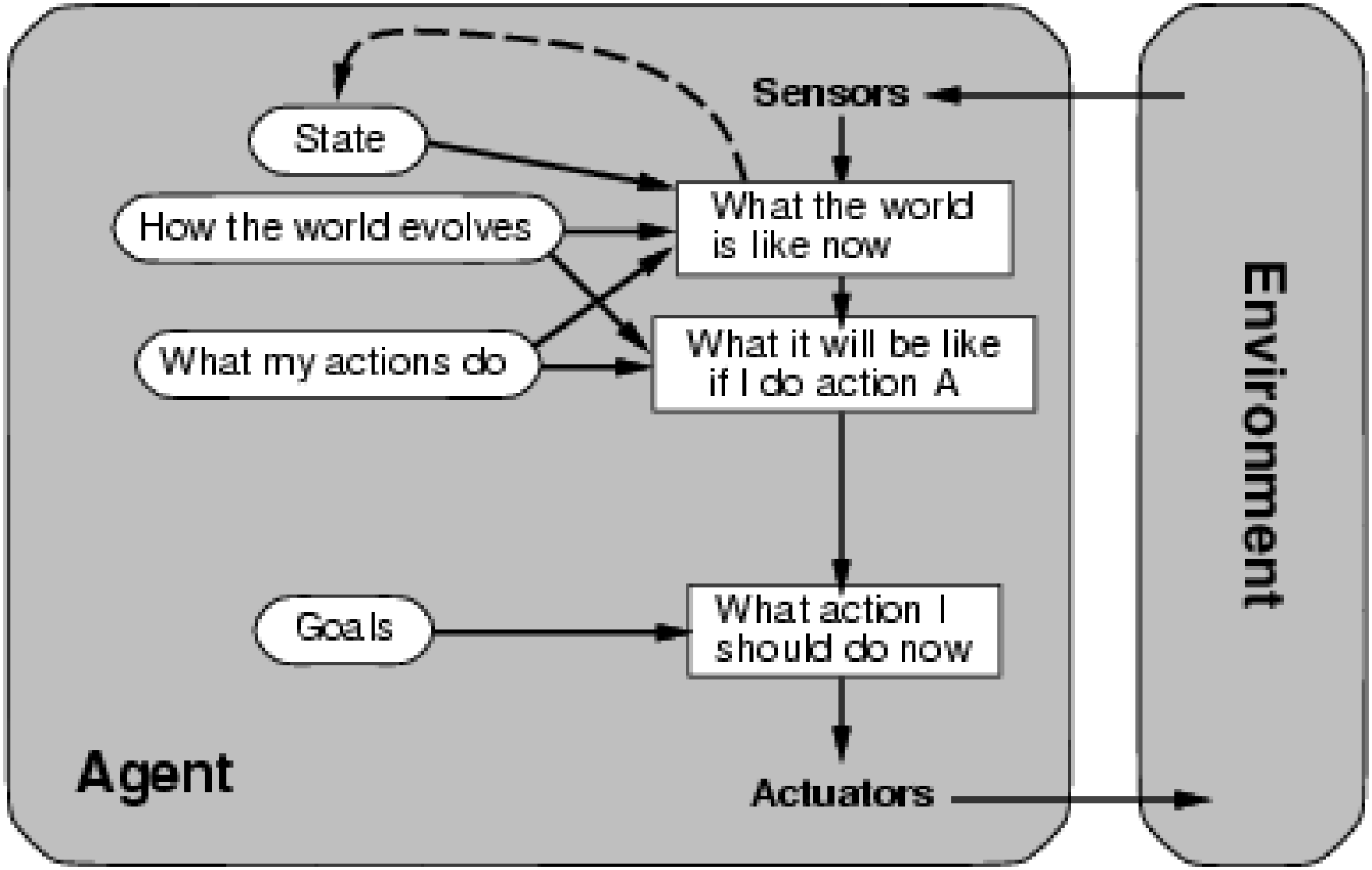
Les fondements IA: Résolution automatique de problèmes

- ✓ Résolution de problèmes par la recherche
- ✓ Recherche heuristique et planification
- ✓ Recherche en situation d'adversité et jeux

Comment les humains prennent-ils des décisions ?

1. Observer la situation actuelle.
2. Énumérer les options possibles.
3. Évaluer les conséquences des options (simulation).
4. Retenir la meilleure option possible satisfaisant le but.

Rappel : Agent basé sur des buts

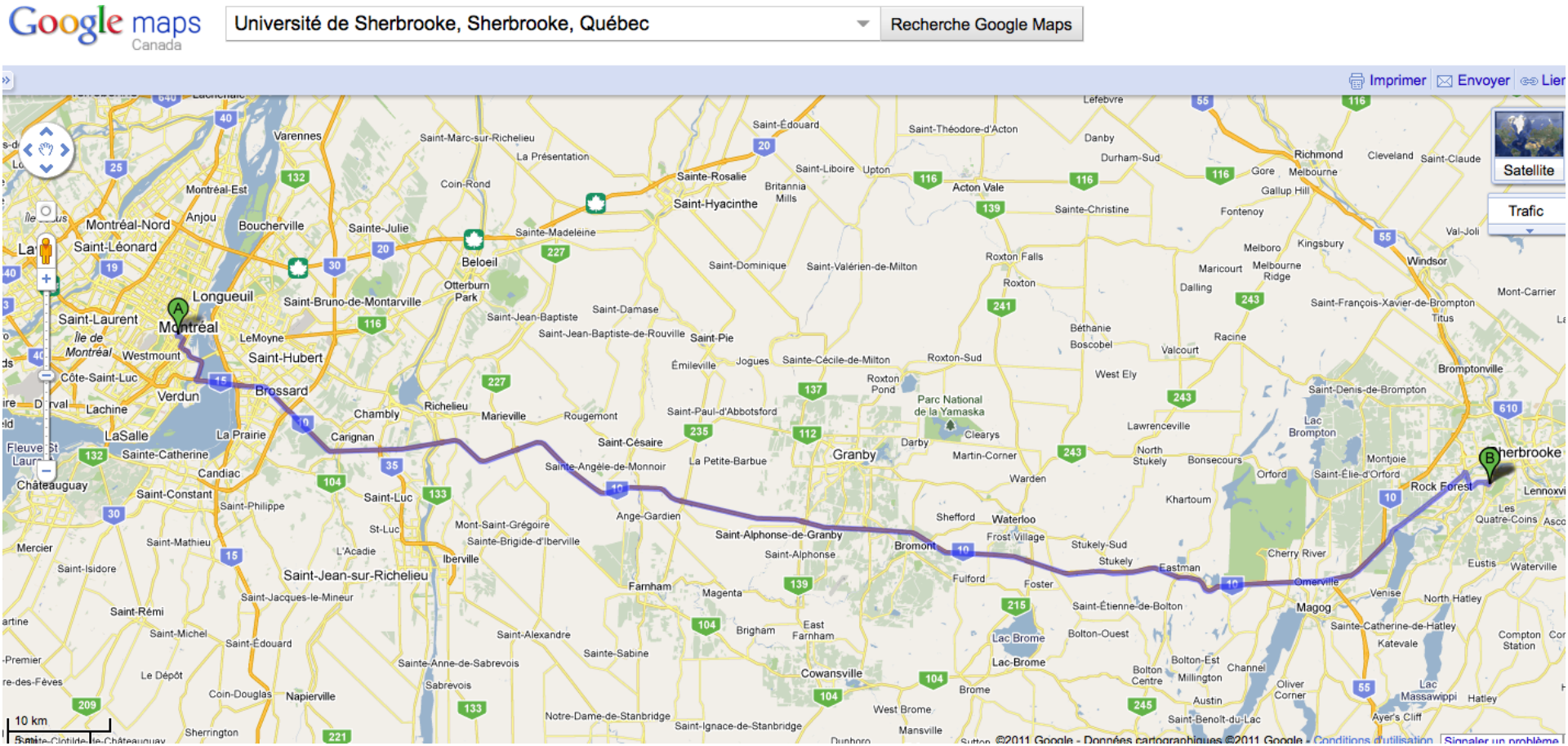


Agent basé buts / Boucle de contrôle

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
    static: seq, an action sequence, initially empty
            state, some description of the current world state
            goal, a goal, initially null
            problem, a problem formulation

    state ← UPDATE-STATE(state, percept)
    if seq is empty then do
        goal ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, goal)
        seq ← SEARCH(problem)
    action ← FIRST(seq)
    seq ← REST(seq)
    return action
```

Application – Recherche dans une carte

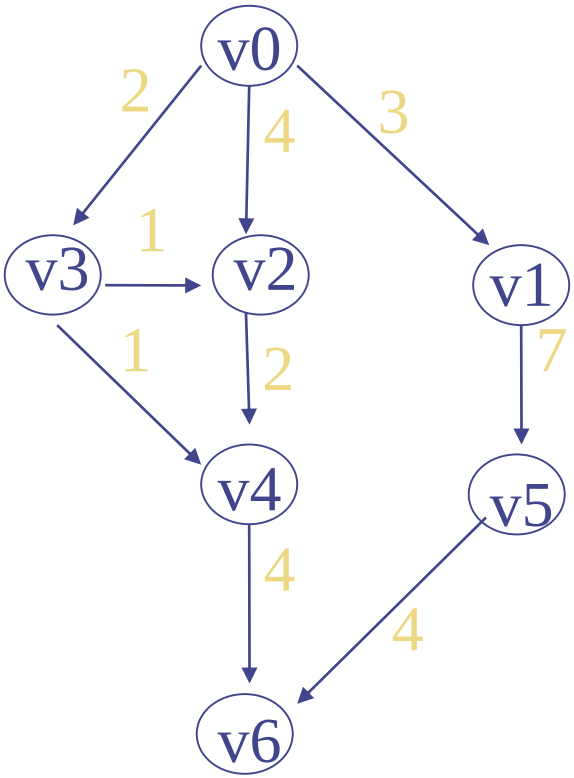


Application – Recherche dans une carte

Domaine :

Routes entre les villes

transitions(v0):
((2,v3), (4,v2), (3, v1))



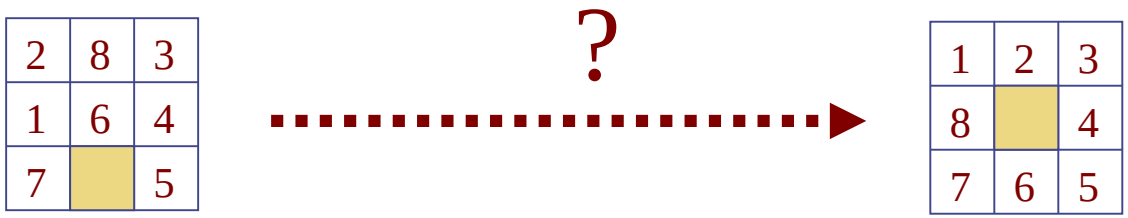
Problème posé (initNode, goal):

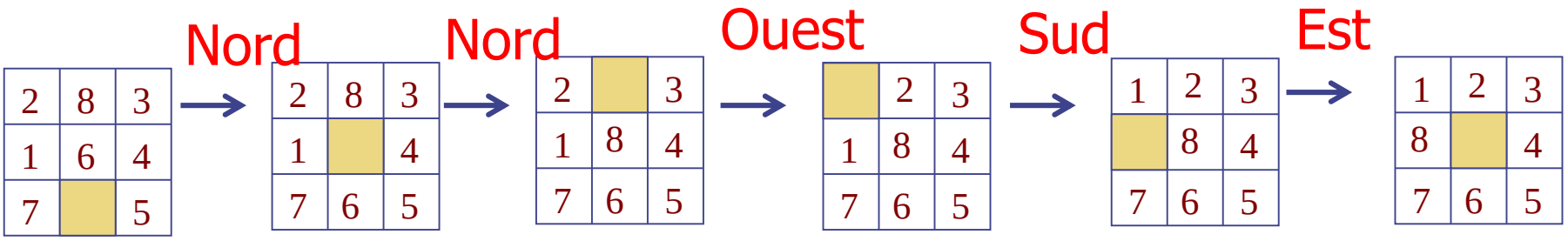
v0: ville de départ (état initial)
v6: destination (but)

En d'autres termes:

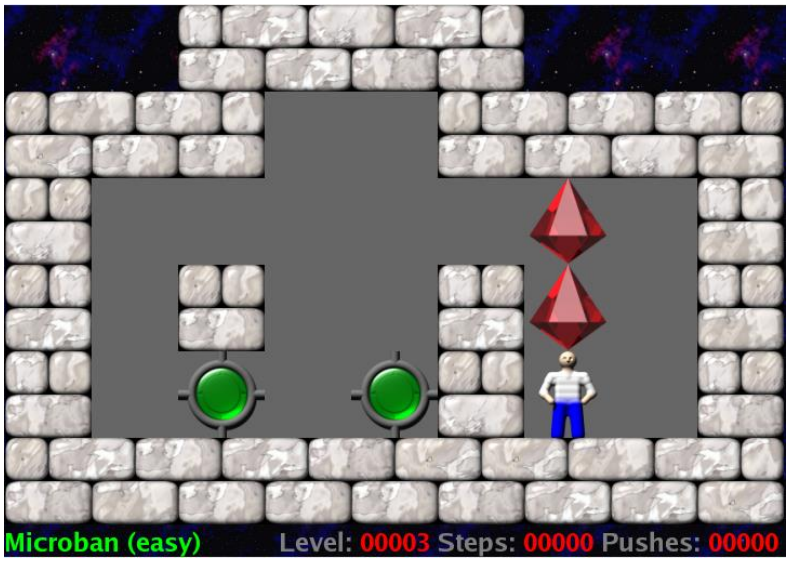
goal(v): vrai si v=v6

Application – Jeu de taquin





Application – Jeu Sokoban

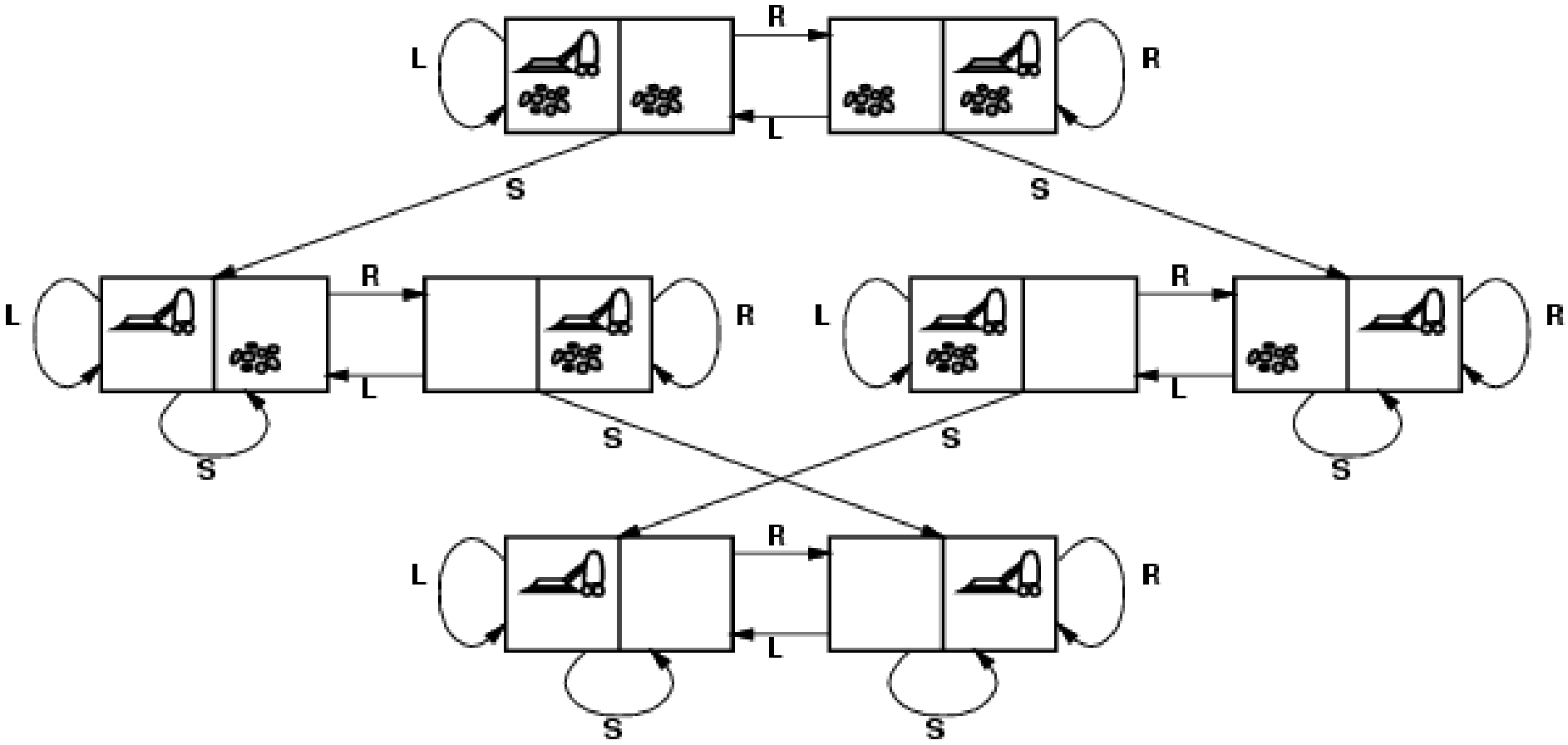


- ✓ Ranger des caisses sur des cases cibles.
- ✓ Il peut se déplacer dans les quatre directions, et pousser (mais pas tirer) une seule caisse à la fois.

Formalisation d'un problème de recherche

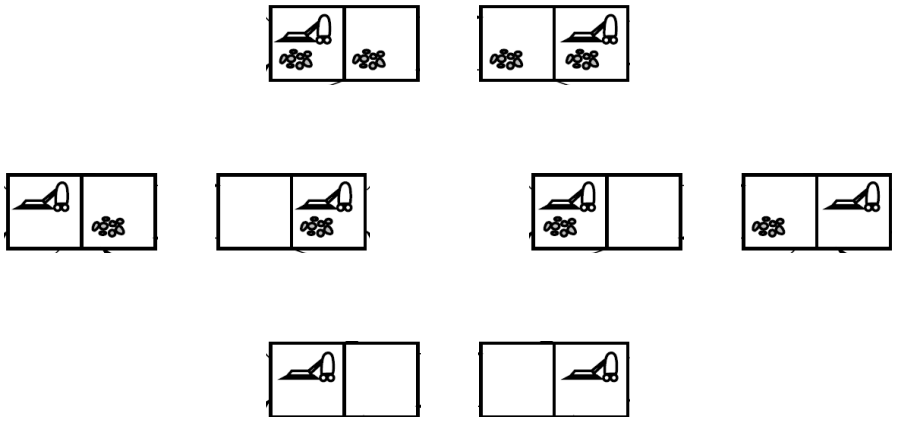
- Entrées
  - Un **nœud (état) initial** n_0 (situation initiale)**.**
  - Une fonction **but(*n*)** qui retourne *true* ssi le but est atteint dans le nœud *n*.
  - Une fonction de transition **successeurs(*n*)** qui retourne/génère les nœuds successeurs de *n*.
- Sortie
  - **Solution = chemin dans un graphe** (séquence nœuds / actions)
- Le **coût d'une solution** est la **somme des coûts des arêtes** dans le graphe.
- Il peut y avoir plusieurs nœuds (états) satisfaisant le but.

Espace des états d'un micro-monde: *Robi l'aspirateur*

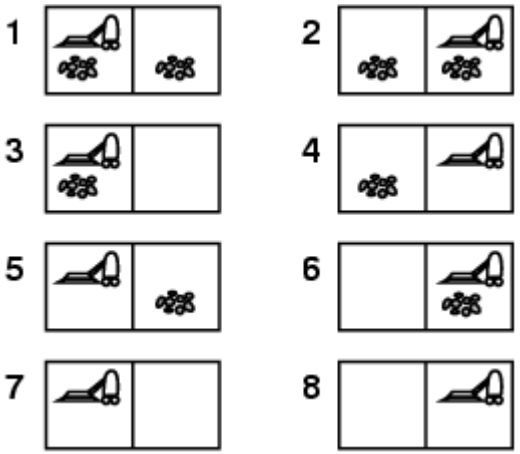


Représentation d'un problème

- Définitions
  - États du problème

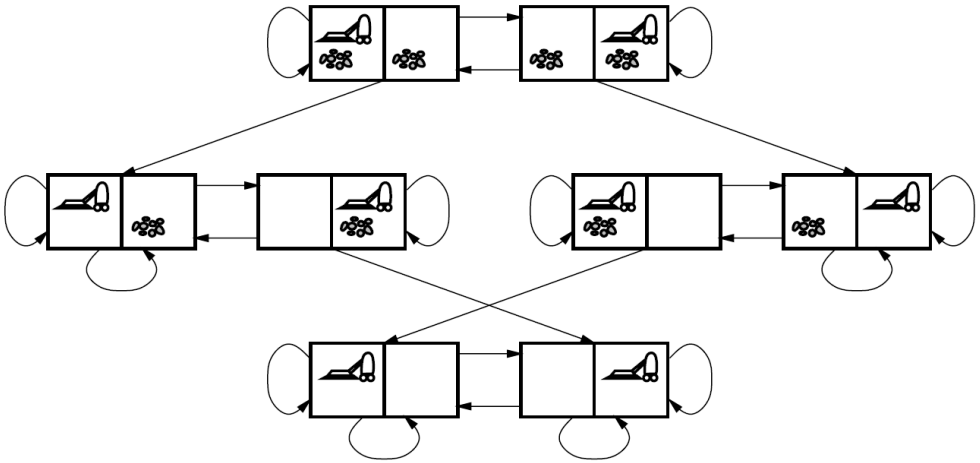


huits états possibles



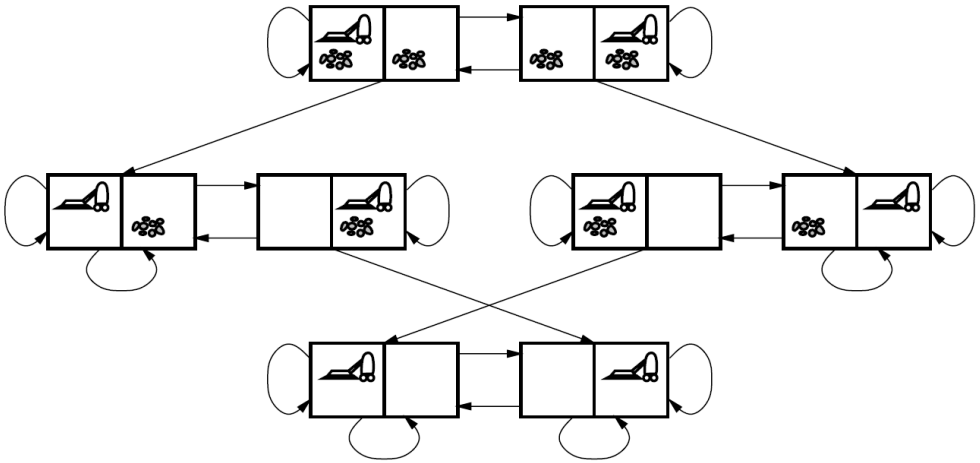
Représentation d'un problème

- Définitions
  - États du problème
  - Transitions (Opérateur de transformation d'état)
  - État initial

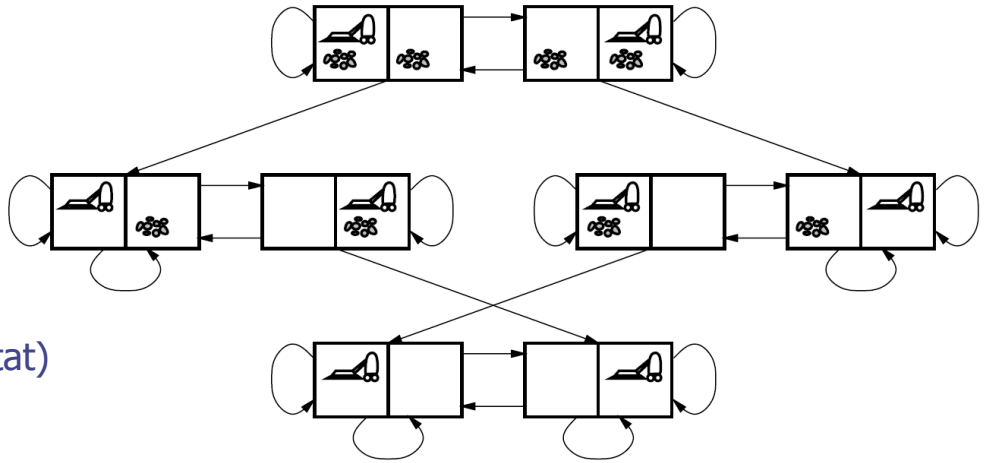


Représentation d'un problème

- Définition
  - États du problème
  - Transitions (opérateur de transformation d'état)
  - État initial
  - État final (But)



Représentation d'un problème



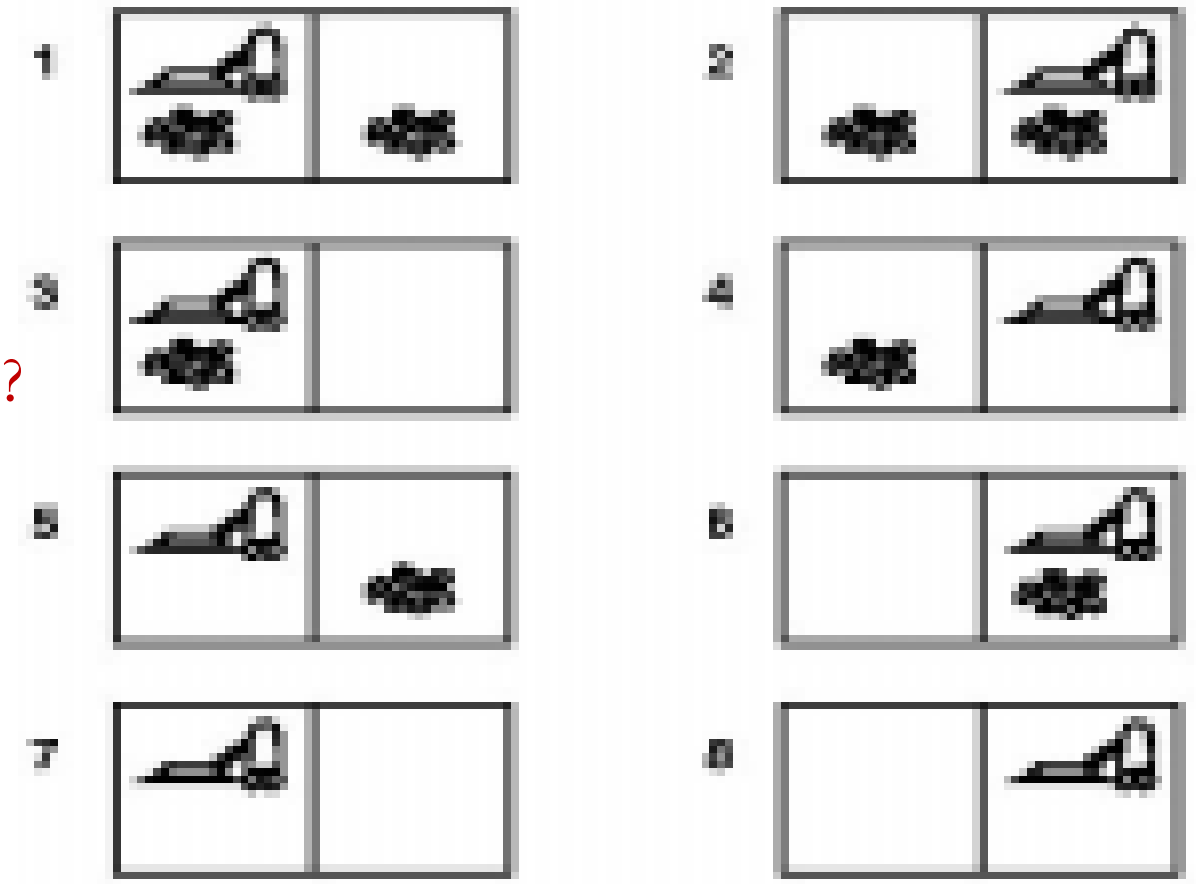
- Définitions
  - États du problème
  - Transitions
    (opérateur de transformation d'état)
  - État initial
  - État final (but)

- Solution au problème = Résultat de la recherche:
  - Un chemin (une séquence d'états allant de l'état initial à l'état final souhaité (au but)

- Lorsqu'un chemin est trouvé, l'agent l'exécute du début à la fin.
- Question pour vous:
  - Identifiez une limitation importante de ce fonctionnement
    - L'environnement peut changer après que le chemin a été trouvé -- Requiert un envir. statique
      - Suppose qu'on connaît tous les états au départ
      - Suppose que les effets des actions de l'agent sont connus et certains

Le monde de l'agent aspirateur

Etats??
Etat Initial??
Actions??
Test du but??
Coût du chemin??



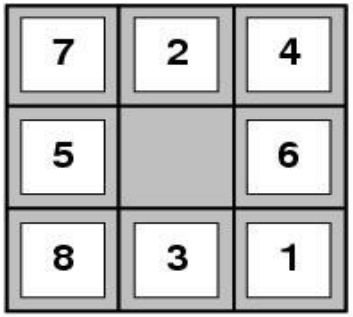
Le monde de l'agent aspirateur

- Etats: L'agent est dans un des deux emplacements, qui peuvent contenir ou non de la poussière (8 états en tout)
- Etat Initial: n'importe quel état
- Actions: Gauche, Droite, Aspirer
- Test du but: Vérifier que tous les emplacements sont propres
- Coût du chemin: somme du nombre d'étapes qui composent le chemin

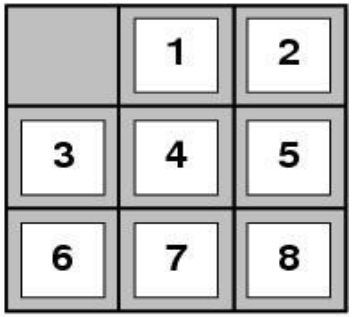
Le jeu de taquin à 8 pièces

- Plateau de 3X3 cases
- 8 cases sont occupées par une pièce numérotée
- 1 case est vide
- But: Atteindre une configuration donnée à partir d'un état initial

- Etats??
- Etat Initial??
- Actions??
- Test du but??
- Cout du chemin??



Start State



Goal State

Le jeu de taquin à 8 pièces

- Etats: L'emplacement des 8 pièces et celui de la case vide
- Etat initial: n'importe quel état
- Fonction successeur: génère les états pouvant résulter de l'exécution de 4 actions (Déplacement vers Gauche, Droite, Haut ou bas)
- Etat final: l'état correspond à la configuration finale
- Coût du chemin: nombre d'étapes qui composent le chemin (coût de chaque étape = 1)

Autres exemples de situations nécessitant la recherche de solutions

- Disposition des composantes sur un circuit intégré
  - situation initiale: des millions de composantes non organisées
  - Transitions possibles indiquées par les contraintes de connectivité
    - pas de superpositions de composantes
    - espace disponible pour passer les chemins entre les composantes
  - But : disposition qui minimise l'espace, les délais de transport de l'information, les pertes d'énergie, et les coûts de fabrication
- Assemblage robotique
- Recherche d'un trajet pour aller de Montréal à l'Auberge du Petit Bonheur
- Conception de protéines

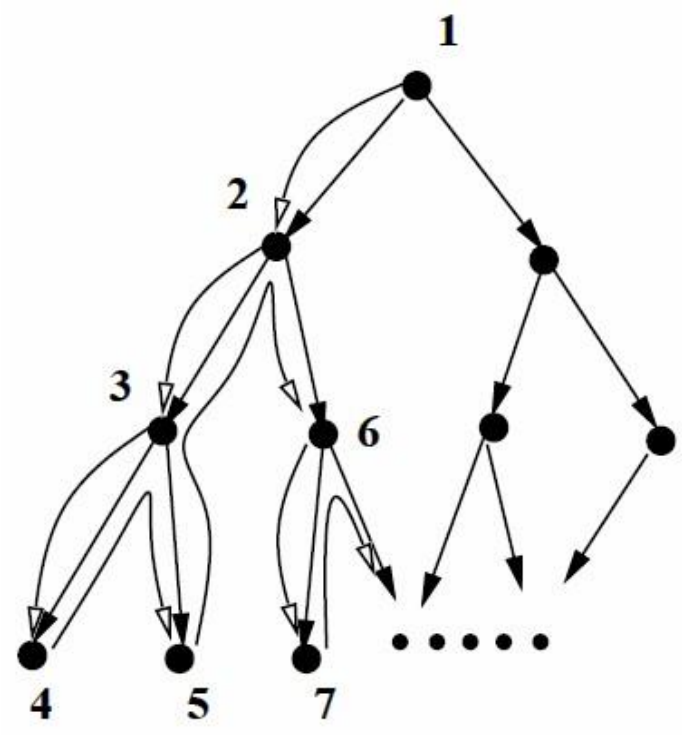
Stratégies de *recherche*: aveugle ou informée

- Recherche aveugle
  - Profondeur d'abord
  - Largeur d'abord
  - Profondeur itérative
  - Largeur itérative
- Recherches informées (avec heuristique)
- Jeux avec adversaire

Stratégie de recherche: objectif et évaluation

- La stratégie indique un ordre de développement des noeuds
- 4 dimensions d'évaluation d'une stratégie:
  - Complétude : permet-elle de toujours trouver une solution s'il en existe une ?
  - Complexité en temps : nombres de noeuds générés pour trouver la solution
    - Question : importance de ce critère? → combien de temps il faut pour trouver la solution
  - Complexité en espace: nombre maximal de noeuds mémorisés
    - Question : importance de ce critère? → combien de mémoire il faut pour effectuer la recherche
  - Optimalité : permet-elle de toujours trouver le chemin le moins coûteux (ex.: le plus court)?
- La complexité en temps et en espace se mesurent en termes de :
  - *b :* facteur de **b**ranchement maximal dans l'arbre de recherche
  - *d :* profondeur du but le plus "en surface"
    (du plus court chemin vers un but, de la solution la moins coûteuse)
  - *m* : profondeur **m**aximale de l'espace d'état
    (longueur maximale de n'importe quel chemin; peut être ∞)

Stratégie de recherche
En profondeur d'abord



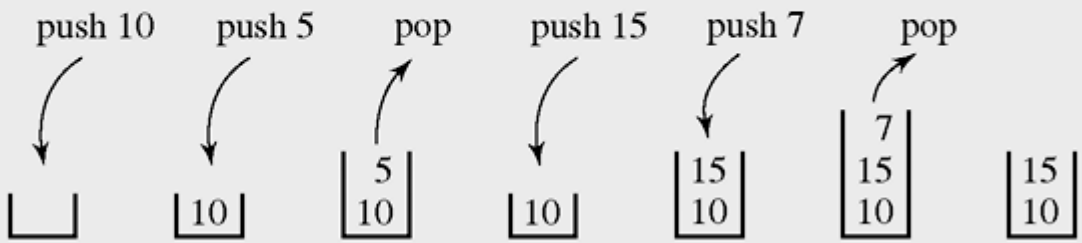
Source: École polytechnique fédérale de Lausanne

- Expansion (et examen) du premier noeud jusqu'à ce qu'il n'y ait plus de successeur
- Retour arrière au niveau précédent
- Modeste en mémoire: ne conserve en mémoire que
  - le chemin direct actuel allant de l'état initial à l'état courant,
  - les enfants (noeuds ouverts) non examinés du parent actuel
  - les noeuds en attente le long du chemin parcouru.

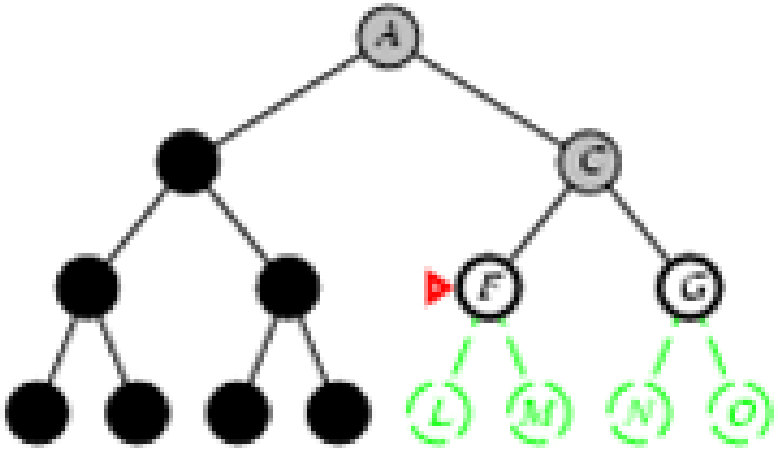
Stratégie de recherche
En profondeur d'abord

- Développer le nœud le plus profond non encore développé
- Implantation: *Exploration_en_arbre* avec une file LIFO=pile => mettre les successeurs à l'avant

FIGURE 4.1 A series of operations executed on a stack.



Stratégie de recherche
En profondeur d'abord (suite)

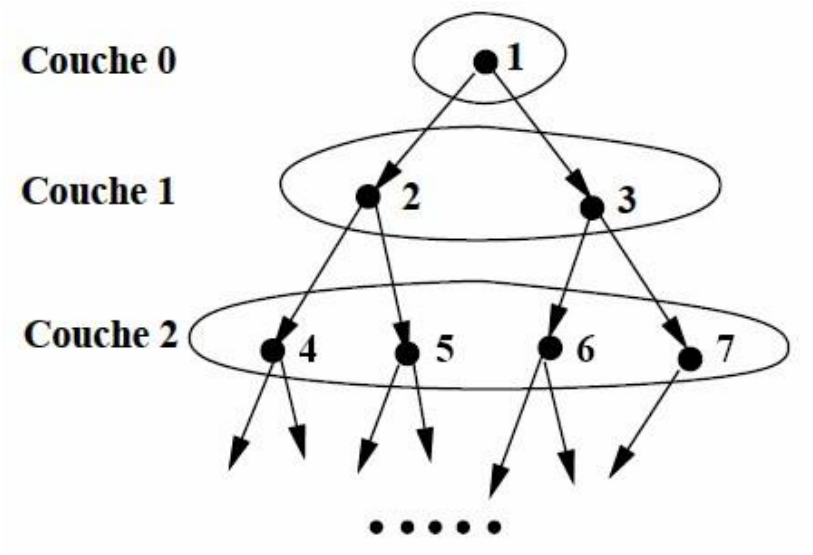


Recherche en profondeur d'abord
Propriétés

- Complète?
  - Non: échoue dans les cas d'espace à profondeur infinie ou d'espace avec boucles.
- Optimale?
  - Non
- Temps?
  - $O(b^m)$: peut être terrible si m (profondeur de l'espace) > d (profondeur du but)
  - Il peut y avoir beaucoup d'exploration en profondeur avant de trouver un but en surface, mais à la droite du graphe
- Espace?
  - $O(bm)$, i.e., espace linéaire

- Une solution: établir une profondeur limite d'exploration

Stratégie de recherche
En largeur d'abord



Source: École polytechnique fédérale de Lausanne

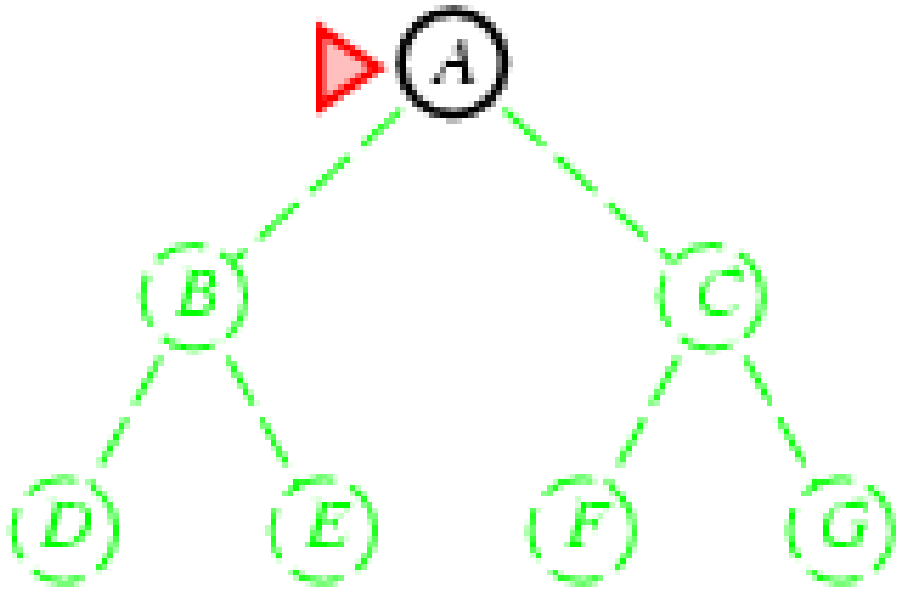
- Expansion et examen par couche
- Trouve toujours le chemin le plus court
- Exige beaucoup de mémoire pour stocker toutes les alternatives à toutes les couches.

Exploration en largeur d'abord

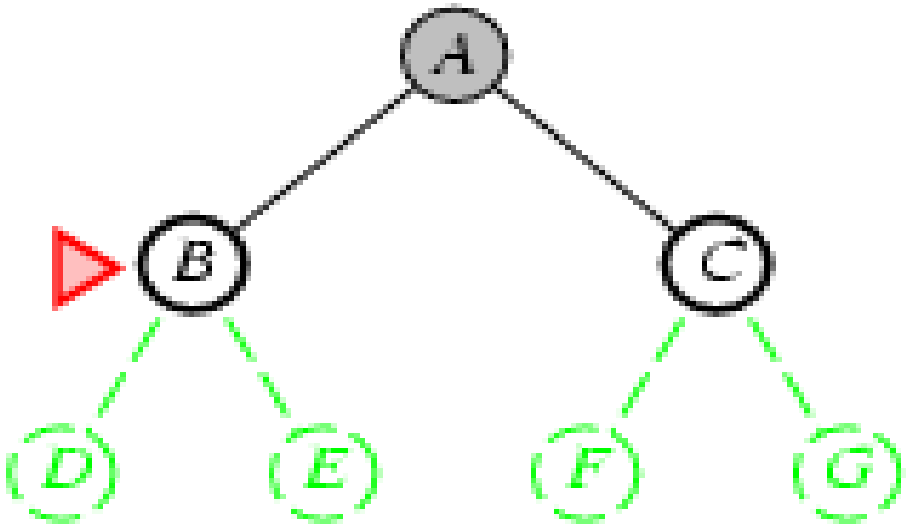
- Principe: Développer les nœuds à un même niveau de l'arbre avant d'aller au niveau suivant.
- La frontière est implantée sous forme de file FIFO (retirer le nœud le plus ancien)
- Développement: mettre les enfants à la fin de la file
- Retirer les nœuds à l'avant de la file

Exploration en largeur d'abord

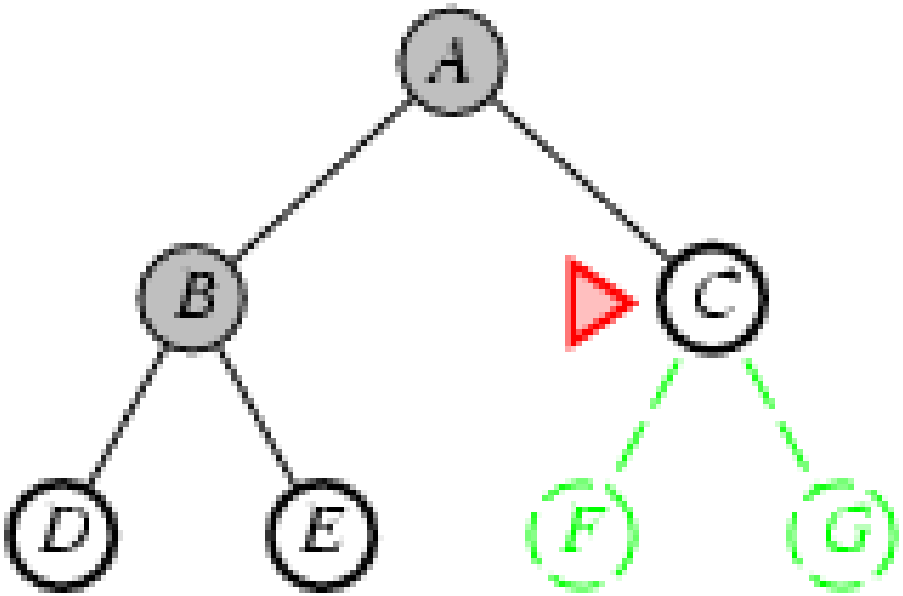
- Développer tous les nœuds d'une profondeur donnée avant de passer au niveau suivant
- Implantation: Utiliser une file FIFO, i.e., les nouveaux successeurs sont rangés à la fin.



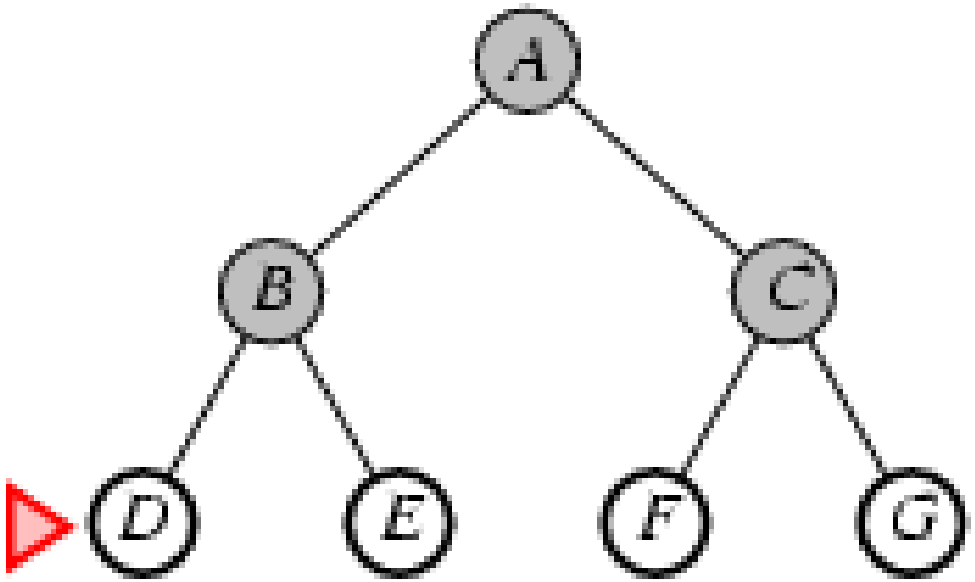
Exploration en largeur d'abord



Exploration en largeur d'abord



Exploration en largeur d'abord



Recherche *en largeur d'abord*
Propriétés

- Complète?
  - Oui (si *b* est fini)
- Optimale?
  - Oui (si le coût par étape = 1)
- Temps?
  - $1+b+b^2+b^3+\ldots+b^d + b(b^d-1) = O(b^{d+1})$
- Espace?
  - $O(b^{d+1})$ (garde tous les noeuds en mémoire)

- L'espace est le gros problème avec cette stratégie (plus que le temps)

Le meilleur des deux mondes: Exploration Itérative en profondeur

- Profondeur d'abord est efficace en espace mais ne peut garantir un chemin de longueur minimale
- Largeur d'abord trouve le chemin le plus court (en nombre d'étapes) mais requière un espace exponentiel
- Exploration Itérative en profondeur effectue une recherche en profondeur limitée avec un niveau de profondeur croissant jusqu'à ce que le but soit trouvé.

Exploration Itérative en profondeur

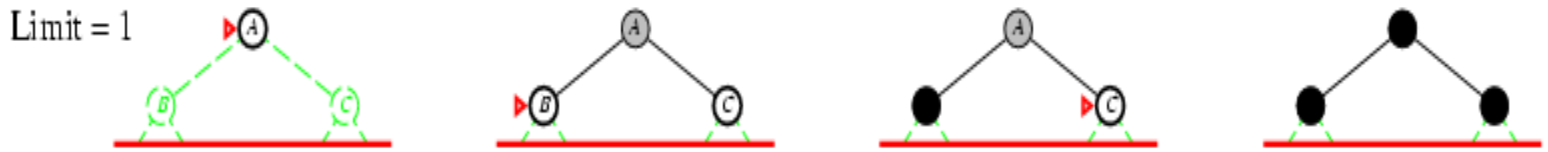
Augmentation graduelle de la limite

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or fail-
ure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
```

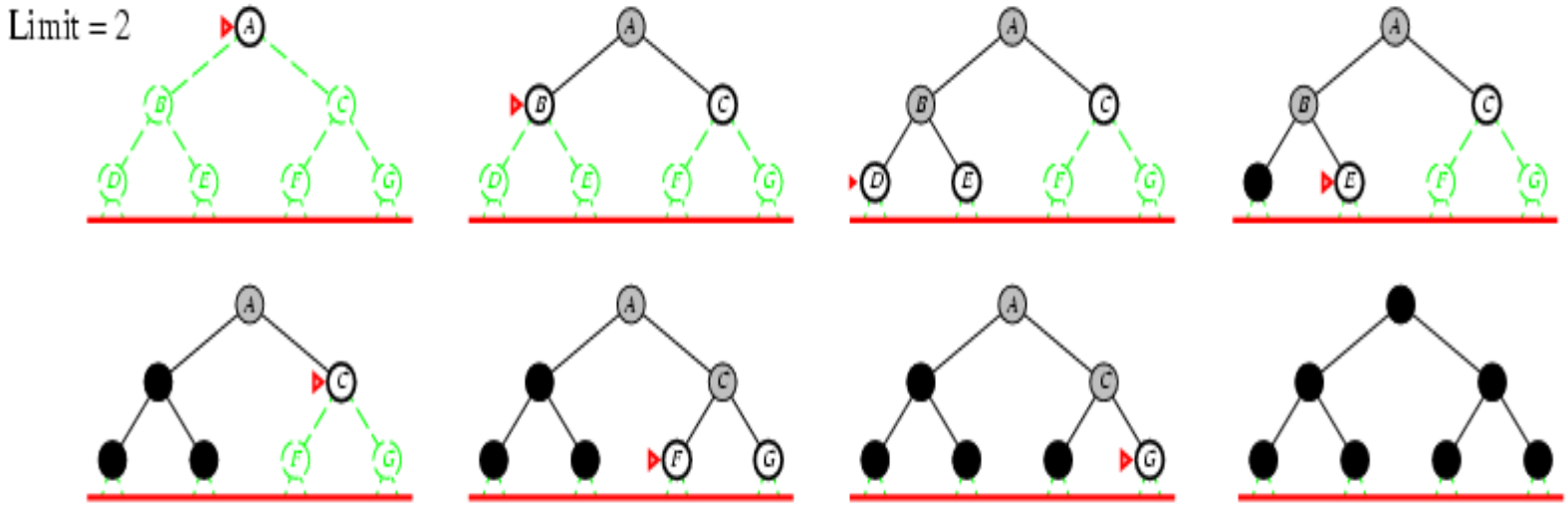
L =0

Limit = 0

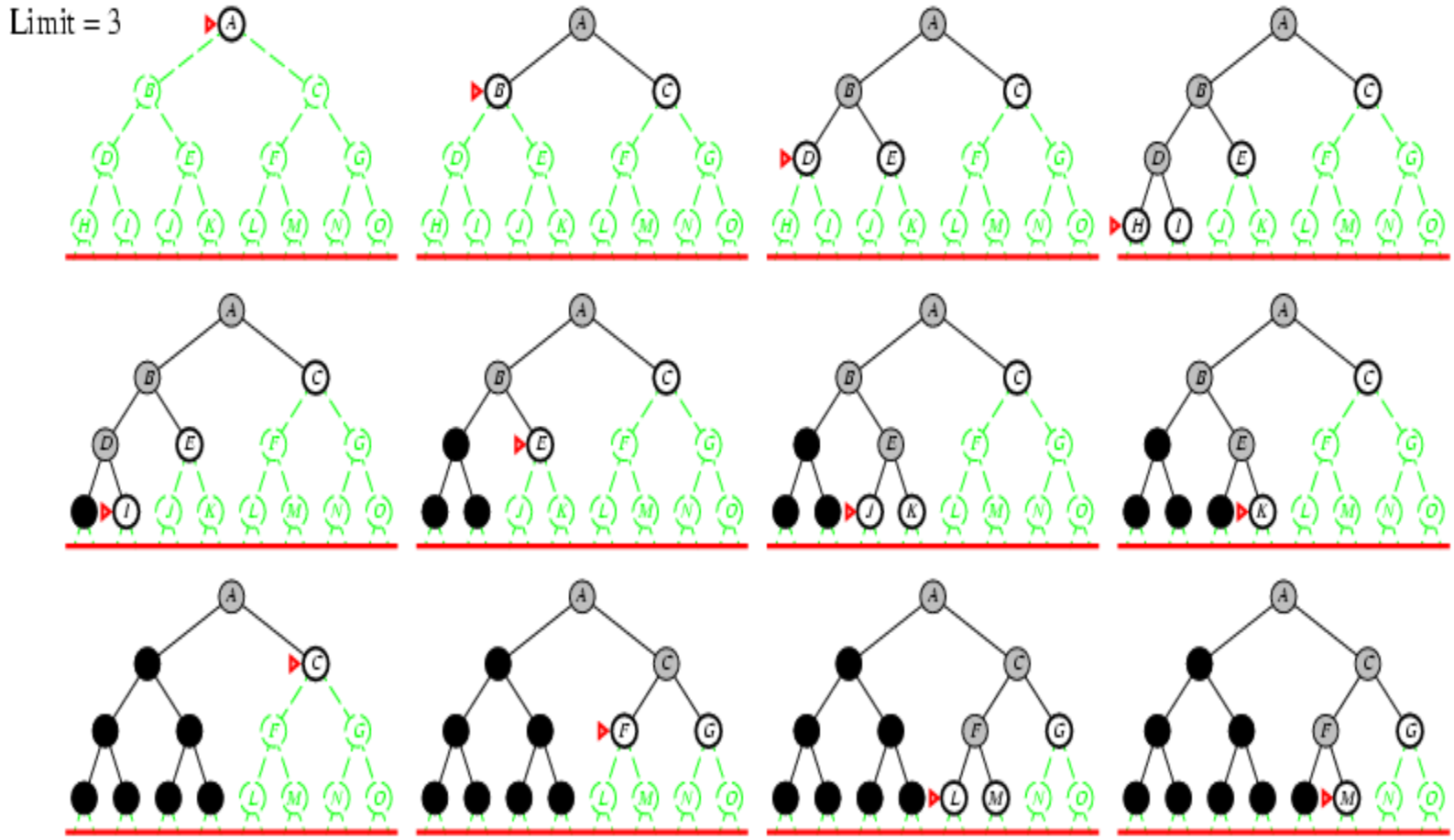
L=1



L=2



L=3



Propriétés de Profondeur Itérative en profondeur

- Complet?
  - Oui (si facteur de branchement est fini)
- Temps?
  - $d\,b^1 + (d-1)b^2 + \ldots + (1)b^d = O(b^d)$
- Espace?
  - $O(bd)$
- Optimal?
  - Oui, si le coût de chemin= fonction non décroissante de la profondeur du nœud

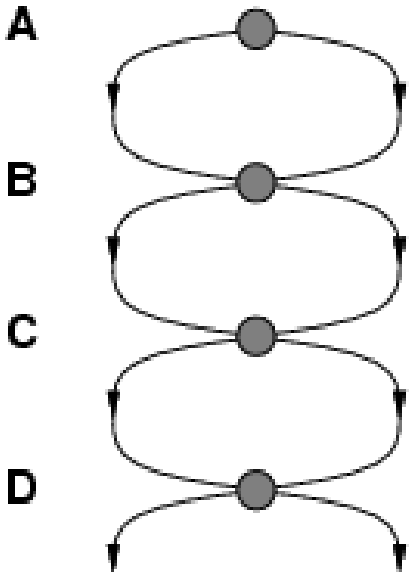
Résumé

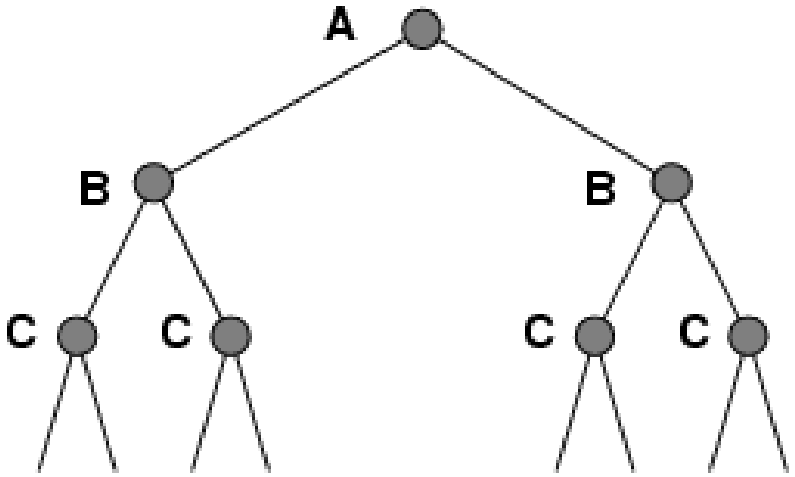
Criterion	Breadth-First	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	No	No	Yes

b: facteur de branchement de l'arbre de recherche
d: profondeur de la solution la moins couteuse
m: prof. max de l'espace

États répétitifs

- Si ces états ne sont pas détectés la recherche peut devenir exponentielle → une recherche théoriquement possible peut devenir techniquement impossible!





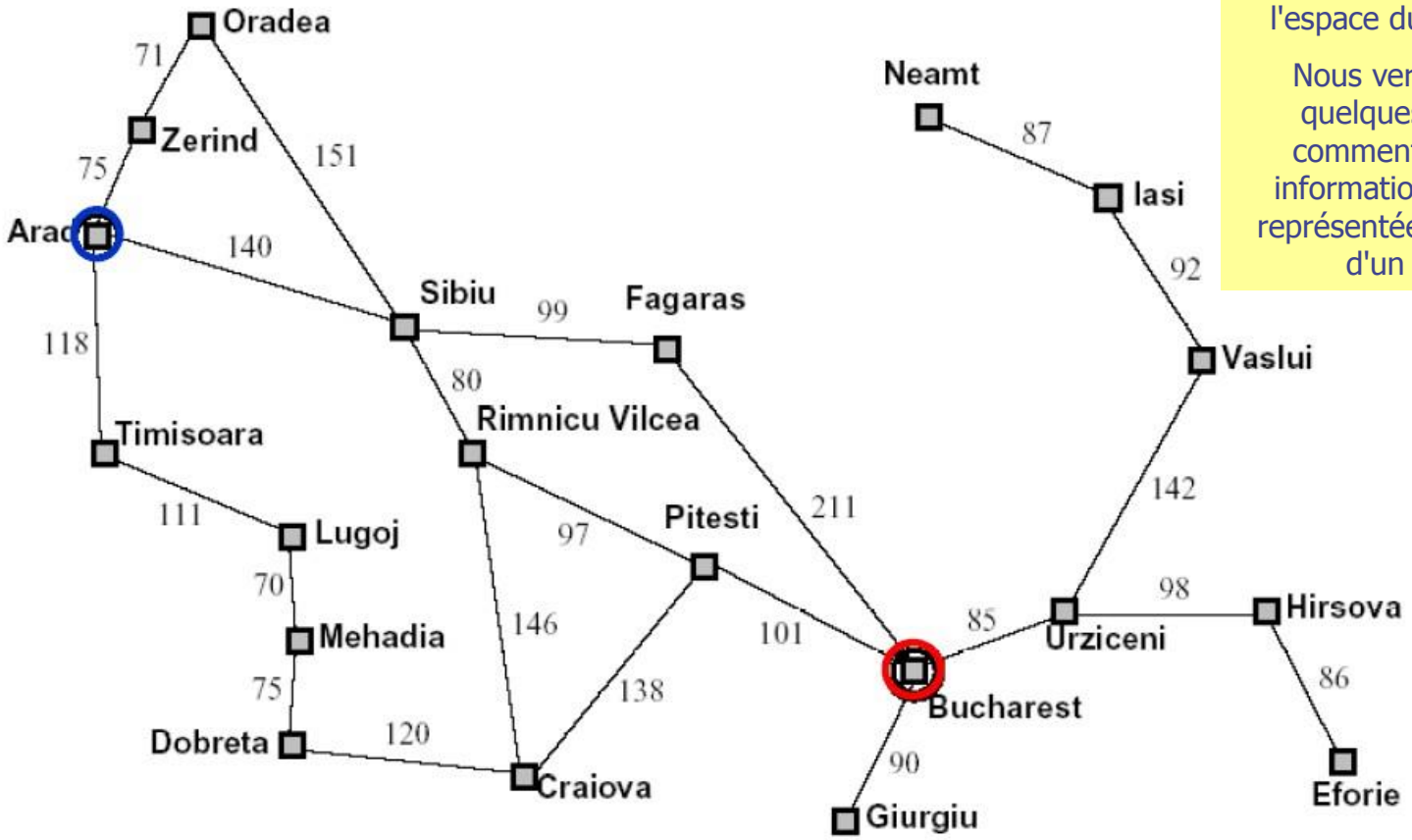
- Solution: Garder les noeuds déjà visités dans une liste appellée Closed

Problème de la recherche aveugle

- Ne garde pas les états du chemin solution au cours de la recherche
- Complexité en espace
  - Dans tous les cas, le nombre de nœuds à mémoriser croît de façon exponentielle
  - Envisager l'élaguation de l'espace de recherche par les heuristiques (recherches informées)

Planifier un voyage

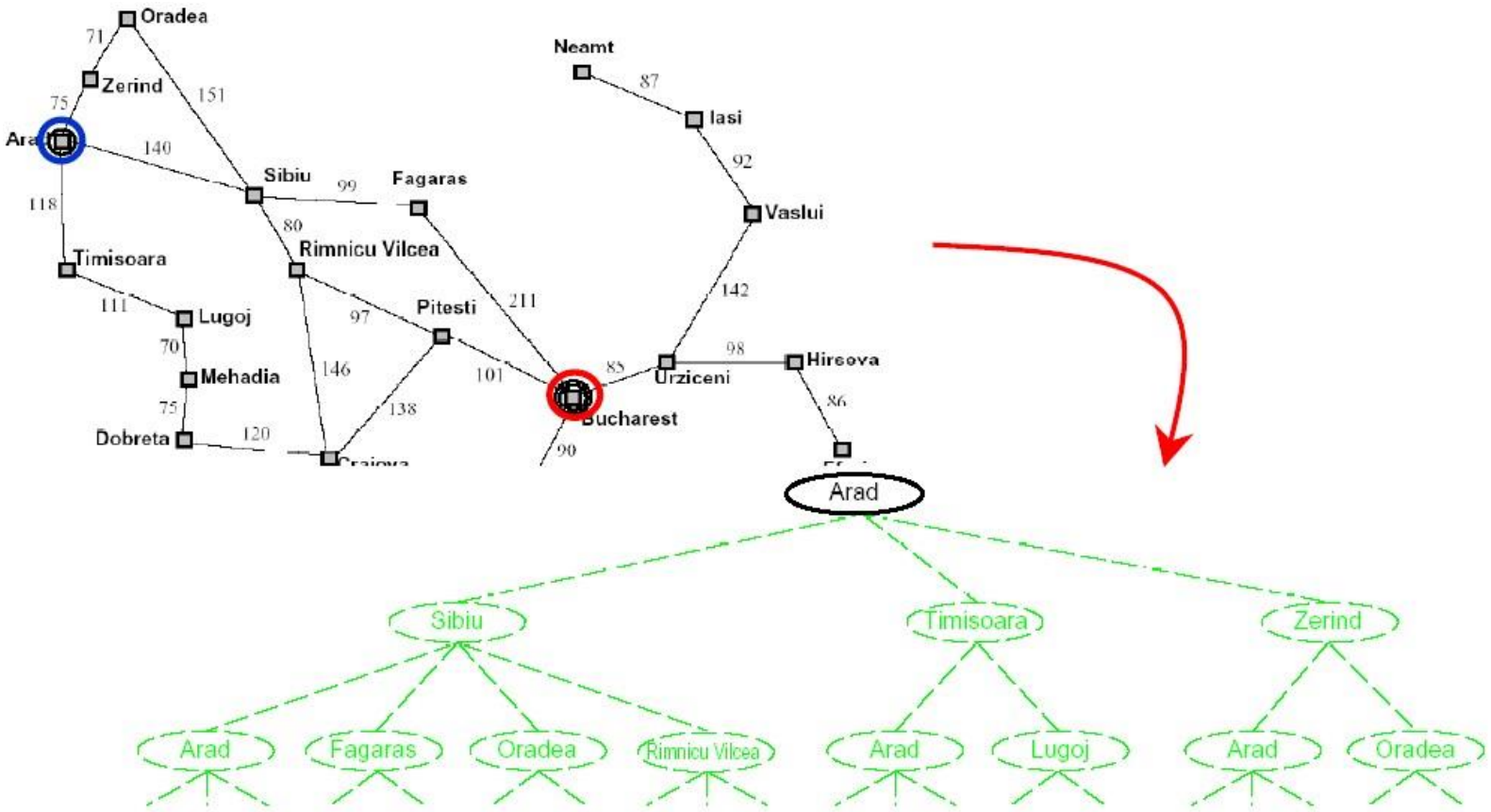
- Pour aller d'Arad à Bucharest, utilisons les multiples stratégies de recherche aveugle présentées jusqu'ici.



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: Justifications

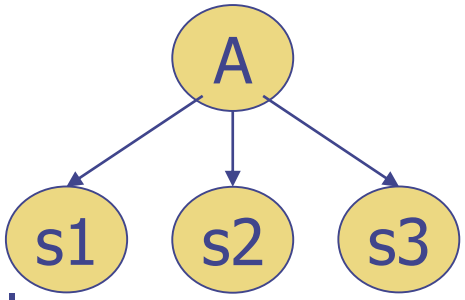
- Les algorithmes de recherche aveugle n'exploitent aucune information concernant la structure de l'arbre de recherche ou la présence potentielle de noeuds-solution pour optimiser la recherche.
- Recherche "rustique" à travers l'espace jusqu'à trouver une solution.
- La plupart des problèmes réels sont 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

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

  fait correspondre à un état $\boldsymbol{s} \in E$ (espace d'états) un nombre $\boldsymbol{h(s)} \in R$ 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



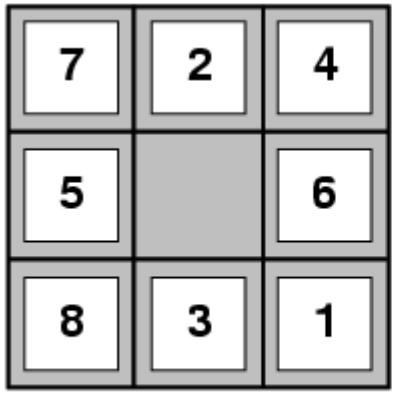
- Le noeud A a 3 successeurs pour lesquels:

  h(s1) = 0.8 *h(s2)* = 2.0 *h(s3)* = 1.6
- la poursuite de la recherche par *s1* est heuristiquement la meilleure

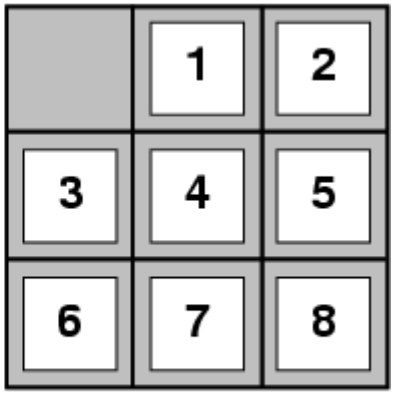
Exemple d'heuristique : *Distance de Manhattan*

Taquin à 8 plaquettes
Suggestion de deux heuristiques possibles:

- $h_1(n)$ = nombre de plaquettes mal placées
- $h_2(n)$ = distance de Manhattan
  - déplacements limités aux directions verticales et horizontales
  - somme des distances de chaque plaquette à sa position finale



Start State



Goal State

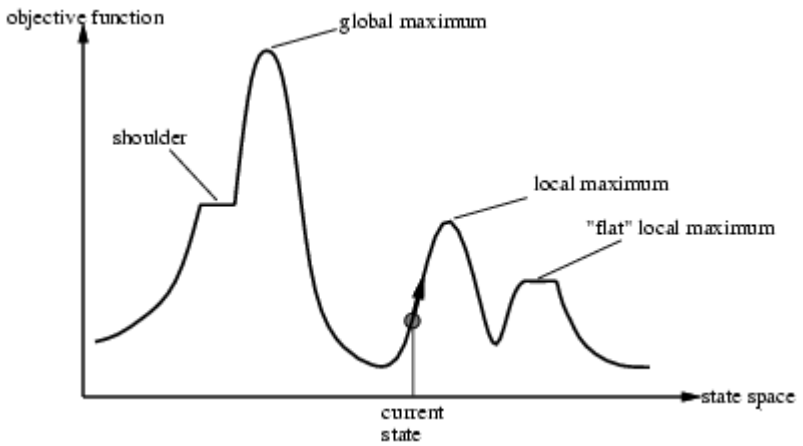
- $h_1(n) = 8$
- $h_2(n) = 3+1+2+2+$

Utilisation d'une heuristique : recherche par *Hill-Climbing*

- Idée:
  - La recherche *par escalade* n'examine que ses voisins immédiats
    - Développer le nœud courant
    - Évaluer ses fils et choisir le meilleur
  - Élimine les frères et les parents du noeud retenu
  - Arrête à l'atteinte d'un état meilleur que tous ses fils.

Recherche par *Hill-Climbing*
Inconvénients

- Inconvénient:
  - Non complète (ne garantit pas de trouver la solution)
  - Sans mémoire => pas de retour arrière possible => On ne peut pallier à un échec
  - Problème du maximum local : La recherche s'arrête parce qu'il a atteint un état meilleur que tous ses fils mais pas forcément meilleur que tous les noeuds.



Source: Artificial Intelligence: A Modern Approach (Second Edition) by Stuart Russell and Peter Norvig
http://aima.eecs.berkeley.edu/slides-ppt/m4-heuristics.ppt

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

- L'algorithme "Best-First search" permet d'explorer les noeuds dans l'ordre (meilleur-que) de leurs valeurs heuristiques
- fonctions heuristiques classiques
  - 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:
  - *Open* = liste ordonnée des noeuds à examiner
  - *Closed* = liste des noeuds déjà examinés

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

```
Open ← [start]; Closed ← [];
Tant que Open n'est pas vide
   noeud ← enleverPremier (Open)
  Si noeud = but, retourner le chemin solution
  Sinon
        Générer les enfants de noeud
        Pour chaque enfant
          calculer sa valeur heuristique (h(enfant))
         cas:
          - l'enfant n'est ni dans Open ni dans Closed
                Ajouter dans Open
          - l'enfant est dans Open
               Si h(enfant) meilleur que sa valeur dans Open
                   remplacer la valeur dans Open
          - l'enfant est dans Close
                 Si h(enfant) meilleur que sa valeur dans Closed
                   enlever l'enfant de Closed et le mettre dans Open avec la valeur
        h(enfant)
          Fin Cas
        FinPour
      FinSi
      Mettre noeud dans Closed; Reordonner Open;
   FinTantQue
   Afficher ("Échec")
```

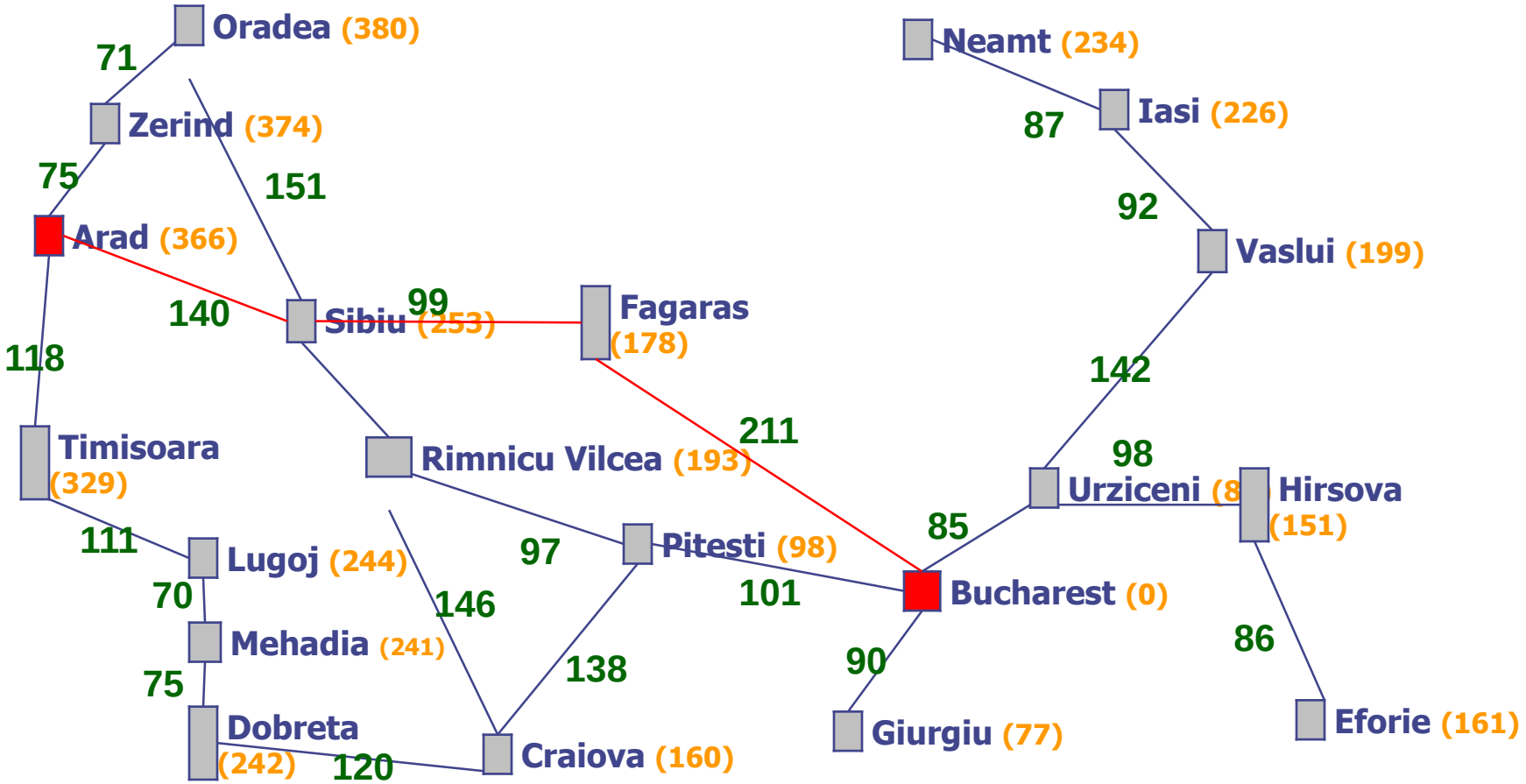
- Cas particuliers de "best-first search":
  - A: avec historique
  - A*: A + évaluation *avare*

Recherche Meilleur-d'abord *avare*

- Stratégie la plus simple des "best-first search"
- fonction heuristique *h(n)* = estimation du coût du noeud n **jusqu'au but**
- recherche avare = minimiser le coût estimé pour atteindre le but
  - le noeud 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(Bucharest)
- Voyage: **h(n) = distance_en_ligne_droite (n, but)**



Voyage en Roumanie
expansion des noeuds 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

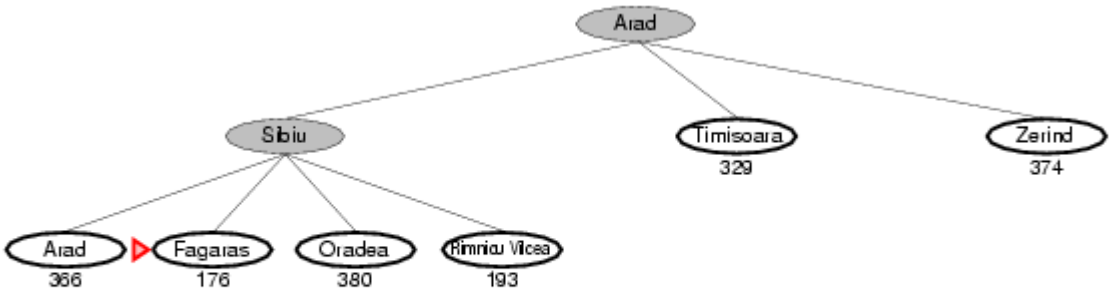
Voyage en Roumanie

expansion des noeuds 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

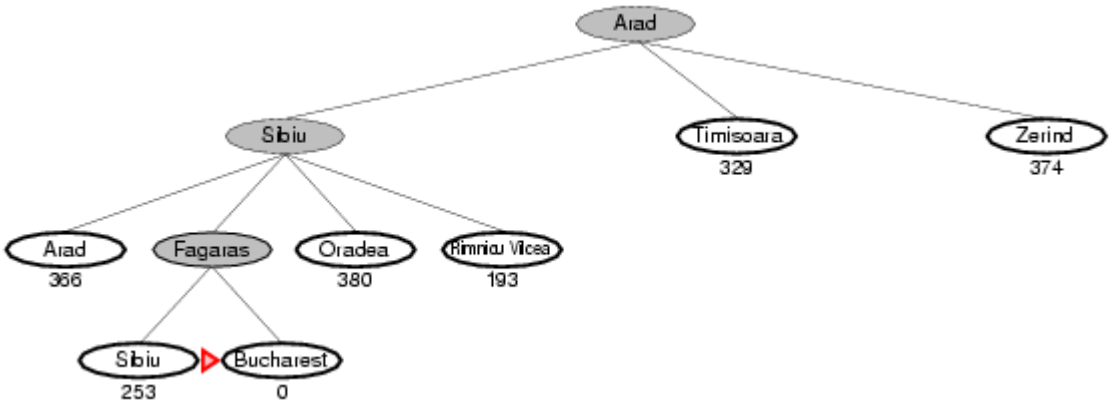
expansion des noeuds par recherche avare



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 noeuds 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 noeuds en mémoire
-
- Admissible? Non (on ne tient pas compte de l'historique du noeud)
  - *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=418 km)*

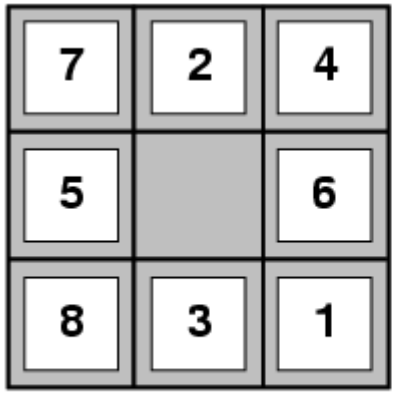
Fonction heuristique *admissible*

- 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

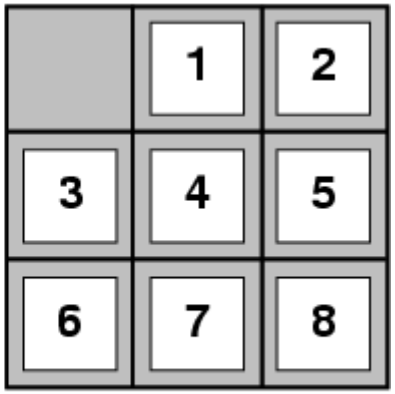
- Autrement dit: ne surestime jamais le coût réel
- Une fonction heuristique admissible est donc toujours optimiste!
- *La recherche avare* minimise le coût estimé $h(n)$ du noeud n au but, réduisant ainsi considérablement le coût de la recherche, mais il n'est pas optimal et pas complet (en général)
- l'algorithme de recherche en coût uniforme minimise le coût $g(n)$ depuis l'état initial au noeud n, il est optimal et complet, mais pas très efficace
- idée: combiner les deux algorithmes et minimiser le coût total $f(n)$ du chemin passant par le noeud n

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

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 + 3x(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

Algorithme A & A*

- Soit la fonction d'évaluation

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

- Meilleur-d'abord + f(n) comme fonction d'évaluation = algorithme **A**
- 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 h(n) = nombre de plaquettes mal placées

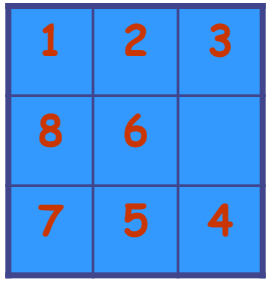
État initial

1	2	3
8	6	
7	5	4

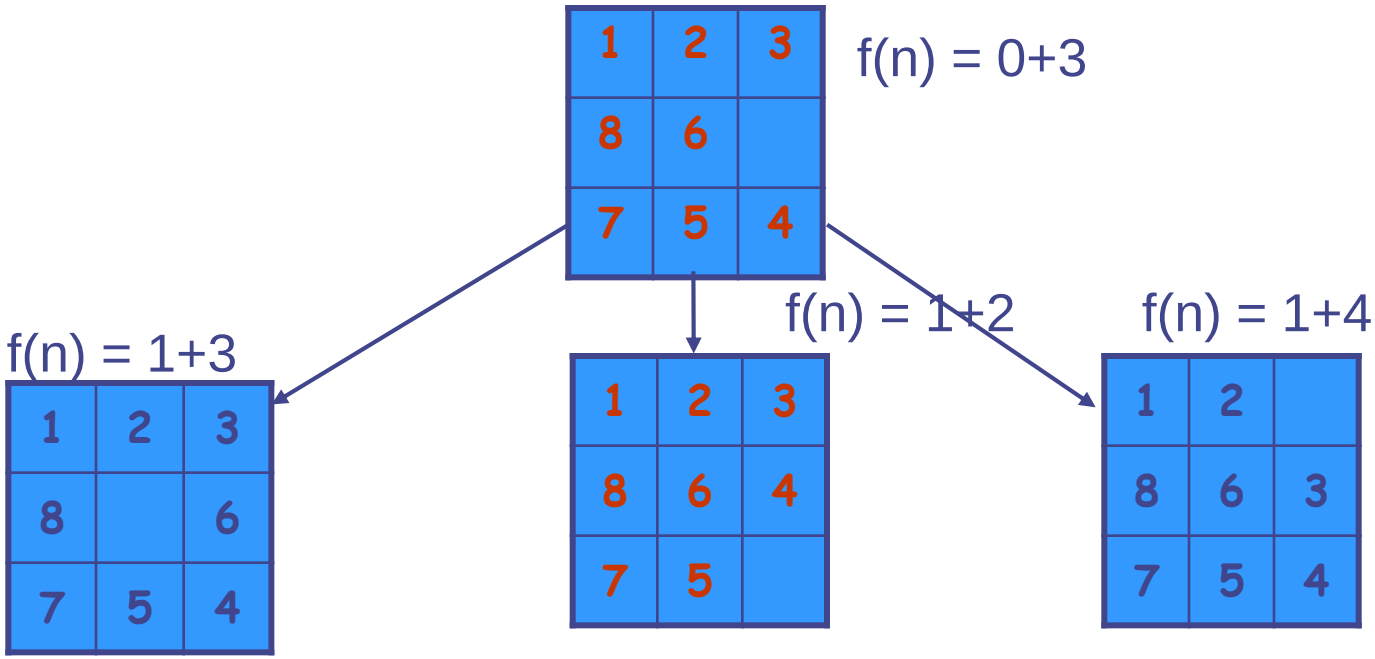
f(n) = 0+3

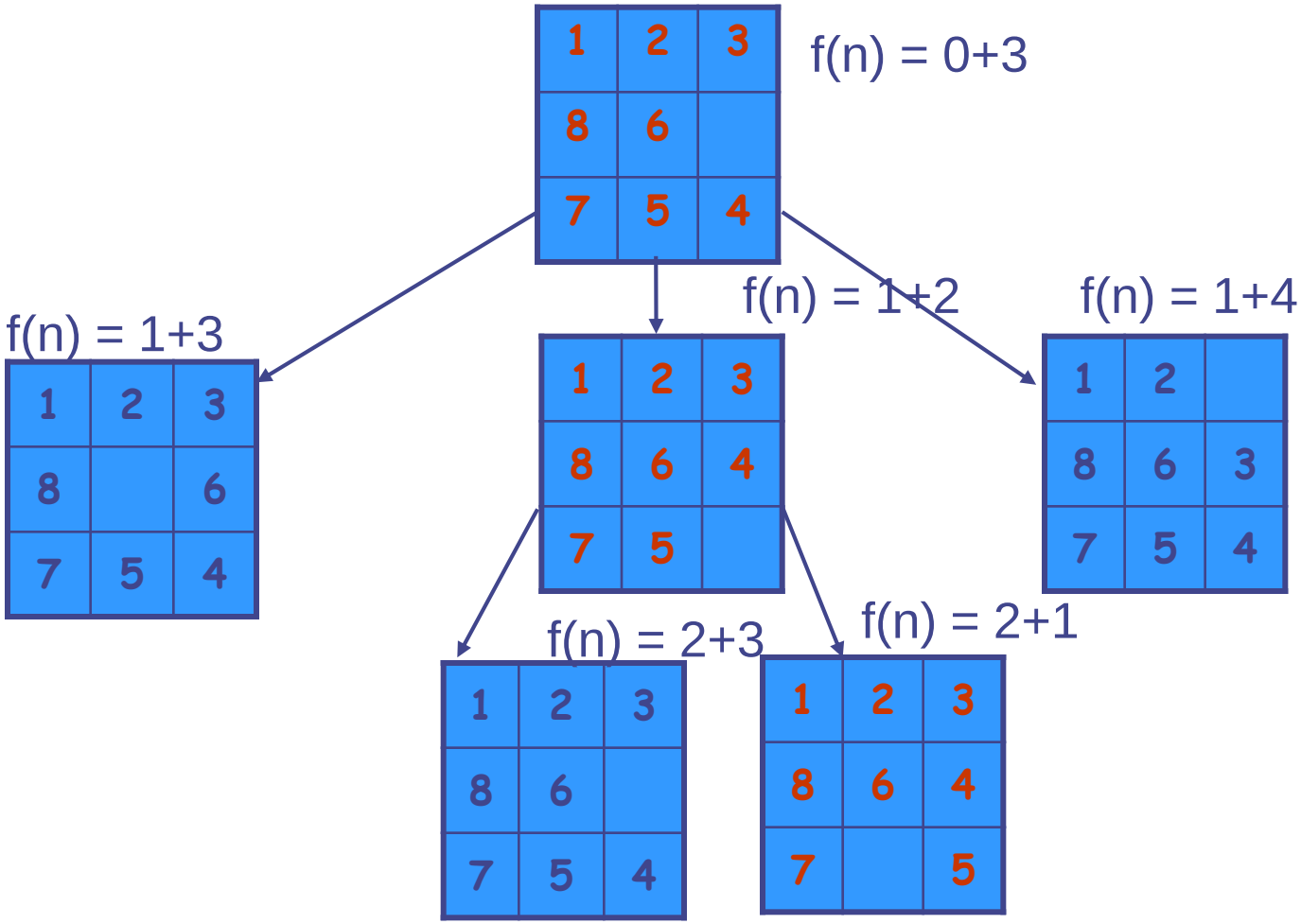
État but

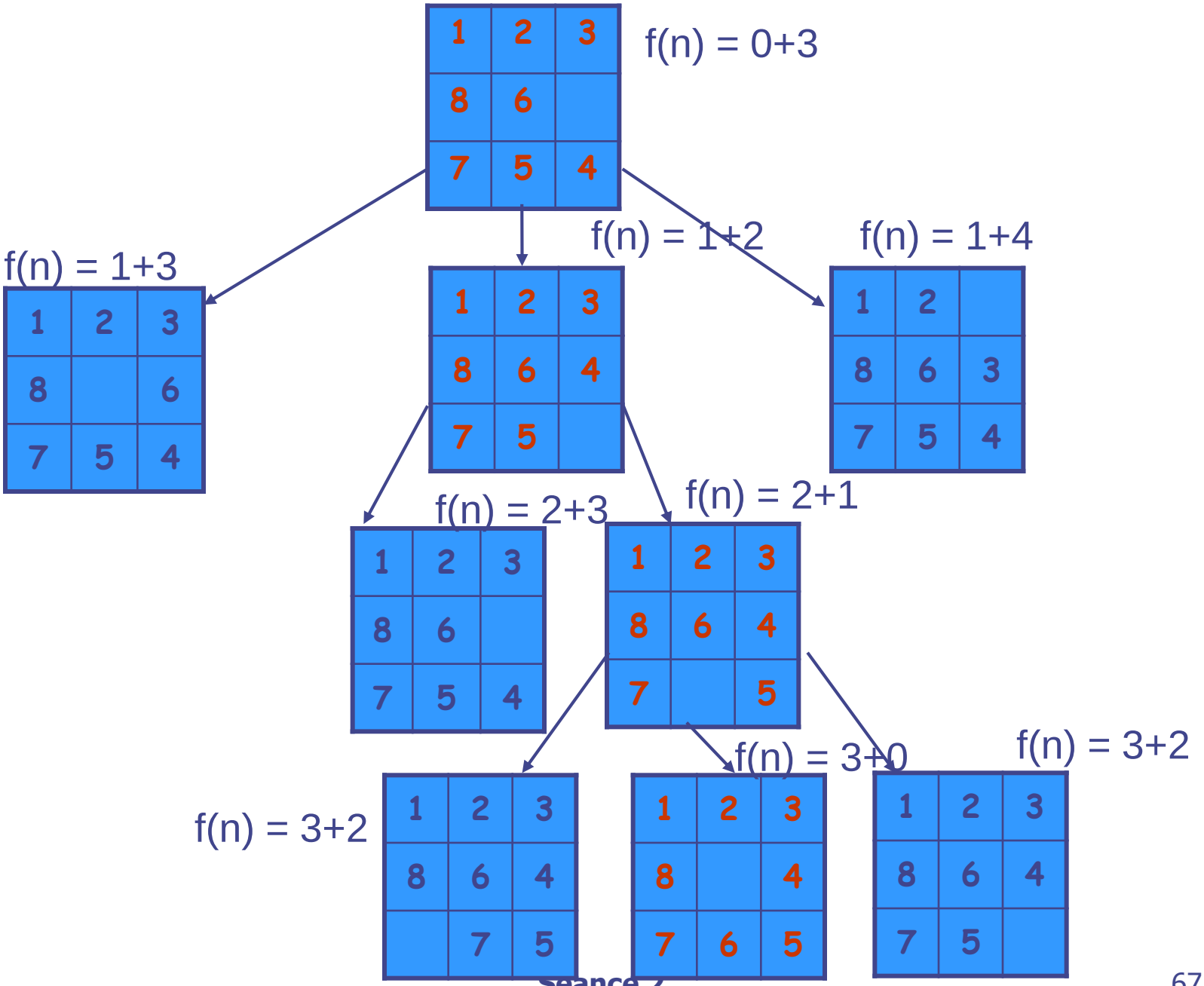
1	2	3
8		4
7	6	5



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

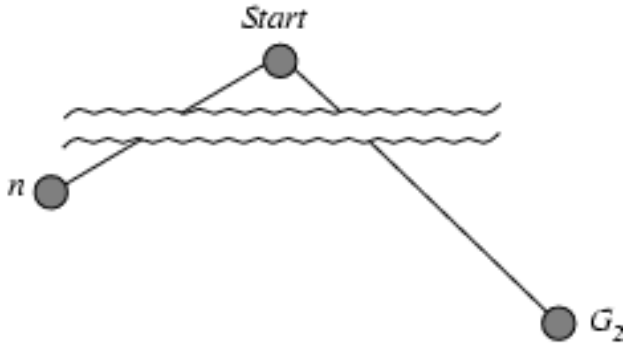

1 2 3
8 6
7 5 4
f(n) = 0+3
f(n) = 1+3
1 2 3
8 6
7 5 4
f(n) = 1+2
1 2 3
8 6 4
7 5
f(n) = 1+4
1 2
8 6 3
7 5 4
f(n) = 2+3
1 2 3
8 6
7 5 4
f(n) = 2+1
1 2 3
8 6 4
7 5
f(n) = 3+2
1 2 3
8 6 4
7 5
f(n) = 3+0
1 2 3
8 4
7 6 5
f(n) = 3+2
1 2 3
8 6 4
7 5

Propriétés de A*

- Complète?
  - Oui – sauf s'il y a un nombre infini de noeuds avec une valeur de $f \leq f(But)$
- Temps ?
  - *exponentielle*
- Espace ?
  - garde tous les noeuds en mémoire
- Admissible ?
  - Oui (peut être facilement prouver que A* ne trouve que le chemin le plus court)

Démo de l'optimalité (l'admissibilité) de A*

- Supposons l'existence d'un noeud but sous-optimal G_2. Soit *n* se trouvant sur le plus court chemin menant vers *G le but optimal*.
-



$f(G_2) = g(G_2)$	car $h(G_2) = 0$
$g(G_2) > g(G)$	car G_2 est sous-optimal
$f(G) = g(G)$	car $h(G) = 0$
$f(G_2) > f(G)$	de ce qui précède
$f(G_2) > f(G)$	de ce qui précède
$h(n) \leq h^*(n)$	car h est admissible
$g(n) + h(n) \leq g(n) + h^*(n)$	
$f(n) \leq f(G)$	

Ainsi $f(G_2) > f(n)$, et A* ne sélectionnera jamais G_2 comme candidat à l'expansion.

Qualité des fonctions heuristiques

- Facteur de branchement effectif
  - soit N = nombre total d'états produits pour obtenir la solution
  - soit d = profondeur à laquelle la solution a été trouvée
  - alors b* est le facteur de branchement d'un arbre fictif parfaitement équilibré tel que

$$N = 1 + b^* + (b^*)2 + \ldots + (b^*)d$$

exemple: si d = 5 et N = 52 > b* = 1.91

- Une bonne fonction heuristique aura une valeur de b* proche de 1
- (la fonction heuristique idéale aurait b* = 1)

Dominance

- Soient 2 fonctions heuristiques admissibles $h1(n)$ et $h2(n)$
  - Si $h2(n) \geq h1(n)$ $\forall n$ alors on dit que $h2$ domine $h1$ et produira une recherche plus efficace
- Exemples de coûts typiques de recherche (taquin-8)
  - d = 14
    - IDS (approfondissement itératif) = 3'473'941 états
    - A* ($h1$) = 539 états
    - A* ($h2$) = 113 états
  - d = 24
    - IDS = beaucoup trop d'états
    - A* ($h1$) = 39'135 états
    - A* ($h2$) = 1'641 états

Variantes de A*

- les problèmes réels sont souvent très complexe
  - l'espace de recherche devient très grand
  - même les méthodes de recherche heuristiques deviennent inefficaces
- A* connaît alors des problèmes de place-mémoire

- Y a-t-il des algorithmes de recherche "économes" en place-mémoire?
  2 variantes de A*:
  - IDA* = A* avec approfondissement itératif
  - SMA* = A* avec gestion de mémoire

Codes de plusieurs algorithmes de recherche ici:

- https://github.com/aimacode

Quand utiliser un algorithme de **recherche**?

- Lorsque l'espace de recherche est de petite taille et
  - qu'il n'y a pas d'autres techniques, ou
  - ce n'est pas rentable de développer une technique plus efficace.
- Lorsque l'espace de recherche est de grande taille et :
  - qu'il n'y a pas d'autre technique, et
  - qu'il n'existe pas de "bonne" heuristique.
- Procéder par recherche comporte des contraintes
  - devoir tout connaître à l'avance, et tout décrire!

Jeux avec adversaire

Jeux vs. problèmes de recherche

- Adversaire imprévisible → Spécifier un déplacement pour chaque possibilité de replique de l'adversaire
- Contraint dans le temps

Jeux vs. problèmes de recherche

- Jouer c'est rechercher le mouvement permettant de gagner
- Les jeux de plateau contiennent les notions de:
  - état initial et état gagnant (but)
  - opérateurs de transition (règles du jeu, déplacements de pièces)
- Une fonction heuristique permet d'estimer s'il y a gain, perte ou match nul
- Premiers algorithmes de jeu
  - algorithme pour jeu parfait, J. von Neumann (1944)
  - horizon fini, évaluation/approximation, K. Zuse (1945), C. Shannon (1950), A. Samuel (1952)
  - réduction de coût par élagage, McCarthy (1956)

- Mais il y a d'importantes différences avec un problème standard de recherche

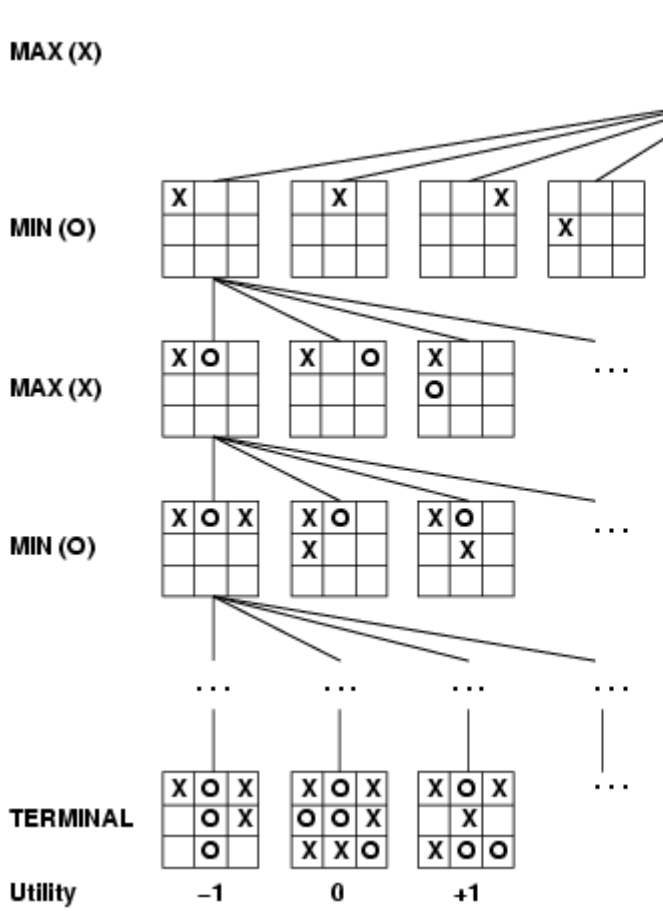
Jeux vs. problèmes de recherche

- Les mouvements de l'adversaire ne sont pas toujours prévisibles
  - il faut donc être capable de prendre en compte toutes les situations

- L'espace de recherche est généralement très vaste:
  exemple: pour le jeu d'échecs
  - nombre de choix par coup: 35 (facteur de branchement)
  - nombre moyen de coups par jeu: 50 (par joueur)
  - nombre total d'états: 35^{100}
  - nombre de noeuds de l'arbre: $\sim 10^{120}$ (pour le jeu de dames $\sim 10^{40}$)
    - Il faudrait 10^{21} siècles pour produire l'arbre complet en générant un nœud en 1/3 de nanoseconde.

- Il est donc impossible d'explorer tout l'espace de recherche pour trouver le meilleur mouvement à effectuer

Arbre de jeu (2-joueurs)



Max et Min sont les 2 joueurs. Un gain pour Max est une perte pour Min et vice-versa.
+1 pour un gain, -1 pour une perte, 0 pour un nul.

La somme des valeurs de la fonction d'évaluation pour les 2 joueurs est nulle à la fin du jeu.

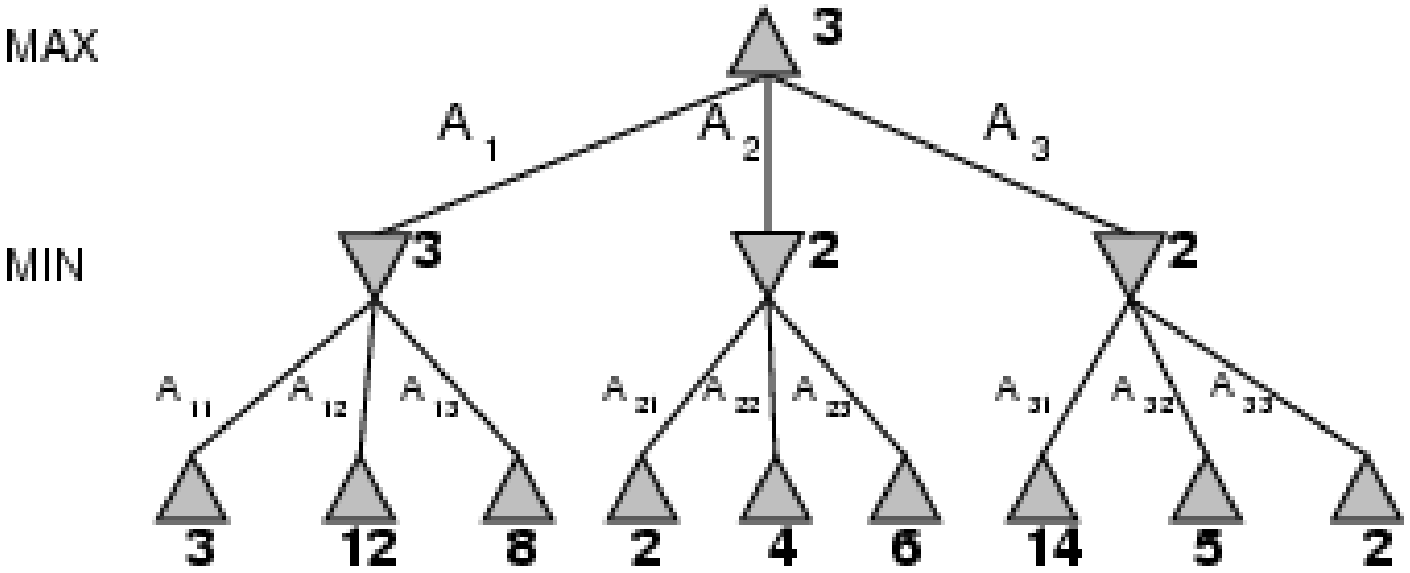
Il faut disposer d'une fonction d'évaluation statique capable de mesurer la qualité d'une configuration par rapport à un joueur (gén. Max)
car il n'est pas possible de produire tout l'arbre de recherche jusqu'à la fin du jeu, càd au moment où la décision gain/perte/nul est claire.

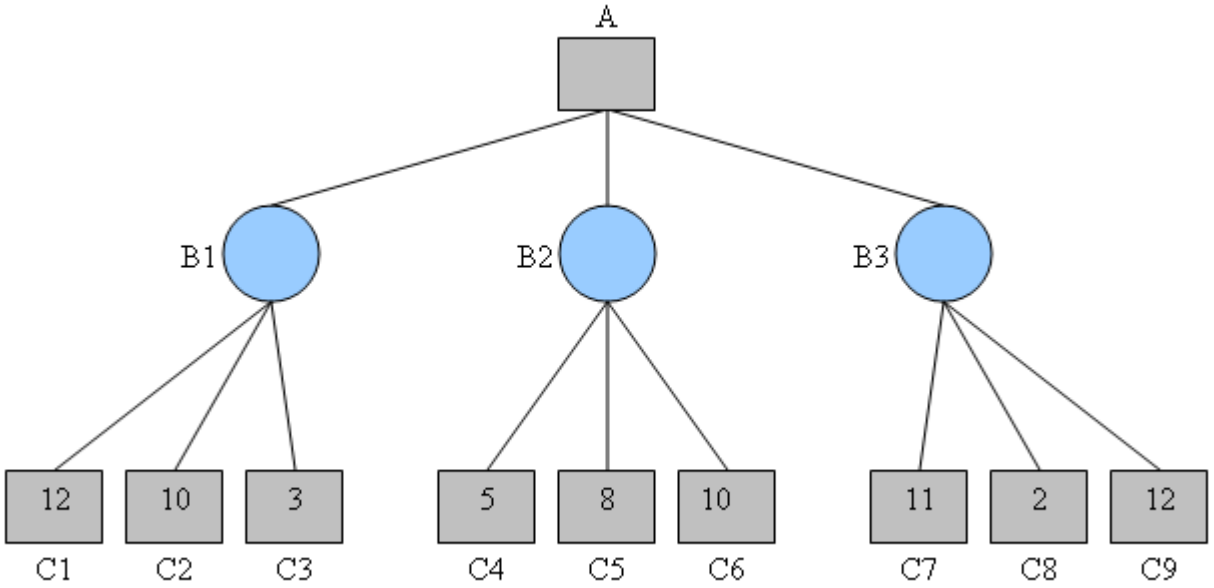
Minimax

- **Principe:** maximiser la valeur d'utilité pour ***Max*** avec l'hypothèse que ***Min*** joue parfaitement pour la minimiser,
  - étendre l'arbre de jeu
  - calculer la valeur de la **fonction de gain pour chaque noeud terminal**
  - propager ces valeurs aux noeuds non-terminaux
    - la valeur minimum (**adversaire**) aux noeuds MIN
    - la valeur maximum (**joueur**) aux noeuds MAX
- Ainsi, on visite l'arbre de jeu pour faire remonter à la racine une valeur (appelée « **valeur du jeu** ») qui est calculée récursivement de la façon suivante :
  - minimax(p) = f(p) si p est une feuille de l'arbre où f est une fonction d'évaluation de la position du jeu
  - minimax(p) = MAX(minimax(O1), …, minimax(On)) si p est un nœud Joueur avec O1…On comme fils
  - minimax(p) = MIN(minimax(O1), …, minimax(On)) si p est un nœud Joueur avec O1…On comme fils

Minimax

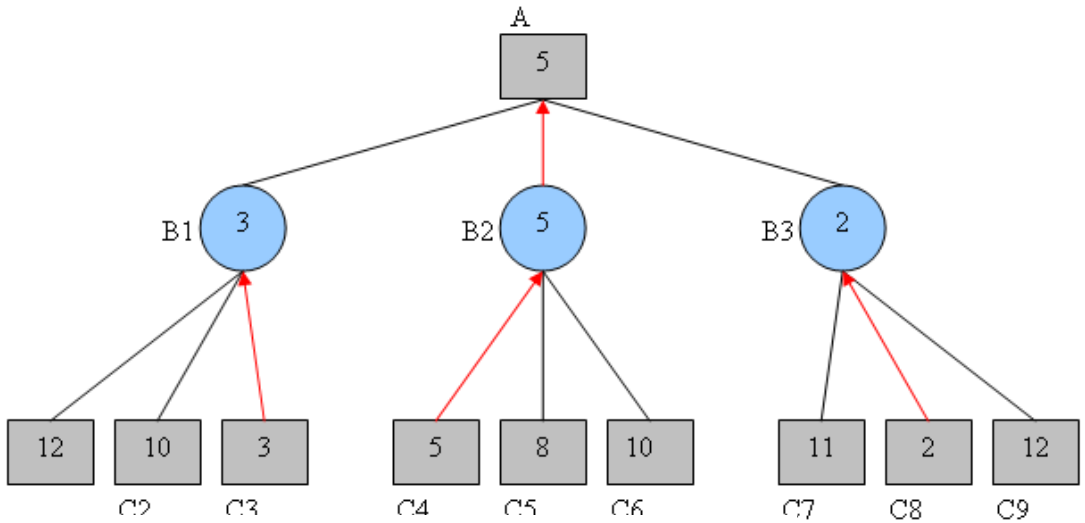
- Exemples

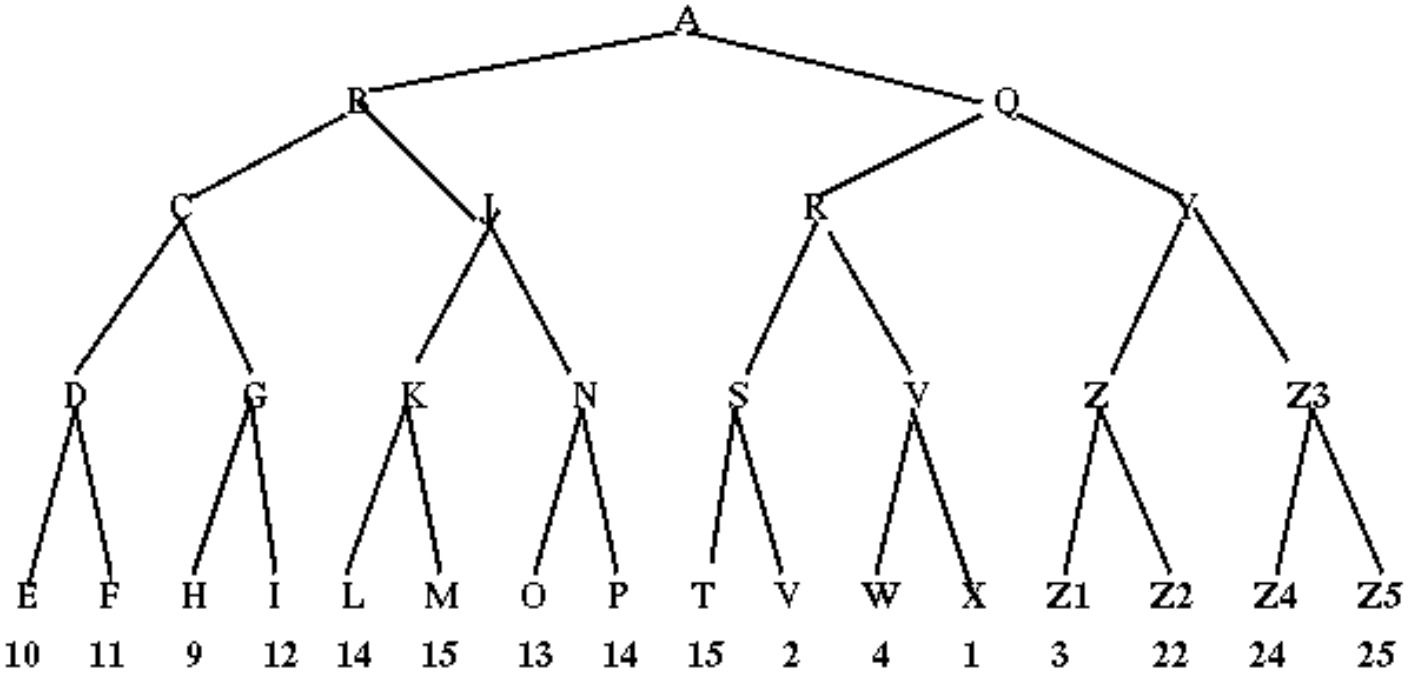




Minimax

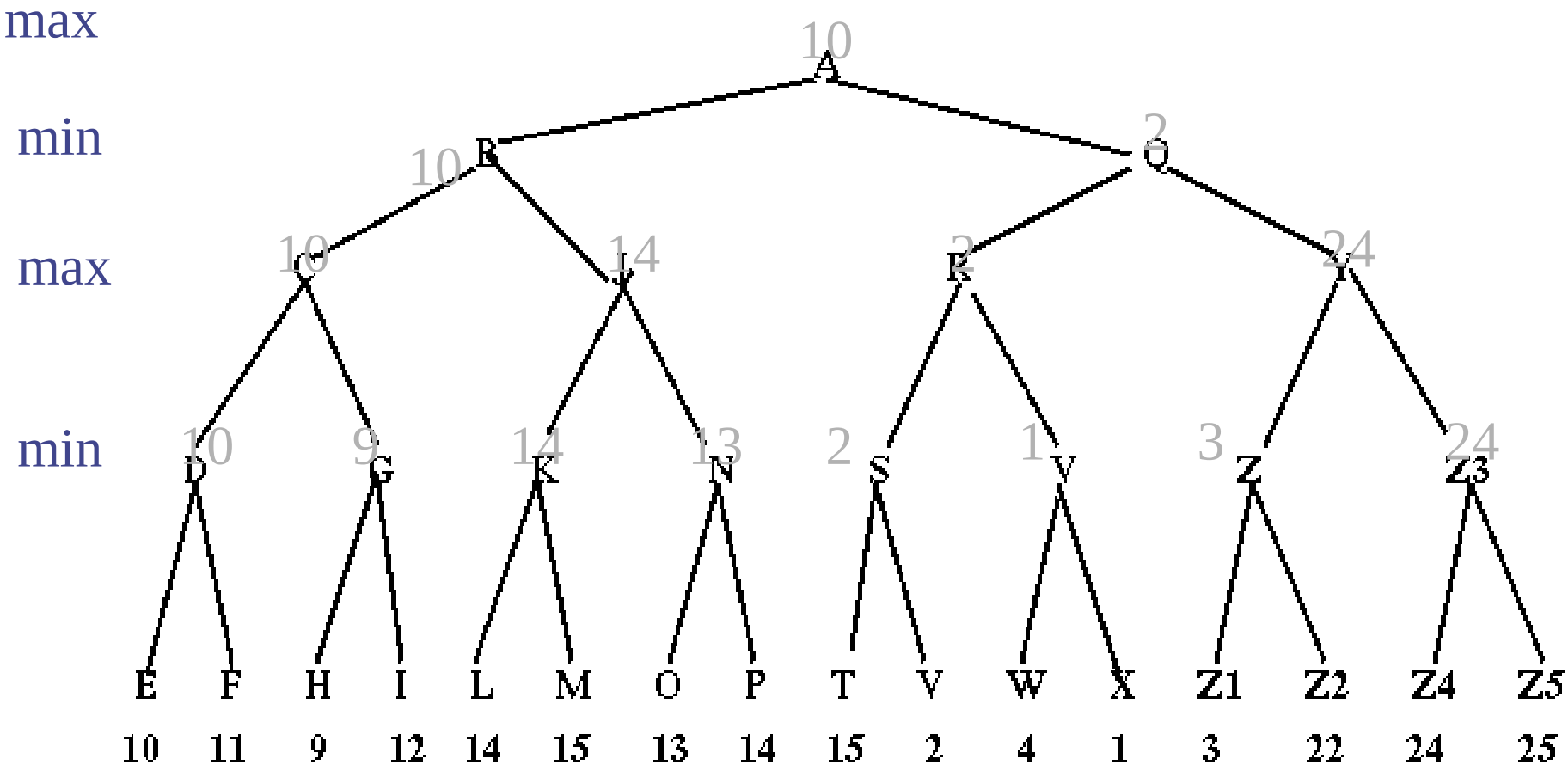
- Exemples





Minimax

- Exemples



Propriétés de minimax

- Complète?
  - Oui (si l'arbre de jeu est fini)
- Optimal?
  - Oui (si l'adversaire l'est aussi)
- Temps ?
  - O(bm)
- Espace ?
  - O(bm) (exploration en profondeur)
- Pour les jeux d'échec, b ≈ 35, m ≈100 → l'algorithme *MiniMax* est totalement impraticable (en temps) !!!

Méthodes de réduction d'espace

Même pour le jeu de Tic-Tac-To la taille de l'espace de recherche est grande:
3^9 = 19'683 noeuds pour un damier entièrement occupé !

Pour des jeux non-triviaux les coups doivent pouvoir être déterminés sans pour autant devoir générer tout l'arbre de jeu

Il faut alors pouvoir évaluer des noeuds non-terminaux

2 solutions:

- *MiniMax* avec profondeur limitée
- élagage α-β

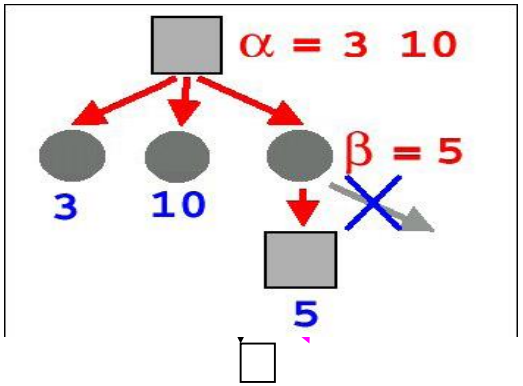
Élagage α-β

- Principe
  - étendre l'arbre de jeu jusqu'à une profondeur h par recherche en profondeur
  - ne plus générer les successeurs d'un noeud dès qu'il est évident que ce noeud ne sera pas choisi (compte tenu des noeuds déjà examinés)
  - chaque noeud Max garde la trace d'une α-valeur = valeur de son meilleur successeur trouvé jusqu'ici
  - chaque noeud Min garde la trace d'une β-valeur = valeur de son plus mauvais successeur trouvé jusqu'ici
  - valeurs initiales: $\alpha = -\infty$ $\beta = +\infty$

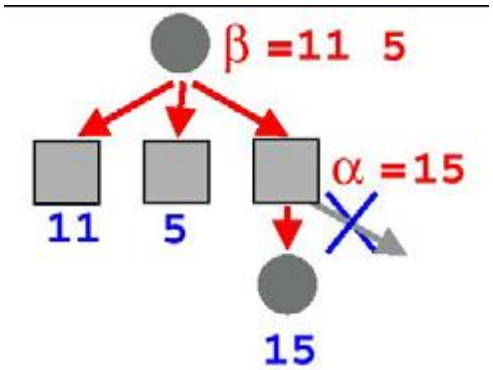
Élagage α-β

- Deux règles:

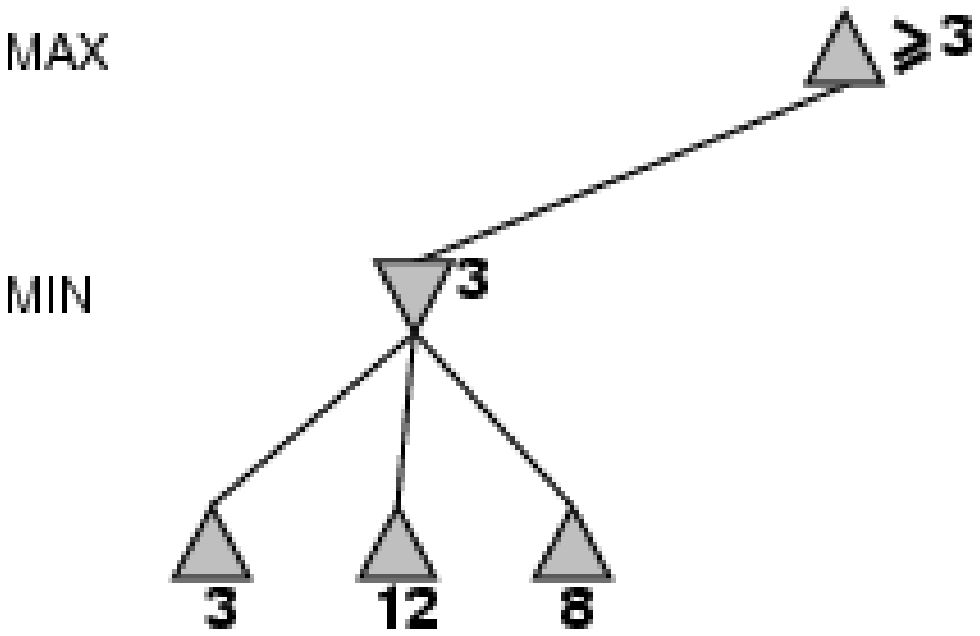
1. Interrompre la recherche d'un noeud Max si son α-valeur ≥ β-valeur de son noeud-parent



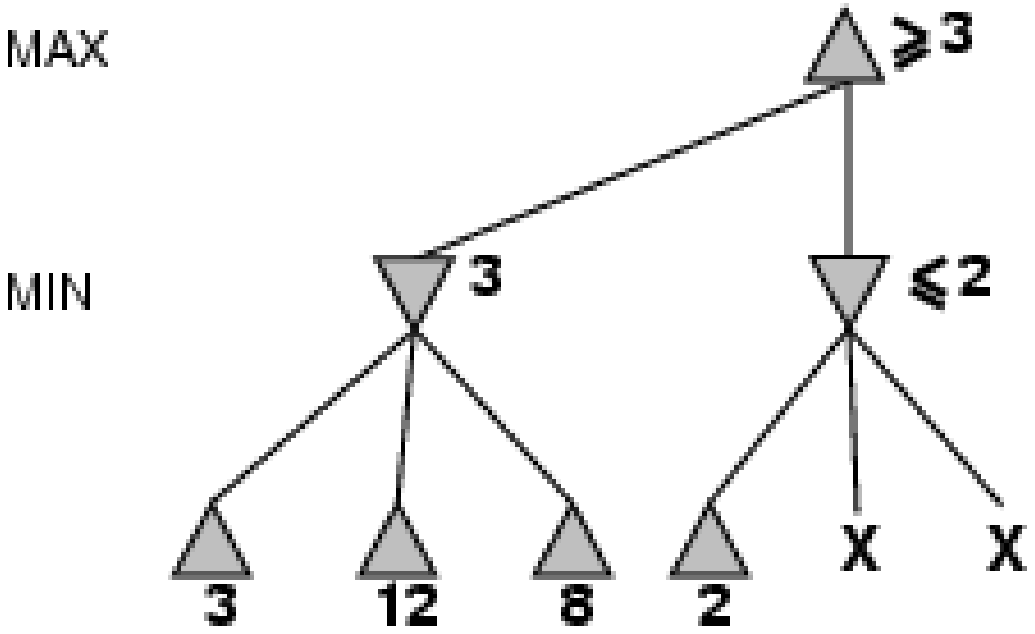
2. Interrompre la recherche d'un noeud Min si sa β-valeur ≤ α-valeur de son noeud-parent



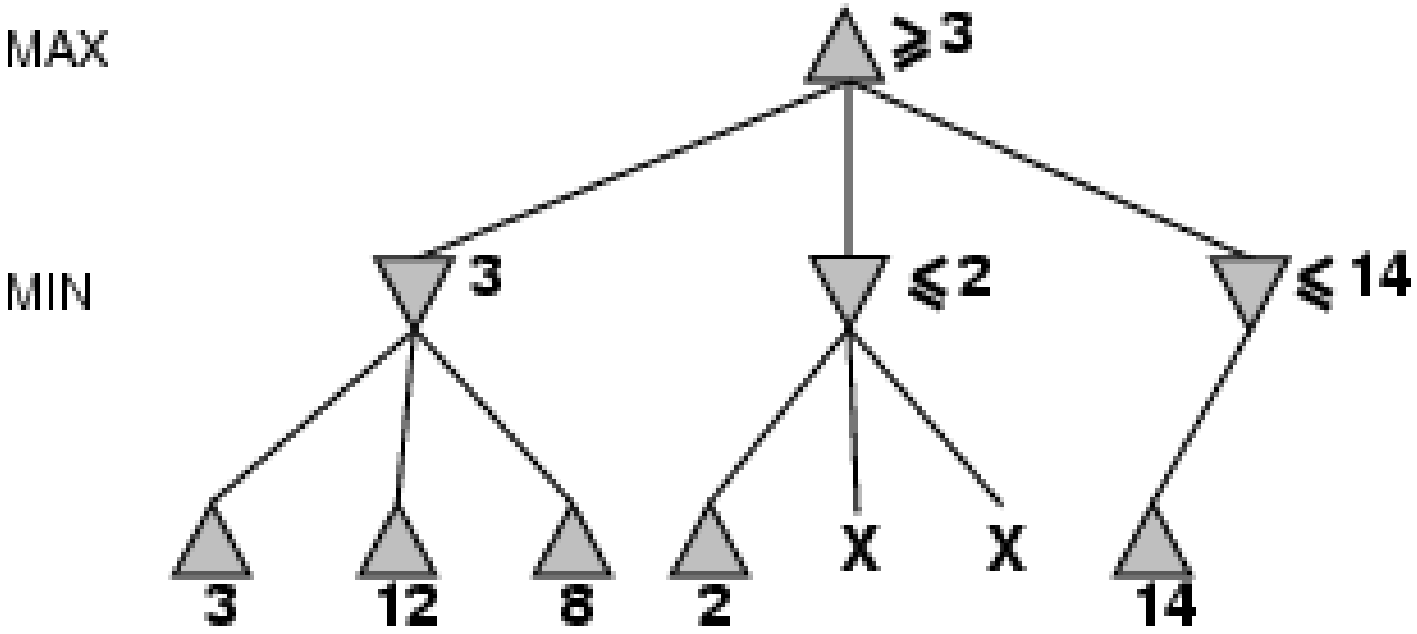
Élagage α-β : exemple



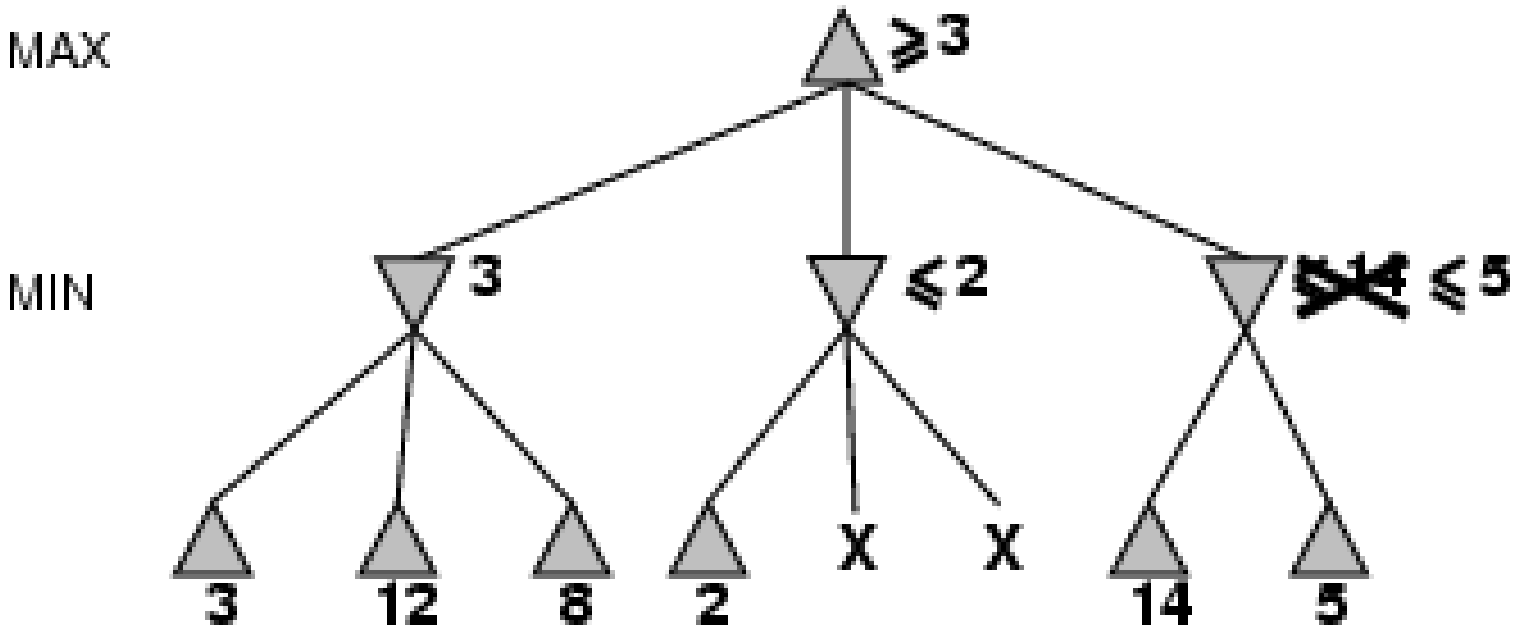
Élagage α-β : exemple



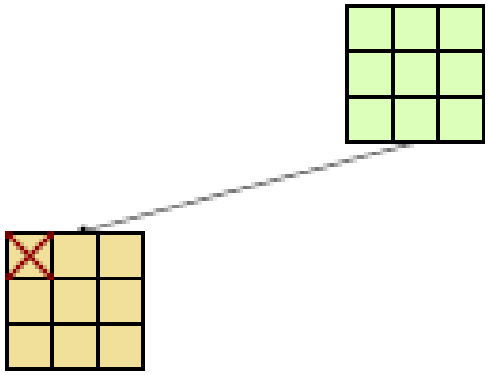
Élagage α-β : exemple



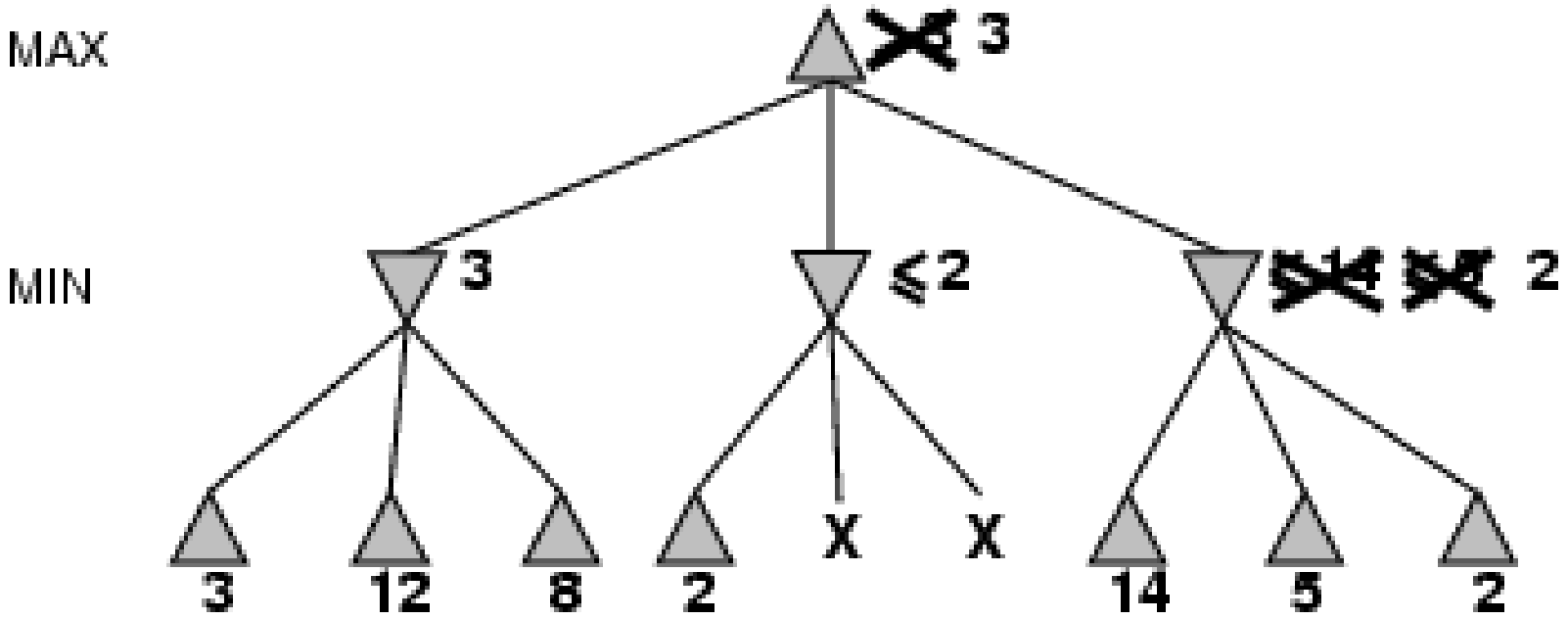
Élagage α-β : exemple



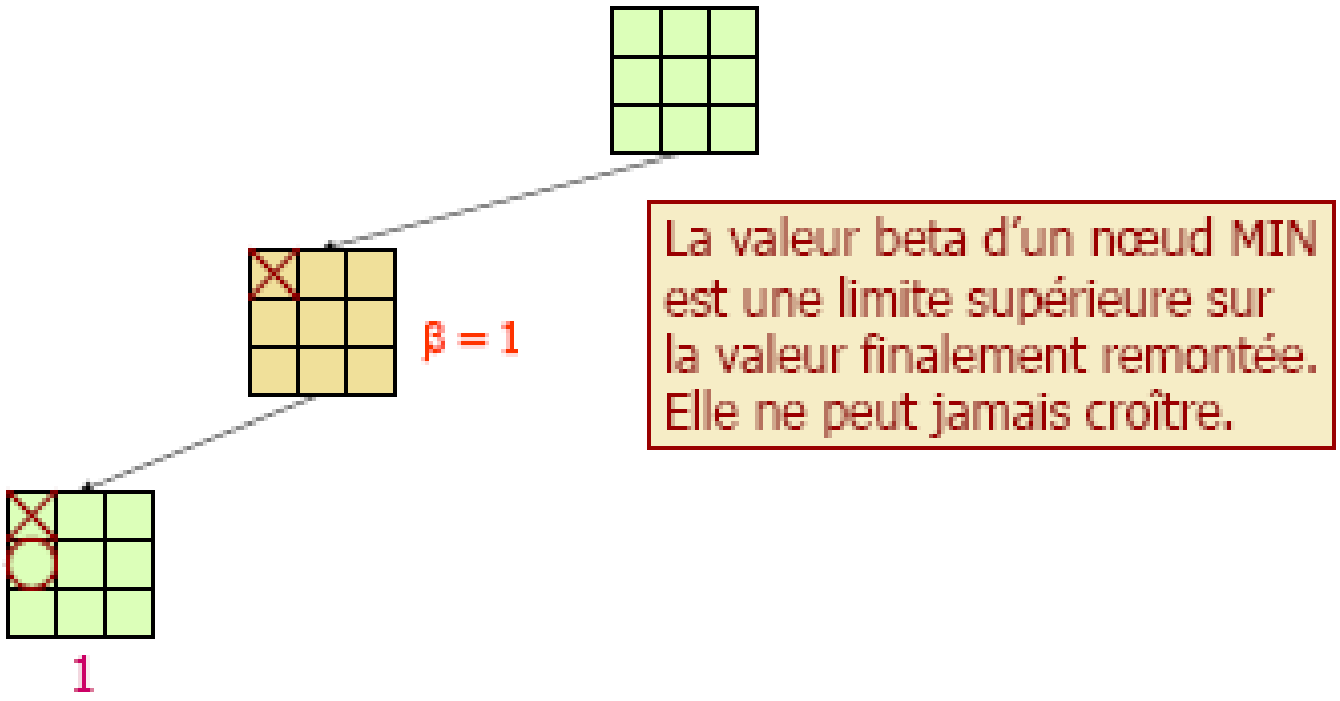
Exemple2



Élagage α-β : exemple

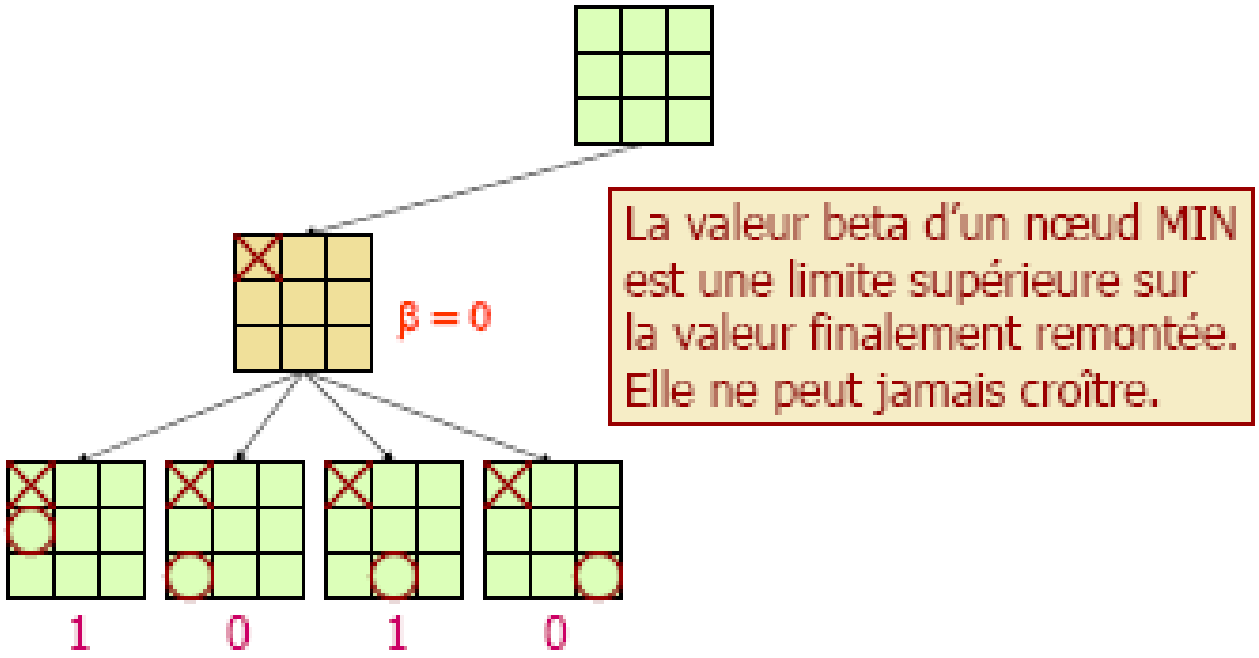


Exemple2



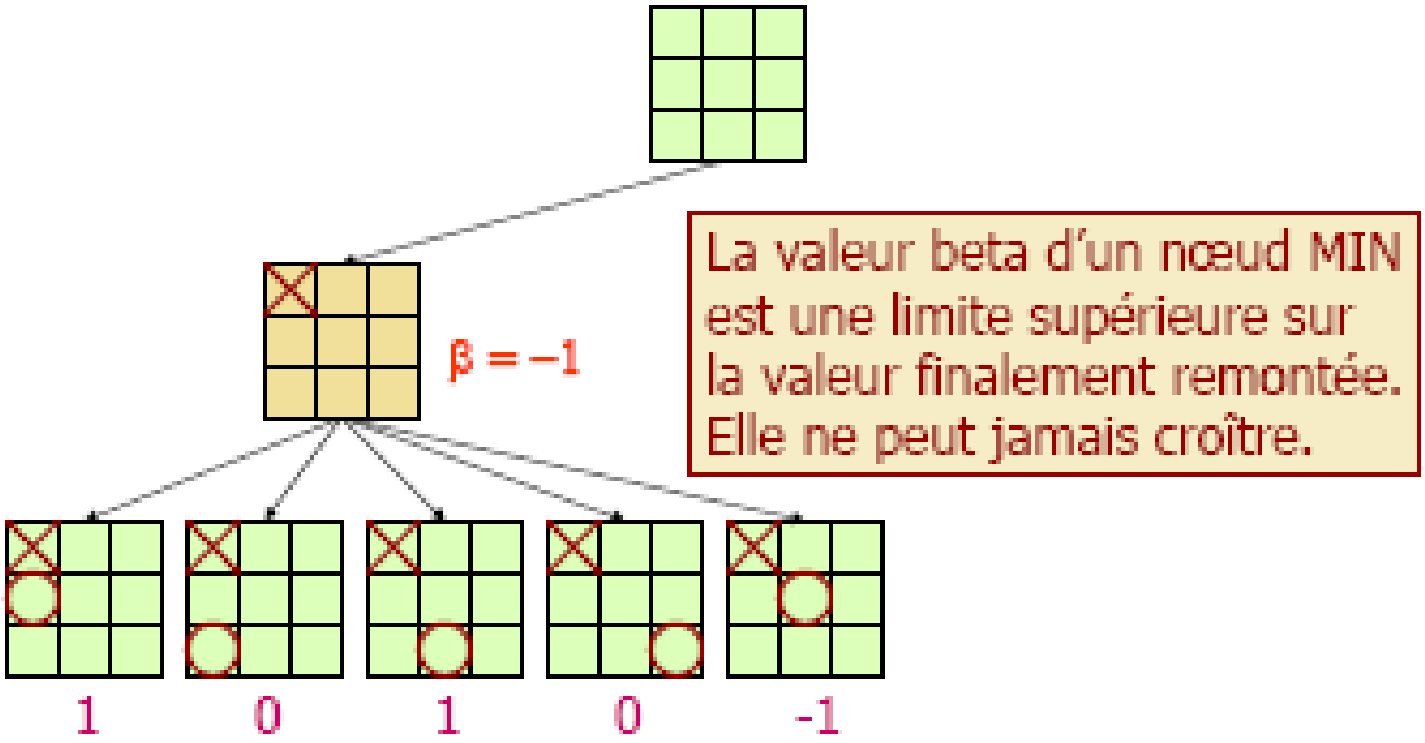
La valeur beta d'un nœud MIN est une limite supérieure sur la valeur finalement remontée. Elle ne peut jamais croître.

Exemple2

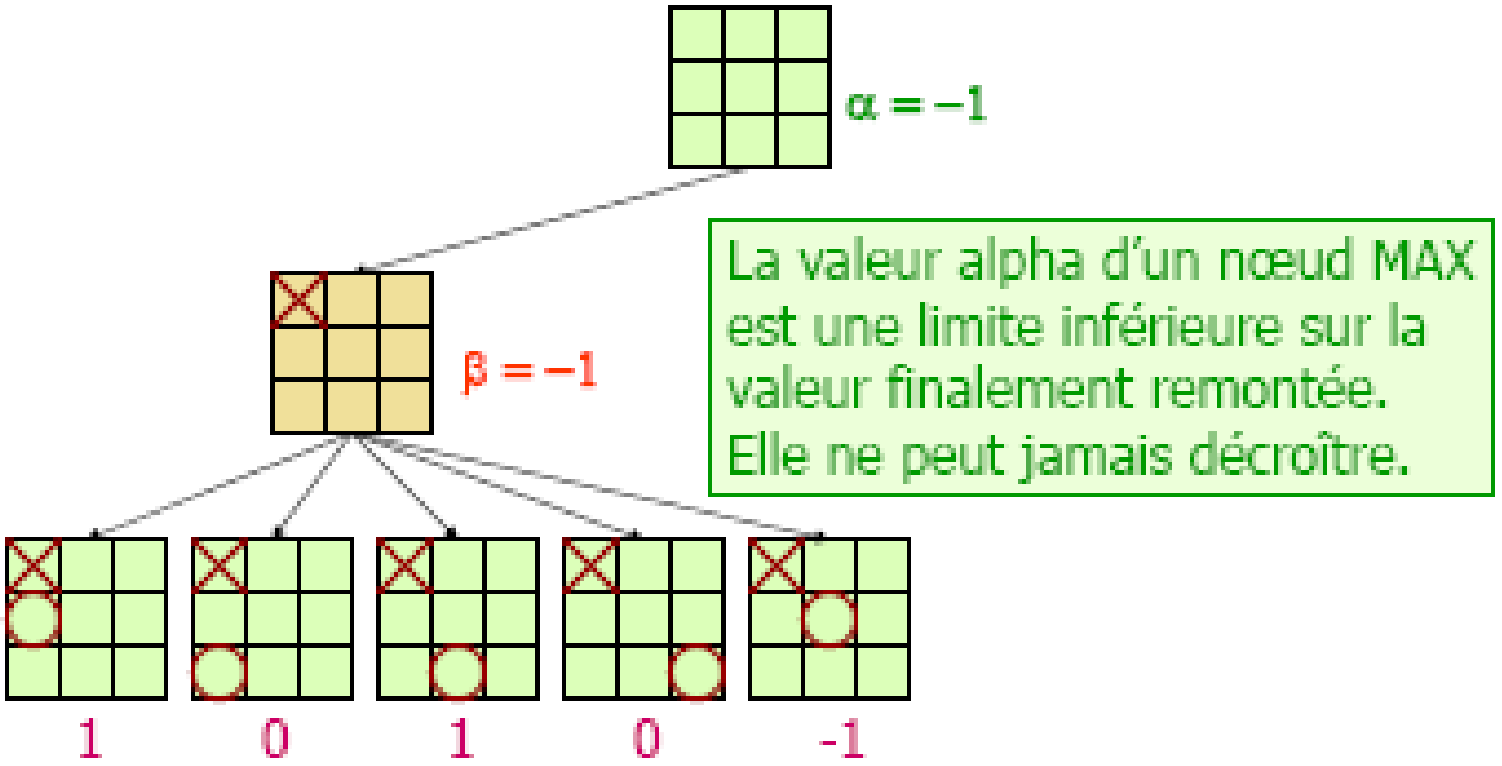


La valeur beta d'un nœud MIN est une limite supérieure sur la valeur finalement remontée. Elle ne peut jamais croître.

Exemple2

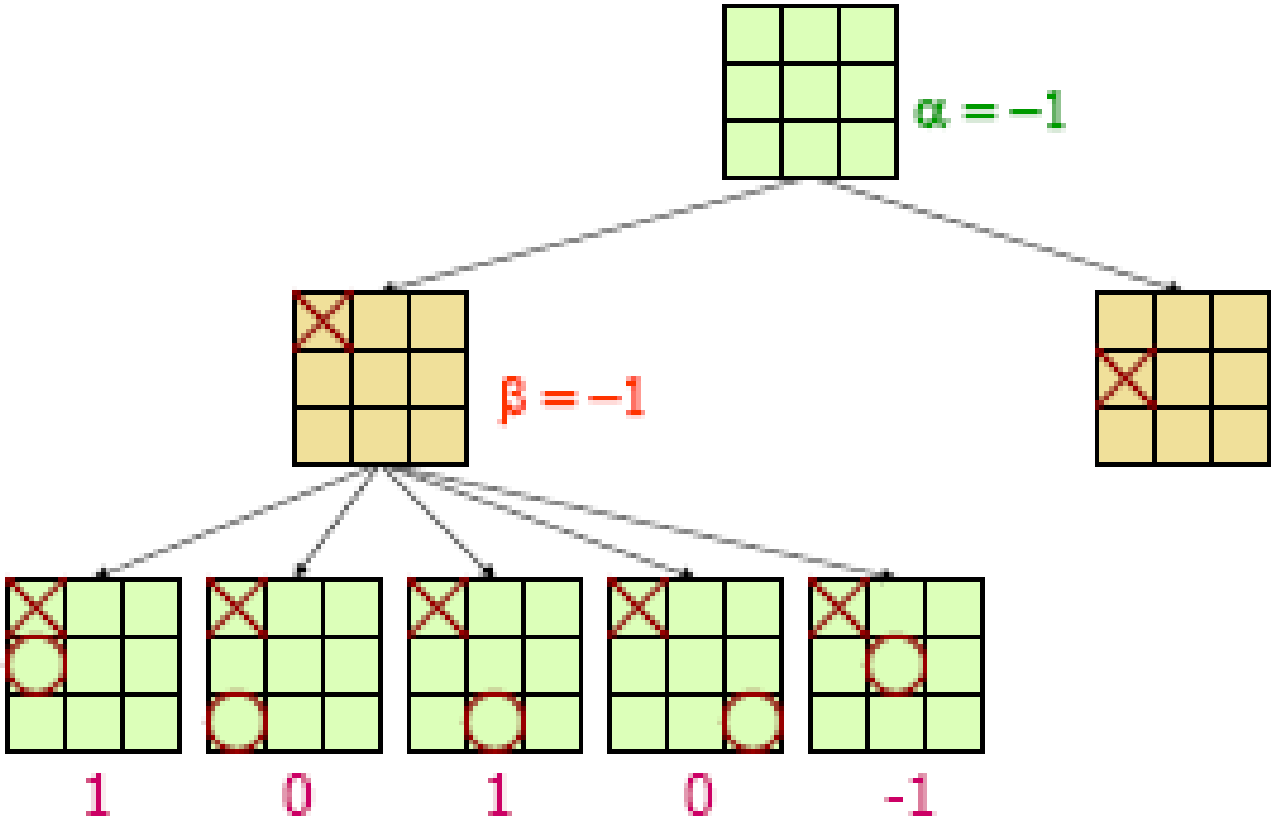

β = –1
La valeur beta d'un nœud MIN est une limite supérieure sur la valeur finalement remontée. Elle ne peut jamais croître.
1
0
1
0
-1

Exemple2

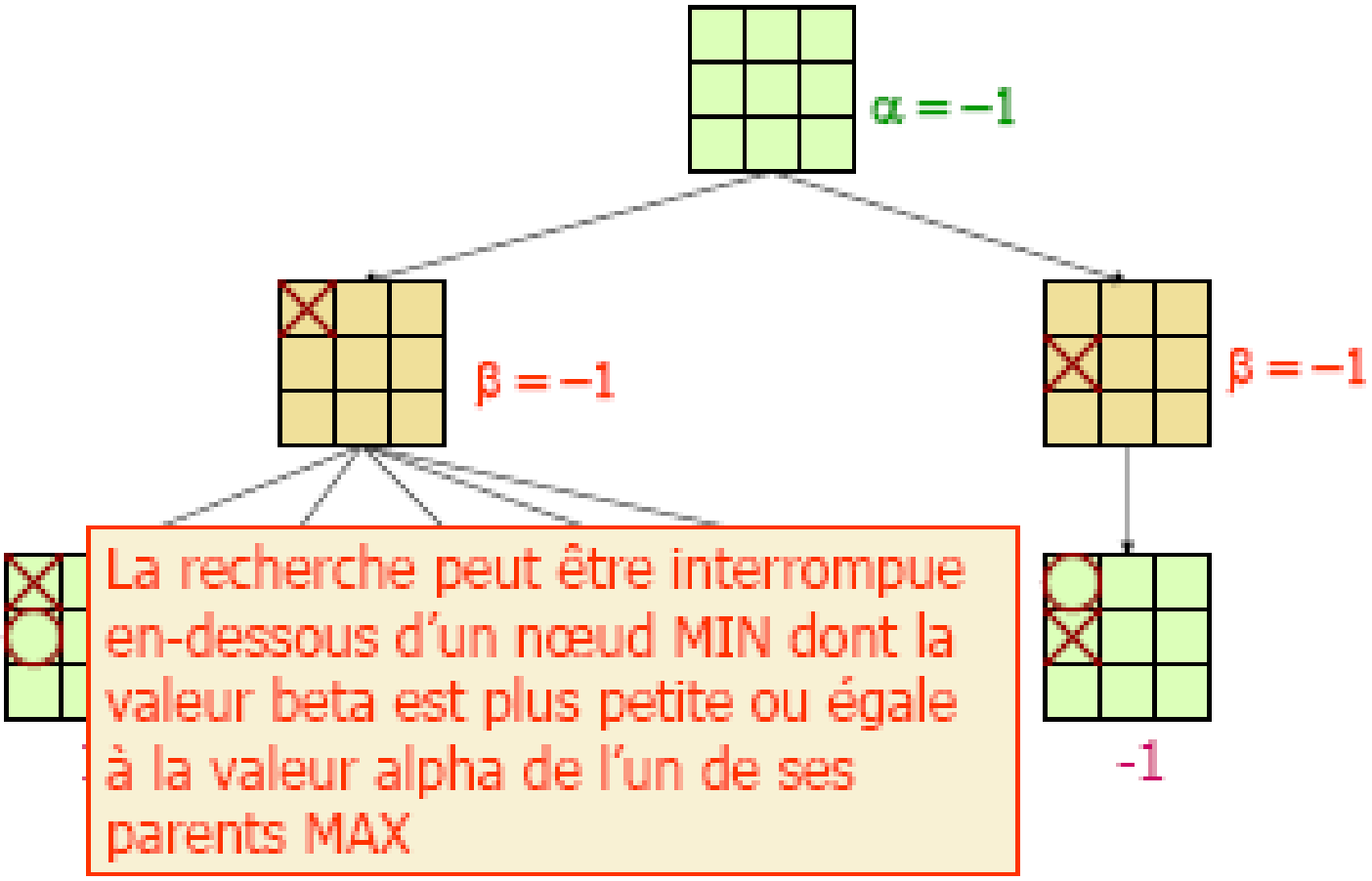


La valeur alpha d'un nœud MAX est une limite inférieure sur la valeur finalement remontée. Elle ne peut jamais décroître.

Exemple2

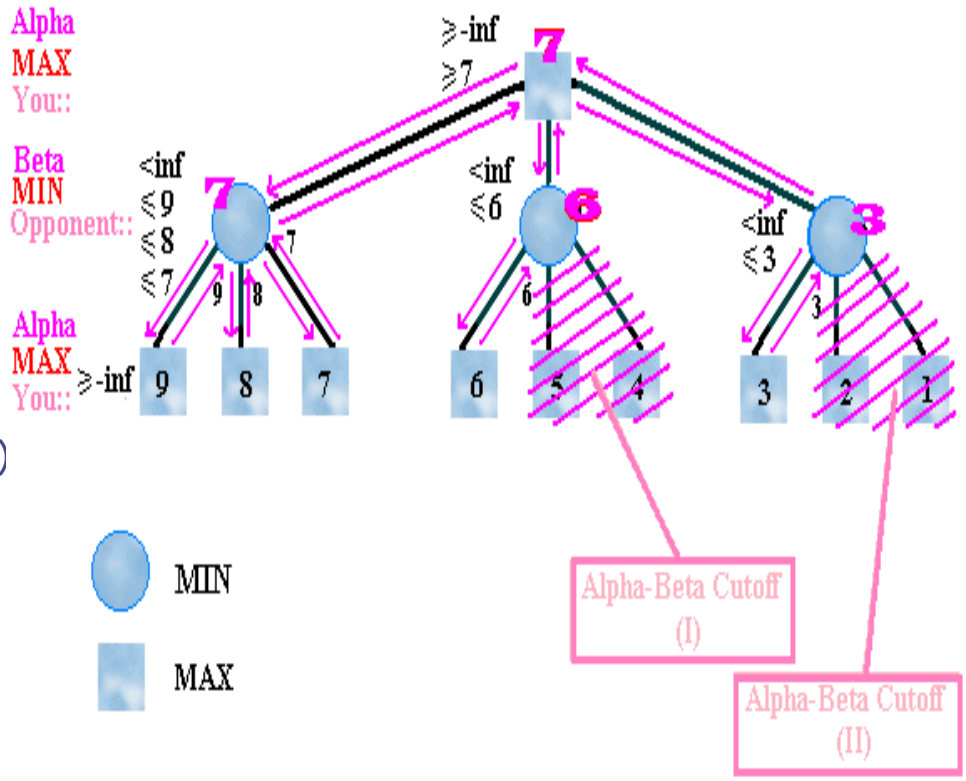


Exemple2


α = −1
β = −1
β = −1
La recherche peut être interrompue en-dessous d'un nœud MIN dont la valeur beta est plus petite ou égale à la valeur alpha de l'un de ses parents MAX
-1

Algorithme

```
fonction ALPHABETA(P, alpha, beta) /* alpha est toujours inférieur à beta */
   si P est une feuille alors
      retourner la valeur de P
   sinon
      si P est un nœud Min alors
         Val = infini
         pour tout fils Pi de P faire
            Val = Min(Val, ALPHABETA(Pi, alpha, beta))
            si alpha ≥ Val alors  /* coupure alpha */
               retourner Val
            beta = Min(beta, Val)
         finpour
      sinon
         Val = -infini
         pour tout fils Pi de P faire
            Val = Max(Val, ALPHABETA(Pi, alpha, beta))
            si Val ≥ beta alors /* coupure beta */
               retourner Val
            alpha = Max(alpha, Val)
         finpour
      finsi
   retourner Val
   finsi
fin
```



Propriétés de α-β

- L'élagage n'affecte pas le résultat final,
- l'efficacité de l'élagage α - β est fonction de l'ordre d'apparition des noeuds successeurs,
- complexité en temps
  - meilleur des cas: $O(bm/2)$
    - permet de doubler la profondeur de recherche pour atteindre une prédiction sur 8 niveaux et ainsi jouer "dignement" aux échecs
  - pire des cas: identique à MiniMax
  - cas moyen: $O((b/\log b)m)$ [Knuth&Moore 75

Minimax & Alpha-Beta: Récapitulons en revistant le tout par une VIDEO

État de l'art

Jeu de dames: Tinsley vs. Chinook

Nom: Marion Tinsley
Profession: professeur de mathématiques
Hobby: le jeu de dames
Record: en plus de 42 ans n'a perdu que 3 (!) parties

Mr. Tinsley encaisse sa 4ème et 5ème défaite contre Chinook

Nom: Chinook
Record: premier programme informatique ayant gagné un championnat du monde face au champion en titre en 1994, invaincu depuis! Méthode: base de fins de parties de 8 pièces (ou moins), pour un total de 444 milliards de positions

Jeu d'échecs

Jeu qui a reçu la plus grande attention au début les progrès ont été très lents

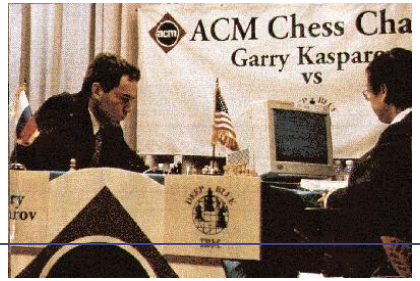
➢ 1970: 1er programme à gagner le championnat ACM d'échecs informatiques, utilisant une recherche α - β, des ouvertures classiques et des algorithmes de fins de parties infaillibles

➢1982: "Belle" est le premier ordinateur spécialisé dans le jeu d'échecs capable d'explorer quelques millions de combinaisons par coups

➢1985: "Hitech" se classe parmi les 800 meilleurs joueurs du monde, il est capable d'explorer plus de 10 millions de combinaisons par coups

➢1997: "*Deep Blue*" bat G. Kasparov. Il effectue une recherche sur 14 niveaux et explore plus de 1 milliard de combinaisons par coup à raison de plus de 200 millions de positions par seconde, utilise une fonction d'évaluation très sophistiquée et des méthodes non divulguées pour étendre certaines recherches à plus de 40 niveaux.

Kasparov vs. Deep Blue



Kasparov		Deep Blue
5'10"	Taille	6' 5"
176 lbs	Poids	2,400 lbs
34 ans	Age	4 ans
50 milliards	Ordinateur	512 processeurs
2 pos/sec	Vitesse	200,000,000 pos/sec
Extensive	Connaissance	Primitive
Electrique/chimique	Source d'énergie	Électrique
Démesuré	Ego	Aucun

1997: Deep Blue gagne par 3 victoires, 1 défaite, et 2 nuls

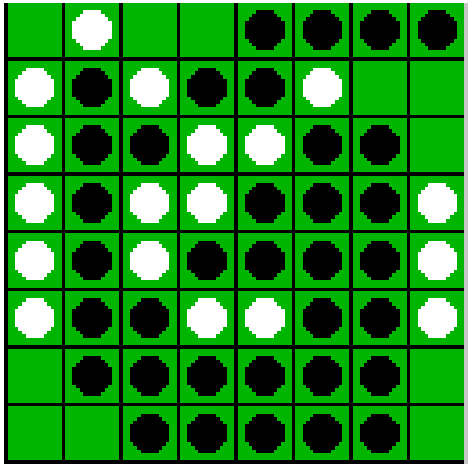
Kasparov vs. Deep Blue Junior

Deep Junior

8 CPU, 8 GB RAM, Win 2000
2,000,000 pos/sec
Disponible à $100

2 Août 2003: Égalité (3-3)!

Othello: Murakami vs. Logistello



Takeshi Murakami
Champion du monde d'Othello

1997: Le logiciel Logistello défait Murakami par 6 jeux à 0