12.10.2022

- concepte / noțiuni
- categorii / clasificare

Inteligență Artificială
Cursul 3

Găsirea unui drum într-un graf - problemă NP-completă. Pentru aceasta trebuie:

- descrierea unui model
- identificarea stărilor speciale și a spațiului problemei
- descrierea și validarea tranzițiilor
- strategie de căutare

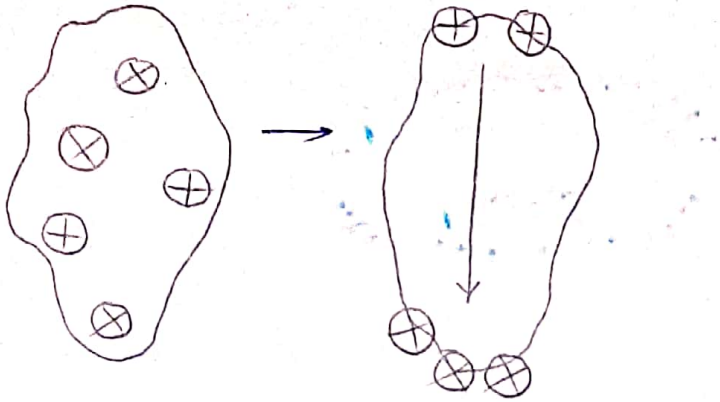
Există 2 tipuri de strategii de căutare:

- neinformate → se explorează complet spațiul problemei și se va găsi sigure o soluție. (sunt lente)
- informate → există reguli euristice

Diferența dintre o regulă de deducție și euristică constă în faptul că regulile de deducție pot fi demonstrate, în timp ce euristicile nu, motiv pentru care pot fi contrazise. Pentru o euristică are trebui să facem milioane de experimente a căror probabilitate să o calculăm, ceea ce este foarte costisitor d.p.d.v. al execuției și memorării. Trebuie să ne asumăm posibilitatea apariției unor erori având în vedere cât de complex este spațiul experimentelor. În practică probabilitățile se măsoară pt. spații mai mici.

Fie un spațiu al problemei cu stări inițiale și finale. Ce putem face pentru a direcționa într-un sens căutarea în spațiu? În acest caz, ne vom folosi de euristici pentru descompunerea spațiului.

h:



Într euristică există o proprietate: se vor plasa stările finale în extreme.

$h: S \to [min, max]$

$h(IS) \to min / max$; $h(FS) = max / min$

Euristica → admisibilă; nu supraestimează niciodată distanța dintre 2 stări

Pb. Turnurile Hanoi: avem un nr. de piese care trebuie așezate pe un anumit turn. Dacă piesa de start este 1, iar scopul este n, atunci trebuie să găsim toate stările posibile.

$(n, \underbrace{1, 1 \dots 1}_{n})$

$(n, n, n, \dots, n)$

Ne dorim o euristică care să încadreze un număr cât mai apropiat de nr. de stări din problemă.

$n, n \longrightarrow \frac{n}{n+2}$

$h_2(IS) = n \longrightarrow n + n + 1 + n + 2 + \dots + n + n$

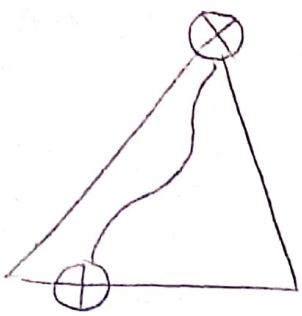
$h_2(FS) = n \cdot n$

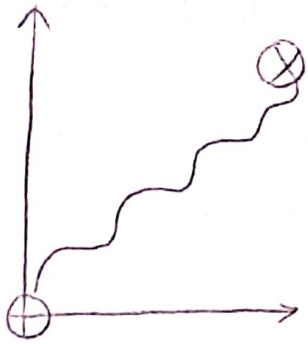
La fiecare pas, euristica trebuie să calculeze n stări.

Există 3 moduri de a reprezenta o soluție.

- path in a graph
- path in a space → ne spune despre complexitatea problemei și este folosit mai mult pt strategii

- heuristic function → o pereche între-un graf; ne prezintă avantajele acelei euristici





Strategii informale

1) Greedy → alege mereu cea mai bună mutare prin evaluarea tuturor stărilor disponibile accesibile din starea curentă. Apoi se va selecta cea mai apropiată stare neexplorată de cea dorită.

```
while isFinal(CS) {
    for i=1 to n {
        for j=1 to m {
            if valid(CS,(i,j))
```

CS = current State
IS = initial state
FS = final state

stări. add(tranziție(CS, i,j))}}}.

CS = max(strategie(h(

Best case: nu garantează soluția optimă, însă este mult mai rapid decât BFS.

Worst case: aceeași complexitate cu BFS și face alegerile greșite.

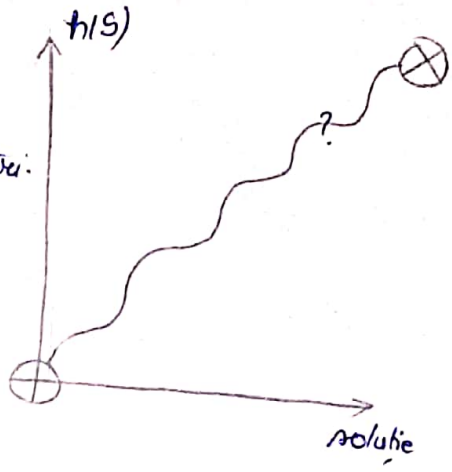
2) Hillclimbing → din stările accesibile din starea curentă vom alege o stare cel puțin la fel de apropiată de starea finală: h(new State) ≥ h(current State)

Această strategie nu spune ce decizie să luăm, ci doar cum să le filtrăm.

ex. dintre toate stările selectate (filtrate) se va alege prima (o variantă de Hillclimbing) ③

De multe ori, hillclimbing merge mai repede decât Greedy.

Reprezentarea unei soluții hillclimbing arată ca un grafic în scări:



Avantajul constă în faptul că este cea mai rapidă strategie rezonabilă.

Nici pt. Greedy, nici pt. hillclimbing nu avem garanția că acea soluție este cea optimă. Poate fi optimă doar din perspectiva acelei euristici.

Hillclimbing este cea mai rapidă strategie neinformată cu mai multe variante, în funcție de cum selectăm stările după h Hreare.

Dezavantajul → euristica ne poate duce în stări de optim local (toate stările din cea curentă sunt mai rele, caz în care alg. se blochează).

Stările de optim local nu sunt ușor de prevăzut.

Să presupunem că uneori vom permite îndepărtarea de starea finală.

3) A*

Ne dorim toate sol. posibile, deci trebuie să arătăm că am parcurs exhaustiv spațiul problemei (strategiile informate ; BFS este singurul care returnează sol. corectă).

IDDFS → explorăm prin DFS până la adâncimea maximă d.

A* explorează d(S)+h(S), unde h(S) este distanța estimată până la starea finală.

Ca IDDFS era dat de numărul de pași de la starea inițială până la cea curentă.

Acum = h(Sc) + pași(cs)

La A* nu trebuie să ne oprim la starea finală

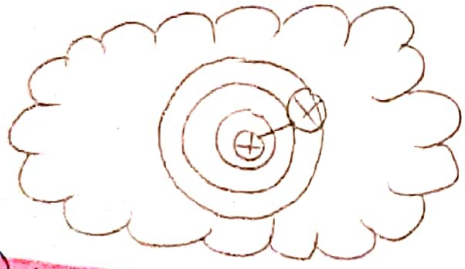
A* pathfinding animation. - site în care ne putem juca cu algoritmul.

Diferența dintre A* și Dijkstra.

Dijkstra explorează cât mai aproape de starea inițială (concentric)

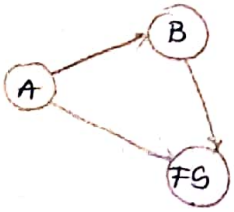
BFS / IDAS explorează la distanță egală față de starea inițială, în timp ce la A* împinge soluțiile explorează către starea finală.

BFS / IDAS (concentric)



A*

Optimizarea lui A* → euristica să nu fie doar admisibilă, ci și persistentă.



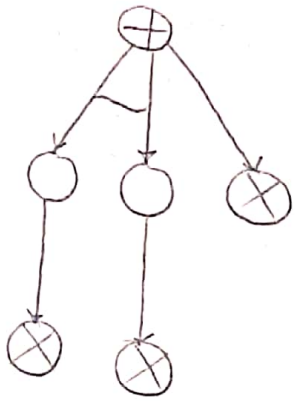
Includerea oricărei stări suplimentare în soluție trebuie să fie mai lungă decât sol. fără acea stare (prop. de consistență)

$h(A) \leq h(A,B) + h(B)$ pentru B o stare vecină a lui A.

Simplified Memory Bounded A* → se bazează pt pruning pt eliminarea stărilor care au costul prea mare decât a fost estimat.

Spațiile de probleme pot fi → nedeterministe → nu știm în ce stare vom ajunge și cu ce probabilitate
→ deterministe → știm stările posibile și probabilitățile.

Arborele ANA-OR



Spații parțial observabile: solitaire: știm ce carte urmează dar nu și ce urmează după ea.

Spații care mai pot fi enumerate: avem foarte multe butoane și nu știm ce face niciunul.

Sau ce opțiuni vom avea disponibile după apăsarea unuia / după o tranziție.