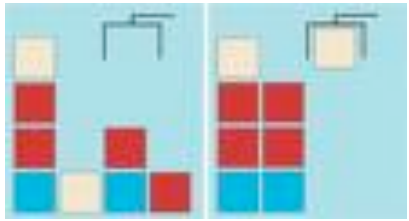
1. Résolution de problèmes



Problèmes et espaces de pb

- Classe de pbs : pb dont on dispose d'une modélisation ou d'une description formelle

- Fournir la description formelle d'un problème
  - Définir un espace d'états contenant toutes les configurations possibles des objets utiles
  - Etats initiaux
  - Etats finaux (buts)
  - Ensemble de règles décrivant les actions (opérateurs)

Exemple du jeu d'échecs

- Espace d'états : toutes les configurations licites
- Etat initial : configuration de l'échiquier au départ
- Etat final : toute configuration en position d'échec et mat
- Règles du type

  pion blanc en case(i,2), et case(i,3) est vide, et case(i,4) est vide
  → déplacer pion blanc de case(i,2) vers case(i,4)

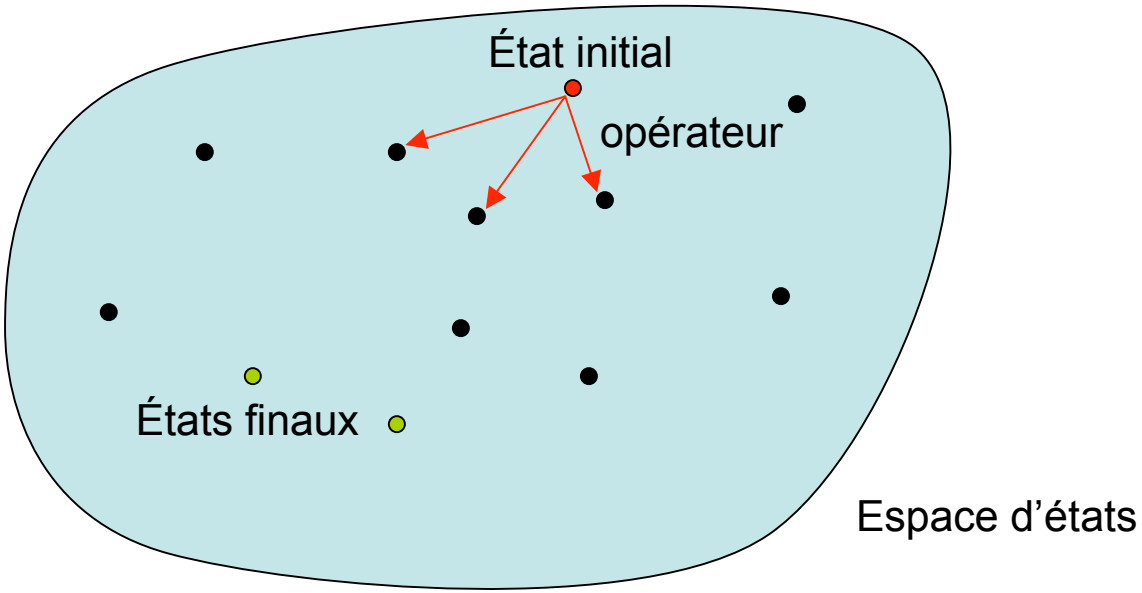
Graphe orienté

Chaque sommet est un état du pb

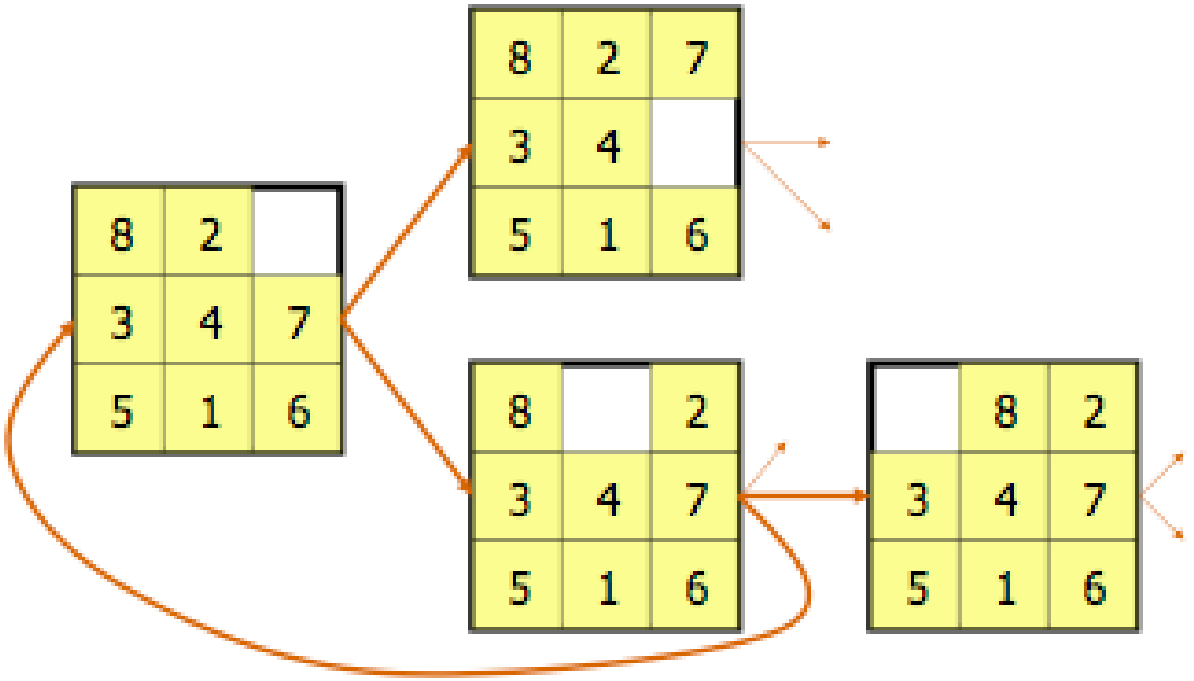
∃ Arc entre l'état u et l'état v ssi un opérateur permet de transformer u en v

- Avec circuit : deux opérateurs consécutifs peuvent annuler leur effet (exemple du taquin)
- Avec cycle : possibilité d'atteindre le même état de différentes façons (exemple des 8 reines)
- Non connexe : si les états terminaux ne sont pas dans la même composante connexe que l'état initial, alors pas de solution (exemple du taquin)

Graphe à états



Exemple du taquin

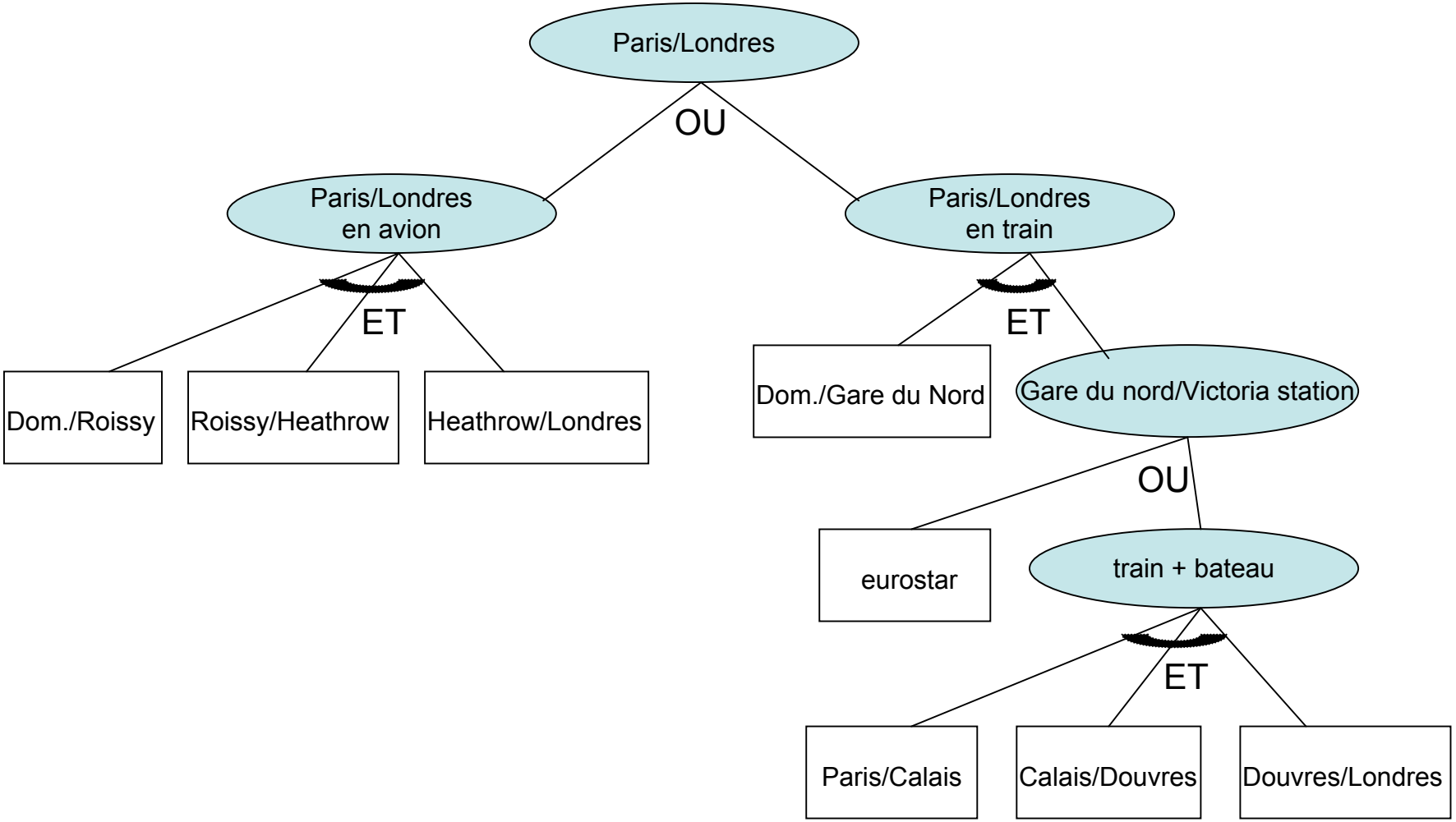


Graphe ET/OU

- Décomposition du pb en plusieurs sous-pbs
- Résolution du pb global = résolution de tous les sous-pbs

  → représentation par graphe ET/OU
- Bien adapté si l'ordre des actions à accomplir n'est pas déterminant
- Espace d'états plus aproprié si la solution est une suite ordonnée d'opérations à accomplir
- Dans certains cas, les 2 représentations sont possibles

Exemple


Paris/Londres
OU
Paris/Londres
en avion
Paris/Londres
en train
ET
ET
Dom./Roissy
Roissy/Heathrow
Heathrow/Londres
Dom./Gare du Nord
Gare du nord/Victoria station
OU
eurostar
train + bateau
ET
Paris/Calais
Calais/Douvres
Douvres/Londres

Processus de résolution
Espace d'états

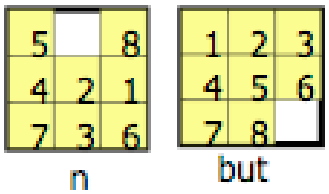
- 2 opérations-clés
  - Opération de développement des alternatives : opérateurs
  - Opération de choix d'une alternative : heuristiques

- Gérer le non-déterminisme en évitant l'explosion combinatoire
  - Pb de terminaison
  - Pb d'admissibilité (fournir la meilleure solution)
  - Pb de complexité

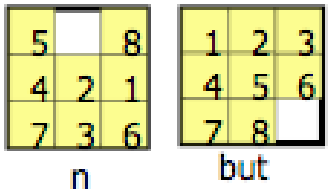
Heuristique

- Estimation du coût d'un état à développer
- Technique destinée à réduire les alternatives et guider les choix non-déterministes de l'algo
- Essentielle pour réduire la complexité de l'algo
- Choix effectués non irrévocables, mais si heuristique efficace, limite les retours arrières
- Généralement dépendante du domaine

Exemple du taquin



H(n) = nb de carrés mal placés
= 6



H(n) = somme des distances (Manhattan)
de chaque carréà sa position finale
= 3+1+3+0+2+1+0+3 = 13

Résolution

- Types de développement d'état
  - Complet : tous les successeurs de l'état sont engendrés et toutes les alternatives issues de cette étape sont considérées
  - Partiel : on se donne une longueur maximale du chemin parcouru formant une possible solution
- Type d'organisation des alternatives
  - LIFO (pile) : recherche en profondeur
  - FIFO (file) : recherche en largeur
  - Recherche ordonnée : ordre de coût croissant, plus intéressante (ordre complet sur toutes les alternatives existantes)

Notations

- k(u,v) : coût des chemins de l'état u à l'état v ; = ∞ s'il n'existe pas de chemin entre u et
- h(u) : évaluation du coût des chemins de u à un état terminal
- g(u) : minimum du coût des chemins connus de u_0 (état initial) à u
- f(u) : h(u) + g(u)
- h*, g* et f* : fonctions optimales

Propriétés de h

La qualité de h est donnée par h(u)-h*(u)

1. h est *presque parfaite* ssi $h(u)<h(v) \Rightarrow h^*(u)<h^*(v)$
2. h est *consistante* ssi $h(u)-h(v) \leq k^*(u,v)$ [si ∃ un chemin de u à v]
3. h est *monotone* ssi $h(u)-h(v)<k(u,v)$ [si $v \in S(u)$]
4. h est *minorante* ssi $h(u) \leq h^*(u)$
5. h est *coïncidente* ssi si $u \in T$, $h(u)=0$

Propriété 1 est trop forte et peu problable.

Consistance et monotonie sont équivalentes.

h minorante ne garantit pas toujours une bonne heuristique.

A* (développement complet + recherche ordonnée)

Données d'entrée : graphe G(U,S) (avec T, u_0, k) et h.
Initialisation P← u_0 ; Q← ∅ ; g(u_0)← 0 ; u← u_0

TantQue [P≠∅ et û∉T] **Faire**
  Supprimer u de P et le mettre dans Q
  Pour chaque {v∈S(u)} **Faire**
    Si [(v∉P∪Q) ou (g(v)>g(u)+k(u,v))]
      Alors g(v) ← g(u)+k(u,v)
        f(v) ← g(v)+h(v)
        père(v) ← u
        Ranger v dans P, dans l'ordre f↑ puis g↓
    FinSi
  FinPour
  Si [P≠∅]
    Alors u← û
  FinSi
FinTantQue

Données de sorties :
Si [P=∅]
Alors le problème n'admet pas de solution
  Sinon fournir la solution chemin (u) = (u_0, ..., père(père(u)), père(u), u)
FinSi

Terminaison de A*

Proposition

- Si tout chemin de longueur infinie a un coût infini,
- Si sur le chemin optimal, la valeur de l'heuristique est bornée,
- Si chaque état a un nombre fini de successeurs,
- Alors l'algorithme A* termine

Admissibilité de A*

Proposition

Si, outre les conditions de terminaison, l'heuristique est minorante (elle sous-estime systématiquement le coût du chemin restant à parcourir : $h(e) \leq h^*(e)$), l'algorithme A* est admissible.

Complexité de A*

N : nombre d'états

- En pire cas : $O(2^N)$
- Si heuristique minorante : $O(N^2)$
- Si heuristique monotone ($\forall$ u, v$\in$S(u) : h(u)-h(v) $\leq$ k(u,v)) : O(N)

- Remarque

  Taquin 3*3 $\rightarrow$ N = 9 ! =362 880

  Rubik's cube $\rightarrow$ N = 4,3 . 10^{19}

Variante de A*

- Si l'heuristique est peu informante, A* devient presque une exploration en largeur
- Une exploration en profondeur ne termine généralement pas (chemins infinis)
- Compromis entre les deux exploration : explorer en profondeur jusqu'à un coût maximal
- Si T ne contient qu'un seul état (ou très peu), si les opérations sont inversibles, alors exploration simultanée à partir de u_0 et de T

A_ε

- Recherche admissible coûteuse : la complexité croit en moyenne exponentiellement à la longueur du chemin trouvé
- Recherche non-admissible (ordonnée par h) ne termine pas toujours
- Compromis entre minimisation du coût de la solution (admissibilité) et minimisation de sa recherche (complexité) : entre admissibilité et non-admissibilité (ε-admissible)
- u est acceptable : $u \in P$ et $f(u) \leq (1+\varepsilon)\, f^*(u_0)$, comme on ne connaît pas $f^*(u_0)$, un estimateur minorant sera utilisé comme seuil d'acceptabilité
- Algorithme A_ε : A_ε développe tout état acceptable, en utilisant une $h_c(u)$: estimation de la longueur du chemin et non son coût (nb itérations)

Processus de résolution Graphe ET/OU H(U,S)

Réduction d'un pb à des sous-pbs plus élémentaires

- H est sans circuit
- U : ensemble des états, u_0 : état initial
- T : ens. des états objectifs
- I(u) : ens. des règles de décomposition applicables à u (succ. d'un OU)
- S(u) : ens. de tous les succ. possibles de u par tous ses connecteurs
- $s_i(u)$: ens. des sous-pbs qui doivent être résolus pour une règle donnée (succ. d'un ET)
- $S^{-1}(u)$: ens. de tous les états dont u est successeur
- k(u,i) : coût du connecteur $s_i(u)$
- $f_i(u)$ = estime le coût de la meilleure solution courante utilisant $s_i(u)$
- $f_i(u) = k(u,i) + \Sigma_{v \in si(u)} f(v)$
- $f_i(u) = f(u) = \min \{f_i(u) \mid i \in I(u)\}$, î(u) : pointeur vers le connecteur minimal
- Pour un état non encore développé, f(u) = h(u)
- $G(u_0)$: meilleure solution partielle connue à l'instant courant (c'est un graphe)
- P : ens. des états pendants ; Q : états déjà développés
- A : ens. des états dont l'estimation f est actualisé
- I'(u) : ens. des connecteurs $s_i(u)$ dont l'estimation f(u) devra être mise à jour

AO* (développement complet + recherche ordonnée)

3 phases

1. Développement d'un état pendant dans la solution courante partielle $G(u_0)$
2. Mise à jour ascendante dans H de l'estimation f et du pointeur î sur le connecteur minimal pour tous les ascendants de l'état u développé.
   L'ordre ascendant de la mise à jour (on choisit dans A un état sans succ. dans A) et l'absence de circuit dans H garantissent que chaque état sera mis à jour 1! fois.
3. Définition de la nouvelle meilleure solution partielle et choix d'un de ses états pendants u sur lequel la recherche se poursuivra

AO*

Données d'entrée

Graphe H(U,S) (avec T, u_0, k) et h.

Initialisation

P← u_0 ; Q← ∅ ; f(u_0)← h(u_0) ; u← u_0

1. TantQue [u ≠ nil et f(u_0) ≠ ∞] **Faire**

Supprimer u de P et le mettre dans Q

Pour chaque {i∈I(u)} **Faire**

Pour chaque {v∈s_i(u) | v∉P∪Q} **Faire**

P←P∪{u}

f(v)←h(v)

FinPour

f_i(u) ← k(u,i) + $\Sigma_{v \in si(u)}$f(v)

FinPour

Si [I(u)=∅]

Alors f(u)←∞

Sinon f(u) ← min {f_i(u) | i∈I(u)}

î(u) ← indice du min

FinSi

2. A←S^{-1}(u)

Pour chaque {w∈A} **Faire**

I'(w) ← {i∈I(w) | u∈s_i(w)}

FinPour

3. TantQue [A≠∅] **Faire**

Prendre et supprimer de A un état v tq A∩S(v)=∅

Pour chaque {i∈I'(v)} **Faire**

f_i(v) ← k(v,i) + $\Sigma_{w \in si(v)}$f(w)

FinPour

Si $f_{\hat{i}}$(v) ≠ min {f_i(v) | i∈I(v)}

Alors î(v) ← indice du nouveau min

FinSi

Si f(v) ≠ $f_{\hat{i}}$(v)

Alors f(v) ← $f_{\hat{i}}$(v)

A ← A∪S^{-1}(v)

Pour chaque {w∈S^{-1}(v)} **Faire**

I'(w) ← I'(w) ∪ {i∈I(w) | u∈s_i(w)}

FinPour

FinSi

I'(v) ← ∅

FinTantQue

G(u_0) ← Définir-Solution (u_0), u ← Choisir-Etat-Pendant (G(u_0))

FinTantQue

AO*

Données de sorties :
Si [$f(u_0) = \infty$]
Alors le problème n'admet pas de solution
    Sinon fournir la solution $G(u_0)$ de coût $f(u_0)$
FinSi

Définir-Solution (u_0) : engendre $G(u_0)$ par un parcours descendant dans H, à partir de u_0, et en suivant les connecteurs marqués î

$B \leftarrow u_0$
$G(u_0) \leftarrow \{u_0, \varnothing\}$ graphe réduit au seul état u_0
TantQue [$B \neq \varnothing$] **Faire**
    Prendre u en tête de B et le supprimer de B
    Pour chaque $\{v \in s_i(u)\}$ **Faire**
        Si $v \notin T \cup P$ et $v \notin G(u)$
            Alors ranger v dans B
        FinSi
        Ajouter à $G(u_0)$ l'état v et l'arc (u,v)
    FinPour
FinTantQue

Terminaison de AO*

- Proposition

  Pour tout graphe ET/OU fini sans circuit et pour toute heuristique h telle que si h(u) = ∞, il n'existe pas de sous-graphe solution de racine u, l'algorithme AO* s'arrête.

- Proposition

  Pour tout graphe ET/OU sans circuit, ne contenant aucun sous-graphe infini (en nb d'états) mais de coût borné, contenant au moins une solution finie, et pour toute heuristique h majorée sur une solution optimale ($\exists$ $G^*(u_0)$ dont tous les noeuds u sont tq $h(u) \leq h$), l'algorithme AO* s'arrête en fournissant une solution.

Propriétés de AO*

- **Admissibilité de AO***

  Si le graphe et l'heuristique vérifient les conditions de la proposition précédente et si l'heuristique est minorante alors AO* est admissible

- **Complexité de AO***

  Pire que pour A* !!!

- **Variante de AO***

  ∃ nombreuses variantes intéressantes, et en particulier AOε qui est ε-admissible

Différences entre A* et AO*

*Dans A**

- un état u dont l'évaluation f(u) a baissé est remis pendant et sera éventuellement redvpé pour rediriger la recherche vers un autre de ses succ.
- tous les chemins partiels de graphe G sont directement accessibles à tout moment de l'algorithme, et les évaluations de leur coût sont connues.

*Dans AO**

- le dvpt d'un état u se fait au plus une fois, mais la mise à jour de son évaluation f(u) et de son pointeur î(u) vers le connecteur minimal $s_{\hat{i}}(u)$ se feront aussi souvent qu'un nouveau descendant de u sera développé.
- un seul sous-graphe partiel est connu à la fois $G(u_0)$ et chaque itération le redéfinit et recalcule son évaluation.

Processus de résolution
Arbre de jeu

- Pb à résoudre = recherche du gain de la partie dans un jeu à 2 joueurs
- A chaque jeu, on associe un arbre de jeu où chaque noeud représente une disposition du jeu
- Feuilles de l'arbre : situations où aucun coup n'est possible

  → 1 des joueurs a gagné ou partie nulle
- A chaque noeud est associée une valeur
- On commence par les feuilles en propageant les valeurs vers la racine
  - prendre le maximum des valeurs des fils aux niveaux où le joueur 1 doit jouer (joueur maximisant)
  - prendre le minimum des valeurs des fils aux niveaux où le joueur 2 doit jouer (joueur minimisant)

MinMax (backtrack search)

Parcours en profondeur postfixé (visite d'un noeud après avoir exploré ses fils)

```
Fonction minmax (j : jeu, mode : chaîne) : réel
Var k : jeu
      valeur : réel
Si j est une feuille
      Alors retourner (coût(j))
      Sinon   Si mode = 'max'
                        Alors valeur ← - ∞
                        Sinon valeur ← + ∞
                FinSi
                Pour chaque {k∈S(j)} Faire
                        Si mode = 'max'
                                Alors valeur ← max(valeur, minmax(k, 'min'))
                                Sinon valeur ← min(valeur, minmax(k, 'max'))
                        FinSi
                FinPour
                Retourner(valeur)
FinSi
```

Coupure α-β

→ Elaguer des branches de l'arbre

Dans l'algo, au niveau du 'pour'

- valeur définitive : valeur d'un noeud
- valeur provisoire : majorant sur la valeur d'un noeud minimisant ou minorant sur la valeur d'un noeud maximisant

Règles

1. Si tous les enfants d'un noeud n ont été examinés ou éliminés, transformer la valeur provisoire de n en valeur définitive.
2. Si un noeud maximisant (resp. minimisant) n a une valeur provisoire v1 et un fils de valeur définitive v2, donner à n la valeur provisoire max(v1, v2) (resp. min(v1, v2).
3. Si un noeud minimisant (resp. maximisant) p, de père q maximisant (resp. minimisant), et si p et q ont des valeurs provisoires respectives v1 et v2, avec $v1 \leq v2$ (resp. $v1 \geq v2$), alors il est possible d'ignorer toute la descendance encore inexplorée de p ; cette coupure est appelée alpha (resp. bêta).