Algoritmi euristici de explorare

Bibliografie

[1] C. Giumale – Introducere in Analiza Algoritmilor - cap. 7

[2] http://www.gamasutra.com/features/19990212/ pathdemo.zip

[3] http://www.policyalmanac.org/games/aStarTutorial.htm

[4] http://www.ai.mit.edu/courses/6.034b/searchcomplex.pdf

[5] Euristici interesante:
http://www.cse.sc.edu/~mgv/csce580f08/gradPres/slidingPuzzlesHeuristicsCaoGause.ppt

[6] Implementari in Python: http://faculty.tamu-commerce.edu/dharter/tamu/classes/2007fall/csci538/labs/hw2-p1-sols.pdf

Cuprins

- Explorarea spațiului stărilor problemei
- Căutări neinformate vs. Informate
- Explorare informată irevocabilă
- Explorări tentative informate
  - Explorare lacomă
  - Explorare tentativă completă
  - Explorare A*

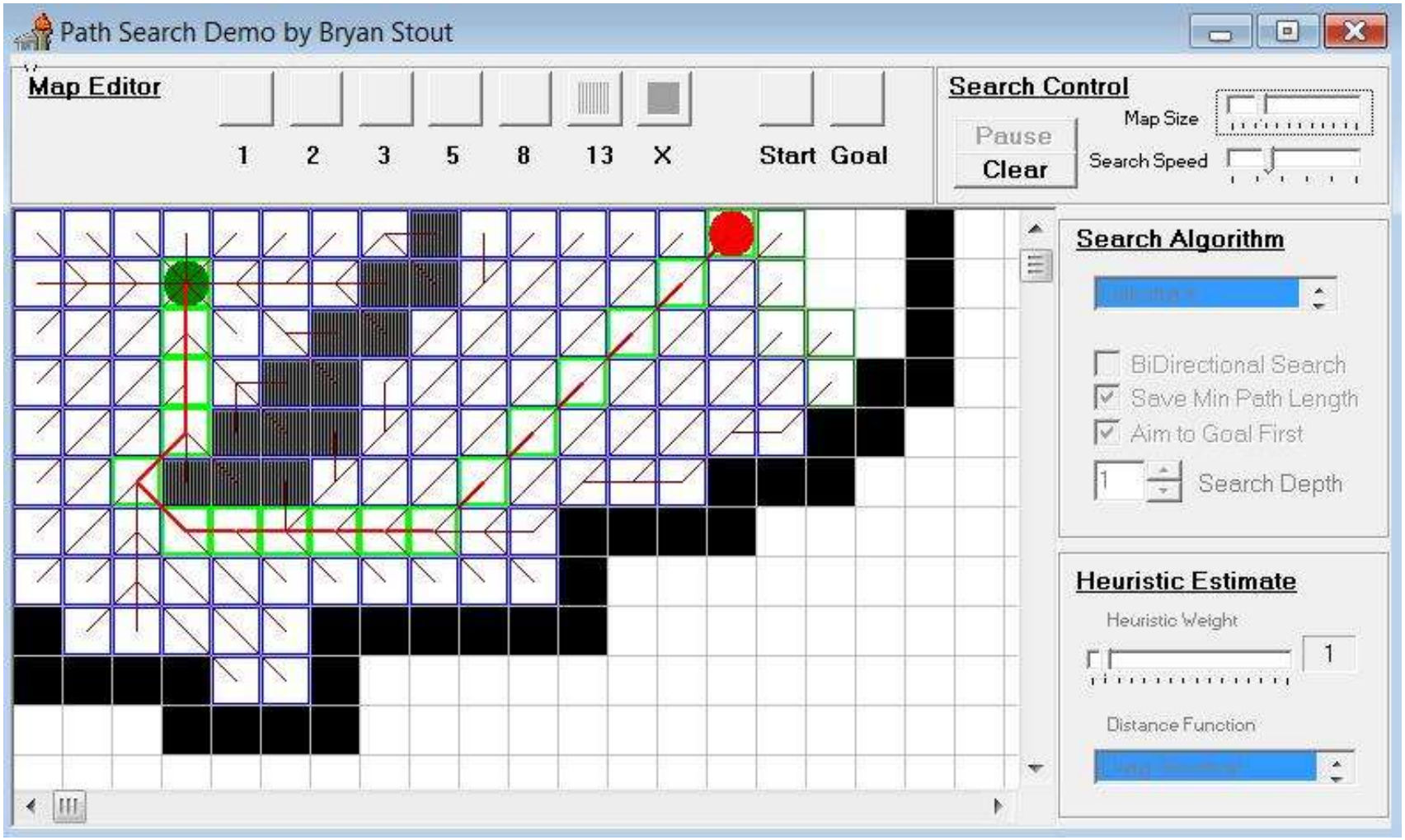
Probleme cu căutările neinformate

- Modelul unor probleme este prea complicat → variantele de rezolvare se bazează pe explorarea spațiului stărilor.

- Probleme:
  - Deseori se calculează prea mult (ex: drumul optim intre 2 puncte folosind căutarea în lățime sau Dijkstra) - ex: Dijkstra.
  - În cazul grafurilor infinite sau nedescoperite încă, algoritmii clasici fie sunt ineficienți, fie nu garantează găsirea soluției.

- Soluție:
  - Rezolvarea să nu se mai bazeze numai pe calculele exacte ci si pe experiența anterioara (eurisitici) → direcționarea căutării.

Exemplu Căutare în lățime



Explorarea spațiului stărilor problemei

Spațiul stărilor unei probleme

- Definiție: Stare a problemei = abstractizare a unei configurații valide a universului problemei, configurație ce determină univoc comportarea locală a fenomenului descris de problemă.

- Definiție: Spațiul stărilor = graf in care nodurile corespund stărilor problemei, iar arcele desemnează tranzițiile valide intre stări.
  - Caracteristică importantă: nu este cunoscut apriori, ci este descoperit pe măsura explorării!
  - Descriere
    - Nodul de start (starea inițiala);
    - Funcție de expandare a nodurilor (produce lista nodurilor asociate stărilor valide in care se poate ajunge din starea curentă);
    - Predicat de testare dacă un nod corespunde unei stări soluție.

Obiectivele navigării prin spațiul stărilor

- Cartografierea sistematică a spațiului stărilor.

- Asamblarea soluțiilor parțiale care in final conduc la soluția finală. Această soluție finală poate fi:
  - Identificarea stărilor soluției (poziționarea a n regine pe tabla de șah fără să se atace);
  - Drumul străbătut de la starea inițiala spre o stare soluție (acoperirea tablei de șah cu un cal);
  - Strategia de rezolvare = arbore multicăi in care rădăcina este starea inițială, iar frunzele sunt stări soluție. In acest arbore, unele noduri corespund unor evenimente neprevăzute care influențează calea de urmat in rezolvare (identificarea monedei false dintr-un grup de 3 monede).

Căutări informate/neinformate; Algoritmi tentativi/irevocabili

- Definiție: Dacă explorarea este ‘la întâmplare’ → algoritm neinformat.

- Definiție: Dacă explorarea se bazează pe informația acumulată in cursul explorării, informație prelucrată euristic (costuri)→ algoritm informat.

- Definiție: Dacă algoritmul de explorare are posibilitatea să abandoneze calea curentă de rezolvare si să revină la o cale anterioară → algoritmi tentativi.

- Definiție: Dacă algoritmul avansează pe o singură direcție → algoritmi irevocabili.

Căutări informate vs neinformate

- Căutările informate beneficiază de informații suplimentare pe care le colectează si le utilizează in încercarea de a ghici direcția in care trebuie explorat spațiul stărilor pentru a găsi soluția.

- Aceste informații sunt stocate:
  - In nodurile din spațiul stărilor:
    - Starea problemei reprezentată de nod;
    - Părintele nodului curent;
    - Copii nodului curent (obținuți prin expandarea acestuia);
    - Costul asociat nodului curent care estimează calitatea nodului f(n);
    - Adâncimea de explorare.
  - In structuri auxiliare pentru diferențierea nodurilor in raport cu gradul de prelucrare:
    - Expandat (închis) – toți succesorii nodului sunt cunoscuți - NEGRU
    - Explorat (deschis) – nodul e cunoscut, dar nu toți succesorii săi - GRI
    - Neexplorat – nodul nu e cunoscut - ALB

Listele CLOSED si OPEN

- OPEN = mulțimea (lista) nodurilor explorate (frontiera intre zona cunoscută si cea necunoscută) – ZONA GRI
- CLOSED = mulțimea (lista) nodurilor expandate (regiunea cunoscută in totalitate) – ZONA NEAGRA
- Explorarea zonelor necunoscute se face prin alegerea si expandarea unui nod din OPEN. După expandare, nodul respectiv e trecut (de obicei) in CLOSED.
- Majoritatea algoritmilor tentativi folosesc lista OPEN, dar doar o parte folosesc lista CLOSED.

Completitudine si optimalitate

- Definiție: Algoritm complet = algoritm de explorare care garantează descoperirea unei soluții, dacă problema acceptă soluție.
  - Algoritmii irevocabili sunt mai rapizi si consumă mai puține resurse decât cei tentativi, dar nu sunt compleți pentru că pierd informație.

- Definiție: Algoritm optimal = algoritm de explorare care descoperă soluția optimă a problemei.

Ø

Algoritm generic de explorare

- Explorare(StInit, test_sol)

  OPEN = {constr_nod(StInit)}; // starea inițiala

  Cât timp (OPEN ≠ Ø) // mai am noduri de prelucrat

  nod = selecție_nod(OPEN); // aleg un nod

  Dacă (test_sol(nod)) ***întoarce*** nod; // am găsit o soluție

  OPEN = OPEN \ {nod} U expandare{nod}; // extind căutarea

  Întoarce insucces; // nu s-a găsit nicio soluție

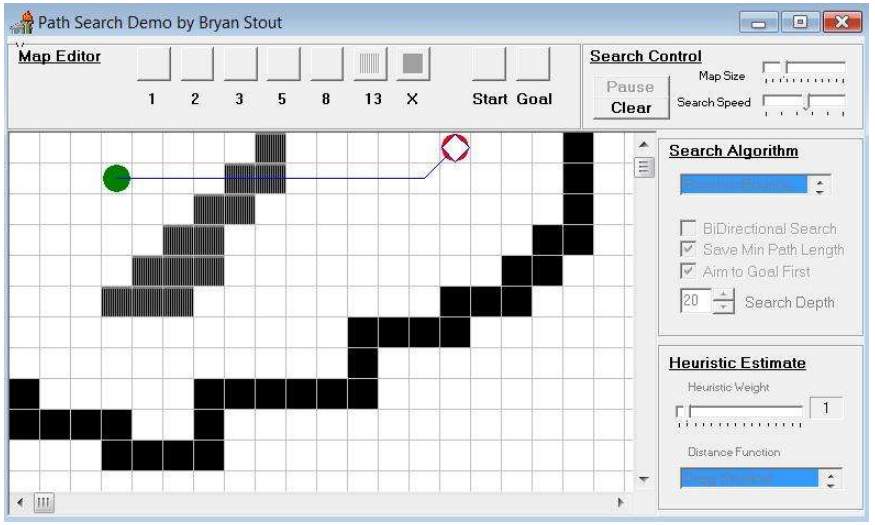
Discuție pe baza algoritmului

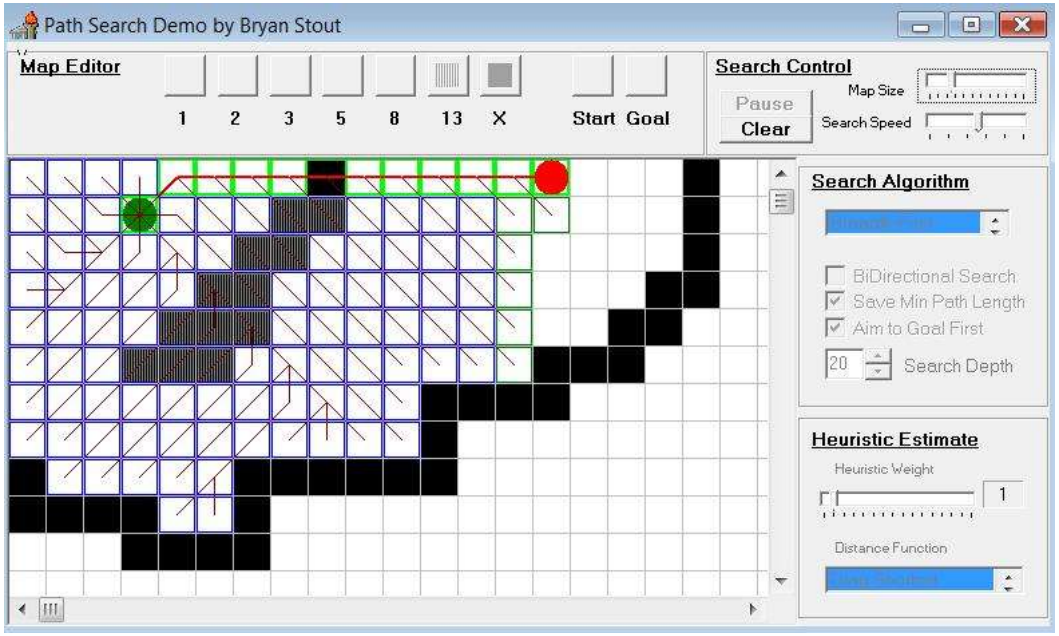
- Dacă **selecție_nod** se realizează independent de costul nodurilor din graful stărilor → căutare neinformată:
  - Dacă e de tip "random" → algoritm aleator - ex: RandomBounce
  - Dacă e de tip "primul venit, primul servit" → OPEN e coadă → Căutare în lățime - BFS – ex: Breadth-first
  - Dacă e de tip "ultimul venit, primul servit" → OPEN e stivă → Căutare în adâncime - DFS – ex: Depth-first (eventual limitat / IDDFS)

- Dacă **selecție_nod** se bazează pe un cost exact sau estimat (euristic) al stărilor problemei → căutare informată:
  - Estimarea costului si folosirea sa in procesul de selecție → esențiale pentru completitudinea, optimalitatea si complexitatea algoritmilor de explorare!

Exemplu de căutări neinformate

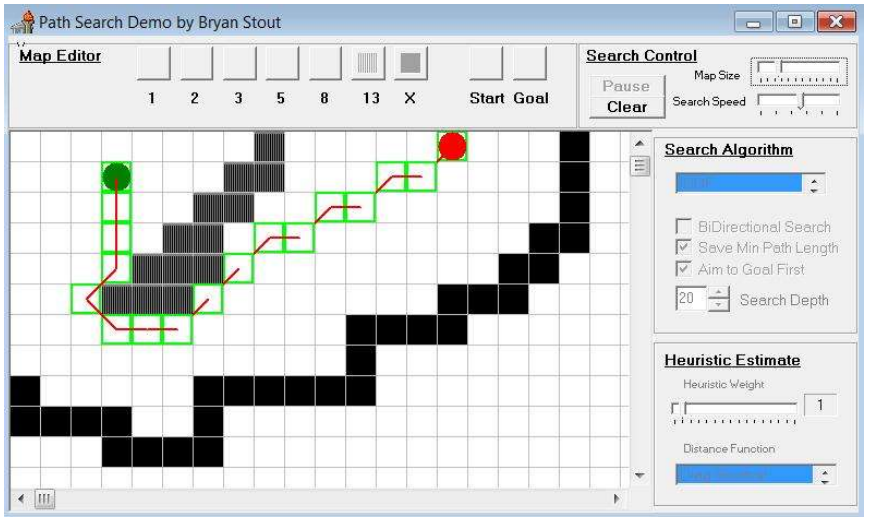
RandomBounce →





BFS ↑

IDDFS →



Explorare informată irevocabilă

Algoritm de explorare informată irevocabilă

- Ex: algoritmul alpinistului („hill climbing”) - algoritmul gradientului maxim.

- Fiecărui nod i se asociază o valoare f(nod) ≥ 0 → calitatea soluției parțiale din care face parte nodul.

- Se păstrează doar cel cu valoare maximă → OPEN are un singur element!

Gradientul Maxim

- Gradient_maxim(StInit, f, test_sol)

```
nod = constr_nod(StInit); // starea inițial
π(nod) = null;                                          Inițializări

Cât timp (!test_sol(nod))                               Testez soluția
    succs = expandare(nod); // nodurile au o valoare estimata prin f
    Dacă (succs = Ø) întoarce insucces;                 Insucces
                                  // nu mai am noduri de prelucrat
    succ = selectie_nod(succs); // f(succ) = max {f(n) | n ∈ succs}
    π(succ) = nod;
    nod = succ;                                         Gasesc calea de continuat
Întoarce nod; // am ajuns la soluție                    Soluția
```

Gradientul Maxim

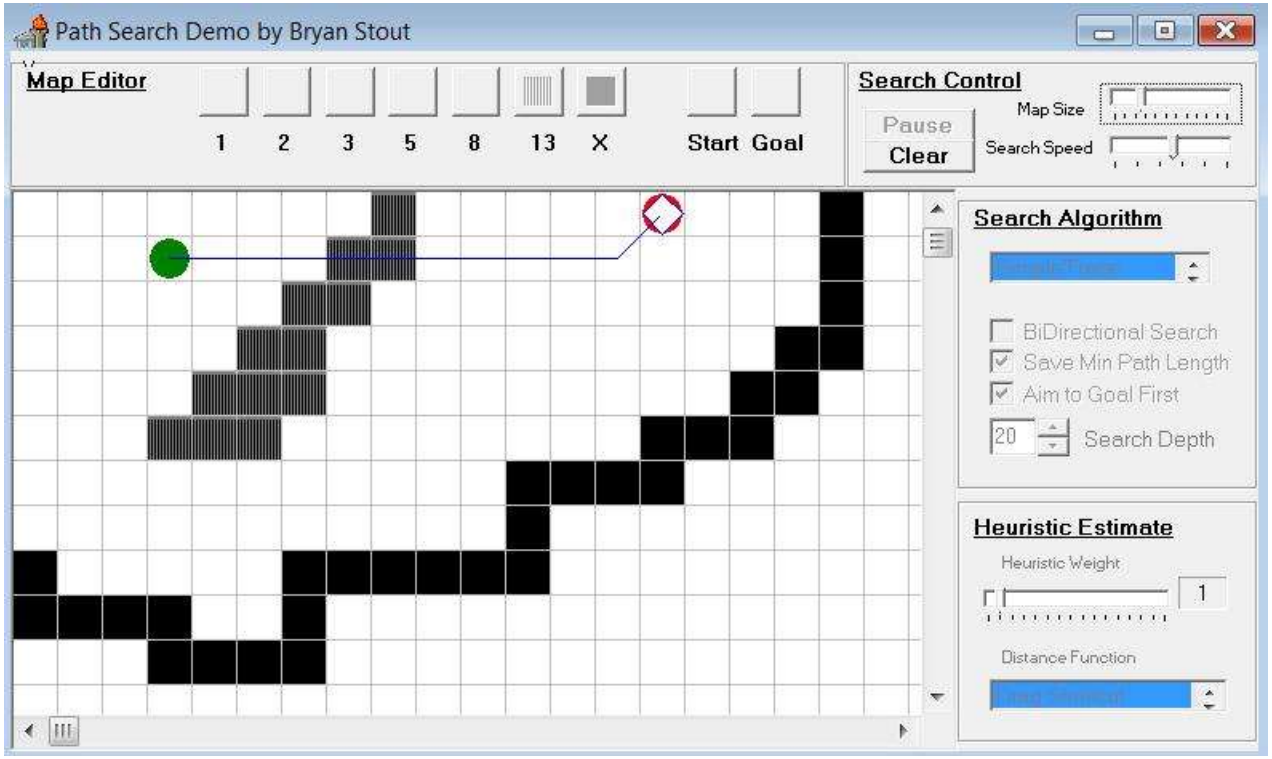
Optimalitate?

Completitudine?

Complexitate?

Ex: SimpleTrace

Exemplu Gradient Maxim



Discuție algoritmul gradientului maxim

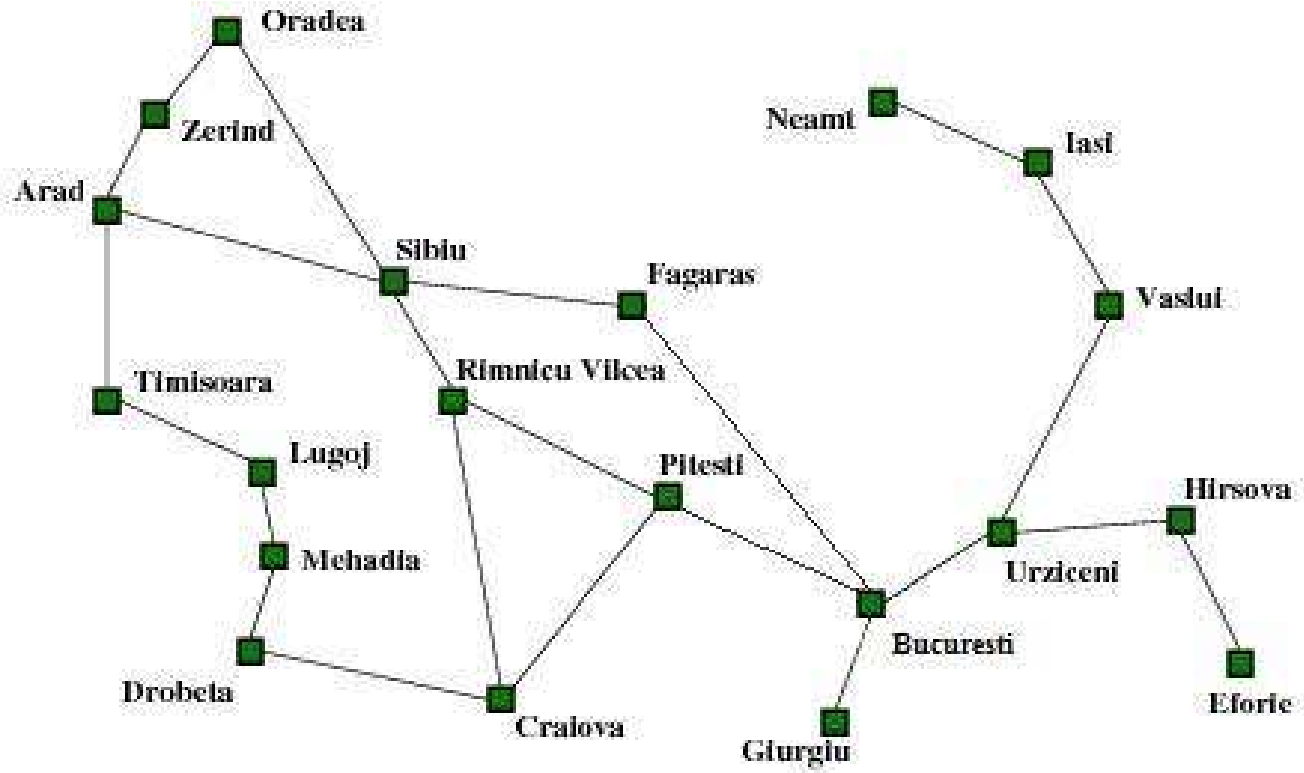
- Algoritmul nu e complet si nu e optimal!
- Complexitate scăzută: O(bd) - b = branching factor, iar d = depth!
- Performanțele algoritmului depind foarte mult de forma teritoriului explorat si de euristica folosită (de dorit să existe puține optime locale si o euristică de evaluare cat mai bună).
- Pseudo-soluție eliminare optim local: se lansează algoritmul de mai multe ori plecând din stări inițiale diferite si se alege cea mai buna soluție obținută.

Explorări tentative informate

Detalii generale

- Păstrează toate nodurile de pe frontieră (OPEN), unii păstrând si nodurile expandate (CLOSED).
- Fiecare nod are un cost asociat f(n) ≥ 0 care estimează calitatea nodului (distanța de la nodul respectiv până la un nod soluție).
- Cu cât f(n) este mai mic, cu atât nodul este mai bun.

Prezentarea problemei



- Trebuie să ajungem in București din diverse puncte ale țării pe ruta cea mai scurtă.

Explorare lacomă - Greedy

- Explorare_lacomă (StInit, f, test_sol)

```
nod = constr_nod(StInit); // starea inițiala
π(nod) = null;
OPEN = {nod};
```

Inițializări

```
Cât timp (OPEN ≠ Ø)  // mai am noduri de prelucrat
    nod = selectie_nod (OPEN); // f(nod) = min {f(n) | n ∈ OPEN}
    Dacă (test_sol(nod)) întoarce nod;
    OPEN = OPEN \ {nod}; // nodul nu e soluție, trebuie expandat
    succs = expand(nod); // expandare nod
    Pentru fiecare (succ ∈ succs)
        { OPEN = OPEN U {succ};  π(succ) = nod; } // actualizare succesori
Întoarce insucces;
```

Soluția

Continui căutarea

Insucces

Optimalitate?

Completitudine?

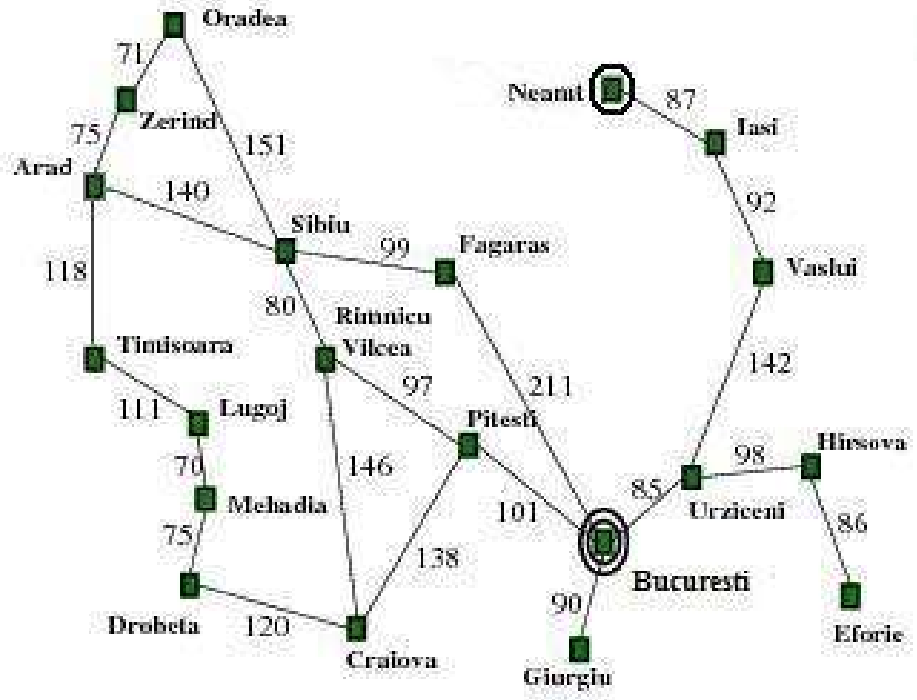
Problema?

f(nod) = distanța de
la nodul curent
până la nodul nod

f(Iași) = 87 → Iași

f(Neamț) = 87

f(Vaslui) = 92 → Neamț



- Drumul Neamț-București? → nu se termină algoritmul!
- → Explorarea lacomă nu e completă → trebuie să se rețină teritoriul deja parcurs ca să se evite ciclurile!

Explorare tentativă completă BF* (BEST FIRST) (1)

- BF*(StInit, f, test_sol)

```
nod = constr_nod(StInit); // starea inițială
π(nod) = null;
OPEN = {nod}; // noduri explorate dar neexpandate
CLOSED = Ø; // noduri expandate
```

Inițializări

```
Cât timp (OPEN ≠ Ø)
    nod = selectie_nod (OPEN); // f(nod) = min {f(n) | n ∈ OPEN}
    Dacă (test_sol(nod)) întoarce nod;
```

Soluția

```
    OPEN = OPEN \ {nod};
    CLOSED = CLOSED U {nod};
    succs = expand(nod);
```

Continuarea căutării

Explorare tentativă completă BF* (BEST FIRST) (2)

Pentru fiecare (succ ∈ succs)

Dacă (succ ∉ CLOSED U OPEN) ***atunci***

{ OPEN = OPEN U {succ}; π(succ) = nod; }

Nod nou

altfel

succ' = apariția lui succ in CLOSED U OPEN

Dacă (f(succ) < f(succ')) // am găsit o cale mai buna către succ si // redeschidem nodul

π(succ') = nod; // actualizez părintele

f(succ') = f(succ); // si costul nodului

Actualizări

Dacă (succ' ∈ CLOSED) // dacă era considerat expandat,

// redeschid

{ CLOSED = CLOSED \ {succ'}; OPEN = OPEN U {succ'}; }

Reprelucrare

Întoarce insucces;

Insucces

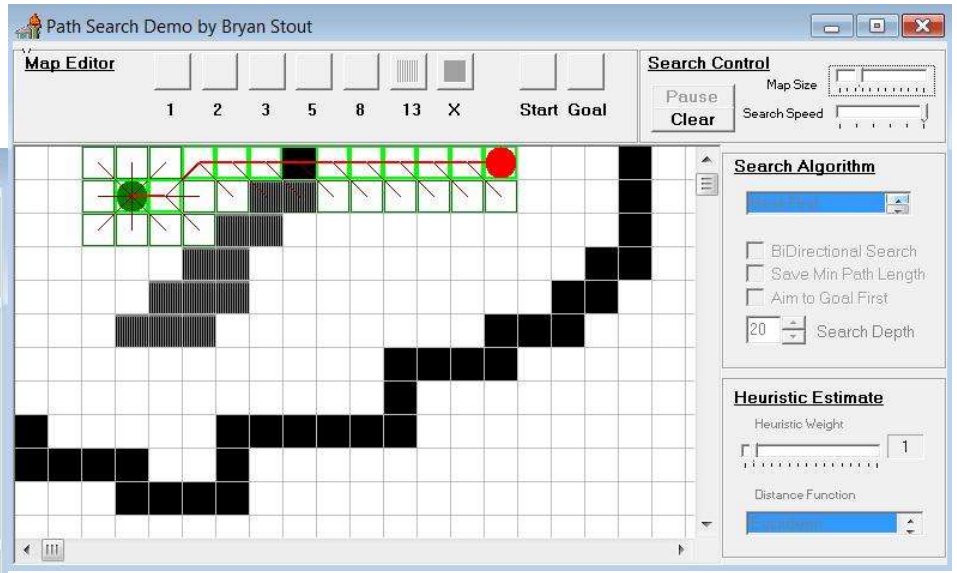
ex: Best-first cu diverse euristici

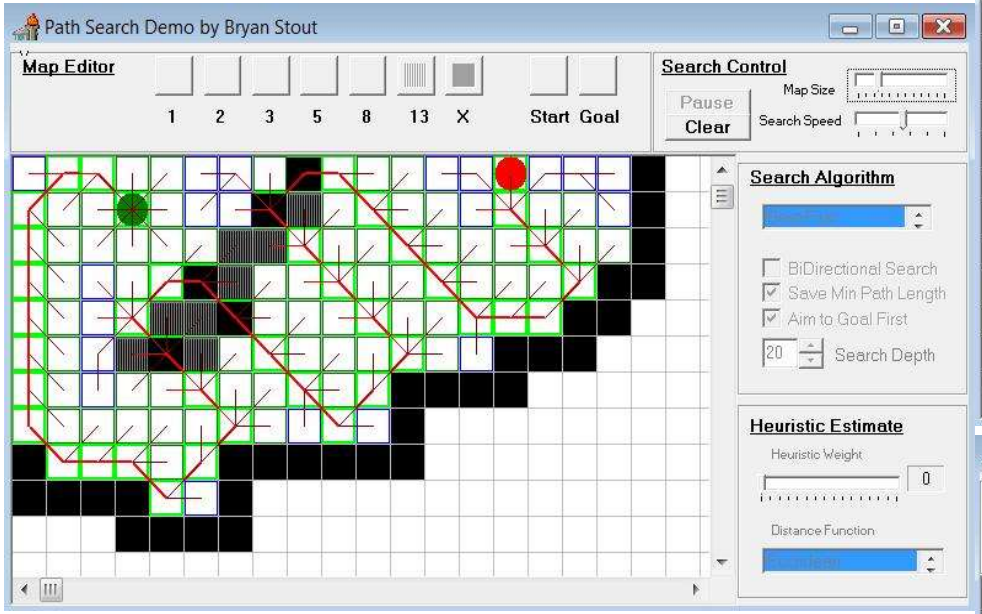
Optimalitate?

Completitudine?

Complexitate?

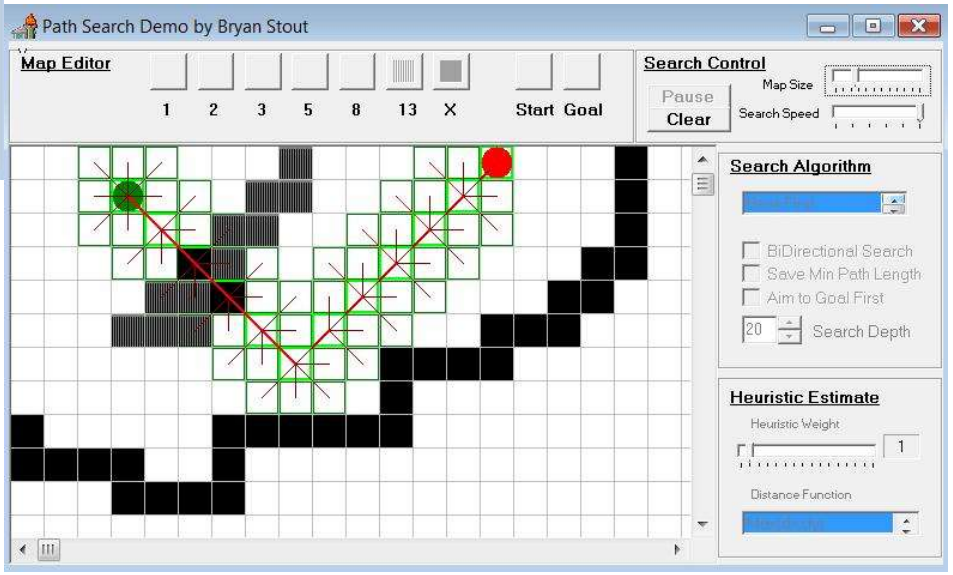
BF* - Euristică Distanta Euclidiană (11 pași, cost 23) →





BF* fără euristici ↑

BF* - Euristică maximul distantei pe x si pe y (11 pași, cost 35)→



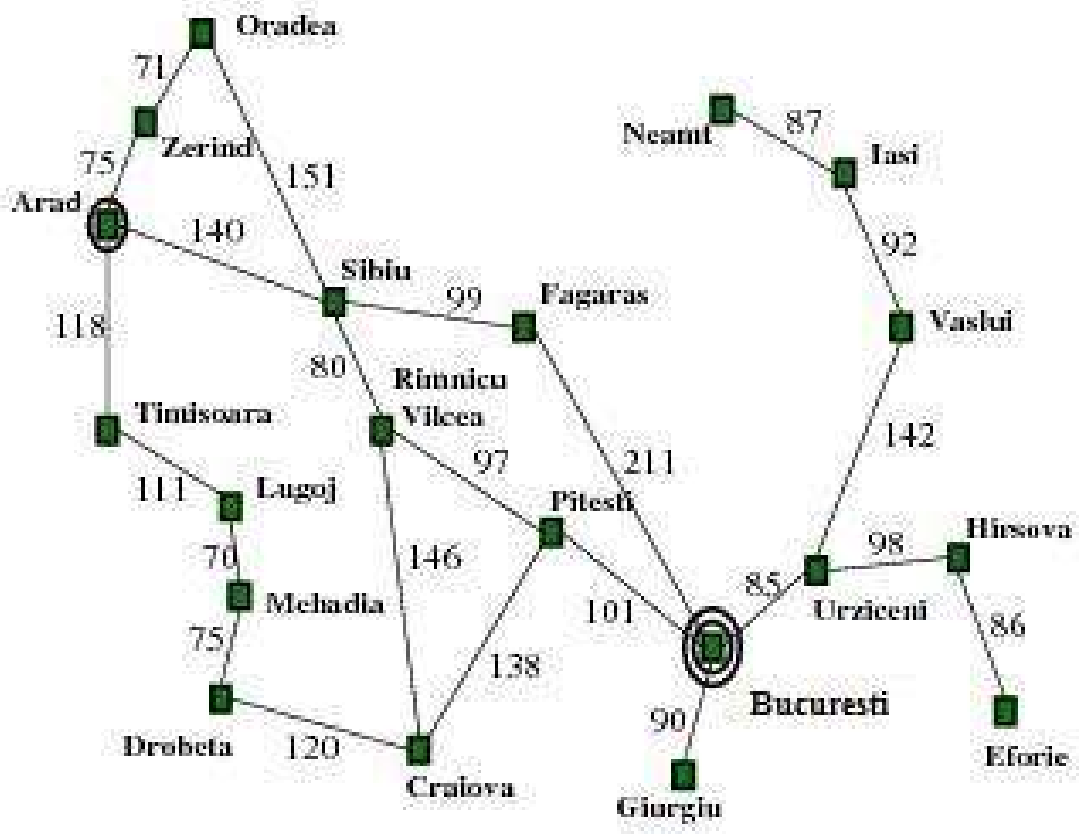
BF* - completitudine, optimalitate si complexitate

- Pastreaza intreg teritoriul explorat:
  - OPEN – nodurile de pe frontiera
  - CLOSED – nodurile expandate (unele noduri pot fi redeschise) → se evita ciclurile

- Algoritmul este complet dar nu este optim → optimalitatea impune pastrarea ordinii solutiilor si depinde de euristica f

- Complexitate: $O(b^{d+1})$

Aplicație BF*



Distanta in linie dreapta pana la Bucuresti	
Arad	366
Craiova	160
Drobeta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- Drumul optim Arad-București (f(nod) = distanța in linie dreaptă până la București)

A*

- Varianta a BF*
- Nu poate fi aplicat mereu → trebuie demonstrat ca păstrează ordinea soluțiilor unde soluțiile problemelor sunt drumuri in spațiul stărilor! (vezi Giumale pentru detalii!)
- Costul unui drum este aditiv (= suma costurilor arcelor) si crescător in lungul drumului.
- Folosește două funcții de cost:
  - h(n) - distanța estimată de la nodul curent până la nodul țintă;
  - g(n) - distanța parcursă de la nodul inițial până la nodul curent;
  - f(n) = g(n) + h(n).

Pastrarea ordinii solutiilor

- Fie f o functie de evaluare a costului nodurilor din spatiul starilor unei probleme, iar P un drum de la nodul initial n_0 la un nod n. Notam f(P|n) costul nodului n calculat in raport cu drumul P.
  - a) Spunem ca functia f pastreaza ordinea solutiilor daca, pentru oricare drumuri distinct $P_1=n_0…n'$, $P_2=n_0…n'$ si $P_3=n'..n$ avem:
    - $f(P_1|n') \leq f(P_2|n') \rightarrow f(P_1+P_3|n') \leq f(P_2+P_3|n')$
    - Extinderea unor solutii partiale cu acelasi segment conduce la drumuri ce conserva ordinea drumurilor partiale din punct de vedere al costului f
  - b) Spunem ca problema P pastreaza ordinea solutiilor daca functia exacta de cost din spatiul starilor problemei pastreaza ordinea solutiilor

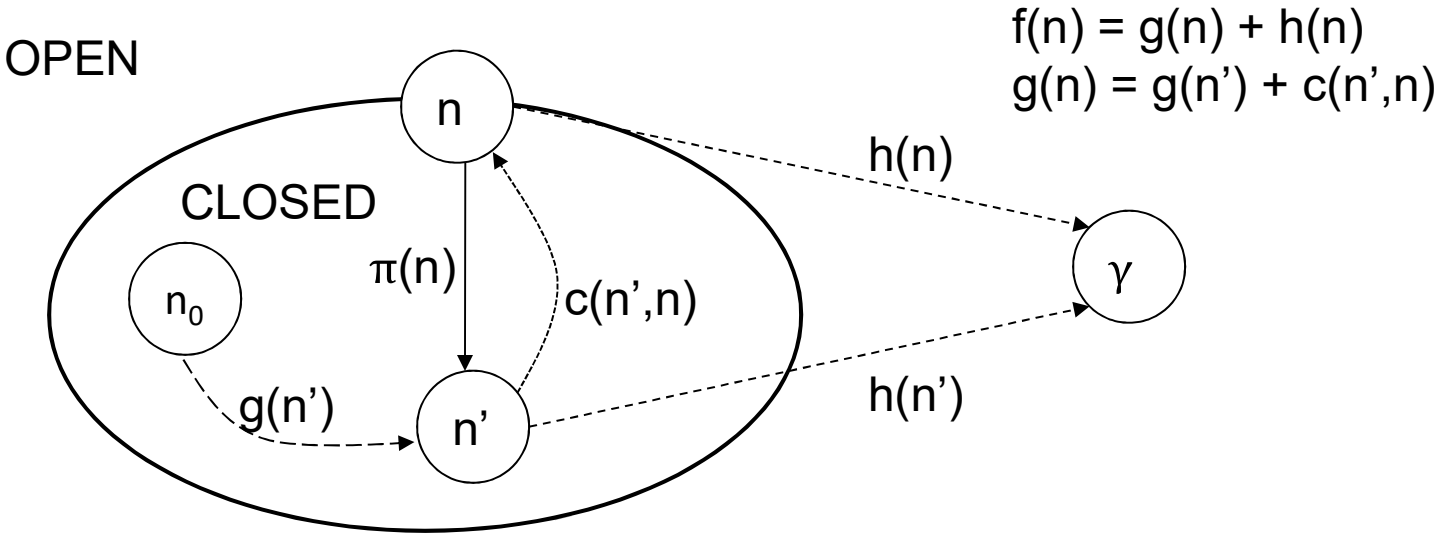
Notații (1)

- **S = (V,E)** – graful asociat spațiului stărilor problemei;
- $\mathbf{n_0}$ – nodul de start asociat stării inițiale a problemei;
- $\Gamma \subseteq \mathbf{V}$ – mulțimea nodurilor soluție. Un nod soluție se notează γ;
- **c(n,n') > 0** – costul arcului (n,n');
- π**(n)** – părintele lui n;
- **g(n)** – costul drumului n_0..n descoperit de algoritm la momentul curent de timp;
- $\mathbf{g_p(n)}$ – costul exact al porțiunii n_0..n din lungul unei căi date P;
- **g*(n)** – costul exact al unui drum optim n_0..n;

Notații (2)

- **h(n) ≥ 0** – costul estimat al drumului optim de la nodul n la cel mai favorabil nod soluție $\gamma \in \Gamma$. In plus $h(\gamma) = 0$, pentru orice $\gamma \in \Gamma$;
- **h*(n)** – costul exact al porțiunii de drum optim n.. γ, pentru cel mai favorabil nod $\gamma \in \Gamma$ ($h^*(n) = \min \{cost(n.. \gamma) | \gamma \in \Gamma\}$);
- **f(n) = g(n) + h(n)** – costul estimat al întregului drum n_0..n.. γ, pentru cel mai favorabil nod $\gamma \in \Gamma$, unde porțiunea de drum n_0..n este cea descoperita de algoritm la momentul curent de timp in cursul execuției;
- **f*(n) = g*(n) + h*(n)** – costul exact al unui drum optim n_0..n.. γ, pentru cel mai favorabil nod $\gamma \in \Gamma$;
- **C = min{f*(γ)| γ ∈ Γ}** – costul exact al unui drum optim n_0.. γ, $\gamma \in \Gamma$. (C = costul soluției optime);

Functia de evaluare A*



A* (1)

- A*(StInit, h, test_sol)

n_0 = constr_nod(StInit); // starea inițială

Inițializări

$f(n_0) = h(n_0)$; $g(n_0) = 0$; $\pi(n_0) = null$; // euristici

OPEN = $\{n_0\}$; CLOSED = $\emptyset$; // si multimi

Cât timp (OPEN ≠ Ø) // mai am noduri de prelucrat

nod = selectie_nod (OPEN); // f(nod) = min {f(n) | n ∈ OPEN}

Dacă (test_sol(nod)) ***întoarce*** nod;

Soluția

OPEN = OPEN \ {nod}; // updatez OPEN

CLOSED = CLOSED U {nod}; // si CLOSE

succs = expand(nod); // determin nodurile succesoare

Continuarea căutării

A* (2)

Pentru fiecare (succ ∈ succs) { // prelucrare succs

g_succ = g(nod) + c(nod,succ); // calculez g

f_succ = g_succ + h(succ); // calculez f = g + h

Prelucrare succesori

Dacă (succ ∉ CLOSED U OPEN) ***atunci*** // nod nou descoperit →

{ OPEN = OPEN U {succ}; g(succ)= g_succ; f(succ)= f_succ; π(succ) = nod;}

// il bag in OPEN

Nod nou

altfel // a mai fost prelucrat

Dacă (g_succ < g(succ)) { // verific daca noul g este mai mic decat // anteriorul

g(succ)= g_succ; f(succ)= f_succ; π(succ) = nod; // cale mai buna

Actualizări

Dacă (succ ∈ CLOSED) // daca era considerat expandat, il redeschid

{ CLOSED = CLOSED \ {succ'}; OPEN = OPEN U {succ'};

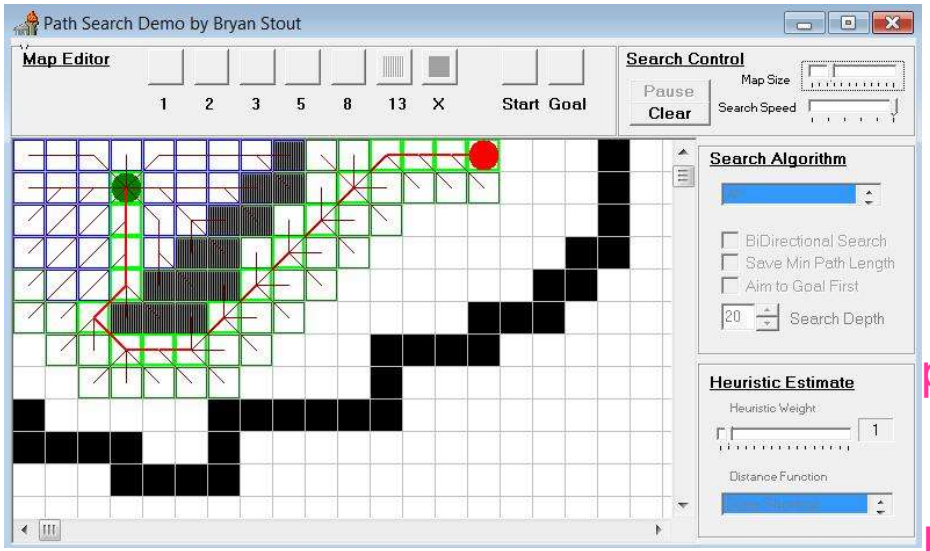
Reprelucrare

Insucces

Întoarce insucces;

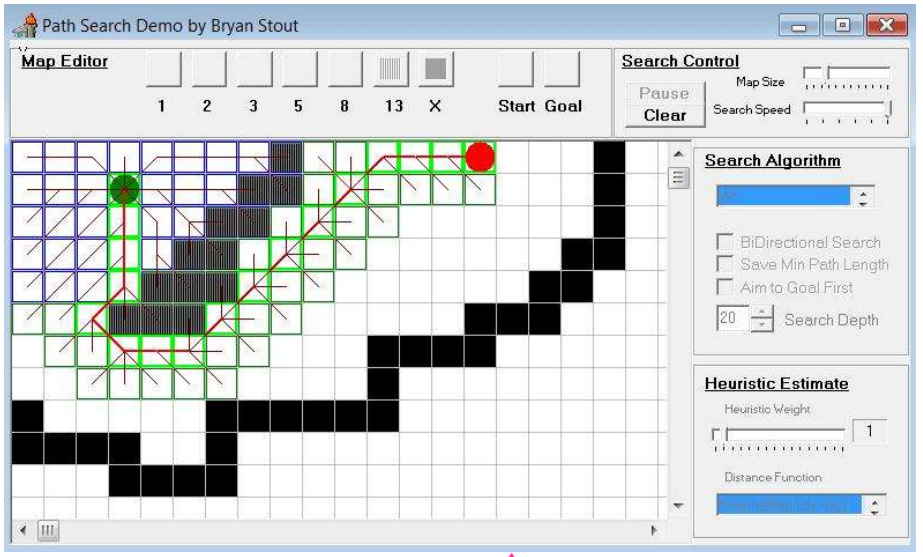
ex: A* cu diverse euristici

Exemple A* cu diverse euristici



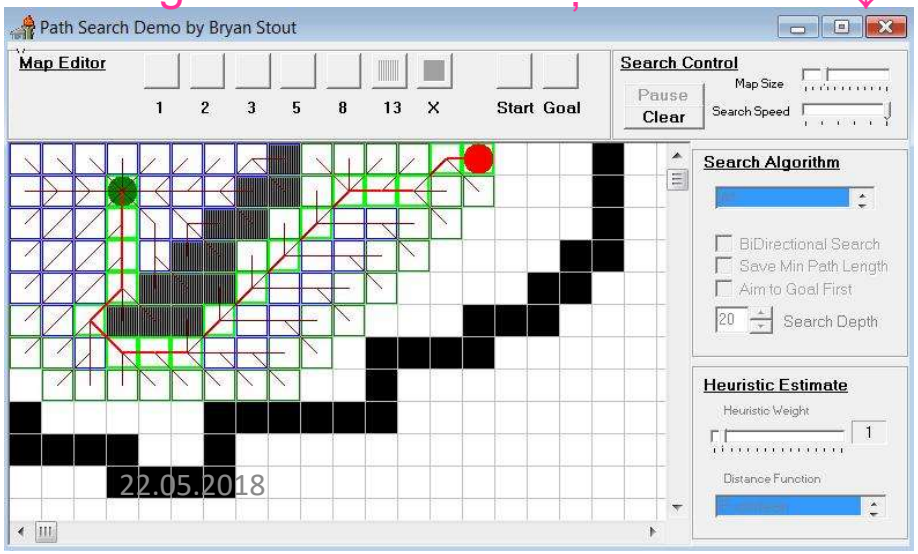
A*

16 pași



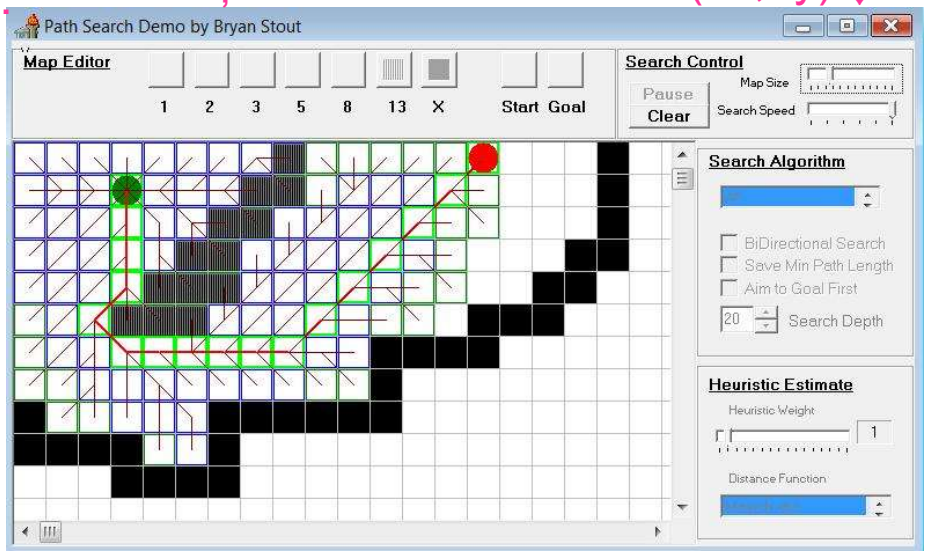
Euristici:

Diagonala↑



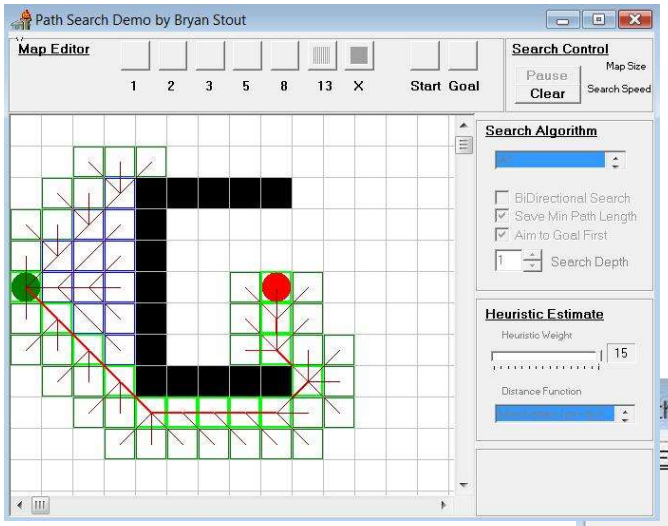
Distanța Euclidiana↓

Distanța Manhattan ↑



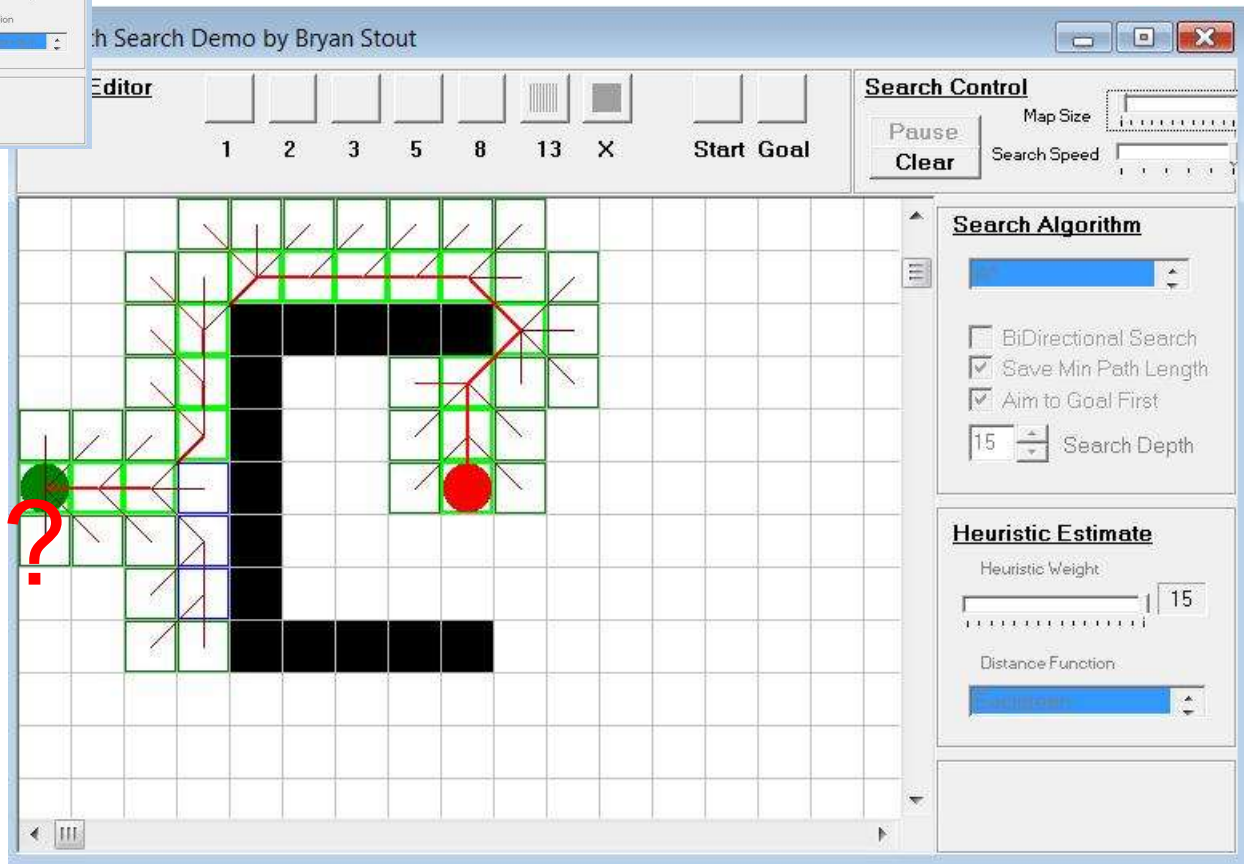
Max(dx,dy) ↓

Problema



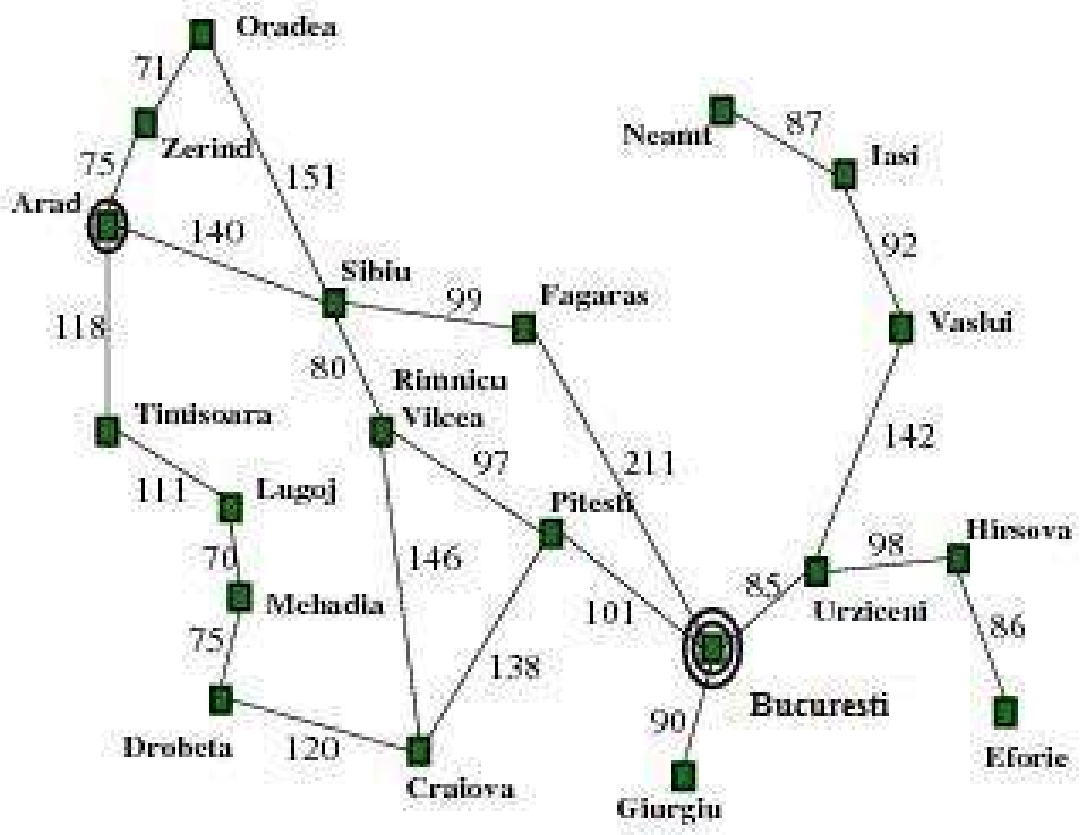
← A* – Distanța Manhattan – 12 pași

↓ A* – Distanța Euclidiana – 14 pași



Cum se explică???

Aplicație A*



Distanta in linie dreapta pana la Bucuresti	
Arad	366
Craiova	160
Drobeta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- Drumul optim Arad-București (h(n) = distanța in linie dreaptă pană la București, g(n) = distanța parcursă)

Algoritmul A* - completitudine si optimalitate (1)

- Teorema 7.1: Algoritmul A* este complet chiar dacă graful explorat nu este finit.

- Lema 7.1: Fie P = $n_0,n_1,...,n_m$ un drum oarecare in graful explorat de A*, astfel încât la un moment T al explorării toate nodurile din P sunt in CLOSED. Atunci, la orice moment de timp egal sau superior lui T, există inegalitatea $g(n_i) \leq g_p(n_i)$, i = 0,m:
  - costul nodurilor din CLOSED poate sa scadă, dar de fiecare dată când acest lucru se întâmpla, se pierde timp → scoaterea nodului din CLOSED, punerea in OPEN, prelucrarea acestuia încă o dată → trebuiesc evitate aceste situații → alegerea unei euristici cât mai bune care să minimizeze numărul acestor actualizări!

Algoritmul A* - completitudine si optimalitate (2)

- Definiție 7.2: Funcția euristică h este admisibilă dacă pentru orice nod n din spațiul stărilor $h(n) \leq h^*(n)$. Cu alte cuvinte, o euristică admisibilă h este optimistă si $h(\gamma) = 0$ pentru orice nod $\gamma \in \Gamma$.

- Teorema 7.2: Algoritmul A* ghidat printr-o euristică admisibilă descoperă soluția optimă dacă există soluții.

Algoritmul A* - completitudine si optimalitate (3)

- **Definitie**: Fie P =$n_0..n_q$ o cale de la nodul n_0 la nodul n_q in graful asociat spatiului starilor explorat de A*.
  - Calea P este **C-limitata** daca pentru orice nod n din P avem $g_p(n)+h(n) \leq C$.
  - Calea P este **C-strict-limitata** daca pentru orice nod n din P avem $g_p(n)+h(n) < C$

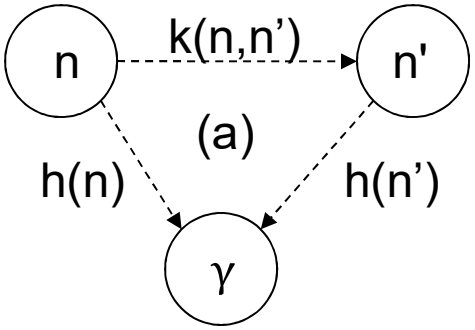
- **Teorema**: **Conditia necesara** pentru ca un nod n sa fie expandat de un algoritm A* condus de o euristica admisibila este sa existe o cale C-limitata $n_0..n$.

- **Teorema**: **Conditia suficienta** pentru ca un nod n sa fie expandat de un algoritm A* condus de o euristica admisibila este sa existe o cale C-strict-limitata $n_0..n$.

Euristici – consistență si monotonie

- Definiție 7.4: O euristică h este consistentă dacă pentru oricare două noduri n si n’ ale grafului explorat, astfel încât n’ este accesibil din n, există inegalitatea: $h(n) \leq h(n') + k(n,n')$, unde $k(n,n')$ este costul unui drum optim de la n la n’.

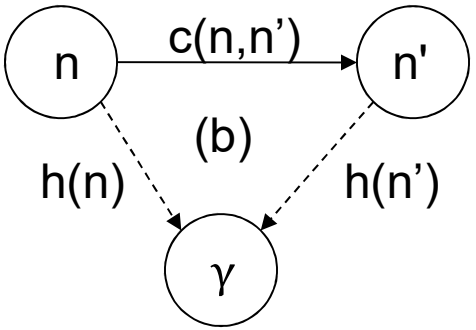
- Definiție 7.5: O euristică h este monotonă dacă pentru oricare două noduri n si n’ ale grafului explorat, astfel încât n’ este succesorul lui n, există inegalitatea $h(n) \leq h(n') + c(n,n')$, unde $c(n,n')$ este costul arcului (n,n').



$h(n) \leq h(n') + k(n,n')$

Regula triunghiului pentru euristici:

← Consistente
→ Monotone



$h(n) \leq h(n') + c(n,n')$

Consistență = monotonie

- Teorema 7.5: O euristică este consistentă ⇔ este monotonă.
  - Demonstrație:
    - h – consistentă → h – monotonă. Alegem n' ∈ succs(n) → k(n,n') <= c(n,n') → h(n) ≤ h(n') + k(n,n') ≤ h(n') + c(n,n') → h – monotonă.
    - h – monotonă → h – consistentă. Fie n = $n_1,n_2,\ldots,n_q$ = n', un drum optim n..n' cu cost k(n,n'). → h(n) = $h(n_1) \leq h(n_2) + c(n_1,n_2) \leq h(n_3) + c(n_1,n_2) + c(n_2,n_3)\ldots \leq h(n_q) + c(n_1,n_2) + c(n_2,n_3) + \ldots c(n_{q-1},n_q) = h(n_q) + k(n_1,n_q)$ → h(n) ≤ h(n') + k(n,n') → h – consistentă.

Consistență → admisibilitate

- Teorema 7.6: O euristică consistentă este admisibilă.
  - Demonstrație:
    - Fie h o euristica consistentă → $h(n) \leq h(n') + k(n,n')$, $\forall$ n' accesibil din n. Fie $n' = \gamma \in \Gamma$ → $k(n, \gamma) = \min \{ k(n, \gamma') \mid \gamma' \in \Gamma \} = h^*(n)$ → $h(n) \leq h(\gamma) + h^*(n)$, dar $h(\gamma) = 0$ → $h(n) \leq h^*(n)$ → euristică admisibilă.

- Corolar 7.2: O euristică monotonă este admisibilă.

Teoreme A*

- Fie un algoritm A* ghidat de o euristica monotona, iar $n_0,n_1,\ldots,n_q$ ordinea nodurilor expandate in cursul functionarii algoritmului. Atunci, exista relatiile $f(n_0)\leq f(n_1)\leq \ldots \leq f(n_q)$.Secventa costurilor nodurilor expandate de A* nu este descrescatoare.

- Daca A* foloseste o euristica monotona, atunci orice nod n ∈ CLOSED nu mai este redeschis de algoritm (n nu mai este transferat in OPEN).

- Daca A* este condus de o euristica monotona, atunci orice nod n ∈ CLOSED, drumul $n,\pi(n), \pi(\pi(n)), \ldots, n0$ este optim si este continut in CLOSED.

- Fie n un nod din graful explorat de A* condus de o euristica monotona.
  - a) conditia necesara pentru expandarea nodului n este: $g^*(n) + h(n) \leq C$.
  - b) conditia suficienta pentru expandarea nodului n este: $g^*(n) + h(n) < C$.

Dominanța - Definiții

- Definiție 7.6: Fie h_1 si h_2 două euristici admisibile.
  - 1. Spunem ca h_1 este mai informată decât h_2 dacă $h_2(n) < h_1(n)$ pentru orice nod $n \notin \Gamma$ din graful spațiului de stare explorat.
  - 2. Spunem ca h_1 este aproximativ mai bine informată decât h_2 dacă $h_2(n) \leq h_1(n)$ pentru orice nod $n \notin \Gamma$ din graful spațiului de stare explorat.

- Definiție 7.7: Un algoritm A_1^* domină un algoritm A_2^* dacă orice nod expandat de A_1^* este expandat si de A_2^*. (eventual, A_2^* expandează noduri suplimentare față de A_1^*, deci A_1^* poate fi mai rapid ca A_2^*.)

- Definiție 7.8: Un algoritm A_1^* domină aproximativ un algoritm A_2^* dacă orice nod expandat de A_1^* este expandat si de A_2^* cu eventuala excepție a unor noduri care satisfac condiția $h_1(n) = h_2(n) = C - g^*(n)$.

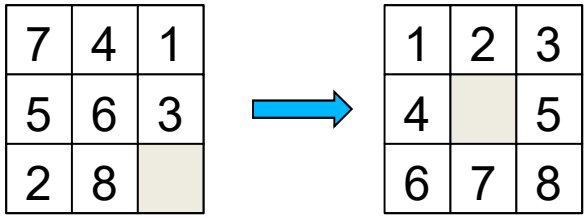
Dominanța - Teoreme

- Teorema 7.11: Dacă o euristică monotonă h_1 este mai informată decât o euristică monotonă h_2, atunci un algoritm A_1^* condus de h_1 domină un algoritm A_2^* condus de h_2.

- Teorema 7.12: Dacă o euristică monotonă h_1 este aproximativ mai bine informată decât o euristică monotonă h_2, atunci un algoritm A_1^* condus de h_1 domină aproximativ un algoritm A_2^* condus de h_2.

Dominanța - Exemplu

- Considerăm jocul 8-pătrățele care trebuie aranjat pornind de la forma inițială prin mutarea locului 'liber' astfel incat să ajungem la forma finală:



- Două euristici posibile:
  - h_1 = numărul pătrățelelor a căror poziție curentă diferă de poziția finală;
  - $h_1 = \Sigma_{p \in piese}(\delta_p)$, unde $\delta_p = 0$ dacă poziția curenta coincide cu cea finala si $\delta_p = 1$, altfel
  - h_2 = distanța Manhattan = suma distanțelor pe verticală si orizontală intre pozițiile curente ale pătrățelelor si pozițiile lor finale
  - $h_2 = \Sigma_{p \in piese}(dist\_h_p + dist\_v_p)$

Admisibilitate? Monotonie? Dominanța? Care euristică va fi aleasă pentru A*?

Complexitate A*

- Liniară dacă $|h(n) - h^*(n)| \leq \delta$, unde $\delta \geq 0$ este o constantă.
- Subexponențială, dacă $|h(n) - h^*(n)| \leq O(\log(h^*(n)))$.
- Exponențială, altfel, (dar mult mai bună decât a căutărilor neinformate).
- Mai multe explicații găsiți in Giumale 7.4.4

ÎNTREBĂRI?

Bibliografie

[1] C. Giumale – Introducere in Analiza Algoritmilor – cap. 6.1

[2] Cormen – Introducere in algoritmi – cap. 8.3

[3]http://www.soe.ucsc.edu/classes/cmps102/Spring04/TantaloAsymp.pdf

[4] http://www.mersenne.org/