OPTIMALNA GLOBINA PREISKOVANJA PRI LRTA*

Mitja Luštrek
Odsek za inteligentne sisteme
Institut Jožef Stefan
Jamova 39, 1000 Ljubljana, Slovenija
Telefon: +386 1 4773380; telefaks: +386 1 4773131
E-pošta: mitja.lustrek@ijs.si

POVZETEK

Za preiskovalne algoritme z takojšnjim odzivom, kakršen je LRTA*, velja, da najdejo boljše rešitve, če uporabljajo večjo globino preiskovanja. To pa ne drži vedno in tudi kadar drži, je večja globina od najmanjše potrebne časovno potratna. Zaradi tega v tem prispevku preučimo dve metodi za vnaprejšnje določanje optimalne globine preiskovanja. Eno od metod dopolnimo z uporabo abstraktnih stanj, kar omogoči, da število stanj, za katere je optimalno globino preiskovanja treba določiti, zmanjšamo. Dopolnjeno metodo smo preizkusili na problemu iskanja poti po zemljevidih iz komercialnih računalniških iger. Po nekajurnem določanje optimalnih globin je uporaba teh globin pri iskanju poti dala okrog desetkraten prihranek časa in dosegala dolžine poti primerljive z uporabo nespremenljive globine.

1 UVOD

Najbolj znan algoritem za enoagentno hevristično preiskovanje je prav gotovo A* [6]. Ob uporabi popolne (optimistične) hevristične funkcije A* vedno najde optimalno pot do cilja. Žal pa je pri nekaterih problemih prostor preiskovanja prevelik, da bi mu bil A* (dovolj hitro) kos. V takih primerih se uporabljajo preiskovalni algoritmi s takojšnjim odzivom (*real-time*). Klasičen tovrsten algoritem je learning real-time A* ali LRTA* [7], ki se mu bomo posvetili v tem prispevku.

LRTA* in sorodni algoritmi preiščejo del prostora okrog trenutnega stanja in nato naredijo korak v najbolj obetavni smeri proti cilju. Postopek ponavljajo, dokler cilja ne dosežejo. Vprašanje, ki se ob rabi tovrstnih algoritmov zastavi, je, do kakšne globine naj algoritem v vsakem koraku prostor preišče. Razširjeno je prepričanje, da je najdena pot do cilja boljša, če je globina preiskovanja večja. Nedavne ugotovitve [1, 3, 8, 9] pa kažejo, da to ne drži vedno, da včasih prihaja do tako imenovane patologije, kar pomeni, da se manjše globine preiskovanja obnesejo bolje. Ker poleg tega plitvejše preiskovanje terja tudi manj procesorskega časa, bi bilo zaželeno za vsako stanje poznati optimalno globino preiskovanja.

Optimalno globino preiskovanja v nekem stanju bi si želeli določiti kar neposredno iz topologije dela prostora, v katerem stanje leži. Žal pa kaka splošno uporabna metoda za to ni znana. Tako si lahko pomagamo z vnaprejšnjim določanjem optimalnih globin s pomočjo preiskovanja. Tak postopek je časovno zelo zahteven, vendar si ga pri nekaterih problemih lahko privoščimo. Npr. v računalniški igri, kjer mora biti množica enot zmožna hitrega hkratnega iskanja poti po zemljevidu, izdelovalec igre, preden igro pošlje v trgovine, zlahka nameni nekaj ur ali dni procesiranja določanju optimalnih globin preiskovanja.

Prispevek je razdeljen na šest poglavij. Poglavje 2 opredeli problem in predstavi algoritem LRTA*. Poglavje 3 obravnava naivno in časovno zahtevno metodo za določanje optimalne globine, v poglavju 4 pa je opisana učinkovitejša metoda. V poglavju 5 to metodo dopolnimo z uporabo abstraktnih stanj, kar omogoči, da število stanj, za katere je optimalno globino preiskovanja treba izračunati, zmanjšamo. Poglavje 6 prispevek sklene in poda nekaj smernic za nadaljnje delo.

2 PROBLEM

Problem, s katerim se ukvarja ta prispevek, je iskanje poti iz začetnega stanja v končno stanje po zemljevidu razdeljenem v kvadratna polja, od katerih so nekatera neprehodna. Agent, ki išče pot, se iz vsakega polja lahko premakne v katerokoli prehodno sosednje polje – cena premika naravnost je 1, cena diagonalnega premika pa $\sqrt{2}$. Prosta polja tvorijo množico stanj S. Preiskovalni problem določajo zemljevid, začetno stanje s_0 in končno stanje s_g.

Za načrtovanje poti agent uporablja algoritem LRTS [2] nastavljen tako, da ustreza LRTA*. Algoritem preišče vsa stanja do d premikov oddaljena od trenutnega stanja $s_c \in S$. Stanja na robu preiskanega področja oceni s hevristično funkcijo h, ki ocenjuje oddaljenost stanja od s_g. Agent nato naredi prvi premik po najkrajši poti do najobetavnejšega stanja na robu preiskanega prostora. To je stanje $s_{f\,opt} \in S$, ki leži na najcenejši poti do s_g. Cena poti $f(s_{f\,opt}) = g(s_{f\,opt}) + h(s_{f\,opt})$, pri čemer je $g(s_{f\,opt})$ cena poti od s_c do $s_{f\,opt}$, $h(s_{f\,opt})$ pa hevristična ocena cene poti od $s_{f\,opt}$ do s_g. Na začetku so hevristične ocene enake cenam poti po povsem prehodnem zemljevidu. Po opravljenem premiku pa $h(s_c)$ dobi vrednost $f(s_{f\,opt})$, s čimer algoritem popravi hevristične vrednosti na področjih, kjer so bile na začetku preveč optimistične, tako da agent lahko najde pot okrog ovir tudi takrat, kadar tega ne more doseči s preiskovanjem do globine d.

V poizkusih opisanih v tem prispevku smo uporabili šest zemljevidov iz komercialnih računalniških iger: tri iz Baldur's Gate in tri iz Warcraft III. Zemljevidi imajo od 5.672 do 18.841 prehodnih stanj. Na vsakem zemljevidu smo naključno izbrali končno stanje, nato pa smo iz vseh drugih stanj iskali pot do njega. Uporabljali smo globine preiskovanja od 1 do 10.

3 OPTIMIZACIJA DOLŽIN CELIH POTI

Najprej smo izmerili, kakšno povprečno dolžino poti dosegajo nespremenljive globine preiskovanja. Rezultati so zbrani v tabeli 1. Poleg dolžine poti smo izmerili tudi povprečno število preiskanih stanj na premik, ki je dobra ocena za porabljen procesorski čas.

d	Dolžina poti	Št. stanj / premik
1	2.074,6	7,9
2	999,6	60,8
3	631,8	158,6
4	474,2	295,6
5	518,7	493,9
6	450,5	702,4
7	425,4	952,4
8	400,4	1.262,4
9	341,6	1.544,3
10	350,7	1.890,5

Tabela 1. *Povprečna dolžina poti in povprečno število preiskanih stanj na premik pri nespremenljivi globini preiskovanja.*

Rezultati v tabeli 1 so v skladu s pričakovanji: s povečano globino preiskovanja se najdene poti krajšajo. Blago patološko obnašanje je opaziti pri $d = 5$ in $d = 10$, kjer se v primerjavi s plitvejšim preiskovanjem poti podaljšajo, vendar so razlike majhne.

Najenostavnejši način določanja optimalne globine preiskovanja je prav gotovo izračun dolžine poti med vsakima dvema stanjema z vsemi možnimi globinami (pri nas od 1 do 10). Če pri iskanju poti uporabimo najboljše preizkušene globine, je povprečna dolžina najdenih poti 310,7 (11,4 % manj kot pri $d = 10$), povprečno število preiskanih stanj na premik pa 1.316,6 (30,3 % manj kot pri $d = 10$). Na tak način korist od vnaprejšnjega določanja optimalne globine preiskovanja očitno ni velika.

Če poznamo optimalno globino preiskovanja za vsako stanje, lahko pri iskanju poti globino spremenimo tudi vsak premik: ko je agent v stanju s_c, uporabi globino, ki bi bila optimalna, če bi v stanju s_c iskanje začel. Povprečna dolžina poti se na ta način skrajša na 178,3 (49,1 % manj kot pri $d = 10$), povprečno število preiskanih stanj na premik pa na 1.192,2 (36,9 % manj kot pri $d = 10$). Ti rezultati so precej bolj spodbudni in kažejo, da bi bilo poznavanje optimalne globine preiskovanja lahko dokaj koristno. Žal pa je izračun dolžine poti za vsaj usmerjen par stanj z vsako globino časovno izjemno zahteven, saj imajo naši zemljevidi v povprečju $8{,}1 \cdot 10^7$ takih parov stanj.

4 OPTIMIZACIJA POSAMIČNIH PREMIKOV

Metoda določanja optimalne globine preiskovanja iz prejšnjega poglavja je neučinkovita, ker mora agent mnoge dele poti prepotovati večkrat. Naj bo $P_{s\,d}$ množica stanj, ki ležijo na poti od stanja $s \in S$ do s_g pri globini preiskovanja d. V tem primeru je najverjetneje pri tej globini preiskovanja znana tudi pot do s_g iz vsakega od stanj $s_{s\,d} \in P_{s\,d}$. (gotovo to ni zato, ker bi bile hevristične vrednosti drugačne, če bi se preiskovanje začelo v stanju $s_{s\,d}$), metoda iz prejšnjega poglavja pa te poti izračuna ponovno. Metodo bi bilo mogoče dopolniti, da bi opisano pomanjkljivost odpravili, vendar ta pomanjkljivost ni edina.

Druga pomanjkljivost je, da se v stanju $s \in S$ uporabi globina preiskovanja, ki je morda potrebna le na majhnem delu poti od s do s_g, na večjem pa bi se lahko uporabilo plitvejše preiskovanje, ki bi prihranilo procesorski čas. Še več, zaradi določanja optimalne globine v stanju s na podlagi cele poti od s do s_g se lahko zgodi, da je najdena pot daljša, kot če bi bila izbrana le na podlagi prvega premika iz s.

Preden predstavimo algoritem, ki odpravlja opisani pomanjkljivosti, uvedimo nekaj definicij. Naj bo A množica premikov, ki jih lahko naredi agent. Preslikava δ: $S \times A \rightarrow S$ določa stanje, v katero agent pride iz danega stanja z danim premikom. LRTA* v stanju $s \in S$ pri globini preiskovanja d izbere premik $a\,(s, d)$. Pri optimalnem premiku $a^*\,(s)$ leži stanje $s' = \delta\,(s, a^*\,(s))$ na najkrajši poti od s do s_g. Optimalna globina preiskovanja $d^*\,(s)$ je globina, pri kateri LRTA* v stanju s izbere premik $a^*\,(s)$. Če imamo na voljo le globine preiskovanja do d_{max}, je d_{max}-optimalna globina $d_{d\,max}^*\,(s)$ enaka $d^*\,(s)$, če $d^*\,(s) \leq d_{max}$, sicer pa je enaka $1 \leq d \leq d_{max}$, pri čemer je pot od stanja s do s_g, na kateri leži $s' = \delta\,(s, a\,(s, d))$, najkrajša.

Naslednji algoritem učinkovito določi d_{max}-optimalne globine preiskovanja za vsa stanja $s \in S$.

for vsako stanje $s \in S$ **do**
  if d_{max}-optimalna globina $d_{d\,max}^*\,(s)$ ni določena **then**
    $s' := s$
    repeat
      najdi $d_{d\,max}^*\,(s')$ s preiskovanjem z $d = 1 \ldots d_{max}$
      $s' := \delta\,(s', a\,(s', d_{d\,max}^*\,(s')))$
    until $d_{d\,max}^*\,(s')$ določena v prejšnjem preiskovanju (takem, ki se ni začelo v s)
  end if
end for

Problematična točka algoritma je določanje d_{max}-optimalne globine za neko stanje s. Izvede se s preiskovanjem z globinami $d = 1 \ldots d_{max}$, za katere se predlagani premik $a\,(s, d)$ primerja z $a^*\,(s)$. Seveda pa je za to $a^*\,(s)$ treba poznati. Določi se lahko z Dijkstrovim algoritmom [5], ki učinkovito izračuna dolžino poti iz vsakega stanja $s \in S$ do s_g. Vendar pa se ob tem zastavi vprašanje, zakaj sploh določati optimalno globino preiskovanja, če poznamo optimalni premik. Prvi razlog je, da je optimalna globina

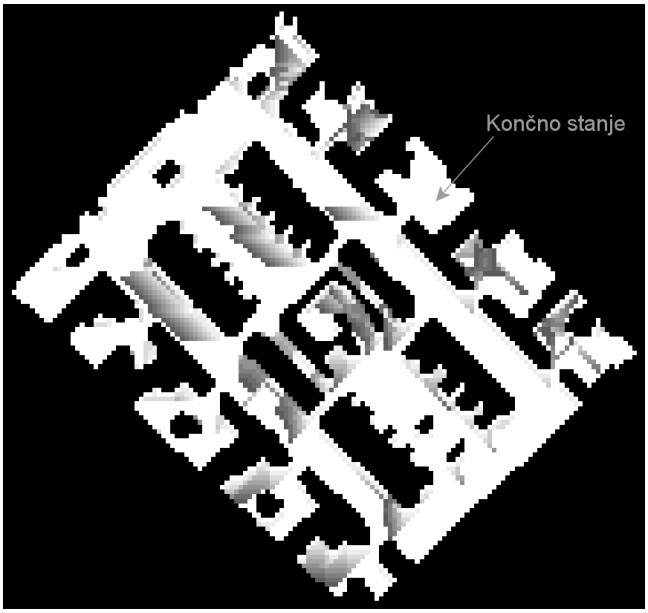
preiskovanja uporabna tudi v spreminjajočem se okolju, v kakršnem optimalni premiki lahko hitro postanejo neuporabni. Drugi pa se bo pokazal kasneje, ko bomo predstavili razširitev algoritma, ki omogoča izračun optimalne globine (in posledično optimalnega premika) le za nekatera stanja.

Povprečna dolžina poti z uporabo novega algoritma na šestih zemljevidih je 114,4, povprečno število preiskanih stanj na premik pa 155,7. To je v primerjavi z nespremenljivo globino 10 bistvena izboljšava: pri dolžini poti za 67,4 %, pri preiskanih stanjih pa kar za 91,8 %. Razlog za tolikšno zmanjšanje števila preiskanih stanj vidimo, če si ogledamo, kako pogosto je bila uporabljena katera izmed globin. Rezultati so zbrani v tabeli 2.

d_{10}*	Delež stanj (%)	d_{10}*	Delež stanj (%)
1	73.0	6	1.8
2	8.0	7	1.7
3	4.7	8	1.9
4	3.3	9	1.9
5	2.4	10	2.3

Tabela 2. *Deleži stanj, kjer je naš algoritem določil posamične globine preiskovanja za 10-optimalne.*

Slika 1 ilustrira delovanje našega algoritma: na enem izmed šestih zemljevidov so z odtenki sivine ponazorjene optimalne globine preiskovanja, črna polja pa so neprehodna.

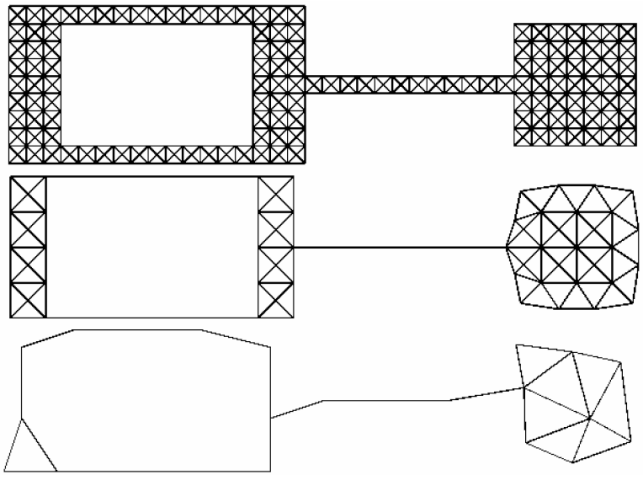


Slika 1. *Polja obarvana glede na 10-optimalno globino preiskovanja: bela ... d_{10}* = 1, najtemnejša siva ... d_{10}* = 10, črna ... neprehodno.*

5 ABSTRAKCIJA STANJ

Algoritem iz prejšnjega poglavja sicer dosega dobre rezultate in tudi omogoča bistveno hitrejše določanje optimalnih globin preiskovanja kot metoda iz poglavja 3, vendar je časovno še vedno zelo zahteven. A kot kaže slika 1, so stanja s podobno optimalno globino preiskovanja združena v skupine, iz česar je moč sklepati, da bi lahko brez prevelike izgube točnosti globino določili za le po eno stanje iz vsake skupine. Prirejanje optimalnih globin abstraktnim stanjem, ki združujejo po več konkretnih stanj, je predlagal Vadim Bulitko [9].

Uporabili smo abstrakcijo s klikami [4], ki v prvem koraku prostor razdeli na četverice popolnoma povezanih stanj (to pomeni, da je iz vsakega z enim premikom mogoče priti v vsako drugo). Kjer zaradi topologije prostora ni mogoče najti četveric, se uporabijo manjše skupine. Vsaka četverica ustreza enemu abstraktnemu stanju nivoja 1. Abstraktna stanja nivoja 1 se na enak način združijo v abstraktna stanja nivoja 2 itd. Postopek kaže slika 2, ki je povzeta po [4]. Stanja so predstavljena kot točke, možni premiki pa kot povezave med njimi.



Slika 2. *Združevanje stanj v abstraktna stanja: na vrhu konkretna stanja, na sredini abstraktna stanja nivoja 1, na dnu pa abstraktna stanja nivoja 2.*

Naš algoritem za iskanje optimalne globine preiskovanja lahko uporabi abstraktna stanja poljubnega nivoja. Iskanje poti do s_g namesto v vsakem konkretnem stanju $s \in S$ začne v vsakem stanju $s_i \in S_i$, pri čemer je S_i množica stanj i-tega abstraktnega nivoja ($S_0 = S$). Za konkretno stanje, ki leži približno na sredini stanj pripadajočih abstraktnemu stanju s_i, poišče $d_{\max}$-optimalno globino preiskovanja in jo pripiše vsem konkretnim stanjem, ki pripadajo s_i. Iskanje $d_{\max}$-optimalne globine nato ponovi šele, ko agent pride v konkretno stanje, ki pripada drugemu abstraktnemu stanju. Zaradi tega se to iskanje izvede le tolikokrat, kolikor je na zemljevidu abstraktnih stanj.

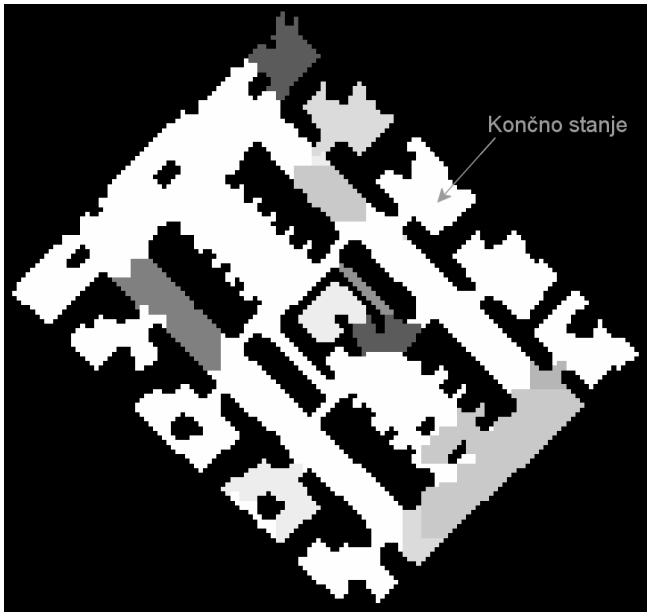
Tabela 3 kaže rezultate pri uporabi različnih nivojev abstrakcije na šestih zemljevidih. Časi za izračun 10-optimalnih globin preiskovanja za vse usmerjene pare abstraktnih stanj so ekstrapolirani iz časov izmerjenih za vsa začetna in eno končno stanje.

Nivo abs.	Št. parov abs. stanj	Čas (h)	Dolžina poti	Št. stanj / premik
0	$8{,}1 \cdot 10^7$	12.231	114,4	155,7
1	$7{,}5 \cdot 10^6$	788	130,0	158,7
2	$9{,}7 \cdot 10^5$	239	323,3	89,9
3	164.971	41	352,4	152,5
4	34.225	3,9	463,1	59,6
5	7.048	0,86	470,0	70,2
6	1.469	0,50	660,0	32,8
7	393	0,14	944,5	25,6
8	100	0,019	1004,4	13,6
9	20	0,0056	1015,3	17,0

Tabela 3. *Povprečno število usmerjenih parov stanj, čas za izračun 10-optimalnih globin preiskovanja, povprečna dolžina poti in povprečno število preiskanih stanj na premik pri uporabi različnih nivojev abstrakcije.*

Iz tabele 3 je razvidno, da sta v našem primeru uporabna nivoja abstrakcije 4 in 5. Da bi z nespremenljivo globino preiskovanja dosegli v povprečju krajše poti, bi ta globina morala biti vsaj 6 (razvidno iz tabele 1). V primerjavi z nespremenljivo globino 6 pa se pri uporabi 10-optimalnih globin preišče le 10,0 % (abstraktni nivo 4) ali 8,5 % (abstraktni nivo 5) stanj na potezo, kar pomeni sorazmeren prihranek procesorskega časa.

Slika kaže 10-optimalne globine preiskovanja za zemljevid s slike 1, le da so globine določene na podlagi abstraktnih stanj nivoja 5.



Slika 3. *Polja obarvana glede na 10-optimalno globino preiskovanja določeno na podlagi abstraktnih stanj nivoja 5: bela ... $d_{10}^* = 1$, najtemnejša siva ... $d_{10}^* = 10$, črna ... neprehodno.*

6 ZAKLJUČEK

Opisali smo dve metodi za določanje optimalnih globin preiskovanja za uporabo z algoritmom LRTA*. Druga dosega v povprečju krajše poti, pa tudi določanje globin je hitrejše. Kljub temu je za iskanje poti po zemljevidih iz komercialnih računalniških iger prepočasna. Uporabna pa postane, če jo združimo s tehniko za združevanje stanj v abstraktna stanja. To nam omogoči, da optimalno globino preiskovanja izračunamo za manjše število stanj, kar zmanjša potrebni procesorski čas z nekaj tisoč na nekaj ur. Povprečne dolžine poti najdenih z uporabo tako izračunanih globin so primerljive z dolžinami poti najdenih z uporabo nespremenljive globine preiskovanja, vendar je povprečno število preiskanih stanj na premik okrog desetkrat manjše.

V prihodnje je predvsem potrebno našo metodo preizkusiti na bolj reprezentativnem vzorcu parov stanj, kajti do sedaj je bile preizkušena le za eno končno stanju na vsakem zemljevidu. Poleg bi veljalo izboljšati način za izbiro konkretnega stanja, na podlagi katerega se določi optimalna globina preiskovanja za neko abstraktno stanje.

Literatura

[1] Bulitko, V. (2003). Lookahead pathologies and meta-level control in real-time heuristic search. V zborniku 15th Euromicro Conference on Real-Time Systems, Porto, Portugalska, 13-16.

[2] Bulitko, V., in Lee, G. (2006). Learning in real-time search: A unifying framework. Journal of Artificial Intelligence Research, 25, 119-157.

[3] Bulitko, V., Li, L., Greiner, R., in Levner, I. (2003). Lookahead pathologies for single agent search. V zborniku International Joint Conference on Artificial Intelligence (IJCAI), Posters Section, Acapulco, Mehika, 1531-1533.

[4] Bulitko, V., Sturtevant, N., in Kazakevich, M. (2005). Speeding up learning in real-time search via automatic state abstraction. V zborniku National Conference on Artificial Intelligence (AAAI), Pittsburgh, ZDA, 1349-1354.

[5] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1, 269-271.

[6] Hart, P. E., Nilsson, N. J., in Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4 (2), 100-107.

[7] Korf, R. E. (1990). Real-time heuristic search. Artificial Intelligence, 42 (2, 3), 189-211.

[8] Luštrek, M. (2005). Pathology in single-agent search. V zborniku 8. mednarodne multikonference Informacijska družba, Ljubljana, Slovenija, 345-348.

[9] Luštrek, M., in Bulitko, V. (2006). Lookahead pathology in real-time path-finding. V zborniku National Conference for Artificial Intelligence (AAAI), Learning for Search Workshop, Boston, ZDA, 108-114.