Exploration en situation d'adversité

Jerôme Champavère

jerome.champavere @ lifl.fr

http://www.grappa.univ-lille3.fr/~champavere/?page=Enseignement

Les jeux

Théorie des jeux

- Point de vue économie
  - Environnements multi-agents
  - Compétition ou coopération
- Point de vue IA (simple)
  - Alternés
  - Déterministes
  - Information parfaite
  - Somme nulle

Jeux et intelligence artificielle

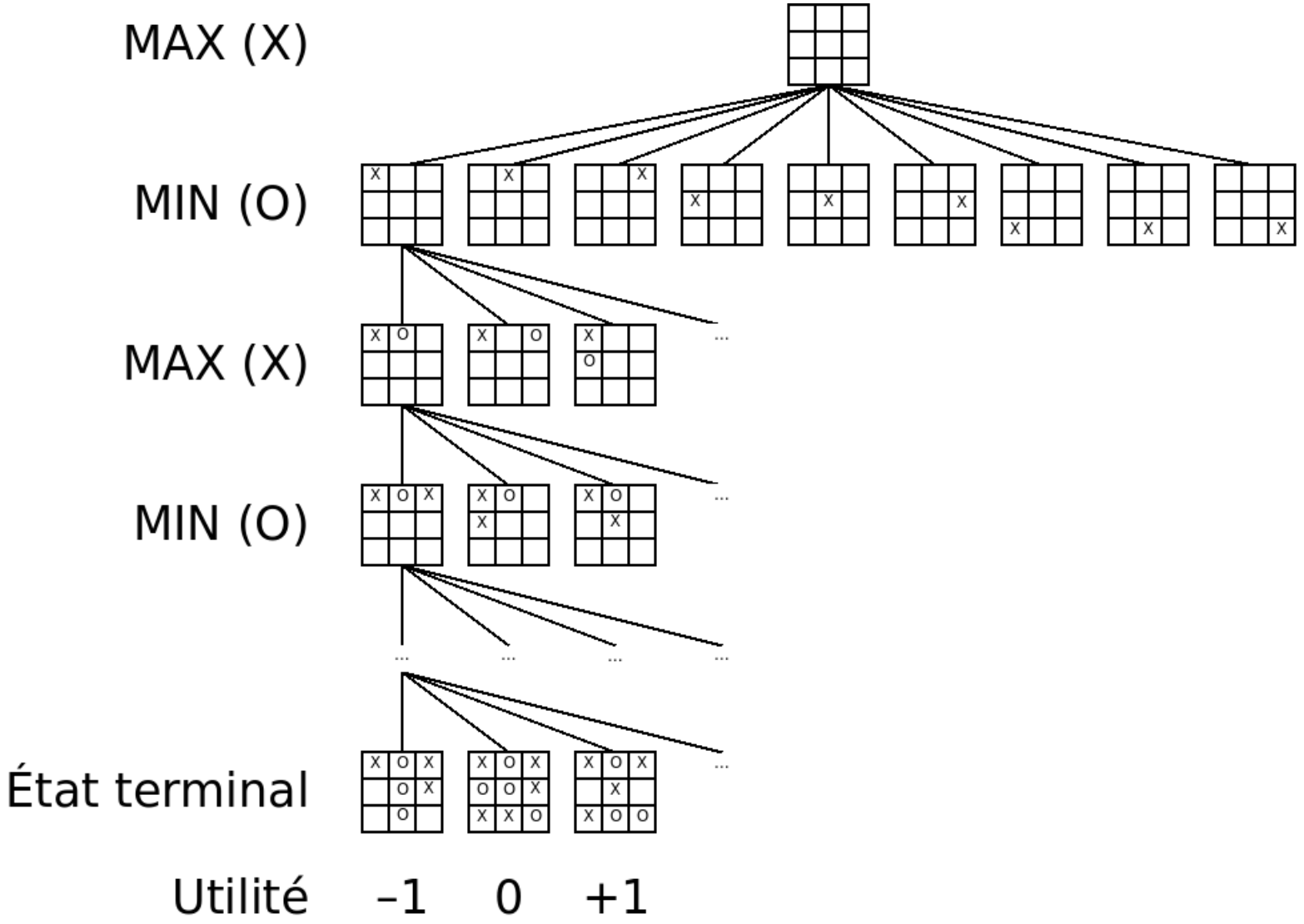
- Domaine historique de l'IA
- Intéressants *parce que* trop difficiles à résoudre
  - Prendre une décision même lorsque le calcul de la décision *optimale* est impossible
  - Inefficacité lourdement pénalisée
  - Quelle est la meilleure utilisation possible du temps ?

Décisions optimales dans les jeux

Problème d'exploration

- Deux joueurs : MAX et MIN
- Formalisation
  - État initial : configuration du jeu au départ
  - Fonction successeur : liste de coups légaux et de leurs résultats
  - Test de terminaison : fin de partie
  - Fonction d'utilité : valeur numérique associée à un état terminal

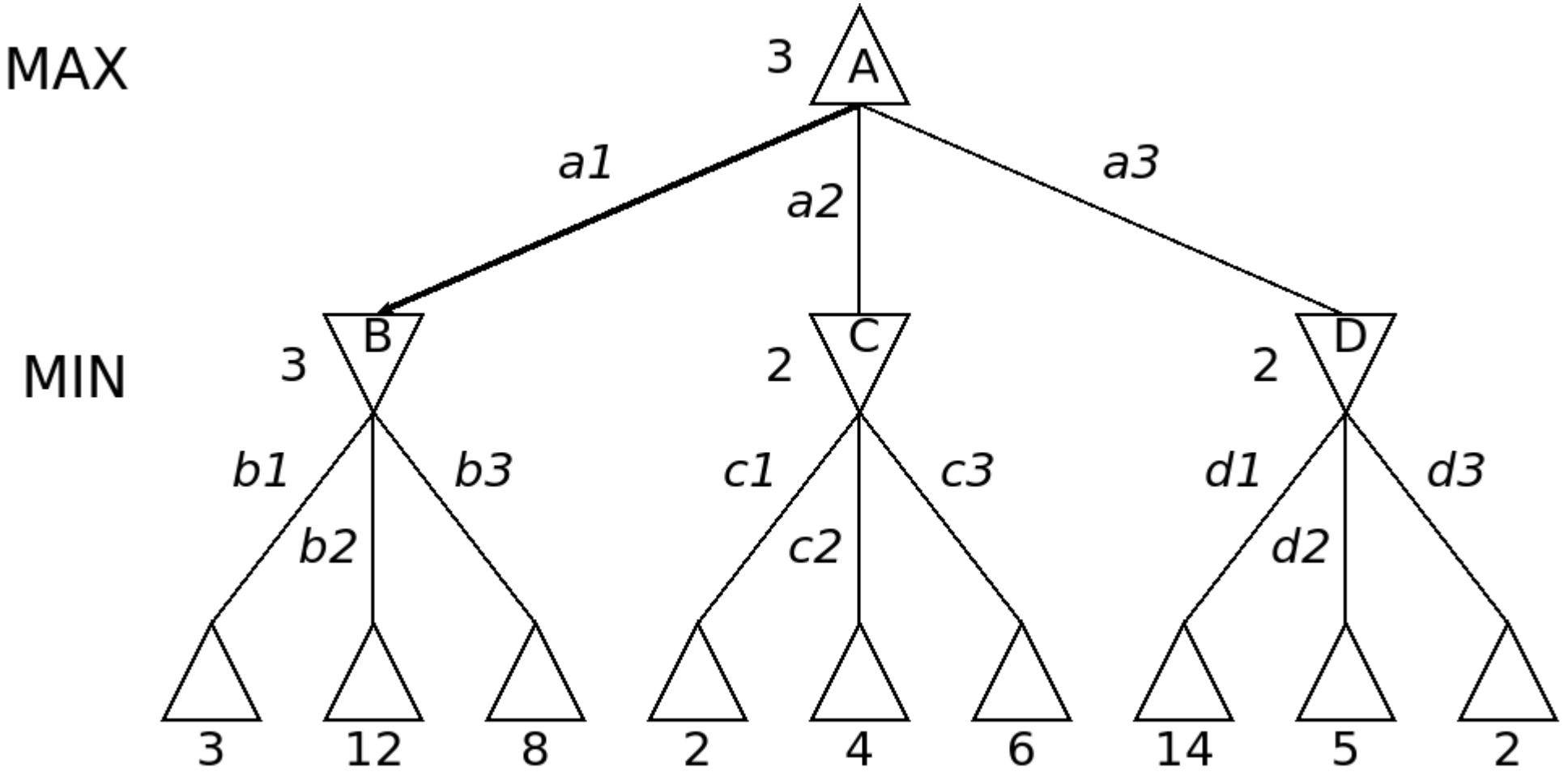
Arbre de jeu



Stratégies optimales

- MIN a la possibilité de réagir
- MAX doit trouver une stratégie contingente
- VALEUR-MINIMAX(n) =
  - UTILITÉ(n) si n est un état terminal
  - $\max_{s \in succ(n)}$ VALEUR-MINIMAX(s) si n est un nœud MAX
  - $\min_{s \in succ(n)}$ VALEUR-MINIMAX(s) si n est un nœud MIN
- Décision minimax à la racine : action qui constitue le choix optimal pour MAX

Arbre d'un jeu à deux demi-coups



Algorithme du minimax

```
function DÉCISION-MINIMAX(état)
    v ← VALEUR-MAX(état)
    return action dans SUCCESSEURS(état) ayant la valeur v

function VALEUR-MAX(état)
    if TEST-TERMINAL(état) then
        return UTILITÉ(état)
    v ← −∞
    for a,s in SUCCESSEURS(état) do
        v ← MAX(v,VALEUR-MIN(s))
    return v

function VALEUR-MIN(état)
    if TEST-TERMINAL(état) then
        return UTILITÉ(état)
    v ← +∞
    for a,s in SUCCESSEURS(état) do
        v ← MIN(v,VALEUR-MAX(s))
    return v
```

Algorithme du minimax

- Exploration en profondeur d'abord complète de l'arbre de jeu
- Complexité en temps : $O(b^m)$
- Complexité en espace : O(bm)

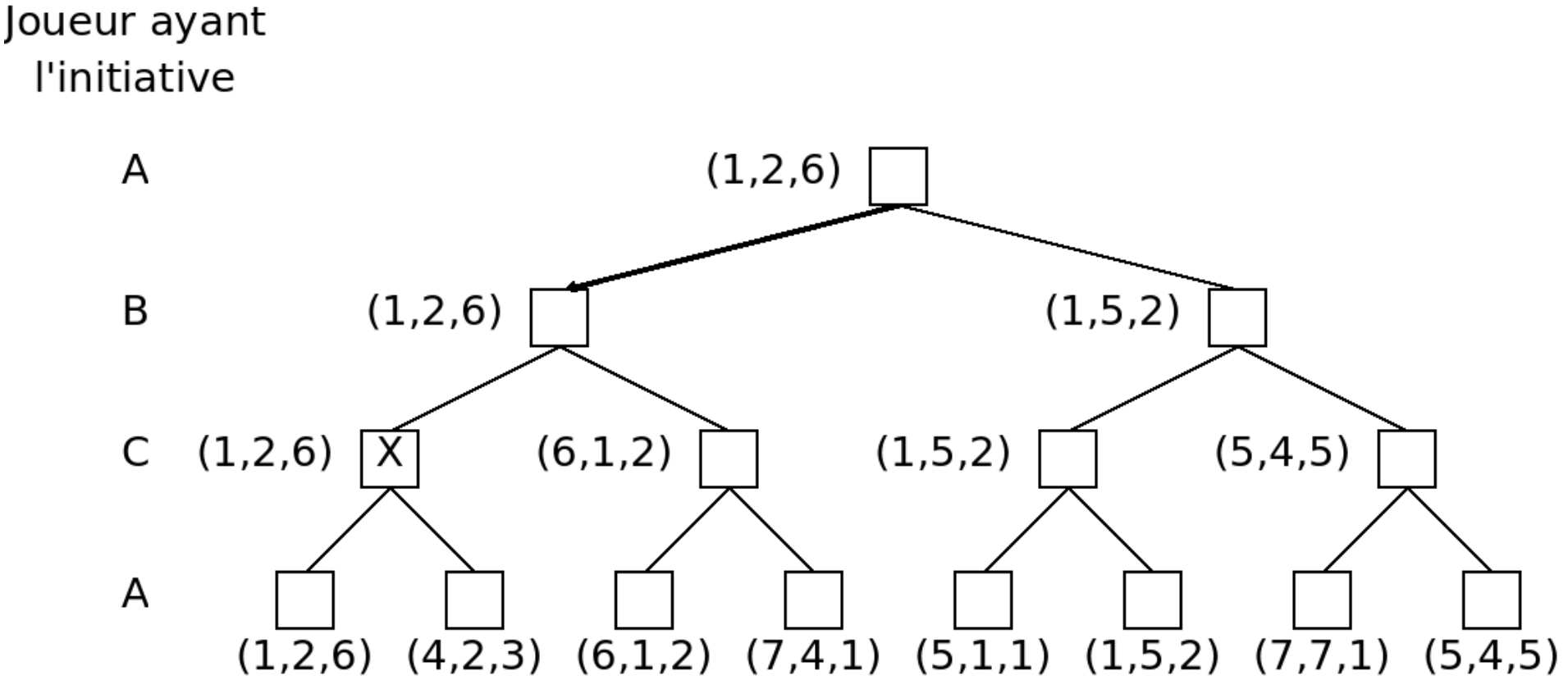
Implémentation negamax

```
function DÉCISION-MINIMAX(état)
    v ← VALEUR-NEGAMAX(état)
    return action dans SUCCESSEURS(état) ayant la valeur v

function VALEUR-NEGAMAX(état)
    if TEST-TERMINAL(état) then
        return UTILITÉ(état)
    v ← −∞
    for a,s in SUCCESSEURS(état) do
        v ← MAX(v,-VALEUR-NEGAMAX(s))
    return v
```

Jeux multi-joueurs

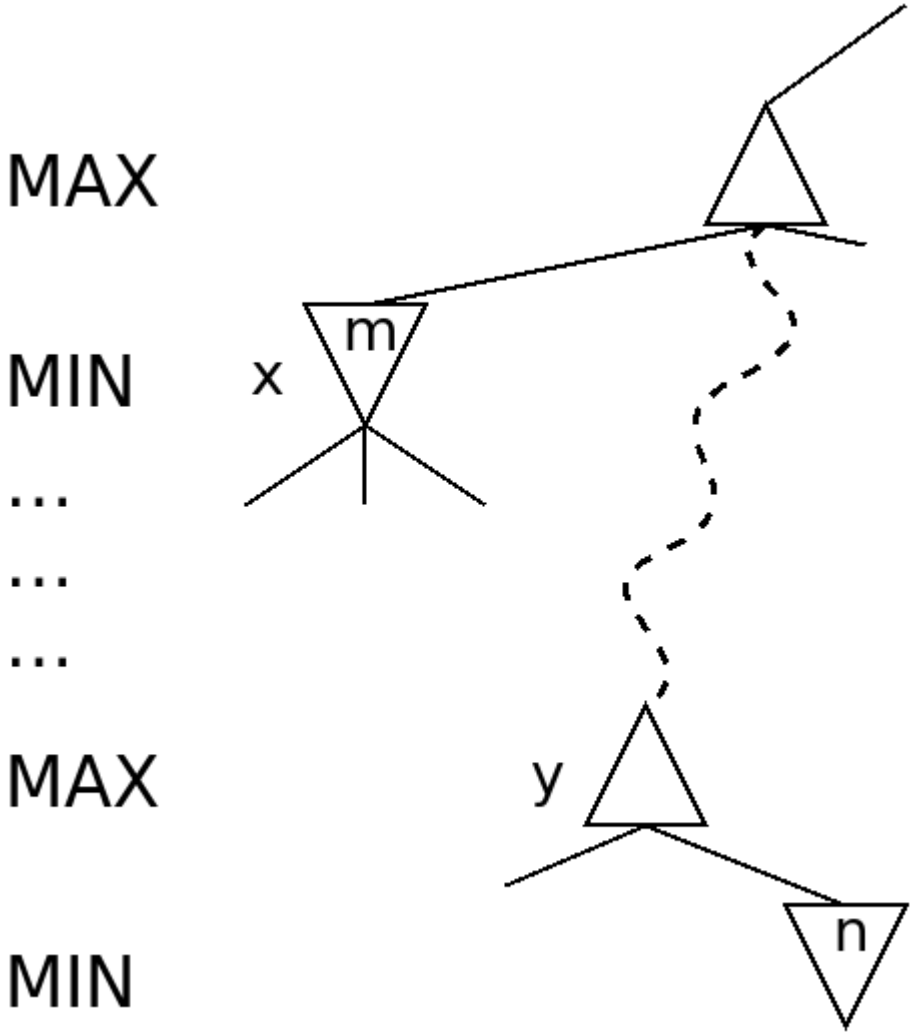
- Vecteur de valeurs
- Possibilité d'alliances



Élagage alpha-bêta

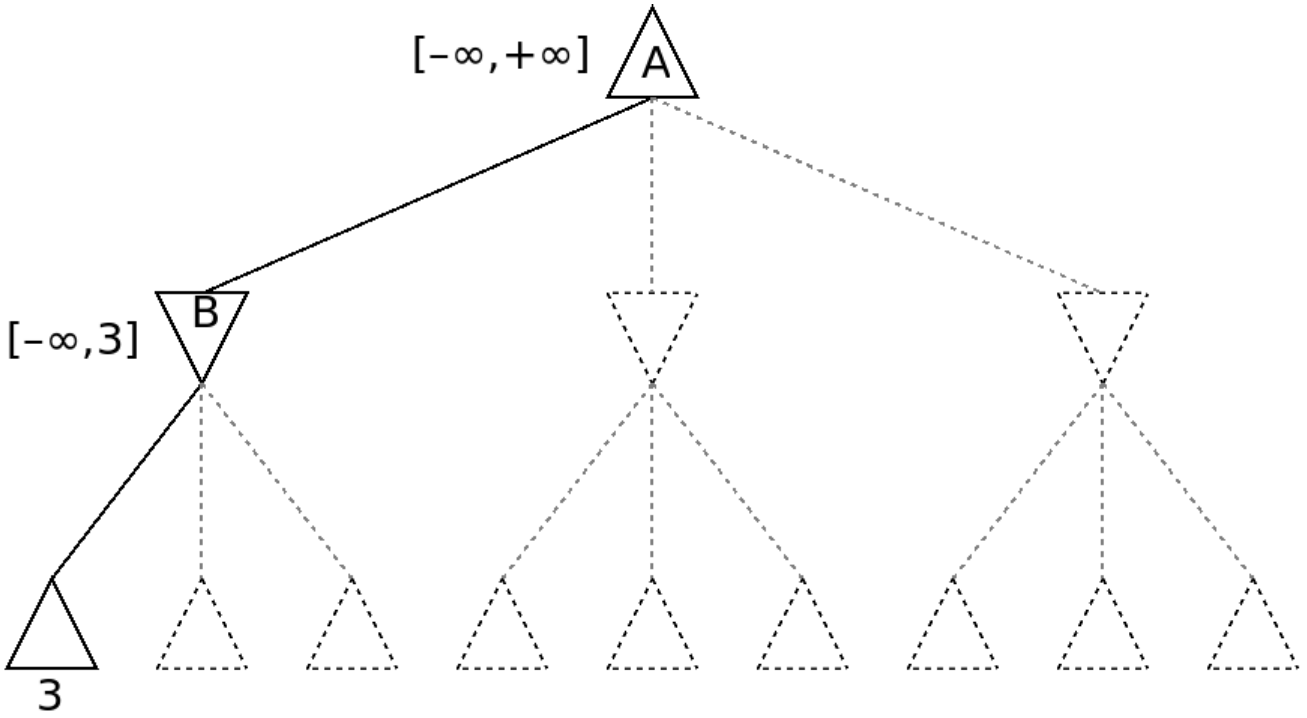
Élaguer « sans risque »

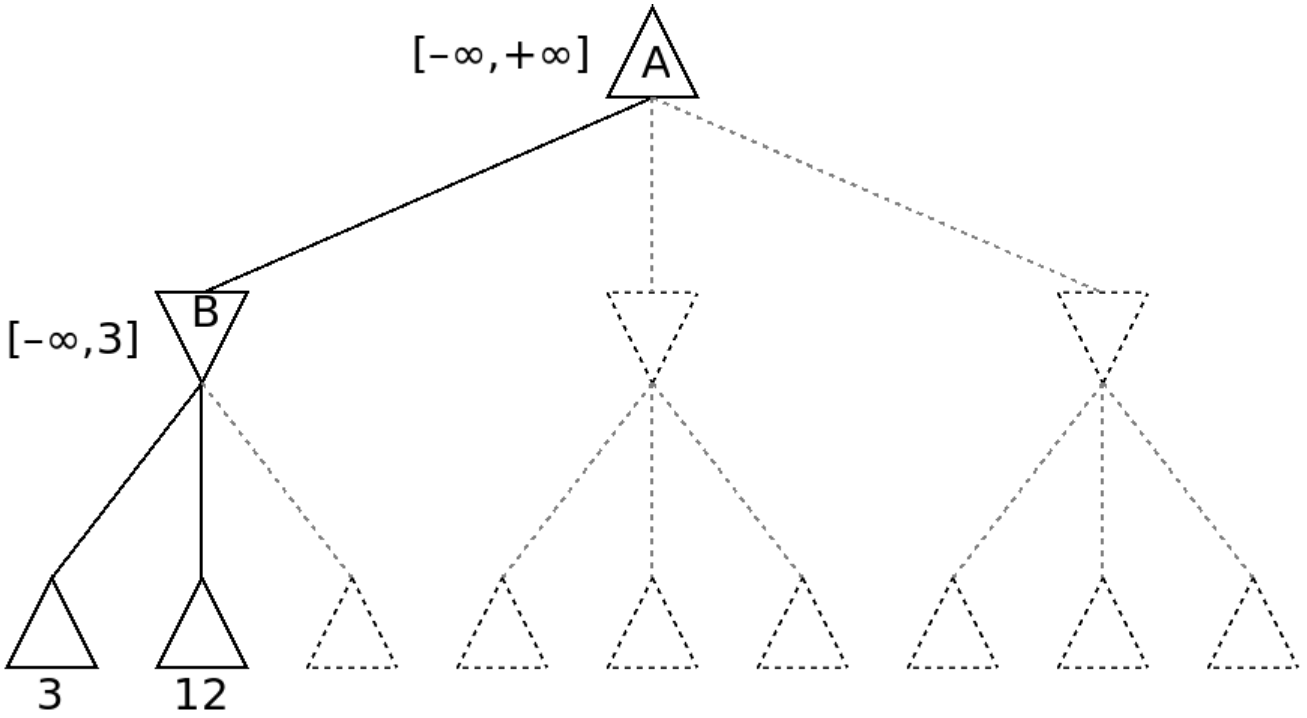
- La décision minimax peut être calculée sans examiner tous les nœuds de l'arbre de jeu
- Si m est un meilleur choix qu'un ancêtre de n pour MAX (x > y), alors n ne sera jamais atteint dans le jeu

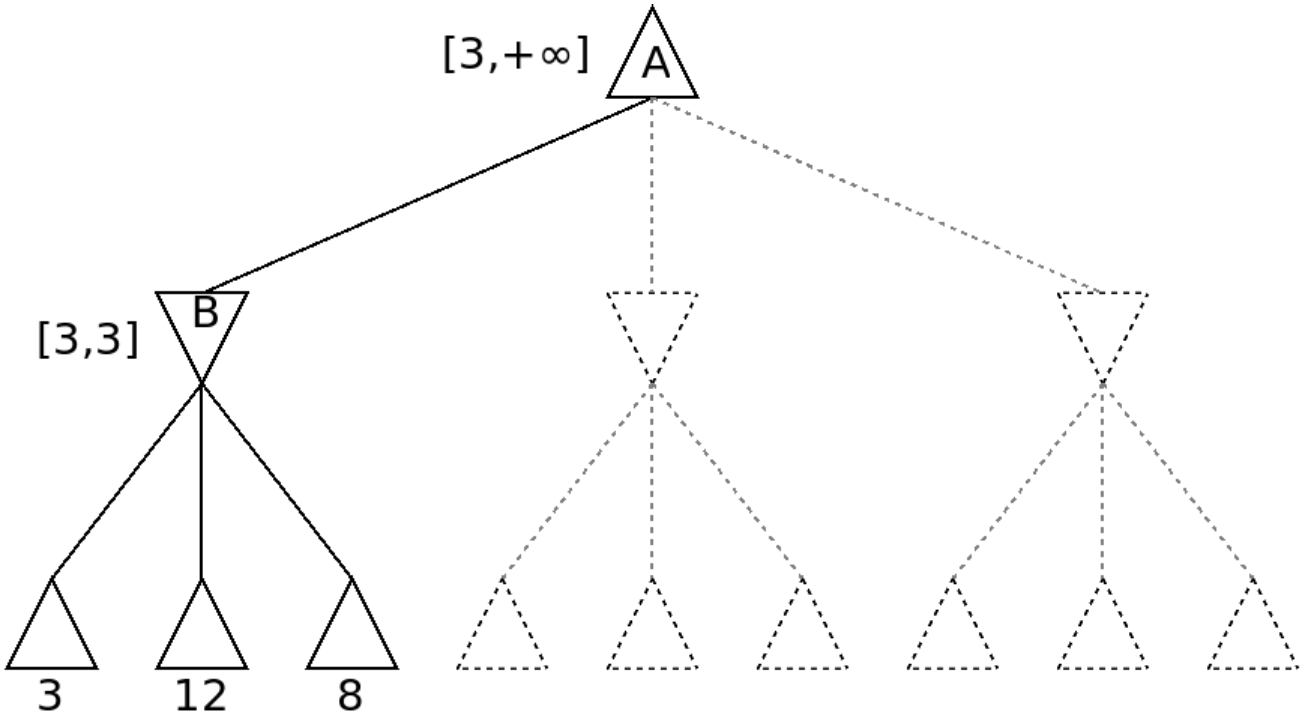


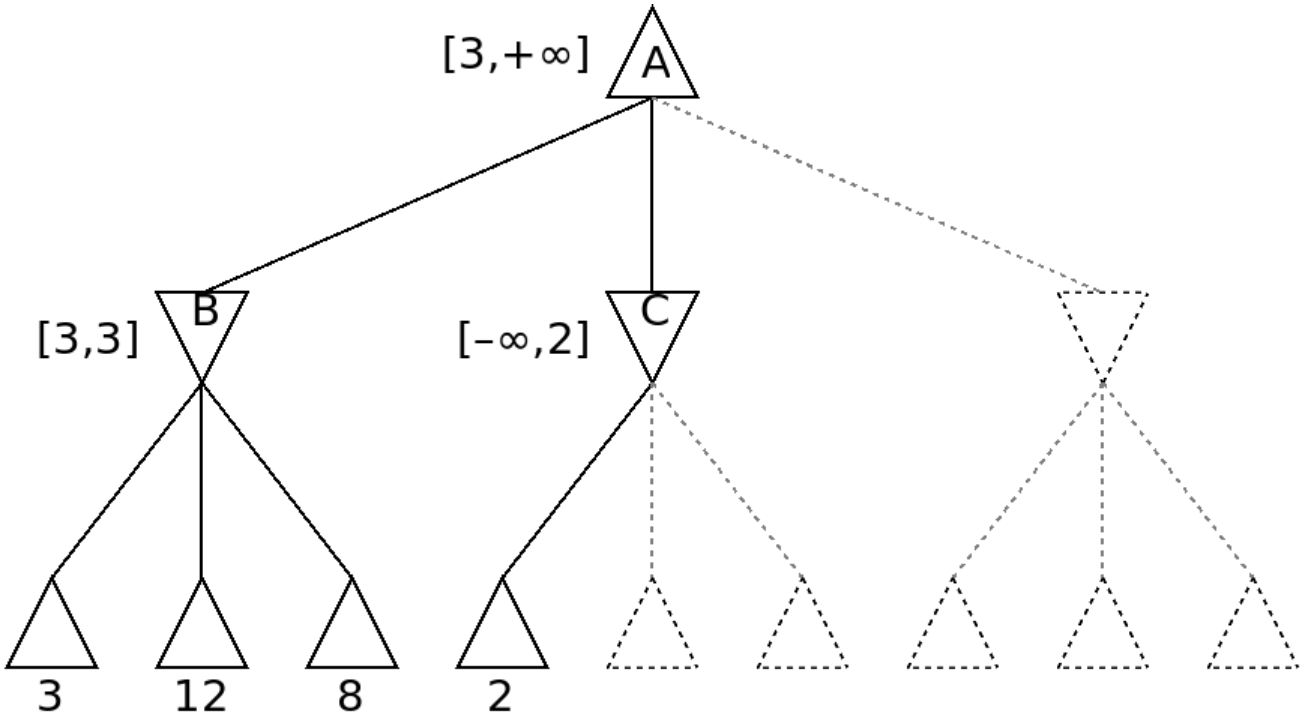
Recherche alpha-bêta

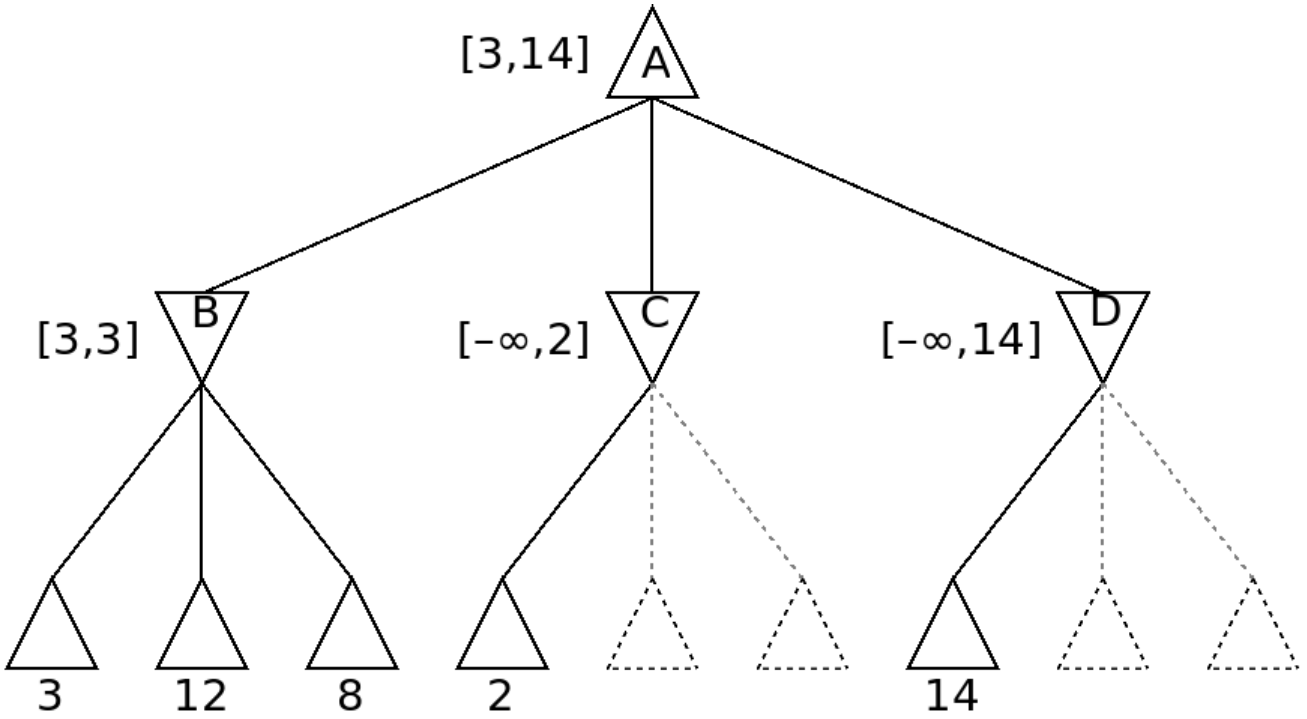
- α : meilleur choix (valeur la plus élevée) à un instant donné pour MAX sur le chemin
- β : meilleur choix (valeur la moins élevée) à un instant donné pour MIN sur le chemin
- Élagage dès que la valeur du nœud courant est moins bonne que la valeur concurrente α (resp. β) pour MAX (resp. MIN)

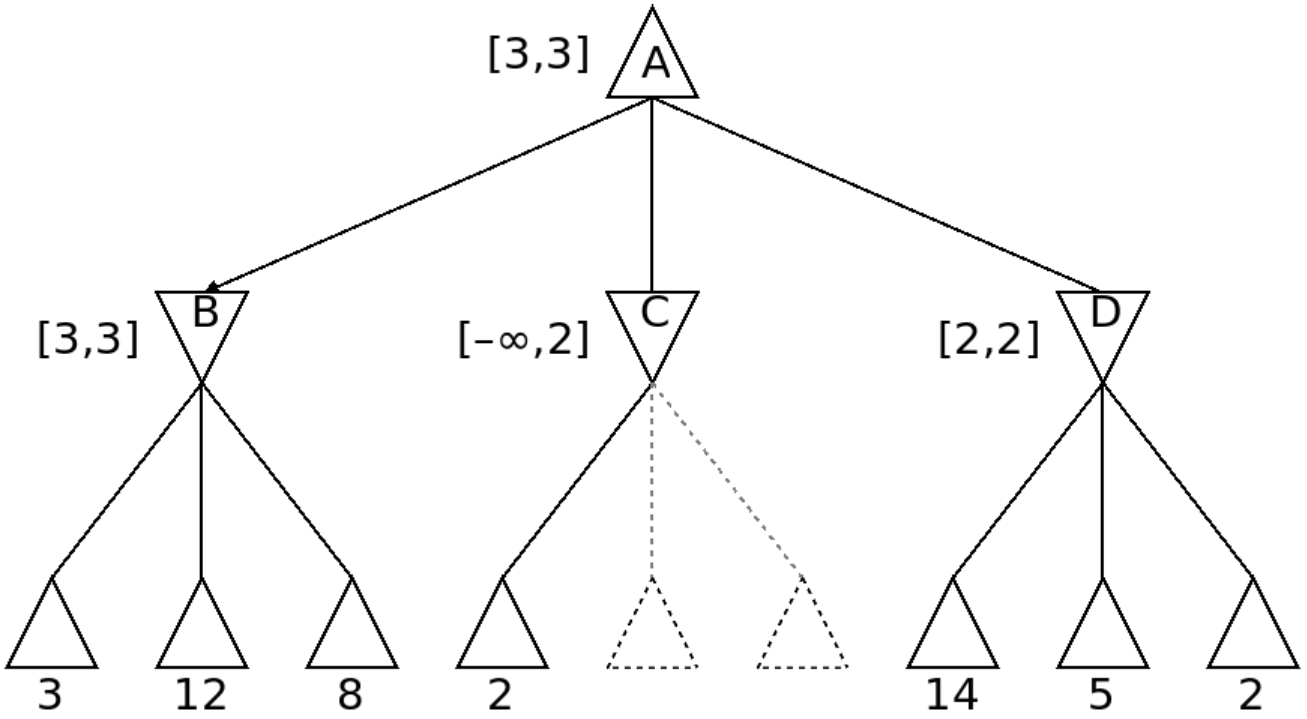

[−∞,+∞]
A
[−∞,3]
B
3


[−∞,+∞]
A
[−∞,3]
B
3
12


[3,+∞]
A
[3,3]
B
3
12
8


[3,+∞]
A
[3,3]
B
[–∞,2]
C
3
12
8
2


[3,14]
A
[3,3]
B
[−∞,2]
C
[−∞,14]
D
3
12
8
2
14


[3,3]
A
[3,3]
B
[−∞,2]
C
[2,2]
D
3
12
8
2
14
5
2

```
function RECHERCHE-ALPHA-BÊTA(état)
    v ← VALEUR-MAX(état,+∞,−∞)
    return action dans SUCCESSEURS(état) ayant la valeur v

function VALEUR-MAX(état,α,β)
    if TEST-TERMINAL(état) then
        return UTILITÉ(état)
    v ← −∞
    for a,s in SUCCESSEURS(état) do
        v ← MAX(v,VALEUR-MIN(s,α,β))
        if v ≥ β then
            return v
        α ← MAX(α,v)
    return v

function VALEUR-MIN(état,α,β)
    if TEST-TERMINAL(état) then
        return UTILITÉ(état)
    v ← +∞
    for a,s in SUCCESSEURS(état) do
        v ← MIN(v,VALEUR-MAX(s,α,β))
        if v ≤ α then
            return v
        β ← MIN(β,v)
    return v
```

Recherche alpha-bêta

- L'efficacité dépend de l'ordre des successeurs
- Si les nœuds sont « bien » ordonnés
  - Temps : $O(b^{m/2})$ pour sélectionner le meilleur coup
  - Branchement effectif : $\sqrt{b}$ au lieu de b
- Ordre aléatoire : $O(b^{3m/4})$

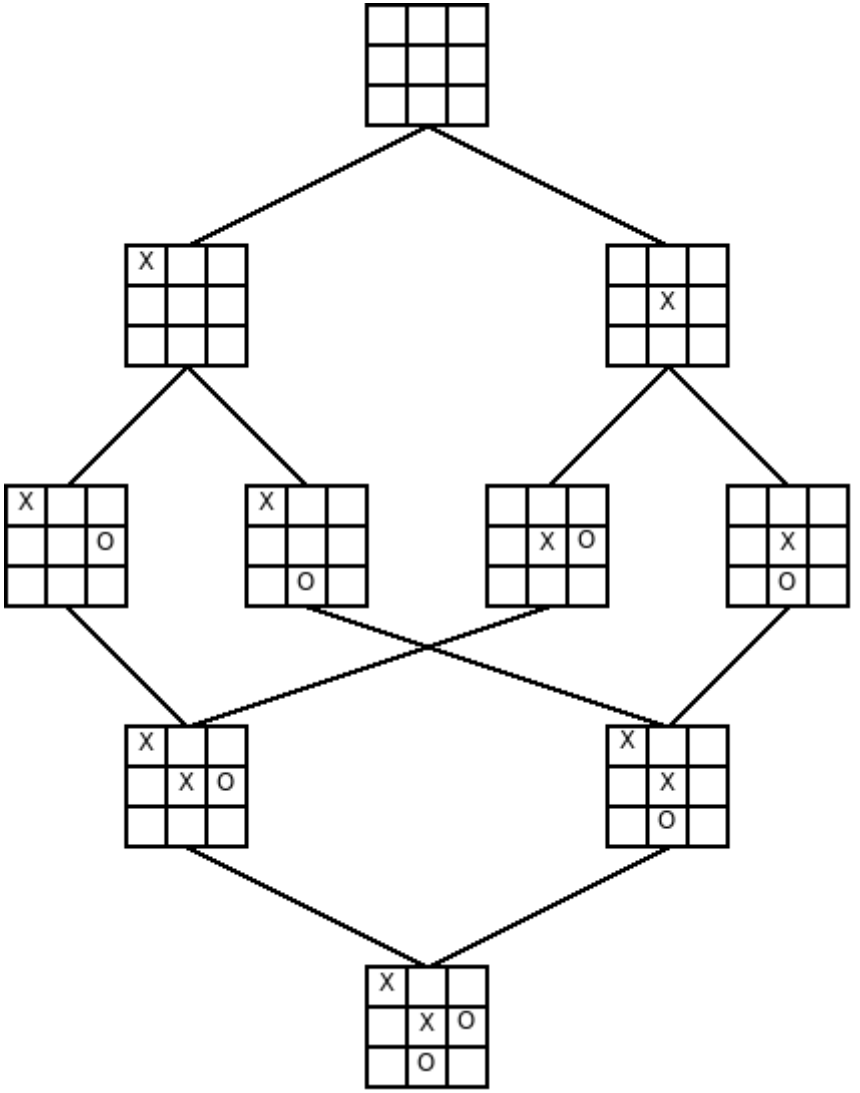
Implémentation negamax

```
function RECHERCHE-ALPHA-BÊTA(état)
    v ← VALEUR-NEGAMAX(état,+∞,−∞)
    return action dans SUCCESSEURS(état) ayant la valeur v

function VALEUR-NEGAMAX(état,α,β)
    if TEST-TERMINAL(état) then
        return UTILITÉ(état)
    v ← −∞
    for a,s in SUCCESSEURS(état) do
        v ← MAX(v,−VALEUR-NEGAMAX(s,−β,−α))
        if v ≥ β then
            return v
        α ← MAX(α,v)
    return v
```

Transpositions

- Permutations d'une séquence de coups qui aboutissent à la même position
- Table des transpositions : table de hachage qui mémorise les positions déjà rencontrées



Décisions imparfaites en temps réel

Fonctions d'évaluation

- Estimation de l'utilité espérée du jeu pour une position donnée
- Ordonner les états terminaux comme la vraie fonction d'utilité
- Calcul rapide
- Bonne chances de victoire pour les états non terminaux

Fonctions d'évaluation

- Calcul de différents attributs d'un état
- Catégorie d'états
  - Mêmes valeurs pour tous les attributs
  - Valeur espérée : moyenne pondérée des issues possibles (basée sur l'expérience)
- Combinaison linéaire des attributs pour chaque état

$$EVAL(s) = w_1 f_1(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

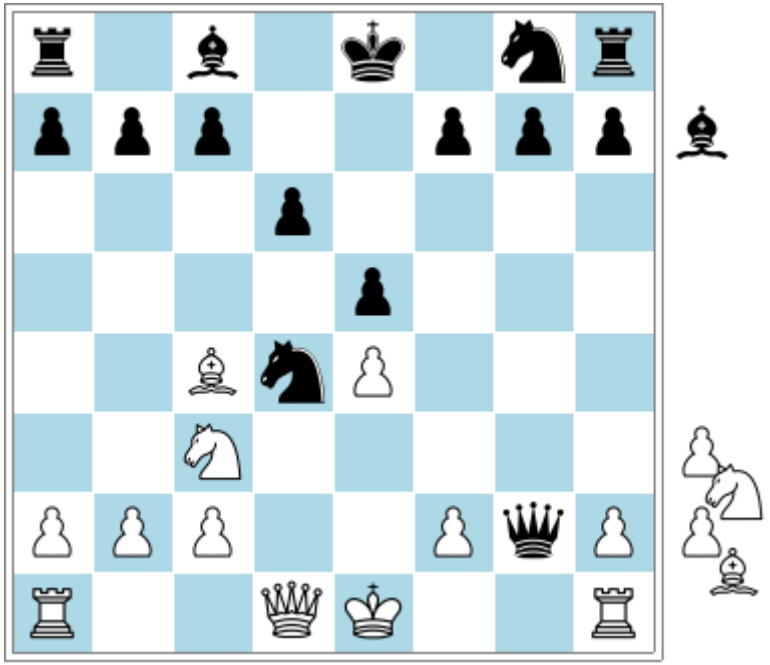
Recherche avec arrêt

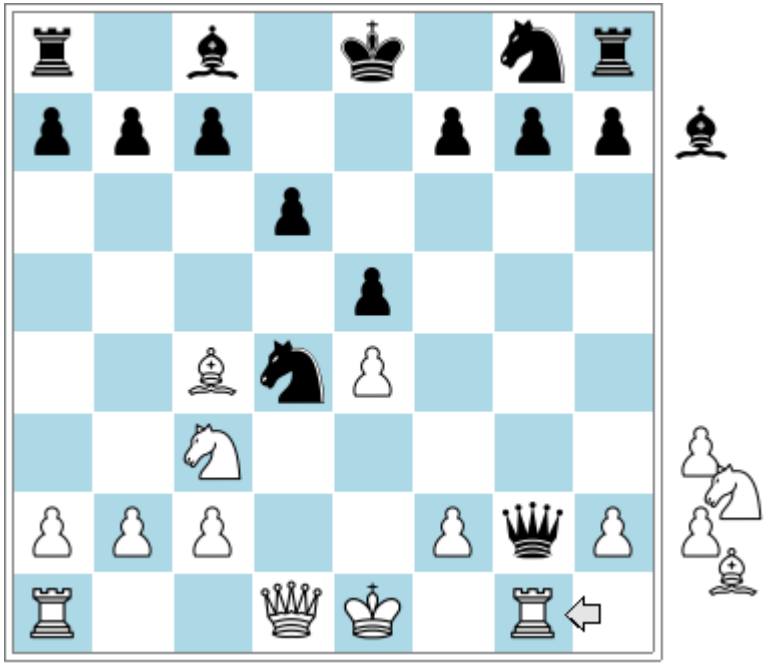
- *Cut-off*
- Modification de l'algorithme alpha-bêta
  - Fixer une profondeur limite
  - Incrémenter la profondeur courante à chaque appel récursif
  - Remplacer la condition d'arrêt de l'exploration

```
…
~~if TEST-TERMINAL(état) then return UTILITÉ(état)~~
if TEST-CUTOFF(état,profondeur) then return ÉVAL(état)
…
```

- Autre méthode : appliquer l'exploration itérative en profondeur

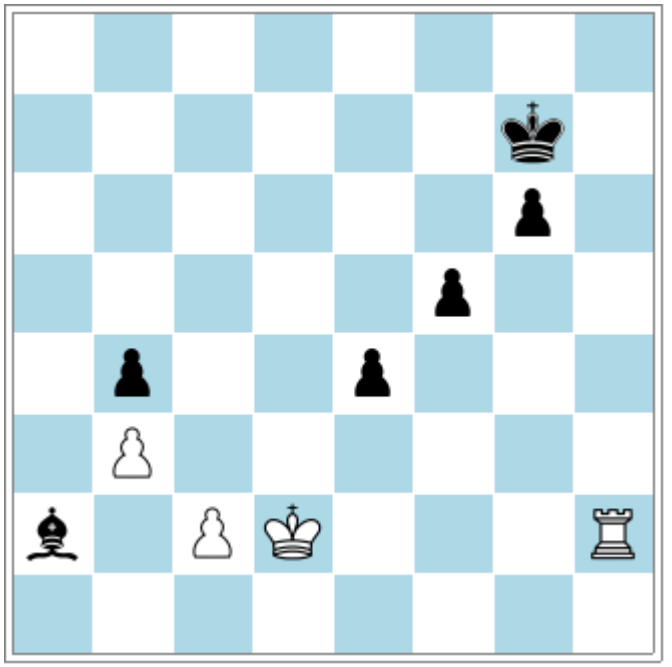
Recherche avec arrêt





Recherche avec arrêt

- Effet d'horizon
- Recherche de stabilité (*quiescence search*)
- Extensions singulières



Élagage en avant

- *Forward pruning*
- Suppression immédiate de certains coups sans autre considération
- Exploration en faisceau (*beam search*)
- Variante probabiliste

Jeux comportant une part de hasard

Valeur minimax espérée

- Nœuds de hasard dans l'arbre de jeu
- EXPECTIMINIMAX(n) =
  - UTILITÉ(n) si n nœud terminal
  - $\max_{s \in succ(n)}$EXPECTIMINIMAX(s) si n nœud MAX
  - $\min_{s \in succ(n)}$EXPECTIMINIMAX(s) si n nœud MIN
  - $\sum_{s \in succ(n)} P(s) \cdot$EXPECTIMINIMAX(s) si HASARD
- Complexité en temps : $O(b^m p^m)$, avec p facteur de hasard

État de l'art

Programmes de jeux

- « Les jeux sont à l'IA ce que la F1 est à l'industrie automobile. »
- Premiers champions du monde non humains
  - Dames 8 x 8 : Chinook (1994)
  - Échecs : Deep Blue (1997)
  - Othello : Logistello (1997)
- Go : niveau amateur
- Backgammon : prise en compte de l'incertitude
- Bridge : information imparfaite

Deep Blue

- Superordinateur IBM : 30 processeurs RS/6000 et 480 processeurs VLSI
- Exploration alpha-bêta itérative en profondeur : 126 millions de nœuds par seconde, 30 milliards de positions par coup, profondeur 14
- Table de transpositions
- Extensions au-delà de la profondeur limite (jusqu'à 40 coups)
- Fonction d'évaluation : plus de 8 000 attributs
- Bibliothèque d'ouvertures : 4 000 positions
- Base de données de 700 000 parties de niveau grand maître
- Fins de parties : base de données avec toutes les positions résolues à cinq pièces (voire six)
- Heuristique du coup nul : borne inférieure sur la valeur d'une position

Conclusion

- Approche dominante : minimax/alpha-bêta
- L'efficacité des machines repose essentiellement sur des capacités de calcul phénoménales
- Les humains jouent avec des objectifs