Algorithmes de recherche informés (heuristiques)

Chap. 4

1

Plan

- Recherche meilleur-d'abord
- Recherche meilleur-d'abord glouton
- A^*
- Heuristiques

2

Rappel: Recherche dans l'arbre

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
```

- Une stratégie de recherche est définie par l'ordre d'expansions de noeuds choisi
-
-

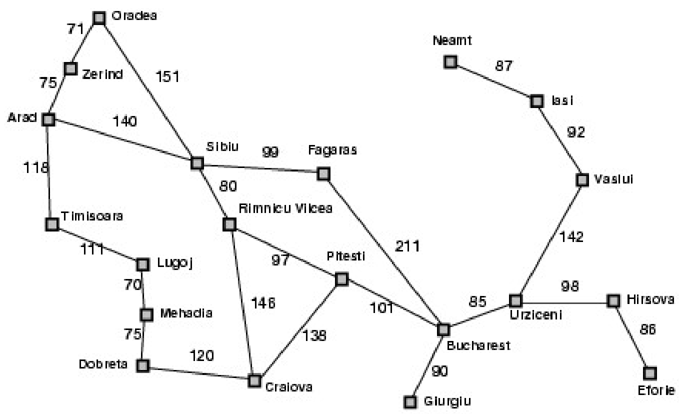
Recherche meilleur-d'abord

- Idée: utiliser une fonction d'évaluation *f(n)* pour chaque nœud
  - estimer la « désirabilité »
  - selon les caractéristiques du nœud (état), et non seulement la topologie de l'espace
  - → Expansion du nœud non exploré le plus désirable

- Implantation:
  Ordonner les nœuds dans la frange dans l'ordre décroissant de désirabilité

- Cas spéciaux:
  - Recherche meilleur d'abord glouton
  - Recherche A^*

Roumanie avec coût d'étape en km



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

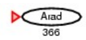
5

Recherche meilleur-d'abord glouton (Greedy best-first search)

- Fonction d'évaluation $f(n) = h(n)$ (heuristique)
- = estimation du coût de *n* à *but*
- e.g., $h_{SLD}(n)$ = distance en ligne droite de *n* à Bucharest
- Recherche meilleur-d'abord glouton développe le nœud qui paraît le plus proche du but
- Algorithme:
  - Tant que current ≠goal & il y a des nœuds suivants:
    Sélectionner h(best_next),
    Aller à best_next

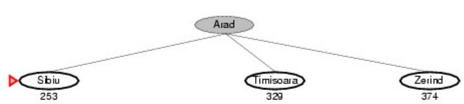
6

Recherche meilleur-d'abord glouton - Exemple



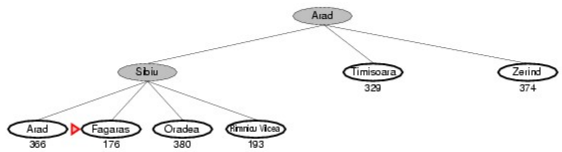
7

Recherche meilleur-d'abord glouton - Exemple



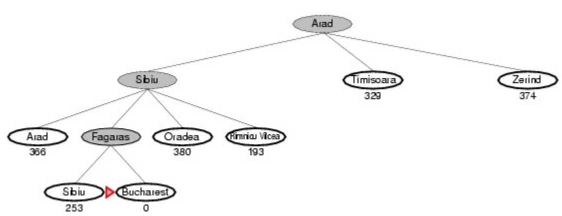
8

Recherche meilleur-d'abord glouton - Exemple



9

Recherche meilleur-d'abord glouton - Exemple



10

Propriétés de la recherche meilleur-d'abord glouton

- Complète? Non – peut être coincé dans un boucle, e.g., Iasi → Neamt → Iasi → Neamt →
- Temps? $O(b^m)$, mais une bonne heuristique peut beaucoup améliorer ça
- Espace? $O(b^m)$ – garde tous les nœuds en mémoire (potentiellement)
- Optimal? Non
-

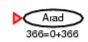
11

Recherche A

- Idée: faire une estimation du chemin complet (du nœud initial jusqu'à un but)
-
- Fonction d'évaluation $f(n) = g(n) + h(n)$
  - $g(n)$ = coût jusqu'à présent pour atteindre n
  - $h(n)$ = coût estimé de n à un but
  - $f(n)$ = coût total du chemin passant par n à un but
- C'est une fonction plus complète que celle utilisé dans Meilleur-d'abord glouton

12

Exemple rechrche A

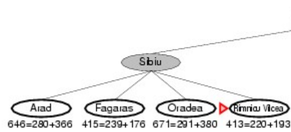


13

Exemple rechrche A

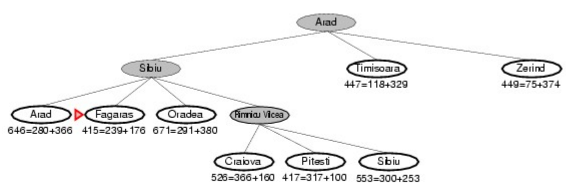
14

Exemple rechrche A



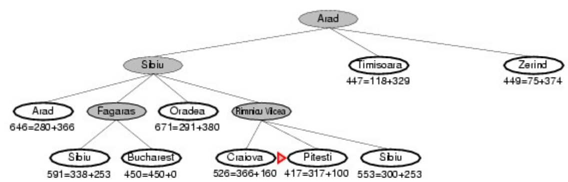
15

Exemple rechrche A



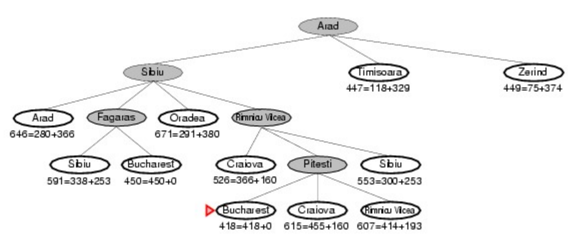
16

Exemple rechrche A



17

Exemple rechrche A



18

Heuristiques admissibles

- Une heuristique $h(n)$ est admissible si pour chaque noeud n,

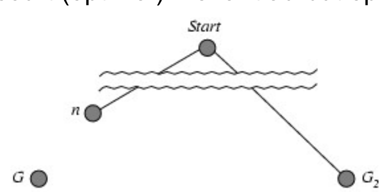
$$h(n) \leq h^*(n),$$

où $h^*(n)$ est le vrai coût pour atteindre un but à partir de n.
- Une heuristique admissible ne surestime jamais le coût pour atteindre au but, i.e., elle est optimiste
- Exemple: $h_{SLD}(n)$ (ne surestime jamais la vraie distance en route)
- Théorème: Si $h(n)$ est admissible, A utilisant `TREE-SEARCH` est optimal
- Un algorithme A admissible est un algorithme A*.
-

19

Optimalité de A^* (proof)

- Supposons qu'un but non optimal G_2 a été généré et inséré dans la frange. Soit n un noeud non exploré dans la frange qui fait partie du chemin le plus court (optimal) menant au but optimal G.
-

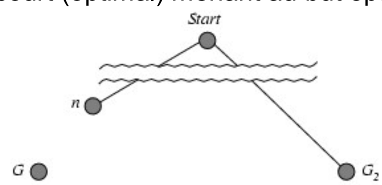


- $f(G_2) = g(G_2)$ car $h(G_2) = 0$
- $g(G_2) > g(G)$ car G_2 est suboptimal
- $f(G) = g(G)$ car $h(G) = 0$
- $f(G_2) > f(G)$ à partir de ci-dessus

20

Optimalité de A^* (proof)

- Supposons qu'un but non optimal G_2 a été généré et inséré dans la frange. Soit *n* un nœud non exploré dans la frange qui fait partie du chemin le plus court (optimal) menant au but optimal *G*.



- $f(G_2)$ $> f(G)$ à partir de ci-dessus
- $h(n)$ $\leq h^*(n)$ puisque h est admissible
- $g(n) + h(n)$ $\leq g(n) + h^*(n)$
- $f(n)$ $\leq f(G)$

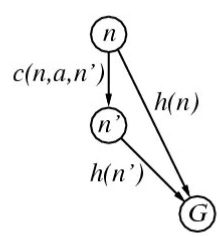
Donc $f(G_2) > f(n)$, et A^* ne va jamais sélectionner à développer G_2 pour terminer l'algorithme

Heuristiques consistantes

- Une heuristique est consistante si pour chaque nœud *n*, chacun de ses successeurs *n'* générés par une action,

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

- Si *h* est consistante, nous avons



$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n,a,n') + h(n') \\
&\geq g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

- i.e., $f(n)$ est non-décroissante en suivant un chemin quelconque.

- Théorème: si $h(n)$ est consistante, A^* utilisant `GRAPH-SEARCH` est optimal

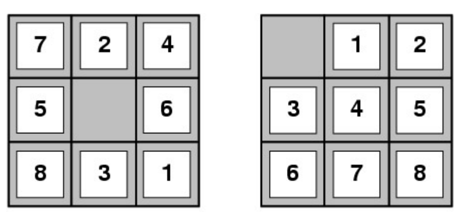
Propriétés de A*

- Complète? Oui (à moins qu'il y a un nombre infini de nœuds avec f ≤ *f(G)*)
- Temps? Exponentiel
- Espace? Garde tous les nœuds en mémoire
- Optimal? Oui

Heuristiques admissibles

E.g., Pour 8-puzzle:
- $h_1(n)$ = nombre de tuiles mal placés
- $h_2(n)$ = distance Manhattan totale

(i.e., nb. de carrées pour arriver à la place désirée pour chaque tuile)



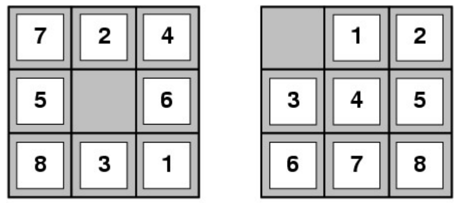
Start State Goal State

- $h_1(S) = ?$
- $h_2(S) = ?$
-

Heuristiques admissibles

E.g., Pour 8-puzzle:
- $h_1(n)$ = nombre de tuiles mal placés
- $h_2(n)$ = distance Manhattan totale

(i.e., nb. de carrées pour arriver à la place désirée pour chaque tuile)



Start State Goal State

- $h_1(S)$ = ? 8
- $h_2(S)$ = ? 3+1+2+2+2+3+3+2 = 18

25

Dominance

- Si $h_2(n) \geq h_1(n)$ pour tout n (supposons que les 2 sont admissibles)
- Alors h_2 domine h_1
- h_2 est meilleure pour la recherche

- Coûts typiques de recherche (nb. moyen de nœuds explorés) :

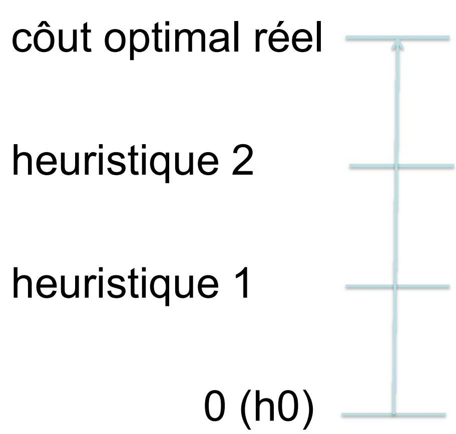
- *d=12* IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nœuds
  $A^*(h_2)$ = 73 nœuds
- *d=24* IDS = trop de nœuds
  $A^*(h_1)$ = 39,135 nœuds
  $A^*(h_2)$ = 1,641 nœuds
-

26

Problèmes relaxés

- Un problème a généralement des contraintes sur les actions
- Si on enlève une ou plusieurs de ces contraintes, on a un problème relaxé
- Le coût pour une solution optimale du problème relaxé est admissible pour le problème original
- Si les règles du jeu de 8-puzzle sont relaxées pour qu'on puisse placer un tuile n'importe où, alors $h_1(n)$ donne la solution du chemin le plus court
- Si les règles sont relaxées pour qu'un tuile puisse bouger vers un carré adjacent, alors $h_2(n)$ donne le chemin le plus court

Fonctions heuristiques



- h2 domine h1

En pratique

- Tous les problèmes ne nécessitent pas toujours la solution optimale
- On se contente souvent d'une solution « acceptable »
  - Difficulté de trouver une heuristique admissible « tractable » (temps, espace)
  - Le problème est assez petit et on n'a pas à le faire très souvent
  - …
- On utilise souvent un algorithme non optimal
- On peut aussi surestimer le coût parfois (non admissible), mais l'estimation reste assez proche du réel

29