

Résolution des problèmes

Rym Guibadj

LISIC, ULCO, EILCO

Objectifs

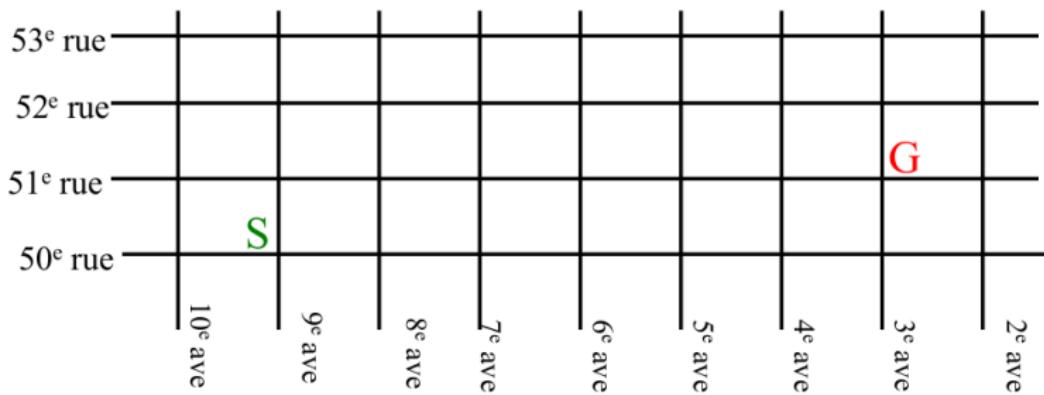
Agents guidés par un but

- Trouver la bonne succession d'étapes pour atteindre un but donné
- Résolution de problème par recherche dans un graphe
- Algorithmes d'exploration

Exemples

- Planifier un trajet
- Résoudre un casse-tête
- Jouer
- etc

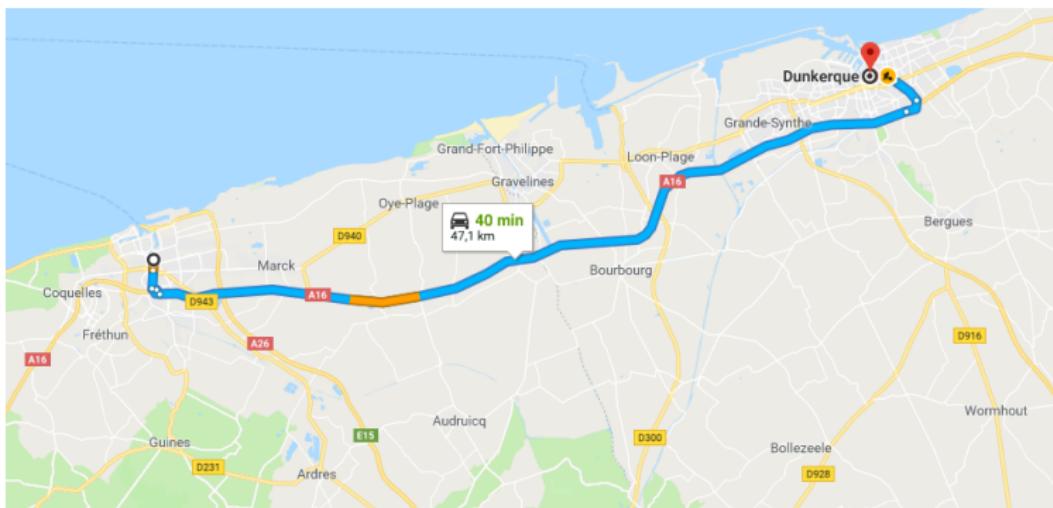
Exemple : trouver un chemin dans la ville



Exemple : trouver un chemin dans la ville

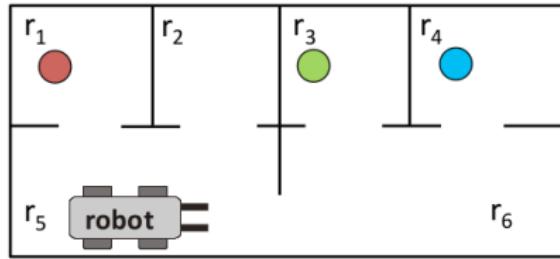


Exemple : Google Maps

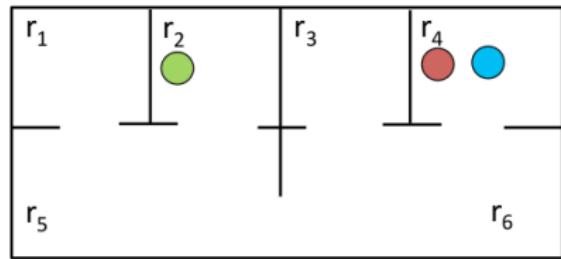


Exemple : livrer des colis

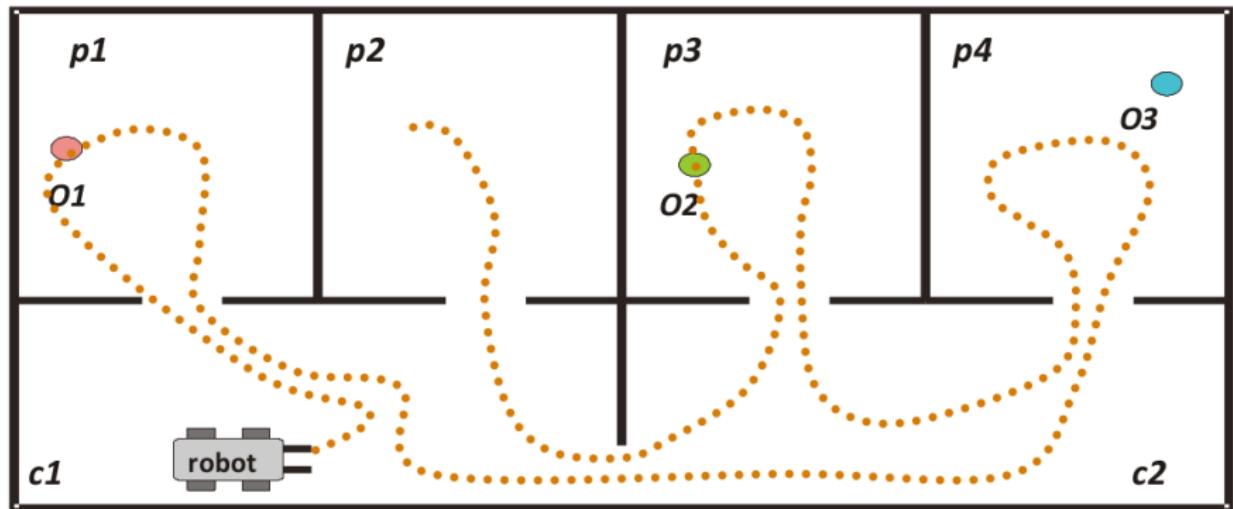
Etat initial



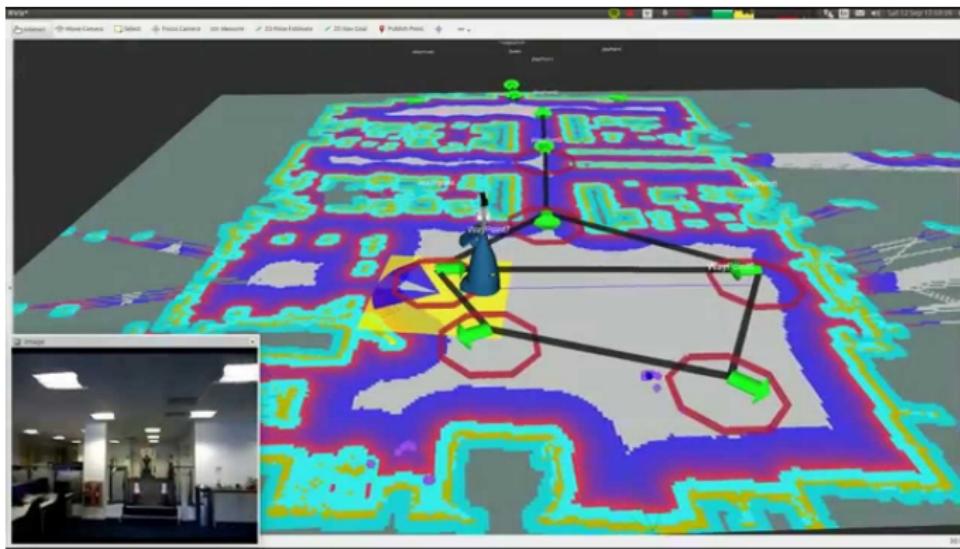
But



Exemple : livrer des colis



Exemple : navigation d'un robot



Exemple : N-Puzzle

Etat initial

2	8	3
1	6	4
7		5

But

1	2	3
8		4
7	6	5

Nord

2	8	3
1		4
7	6	5

Nord

2		3
1	8	4
7	6	5

Ouest

	2	3
1	8	4
7	6	5

Sud

1	2	3
	8	4
7	6	5

Est

1	2	3
8		4
7	6	5

Formulation du problème

Correctement formuler le problème revient à définir :

- L'état initial du problème
- Une description des actions possibles
- Un Modèle de transition
 - une fonction qui renvoie l'état résultant de l'execution d'une action dans un état
 - Successeur : désigne tout état accessible à partir d'un état donné avec une seule action
- Un test de but
 - détermine si un état est un état but
- Le coût d'une action

Résolution du problème

- La résolution de plusieurs problèmes peut être faite par une recherche dans un graphe.
- Chaque noeud correspond à un état de l'environnement.
- Chaque chemin à travers un graphe représente alors une suite d'actions prises par l'agent.
- Pour résoudre notre problème, suffit de chercher le chemin qui satisfait le mieux notre mesure de performance.

Algorithmes de recherche dans un graphe

- Entrées
 - un noeud initial
 - une fonction $goal(n)$ qui retourne *true* si le but est atteint
 - une fonction $successors(n)$ qui retourne les noeuds successeurs de n
 - une fonction $cost(n, n')$ strictement positive, qui retourne le coût de passer de n à n'

- Sorties
 - un chemin dans un graphe (séquence noeuds / arrêtes)
 - le coût d'un chemin est la somme des coûts des arrêtes dans le graphe

- Enjeux
 - trouver un chemin solution, ou
 - trouver un chemin optimal, ou
 - trouver rapidement un chemin

Algorithmes de recherche dans un graphe

- Recherche simple
 - en largeur
 - en profondeur
 - en profondeur limitée
 - ...
- Recherche heuristique
 - meilleur d'abord
 - gourmande
 - A*
 - ...

Evaluation des méthodes de recherche

- **Complétude** : L'algorithme garantit-il de trouver une solution lorsqu'il en existe une ?
- **Optimalité** : L'algorithme garantit-il de trouver la meilleure solution ?
- **Complexité en temps** : Quel est le temps nécessaire pour trouver une solution ?
- **Complexité en espace mémoire** : Quelle est la mémoire nécessaire pour effectuer l'exploration ?
- **Ces critères dépendent de :**
 - Facteur de branchements maximum de l'arbre de recherche
 - Profondeur à laquelle se trouve la meilleure solution
 - Profondeur maximale de l'espace de recherche

Algorithme de recherche

Algorithm 1 Recherche générale

Argument : problème

Sortie : échec ou une solution

Initialiser l'arbre de recherche du problème par l'état initial

Boucler

Si il n'y a pas de candidat (noeud) possible **alors**

Retourner échec

Sinon

 Choisir un candidat

Si le candidat satisfait le but **alors**

Retourner solution

Sinon

 étendre le noeud

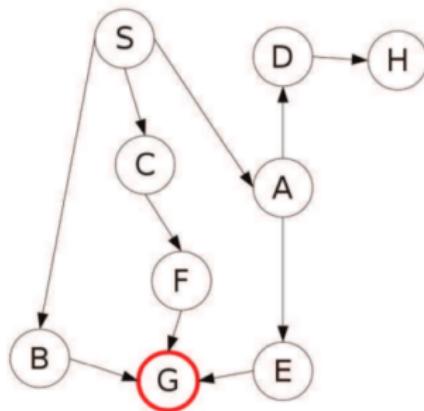
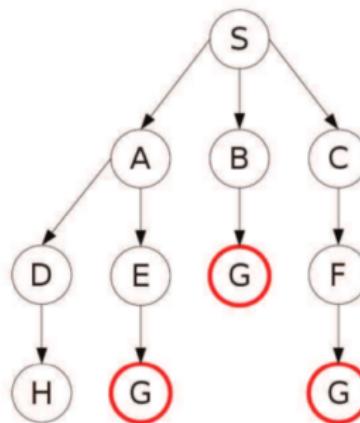
 ajouter les noeuds fils dans l'arbre

Fin si

Fin si

Fin Boucler

Exemple

Espace d'états**Espace de recherche**

Recherche en largeur

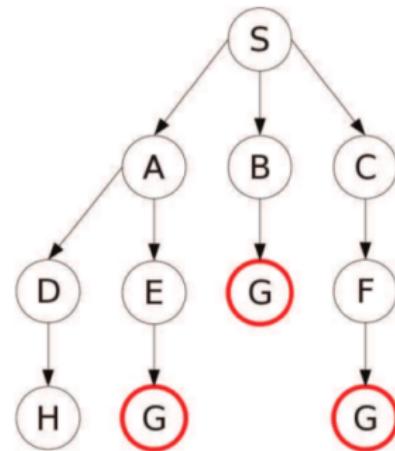
Idée

Priorité aux voisins

Implémentation

Utiliser une file pour stocker les nouveaux noeuds

Noeuds étudiés (FERMES)	Noeuds à étudier (OUVERTS)
\emptyset	{S}
{S}	{C, B, A}
{S, A}	{E, D, C, B}
{S, A, B}	{G, E, D, C}
{S, A, B, C}	{F, G, E, D}
{S, A, B, C, D}	{H, F, G, E}
{S, A, B, C, D, E}	{H, F, G}
{S, A, B, C, D, E, G}	{H, F}



Recherche en profondeur

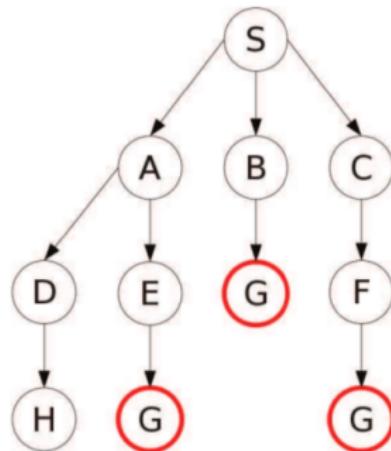
Idée

Priorité aux fils

Implémentation

Utiliser une pile pour stocker les nouveaux noeuds

Noeuds étudiés (FERMÉS)	Noeuds à étudier (OUVERTS)
\emptyset	{S}
{S}	{C, B, A}
{S, C}	{F, B, A}
{S, C, F}	{G, B, A}
{S, C, F, G}	{B, A}



Recherche en profondeur "limitée"

Idée

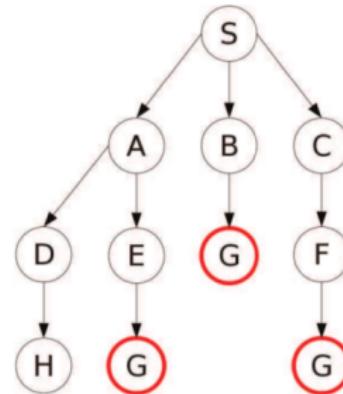
Priorité aux fils avec une profondeur limitée (L)

Implémentation

Utiliser une pile pour stocker les nouveaux noeuds

Exemple avec $L = 2$

Noeuds étudiés (FERMES)	Noeuds à étudier (OUVERTS)
\emptyset	{ S }
{ S }	{ C, B, A }
{ S, C }	{ B, A }
{ S, C, B }	{ A }
{ S, C, B, A }	{ \emptyset }



Recherche par approfondissement itératif

Idée

Profondeur limitée avec L allant de 1 à $+\infty$

Implémentation

Utiliser une pile pour stocker les nouveaux noeuds

- Le problème de la recherche en profondeur limitée est de fixer la bonne valeur de limite (L)
- L'approfondissement itératif combine les avantages de la recherche en largeur et en profondeur :
 - Optimal et complet comme la recherche en largeur
 - Economie en espace comme la recherche en profondeur
- ⇒ C'est l'algorithme de choix si l'espace de recherche est grand et si la profondeur est inconnue

Recherche simple "bi-directionnelle"

Idée

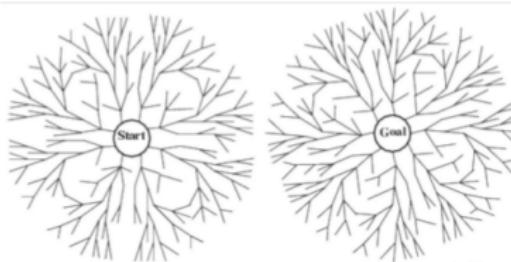
Effectuer une recherche par chaînage avant à partir de l'état initial et effectuer une recherche par chaînage arrière à partir de l'état final

Implémentation

Effectuer les deux recherches et stopper lorsqu'elles se rencontrent

Condition

Il faut disposer de l'ensemble des actions inverses



Comparaison de la recherche simple

Critères	Largeur	Coût Uniforme	Profondeur	Profondeur limitée	Profondeur itérative	Bi-directionnelle
Complétude	Oui ¹	Oui ¹	Non	Non	Oui ¹	Oui ³
Optimalité	Oui ²	Oui	Non	Non	Oui ²	Oui ^{2 3}
Temps	b^d	b^d	b^m	b^L	b^d	$b^{d/2}$
Mémoire	b^d	b^d	$b.d$	$b.L$	$b.d$	$b^{d/2}$

- b : facteur de branchement maximum de l'arbre de recherche
- d : profondeur à laquelle se trouve la meilleure solution
- m : profondeur maximum de l'espace de recherche
- L : limite de la profondeur de recherche limitée

-
1. Si b est fini
 2. Si les coûts sont identiques
 3. Si les deux directions sont des largeurs d'abord

Recherche heuristique

Algorithmes de recherche simple non satisfaisants

- Ils n'exploitent aucune information concernant la structure de l'arbre de recherche
- La plupart des problèmes réels sont susceptibles de provoquer une explosion combinatoire du nombre d'états possibles

Algorithmes de recherche heuristique

- Ils utilisent des informations heuristiques pour améliorer le processus de recherche

Une information heuristique

- Une heuristique est une règle ou une méthode qui améliore (presque) toujours un processus de décision.
- Une heuristique est spécifique à un problème
- Elle permet d'orienter la recherche à l'aide de connaissances sur le domaine



Recherche heuristique

Fonction de coût

- Egalement appelée fonction d'évaluation. Pour chaque noeud n , $f(n)$ est le coût (ou évaluation) du noeud
- Souvent non connue \Rightarrow Utilisation d'une approximation

Fonction heuristique

- $h : E \rightarrow \mathbb{R}$. (E étant l'espace d'états)
- Pour un état s , $h(s)$ est une estimation du coût du chemin de s au but

Propriétés

h doit posséder les propriétés suivantes :

- $h(s) \geq 0$, si $s \neq G$
- $h(s) = 0$, si $s = G$
- $h(s) = \infty$, si G est inatteignable depuis s



Approche meilleur d'abord

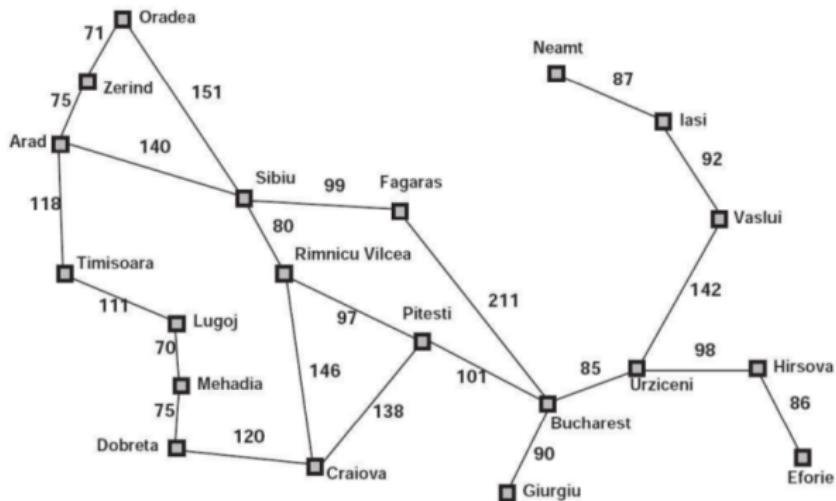
Principe

- Prendre le noeud le moins couteux en fonction de $f(n)$ à chaque étape
- On dispose pour le problème d'une fonction heuristique $h(n)$

Algorithmes

- Algorithme glouton
 - On minimise le coût du chemin vers le but à chaque étape
 - $f(n) = h(n)$
- Algorithme A^*
 - Intégration du coût du chemin déjà trouvé dans l'estimation
 - $f(n) = h(n) + g(n)$
 - avec $g(n)$ le coût entre le noeud de départ et n

Exemple de problème : trouver un chemin



Straight-line distance to Bucharest

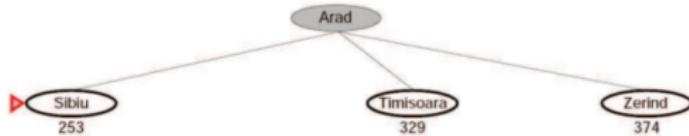
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- Trouver un "bon" chemin entre Arad et Bucarest
- Heuristique : $h(n)$: distance à vol d'oiseau de n ? Bucarest

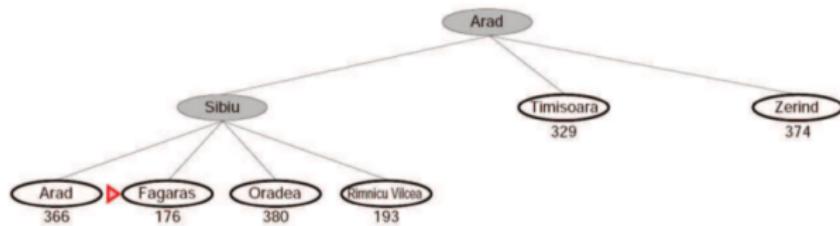
Stratégie gloutonne



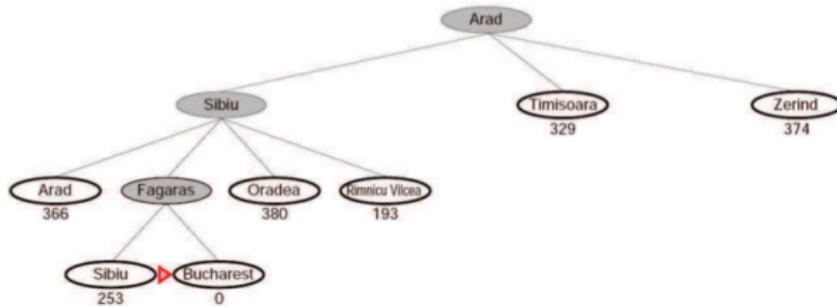
Stratégie gloutonne



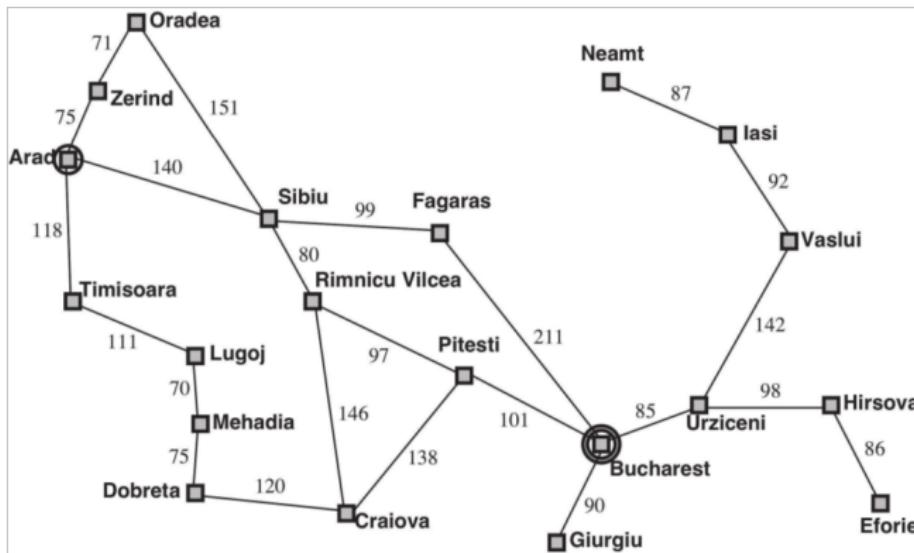
Stratégie gloutonne



Stratégie gloutonne



Stratégie gloutonne



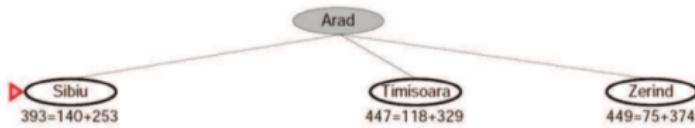
Solution trouvée

- Chemin : Arad - Sibiu - Faragas - Bucarest
- Coût : $140 + 99 + 211 = 450$

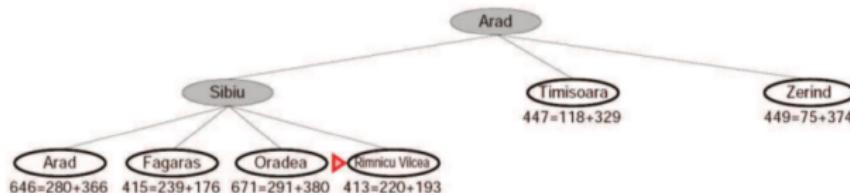
Stratégie A*



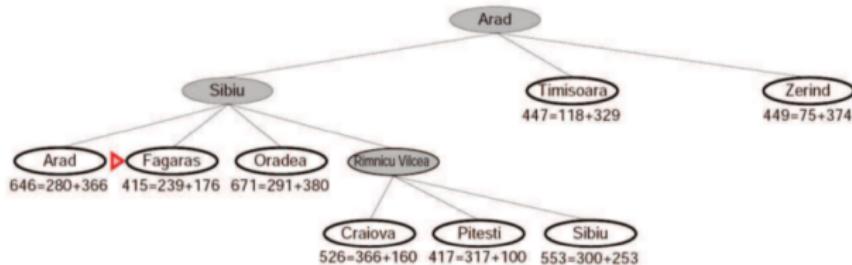
Stratégie A*



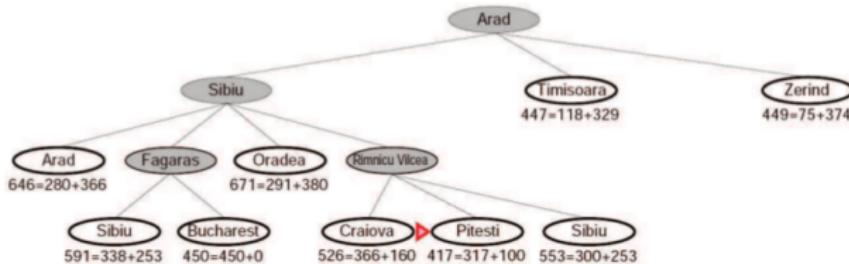
Stratégie A*



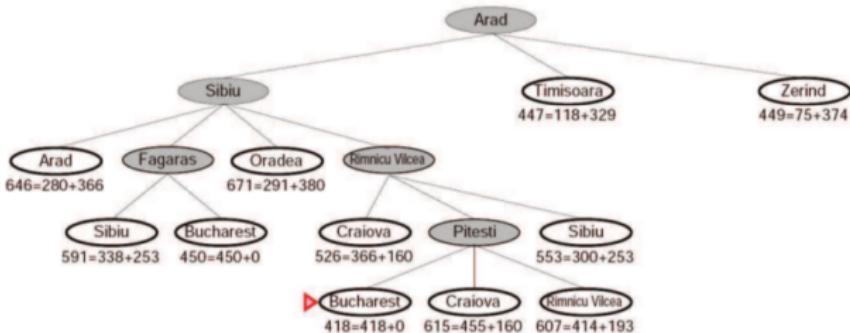
Stratégie A*



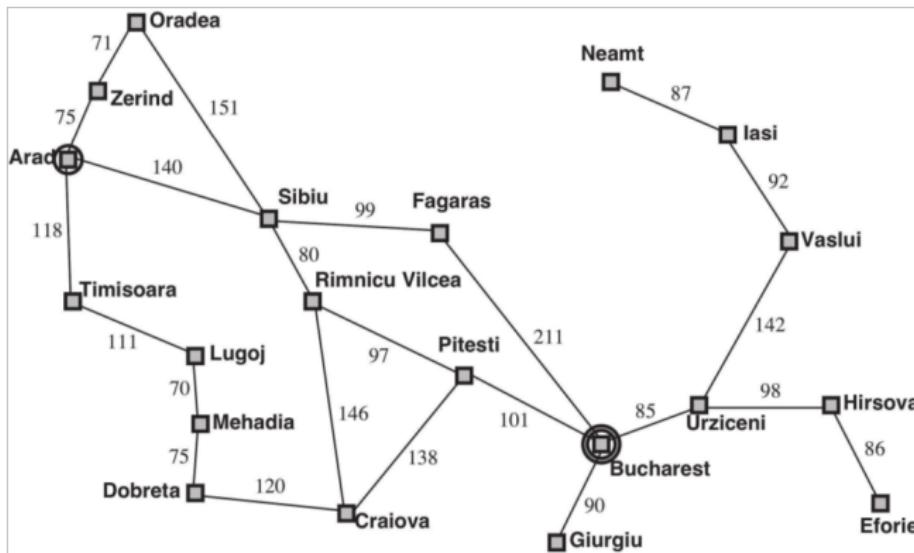
Stratégie A*



Stratégie A*



Stratégie A*



Solution trouvée

- Chemin : Arad - Sibiu - Rimnicu Vilcea - Pitesti - Bucarest
- Coût : $140 + 80 + 97 + 101 = 418$

Remarques

La recherche gloutonne

La recherche gloutonne utilise une estimation du coût minimum au but pour réduire le temps de recherche mais elle n'est ni complète ni optimale

La recherche A^*

- La recherche A^* combine la recherche en coût uniforme et la recherche gloutonne.
- L'heuristique ne doit jamais surestimer le coût réel.
- Complète et optimale

Heuristique

La qualité de l'heuristique influence la durée de la recherche

Propriétés A^*

- Si le graphe est fini, A^* termine toujours.
- Si un chemin vers le but existe, A^* va en trouver un.
- Si la fonction heuristique h retourne toujours une estimation inférieure ou égale au coût réel à venir, on dit que h est admissible.
- Si h est admissible, A^* retourne toujours un chemin optimal.
- Si les coûts des arcs sont tous égaux et strictement positifs et que $h(n)$ retourne 0 quelque soit le noeud n , alors A^* devient une recherche en largeur

Propriétés A*

- Soit $f^*(n)$ le coût exact du chemin optimal du noeud initial au noeud but, **passant par n** .
- Soit $g^*(n)$ le coût exact du chemin optimal du noeud n au noeud but
- On a donc que $f^*(n) = g^*(n) + h^*(n)$
- Si l'heuristique est admissible, pour chaque noeud n exploré par A^* , on peut montrer que l'on a toujours $f(n) \leq f^*(n)$

Propriétés A^*

- Si pour chaque noeud n et son successeur n_s , nous avons toujours

$$h(n) \leq \text{cost}(n, n_s) + h(n_s)$$

On dit alors que h est **admissible**.

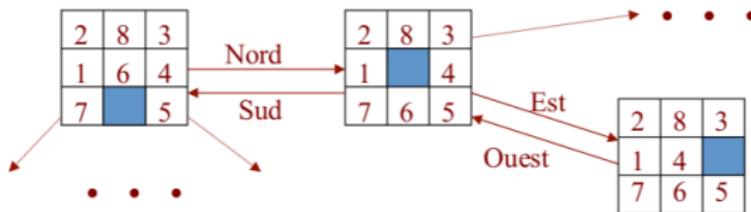
- Si on a deux heuristiques admissibles h_1 et h_2 , tel que $h_1(n) < h_2(n)$ alors $h_2(n)$ conduit plus vite au but (avec h_2 , l'algorithme A^* explore moins ou autant de noeuds que h_1 pour arriver au but).
- Si h n'est pas admissible, l'algorithme A^* donne toujours une solution lorsqu'elle existe, mais il n'y a pas de certitude qu'elle soit optimale

Propriétés A*

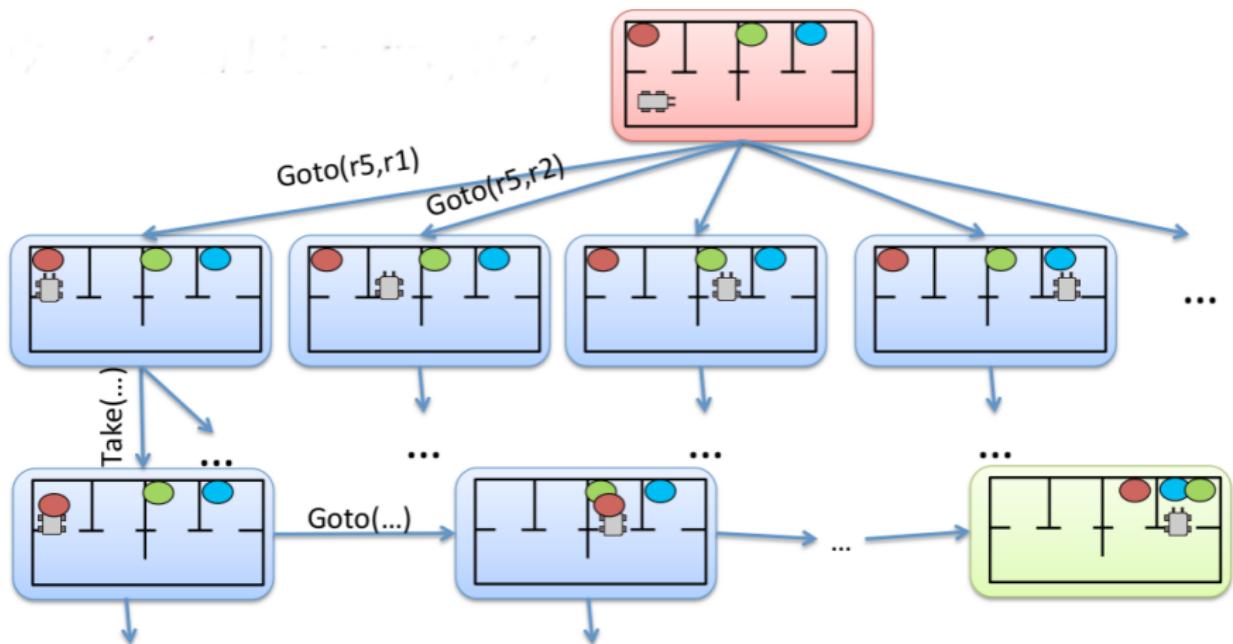
- Beam search
 - on met une limite sur le nombre de noeuds exploré à chaque niveau
 - recommandé lorsqu'on ne dispose pas d'assez d'espace mémoire
- Iterative deepening
 - on met une limite sur la profondeur
 - on lance A* jusqu'à la limite de profondeur spécifiée
 - si pas de solution, on augmente la profondeur et on recommence A*
 - on réitère le processus jusqu'à trouver une solution

Application : jeu d'énigme

- **Etat** : configuration légale du jeu
- **Etat initial** : configuration initiale
- **Etat final (but)** : configuration gagnante
- **Transitions**

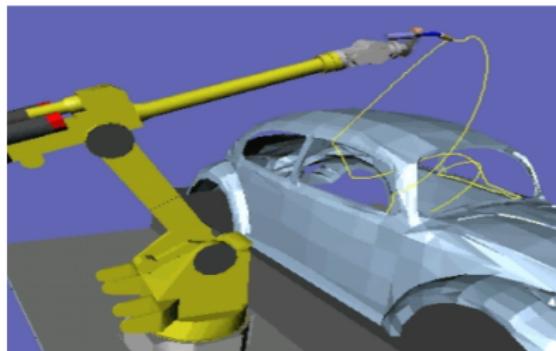


Application : planification



Application : industrie automobile

Calculer une trajectoire géométrique d'un solide articulé sans collision avec des obstacles statiques



Entrées :

- Géométrie du robot et des obstacles
- Cinétique du robot (degrés de liberté)
- Configuration initiale et finale

Planificateur de trajectoires

Sortie :

- Une séquence continue de configurations rapprochées, sans collision, joignant la configuration initiale à la configuration finale

Algorithmique pour les jeux

Pourquoi les jeux ?

- problème bien structuré (règles bien définies)
- Mesure de performance facile
- Défi intellectuel
- Complexe
- Amusant

Exemples

- Morpion
- Dames
- Rubik's cube
- Echecs
- Go

Historique

- 1950 : Min-Max [Shannon]
- 1958 : Elagage $\alpha - \beta$ [Mc Carthy, Newell Simon]
- ...
- 1979 : Backgammon : le champion du monde est battu
- 1994 : Dames : le champion du monde est battu (Chinook)
- 1997 : Othello : le champion du monde est battu (Logistello)
- 1999 : Echecs : Kasparov est battu (Deep blue)
- 2015 : Go : Un des meilleurs joueurs mondiaux (Fan Hui) est battu par AlphaGo de Google DeepMind.

Caractéristiques

Incertitude

Toutes les actions ne sont pas contrôlées par l'ordinateur

Grande dimension

L'espace de recherche est vaste et souvent impossible à explorer entièrement

Fonction d'évaluation

Définition d'une fonction de gain en fonction d'un état

Classification des jeux

	Déterministique	Non déterministique
Information parfaite	Dame, Echecs, Go, Puissance 4	Backgammon, Monopoly
Information imparfaite	Mastermind, Diplomatie	Bridge, poker, Scrabble

Fonction d'évaluation

But

Une fonction d'évaluation estime la valeur d'une position. Cette valeur peut être négative et est indépendante des décisions passées ou venir (à la différence d'une heuristique).

- $f(n) > 0$: position favorable
- $f(n) < 0$: position défavorable
- $f(n) = 0$: position équilibrée
- $f(n) = +\infty$: position gagnante pour le joueur courant
- $f(n) = -\infty$: position perdante pour le joueur courant

Propriétés

Généralement

- Il s'agit généralement d'une fonction linéaire
- Jeux à somme nulle : ce que l'un gagne l'autre perd

Exemples de fonctions d'évaluation

Morpion

$$F(x) = (\#\text{3cases pour moi}) - (\#\text{3cases pour mon adversaire})$$

3cases : ligne(s), colonne(s) ou diagonale(s) pour lesquelles il est possible d'aligner 3 pions

Echecs

$$\begin{aligned} F(x) = & (P - P') + 3(N - N' + B - B') \\ & + 5(R - R') + 9(Q - Q') + 200(K - K') \\ & - 0.5(D - D' + S - S' + I - I') \\ & + 0.1(M - M') \end{aligned}$$

- P, N, B, R, Q, K représentent les pions, cavaliers, fous, tours, reines, rois pour les blancs
- D, S, I représentent respectivement les pions doublés, les pions arriérés et les pions isolés pour les blancs
- M représente la mobilité
- Les lettres "primes" représentent la même chose du point de vue des noirs



Algorithme Min-Max

Principe

- Construction de l'arbre jusqu'à une profondeur donnée d
- Appel à la fonction d'évaluation
- Le joueur "Max" (resp. "Min") va tenter de maximiser (resp. minimiser) son score
- Idée : On s'attend au pire \Rightarrow il est rationnel

Algorithme Min-Max

Algorithm 2 Min-Max

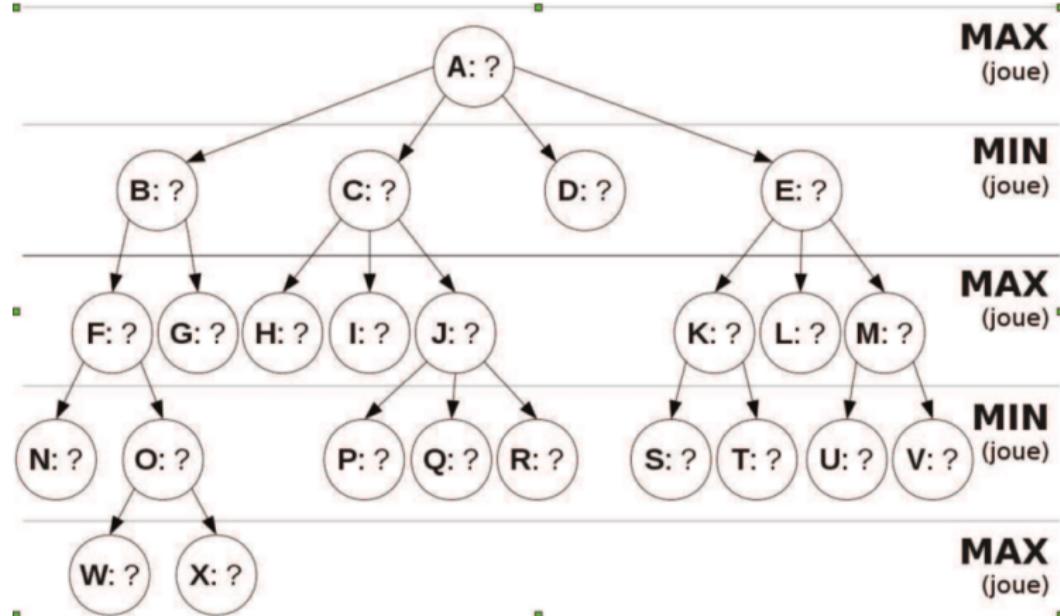
```
arguments node  $n$ , depth  $d$ 
if  $n$  is terminal or  $d = 0$  then
    return evaluation of  $n$ 
else
    if  $n$  is a node MAX then
         $\alpha \leftarrow -\infty$ 
        for each child  $c$  of  $n$  do
             $\alpha \leftarrow \max(\alpha, \text{minimax}(c, d - 1))$ 
        end for
        return  $\alpha$ 
    else
         $\alpha \leftarrow +\infty$ 
        for each child  $c$  of  $n$  do
             $\alpha \leftarrow \min(\alpha, \text{minimax}(c, d - 1))$ 
        end for
        return  $\alpha$ 
    end if
end if
```

Algorithme Min-Max

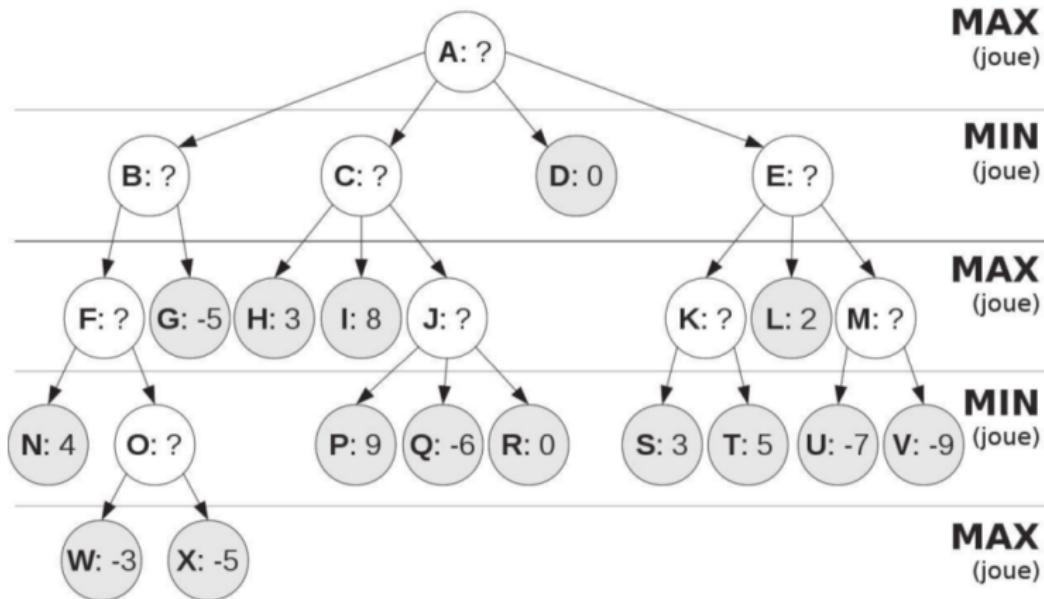
Algorithm 3 Simplification Min-Max

```
arguments node  $n$ , depth  $d$ 
if  $n$  is terminal or  $d = 0$  then
    return evaluation of  $n$ 
else
     $\alpha \leftarrow -\infty$ 
    for each child  $c$  of  $n$  do
         $\alpha \leftarrow \max(\alpha, \text{-minimax}(c, d - 1))$ 
    end for
    return  $\alpha$ 
end if
```

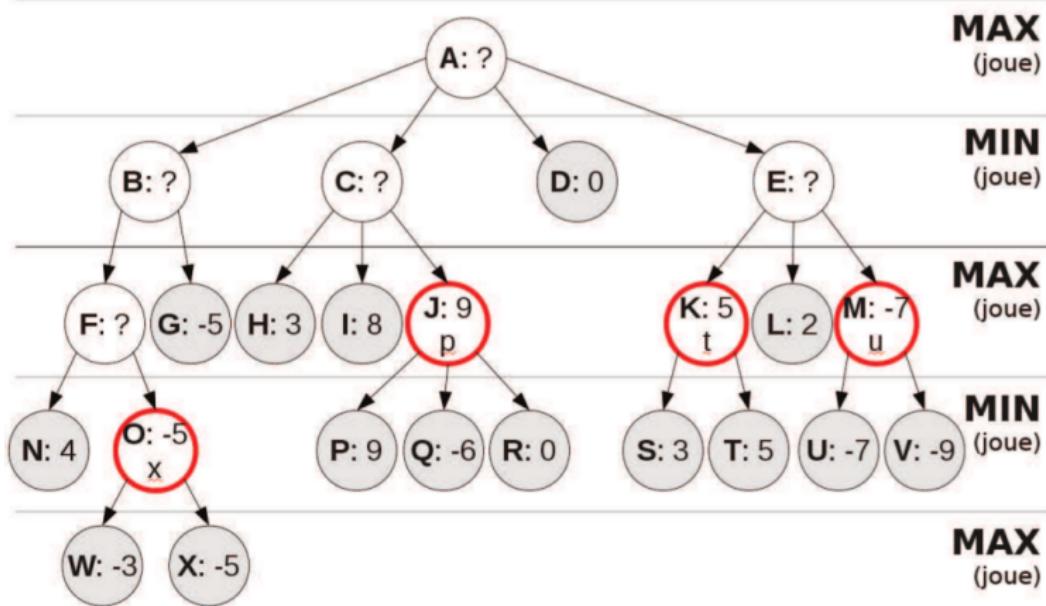
Exemple de Min-Max



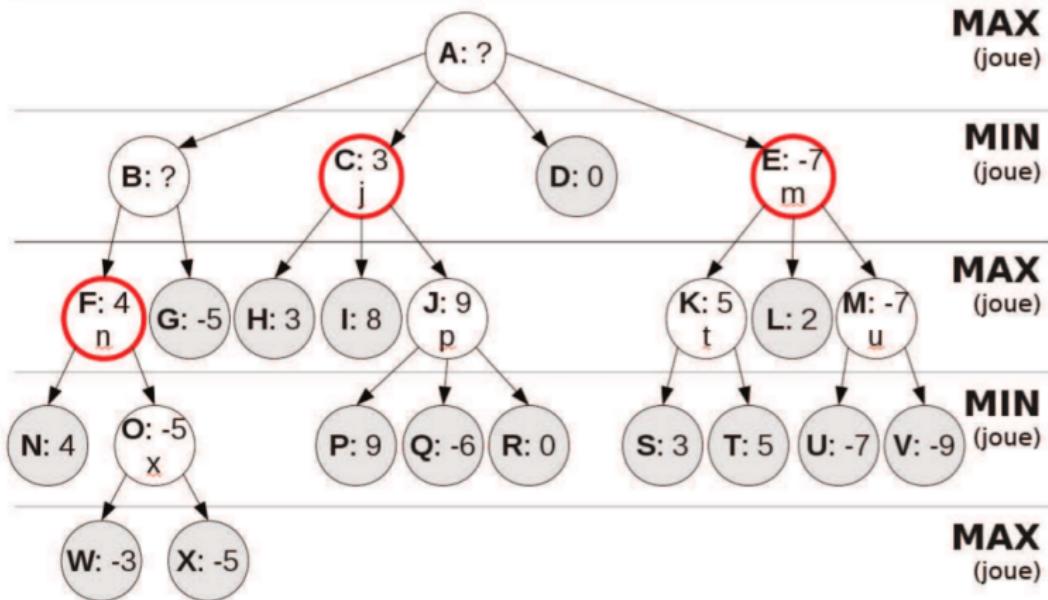
Exemple de Min-Max



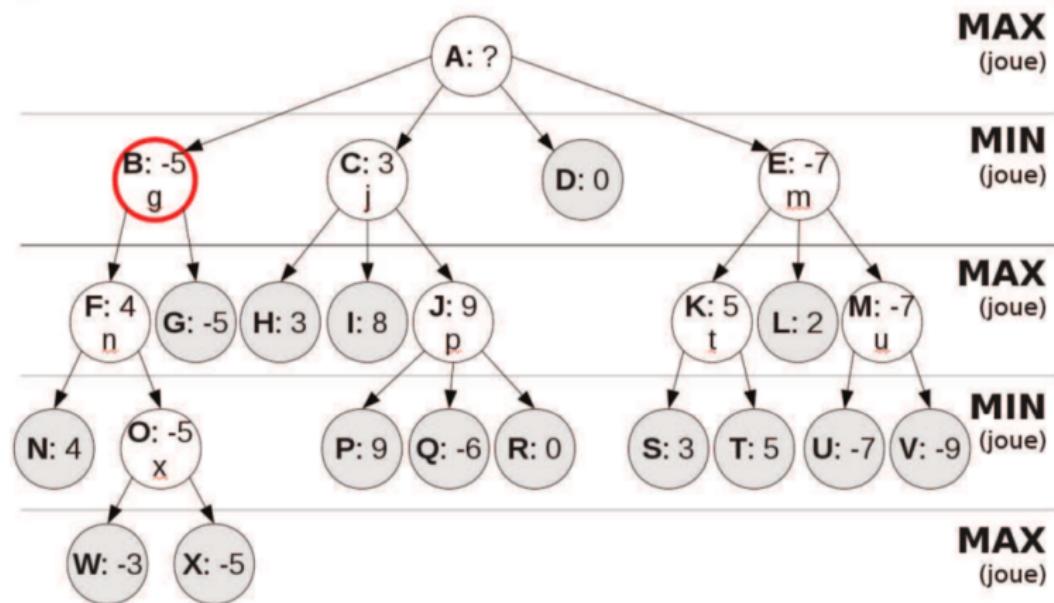
Exemple de Min-Max



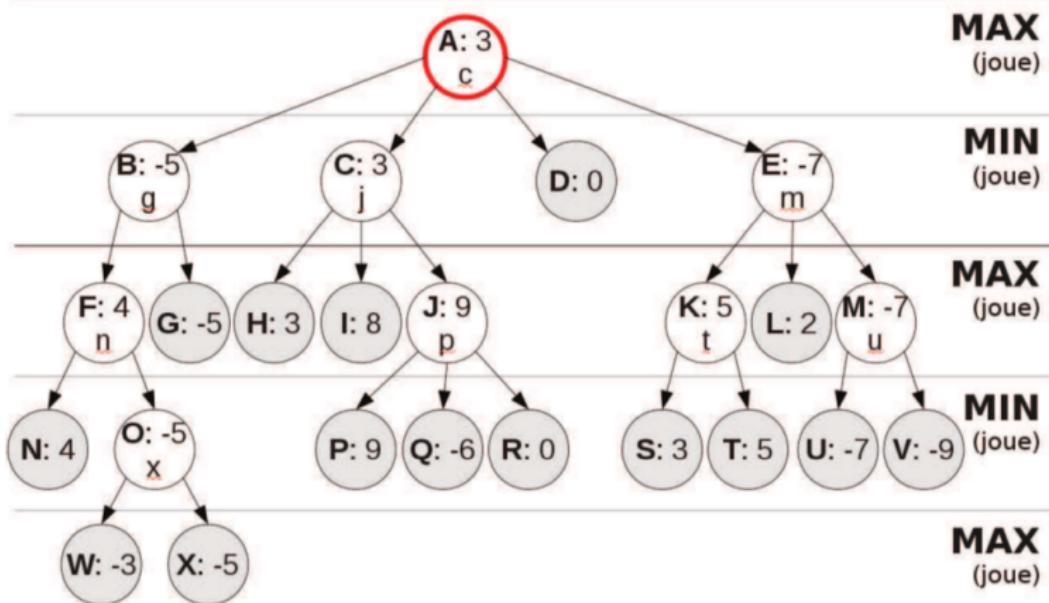
Exemple de Min-Max



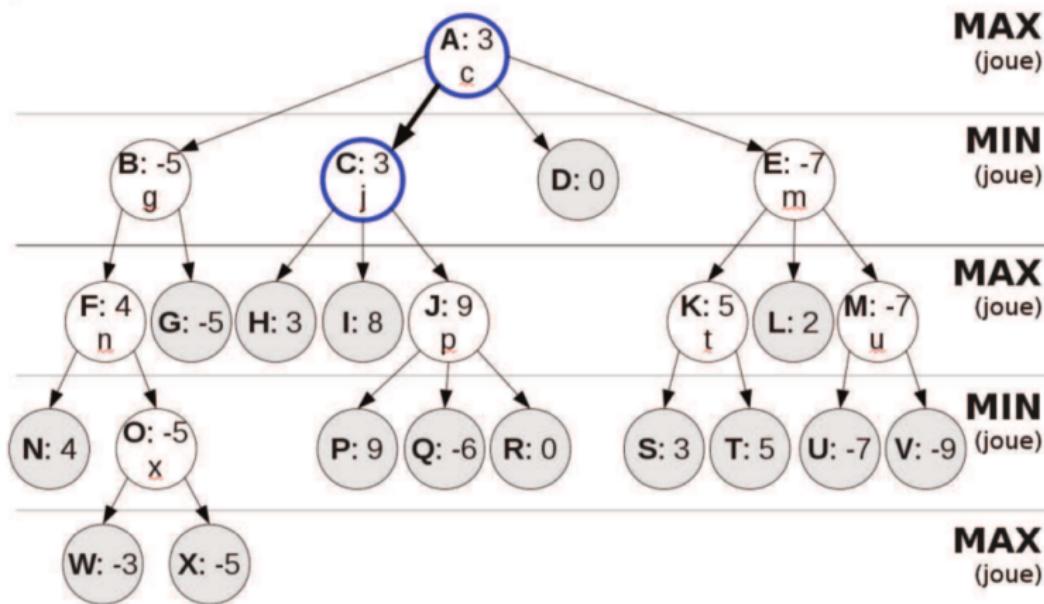
Exemple de Min-Max



Exemple de Min-Max



Exemple de Min-Max



Amélioration α - β

Idée (Pat Winston)

"Si vous pensez que c'est assurément mauvais, ne perdez pas de temps à vérifier à quel point c'est vraiment mauvais"

Principe

Elaguer les branches qui de toute façon ne seront pas choisies

Amélioration α - β

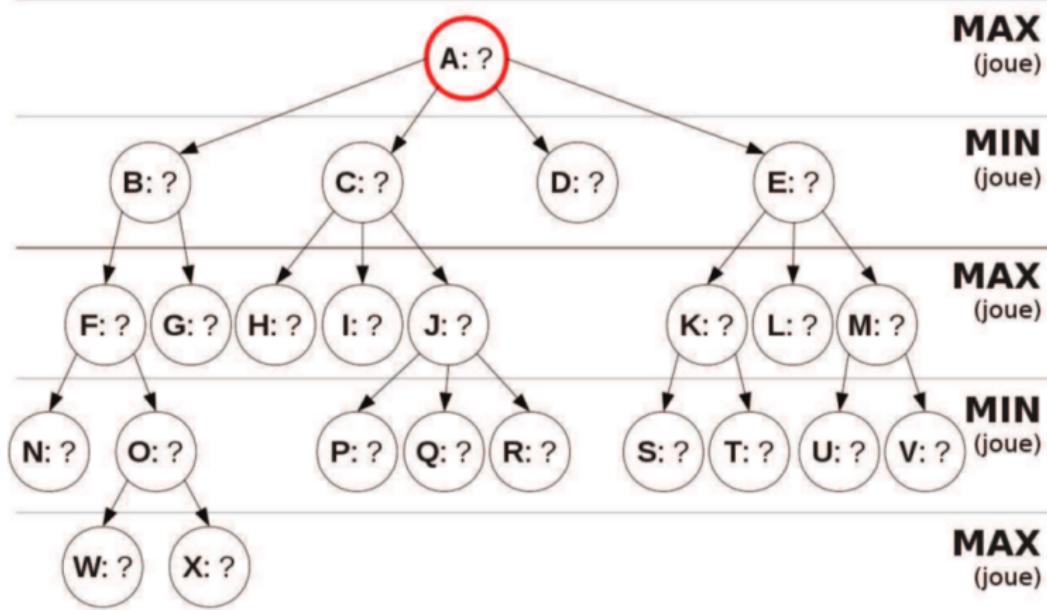
- Etendre l'arbre de jeu jusqu' à une profondeur d par recherche en profondeur
- Ne pas développer un noeud dès qu'il devient évident que ce noeud ne sera pas choisi par la suite
- Chaque noeud max garde la trace α équivalente à la valeur du meilleur successeur
- Chaque noeud min garde la trace β équivalente à la valeur du pire successeur
- Règles d'élagage :
 - Elaguer si pour un noeud max, la valeur α est supérieure ou égale à la valeur β du parent
 - Elaguer si pour un noeud min, la valeur β est inférieure ou égale à la valeur α du parent

Amélioration α - β

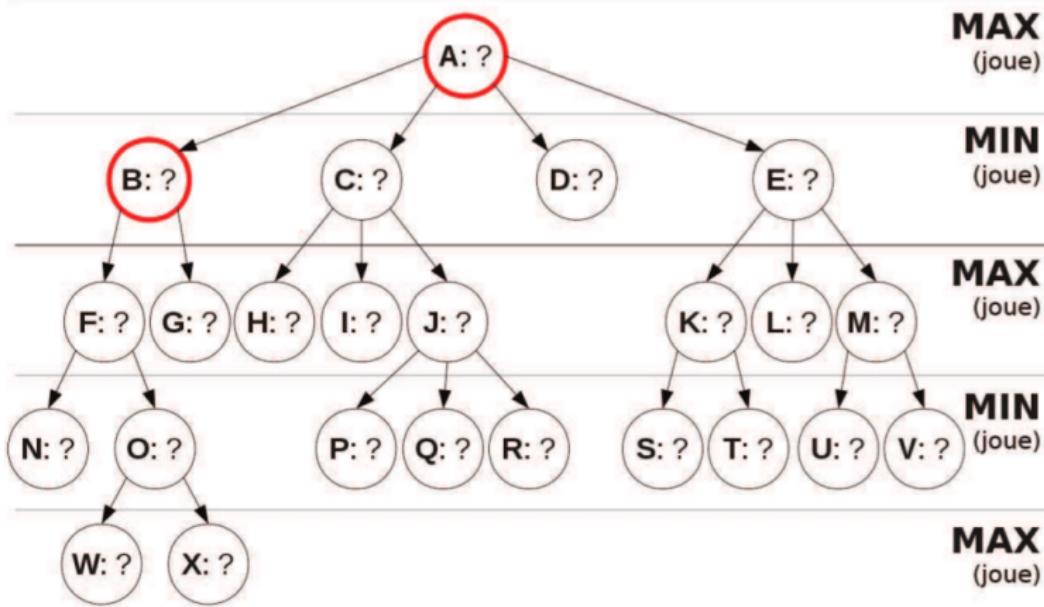
Algorithm 4 Algorithme α - β

arguments node n , depth d , α , β
if n is terminal or $d = 0$ **then**
 return evaluation of n
else
 $\alpha \leftarrow -\infty$
 for each child c of n **do**
 $\alpha \leftarrow \max(\alpha, \text{-alphabeta}(c, d - 1, -\beta, -\alpha))$
 if $\beta \leq \alpha$ **then**
 break
 end if
 end for
 return α
end if

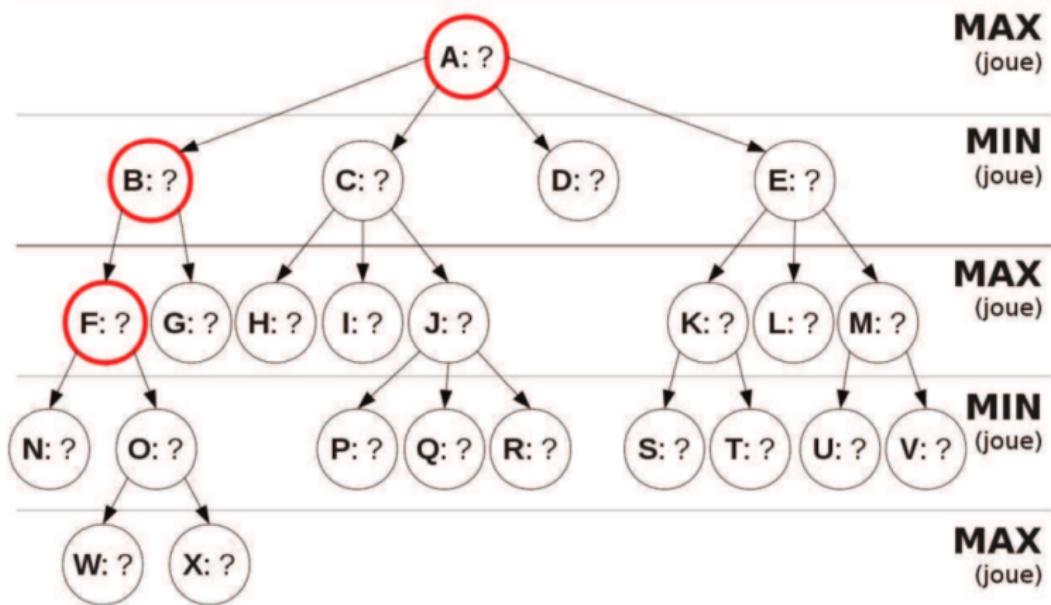
Exemple d' α - β



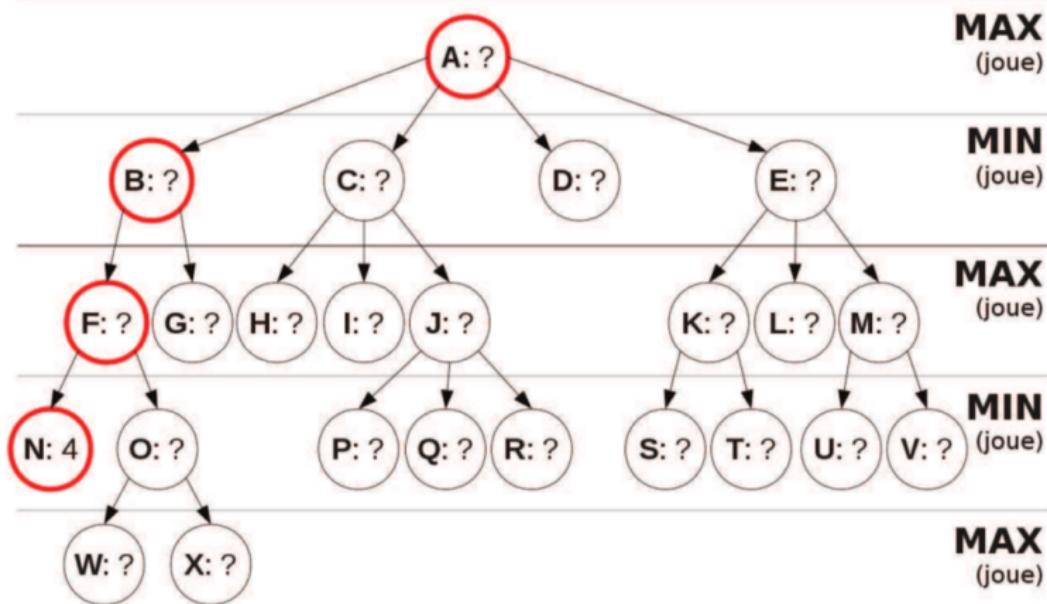
Exemple d' α - β



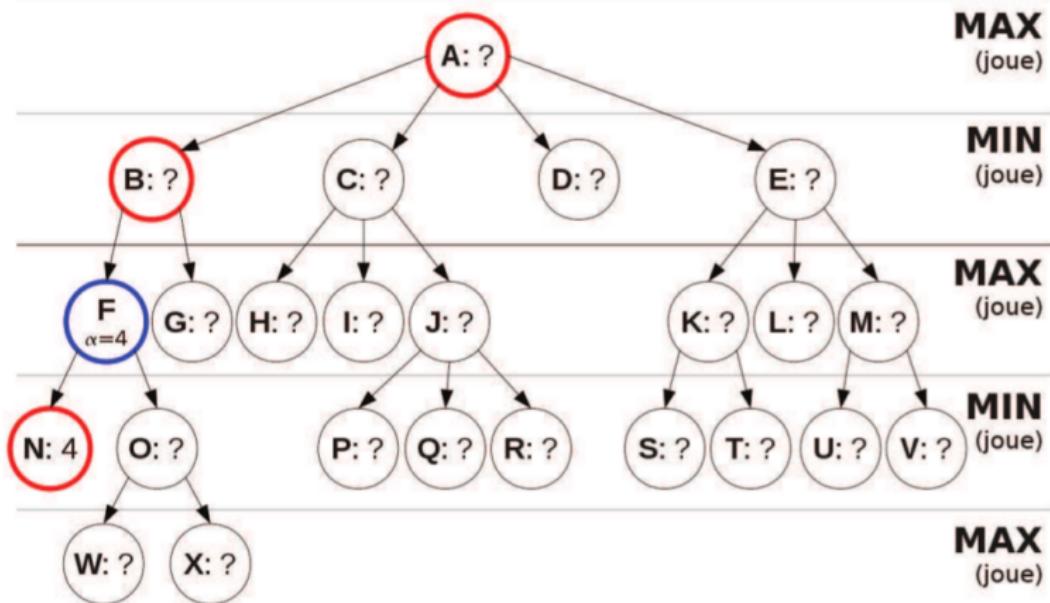
Exemple d' α - β



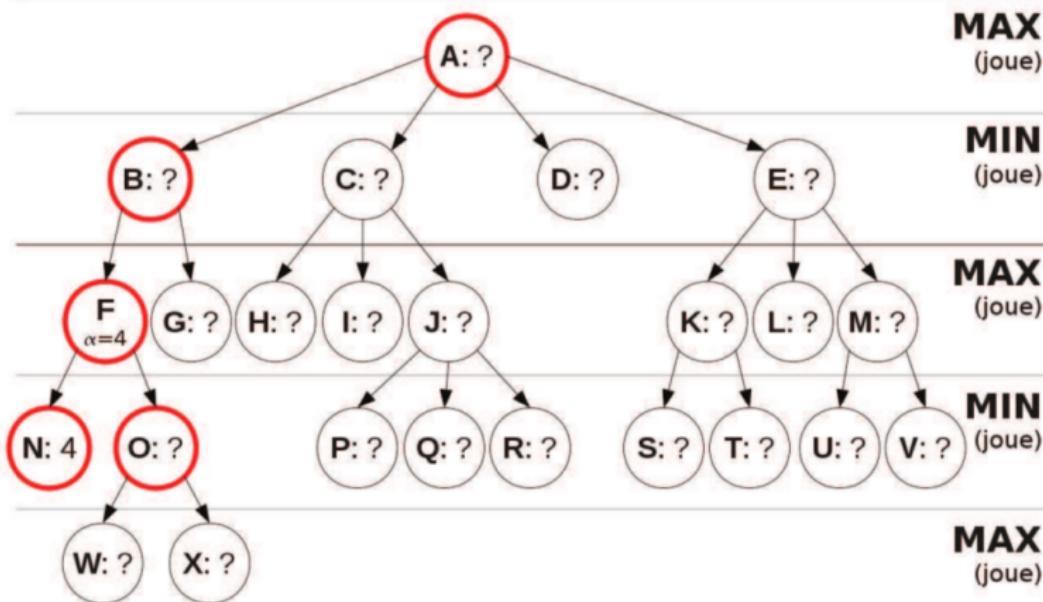
Exemple d' α - β



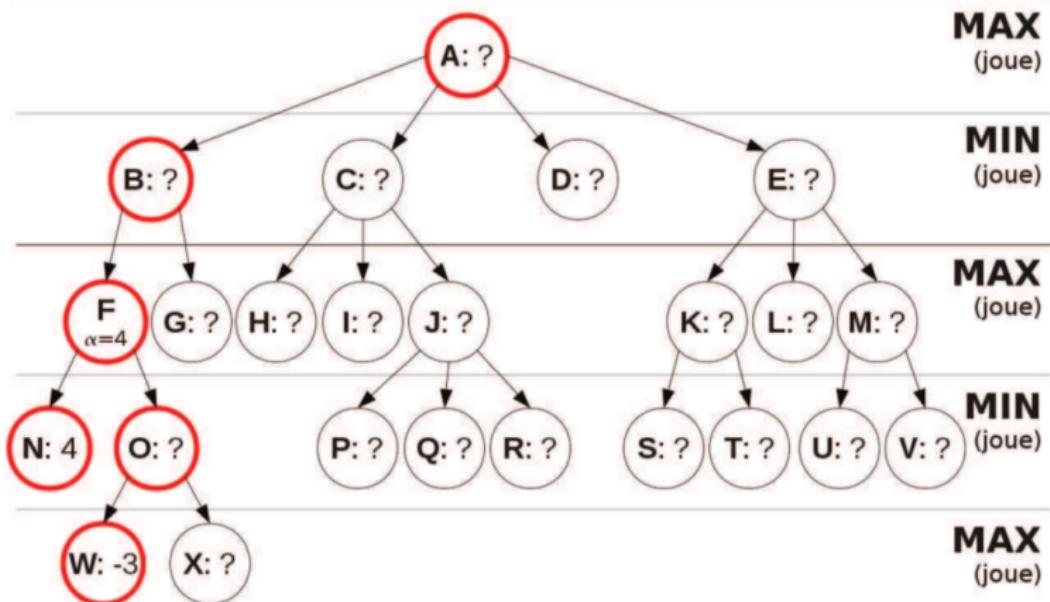
Exemple d' α - β



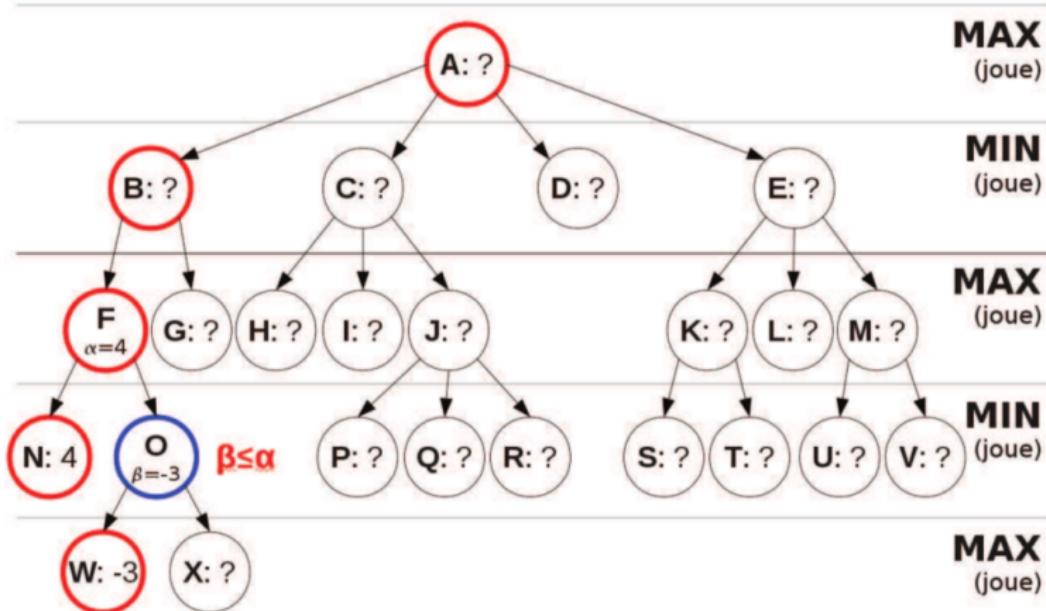
Exemple d' α - β



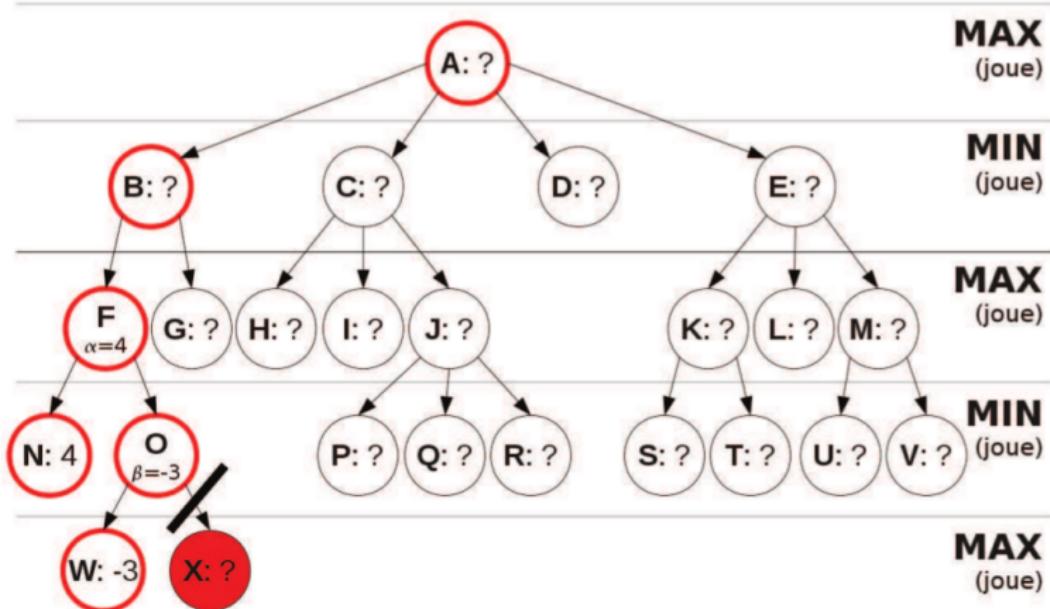
Exemple d' α - β



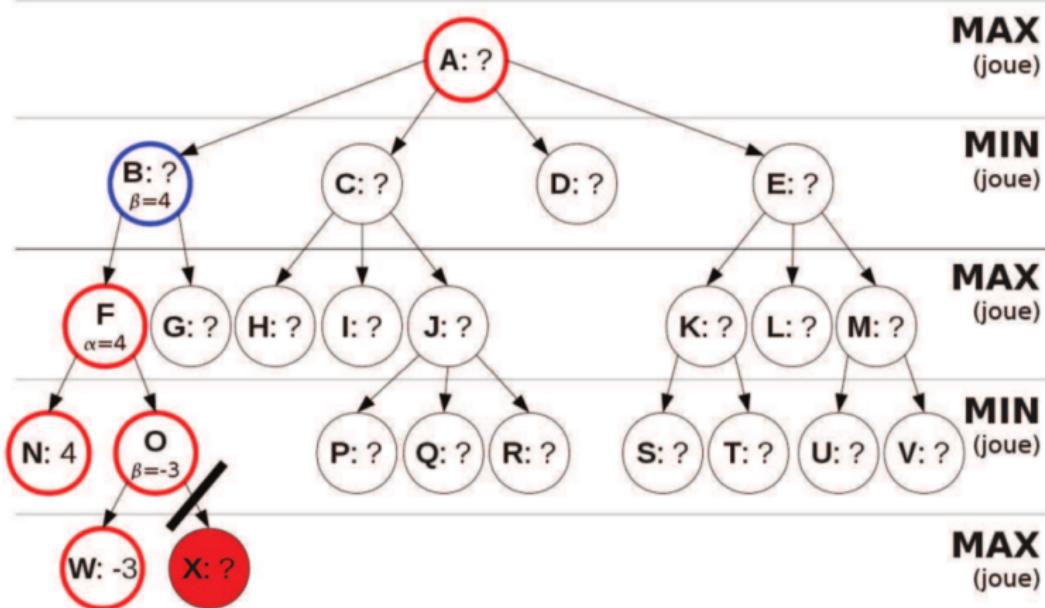
Exemple d' α - β



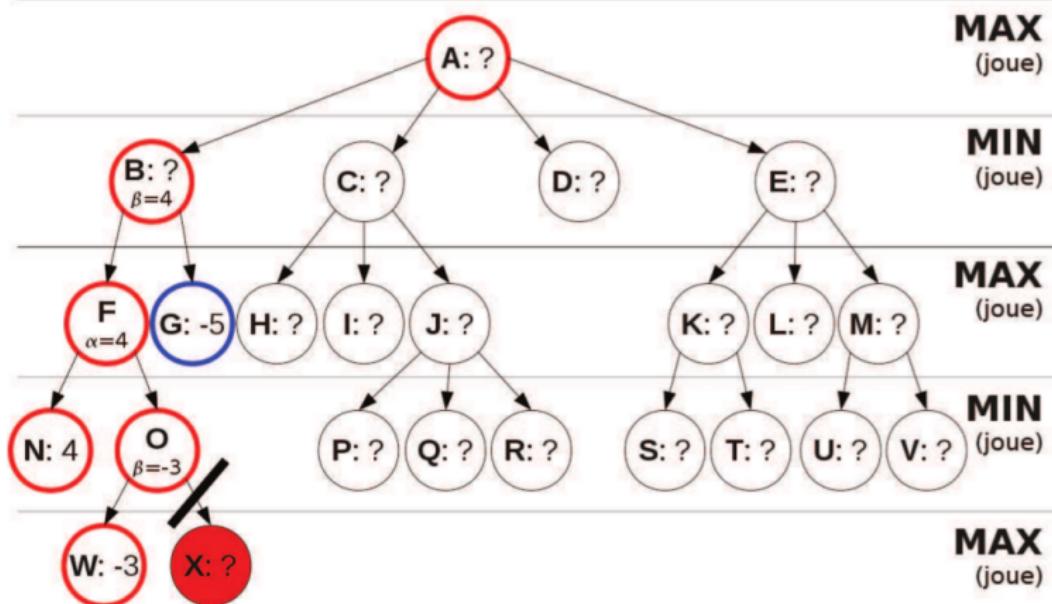
Exemple d' α - β



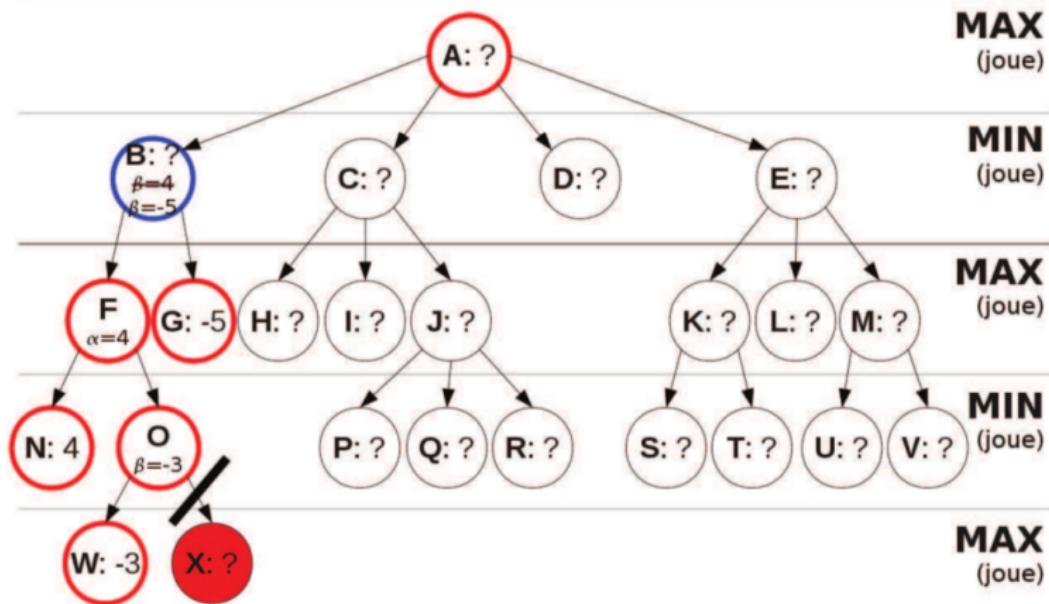
Exemple d' α - β



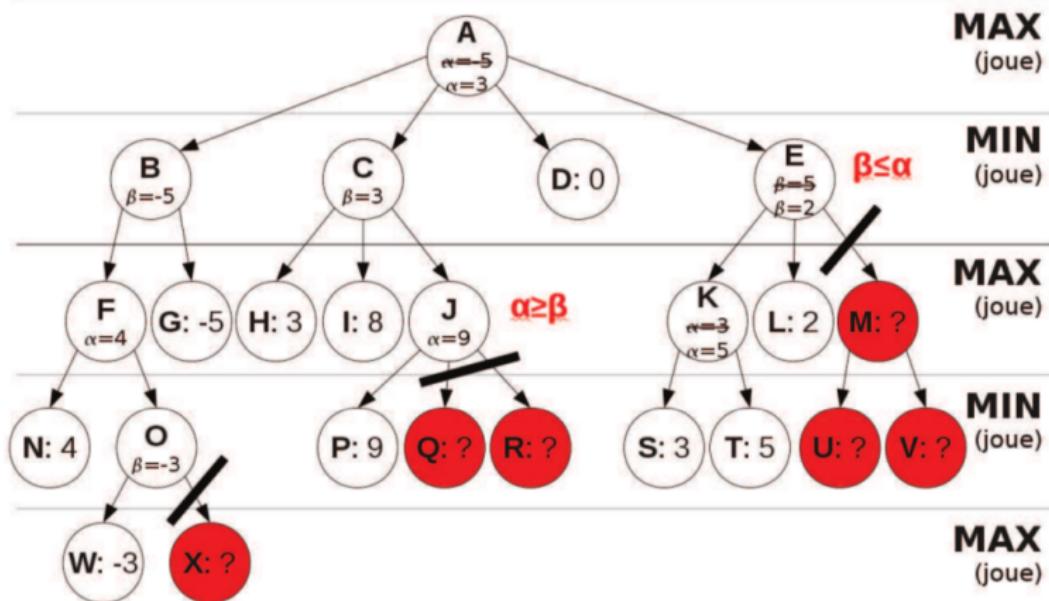
Exemple d' α - β



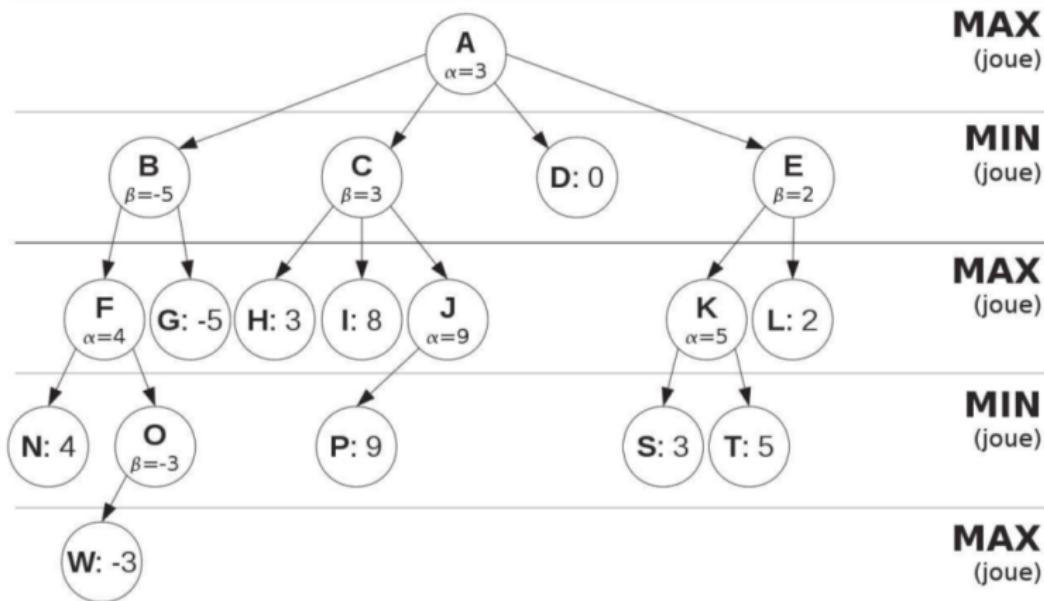
Exemple d' α - β



Exemple d' α - β



Exemple d' α - β



Algorithme d' α - β

- Propriétés

- Pire cas : pas d'élagage et on examine b^d noeuds
- Meilleur cas : le meilleur coup est la première action étudiée et on examine $b^{d/2}$
- Renvoie toujours la même réponse que Min-max

- Améliorations

- Mémorisation de la variante principale
- Recherche de quiescence
- Approfondissement itératif
- Heuristique de l'historique
- ...