Inteligență Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs 3

Strategii de cautare

- Cautari locale
- Cautari on-line
- Problema satisfacerii restrictiilor (CSP)

1. Strategii de cautare locala

- **Cautari locale** – opereaza asupra starii curente generand succesorii
- Calea catre solutie nu are importanta
- Gasire solutie – CSP - nu conteaza calitatea solutiei
- Gasire solutie optima – functie de evaluare sau de cost
- Probleme in gasirea solutiei optime pe baza de cautari locale in functie de forma spatiului de cautare
- Maxim global, maxim local, platou, umar

Cautari locale

Caracteristici

- Folosesc putina memorie
- Gasesc solutii destul de bune in spatii finite f mari si chiar in spatii infinite
- Folosite si pt probleme de optimizare (cu fct obiectiv)
- Algoritmi pentru spatii de cautare discrete
  - Hill climbing
  - Simulated annealing

Algoritm: **Hill climbing** (greedy local search)

/* intoarce o stare cu functia de evaluare un maxim local */

1. S ← stare initiala
2. Genereaza S_j=Succ(S), toti succesorii starii S
3. S' ← S_j cu Eval(S_j) maxim dintre toti succesorii S_j
4. **daca** Eval(S') ≤ Eval(S) **atunci intoarce** S
5. S ← S'
6. **repeta** de la 2

sfarsit

- Cautarea se orienteaza permanent in directia cresterii valorii;
- Se termina cand ajunge la un maxim (nici un succesor nu are valoarea mai mare)

Hill Climbing

- Problema 8 regine
- S – tabla completa
- Succesori – toate starile posibil de generat prin mutarea a 1 regina intr-un alt patrat in coloana
- o stare are 8 x 7 = 56 succesori
- Fct de evaluare = nr de perechi de regine care se ataca
- Minim global = 0
- 86% instante nu gaseste solutia (3 pasi)
- 14% gaseste soluta (4 pasi)
- Spatiul starilor $8^8 \approx$ 17 milioane de stari

Hill Climbing - imbunatatiri

- Miscari laterale – se spera sa fie “umar” si nu “platou” (continui si pentru Eval(S') = Eval(S))
- Daca “platou” – risc de bucla infinita – necesita limita in cautare
- Cu aprox. 100 miscari laterale – problema 8 regine – 94% instante pt care se gaseste solutia

Hill Climbing - imbunatatiri

Hill climbing stohastic

- Dintre starile succesoare cu $Eval(S_j) \geq Eval(S)$, se alege aleator un S_j

First choice hill climbing

- Genereaza aleator succesori pana gaseste $Eval(S_j) \geq Eval(S)$, continua cautarea cu S_j

Random restart Hill Climbing

- Repeta diferite HC cu stari initiale generate aleator
- Daca fiecare HC are o probabilitate de succes de p $\rightarrow$ 1/p repetitii

Simulated annealing

- Simuleaza un proces fizic (calirea)
- T – temperatura
- Scaderea gradientului – <u>solutii de cost minim</u>
- Alege o miscare la intamplare
- Daca starea este mai buna, cauta in continuare de la aceasta
- Altfel alege starea mai "proasta" cu o probabilitate
- Scade probabilitatea de selectie a starilor mai "proaste" pe masura ce temperatura scade

Algoritm: **Cautare Simulated Annealing**

1. T ← temperatura initiala
2. S ← stare initiala
3. v ← Eval(S)
2. **cat timp** T > temp finala **executa**

    2.1 **pentru** i=1,n **executa**

        S' ← Succ(S)

        v' ← Eval(S')

        daca v' < v **atunci S** ← S'

        altfel S ← S' cu prob. exp(-(v'-v)/T)

            (in rest S nemodificat)

    2.2 T ← 0.95 * T

sfarsit

Local beam search

- K stari generate aleator
- Stochastic local beam search

Cautari locale in spatii continue

- Factor de ramificare infinit
- First choice HC, Simulated annealing
- ***Problema:*** dorim sa plasam 3 supermarket-uri pe o harta a.i. suma patratelor distantelor de la fiecare comuna de pe harta la cel mai apropiat supermarket sa fie minima.
- Spatiul starilor este definit prin coordonatele supermarket-urilor

$$(x_1,y_1)\ (x_2,y_2)\ (x_3,y_3)$$

- (spatiu *n*-dimensional cu *n* variabile)
- O miscare in acest spatiu – miscarea unui sm pe harta
- C_i- multimea de orase care sunt cel mai aproape de sm ***i*** in starea curenta

$$f(x_1,y_1,x_2,y_2,x_3,y_3) = \Sigma_{i=1,3}\ \Sigma_{c\in Ci}\ (x_i-x_c)^2+(y_i-y_c)^2$$

Cautari locale in spatii continue

Cum putem gasi solutia?

VARIANTE

- Discretizare spatiu – discretizarea vecinatatii fiecarei stari

(de ex mutam cate un sm o data fie in directa x fie in directia y cu cate o cantitate fixa M => 12 stari succesoare)

- Utilizarea gradientului

Si aplicam HC prin actualizarea starii curente cu x + constanta * gradient

2. Strategii de cautare on-line

- Cautare “offline”
- **Cautare “on-line” – cautare + actiune**
- Exemplu: robot care investigheaza mediul
- **Cautare online:** pentru fiecare actiune, agentul primeste perceptia care ii spune in ce stare a ajuns.

  Construieste Rezultat[s,a]
- **Cautari locale online:**
  - HC, nu pot random restart,
  - Adauga memorie la HC – memoreaza cea mai buna estimare curenta

Cautare on-line

- Caracteristici

- Metode
  - On-line DFS
  - Programare dinamica asincrona (ADP)
  - Learning Real-Time A* (LRTA*)
  - Cautare cu tinta mobila

2.1 On-line DFS

- Agentul nu are un model al mediului, il descopera pe masura ce il parcurge
- Nodurile sunt expandate pe baza localittaii lor

On-line DFS

On-line DFS(s′) intoarce o actiune

var globale: s,a, starea si actiunea anterioare, initial nule

Rezultat – o tabela care mapeaza (s,a) la s', initial vida

Neincercate – o tabela care mapeaza s la o lista de actiuni neincercate

Non-revenite – o tabela ce mapeaza s la o lista de stari la care nu s-a revenit inca

daca s' este stare scop **atunci intoarce** stop

daca s' este stare noua **atunci** Neincercate[s']← actiuni(s′) /* Neincercate null pt s' */

daca s ≠ null **atunci**

Rezultat[s,a] ← s′

adauga s la inceputul lui Non-revenite[s']

daca Neincercate[s'] este vida **atunci**

daca Non-revenite[s'] este vida **atunci intoarce** stop

a ← o actiune b a.i. Rezultat[s',b] = pop(Non-revenire[s'])

s' ← null

altfel a ← pop(Neincercate[s'])

s ← s'

intoarce a

2.2 Programare dinamica asincrona (ADP)

Programare dinamica

Descompunerea problemei in subprobleme ne-distincte

Principiul optimalitatii – o cale este optima $\leftrightarrow$ orice segment (subcale) a acesteia este optima

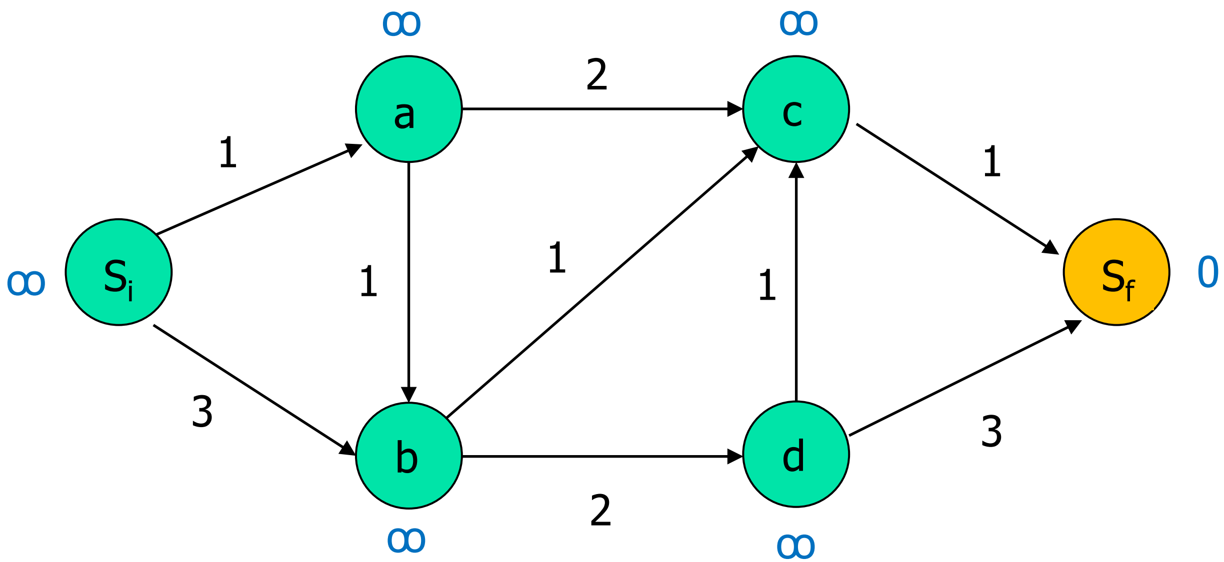
ADP: Daca se cunoaste **h*** pt fiecare nod, calea de cost minim poate fi obtinuta repetand procedura:

- Pentru fiecare nod succesor **j** a nodului curent **i** **calculeaza** $f^*(j)=c(i,j) + h^*(j)$
- Mergi la starea j pt care **$f^*(j)$ este minim**

Programare dinamica asincrona

Presupunem urmatoarea situatie

- Pentru fiecare nod **i** exista un proces care corespunde lui **i**
- Fiecare proces inregistreaza **h(i)** – estimarea lui **h*(i)**
- Valoarea initiala a lui **h(i)** este arbitrara cu exceptia nodurilor stare finala
- Fiecare proces **i** actualizeaza **h(i):**
  - pentru fiecare vecin **j - calculeaza f(j)=c(i,j) + h(j)**
  - $h(i) \leftarrow \min_j f(j)$



ADP

$$f(j)=c(i,j) + h(j)$$

Programare dinamica asincrona

Ordinea de actualizare este arbitrara

Daca costul arcelor este pozitiv, converge la valorile reale

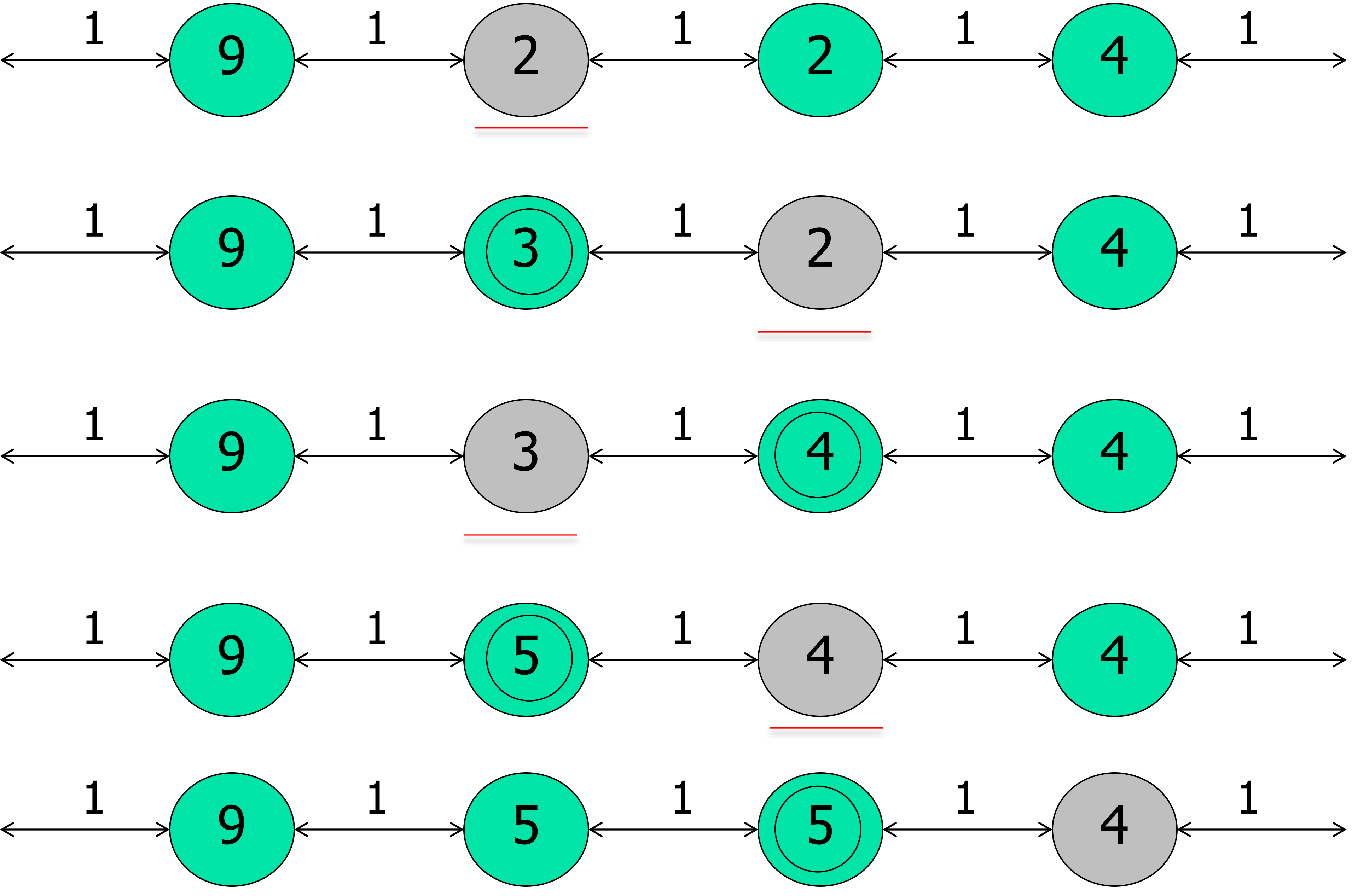
Spatiu de stari mare – nepractic

Foloseste ca fundament pt A* in timp real

- Modific A* pentru a putea fi utilizat on-line
- De ce nu pot folosi direct A*?

2.3 Learning Real-Time A* (LRTA*)

- Intrepatrunde calculul miscarii urmatoare cu executia miscarilor si alege miscarile intr-un timp constant
- Construieste si actualizeaza o tabela ce contine estimari eursitice ale distantei fiecarei stari la starea scop.
- Actualizeaza estimarea de cost a starii pe care a parasit-o si alege miscarea (aparent) cea mai buna
- Initial, intrarile in aceasta tabela corespund unor evaluari euristice sau 0 si sunt subestimari
- Prin explorarea repetata a spatiului, valorile sunt actualizate pana cand ajung, intr-un final, la valorile corecte


1
9
1
2
1
2
1
4
1
1
9
1
3
1
2
1
4
1
1
9
1
3
1
4
1
4
1
1
9
1
5
1
4
1
4
1
1
9
1
5
1
5
1
4
1

Learning Real-Time A* (LRTA*)

Agentul considera numai nodul curent

Agent in nodul **i**

Agentul inregistreaza distanta estimata pentru un nod

1. Lookahead

- Pentru fiecare vecin **j** a lui **i calculeaza**

$$\mathbf{f(j)=c(i,j) + h(j)}$$

h(j) – estimarea caii $j \rightarrow S_f$

c(i,j) – cost arc $i \rightarrow j$

Learning Real-Time A* (LRTA*)

2. Actualizeaza h

- Actualizeaza h estimat a nodului **i**

  $$\mathbf{h(i) \leftarrow min_j\, f(j)}$$

3. Selecteaza actiunea si memoreaza

- Mergi la starea **j** cu valoarea **minima f(j)** – actiunea ***a***

- Memoreaza $i \rightarrow_{\boldsymbol{a}} j$

Valorile initiale ale lui **h** trebuie sa fie admisibile

Learning Real-Time A* (LRTA*)

- Agentul aflat in starea ***s'*** face backtrack intr-o stare anterior vizitata ***s*** in momentul in care estimarea rezolvarii problemei din starea ***s'*** + costul de intoarcere in ***s*** este *mai mic* decat costul de a merge inainte din starea ***s'***.
- In LRTA* meritul fiecarui nod este calculat relativ la pozitia curenta (nu la starea initiala).
- Este suficient sa memoram *tabela hash H* cu ***h*** a nodurilor deja vizitate

Learning Real-Time A* (LRTA*)

Algoritm LRTA*(s′) intoarce o actiune

intrari: s′, o perceptie care identifica starea curenta

variabile globale: Rezultat: o tabela indexata dupa stari si actiuni, initial vida

H – o tabela cu estimarea starilor, initial vida

s, a – starea anterioara si actiunea anterioara, initial nule

daca s′ este stare scop **atunci intoarce** stop

daca s′ este stare noua ($s' \notin H$) **atunci** $H[s'] \leftarrow h(s')$

daca $s \neq$ null **atunci**

$Rezultat[s,a] \leftarrow s'$

$H[s] \leftarrow \min_{b \in Actiuni(s)}$ **LRTA*Cost**(s, b, Rezultat[s,b], H)

$a \leftarrow$ o actiune b din Actiuni(s′) care minimizeaza LRTA*Cost(s′, b, Rezultat[s′,b], H)

$s \leftarrow s'$

intoarce a

Agentul selecteaza o actiune in fct de valorile starilor invecinate, care sunt actualizate pe masura ce agentul exploreaza spatiul

Algoritm LRTA*Cost(s,a,s′, H) intoarce o estimare de cost

daca s′ nu este definita **atunci intoarce** h(s)

altfel intoarce $c(s,a,s') + H[s']$

Learning Real-Time A* (LRTA*)

- Intr-un spatiu de cautare finit cu costuri pozitive, in care exista o cale de la orice stare S la S_f si se utilizeaza estimari admisibile nenegative, LRTA* este complet (va ajunge in final in starea scop)
- In plus « invata » solutia optima in timp

2.4 Cautare cu tinta mobila (MTS)

- Generalizare a LRTA*
- Starea scop se schimba pe parcursul cautarii
- MTS – generalizare a LRTA*
- MTS trebuie sa obtina informatie despre locatia starii scop
- Consideram spatiul ca un graf neorientat
- PS nu are harta mediului dar stie pozitia T
- PS stie starile adiactente

Cautare cu tinta mobila

- Presupunem ca PS si T se misca alternativ
- Presupunem ca viteza T este mai mica decat cea a PS (la cativa pasi T sta pe loc)
- Stare finala (scop) = PS si T au aceeasi pozitie
- x, x′ – pozitia curenta si pozitia vecinilor pt PS
- y, y′ – pozitia curenta si pozitia urmatoare pt T

Cautare cu tinta mobila (MTS)

- Presupun ca toate arcele au ***cost 1***
- Matrice de valori **h(x,y)** – estimatrea distantei intre PS si T
- Matricea poate fi mare dar este suficient sa actualizam numai valorile care se schimba
- h admisibila
- 2 evenimente, fiecare face o actualizare a valorii euristicilor:
  - Miscare a PS
  - Miscare a T

Cautare cu tinta mobila (MTS)

Miscare PS

- Calculeaza $h(x',y)$ pentru fiecare vecin x' a lui x
- Miscare la x' cu $h(x',y)$ minim

$$x \leftarrow x'$$

- Actualizeaza valoarea $h(x,y)$ astfel

$$h(x,y) = \max \{ h(x,y), \min_{x'} (h(x',y)+1) \}$$

Deoarece orice cale de la x la y trebuie sa treaca prin vecinul x', costul drumului minim de la x la y va fi cel putin la fel de mare ca cel al drumului prin oricare vecin al lui x.

Cautare cu tinta mobila (MTS)

Miscare T

PS observa miscarea T de la y la y′

- Calculeaza h(x,y′) pentru noua pozitie y′ a lui T
- Actualizeaza valoarea h(x,y) astfel

$$h(x,y) = \max \{ h(x,y), h(x,y')-1 \}$$

- Actualizeaza starea scop cu noua pozitie a lui T

$$y \leftarrow y'$$

Daca $h(x,y') > h(x,y)+1$ atunci $h(x,y) \leftarrow h(x,y')-1$ deoarece, distanta y si y′ fiind 1, distanta la vechea pozitie trebuie sa fie cel putin la fel de mare ca distanta la noua pozitie -1

Cautare cu tinta mobila (MTS)

- Intr-un spatiu de cautare finit cu costuri pozitive in care exista o cale de la fiecare stare S la starea scop S_f, daca se porneste cu valori ale functiei euristice admisibile si se permit miscari ale PS si T in orice directie cu cost unitar, PS care executa MTS va ajunge la T daca T sare periodic peste miscari.