Intelligence artificielle
La résolution automatique de problèmes

Hatem Ghorbel & Stefano Carrino

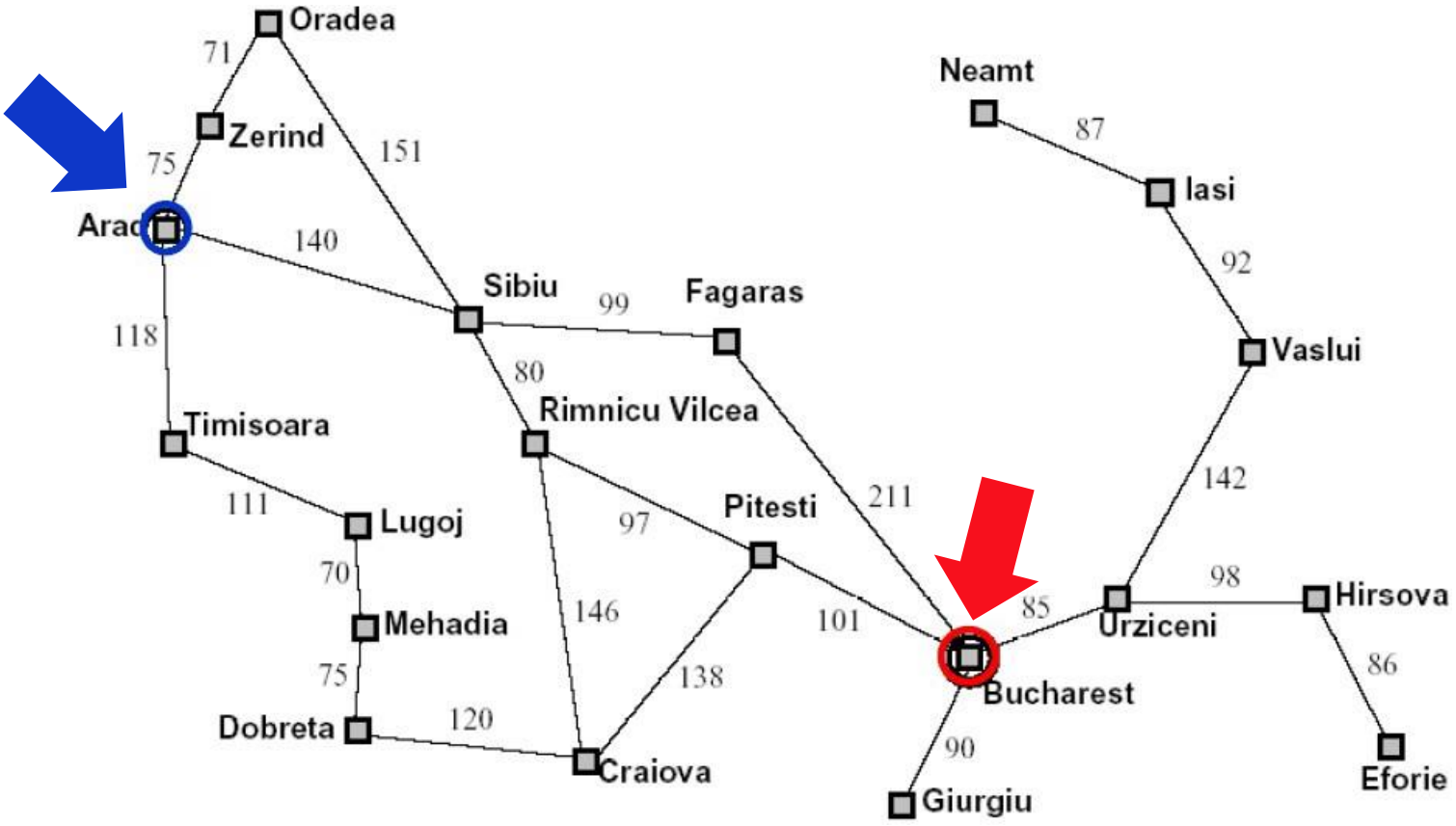
Ref. *Artificial Intelligence: A Modern Approach*
by Stuart Russell (Author), Peter Norvig

Problème de la recherche aveugle

- Ne garde pas les états du chemin solution au cours de la recherche
- Complexité en espace
  - Le nombre de nœuds à mémoriser croît de façon exponentielle (dans le cas de largeur d'abord)
  - Envisager l'élagation de l'espace de recherche par les ***heuristiques*** (recherches informées)

Planifier un voyage

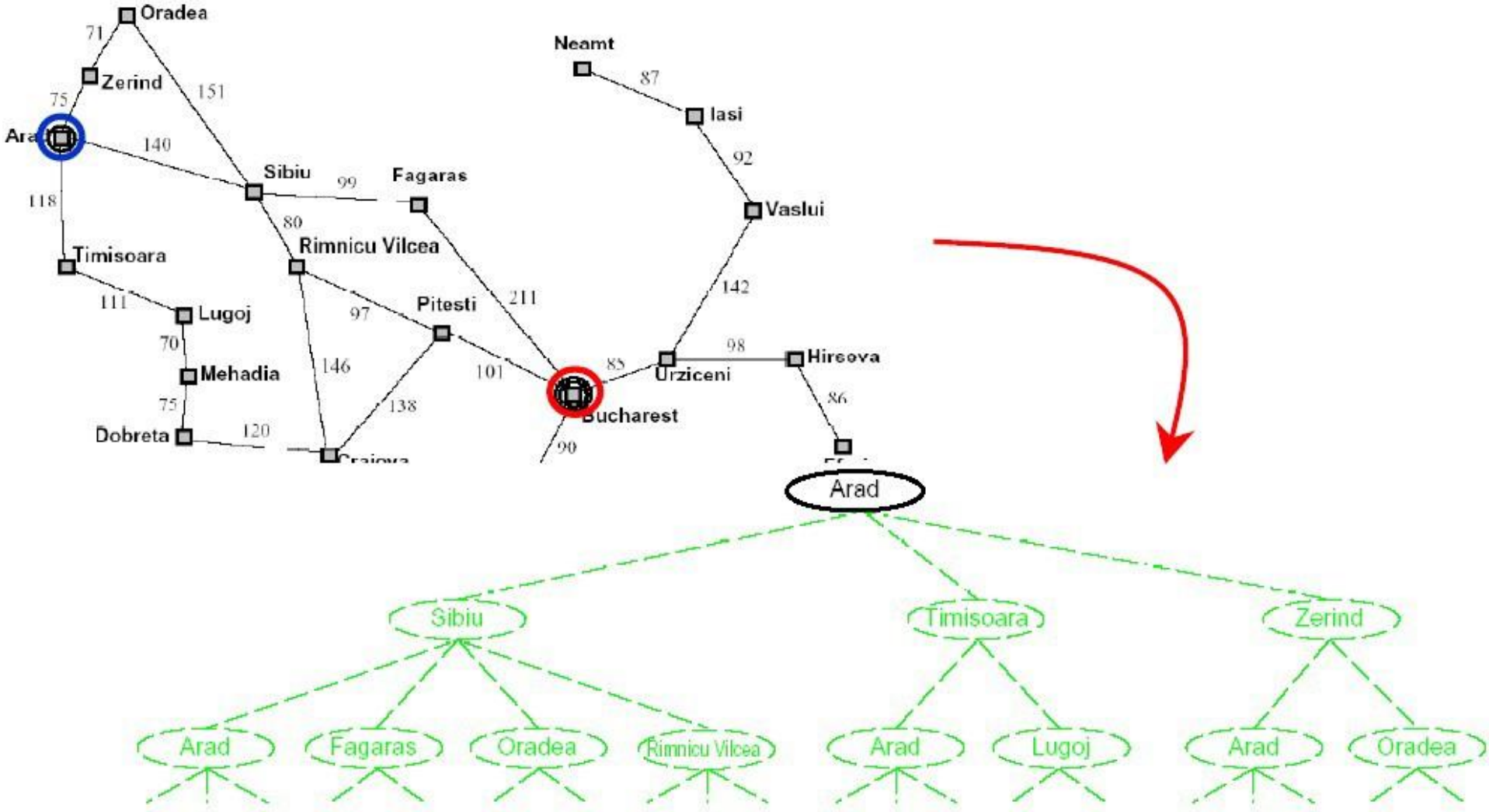
- Pour aller d'*Arad* à *Bucarest*, utilisons les multiples stratégies de recherche aveugle présentées jusqu'ici.



Ceci est un graphe de l'espace du problème.

Nous verrons dans quelques instants comment la même information peut être représentée sous forme d'un arbre.

Graphe de l'espace du problème vs. Arbre



Source: Alain Boucher, Institut de la Francophonie pour l'Informatique
http://www.ifi.auf.org/personnel/Alain.Boucher/cours/intelligence_artificielle/05-Recherche.pdf.

Rappel : qu'est-ce qu'une heuristique?

- *Fonction qui établit des* pistes *pour orienter vers la solution, et élaguer les candidats peu intéressants*

Méthodes de recherche heuristique

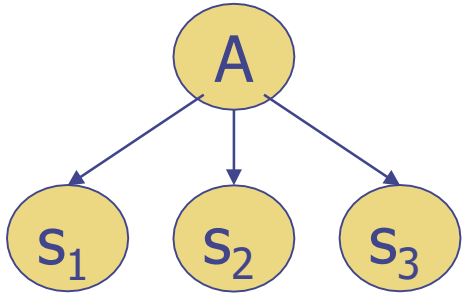
- Les algorithmes de recherche aveugle n'exploitent aucune information concernant la structure de l'arbre de recherche ou la présence potentielle de nœuds-solution pour optimiser la recherche.
- Recherche "rustique" à travers l'espace jusqu'à trouver une solution.
- La plupart des problèmes réels sont susceptibles de provoquer une explosion combinatoire du nombre d'états possibles.
- Un algorithme de recherche heuristique utilise l'information disponible pour rendre le processus de recherche plus efficace.
- Une information heuristique est une règle ou une méthode qui presque toujours améliore le processus de recherche.

Fonction heuristique

- Une fonction heuristique

$$h: E \rightarrow \mathbb{R}$$

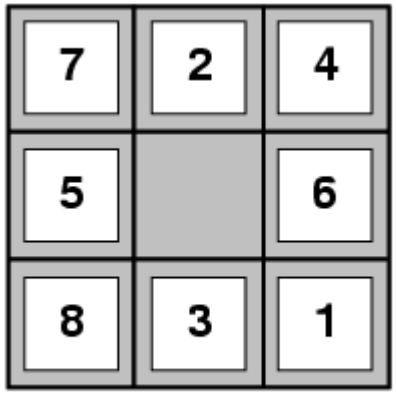
- fait correspondre à un état ***s de E*** (espace d'états) un nombre ***h(s) de*** $\mathbb{R}$ qui est (généralement) une estimation du rapport coût/bénéfice qu'il y a à étendre le chemin courant en passant par ***s***.
- Contrainte: *h(solution) = 0*
- Exemple:
  - Le nœud A a 3 successeurs pour lesquels:
    $h(s_1) = 0.8 \quad h(s_2) = 2.0 \quad h(s_3) = 1.6$
  - La poursuite de la recherche par *s1* est heuristiquement la meilleure



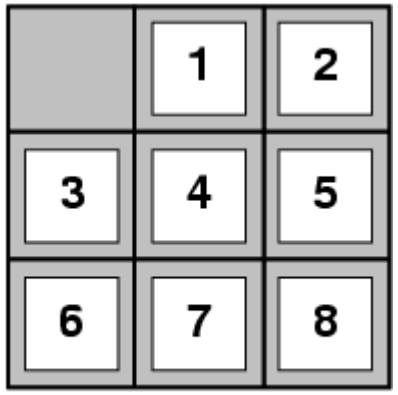
Exemple d'heuristique : Distance de Manhattan

- Taquin à 8 plaquettes
- Suggestion de deux heuristiques possibles :
  - *h1(n)* = nombre de plaquettes mal placées*
  - *h2(n)* = distance de Manhattan
    - déplacements limités aux directions verticales et horizontales
    - somme des distances de chaque plaquette à sa position finale

- *h1(n) = 8*
- *h2(n) = 3+1+2+2+...= 18*



Start State



Goal State

* Note: espace vide != plaquette

Utilisation d'une heuristique dans la stratégie du Meilleur-d'abord

- L'algorithme "*Best-First search*" permet d'explorer les nœuds dans l'ordre (meilleur-que) **de leurs valeurs heuristiques**
- Fonctions heuristiques classiques pour un problème TSP:
  - distance "Manhattan" (déplacements limités aux directions verticales et horizontales)
  - distance à vol d'oiseau

- On utilise 2 listes pour garder l'historique et permettre les retours-arrières:
  - Frontière = liste ordonnée des nœuds à examiner
  - Historique = liste des nœuds déjà examinés

Utilisation d'une heuristique dans la stratégie du Meilleur-d'abord (suite)

```
Frontière ← [start]; historique ← [];
Tant que Frontière n'est pas vide
   noeud ← enleverPremier (Frontière)
   Si nœud = but, retourner le chemin solution
   Sinon
      Générer les enfants de nœud
      Pour chaque enfant calculer sa valeur heuristique (h(enfant))
         cas:
         - l'enfant n'est ni dans Frontière ni dans Historique: Ajouter
               dans Frontière
         Fin Cas
     Fin Pour
   FinSi
   Mettre nœud dans Historique; Réordonner Frontière;
FinTantQue
Afficher ("Échec")
```

Utilisation d'une heuristique dans la stratégie du Meilleur-d'abord (suite)

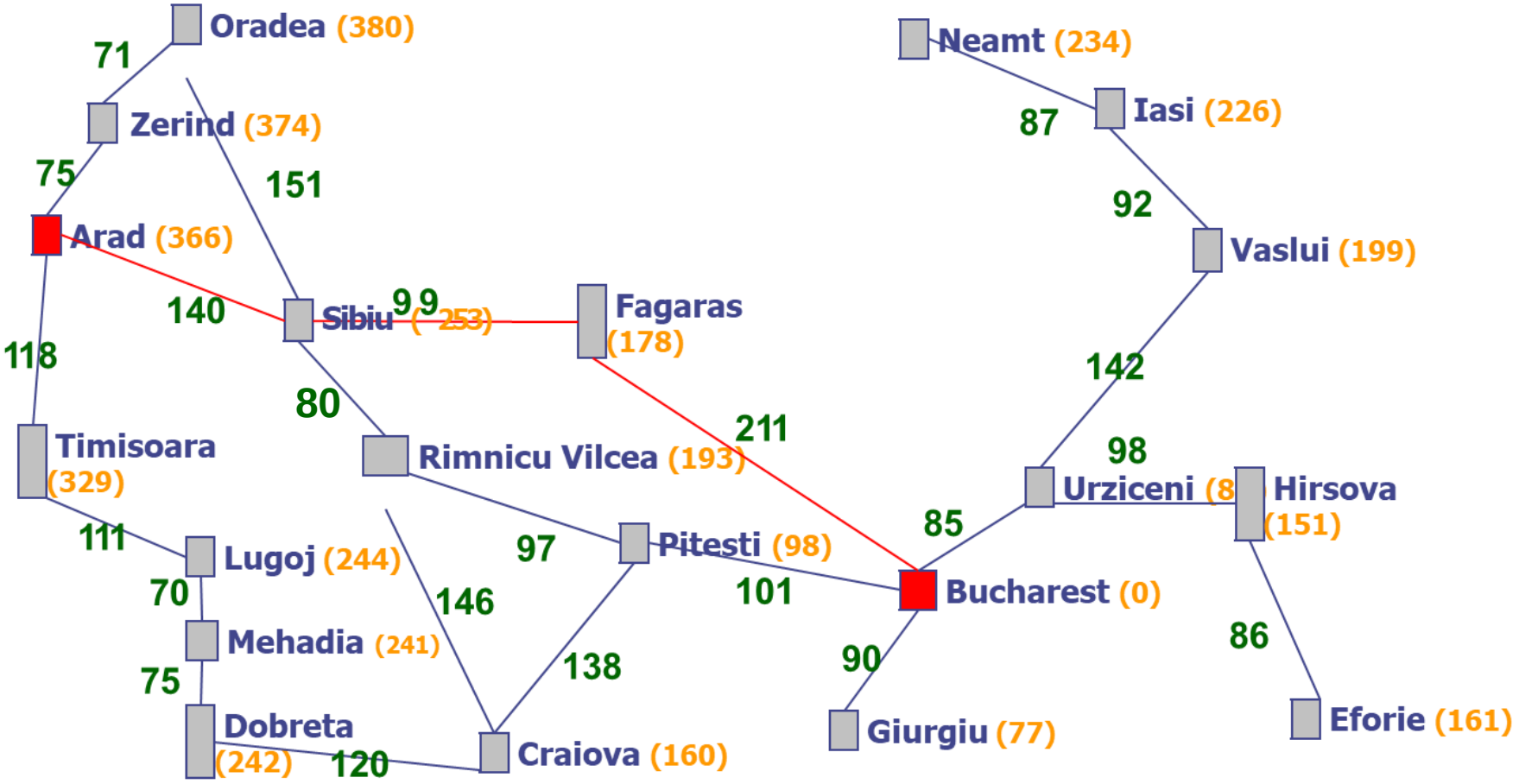
```
Frontière ← [start]; historique ← [];
Tant que Frontière n'est pas vide
   noeud ← enleverPremier (Frontière)
   Si nœud = but, retourner le chemin solution
   Sinon
      Générer les enfants de nœud
      Pour chaque enfant calculer sa valeur heuristique (h(enfant))
         cas:
         - l'enfant n'est ni dans Frontière ni dans Historique: Ajouter
               dans Frontière
         Fin Cas
     Fin Pour
   FinSi
   Mettre nœud dans Historique; Réordonner Frontière;
FinTantQue
Afficher ("Échec")
```

Recherche Meilleur-d'abord *avare*

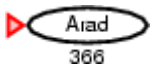
- Stratégie la plus simple des "best-first search"
- Fonction heuristique *h(n)* = estimation du coût du nœud n jusqu'au **but**
- **Recherche avare** = minimiser le coût estimé pour atteindre le but
  - le nœud qui *semble* être le plus proche du but sera étendu en priorité
  - en considérant uniquement l'état actuel, pas l'historique des coûts accumulés

Exemple : Voyage en Roumanie

- État initial : Dans(Arad)
- But : Dans(Bucarest)
- Voyage: `h(n) = distance_en_ligne_droite(n, but)`



Voyage en Roumanie
expansion des nœuds par recherche avare

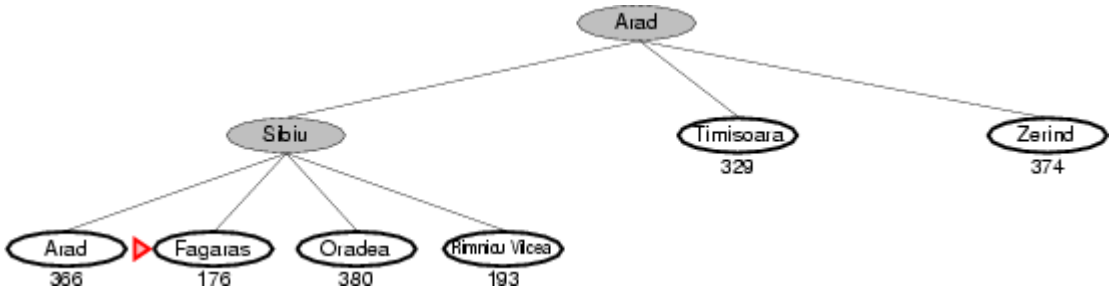


h = dist. à vol d'oiseau	
Ville (v)	**h(v)**
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Voyage en Roumanie
expansion des nœuds par recherche avare

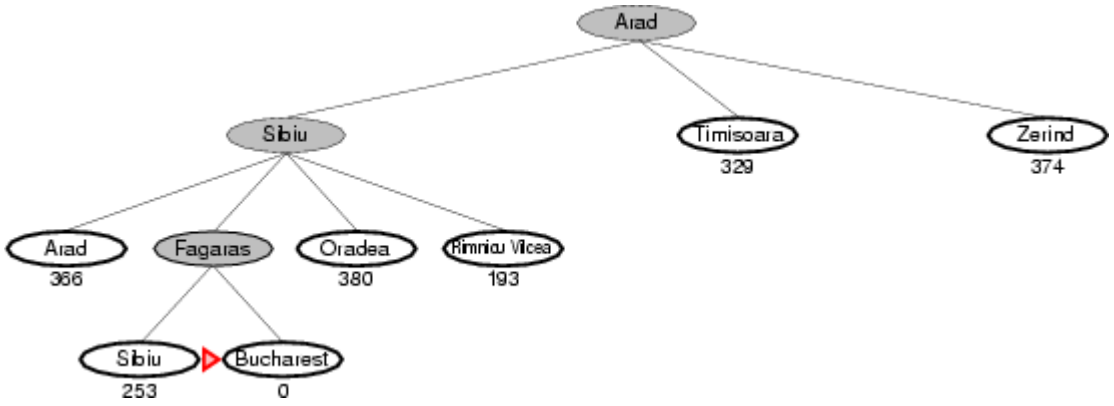
h = dist. à vol d'oiseau	
Ville (v)	**h(v)**
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Voyage en Roumanie
expansion des nœuds par recherche avare



h = dist. à vol d'oiseau	
Ville (v)	**h(v)**
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	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Voyage en Roumanie
expansion des nœuds par recherche avare



h = dist. à vol d'oiseau	
Ville (v)	**h(v)**
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	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Propriétés du meilleur-d'abord avare

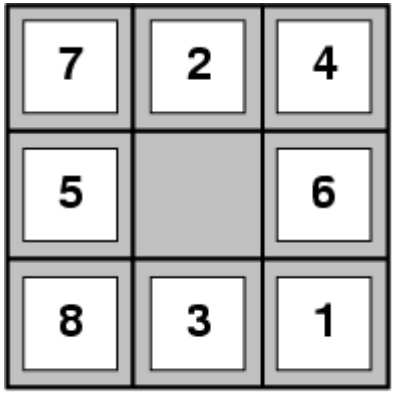
- Complète? Non (si boucle), Ex: Iasi → Neamt → Iasi → Neamt →
- Temps? $O(b^m)$ => (*exponentiel en m = profondeur maximum de l'espace de recherche*), mais une heuristique plus informée peut drastiquement améliorer le temps
- Espace? $O(b^m)$ garde tous les nœuds en mémoire

- Optimale?
  - Non (on ne tient pas compte de l'historique du nœud)
  - Arad > Sibiu > Fagaras > Bucarest (140+99+211=450 km) n'est pas optimale; elle est de 32 km plus longue que
  - Arad > Sibiu > Rimnicu > Pitesti > Bucarest (140+80+97+101=418km)

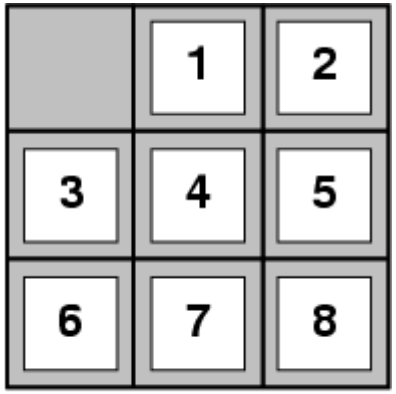
Fonction heuristique *admissible*

- Théorème :
  Une fonction heuristique est admissible si
  $\forall n, 0 \leq h(n) \leq h^*(n)$ *avec* $h^*(n)$ = *coût optimal réel de n au but*
- Autrement dit : ne surestime jamais le coût réel
- Une fonction heuristique *admissible* est donc toujours optimiste !

Exemple d'heuristique : *Taquin à 8 plaquettes*



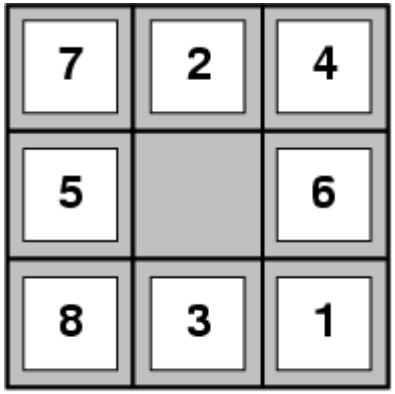
Start State



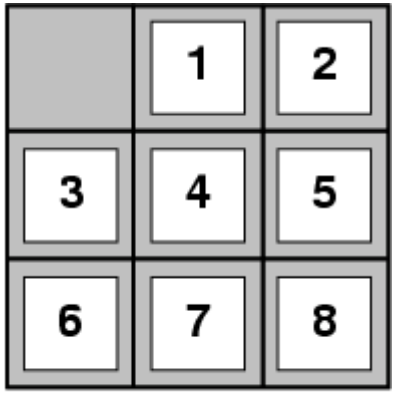
Goal State

- h1(n) = nombre de plaquettes mal placées = **admissible ?**
- h2(n) = somme des distances de chaque plaquette au but = **admissible ?**

Exemple d'heuristique : *Taquin à 8 plaquettes*



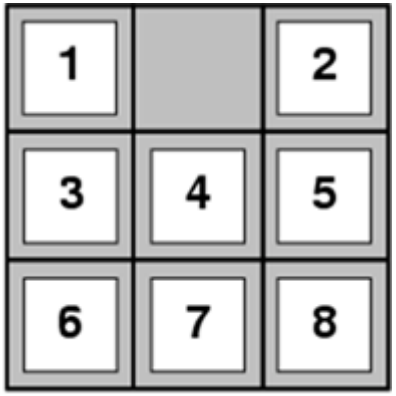
Start State



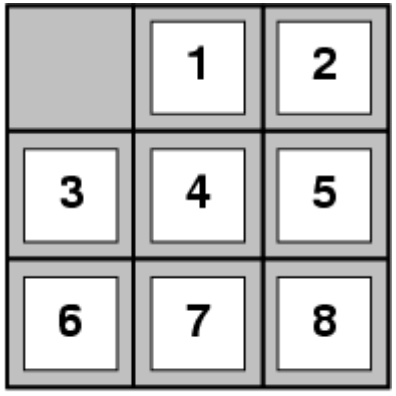
Goal State

- h1(n) = nombre de plaquettes mal placées = **admissible**
- h2(n) = somme des distances de chaque plaquette au but = **admissible**
- h3(n) = nombre de plaquettes (y.c.) case vide mal placées = **admissible ?**
- h4(n) = somme des distances de chaque plaquette au but (y.c.) case vide = **admissible ?**

Exemple d'heuristique : *Taquin à 8 plaquettes*



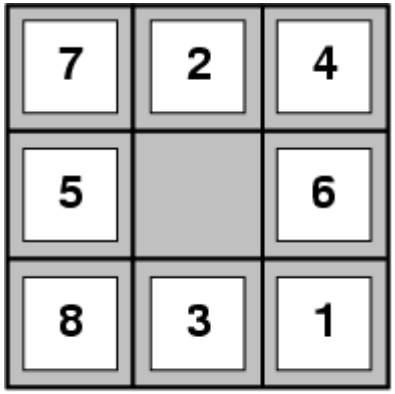
Start State



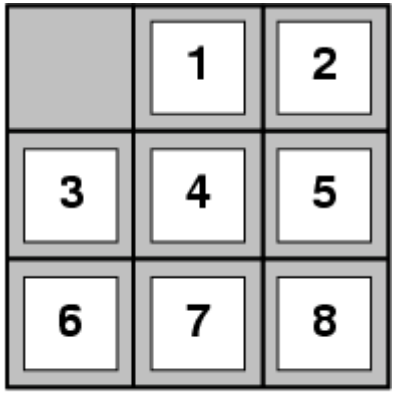
Goal State

- h1(n) = nombre de plaquettes mal placées = **admissible**
- h2(n) = somme des distances de chaque plaquette au but = **admissible**
- h3(n) = nombre de plaquettes (y.c.) case vide mal placées = **admissible ?**
- h4(n) = somme des distances de chaque plaquette au but (y.c.) case vide = **admissible ?**

Exemple d'heuristique : *Taquin à 8 plaquettes*



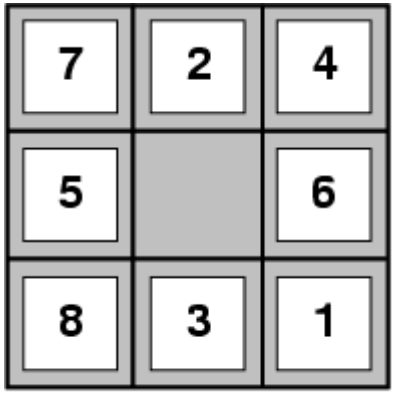
Start State



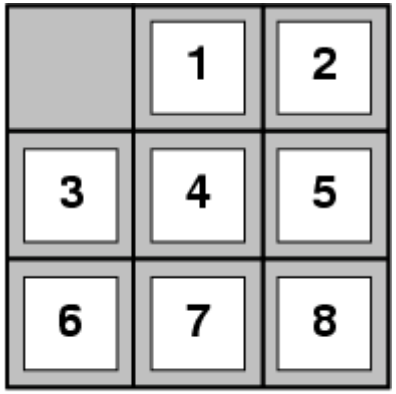
Goal State

- h1(n) = nombre de plaquettes mal placées = 8 **est admissible**
- h2(n) = somme des distances de chaque plaquette au but = 18 **est admissible**
- h5(n) = (somme des distances de chaque plaquette au but) + 3 x (somme de la fonction score de chaque plaquette) = **admissible?**
  - *fonction score = 2 pour une plaquette non centrale si elle n'est pas suivie par la bonne plaquette (pas la bonne séquence) et 0 si non*

Exemple d'heuristique : *Taquin à 8 plaquettes*



Start State



Goal State

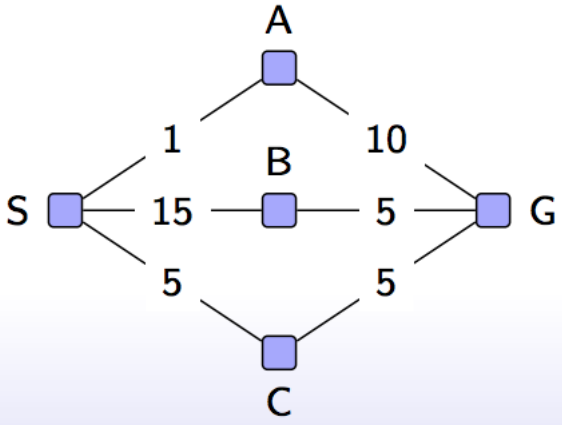
- h1(n) = nombre de plaquettes mal placées = 8 **est admissible**
- h2(n) = somme des distances de chaque plaquette au but = 18 **est admissible**
- h3(n) = (somme des distances de chaque plaquette au but) + 3 x (somme de la fonction score de chaque plaquette) = 3+1+3+0+2+1+0+3 + 3 x (2 + 2 + 2 + 2 + 2 + 2 + 2 + 2) = 66 **n'est pas admissible**
  - *fonction score = 2 pour une plaquette non centrale si elle n'est pas suivie par la bonne plaquette (pas la bonne séquence) et 0 si non*

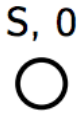
- *Note*: La complexité de la fonction heuristique peut nuire à l'efficacité de la recherche

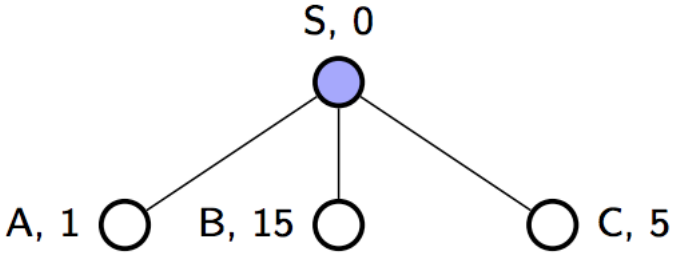
  => Il faut trouver un bon compromis

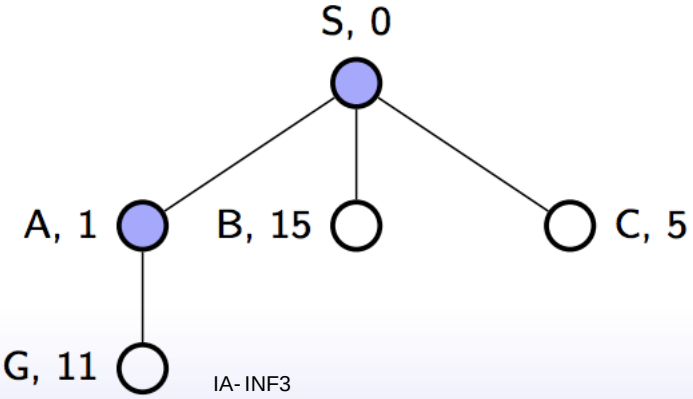
Regarder le passé :
Recherche en coût uniforme

- L'algorithme de *recherche en coût uniforme* minimise le coût *g(n)* depuis l'état initial au nœud *n*



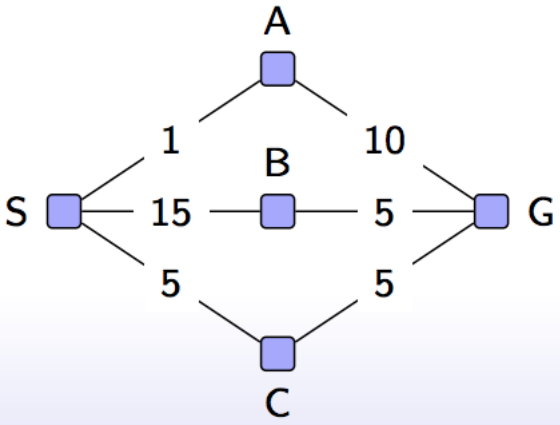


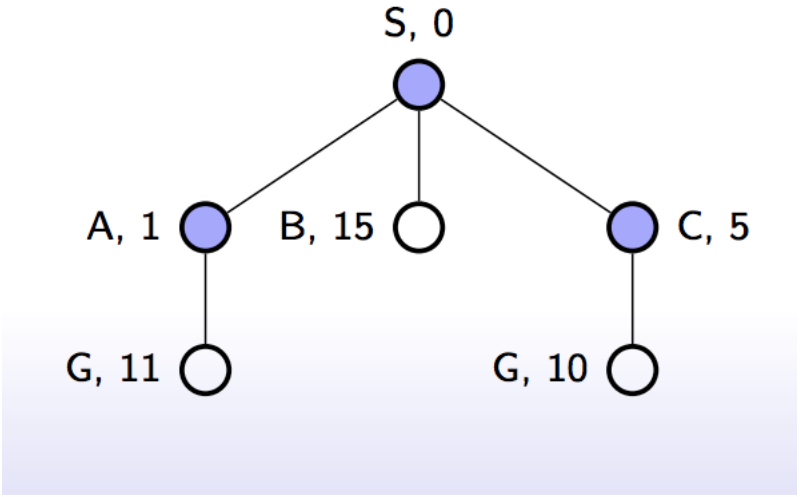


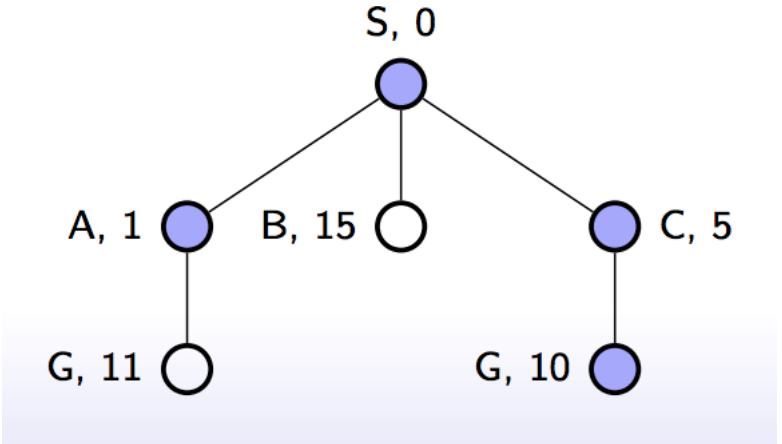


Regarder le passé :
Recherche en coût uniforme

- L'algorithme de *recherche en coût uniforme* minimise le coût *g(n)* depuis l'état initial au nœud *n*

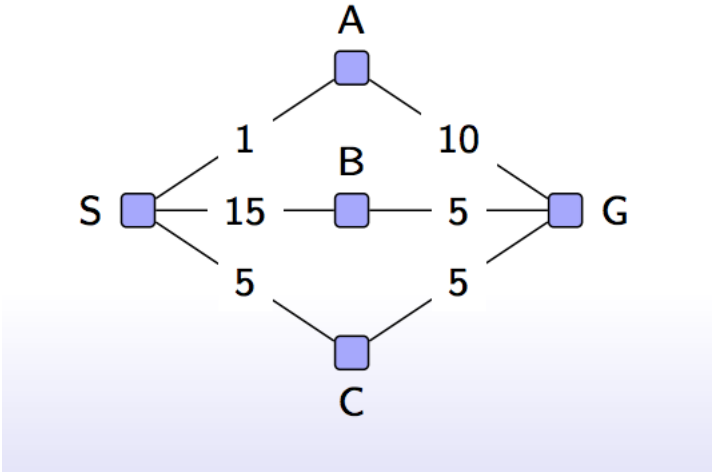


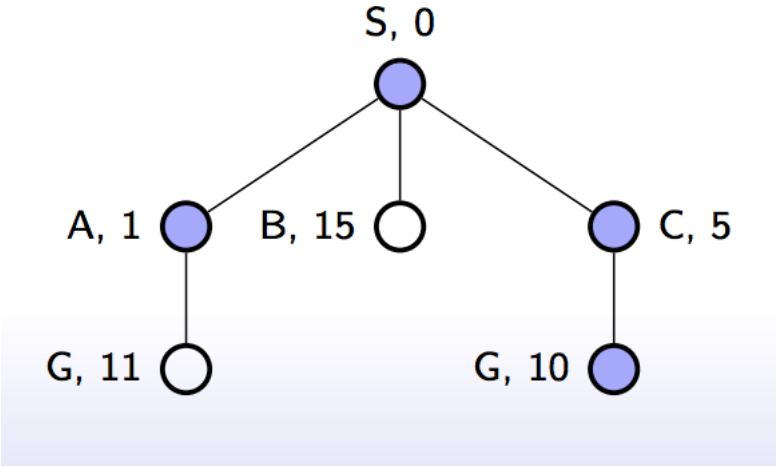




Regarder le passé :
Recherche en coût uniforme

- L'algorithme de *recherche en coût uniforme* minimise le coût *g(n)* depuis l'état initial au nœud *n*
  - il est optimal
  - il est complet
  - mais pas très efficace…





Idée

- La *recherche avare* minimise le coût estimé *h(n)* du nœud *n* au but, réduisant ainsi considérablement le coût de la recherche, mais il n'est pas optimal et pas complet (en général)
- L'algorithme de *recherche en coût uniforme* minimise le coût *g(n)* depuis l'état initial au nœud *n*

- Idée:
  - Combiner les deux algorithmes et minimiser le coût total *f(n)* du chemin passant par le nœud *n*

$$f(n) = g(n) + h(n)$$

Algorithme A*

- Soit la fonction d'évaluation

$$f(n) = g(n) + h(n)$$

- Si en plus *f(n)* est telle que:
  - *g(n)* = coût du meilleur chemin jusqu'à *n*
  - *h(n)* = une fonction heuristique **admissible**
- L'algorithme "recherche avare" avec la fonction *f(n)* est appelé: algorithme **A***

Exemple A* avec Taquin-8

- Taquin-8: $f(n) = g(n) + h(n)$

  - avec g(n) = nombre de steps (dans le jeu de taquin chaque mouvement a le même coût)
  - avec $h(n)$ = nombre de plaquettes mal placées

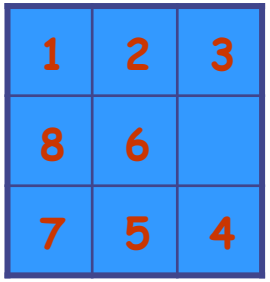
État initial

1	2	3
8	6	
7	5	4

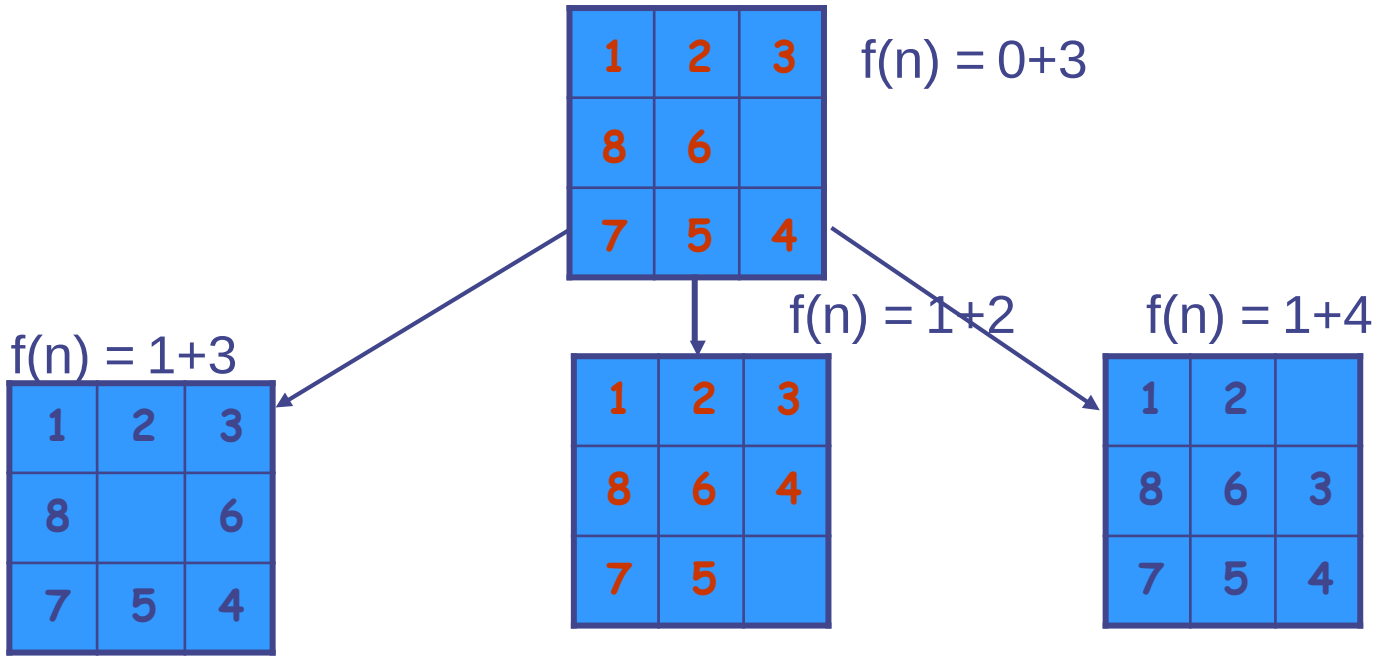
f(n) = ?

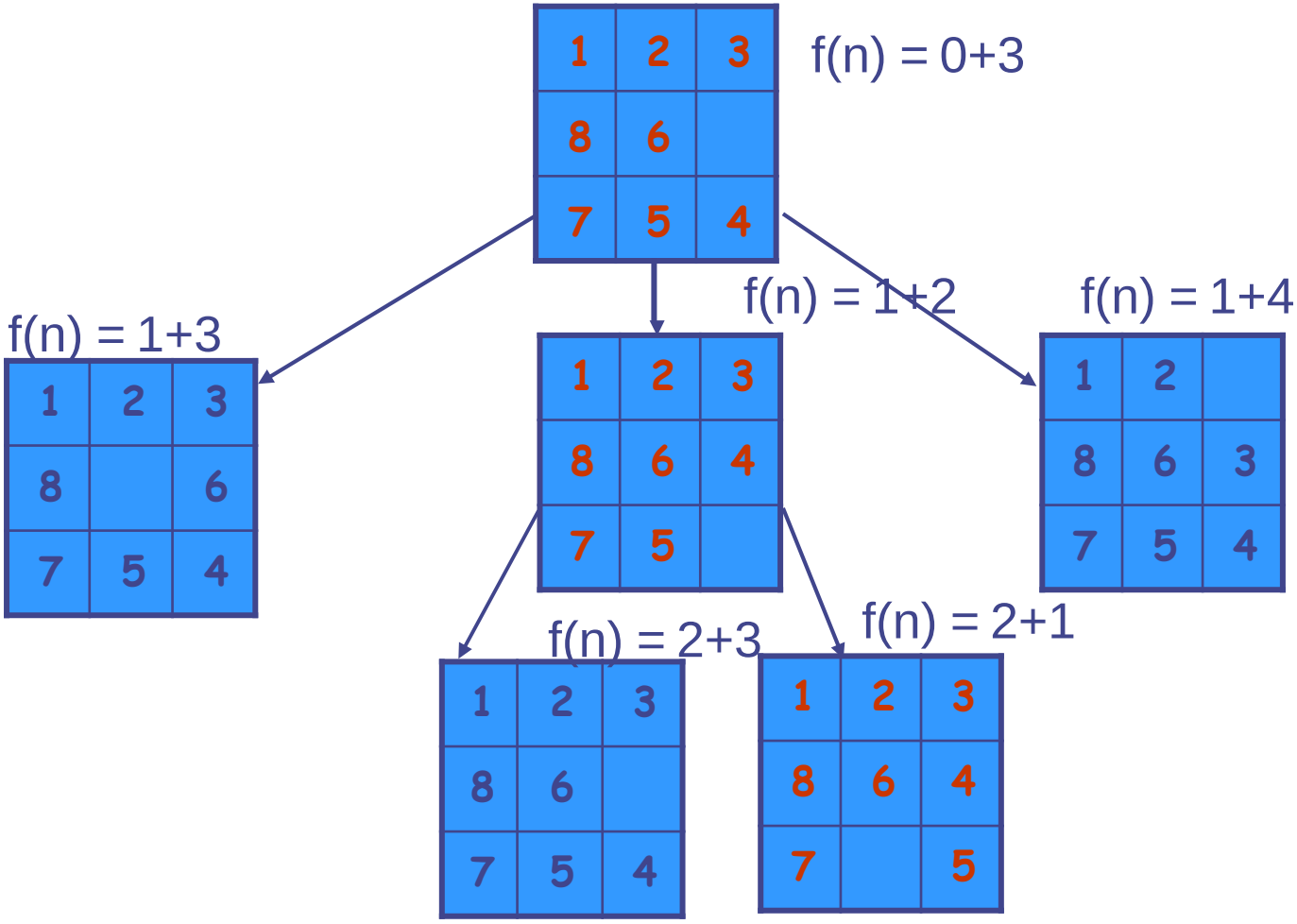
État but

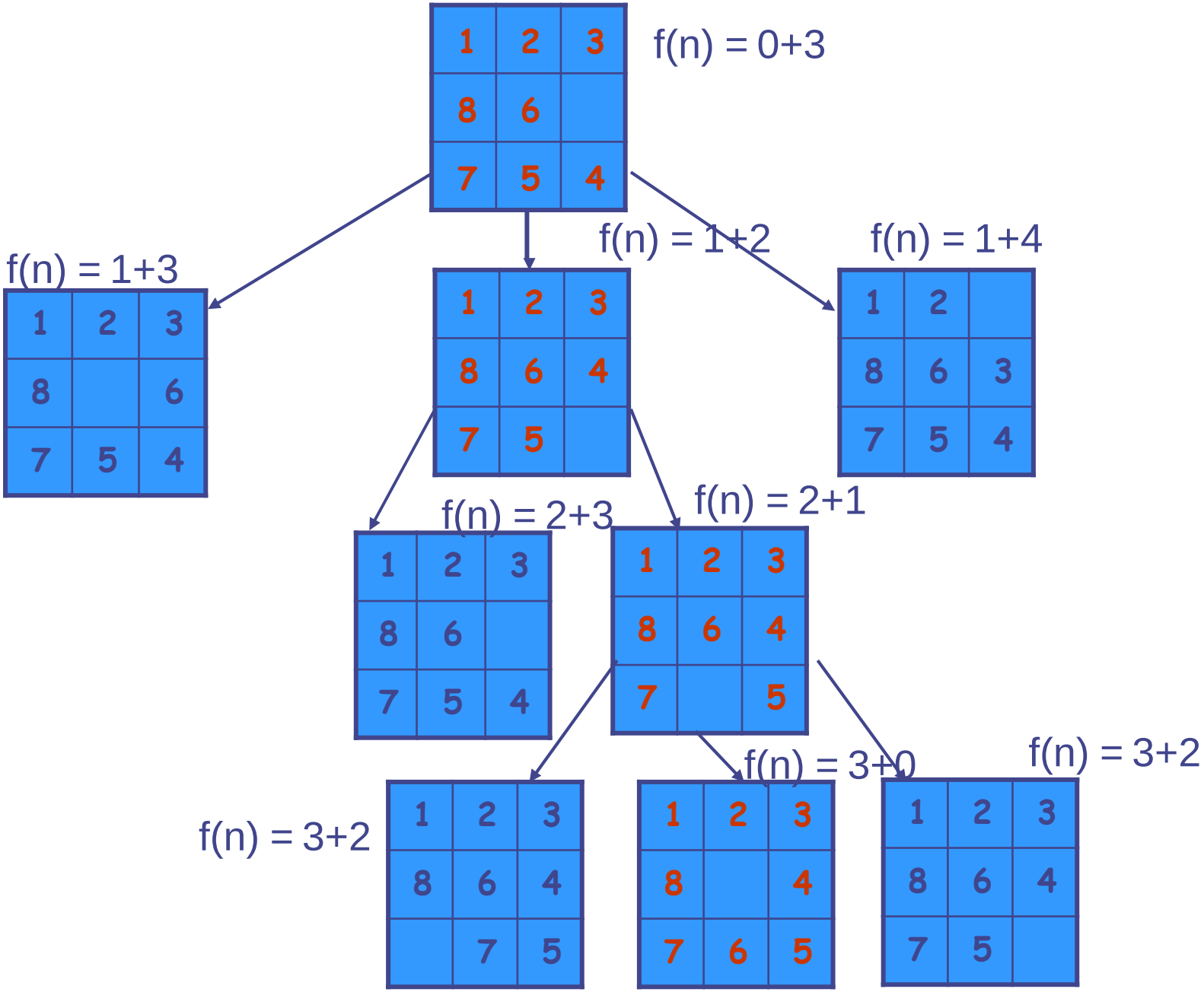
1	2	3
8		4
7	6	5



f(n) = 0+3


1 2 3
8 6
7 5 4
f(n) = 0+3
f(n) = 1+3
1 2 3
8 6
7 5 4
f(n) = 1+2
1 2 3
8 6 4
7 5
f(n) = 1+4
1 2
8 6 3
7 5 4


1 2 3
8 6
7 5 4
f(n) = 0+3
f(n) = 1+3
1 2 3
8 6
7 5 4
f(n) = 1+2
1 2 3
8 6 4
7 5
f(n) = 1+4
1 2
8 6 3
7 5 4
f(n) = 2+3
1 2 3
8 6
7 5 4
f(n) = 2+1
1 2 3
8 6 4
7 5

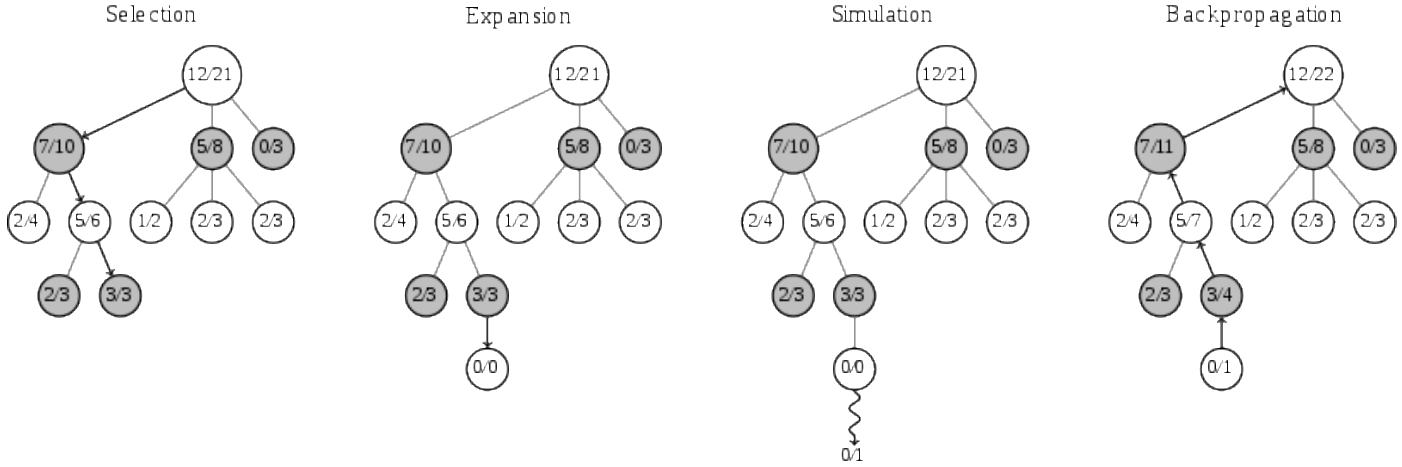

f(n) = 0+3
f(n) = 1+3
f(n) = 1+2
f(n) = 1+4
f(n) = 2+3
f(n) = 2+1
f(n) = 3+2
f(n) = 3+0
f(n) = 3+2

Propriétés de A*

- Complète ?
  - Oui – sauf s'il y a un nombre infini de nœuds avec une valeur de $f \leq f(But)$
- Temps ?
  - *Exponentiel*
- Espace ?
  - Garde tous les nœuds en mémoire
- Optimale ?
  - Oui, si *h()* est admissible (peut être facilement prouvé que A* ne trouve que le chemin le plus court)

Autres algorithmes de recherche heuristique

- Un autre algorithme très utilisé est la Recherche arborescente Monte-Carlo (*Monte Carlo tree search*)
- Il est notamment employé dans les jeux.
  - Ex. *Total War: Rome II* avec son mode campagne IA haut-niveau
  - Les récents programmes informatiques de Go, échecs et shogi, ainsi que les jeux vidéos en temps réel et les jeux à information incomplète tels que le poker.



Source et plus d'infos: https://en.wikipedia.org/wiki/Monte_Carlo_tree_search

Méta-heuristiques ?

- Une des difficultés de la recherche heuristique est de trouver une heuristique appropriée
- L'heuristique dépend beaucoup du domaine d'application et son développement demande souvent des compétences avancées
- On peut donc s'intéresser à des méthodes heuristiques globales
  - proposant une méthode de recherche générique
  - dépendant aussi peu que possible du domaine d'application
- On parle alors de *méta-heuristique*
- Les *algorithmes génétiques* font partie de cette famille d'algorithmes.

Algorithme A*

```
Frontière ← [start]; historique ← [];
Tant que Frontière n'est pas vide
   noeud ← enleverPremier (Frontière)
   Si nœud = but, retourner le chemin solution

   Générer les enfants de nœud
   Pour chaque enfant:

       calculer f, g et h (enfant)
       cas:
         - l'enfant n'est pas dans Frontière:
               Ajouter dans Frontière
         - l'enfant est dans Frontière ou Historique:
               Si g(enfant) meilleur que sa valeur dans Frontière ou
               Historique
                     (rem)placer le nœud dans Frontière
         Fin Cas
      Fin Pour
   Mettre nœud dans Historique; Réordonner Frontière selon f;
FinTantQue
Afficher ("Échec")
```