# Recherche guidée dans un graphe : $A^\star$

M1 IDD 2019–2020 *Intelligence Artificielle*

Stéphane Airiau

LAMSADE

# Résoudre avec des algorithmes de base

- expansion d'un graphe en largeur d'abord (**"breadth-first search" (BFS)**)
- expansion d'un graphe en profondeur d'abord (**"depth-first search" (DFS)**)
- expansion d'un graphe avec côut uniforme (**"Uniform-cost search"**) : développer le noeud qui a le coût le plus bas.
- recherche en profondeur limitée (**"depth-limited search"**) : utilise DFS jusqu'à une profondeur $l$ donnée
  cela évite le problème du chemin infini, mais pause problème si la solution est plus profonde que $l$.
- recherche en profondeur itérative (**"iterative deepening DFS"**) : combine DFS et BFS : itérativement utilise DFS jusqu'à une profondeur de 1, 2, … jusqu'à trouver le but.
  les états sont générés de nombreuses fois
- recherche bidirectionnelle (**"Bidirectional search"**) : effectue deux recherche : une de l'état initial vers l'état final, l'autre dans le sens inverse (donc depuis l'état final).

# Répétition des états

On ajoutera à ces stratégies la faculté de se souvenir ou non des noeuds visités

- sans mémoire : **"tree-search"** risque que les états se répètent
  ⇒risque de boucle
- avec mémoire : **"graph-search"** nécessite des ressources de stockage
  évite les boucles !

⇒ on ajoute une liste d'états visités (parfois appelée "explored set" ou "closed list")

## Critères pour comparer ces algorithmes

- **Complétude** : a-t-on une garantie de trouver une solution quand elle existe ?
- **Complexité du temps de calcul** : combien de temps a-t-on besoin (dans le pire des cas)
- **Complexité de l'espace mémoire** : combien d'espace mémoire a-t-on besoin (dans le pire des cas)
- **Optimalité** : est-ce-que la solution trouvée est optimale ?

# Comparaison

| | BFS | Uniform-Cost | DFS | Depth-limited | Iterative-deepening | Bidirectional |
|---|---|---|---|---|---|---|
| complet ? | ✔ ? ✘ | ✔ ? ✘ | ✔ ? ✘ | ✔ ? ✘ | ✔ ? ✘ | ✔ ? ✘ |
| complexité temps | $\mathcal{O}(?)$ | $\mathcal{O}(?)$ | $\mathcal{O}(?)$ | $\mathcal{O}(?)$ | $\mathcal{O}(?)$ | $\mathcal{O}(?)$ |
| complexité mémoire | $\mathcal{O}(?)$ | $\mathcal{O}(?)$ | $\mathcal{O}(?)$ | $\mathcal{O}(?)$ | $\mathcal{O}(?)$ | $\mathcal{O}(?)$ |
| optimal ? | ✔ ? ✘ | ✔ ? ✘ | ✔ ? ✘ | ✔ ? ✘ | ✔ ? ✘ | ✔ ? ✘ |

- $b$ sera le facteur de branchement de l'arbre
- $d$ est la profondeur de la solution la moins profonde
- $m$ est la profondeur maximale de l'arbre de recherche
- $l$ est la limite de profondeur

## Guider la recherche grace à des heuristiques

On se place dans les problèmes où il y a une **fonction de coût** (positive).

- Les techniques précédentes trouvent des solutions,
  mais on peut trouver des solutions de manière plus rapide

⇨ utilisation de **connaissance** sur la structure du problème

problèmes potentiels :

- d'où vient cette connaissance ?
- cette connaissance doit être correcte !

Rappel : Uniform cost développe le noeud qui a le coût le plus bas.

⇨ le coût le plus bas est une **fonction d'évaluation**, on la notera $f$.

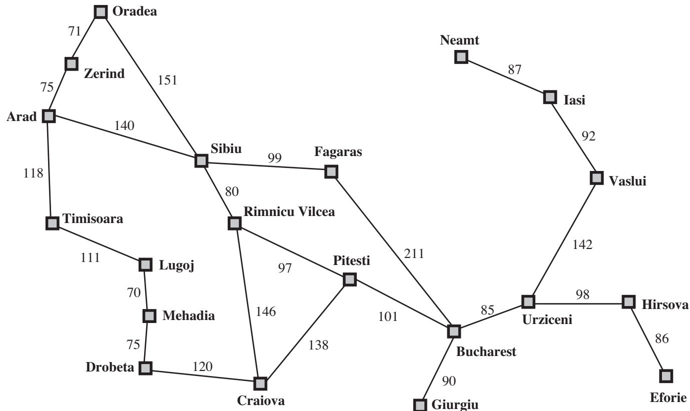
pour le noeud $n$, $f(n)$ indique si $n$ est prometteur

$f(n)$ doit indiquer la qualité de la solution *qui passe* par $n$

- évidemment, si on était devin et on connaissait $f$, il n'y aurait pas de problème à résoudre puisqu'on saurait quel noeud développer !

⇨ on veut développer des **heuristiques** pour estimer $f$.

# Example : Going from Arad to

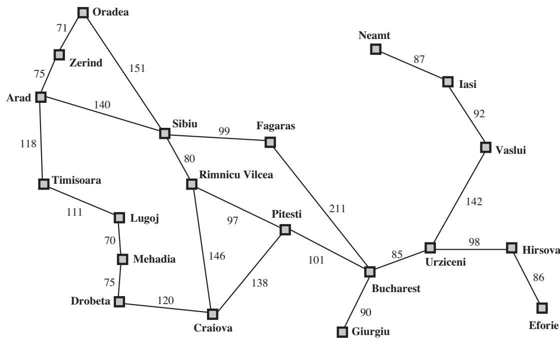


# Uniform-cost search

# Greedy-search // recherche gloutonne

- Uniform-cost : la fonction d'évaluation était le coût jusqu'à présent
- on peut essayer d'essayer d'estimer la distance jusqu'au but
- ➾ $h(v)$ va représenter la distance à vol d'oiseau entre une ville $v$ et Bucharest.
- il n'y a rien dans la description du problème qui parle de distance à vol d'oiseau ! On a utilisé notre *"sens commun"*
- il faut trouver cette distance à vol d'oiseau !
  (mesure sur une carte par exemple, ou calcul avec les coordonnées GPS)

# exemple



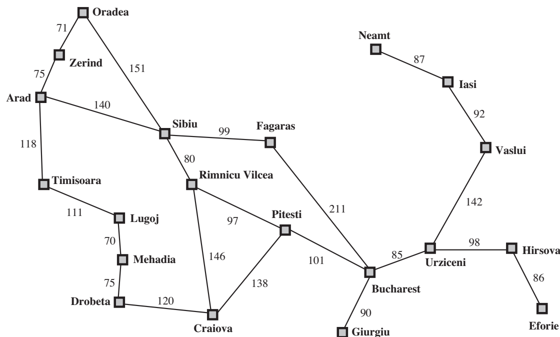

Distance à vol d'oiseau jusqu'à Bucharest

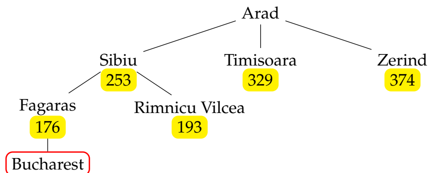
| | | | |
|---|---|---|---|
| Arad | 366 | Mehadia | 241 |
| Bucharest | 0 | Neamt | 234 |
| Craiova | 160 | Oradea | 380 |
| Drobeta | 242 | Pitesti | 100 |
| Eforie | 161 | Rimnicu Vilcea | 193 |
| Fagaras | 176 | Sibiu | 253 |
| Giurgiu | 77 | Timisoara | 329 |
| Hirsova | 151 | Urziceni | 80 |
|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

# exemple

Distance à vol d'oiseau jusqu'à Bucharest

| | | | |
|---|---|---|---|
| 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 |



# Discussion

- on a trouvé une solution rapidement
- mais la solution **n'**est **pas** optimale ! La solution a un coût de 450, et on peut trouver plus court !
- que se passe-t-il ?
  - parfois, la solution optimale doit "s'éloigner" du but temporairement pour aller plus vite ensuite.

⇒ trouver une meilleure fonction d'évaluation

# A$^\star$

Soit $n$ un noeud. On note

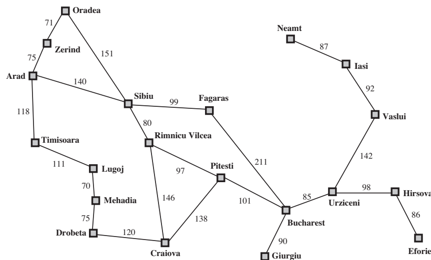
- $g(n)$ le coût pour atteindre le noeud $n$ depuis la racine
- $h(n)$ le coût **estimé** du chemin le moins couteux vers un noeud final.
- $f(n) = g(n) + h(n)$ est donc une estimation du coût minimal passant par le noeud $n$.

La stratégie d'expansion des noeuds est de développer le noeud avec la plus faible valeur de $f = g + h$.

On a toujours deux versions

- **tree-search** : on ne se rappelle pas des états visités
- **graph-search** : on se rappelle des états visités

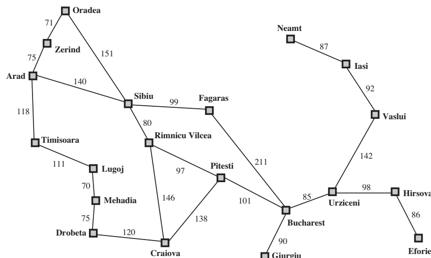
# exemple




Distance à vol d'oiseau jusqu'à Bucharest

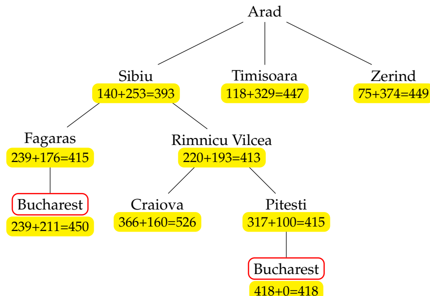
| | | | |
|---|---|---|---|
| Arad | 366 | Mehadia | 241 |
| Bucharest | 0 | Neamt | 234 |
| Craiova | 160 | Oradea | 380 |
| Drobeta | 242 | Pitesti | 100 |
| Eforie | 161 | Rimnicu Vilcea | 193 |
| Fagaras | 176 | Sibiu | 253 |
| Giurgiu | 77 | Timisoara | 329 |
| Hirsova | 151 | Urziceni | 80 |
|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

# exemple




Distance à vol d'oiseau jusqu'à Bucharest

| | | | |
|---|---|---|---|
| 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 |



## Condition d'Optimalité (tree-search)

**Definition** (admissibilité)

Une heuristique est **admissible** ssi elle **sous estime** toujours le coût.

Dans l'exemple, par définition, la distance à vol d'oiseau sous estime la distance totale, donc l'heuristique est admissible.

**Theorem**

$A^\star$ dans la version **tree-search** est optimal si la fonction heuristique $h$ est admissible

# Condition d'Optimalité (tree-search)

**Definition** (admissibilité)

Une heuristique est **admissible** ssi elle **sous estime** toujours le coût.

Dans l'exemple, par définition, la distance à vol d'oiseau sous estime la distance totale, donc l'heuristique est admissible.

**Theorem**

$A^\star$ dans la version **tree-search** est optimal si la fonction heuristique $h$ est admissible

**Proof**

Supposons que l'algorithme atteigne le but $n$ et le chemin retourné est noté $l$. Le coût du chemin $l$ est donc $g(n)$.
Raisonnons par l'absurde : on fait l'hypothèse que le chemin trouvé $l$ n'est pas optimal. Il existe donc un chemin $l'$ dont le coût est $c < g(n)$.
Donc, comme $h$ sous-estime, pour chaque noeud $n'$ de $l'$, on a $g(n') + h(n') \leqslant c < g(n)$. Mais de tels noeuds auraient déjà dûs être développés, on arrive à une contradiction. □

# Condition d'Optimalité (graph-search)

**Definition** (monotonicité)

une heuristique est monotone si pour tout noeud $n$, tout succésseur $n'$ de $n$ on a $h(n) \leqslant c(n,n') + h(n')$.

**Theorem**

$A^\star$ dans la version **graph-search** est optimal si la fonction heuristique $h$ est monotone.

Une heuristique monotone est admissible (cf TD). Généralement, les heuristiques admissibles mais non admissibles sont assez dures à trouver.

## Proof

## Proof

Supposons que $h$ est monotone. Soit un chemin dans le graphe de recherche et $n$ et $n'$ deux noeuds succéssifs. Alors
$f(n) = g(n) + h(n) \leqslant \underbrace{g(n) + c(n,n')}_{g(n')} + h(n')$ (monotonicité de $h$)
$f(n) \leqslant g(n') + h(n') = f(n')$. On en conclue donc que $f$ est croissante le long de tout chemin.

Démontrons maintenant qu'au moment où $A^\star$ choisit de développer le noeud $n$, le chemin optimal menant à $n$ a déjà été trouvé.
Supposons que ce ne soit pas le cas : il doit exister un noeud $n'$ sur le chemin optimal menant à $n$ qui ne soit pas developpé. Comme $f$ est croissante, $f(n') \leqslant f(n)$, et donc $n'$ aurait dû être développé, on arrive à une contradiction.

Ainsi, la séquence de noeuds développés par $A^\star$ est en ordre croissant de la fonction $f$. Donc la première fois qu'on veut développer un noeud but $n$, on a trouvé une solution optimale, car $f(n) = g(n)$ et chaque noeud développé plus tard auront une valeur au moins plus grande.

□