Exploration informée

Master 1 SID

Dr H.Belleili

Recherche informée

- Les stratégies de recherche non-informée ne sont pas très efficaces dans la plupart des cas. Elles ne savent pas si elles s'approchent du but .
- Les stratégies de recherche informée utilisent une fonction d'estimation (heuristique) pour choisir les noeuds à visiter.

Stratégies de recherche informée

- Meilleur d'abord (Best-first)
  - Meilleur d'abord gloutonne (Greedy best-first)
  - A*
- Algorithmes à mémoire limitée
  - IDA*, RDFS et SMA*
- Algorithmes de recherche local et optimisation
  - Escalade (Hill-climbing)
  - Recuit simulé (Simulated annealing)
  - Recherche local en faisceau (local beam)
  - Algorithmes génétiques

Meilleur dôabord

- Lôidée est dôutiliser une fonction dôévaluation qui estime lôintérêt des noeuds et de développer le noeud le plus intéressant.
- Le noeud à développer est choisi selon une fonction dôévaluation $f(n)$.
- Une composante importante de ce type dôalgorithme est une fonction heuristique $h(n)$ qui estime le coût du chemin le plus court pour se rendre au but.
- Deux types de recherche meilleur dôabord: A* et meilleur dôabord gloutonne.

Meilleur dœabord gloutonne

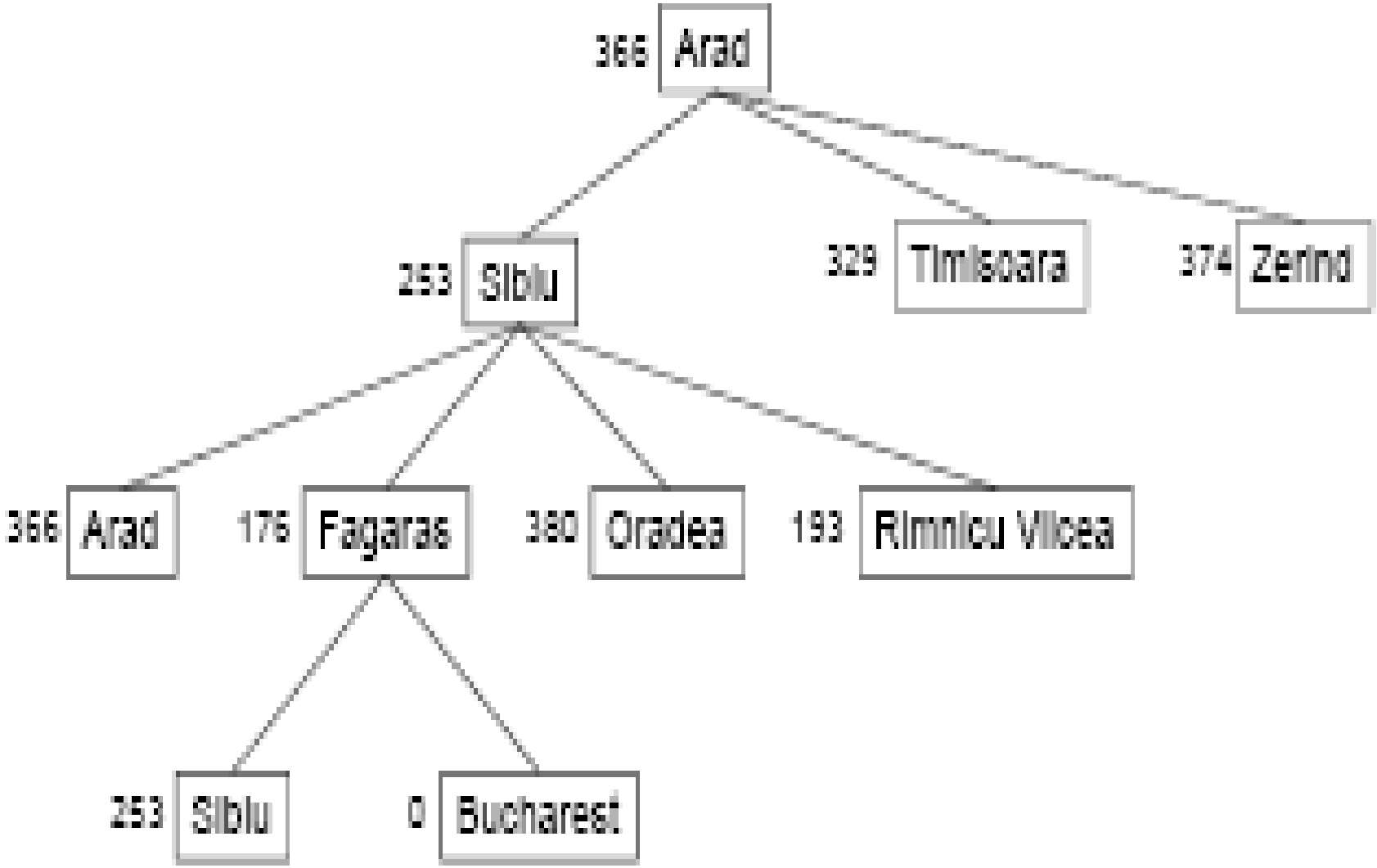
Åf(n) = h(n) = distance à vol dœoiseau

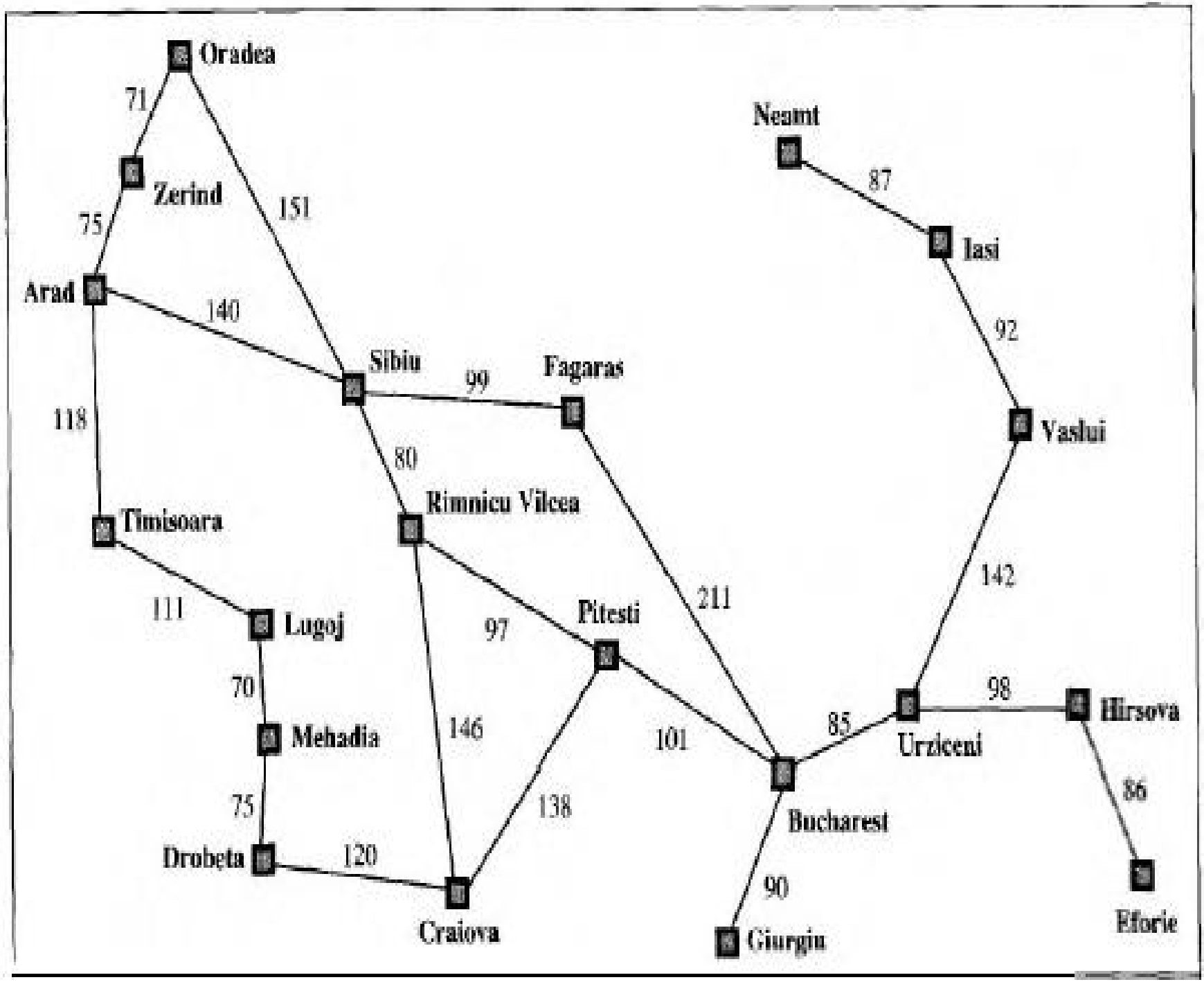
ÅChoisit toujours de développer le noeud le plus proche du but.

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

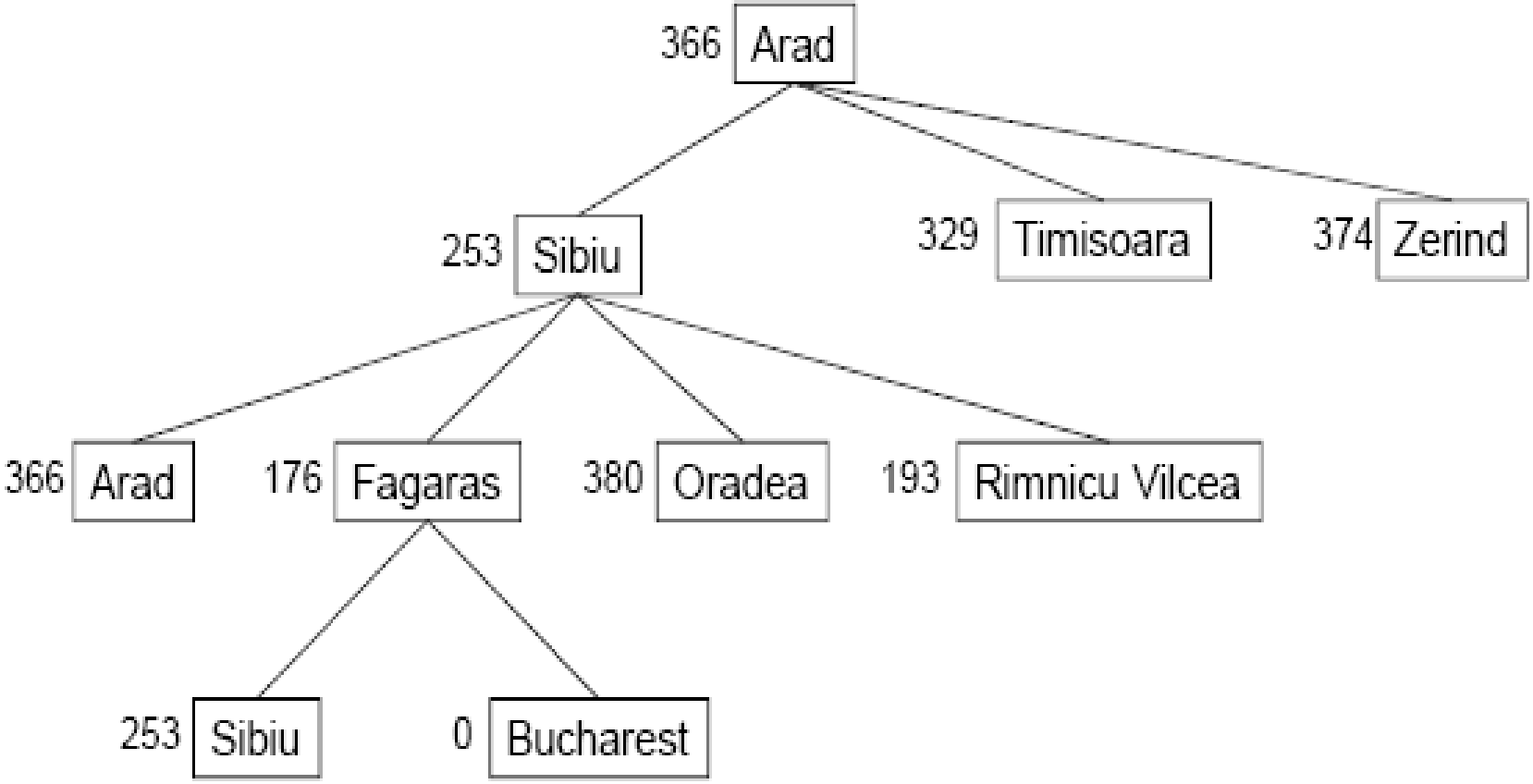
Heuristique: Distance en ligne droite (DLD) vers bucarest

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374


366 Arad
253 Sibiu
329 Timisoara
374 Zerind
366 Arad
176 Fagaras
380 Oradea
193 Rimnicu Vilcea
253 Sibiu
0 Bucharest

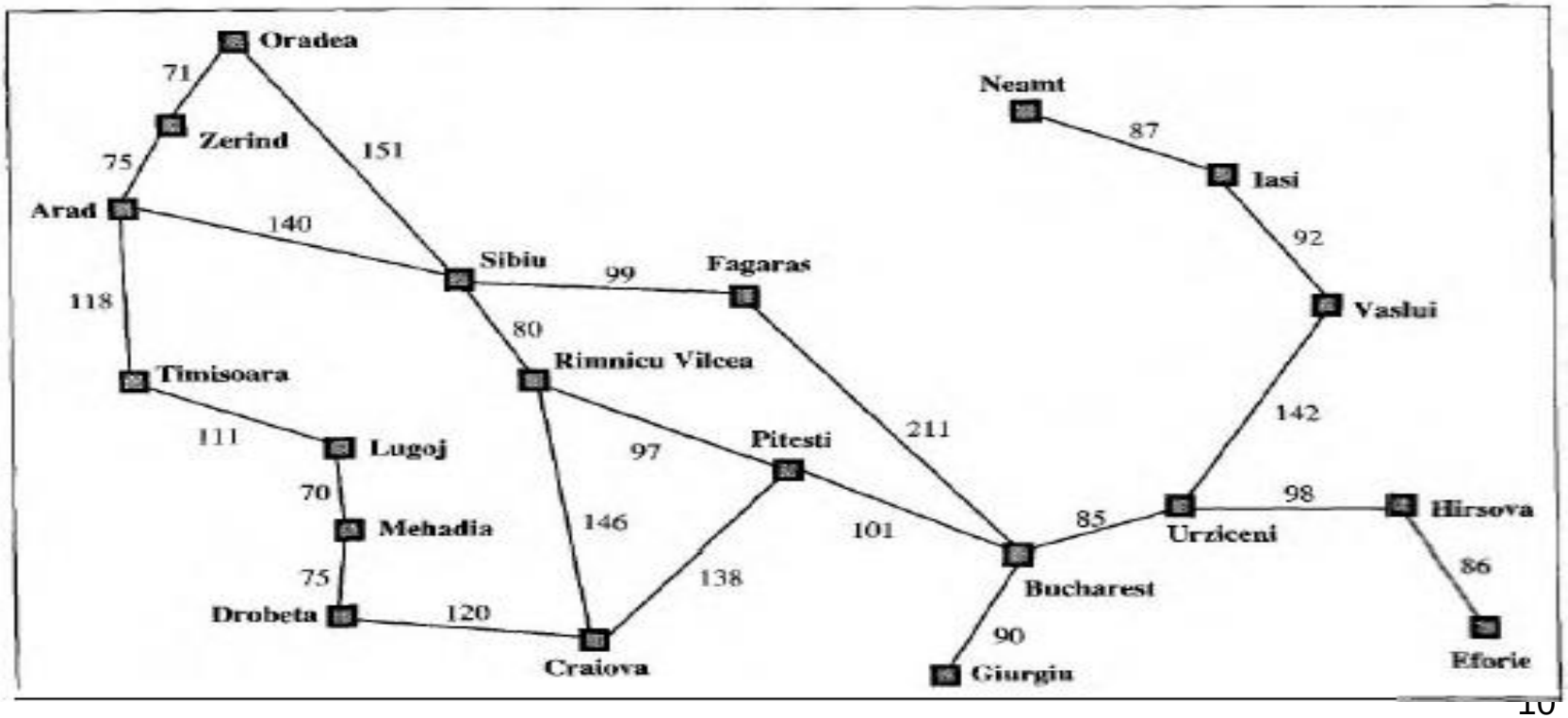

Oradea
71
Neamt
Zerind
87
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
211
Pitesti
Lugoj
97
98
70
Hirsova
Mehadia
146
101
85
Urziceni
75
86
138
Bucharest
Drobeta
120
90
Craiova
Giurgiu
Eforie

Exemple recherche gloutonne



Recherche gloutonne (suite)

- Le coût est minimal mais non optimal: le chemin via Sibiu et Fergas vers Bucarest compte 32 Km de plus que le chemin via Rimnicu Vilcea et Pitesti



Recherche gloutonne

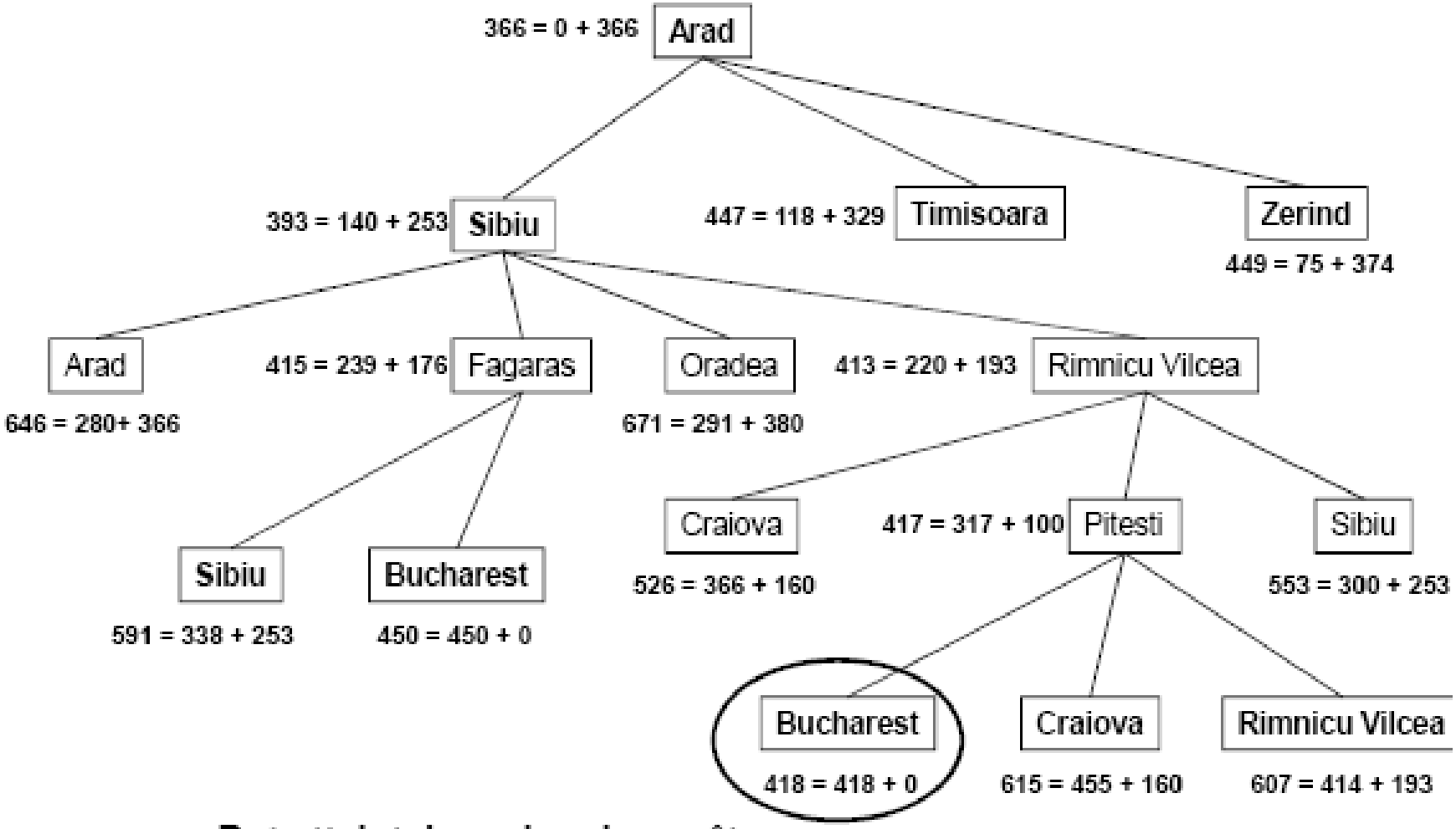
- L'exploration meilleur d'abord gloutonne ressemble à l'exploration en profondeur d'abord
- Elle préfère suivre un chemin tout au long vers le but, mais revient en arrière lorsqu'elle rencontre une impasse
- Sa complexité en temps et en espace est $O(b^m)$,
- Une bonne fonction heuristique permet de réduire cette complexité,
- L'amplitude de la réduction dépend de la **nature du problème** et de la **qualité de l'heuristique**

Algorithme A*

- Fonction d'évaluation: $f(n) = g(n) + h(n)$
  - *g(n)*: coût du noeud de départ jusqu'au noeud *n*
  - *h(n)*: coût estimé du noeud *n* jusqu'au but
  - *f(n)*: coût total estimé du chemin passant par *n*

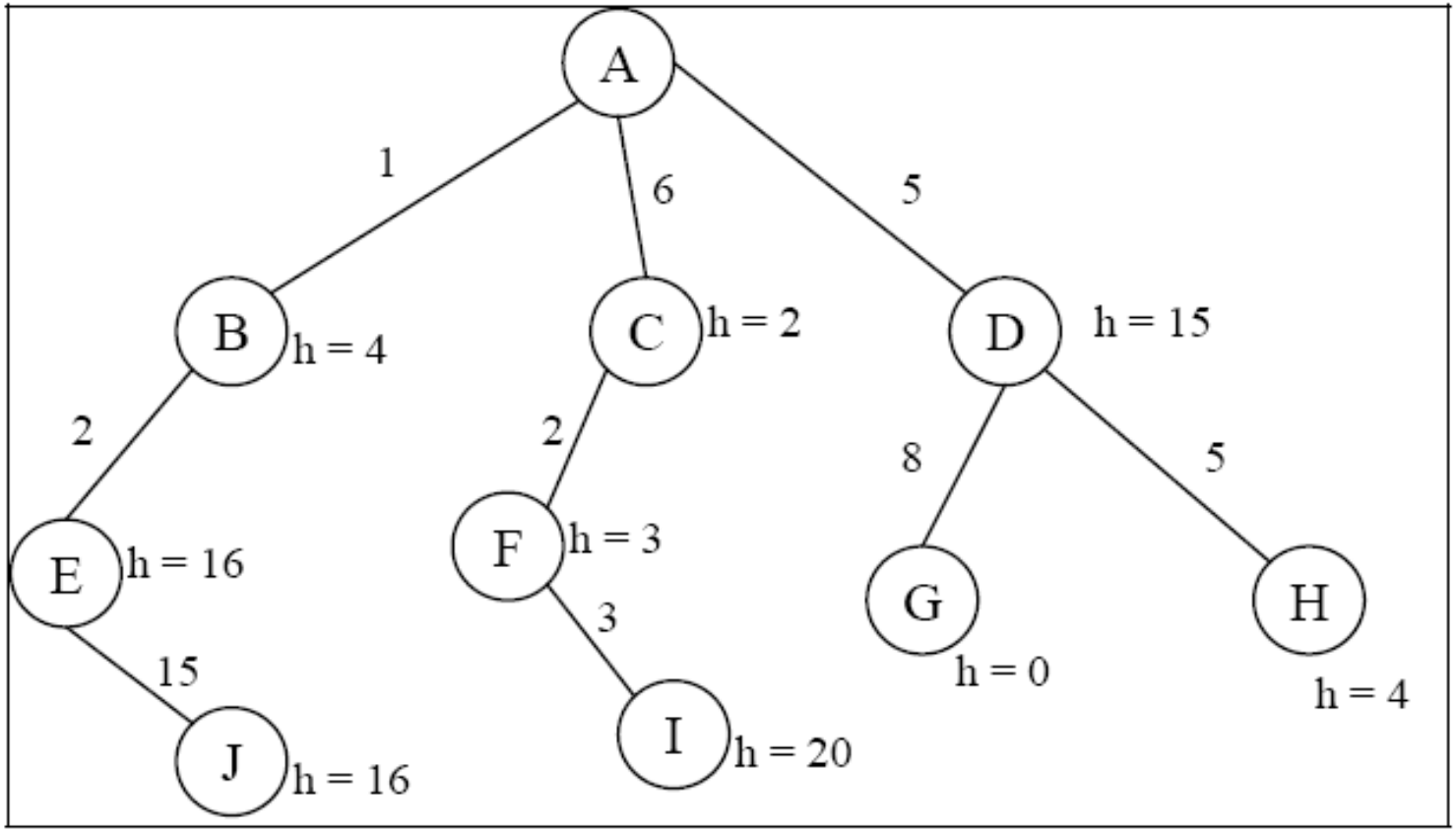
  pour se rendre au but.
- Principe : Si l'on veut trouver la solution la moins coûteuse, il est raisonnable d'essayer d'abord le nœud qui a la valeur la plus faible pour g(n)+h(n)
- Stratégie complète est optimale si la fonction heuristique h(n) est admissible (ne surestime pas le coût pour atteindre le but)
- Exemple : la distance en ligne droite DLD est une heuristique admissible, car le chemin le plus court entre deux points quelconque est la ligne droite. Donc l'Heuristique DLD ne peut jamais être une surestimation

Exemple A*



But atteint, la recherche arrête.

Exercice



Questions

le nombre placé à côté de chaque lien dénote le coût associé au lien entre les noeuds et *h* est le coût estimé du noeud jusqu'au but final G.

a) Donner l'ordre dans lequel les noeuds sont visités lorsqu'on utilise A*. Donner alors *f(n)* pour chacun des noeuds *n*.

b) Un algorithme « glouton » de type meilleur d'abord peut-il faire mieux? Combien de noeuds, un tel algorithme traverse-t-il avant d'atteindre le but G.

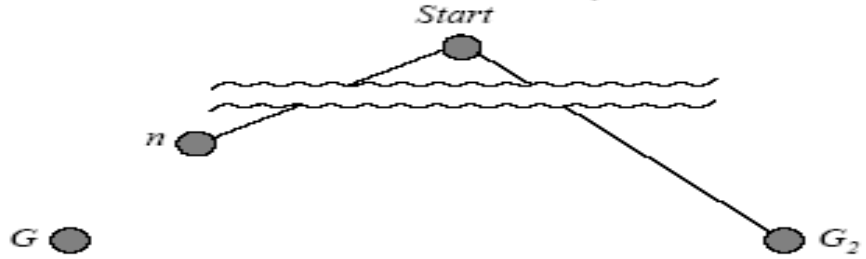
c) Si le noeud I est aussi un but, lequel (ou lesquels) parmi largeur d'abord, profondeur d'abord et profondeur itératif peut (ou peuvent) trouver la solution optimale? Donner pour chaque algorithme la liste des nî uds développés.

d) La fonction heuristique *h* pose un problème dans la mesure où elle surestime trop le coût de D à G. Quelle propriété de A* n'est plus remplie si *h* a ce problème?

Optimalité de A*

(from Russel&Norvig 2003)

Suppose some suboptimal goal G_2 has been generated and is in the queue.
Let n be an unexpanded node on a shortest path to an optimal goal G_1.



$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
&> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\
&\geq f(n) && \text{since } h \text{ is admissible}
\end{aligned}
$$

f(n) ≤ f(G1) < f(G2)

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

A* expands all nodes with $f(n) < C^*$
A* expands some nodes with $f(n) = C^*$
A* expands no nodes with $f(n) > C^*$

C* coût de la solution optimale

Ceci est valable dans le cas d'une exploration en arbre. Dans le cas d'une exploration en graphe cette démonstration ne tient plus

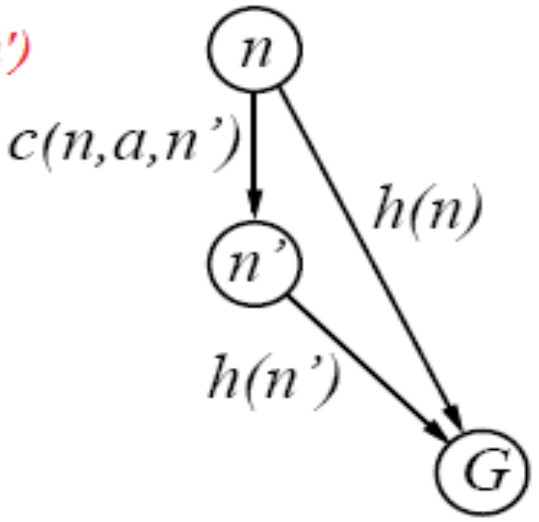
A* et EXPLORATION-EN- GRAPHE

- Dans l'exploration en graphe, des solutions sous-optimales peuvent être retournées car l'exploration en graphe peut rejeter le chemin dupliqué si celui-ci n'est pas le premier généré.
- 2 Solutions:
  - Étendre l'exploration en graphe pour qu'il rejette le plus couteux des 2 chemins trouvés vers le même nœud (il faut comptabiliser les nœuds visités)
  - s'assurer que le chemin optimal vers un état dupliqué est toujours le premier suivi. Pour ce faire il faut que *h(n)* soit consistante (appelée aussi **monotone**)

Consistance d'une heuristique

A heuristic is consistent if

$$h(n) \leq c(n, a, n') + h(n') \quad \textit{or } h(n)\text{-}h(n') <= c(n,a,n')$$



If h is consistent, we have

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

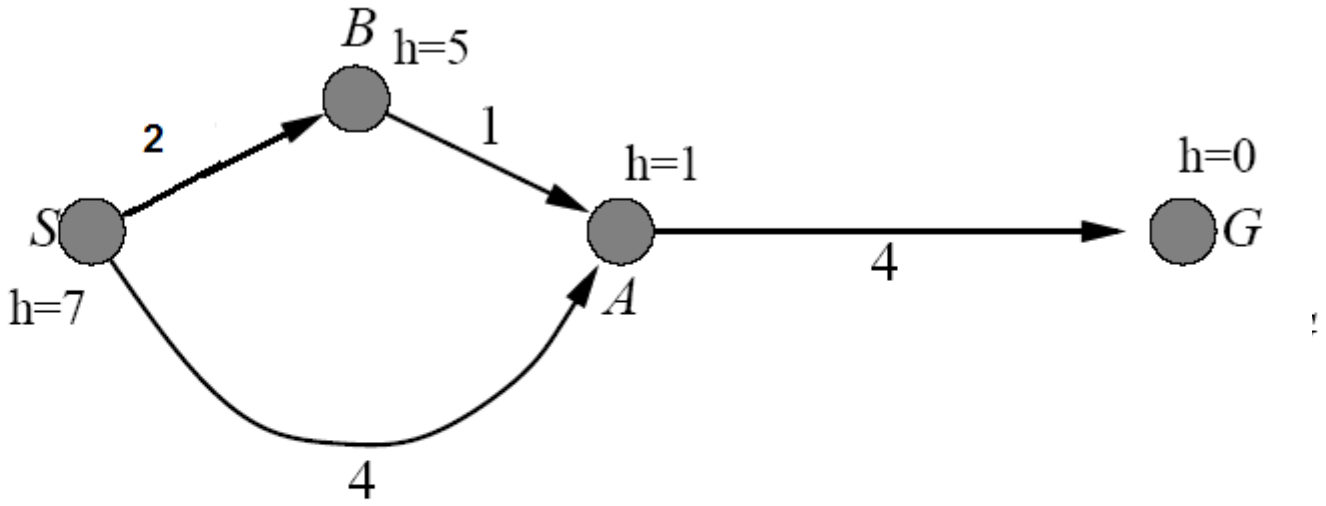
I.e., $f(n)$ *est non décroissante quelque soit le chemin*

Inégalité triangulaire: un côté d'un triangle ne peut pas être plus long que la somme des 2 autres côtés.

Une heuristique consistante est également admissible

Conséquence: A* avec exploration en graphe est optimale si h(n) est consistante.

Exemple



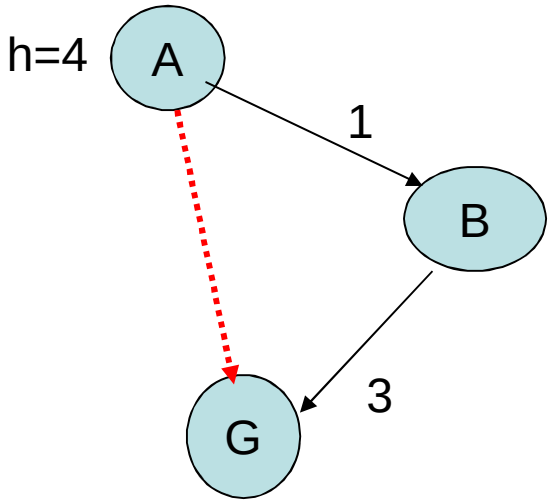
Un graphe avec une heuristique inconsistante avec laquelle la recherche dans le graphe ne retourne pas la solution optimale. h(B) ï h(A)= 4 > c(B,A)=1

Les successeurs de S sont A avec f=5 et B avec f=7. A est developpé en premier donc le chemin via B est écarté car A sera dans la liste fermée.

Récapitulatif

Å Heuristique admissible:

h(A)<=Coût de A à G



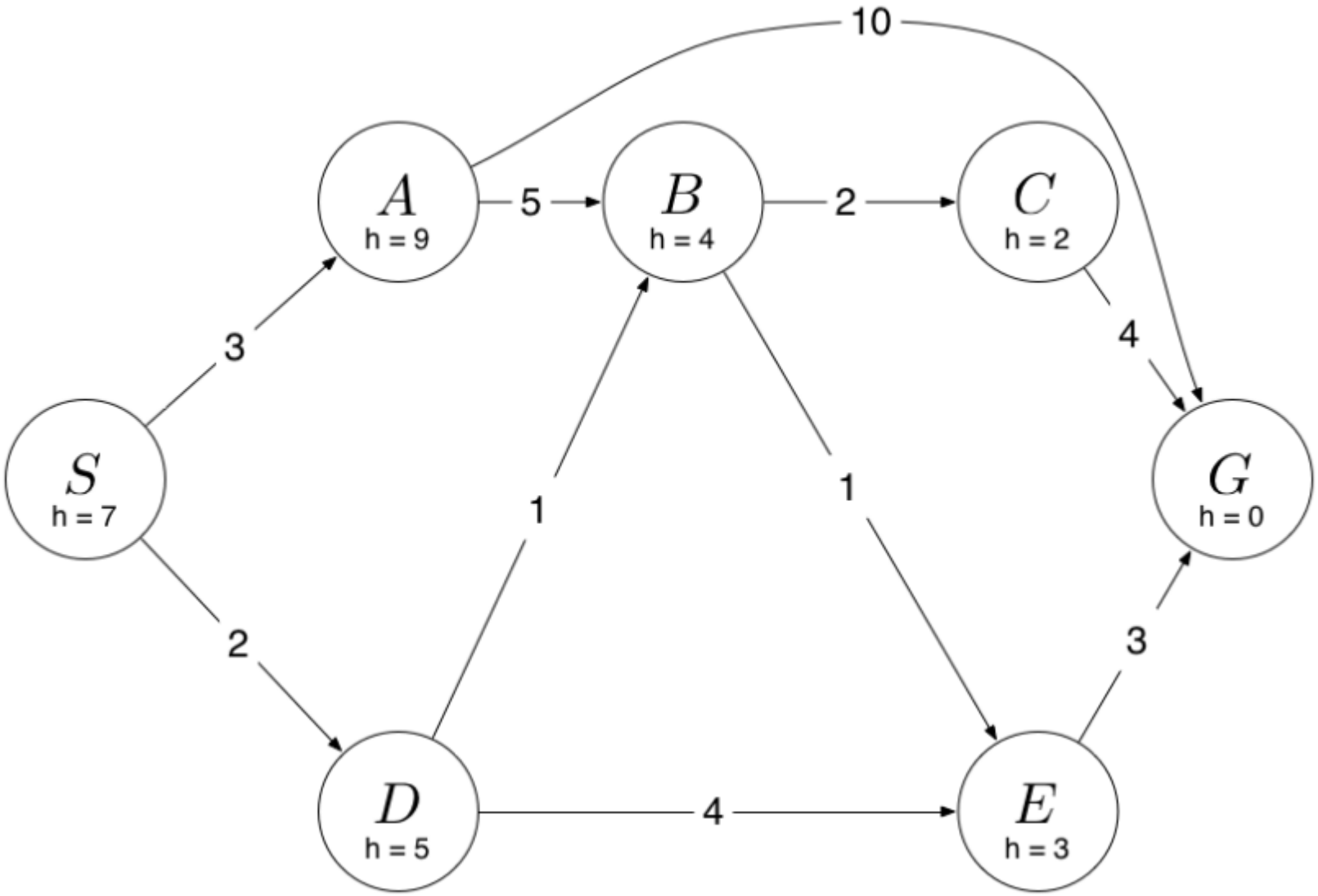
Å Heuristique consistante

h(A)-h(C) est supérieure au coût(A,C) donc **non consistante**.

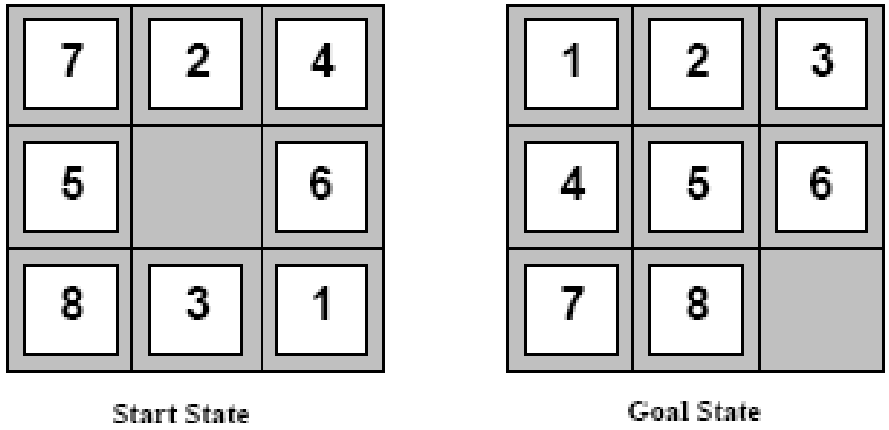
H=2 heuristique consistante

A* Conclusion

- A* ne développe pas les nœuds pour lesquels f(n)> C*: (le sous arbre en dessous de Timosoara n'est pas développé on dit qu'il est Élagué)
- A* possède une efficacité optimale: il n'y a pas d'algorithme qui garantisse de développer moins de nœuds que A*
- Tout algorithme qui ne développe pas tout les nœuds pour lesquels f(n)<C* court le risque de rater la solution optimale
- faiblesse de A*:
  - lorsque le nombre de nœuds autour du but est exponentiel, la croissance sera exponentielle
  - Le temps de calcul dépend de l'heuristique h(n)
  - Comme il conserve tous les nœuds générés en mémoire, A* est généralement à court de mémoire bien avant d'être à court temps. Donc A* n'est pas efficace pour de nombreux problème de grande taille
- Il existe d'autres algorithmes variante de A* qui ont résolus le problème d'espace sans sacrifier l'optimalité ni la complétude: IDA*, RBFS et SMA*


10
A
h = 9
5
B
h = 4
2
C
h = 2
3
4
S
h = 7
1
1
G
h = 0
2
3
D
h = 5
4
E
h = 3

Heuristique admissible



Soient 2 heuristiques:

h1(n): le nombre de pièces mal placées (h1(n) = 6 dans l'exemple)

h2(n): la somme des distances des pièces par rapport à leur position cible (**distance city block** ou **distance de Manhattan**)

h2(n)=4+0+3+3+1+0+2+1=14

Dominance

- Si *h2*(*n*) Ó*h1*(*n*) pour tout *n* (les deux étant admissibles), alors *h2*(*n*) domine *h1*(*n*) et par conséquent *h2*(*n*) est meilleure que *h1*(*n*).
- Il est toujours préférable de choisir lôheuristique dominante, car elle va développer moins de nî uds
  - Tous les nî uds avec f(n) < C* vont être développés.
  - Donc, tous les nî uds avec h(n) < C* - g(n) vont être développés.
  - Par conséquent, tous les nî uds développés par *h2*(*n*) vont aussi être développés par *h1*(*n*), et *h1*(*n*) peut aussi développé dôautres nî uds.

Comment trouver une heuristique?

- Une méthode pour trouver une heuristique est de simplifier le problème. (problème relaxé)
- Un problème relaxé est un problème contenant moins de contraintes sur les actions

- *Règle: Le coût d'une solution optimale à un problème relaxé est une heuristique admissible pour le problème d'origine*

Exemple

- Exemple jeu du taquin:
  - une pièce peut passer du carré A au carré B si A est adjacent à B et si B est vide,
- On peut générer 3 problèmes relaxés en supprimant lôune des conditions ou les deux:
  - (a) Une pièce peut passer du carré A au carré B si A est adjacent à B
  - (b) Une pièce peut passer du carré A au carré B si B est vide
  - (c) Une pièce peut passer du carré A au carré B

On peut dériver *h2* (distance de Manhatan) à partir du problème relaxé (a). *h2* serait le score correct si on déplaçait chaque pièce tour à tour vers sa destination

Question: Quelles sont les heuristiques que lôon peut dériver des problèmes relaxés (b) et(c)?

Problème relaxé (suite)

- Il est crucial que les problèmes relaxés générés puissent être résolus sans exploration car la simplification des règles permet de décomposer le problèmes en sous problèmes indépendants
- ABSOLVER (Prieditis 1993) est un programme qui peut générer automatiquement des heuristiques à partir de définition de problèmes en utilisant la méthode du « problème relaxé » et différentes autres techniques

Apprentissage de la fonction heuristique

- On résoud de nombreux problèmes de taquin et chaque solution optimale à un problème fournit un exemple à partir duquel h(n) peut être appris
- Chaque exemple est constitué d'un état du chemin solution et du coût réel
- À partir de ces exemples on peut utiliser un algorithme d'*apprentissage inductif* pour construire h(n)
- D'autres techniques peuvent être utilisées: réseaux de neurones, arbre de décision, apprentissage par renforcement