i sa njom se dobija stablo koje ima u sebi optimalnu putanju, koja se i detektuje u zadnjoj iteraciji.

Ako se bolje pogleda stablo koje je generisano sa kontrolnom funkcijom *f(x)=413* vidi se da u njemu postoje listovi na različitim dubinama, bez obzira što oni imaju djecu u originalnom grafu pretraživanja (putnoj mreži Rumunije). Ovo je iz razloga što se za generisanje stabla koje će se pretraživati u dubinu, u svakoj iteraciji ne povećava dubina za jedan, već se povećava kontrolna funkcija procjene koja će ograničavati dubinu listova. Proširuju se i ispituju samo oni čvorovi koji imaju funkciju procjene manju od kontrolne. S druge strane, za određivanje kontrolne funkcije procjene za sljedeću iteraciju, uzima se minimalna vrijednost za funkciju procjene svih listova stabla generisanog u tekućoj iteraciji, bez obzira na dubinu na kojoj se nalazi neki list. Dakle, ne posmatraju se samo listovi na najvećoj dubini, Slika 4.

Prednost ove strategije pretraživanja u odnosu na klasično A* pretraživanje se ogleda u tome što se u trenutku kada neki čvor sljedbenik trenutno ispitivanog čvora ima veće *f(n)* od zadatog ne vrši njegovo ispitivanje i proširivanje. Ovaj čvor se proglašava listom i samo se provjeri da li je njegova funkcija procjene manja od one koja trenutno predstavlja minimum funkcije procjene listova. Ukoliko jeste, ta se vrijednost postavlja za tekuću minimalnu funkciju procjene listova. Nakon toga se ovaj list izbacuje iz memorije i prelazi se na sledećeg sljedbenika trenutno ispitivanog čvora. Ukoliko se svi sljedbenici trenutno ispitivanog čvora proglase za listove izbacuje se iz memorije i ispitivani čvor. Algoritam se vraća nazad na najbliži zatvoreni plići čvor koji ima otvorenih sljedbenika (na isti način kao kod pretraživanja u dubinu). Postupak se završava kada se nađe ciljni čvor ili se svi otvoreni čvorovi zatvore ili proglase listovima i izbace iz memorije – što će rezultovati izbacivanjem svih čvorova iz memorije. Kontrolna *f(n)* za sljedeću iteraciju je minimalna vrijednost funkcije procjene svih čvorova koji su proglašeni listovima u ovoj iteraciji.



Slika 4 IDA* za putnu mrežu Rumunije i nalaženje najkraćeg puta od Arada do Bukurešta. Za svaku kontolnu funkciju procjene, prikazanu u pravougaoniku dato je rezultujuće stablo koje se pretražuje u dubinu. Listovi se ne ispituju. Koriste se za određivanje *f(x)* za sljedeću iteraciju.



Slika 5 Čvorovi koji se nalaze u memoriji u trenutku nalaženja najkraćeg puta od Arada do Bukurešta. Uz njih se pamti i najmanja vrijednost funkcije procjene svih listova.

S obzirom na ovakav način pretraživanja, memorijska zahtijevnost IDA* algoritma je ista kao kod pretraživanja u dubinu, dakle linearna *O(bd)*. ***Algoritam je optimalan i kompletan.*** Optimalan je, jer ne radi klasično pretraživanje u dubinu, odnosno za neku putanju staje kada naiđe na čvor koji ima funkciju procjene veću od kontrolne. Ne provjerava da li je taj čvor cilj. Na ovaj način izbjegava nalaženje neoptimalnih rješenja koja su na manjim dubinama od optimalnog ili do kojih bi se prvo došlo klasičnim pretraživanjem u dubinu. U svakoj iteraciji se ide do čvorova čija je funkcija procjene manja ili jednaka kontrolnoj, koja je sa druge strane jednaka minimalnoj za sve listove iz prethodne iteracije. Ova kontrolna funkcija procjene je ili jednaka cijeni optimalne putanje *f(C*)*, što znači da će se cilj detektovati u tekućoj iteraciji ili je manja od nje, što znači da je potrebno još iteracija. IDA* proširuje sve čvorove sa $f(n)<f(C^*)$, a ne proširuje nijedan čvor sa $f(n) \geq f(C^*)$. Vremenska složenost predstavlja glavni nedostatak IDA* algoritma i u velikoj mjeri zavisi od kvaliteta heurističke funkcije. IDA* će imati veći broj otvorenih čvorova od A* algoritma, ali ukoliko se heuristička funkcija dobro odabere taj broj neće biti mnogo veći [2]. Imaće veći broj čvorova od A* iz razloga što ide u jednu stranu po dubini i ispituje i proširuje otvorene čvorove na koje nailazi ako im je funkcija procjene manja od kontrolne, a ne uzima u obzir preostale otvorene čvorove sve dok ne ispita i eventualno proširi svu djecu trenutno ispitivanog čvora.

IDA* algoritam je dobar za veliki broj problema sa jediničnom cijenom prelaza [1].

4.4.2 Rekurzivno pretraživanje prvo najbolji RBFS

Ova strategija pretraživanja je modifikacija pretraživanja po principu prvo najbolji, tako da se njegova memorijska zahtijevnost svodi na linearnu *O(bd).*

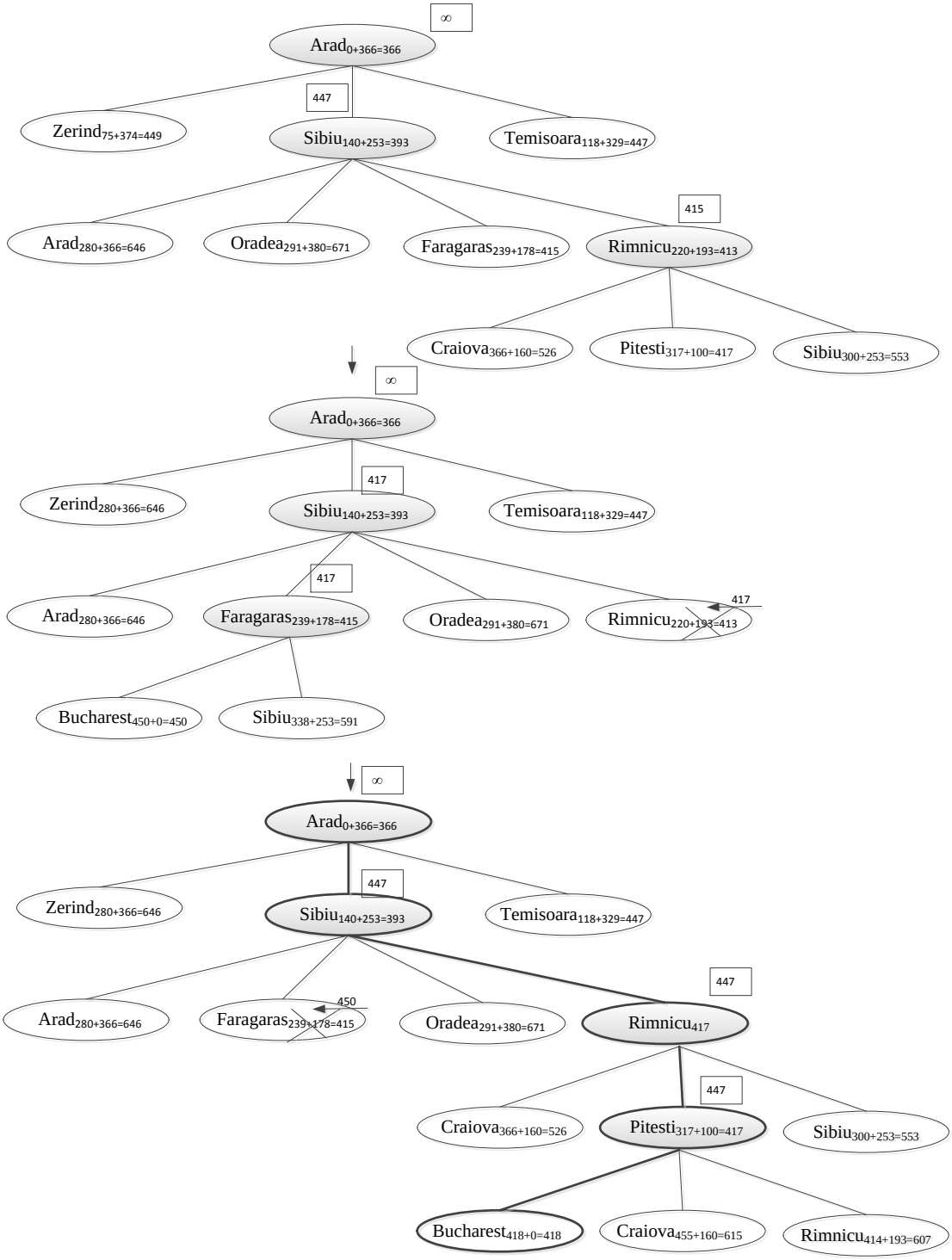
Pretraživanje se vrši po dubini, po principu prvo najbolji, ali se ne ide bezuslovno u dubinu putanjom u koju vodi najbolje dijete zadnjeg proširenog čvora. Za određivanje da li da se nastavi ovom putanjom vrši se poređenje funkcije procjene najboljeg djeteta zadnjeg proširenog čvora sa najmanjom funkcijom procjene preostalih čvorova koji su otvoreni u tom trenutku – cijenom alternativne putanje. Najbolje dijete se ispituje i proširuje samo ukoliko mu je funkcija procjene manja od cijene alternativne putanje. Kada se naiđe na čvor čija sva djeca imaju funkciju procjene veću od cijene alternativne putanje, ne proširuje se nijedno njegovo dijete, već se sva izbacuju iz memorije. Algoritam odustaje od tekuće putanje i prelazi na alternativnu putanju vraćajući se unazad na otvoreni čvor sa najmanjom funkcijom procjene. Vrijednost za funkciju procjene svakog čvora kroz koji se prolazi dok se ne dođe do alternativne putanje postavlja se na najmanju vrijednost funkcije procjene njegove djece. Na ovaj način se pamti funkcija procjene najboljeg lista odbačene putanje, pa čak i poddrveta od kojeg se odustalo, i može se to poddrvo ponovo razviti ukoliko se pokaže da mu je funkcija procjene manja od funkcije procjene svih otvorenih čvorova. Kada se dođe do alternativne putanje, ona postaje najbolja putanja, a pamti se cijena njoj alternativne putanje, Slika 6.

Iako je RBFS efikasniji od IDA* algoritma i njemu je glavni nedostatak potreba za ponovnim generisanjem čvorova koji su već bili ispitivani i generisani, ali su izbačeni iz memorije, Slika 6. Na ovoj slici nijesu prikazani svi koraci algoritma već stabla u trenutku kada se u trenutno najboljoj putanji naiđe na čvor čija sva djeca imaju veće *f(n)* od cijene alternativne putanje (*Rimnicu* u prvom stablu). Zatvoreni čvorovi trenutno najbolje putanje su osjenčeni. U svakom sljedećem stablu se vidi da je funkcija procjene svih čvorova kroz koje se prolazilo na putu do alternativne putanje, nakon odustajanja od prethodne putanje, postavljena na vrijednost funkcije procjene njihovog najboljeg djeteta (*f(x)* za *Rimnicu* je postavljena na *417* u drugom stablu). Kada se na ovaj način dođe do čvora koji je dio alternativne putanje (*Sibiu* u drugom stablu), ta putanja postaje najbolja. Ova putanja se dalje prati i proširuje na isti način kao prethodna najbolja. U zadnjem grafu je prikazan trenutak pronalaska cilja.

Svaki put kada se u RBFS odustane od neke putanje i pređe na alternativnu putanju zapravo imamo jednu iteraciju IDA* algoritma, odnosno potrebno je ponovo proširivati veliki broj odbačenih čvorova kako bi se ponovo formiralo odbačeno poddrvo. Velika je vjerovatnoća da će se proširivanjem trenutno ispitivanog čvora dobiti djeca koja čine putanju skupljom od alternativne, što će dovesti do odustajanja od te putanje, prelaska na alternativnu, razvijanja njenog najboljeg čvora, ispitivanja njegove djece i najvjerovatnije dobijanja skuplje putanje od alternativne, te odustajanja od trenutne putanje.

RBFS je optimalan ukoliko *h(n)* ne precijenjuje udaljenost od cilja. Memorijska zahtijevnost mu nije velika. U svakom trenutku u memoriji je trenutno ispitivana putanja i otvoreni čvorovi, pa je *O(bd)*.

Glavni nedostatak, pored velike vremenske složenosti i IDA* i RBFS strategija, je nemogućnost ispitivanja ponovljenih stanja, osim u trenutnoj putanji, pa se može desiti da ispituju isto stanje više puta.



Slika 6 Pronalaženje puta od Arada do Bukurešta. Ispitani čvorovi trenutno praćene putanje su osjenčeni. Svako stablo ilustruje trenutak kada se u trenutno najboljoj putanji naiđe na čvor čija sva djeca imaju veće *f(n)* od alternativne putanje. Pored svakog čvora je prikazana uokvirena cijena alternativne putanje. U zadnjem grafu je prikazan trenutak pronalaska cilja i odgovarajuća optimalna putanja.

4.4.3 SMA* - pojednostavljeni A* algoritam sa ograničenom memorijom

Pojednostavljeni A* algoritam sa ograničenom memorijom (Simplified Memory-bonded A* - SMA*) koristi svu raspoloživu memoriju za razliku od RBFS koji koristi *O(bd)* memorije bez obzira na to da li mu je dostupno više memorije. SMA* algoritam vrši proširivanje čvorova na isti način kao A* algoritam sve dok ima raspoložive memorije. Kada se iskoristi sva raspoloživa memorija, ne prekida se algoritam, već se iz memorije izbacuje otvoreni čvor sa najgorom (najvećom) funkcijom procjene, a ta se vrijednost čuva tako što postaje funkcija procjene njegovog roditelja (slično kao u RBFS). Ponovno generisanje izbrisanih čvorova se vrši u slučaju da sve alternativne putanje imaju veću funkciju procjene od zapamćene funkcije procjene za najbolji od listova koji su obrisani.

Ukoliko svi otvoreni čvorovi imaju istu funkciju procjene briše se najstariji čvor, odnosno otvoreni čvor koji je prvi generisan, a proširuje najmlađi čvor, to jeste čvor koji je zadnji generisan.

SMA* je kompletan ukoliko postoji rješenje na dubini *d* do koje može doći sa dostupnom memorijom. Optimalan je ukoliko optimalno rješenje postoji na dubini do koje može doći sa dostupnom memorijom. Ukoliko ne postoji, SMA* vraća najbolje rješenje do kojeg je mogao stići sa raspoloživom memorijom. Memorijska složenost mu je konstantna i jednaka je dostupnoj memoriji, dok je vremenska složenost znatno veća u odnosu na A* s obzirom da se može desiti da mora ponovo generisati neki od odbačenih čvorova i podstabala.

Pokazuje se da je SMA* možda najbolji algoritam za opštu upotrebu koji daje optimalno rješenje. Ipak, poželjno ga je koristiti kada je prostor pretraživanja graf, a ne stablo, funkcija procjene je neuniformna i generisanje čvorova je komplikovanije i zahtijevnije od održavanja liste otvorenih i zatvorenih čvorova. U slučaju veoma teških problema kod kojih se često prelazi sa jedne putanje na drugu i mali broj čvorova, u poređenju sa ukupnim prostorom stanja, se može čuvati, može se desiti da se neki problem može dosta brzo riješiti A* algoritmom sa neograničenom memorijom, dok je usljed potrebe za regenerisanjem čvorova neprihvatljivo mnogo vremena potrebno SMA* algoritmu za njegovo rješavanje. Očigledno da ne postoji algoritam koji će dati optimalno rješenje za sve probleme, pa se rješenje nalazi u odbacivanju optimalnosti ili smanjivanju broja generisanih čvorova upotrebom što bolje heurističke funkcije.

Minimum zahtijevanog znanja sa trećeg termina

1. Koji su problemi vezani za A* pretraživanje i koji algoritmi su predloženi u cilju njihovog pretraživanja?
2. Šta je najveći problem do kojeg može doći kada neki algoritam pretraživanja pamti samo trenutno ispitivanu putanju i otvorene čvorove?
3. Zadatak sa problemom ili grafom u kojem je potrebno naći cilj korišćenjem nekog od rađenih algoritama preteživanja.

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. 2012. godine.