WIESLAW ZIELONKA

INTRODUCTION À L'INTELLIGENCE ARTIFICIELLE ET LA THÉORIE DE JEUX

RAPPEL : RECHERCHE NON-INFORMÉE

Recherche non-informée : pas d'information sur le coût minimal entre le sommet courant et destination.

Les algorithmes de recherche non-informée:

- recherche en largeur (LIFO)
- recherche en profondeur (FIFO)
- recherche à coût uniforme (file de priorité)

RECHERCHE INFORMÉE (AVEC AIDE DE HEURISTIQUE)

- appelé parfois BEST FIRST SEARCH
- utilise la fonction heuristique h(n) — le coût estimé de chemin le plus court de sommet n jusqu'à la destination
- peut utiliser g(n) — le coût de chemin courant de sommet initial vers le sommet courant n (calculé par l'algorithme)
- f(n) — la fonction d'évaluation, à chaque itération on cherche dans la frontière le sommet n avec f(n) minimal (où f(n) dépend de h(n) et peut-être de g(n)). A chaque itération on développe le sommet n avec f(n) minimal.

HEURISTIQUE

pour le problème de plus court chemin entre les villes

h(n) - la distance à vol d'oiseau de la ville n jusqu'à la ville destination

ALGO DE RECHERCHE GLOUTON (GREEDY BEST-FIRST SEARCH)

- Dans l'algo de recherche glouton

  f(n) := h(n)

  et on utilise la file de priorité où pour chaque sommet n on stocke la valeur h(n)

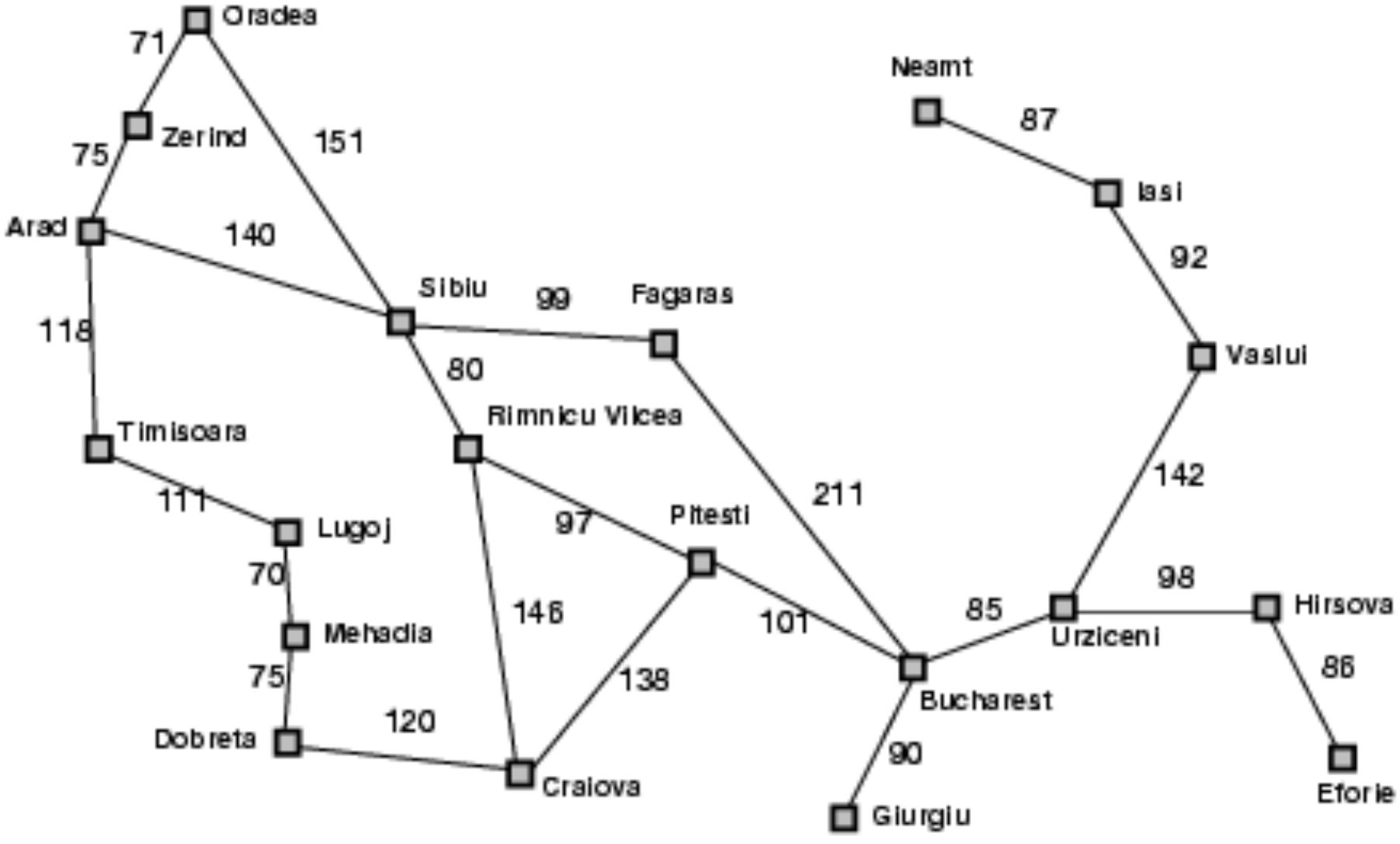
- à chaque étape on prend de la file de priorité le sommet n avec f(n)=h(n) minimal

- le coût g(n) du chemin de sommet source vers n n'est pas pris en compte

- la version tree-like de l'algorithme n'est pas complète

- la version graph-like est complète mais pas optimale

Romania with step costs in km

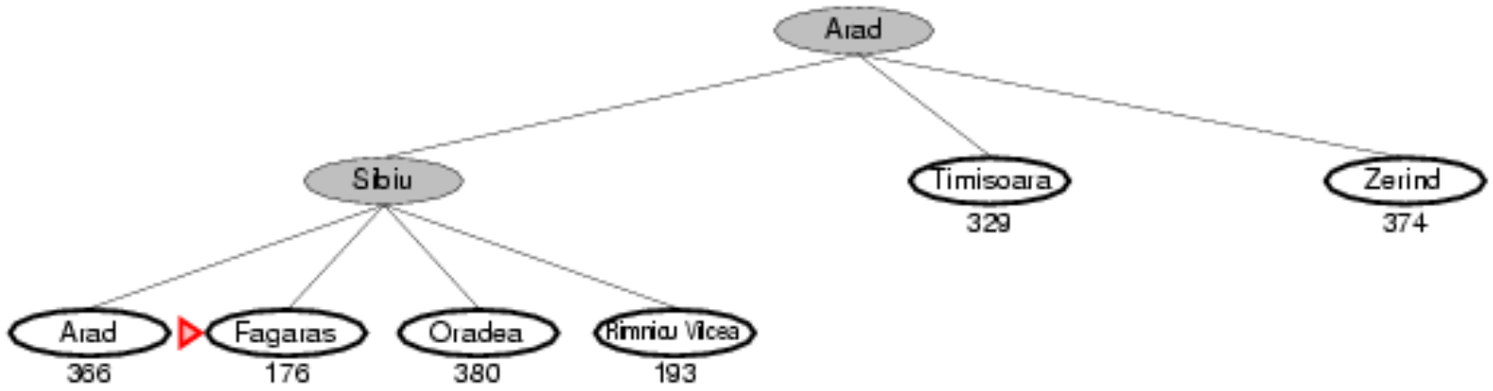


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

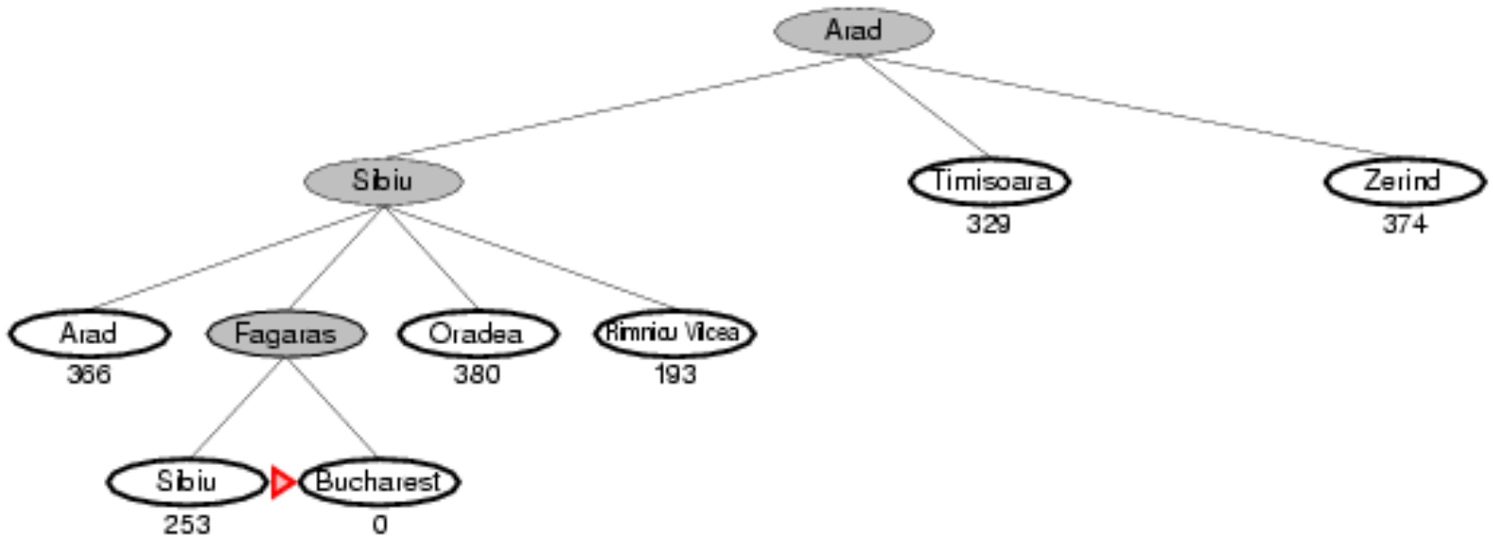
Greedy best-first search example

Greedy best-first search example

Greedy best-first search example



Greedy best-first search example



ALGO DE RECHERCHE GLOUTON (GREEDY BEST-FIRST SEARCH)

- complet ?
- optimal ?

ALGO COÛT UNIFORME

L'algorithme de recherche informée avec h(n)=0 pour tout n et f(n)=g(n) (c'est-à-dire la heuristique triviale) on obtient la recherche à coût uniforme (déjà étudiée).

ALGORITHME A^* - MINIMISER LE COÛT TOTAL

- dans chaque sommet n on stocke g(n) et
$$f(n)=g(n)+h(n)$$
- la file de priorité pour les valeurs de f(n), à chaque étape on choisit un noeud avec f(n) minimal

ALGORITHME A^*

1. mettre le sommet initial s dans FRONTIERE, g(s)=0, f(s)=h(s)
2. Si FRONTIERE vide, sortir avec échec
3. retirer de FRONTIERE et mettre dans FERMES le noeud n avec la valeur f(n) minimale
4. si n est le sommet destination alors terminer et retracer la solution : le chemin de s à n
5. développer n pour engendrer tous ses successeurs. Pour tout successeur m de n :
   A. si m n'est ni dans FRONTIERE ni dans FERMES calculer f(m)=g(m) +h(m), où g(m)=g(n)+coût(n, m)
6. aller à 2

INFRASTRUCTURE POUR L'ALGORITHME A^*

- n.état — état du noed
- n.parent — le parent du noeud
- n.action — l'action qui, appliquée au parent, donne l'état du noued
- n.coût — le coût du chemin de la racine jusqu'au noeud, autrement la valeur g(n)
- n.f — la valeur f(n), la file de priorité selon les valeur n.f

```
fonction créer_noeud(Noeud parent, Action  action)
                                          retourne Noeud
  n = new noeud()
  n.état = transition(parent.état, action)
  n.action = action
  n.coût = parent.coût + coût(parent.état, action)
  n.parent = parent
  n.f = n.coût + h(n.état)
```

INFRASTRUCTURE POUR L'ALGORITHME A^*

```
fonction créer_noeud_initial(État état)
                                        retourne Noeud
  n = new noeud()
  n.état = état
  n.action = null
  n.coût = 0
  n.parent = null
  n.f = 0
```

HEURISTIQUE ADMISSIBLE

la fonction heuristique h est admissible si pour chaque sommet n

> h(n) <= coût réel du chemin le plus court de n vers un état destination

donc heuristique admissible est une sous-estimation du coût minimal réel.

HEURISTIQUE CONSISTANTE

Une heuristique est consistante (ou cohérente) si elle satisfait l'inégalité de triangle:

pour chaque couple de sommets n et m et action a telle que

m=transition(n,a) on a

h(n) <= coût(n,a,m) + h(m)

coût(n,a,m) - coût de l'action a

HEURISTIQUE CONSISTANTE

heuristique h

si h est consistante alors h est admissible

On suppose aussi que

h(d)=0 pour tout sommet d de destination.

OPTIMALITÉ DE A*

A. Si la heuristique h est admissible alors la version tree-search de l'algorithme A* est optimale.

B. Si la heuristique h est consistante alors la version graph-search de l'algorithme A* est optimale.

COMPARER LES HEURISTIQUES ADMISSIBLES

soit h1 et h2 admissibles.

Si pour chaque sommet d h1(d) <= h2(d) alors h2 comme une meilleure estimation de la distance minimale entre de et l destination (meilleure = plus proche de la distance minimale réelle).

A* avec h2 a une complexité en temps inférieure de A* avec h1.

HEURISTIQUES

ABSOLVER - un programme qui trouve des heuristiques pour les problèmes relâchés

Il a trouvé la meilleures heuristique pour le jeu taquin et la première heuristique intéressante pour le cube de Rubik.

si h1,...,hn sont admissible alors

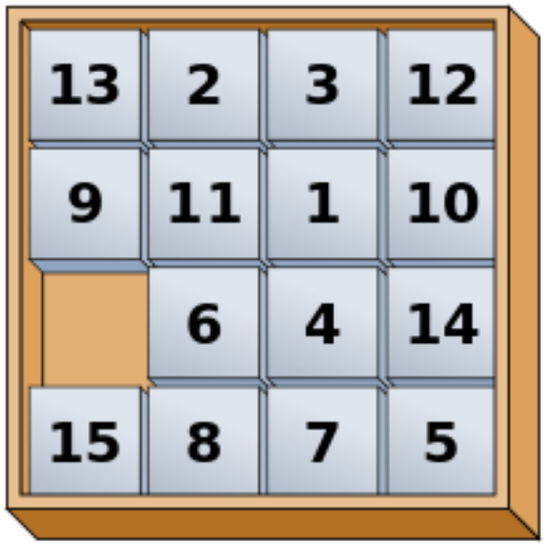
h = max{ h1,...,hn }

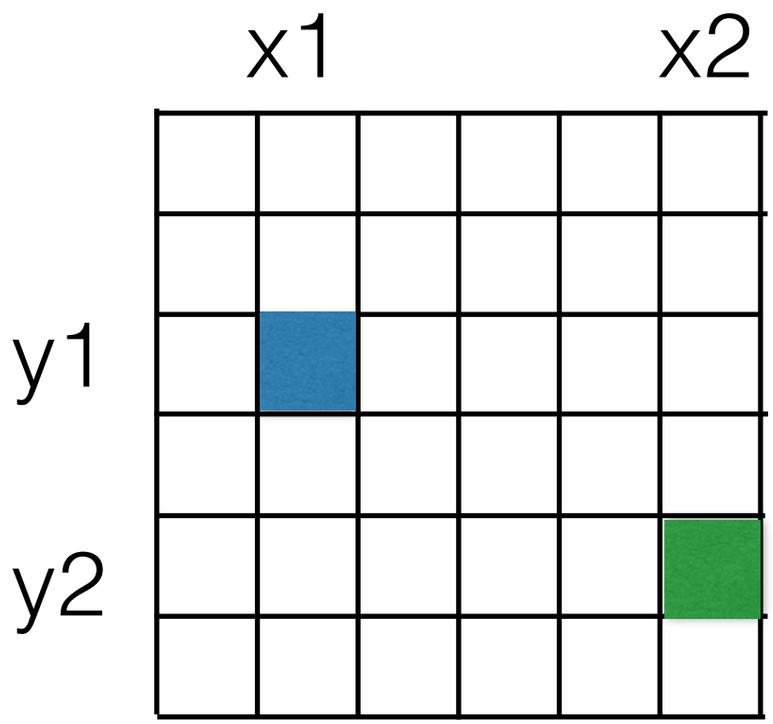
est admissible et domine chaque hi

COMMENT TROUVER LES HEURISTIQUES OPTIMALES?

- en assouplissant les conditions (ajoutant les actions)
- en travaillant avec des sous-problèmes

Jeu de taquin
heuristique admissible





distance de Manhattan=|x1-x2|+|y1-y2|

h=somme sur toutes les pièces de la distances
de Manhattan entre la position courante d'une pièce et
la position finale de la pièce

Heuristique admissible par assouplissement de règles

L'heuristique précédente pour le jeu de taquin est obtenue en assouplissant les règles du jeu :

- une pièce peut être déplacée sur la case voisine même si la case voisine est occupée

- distance de Manhattan = distance minimale exact entre la configuration courante et la configuration finale avec les nouvelles règles du jeu

Heuristiques admissibles à partir de sous-problèmes

x	x	x	x
x	4	x	2
1		x	x
x	x	x	3

1	2	3	4
x	x	x	x
x	x	x	x
x	x	x	

h=le nombre de pas pour passer de la configuration gauche à droite avec les règles de taquin habituelles

Une fois la deuxième configuration atteinte la heuristique est 0 : ni très efficace ni très utile

Heuristiques admissibles à partir de sous-problèmes disjoints

x	x	x	x
x	4	x	2
1		x	x
x	x	x	3

1	2	3	4
x	x	x	x
x	x	x	x
x	x	x	

h1=le nombre de pas pour passer de la configuration gauche à droite avec les règles de taquin habituelles en ne comptant que les déplacement de pièces numérotées

h2=même chose avec les pièces 5,6,7,8 visibles

h3= même chose acec les pièces 9,10,11,12 visibles

h4=même chose avec les pièces 13,14,15 visibles

h=h1+h2+h3+h4

Pour la taille 5 sur 5 accélération de 1000000 par rapport à Manhattan

HEURISTIQUES ADMISSIBLES A PARTIR DE SOUS-PROBLEMES

taquin 4 sur 4

remplacer les pièces 1,2,3,4 par une pièce unique *

Résoudre taquin avec l'état but

```
 *   *  *  *
 5   6  7  8
 9  10 11 12
13  14 15
```

Prendre comme la heuristique le nombre de coups minimal dans ce sous- problème.
Résoudre le taquin pour d'autres sous-problèmes similaires.
Constituer une base de données avec les sous-problèmes (pattern database).

Accélération 10000 par rapport à la distance Manhattan pour le taquin de 4 sur 4,

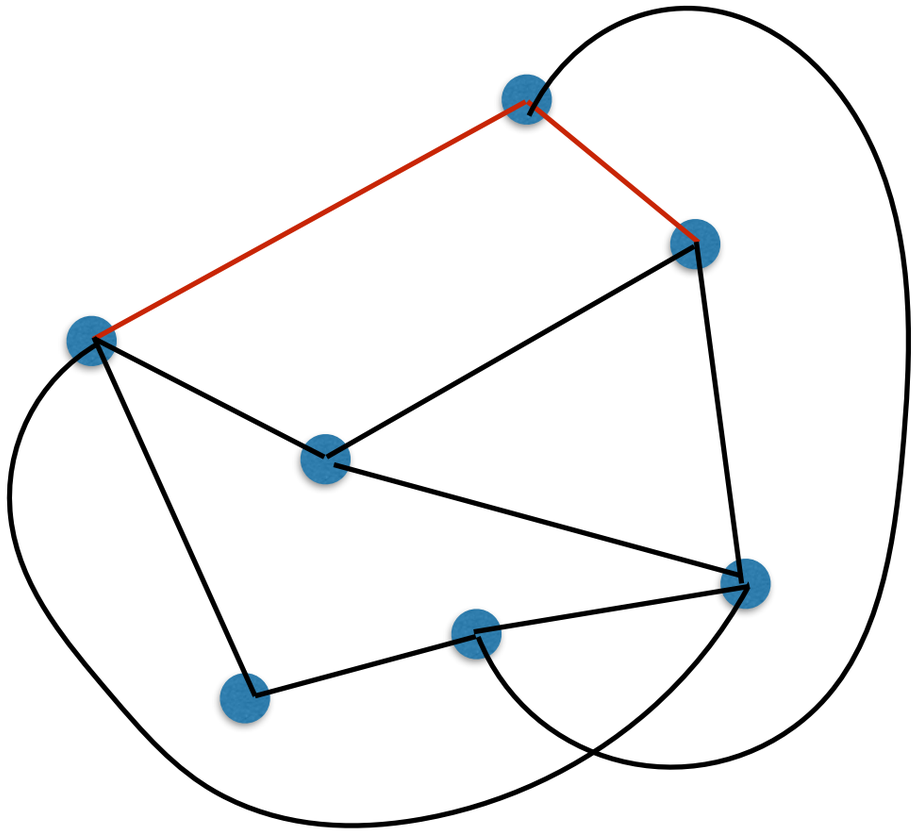
l'accélération de l'ordre d'un million pour le taquin de 5 sur 5.

HEURISTIQUES ADMISSIBLES A PARTIR DE SOUS-PROBLEMES

Heuristique de Gasching : on peut déplacer une pièce quelconque sur une position vide. Comment calculer efficacement cette heuristique?

Heuristique pour le problème de voyageur de commerce

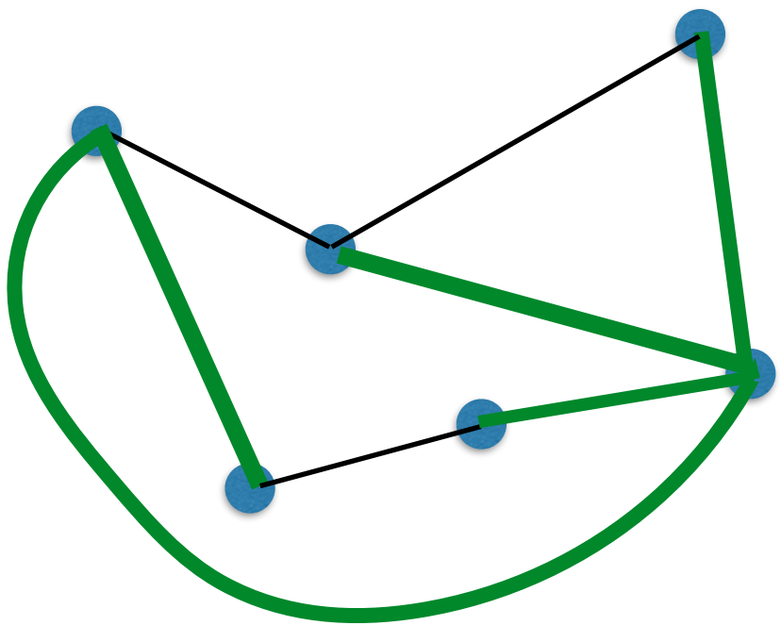
Pour un chemin partiel estimer le coût minimal pour terminer le tour.



tour partiel

Heuristique pour le problème de voyageur de commerce

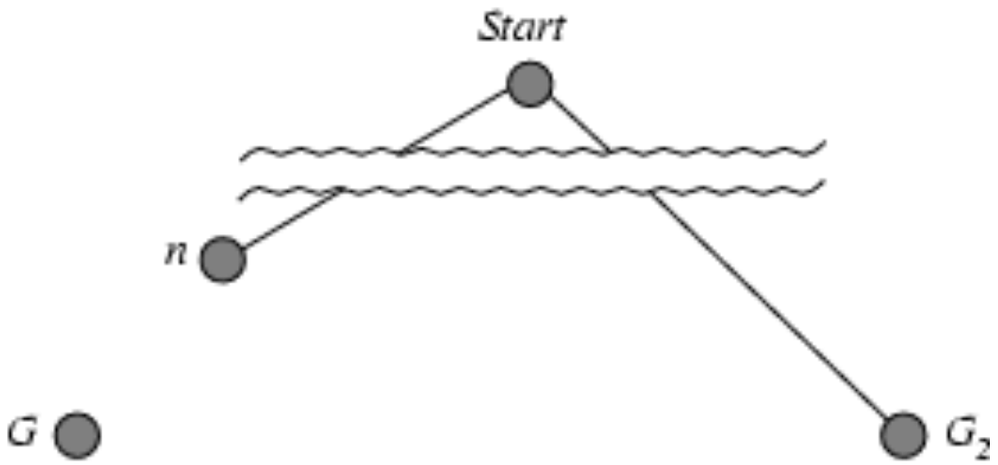
Pour un chemin partiel estimer le coût minimal pour terminer le tour.



- supprimer les sommets à l'intérieur du chemin et les arêtes adjacentes
- h = le poids de l'arbre recouvrant de poids minimal

Optimalité de A* (demonstration)

- Supposons que l'état destination G_2 *non optimal* a été généré et se trouve dans la frontière. Soit *n* un autre état dans la frontière tel que le plus court chemin vers un état destination optimal *G* passe par *n.*



g(k) - la longueur du plus court chemin de l'état source vers k

h - heuristique admissible

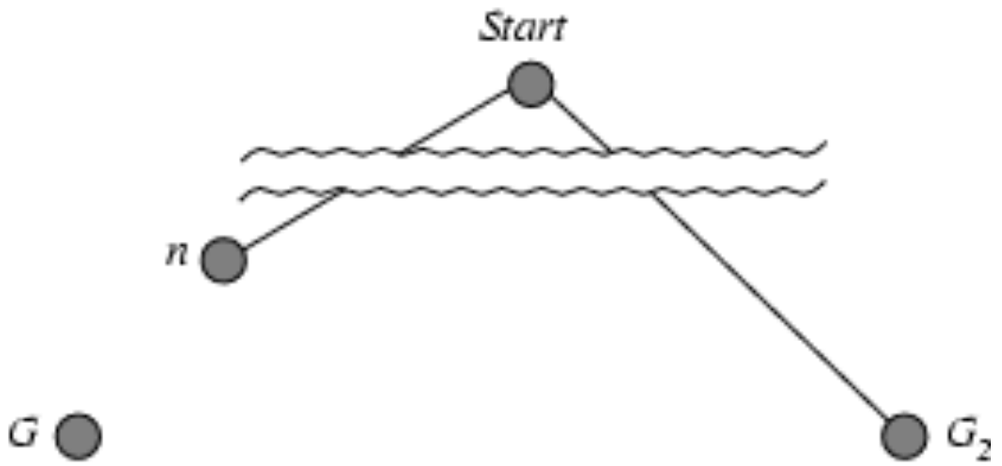
f(k)=g(k)+h(k)

- $f(G_2) = g(G_2)$ parce que $h(G_2) = 0$
- $g(G_2) > g(G)$ parce G_2 n'est pas optimal
- $f(G) = g(G)$ parce que $h(G) = 0$

» $f(G_2) > f(G)$ conclusion de ci-dessus

Optimalité de A* (demonstration)

- Supposons que l'état destination G_2 *non optimal* a été généré et se trouve dans la frontière. Soit *n* un autre état dans la frontière tel que le plus court chemin vers un état destination optimal *G* passe par *n.*



h*(k) - la longueur de chemin le plus court de k vers un état destination

- $f(G_2)$ > f(G) transparent précédent
- h(n) ≤ h*(n) parce que h est admissible
- g(n) + h(n) ≤ g(n) + h^*(n)

f(n) ≤ f(G)

» Donc $f(G_2) > f(n)$, et A* sélectionne n et pas G_2 pour developper (et pour sortir de la frontière)