Exploration systématique de graphes

D.E ZEGOUR
École Supérieure d'Informatique
ESI

Exploration systématique de graphes

Sommaire

Exploration systématique de graphes

Introduction

Méthode de recherche (fouille) systématique pour la recherche d'une solution dans un graphe orienté implicite, le plus souvent sans circuit ou même arborescent.

Exemple : trouver son chemin dans un labyrinthe

Le parcours se fait généralement en profondeur.

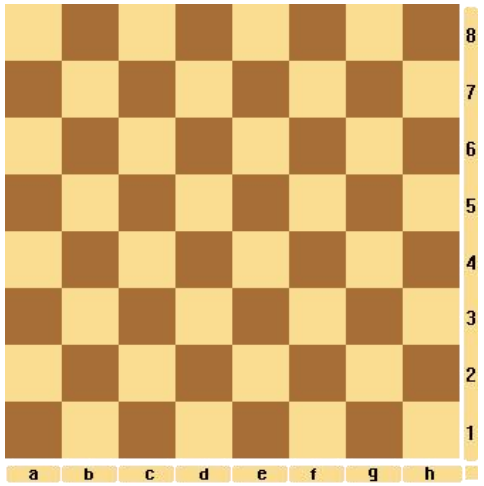
Si le graphe est très grand(ou infini) ou si l'on veut plutôt chercher une solution avec peu de manipulations, on utilise un parcours en largeur.

Exploration systématique de graphes

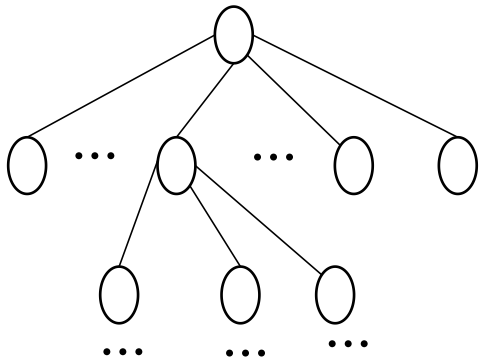
Application 1 : Problèmes des *8 reines*

Il faut placer 8 reines dans un échiquier (matrice 8X8) sans qu'aucune d'entres elles ne soit en prise par une autre.

Deux reines sont en prise, si elles se trouvent sur une même ligne, une même colonne ou une même diagonale



Exploration systématique de graphes



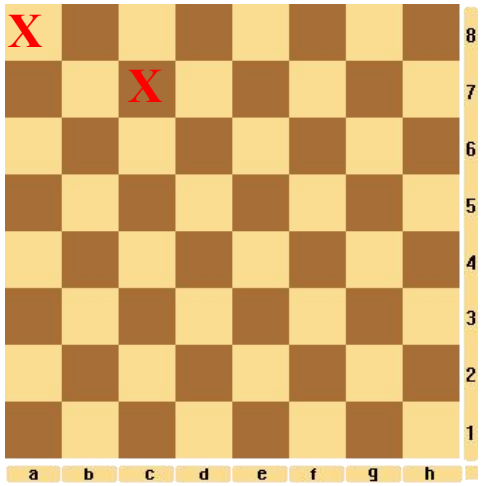
Application 1 : Problèmes des *8 reines*

On démarre d'un échiquier vide.

Premier niveau : Il y a 64 cas possibles de placer la premier reine.

Deuxième niveau : Pour chaque cas du niveau 1, il y a 63 possibilités de placer une reine.(Chaque cas qui ne répond pas à la condition est écarté).

Et ainsi de suite....



Exploration systématique de graphes

Application 1 : Problèmes des *8 reines*

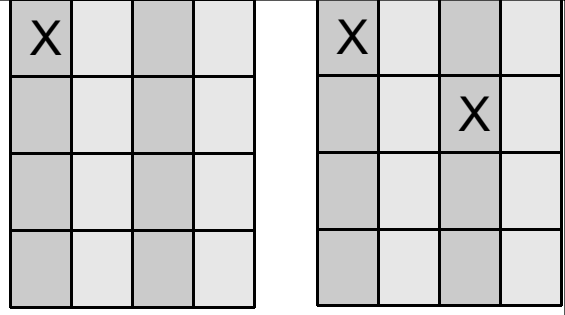
On peut améliorer l'algorithme en considérant uniquement les positions où deux reines ne sont ni sur la même ligne ni sur la même colonne.

On peut ainsi représenter tout l'échiquier par un vecteur V [1, 8] où les indices désignent les lignes et les contenus les colonnes.

Tableau V[1..8]

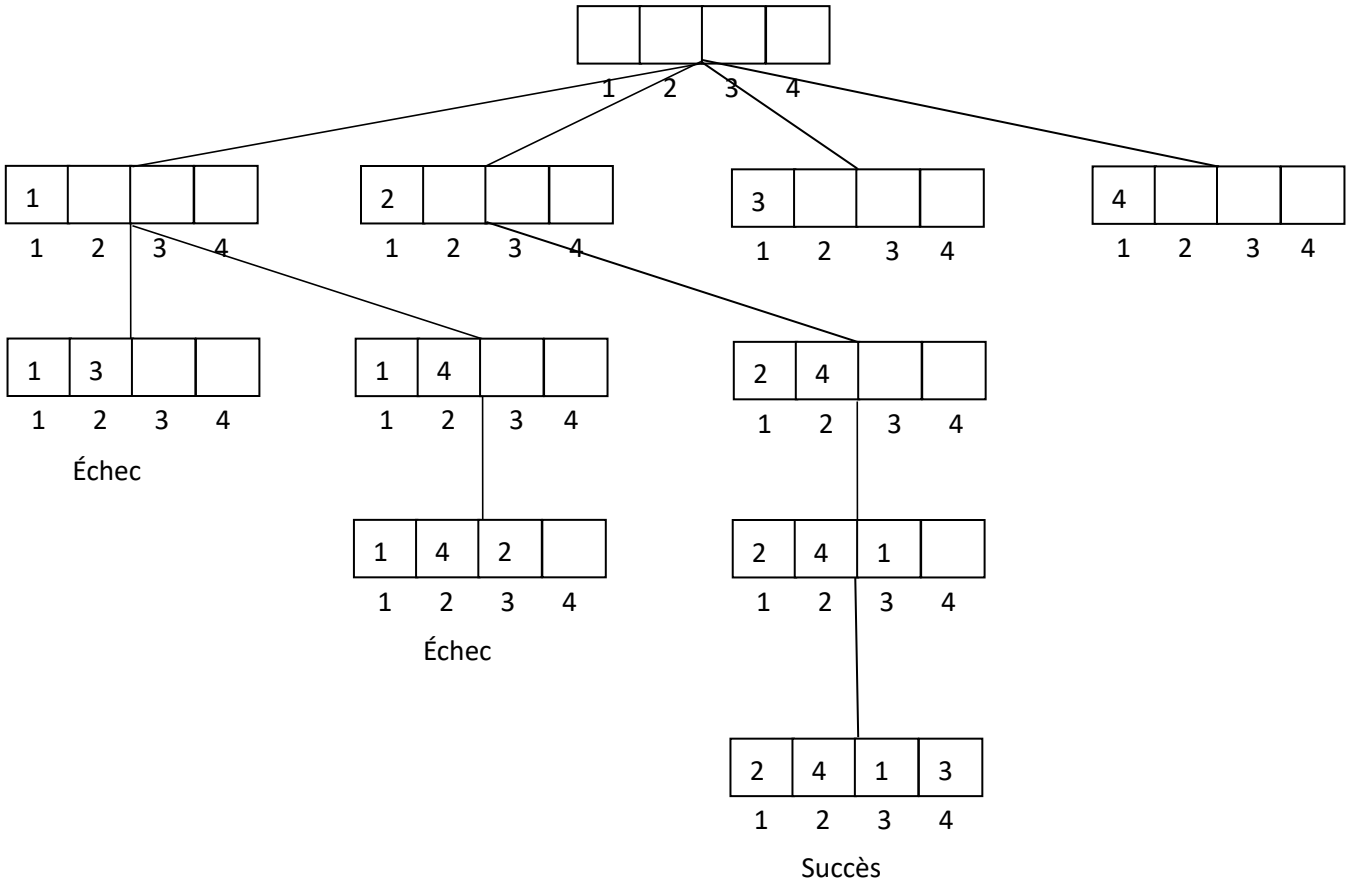
Présence de la reine:
- Première ligne, première colonne
- Deuxième ligne, troisième colonne
- Troisième ligne, septième colonne

Exploration systématique de graphes



Application 1 : Problèmes des *8 reines*

Echiquier 4 X 4.



Un arbre implicite se construit avec une recherche en profondeur.

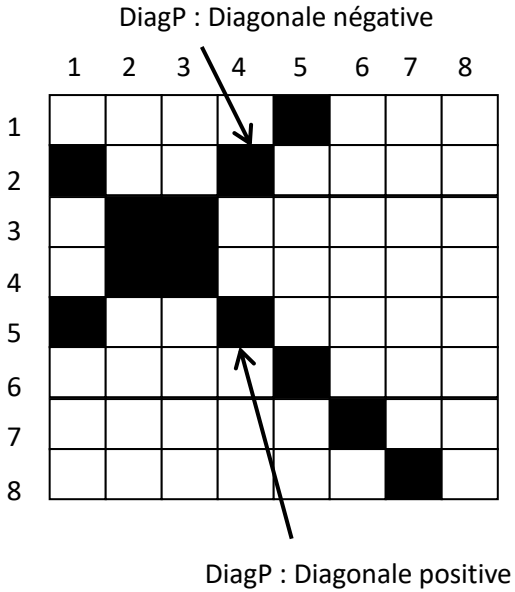
Exploration systématique de graphes

Application 1 : Problèmes des *8 reines*

Pour une diagonale positive, la différence des indices est constante.

Les valeurs possibles pour les diagonales positives sont : 0, 1, 2, ...7, -1, -2, ..., -7

Pour une diagonale négative, la somme des indices est constante



Les valeurs possibles pour les diagonales négatives sont : 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16.

Exploration systématique de graphes

Application 1 : Problèmes des *8 reines*

```
Procedure Placer (K) { On a déjà réussi à placer K reines et on place la reine suivante}
Si K = 8     Ecrire (V)
Sinon
    Pour j=1, 8  {j désigne colonne et (k+1) désigne la ligne}
        Si (j n'appartient pas à Col ET (k+1 - j) n'appartient pas à DiagP ET
            (k+1 + j) n'appartient pas à DiagN) :
              V[k+1] := j
              Col := Col U {j}  ;      DiagP := Diagp U {k+1 - j}   ;    DiagN := DiagN U {k+1 + j}
              Placer( k+1 )
         Fsi
     Finpour
Fsi
```

Appel initial : Placer(0)
Col, DiagP et DiagN initialisés à {}.

Exploration systématique de graphes

Application 1 : Problèmes des *8 reines*

Complexité

Equation de reccurence (pour n reines)

T(n) : nombre de reines restantes à placer

$T(0) = a$

$T(n) = n\,T(n-1) + b,\ n > 0$

$= n\,((n-1)\,T(n-2) + b) + b$

$= n\,(n-1)\,T(n-2) + nb + b$

$= n\,(n-1)\,(\,(n-2)\,T(n-3) + b) + nb + b$

$= n\,(n-1)\,(n-2)\,T(n-3) + n\,(n-1)b + nb + b$

$= n\,(n-1)\,(n-2)\,(n-3)\,T(n-4) + n(n-1)(n-2)b + n(n-1)b + nb + b$

...

$= n\,(n-1)\,(n-2)\,....(n - i + 1)\,T(n - i) + n(n-1)...(n-i+2)b + n(n-1)...(n-i+3)b + ... + nb + b$

Exploration systématique de graphes

Application 1 : Problèmes des *8 reines*

Complexité

$= n (n-1) (n-2) ....(n - i +1) T(n - i) + n(n-1)...(n-i+2)b + n(n-1)...(n-i+3)b + ... + nb + b$

pour i=n

$= n (n-1) (n-2) ....1 T(0) + b (1 + n + n(n-1) + n(n-1)(n-2) + ... + n (n-1)(n-2)...2)$

$= a n! + b (1 + n! + n!/2 + ... + n!/(n-1)!)$

C'est O(n!)

Pour n=8, 92 possibilities

Exploration systématique de graphes

Application 2 : Problèmes des *cruches d'eau*

Deux cruches A(4litres) et B(3litres).
On part de (0, 0), A = 0 et B = 0
et on veut arriver à (2, n).

La technique de backtracking consiste
à appliquer les règles de 1 à 8,
dans cet ordre,
pour chaque nouvel état visité.

Regles:

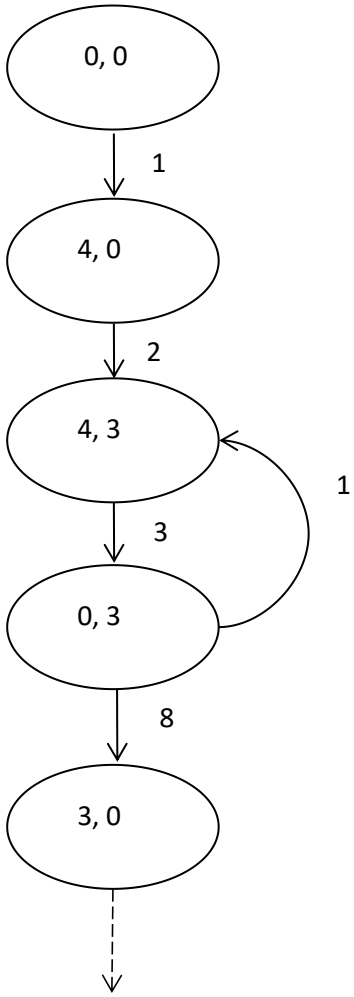
1. **Si A non pleine : Remplir A**
2. **Si B non pleine : Remplir B**
3. **Si A non vide alors vider A**
4. **Si B non vide alors vider B**
5. **Si A non vide et A>(3-B) alors verser le contenu de A dans B jusqu'à ce que B soit plein.**
6. **Si B non vide et B>(4-A) alors verser le contenu de B dans A jusqu'à ce que A soit plein.**
7. **Si A non vide et A <= (3-B) alors verser tout le contenu de A dans B.**
8. **Si B non vide et B <= (4-A) alors verser tout le contenu de B dans A.**

Exploration systématique de graphes

Application 2 : Problèmes des *cruches d'eau*

Regles:

1. Si A non pleine : Remplir A
2. Si B non pleine : Remplir B
3. Si A non vide alors vider A
4. Si B non vide alors vider B
5. Si A non vide et A>(3-B) alors verser le contenu de A dans B jusqu'à ce que B soit plein.
6. Si B non vide et B>(4-A) alors verser le contenu de B dans A jusqu'à ce que A soit plein.
7. Si A non vide et A <= (3-B) alors verser tout le contenu de A dans B.
8. Si B non vide et B <= (4-A) alors verser tout le contenu de B dans A.



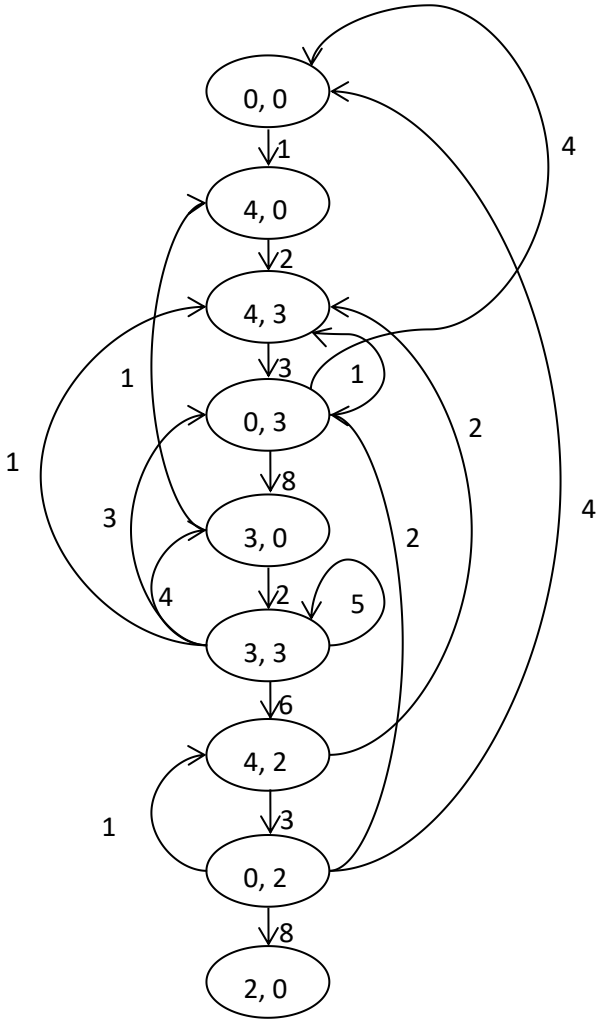
Exploration systématique de graphes

Application 2 : Problèmes des *cruches d'eau*

Etat final

La flèche verticale descendante désigne les nouveaux états,

Les flèches montantes désignent les retours arrières vers les états déjà visités.



Exploration systématique de graphes

Application 2 : Problèmes des *cruches d'eau*

Algorithme

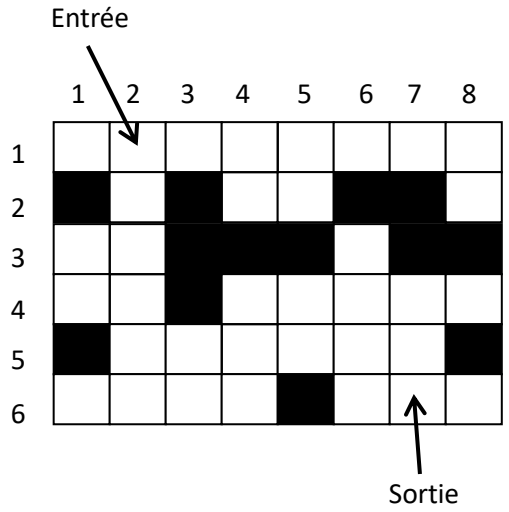
Config <-- (0, 0)
Tq Config <> (2, n)
    Appliquer les régles P1, P2,...,P8 dans cet ordre
    Ecarter toute règle qui produit une configuration existante.
    --> Config = (x, y)
Ftq

Exploration systématique de graphes

Application 3 :Sortie d'un labyrinthe

Le problème consiste, une fois rentré dans le labyrinthe, de sortir de celui-ci.

On suppose, ici, qu'il y a une seule entrée et une seule sortie.

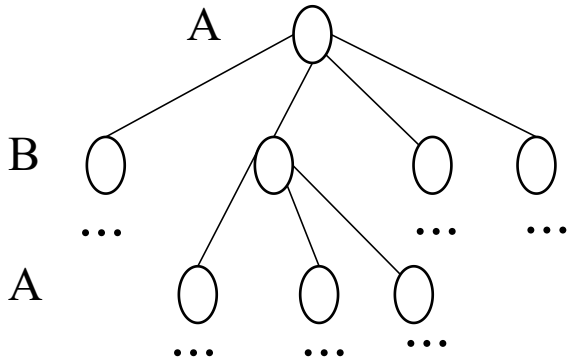


Déplacements possibles :

1. vers le droite
2. vers le bas
3. vers le haut
4. vers la gauche

Mémoriser les états déjà visités

Exploration systématique de graphes



Principe du MIN-MAX

On considère les jeux de stratégie entre deux joueurs.

Les deux joueurs sont soumis aux mêmes règles(symétrie).

Le jeu est déterministe(le hasard n'intervient pas).

Ce type de jeu peut être représenté sous forme d'une arborescence.

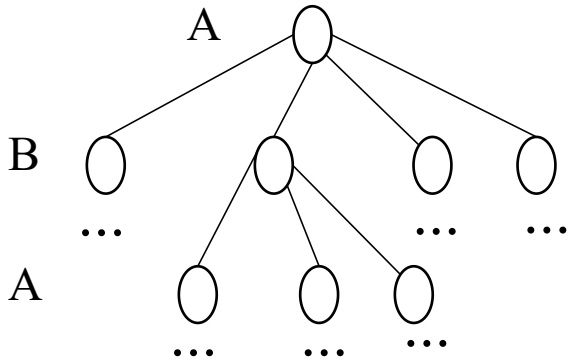
Chaque nœud représente une configuration possible du jeu.

Un arc représente une transition légale entre deux configurations.

La racine constitue la configuration initiale, les feuilles les configurations finales(gagné, perdu, nul).

L'un des joueurs est la machine.

Exploration systématique de graphes



Principe du MIN-MAX

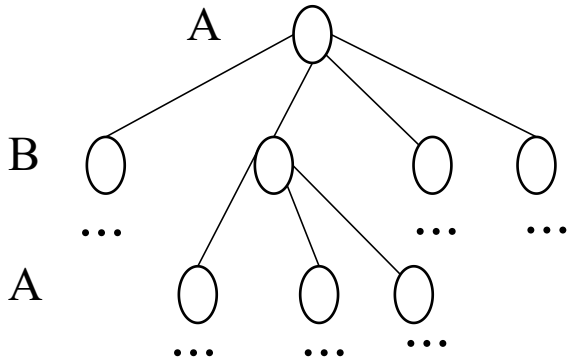
A chaque nœud de l'arbre est associé une valeur. On commence par attribuer des valeurs aux feuilles.(+1 si A gagne, -1 si A perd, 0 si nul)

Les valeurs sont propagées vers les noeuds ascendants, jusqu'à arriver à la racine de la façon suivante:

- si c'est au tour de A (Maximisant)de jouer, le nœud correspondant prend la plus grande des valeurs des ses fils(le coup le plus profitable pour A)

- si c'est au tour de B (Minimisant) de jouer, le nœud correspondant prend la plus petite des valeurs des ses fils.(le coup le plus profitable pour B)

Exploration systématique de graphes



Principe du MIN-MAX

Si la racine prend comme valeur 1, le joueur A gagne s'il ne fait pas d'erreurs.

Si la racine prend la valeur -1, le joueur A perd si B ne fait pas d'erreurs.

Si la racine prend la valeur 0, aucun des deux joueurs n'a de stratégie gagnante, mais tous deux peuvent s'assurer, au pire, d'un match nul en jouant aussi bien que possible

Exploration systématique de graphes

Exemple 1 : Tic-Tac-Toe(jeu des X et des O)

Consiste à placer des X et des O sur une grille de 9 cases, jusqu'à ce que l'un des joueurs aligne trois de ses symboles.

Partant d'une situation donnée j et c'est au tour du joueur X (Maxmisant) de jouer par exemple.

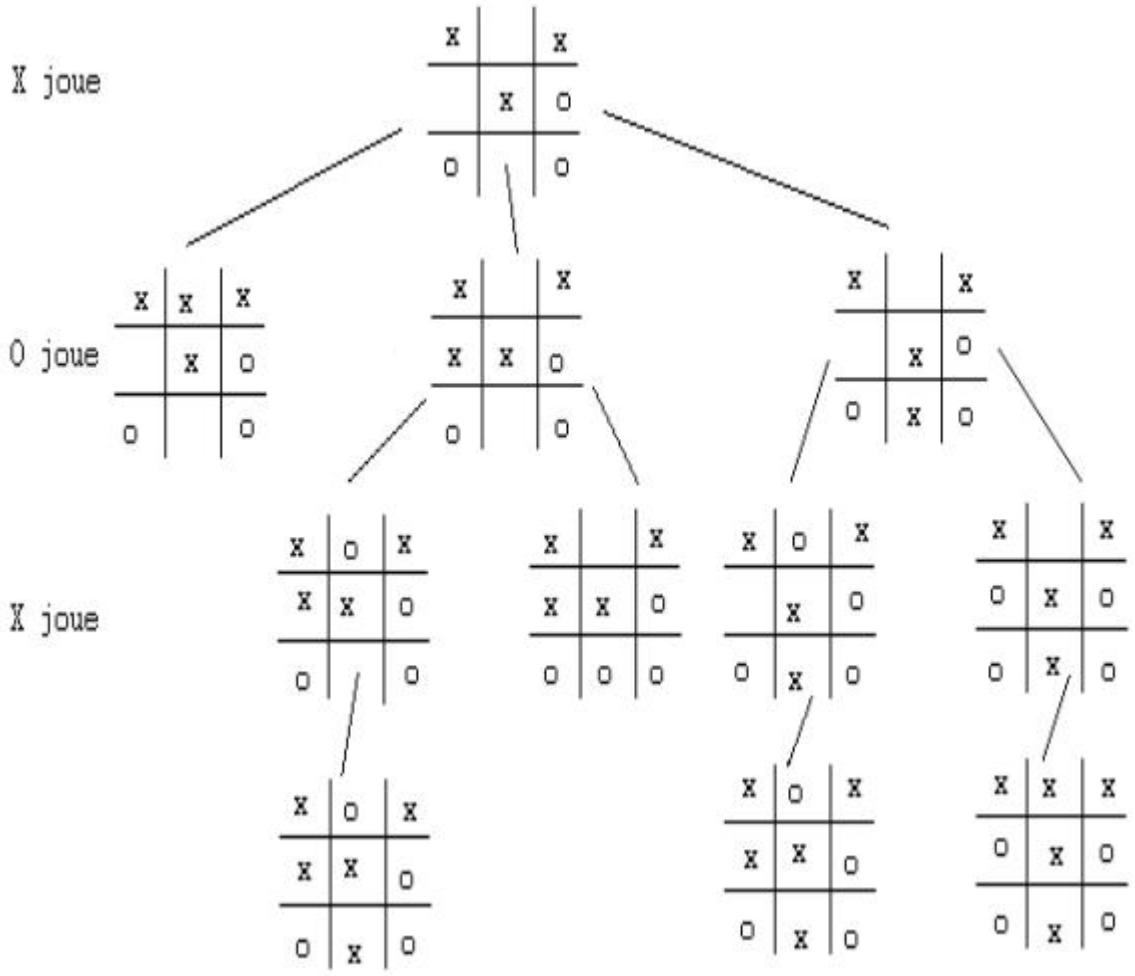
Quel est le bon choix ?

Il faut donc calculer le MinMax pour se décider.

Donc X avant de jouer doit évaluer les fils et prendre le max.

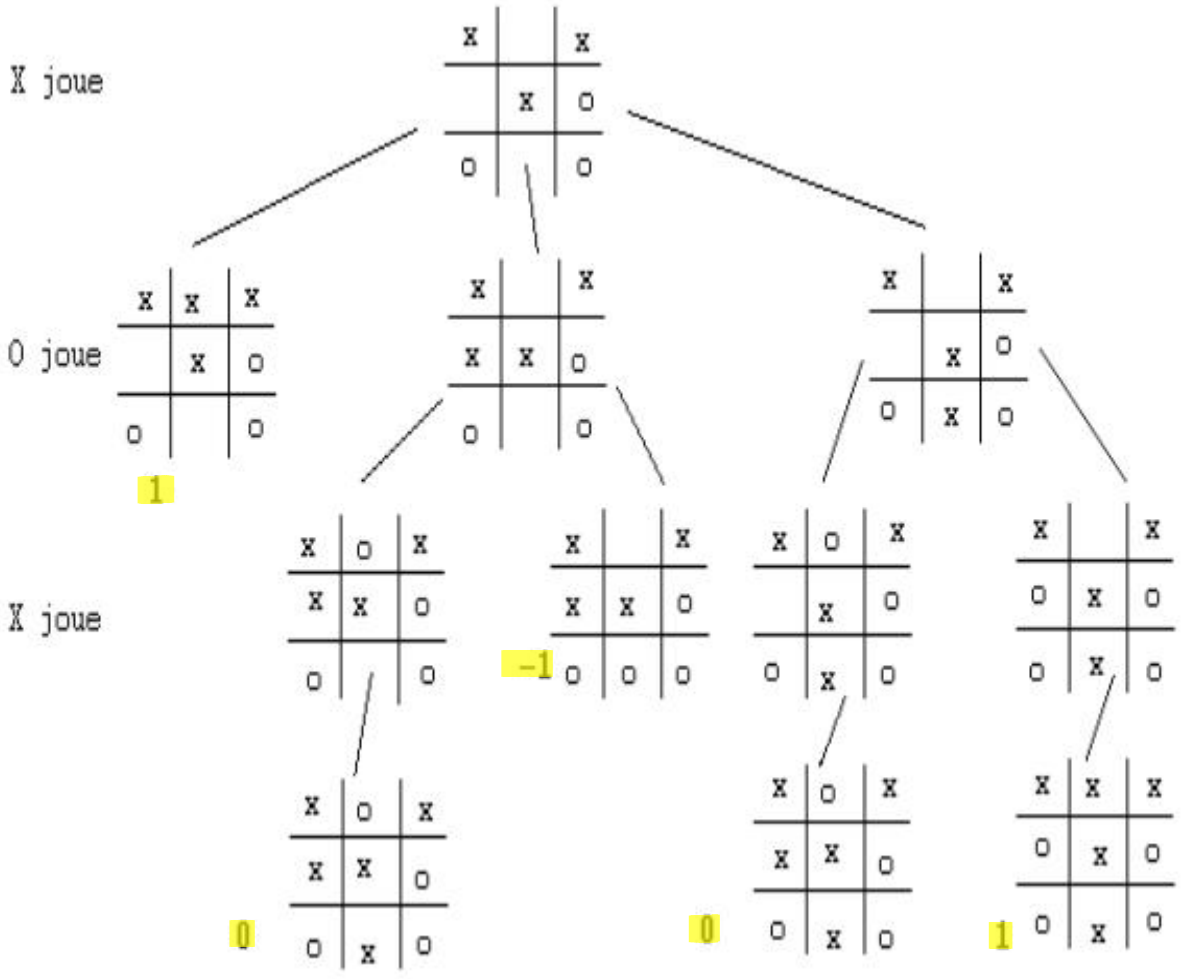
Exploration systématique de graphes

Exemple 1 : Tic-Tac-Toe (jeu des X et des O)



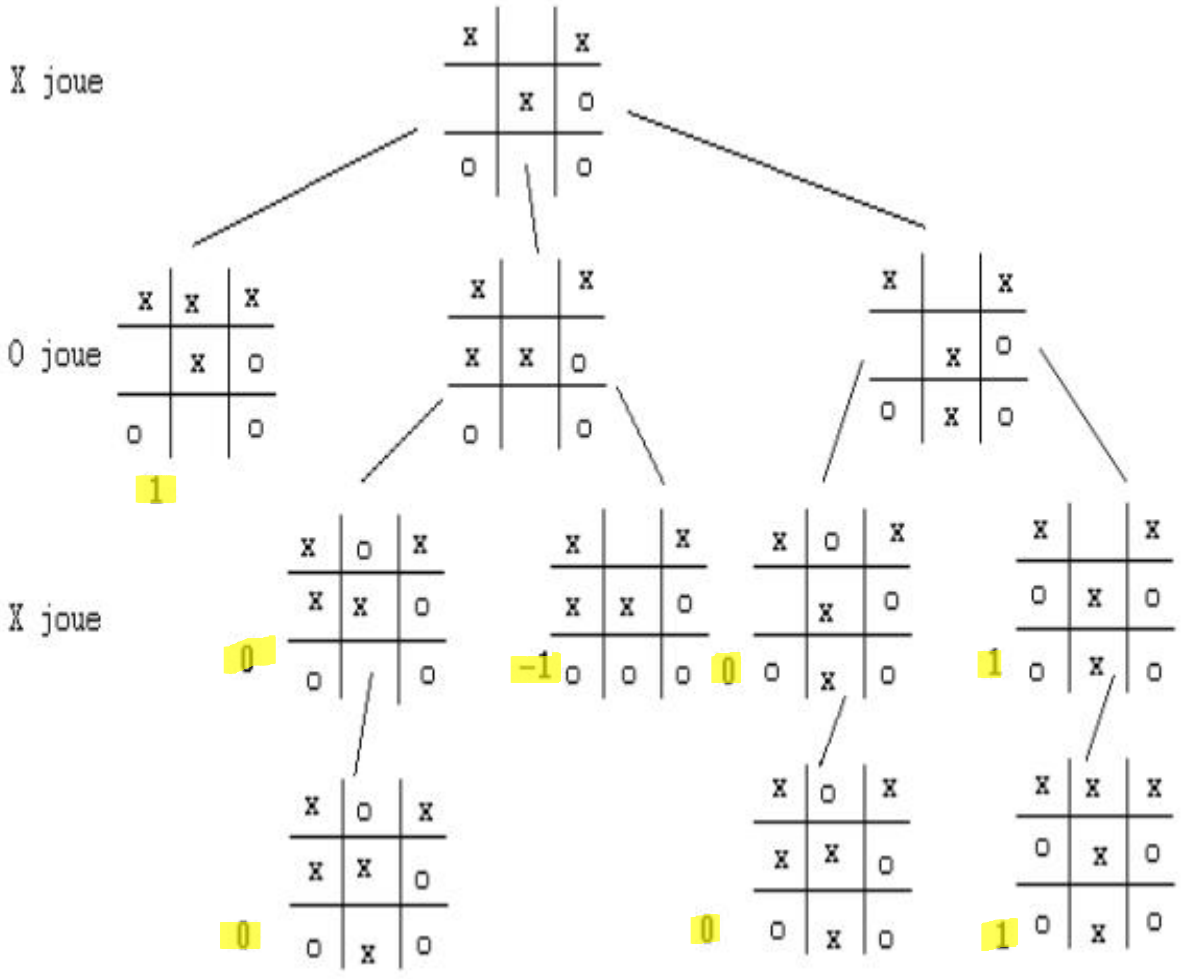
Exploration systématique de graphes

Exemple 1 : Tic-Tac-Toe (jeu des X et des O)



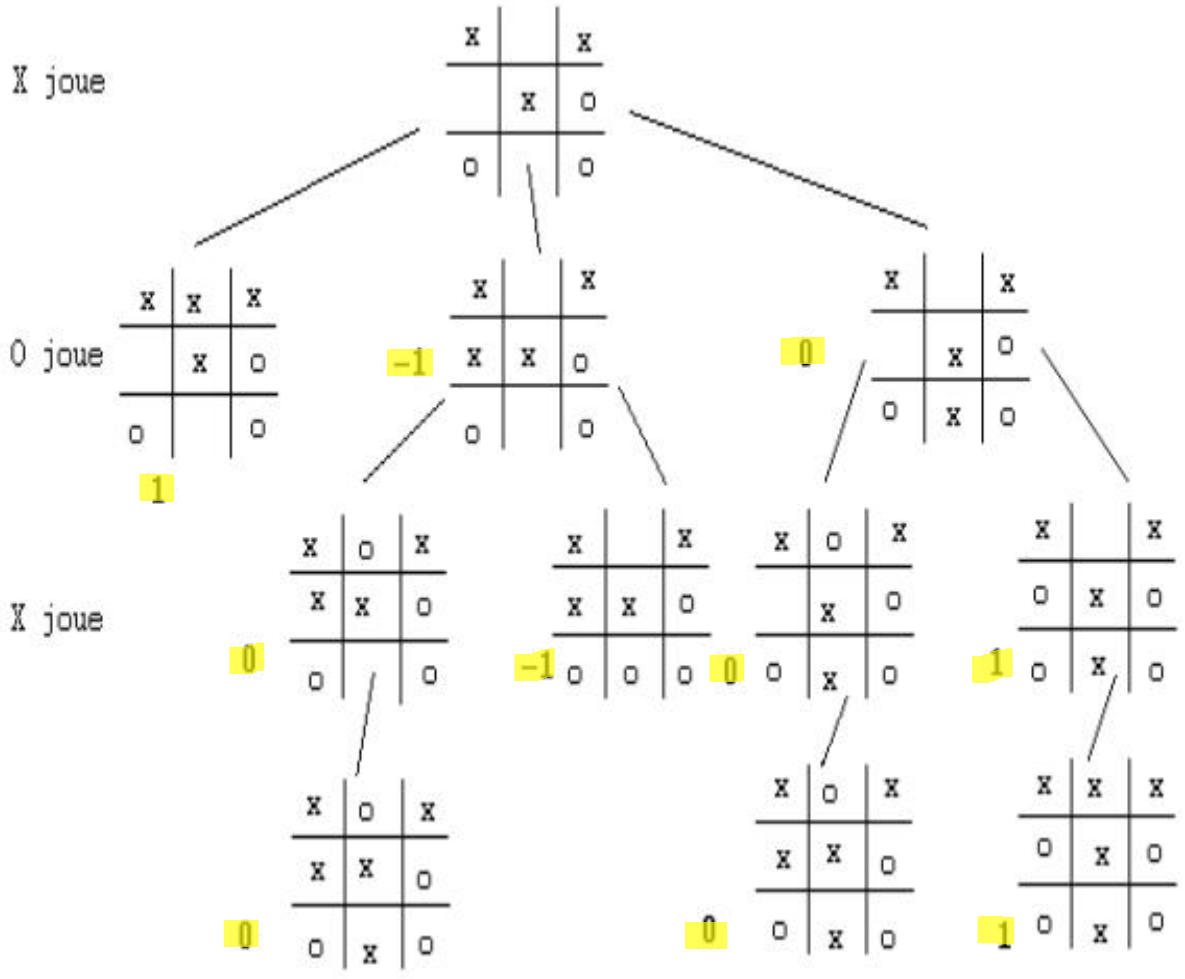
Exploration systématique de graphes

Exemple 1 : Tic-Tac-Toe (jeu des X et des O)



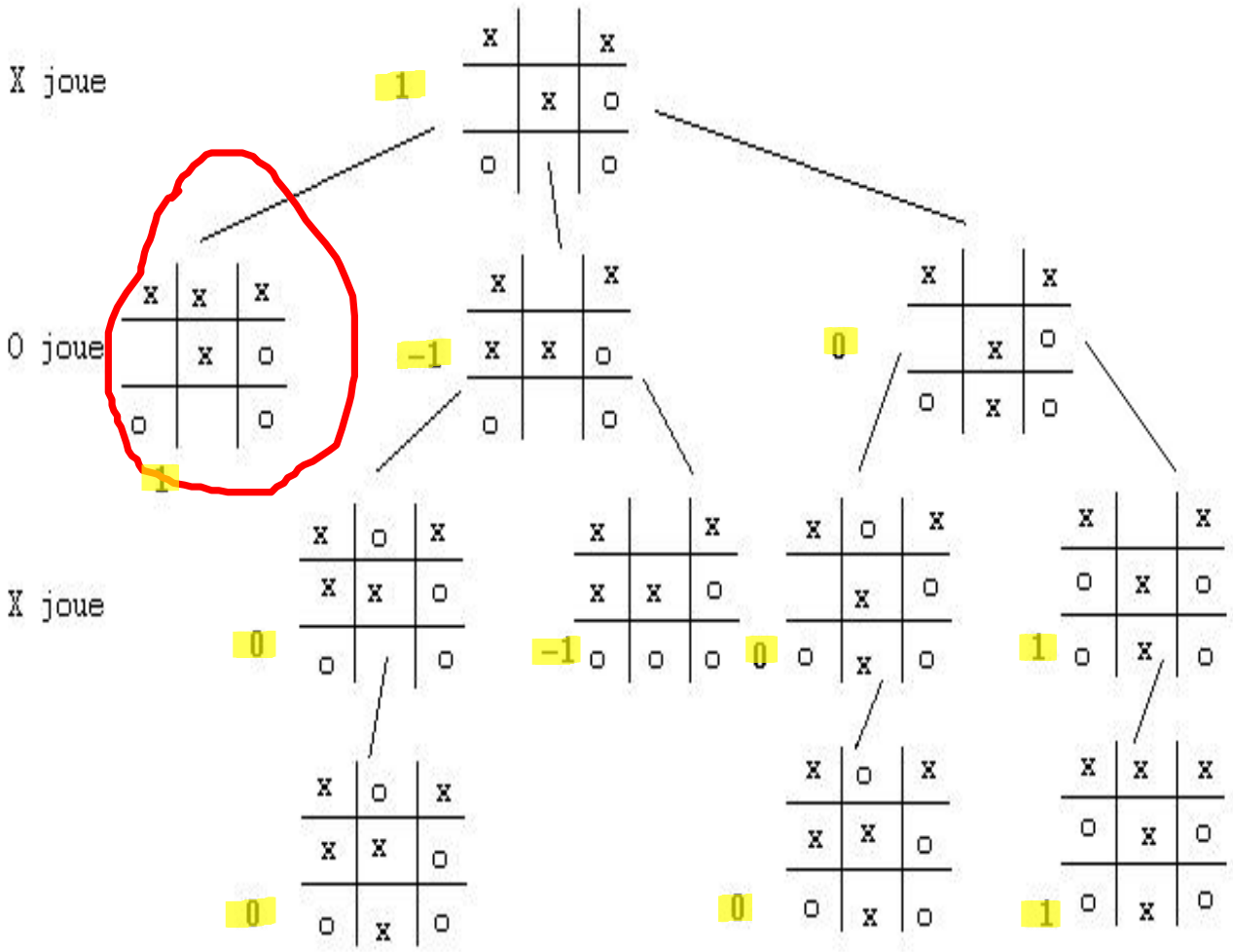
Exploration systématique de graphes

Exemple 1 : Tic-Tac-Toe (jeu des X et des O)



Exploration systématique de graphes

Exemple 1 : Tic-Tac-Toe (jeu des X et des O)



Exploration systématique de graphes

Exemple 2 : Jeu des billes

1. On dispose de n billes sur une table.
2. A prend un nombre de billes entre 1 et k. S'il reste 0 bille A gagne.
3. B prend un nombre de billes entre 1 et k. S'il reste 0 bille B gagne.
4. Sinon allera 2.

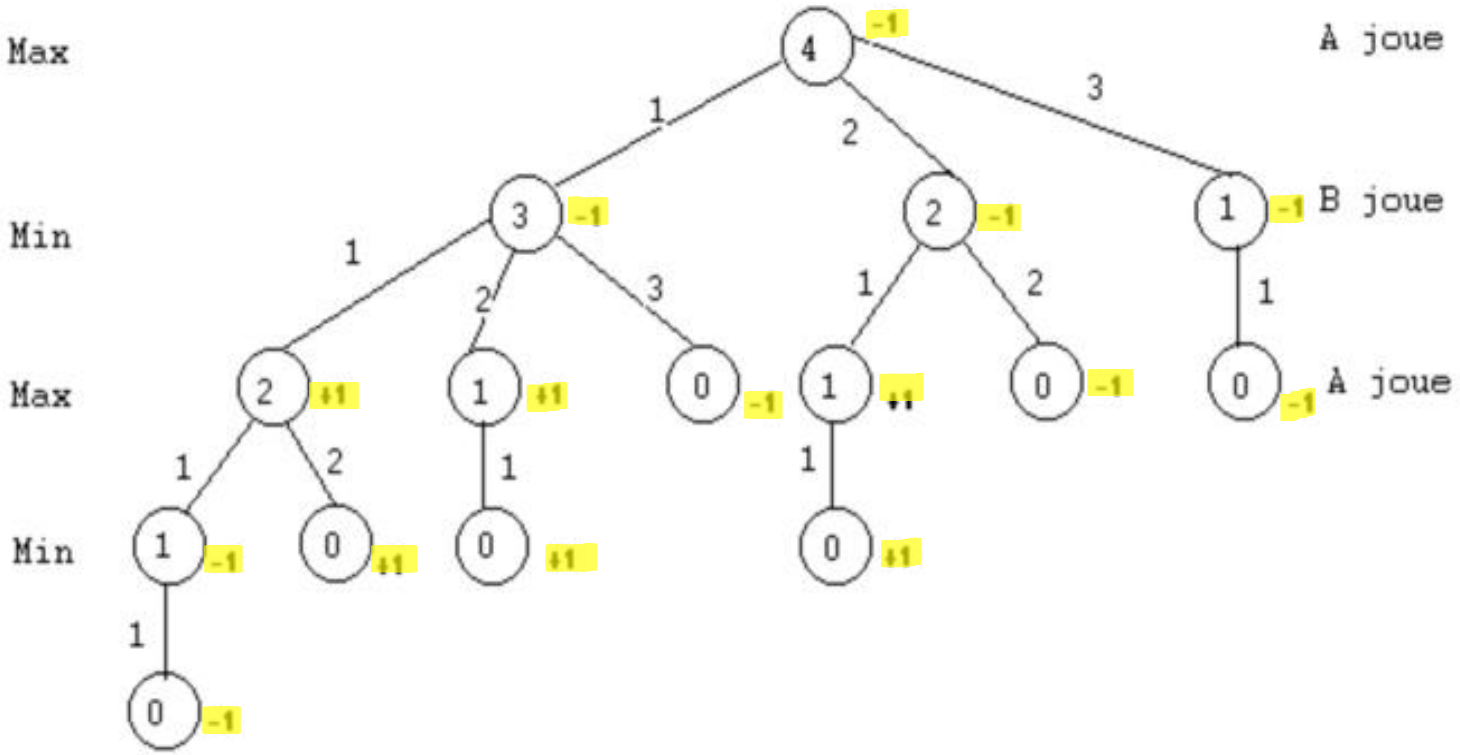
Pour n=4 et k=3, le joueur B a une stratégie gagnante

Pour n=4 et k=2 le joueur A a une stratégie gagnante.

Exploration systématique de graphes

Exemple 2 : Jeu des billes

Pour n=4 et k=3, le Joueur B a une stratégie gangante



Exploration systématique de graphes

Algorithme du MIN-MAX

Utilise un parcours d'arbre post-fixé.

Fonction Minmax(j, mode)

évalue le coût de la situation j, sachant que si mode = Max c'est au premier joueur de jouer et au second si mode = Min. La fonction retourne le coût de la situation.

```
Fonction Minmax(j, mode)
 Si j est une feuille :   Retourner le coût de j
 Sinon
  Si mode = Max  :     Val := - infini
  Sinon Val := + infini Fsi
  Pour chaque fils K du jeu j:
   Si mode = Max
            Val := Max(Val,Minmax(K,Min)
   Sinon
            Val := Min(Val,Minmax(K, Max)
   Fsi
  Finpour
  Retourner(Val)
 Fsi
```

Complexité $O(b^d)$, b:degré maximal et d : profondeur

Exploration systématique de graphes

Algorithme du MIN-MAX

Remarque 1 :

Si l'arbre de jeu n'est pas très grand, le construire et le garder totalement en mémoire centrale. Donc éviter de calculer à chaque fois le MinMax des nœuds

Remarque 2 :

Les valeurs attribuées aux nœuds peuvent être généralisées à des nombres quelconques (appelés coûts).

Comme exemple : jeu des échecs (Arbre de jeu trop grand).

--> Profondeur fixée en fonction de la capacité de calcul de la machine.

--> Utilisation d'une fonction d'estimation des coûts selon la disposition de l'échiquier(nombre de pièces, densité de défense autour du roi,...)

[la probabilité que la machine gagne à partir de cette position.]

Exploration systématique de graphes

Principe de l'élagage Alpha-Béta (coupe de branche " pruning")

Par une simple remarque, on peut éliminer une grande partie des arbres de recherche en ignorant certains descendants d'un nœud.

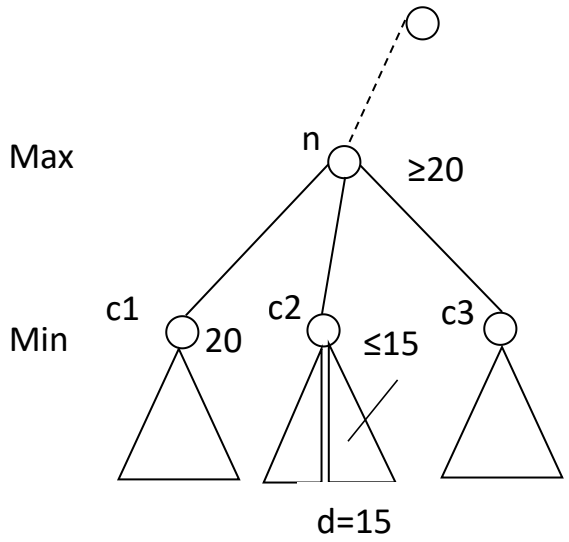
1) En parcourant tous les descendants de c1, on trouve que c1 a une valeur égale à 20.

2) Avant d'explorer les prochains fils de n, on peut déjà dire que n >= 20. (car n est un nœud maximisant)

3) En explorant c2 et ses fils, on tombe sur un nœud d de valeur 15.

4)Comme d est un fils de c2, on aura c2 <= 15(nœud minimisant)

5) A ce niveau, on peut abandonner l'exploration des autres fils de c2 (Coupe)

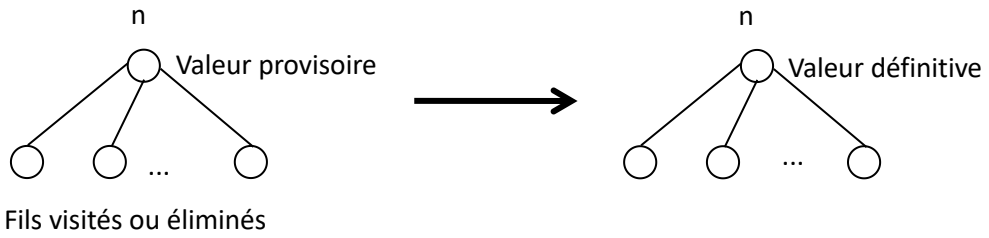


Passer directement à c3.

Exploration systématique de graphes

Principe de l'élagage Alpha-Béta (coupe de branche " pruning")

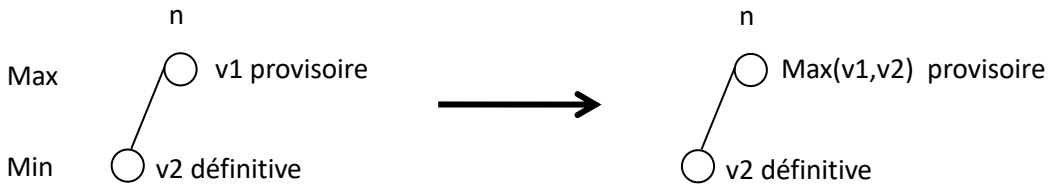
Règle 1. Si tous les enfants d'un nœud n ont été examinés ou éliminés, transformer la valeur de n (jusqu'ici provisoire) en une valeur définitive.



Exploration systématique de graphes

Principe de l'élagage Alpha-Béta (coupe de branche " pruning")

2. Si un nœud maximisant n a une valeur provisoire v1 et un fils de valeur définitive v2, donner à n la valeur provisoire Max(v1, v2).

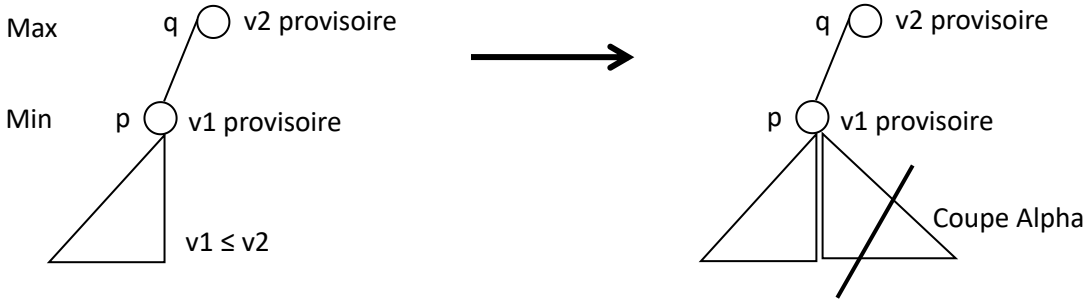


Si n est minimisant on lui donne la valeur provisoire (dans les mêmes conditions) Min(v1, v2).

Exploration systématique de graphes

Principe de l'élagage Alpha-Béta (coupe de branche " pruning")

3. Si p est un nœud minimisant de père q maximisant avec des valeurs provisoires respectives v1 et v2 avec v1 <= v2, alors ignorer toute la descendance encore inexplorée de p (coupe de type Alpha).



Une coupe de type Béta est définie de manière analogue dans le cas où p est maximisant et q minimisant et v1 >= v2.

Exploration systématique de graphes

Algorithme du MIN-MAX avec élagage Alpha-Béta

Initialisation
Alpha = - infini
Béta = + infini

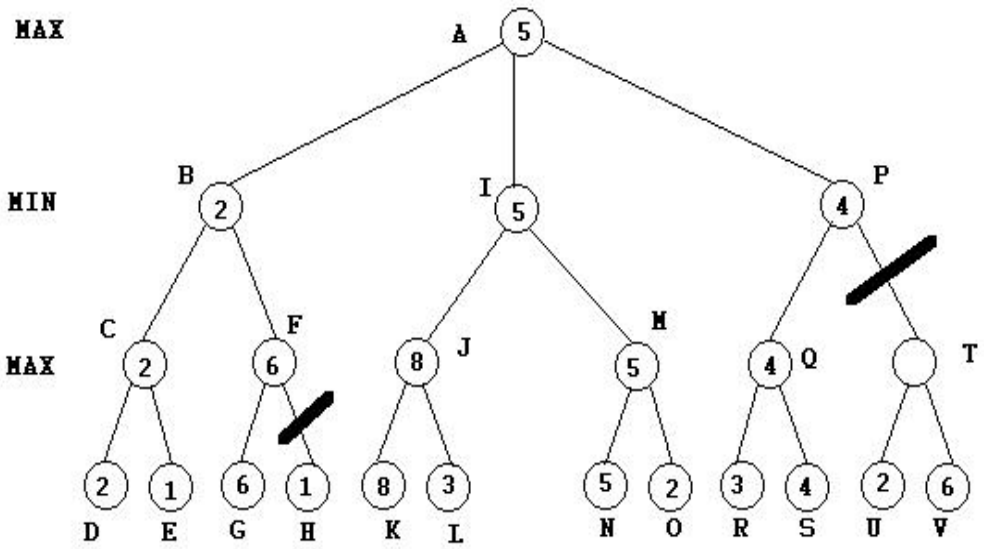
Complexité $O(b^{d/2})$,
b:degré maximal et d : profondeur

```
Fonction Minmax(j, mode, Alpha, Béta)
 Si j est une feuille :   Retourner le coût de j
 Sinon
  Si mode = Max  :     Val := - infini
  Sinon Val := + infini Fsi
  Pour chaque fils K du jeu j:
   Si mode = Max
              Val := Max(Val,Minmax(K,Min, Alpha,Béta)
              Si Alpha >= Val Retourner(Val) Fsi
              Beta := Min(Beta, Val)
   Sinon
              Val := Min(Val,Minmax(K, Max, Alpha, Béta)
              Si  Béta <= Val Retourner(Val) Fsi
              Alpha := Max(Alpha, V)
   Fsi
  Finpour
  Retourner(Val)
Fsi
```

Exploration systématique de graphes

Principe de l'élagage Alpha-Béta (coupe de branche " pruning")

Exemple :



•<u>Scénario :</u>

Noeud visité en postordre : D avec valeur définitive 2

R2 : V(C) = 2 provisoire

Noeud visité E avec valeur définitive 1

R1 : V(C) = 2 définitive

R2 : V(B) = 2 provisoire

Noeud visité G de valeur 6

R2 : V(F) = 6

R3 : Coupe Béta sur la branche H

R2 : V(A) = 2 provisoire

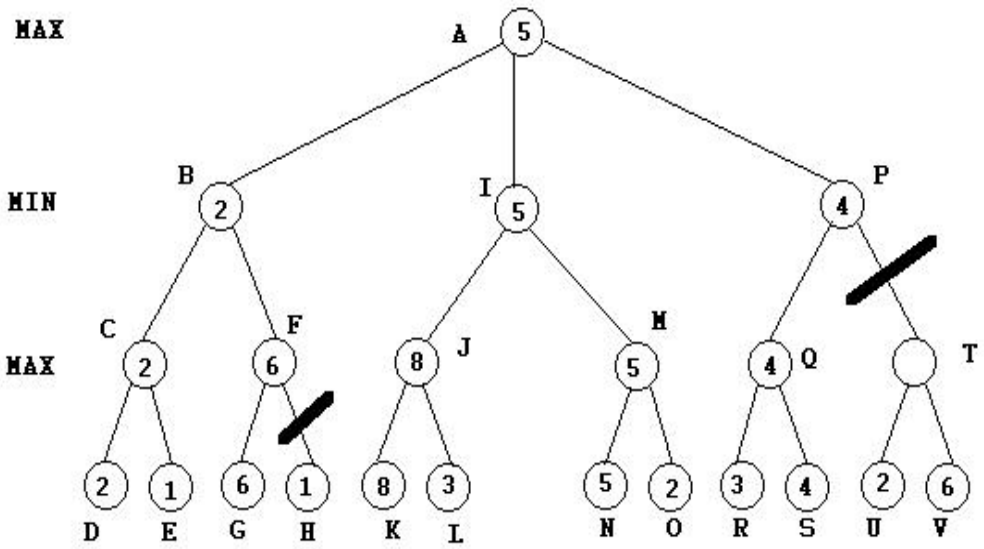
Noeud visité K de valeur 8

R2 : V(J) = 8 provisoire

Exploration systématique de graphes

Principe de l'élagage Alpha-Béta (coupe de branche " pruning")

Exemple :



Noeud visité L de valeur 3
R1 : V(j) = 8 définitive
R2 : V(I) = 8 provisoire
Noeud visité N de valeur 5
R2 : V(M) = 5 provisoire
Noeud visité O de valeur 2
R1 : V(M) = 5 définitive
R1 : V(I) = 5 définitive
R2 : V(A)=5 provisoire =Max(2,5)
Noeud visité R de valeur 3
R2 : V(O) = 3
Noeud visité S de valeur 4
R1 : V(Q) = 4
R2 : V(P) = 4 provisoire
R3 : Coupe Alpha sur branche TUV

Exploration systématique de graphes

Exploration en largeur

Si le graphe est très grand(ou infini) ou si l'on veut plutôt chercher une solution avec peu de manipulations, on utilise un parcours en largeur.

Exemple : Passage de 15 à 4 en appliquant un nombre minimal de fonctions f et g définies par f(x) = 3x et g(x) = x div 2.

$4 = gfg^2(15)$