Cautare euristica

Curs 6

Sumar al strategiilor de cautare

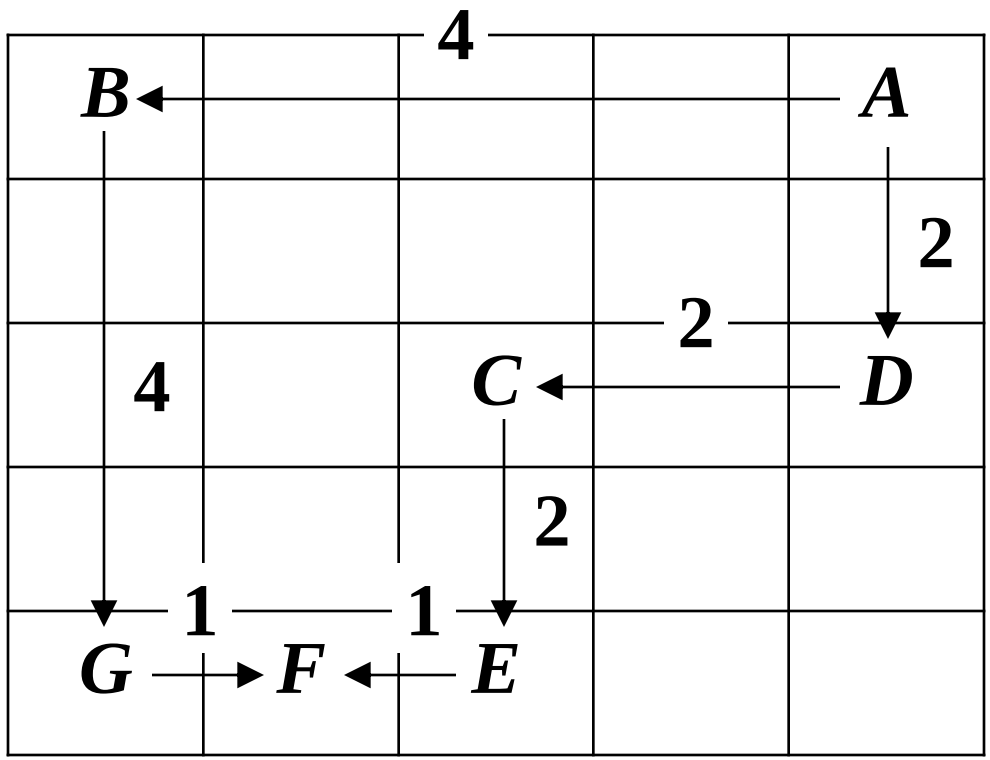
Strategie	**Selectia din frontiera**	**Garanteaza oprirea ?**	**Complexitatea spatiala**
Adancime	Ultimul nod adaugat	Nu	Liniara
Nivel	Primul nod adaugat	Da	Exponentiala
Adancime euristica	Minim local h	Nu	Liniara
Intai-cel-mai-bun	Minim global h	Nu	Exponentiala
Cost minim	Minim global g	Da	Exponentiala
A^*	Minim global f	Da	Exponentiala

Cautare euristica (engl. *heuristic searching*)

- ***Cautare euristica*** **= cautare care foloseste informatii suplimentare pentru ghidarea procesului de descoperire a unei solutii.**
- **Aceasta informatie este de forma unei functii *h*(*n*) definita pe multimea nodurilor cu valori reale pozitive astfel incat *h*(*n*) este o estimare a lungimii caii de la nodul *n* la un nod obiectiv, *h* : $\mathcal{N} \rightarrow \mathbb{R}^+$, $\mathcal{N}$ = multimea nodurilor.**
- **Functia *h* se numeste *subestimator* sau echivalent *admisibila* daca *h*(*n*) este mai mica sau egala cu lungimea celei mai scurte cai de la *n* la un nod solutie, adica $h(n) \leq h^*(n)$ pentru orice nod *n*. Daca *n* este nod obiectiv si *h* este admisibila atunci *h*(*n*) = 0.**
- **Pentru calculul lui *h*(*n*) se vor folosi informatii despre nodul *n* care sa se poata obtine cat mai usor. Exista o contradictie intre volumul de munca necesar pentru gasirea unei functii euristice si acuratetea cu care aceasta functie descrie lungimea celei mai scurte cai pana la nodul obiectiv. De obicei pentru calculul functiei euristice se folosesc atributele unui nod.**

Un exemplu de functie euristica

$$h(A) = \|AF\| = \sqrt{4^2 + 3^2} = 5$$
$$h(D) = \|DF\| = \sqrt{2^2 + 3^2} = \sqrt{13} \simeq 3.605$$
$$h(C) = \|CF\| = \sqrt{2^2 + 1^2} = \sqrt{5} \simeq 2.236$$
$$h(B) = \|BF\| = \sqrt{4^2 + 1^2} = \sqrt{17} \simeq 4.123$$
$$h(E) = \|EF\| = 1$$
$$h(G) = \|GF\| = 1$$
$$h(F) = \|FF\| = 0$$



Aceasta functie euristica, definita de distanta de la *n* la *F* este admisbila deoarece distanta euclidiana este cel mai scurt drum intre doua puncte.

Cautare intai-cel-mai-bun (engl. *best-first*)

- ***Cautare euristica*** **= cautare euristica care la fiecare pas selecteaza nodul din frontiera care pare a fi cel mai aproape de un nod obiectiv. Cu alte cuvinte se selecteaza nodul *n* cu cea mai mica valoare pentru *h*(*n*).**
- **Cautarea intai-cel-mai-bun se implementeaza tratand multimea frontiera ca o coada cu prioritati avand drept functie de cost pe *h*(*n*).**

  selecteaza(*Nod*,[*Nod* | *Frontiera*], *Frontiera*)

  ad_la_frontiera(*Vecini*,*F*1,*F*3) ←

  concat(*F*1,*Vecini*,*F*2) ∧

  sorteaza_dupa_h(*F*2,*F*3)

- [(*A*,[],5)] ⇒ [(*B*,[*A*],4.123), (*D*,[*A*],3.605)] ⇒ [(*B*,[*A*],4.123), (*C*,[*D*,*A*],2.236)] ⇒ [(*B*,[*A*],4.123), (*E*,[*C*,*D*,*A*],1)] ⇒ [(*B*,[*A*],4.123), (*F*,[*E*,*C*,*D*,*A*],0)] ⇒ ...
- **Se observa ca pe acest exemplu cautarea intai-cel-mai-bun a determinat solutia de cost minim. Cu toate acestea, cautarea intai-cel-mai-bun nu garanteaza intotdeauna gasirea unei astfel de solutii.**

Cautare in adancime euristica (engl.*heuristic depth-first*)

- ***Cautare in adancime euristica*** **= combina avantajele cautarii in adancime cu cele ale cautarii euristice. Ideea este de a face alegerea locala cea mai buna in functie de valoarea functiei euristice *h*(*n*).**
- **Cautarea in adancime euristica se implementeaza tratand multimea frontiera ca o stiva si sortand vecinii nodului selectat ininte de a-i adauga la frontiera.**

  selecteaza(*Nod*,[*Nod* | *Frontiera*], *Frontiera*)

  ad_la_frontiera(*Vecini*,*F*1,*F*3) ←

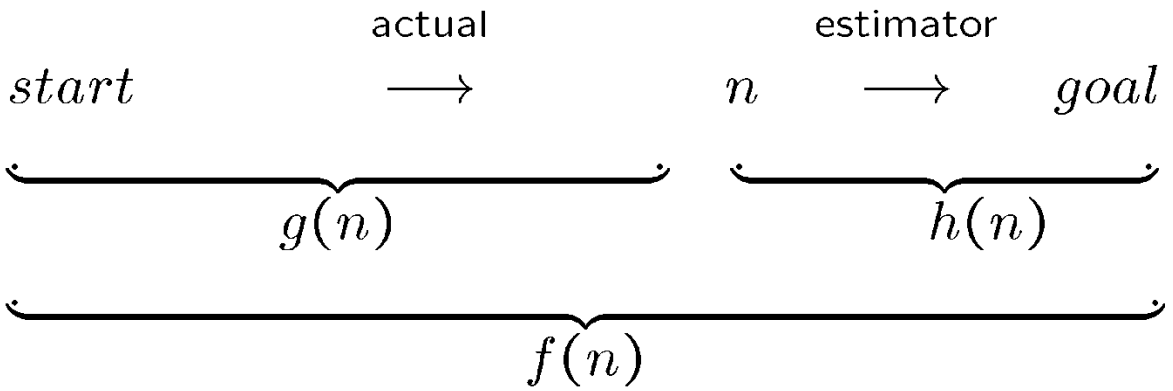
  sorteaza_dupa_h(*Vecini*, *VeciniSortati*) ∧

  concat(*VeciniSortati*, *F*1, *F*2)

- [(*A*,[],5)] ⇒ [(*D*,[*A*],3.605), (*B*,[*A*],4.123)] ⇒ [(*C*,[*D*,*A*],2.236), (*B*,[*A*],4.123)] ⇒ [(*E*,[*C*,*D*,*A*],1), (*B*,[*A*],4.123)] ⇒ [(*F*,[*E*,*C*,*D*,*A*],0), (*B*,[*A*],4.123)] ⇒ ...
- **Se observa ca pe acest exemplu cautarea in adancime euristica a determinat solutia de cost minim. Cu toate acestea, cautarea in adancime euristica nu garanteaza intotdeauna gasirea unei astfel de solutii.**

Cautare A*

- ***Cautare* A* = combina avantajele cautarii cu cost minim cu cele ale cautarii intai-cel-mai-bun. Ideea este de a combina functia *g*(*n*), care reprezinta costul caii pana la nodul *n*, cu functia euristica *h*(*n*), rezultand o noua functie de cost *f*(*n*) = *g*(*n*)+*h*(*n*).**



- **Cautarea A* se implementeaza tratand multimea frontiera ca o coada cu prioritati avand drept functie de cost pe *f*(*n*).**

  ***selecteaza*(*Nod*,[*Nod* | *Frontiera*], *Frontiera*)**

  ***ad_la_frontiera*(*Vecini*,*F1*,*F3*) ←**

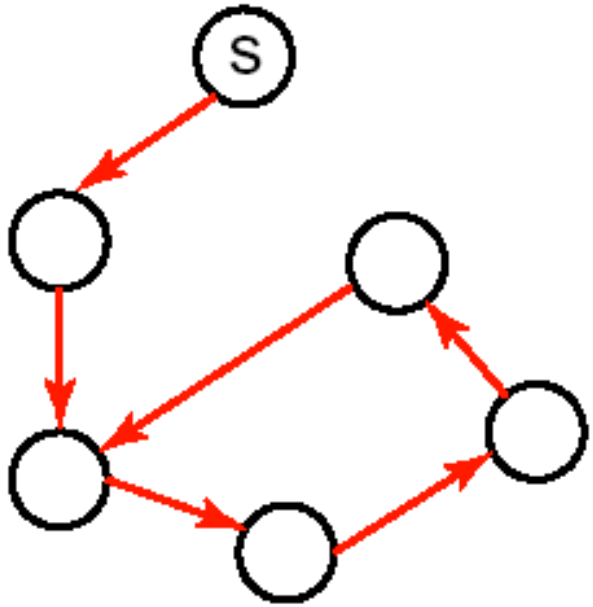
  ***concat*(*F1*,*Vecini*,*F2*) ∧**

  ***sorteaza_dupa_f*(*F2*,*F3*)**

- **[(*A*,[],*g*/0,*h*/5)] ⇒ [(*D*,[*A*], *g*/2,*h*/3.605), (*B*,[*A*],*g*/4,*h*/4.123)] ⇒ [(*C*,[*D*,*A*],*g*/4,*h*/2.236), (*B*,[*A*], *g*/4,*h*/4.123)] ⇒ [(*E*,[*C*,*D*,*A*],*g*/6,*h*/1), (*B*,[*A*], *g*/4,*h*/4.123)] ⇒ [(*F*,[*E*,*C*,*D*,*A*],*g*/7,*h*/0), (*B*,[*A*], *g*/4,*h*/4.123)] ⇒ ...**

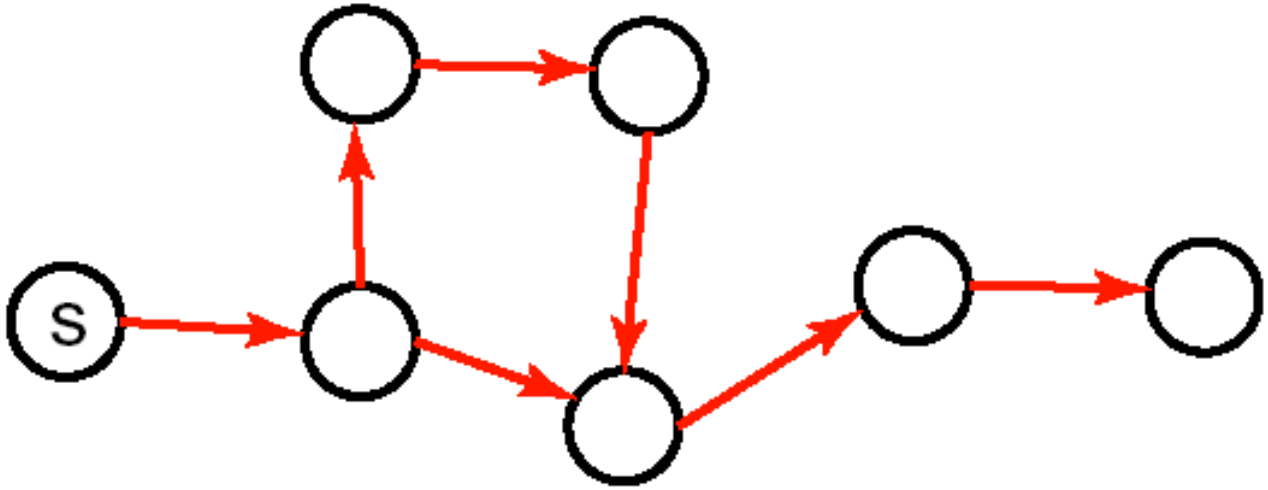
Detectarea ciclurilor

- Presupune eliminarea vecinilor unui nod *n* care se afla deja pe calea de la nodul de start la *n*.
- Daca graful se memoreaza explicit si se face o cautare in adancime atunci acest test se poate face in timp constant folosind un vector de biti pentru nodurile grafului.



- Pentru celelalte metode acest test necesita un timp liniar in lungimea caii pana la nodul *n*.
- Verificarea se poate face fie cand un nod este adaugat la frontiera, fie cand este selectat din frontiera pentru expandare.

Eliminarea cailor multiple catre acelasi nod



- Ideea este de a elimina din frontiera orice nod catre care deja s-a gasit o cale.
- Aceasta tehnica va conduce implicit si la detectarea ciclurilor, fiind mai generala.
- Dezavantjul este insa ca trebuie stocate explicit toate nodurile catre care s-au gasit cai, nu doar nodurile din frontiera.

Eliminarea cailor multiple catre acelasi nod – cai optimale

- **Daca se cere gasirea unei cai de cost minim atunci apare urmatoarea problema: ce se intampla daca dupa ce s-a gasit deja o cale de la nodul de start la un nod *n* se gaseste ulterior o alta cale de la nodul de start la *n* mai scurta decat prima ?**
- **Exista trei posibilitati:**
  - **Se elimina din frontiera toate caile care folosesc calea mai lunga catre nodul *n*.**
  - **Se pot actualiza toate segmentele de cai care contin nodul *n*, catre nodurile din frontiera cu calea mai scurt catre *n* care a fost gasita.**
  - **Se asigura faptul ca situatia descrisa nu poate sa apara niciodata astfel incat ori de cate ori se gaseste o cale catre un nod *n* aceasta are un cost minim.**

Eliminarea cailor multiple in cautarea A*

- **Sa presupunem ca s-a selectat din frontiera un nod *n*, dar ca exista o cale mai scurta catre *n* care trece printr-un nod *m* din frontiera. Atunci:**
  $g(n) + h(n) \leq g(m) + h(m)$ deoarece *n* s-a selectat inaintea lui m
  $g(m) + d(m,n) < g(n)$ deoarece calea catre *n* via *m* este mai scurta decat calea curenta pana la *n*
  Combinand cele doua inegalitati rezulta:
  $d(m,n) < g(n) - g(m) \leq h(m) - h(n)$, adica $d(m,n) < h(m) - h(n)$.
  Impunand conditia ca $h(m) \leq d(m,n) + h(n)$ pentru orice pereche de noduri *m* si *n* pentru care exista o cale de la *m* la *n*, inegalitatea de mai sus nu poate avea loc.
- **Conditia $h(m) \leq d(m,n) + h(n)$ se mai numeste si *monotonia* functiei *h*. Explicatia ar fi ca din aceasta conditie rezulta ca $f(m) \leq f(n)$, adica valorile *f* ale elementelor selectate din frontiera nu descresc.**
- **Observatie: Este suficient sa se faca verificarea conditiei de monotonie pentru orice doua noduri *m* si *n* astfel incat *n* este un vecin al lui *m*.**
- **Exemplu: Functia euristica *h* egala cu distanta euclidiana este monotona.**

Cautare in adancime marginita

- **Detectia ciclurilor este costisitoare pentru cai lungi, iar cautarea in adancime se bucleaza la infinit daca nu se face detectarea ciclurilor.**
- **O solutie este limitarea adancimii de cautare. Acest lucru inseamna ca nu se mai depun in multimea frontiera noduri al caror costdepaseste o valoare data. Aceasta conditie se numeste *marginire* (engl.*bounding*).**
- **Avantaj: garnteaza oprirea algoritmului de cautare.**
- **Dezavantaje:**
  - **Daca marginea aleasa este mai mica decat costul solutiei optime atunci solutia optima nu va fi gasita niciodata.**
  - **Daca insa marginea aleasa este mult mai mare decat costul solutiei optime atunci cautarea va explora nejustificat anumite cai care nu duc la solutii optime.**

Cautare prin adancire iterativa (engl.*iterative deepening*)

- **Strategiile care garanteaza oprirea consuma un spatiu de memorie exponential.**
- **Strategia de cautare in adancime consuma un spatiu de memorie liniar, dar nu garanteaza oprirea. Varianta cu marginire, care garanteaza oprirea, prezinta insa alte dezavantaje.**
- **Cautarea prin adancire iterativa incearca sa combine doar avantajele metodelor de mai sus. Ideea este de a recalcula frontiera in loc de a o stoca explicit. Pentru recalculare se va folosi cautarea in adancime cu marginire.**
- **Cautarea prin adancire iterativa se poate combina cu toate metodele prezentate. Sunt interesante combinarile cu cautarea pe nivel si cautarea A*.**
- **Timpul de cautare creste cu un factor constant, care este cu atat mai mic cu cat factorul de ramificare inainte este mai mare. Presupunand ca s-a ajuns cu cautarea la nivelul k, cele b^k noduri ale frontierei s-au generat o singura data, cele b^{k-1} noduri de pe nivelul $k-1$ s-au generat de dou ori, … si cele de pe nivelul 1 s-au generat de k ori. Rezulta ca numarul total de noduri generate este: $b^k + 2b^{k-1} + 3b^{k-2} + \ldots + kb = b^k(b/(b-1))^2$.**