Algoritmul de găsire de drum LPA\*

Dolete Anton, GRUPA 411-EIA

Lifelong Planning A\* = algoritmul A\* dar în care refolosim informaţii din căutările anterioare

**Introducere**

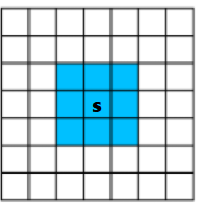
Inteligenţa artificială s-a preocupat de automatizarea unor sarcini de căutare de drumuri în areale vaste. Majoritatea eforturilor s-au concentrat pe îndeplinirea unor sarcini de căutare fixe, cu un singur scop/rezolvare într-un **câmp de căutare fix, cunoscut în întregime de la bun început.** Totuşi, realitatea fizică este într-o permanentă schimbare, deci algoritmii de căutare ar trebui resetaţi de fiecare dată când se detectează o schimbare în areal; asta presupune fie un timp de căutare mai îndelungat fie resurse de calcul mult mai performante.

La schimbarea arealului, algoritmii de căutare obişnuiţi recalculează de la zero, algoritmii cu căutare incrementală analizează noul areal şi îşi concentrează efortul doar pe acele zone diferite, nu pe întreg arealul. Practic ele refolosesc părţile comune dintre vechiul şi noul areal, economisind timp şi putere de calcul în găsirea noului drum. LPA\* este un astfel de algoritm adaptiv la schimbări de areal.

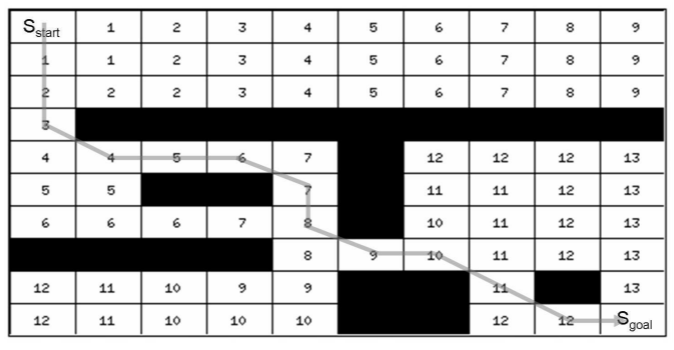
**Descrierea funcţională a algoritmului LPA\***

Acest algoritm rezolvă următoarea problemă de căutare, aplicată unui graf a cărui cost de parcurgere creşte sau scade în timp. Denumim cu S setul de noduri ale grafului, formele anterioare ale grafului le denumim cu P(s), formele viitoare ale grafului cu V(s) şi costul de a ne muta de la punctul s la punctul s1 cu c(s,s1).

LPA\* va determina **cea mai scurtă cal**e de la nodurile start la finish(**cel mai mic cost** de parcurgere a grafului), ştiind topologia grafului şi costurile de parcurgere dintre noduri. O vizualizare simplă a costurilor de parcurgere ar fi împărţirea arealului într-o matrice conectată pe 8 direcţii, astfel că de la un nod de pornire numit s putem pleca în orice direcţie:



Costul de deplasare în orice direcţie îl vom considera 1; dacă în schimb una din căsuţe era imposibil de accesat şi parcurs, costul ei îl vom considera infinit. În figura de mai jos se observă punctul de start şi punctul ţintă; numerele din căsuţe reprezintă costul de a ajunge din punctul de start până la acea căsuţa ca fiind numărul de mutări.



În primă fază, LPA\* porneşte asemeni cu algoritmul A\*. Îşi calculează costul ca fiind o sumă de distanţa parcursă până acum şi distanţa rămasă până la ţintă. Notăm cu **F costul cel mai mic dintre nodul curent şi nodul următor**,

F(n) = G(n)+RHS(n) unde

**G este costul mutării de la start la alt nod** iar

**RHS este euristica** (calea estimată) **pentru nodul curent când se schimbă arealul.**

Cu excepţia nodurilor de start şi de ţintă, fiecare nod are predecesor şi succesor. Orice nod a cărui colţ „indică” spre nodul X este un predecesor a lui X, orice nod mai aproape de ţintă decât X este un succesor a lui X.

Pentru fiecare nod ţinem cont de cele două costuri, G şi RHS. **Dacă G=RHS atunci nodul este consistent.** Dacă toate nodurile sunt consistente, LPA\* va determina calea cea mai scurtă în exact aceiaşi manieră precum A\*.

**Rularea algoritmului**

Prima iteraţie a algoritmului este identică şi pentru LPA\* şi pentru A\*; în fond marele avantaj al LPA\* este evident doar la schimbarea arealului. Algoritmul începe prin setarea nodului de start ca având RHS = 0 şi G = infinit. Restul nodurilor au G = RHS = infinit până când sunt parcurse. În acest moment doar nodul de start se află în stiva de priorităţi căci este singurul neconsistent.

Valorile nodurilor sunt calculate(expandate) după cum urmează:

* Dacă G=RHS nodul este consistent şi va fi scos din stivă
* Dacă RHS < G, G este adus la valoarea lui H şi nodul este scos din stivă
* Dacă RHS > G, G este făcut infinit. Dacă după această operaţie nodul este consistent, el este scos din stivă. Dacă nu, cheia lui este recalculată.

Expandarea continuă cu următorul nod din vârful stivei până când se întrunesc simultan condiţiile:

* Nodul ţintă devine consistent
* Nodul din vârful stivei are o cheie mai mare sau egală decât cheia nodului ţintă

Nodurile sunt puse într-o **stivă de priorităţi**; prioritatea fiecărui nod se calculează după o cheie de sortare bidimensională:

k1(n) = [minim(g(n),rhs(n))+h(n,ţintă)] (corespunde valorilor din A\*)

k2(n) = minim(g(n),rhs(n))

După ce expandarea s-a terminat, este evaluată cea mai scurtă cale. Se începede la nodul ţintă şi pornim de la predecesorul nodului curent a cărui cost total este minim. Dacă mai multe noduri au acelaşi cost, oricare dintre ele reprezintă o cale de cost minim, putem alege oricare dintre ele. Repetăm acest pas până la nodul de start. Dacă costul este infinit, atunci nu există soluţie.

**Proprietăţi interesante ale LPA\***

* Cu cât euristica fiecărui nod este mai bună(estimarea costului este mai aproape de adevăr), cu atât numărul de noduri ce trebuie expandate scade, deci orice date iniţiale despre areal, dacă se cunosc, scad timpul de calcul;
* Modul în care este implementată stiva cu priorităţi are un impact major asupra cerinţelor de calcul a algoritmului;
* Va oferi întotdeauna soluţia de cost minim;

**Avantajele/dezavantajele LPA\***

Până în acest punct, orice algoritm ar ajunge la soluţia ideală, de cost minim, odată ce se cunoaşte costul pentru fiecare nod al grafului, similar cu algoritmul A\*. Considerăm totuşi că nodurile de start şi ţintă sunt fixe. Când arealul se schimbă şi un nod nu mai este constant, el este plasat într-o stivă nouă spre a fi recalculat.

Aici intervine marea diferenţă dintre LPA\* si A\*. La schimbarea arealului, LPA\* analizează toate nodurile afectate de schimbare, nu întreg arealul. Valorile RHS sunt recalculate, **nodurile consistente sunt** **eliminate** din stiva de căutare, nodurile care **NU** **mai sunt consistente** sunt **adăugate pe stivă**, iar **cheile** nodurilor rămase neconsistente **sunt recalculate.**

Acest lucru este foarte important pentru că:

- atât A\* cât şi LPA\* au cerinţe de memorie foarte mari; practic trebuie să stocăm întreg arealul în memorie; cum resursele sunt întotdeauna limitate faţă de nevoile ideale, arealul în care robotul îşi cauta calea este mult mai redus decât arealul fizic în care se află. Prin recalcularea doar a nodurilor schimbate, LPA\* oferă răspunsul mult mai repede decât A\***.**

- **LPA\* scade mult timpul de căutare** în situaţiile în care mare parte din areal **rămâne neschimbat,** însă dacă **arealul este foarte diferit, LPA\* este mai puţin eficient decât A\*** căci va căuta în continuare soluţii pe lângă nodurile neschimbate.

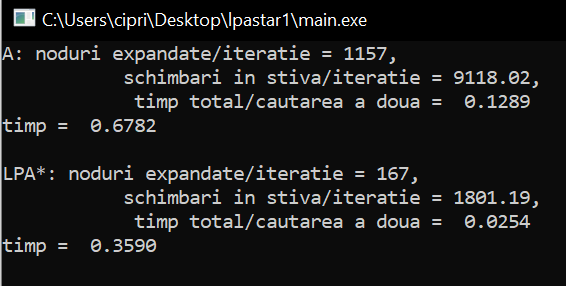
**Cod şi implementare**

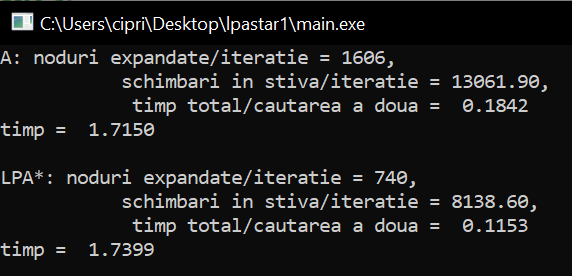
Aplicaţia consideră un areal cu mărimea introdusă de utilizator; în acest caz de 100x100 noduri; după prima iteraţie arealul îşi schimbă un procent (ales de utilizator) din noduri. Valorile iniţiale ale nodurilor şi nodurile schimbate sunt alese aleator. După rularea algoritmilor se arată nodurile schimbate şi durata găsirii traseului.

Pentru a testa cele enunţate s-a exemplificat funcţionarea algoritmului LPA\* prin următorii paşi:

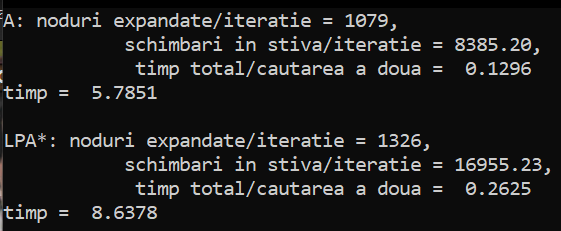
* Se creează un areal random, se rulează A\* şi LPA\*
* Se face o schimbare în areal, se rulează iar A\* şi LPA\*
* Se compara performanţa în timp, necesarul de memorie(stivă) şi numărul de noduri expandate

În cazul în care se produc schimbări nesemnificative în areal, de doar 0.5% din noduri fiind schimbate, LPA\* are un avantaj uriaş:



În schimb, dacă schimbăm 20% din areal, avantajul se reduce

La un număr de 70% dintre noduri schimbate LPA\* şi-a pierdut total avantajul. Pe măsură ce arealul diferă din ce în ce mai mult, LPA\* oferă rezultatul mai târziu şi cu cerinţe de memorie şi mai mari decât A\*.



**Concluzii**

S-a demonstrat abilitatea LPA\* de a rezolva mai bine schimări în areal atunci când ele sunt minore, însă este nepotrivit pentru schimbări semnificative căci urmăreşte să expandeze nodurile apropiate de fosta traiectorie. Dezavantajul cerinţelor mari de memorie şi faptul că LPA\* trece de două ori prin fiecare nod(faţă de A\* care trece doar odată) şi implementarea dificilă sunt contraargumentele în folosirea acestui algoritm decât în situaţii speciale în care se aşteaptă schimbări mici în areal.

Surse şi bibliografie

AI Magazine Volume 25 Number 2 (2004) (© AAAI): Incremental Heuristic Search in AI by Sven Koenig, Maxim Likhachev, Yaxin Liu, and David Furcy

<https://papers.nips.cc/paper/2001/file/a591024321c5e2bdbd23ed35f0574dde-Paper.pdf>

[https://en.wikipedia.org/wiki/Lifelong\_Planning\_A\*](https://en.wikipedia.org/wiki/Lifelong_Planning_A*)

<https://dl.acm.org/doi/10.1016/j.artint.2003.12.001>

<https://www.slideserve.com/klaus/lifelong-planning-a>

<http://theory.stanford.edu/~amitp/GameProgramming/Variations.html>

Sursa codului c++: <http://idm-lab.org/project-a.html> (ajustat şi modificat)

RESURSE

[https://en.wikipedia.org/wiki/Lifelong\_Planning\_A\*](https://en.wikipedia.org/wiki/Lifelong_Planning_A*)

https://github.com/zhm-real/PathPlanning/tree/master/Search\_based\_Planning/Search\_2D