Introduction à l'intelligence artificielle

Wieslaw Zielonka
www.irif.fr/~zielonka

Recherche en présence de l'adversaire

Jeux à somme zéro et l'information parfaite

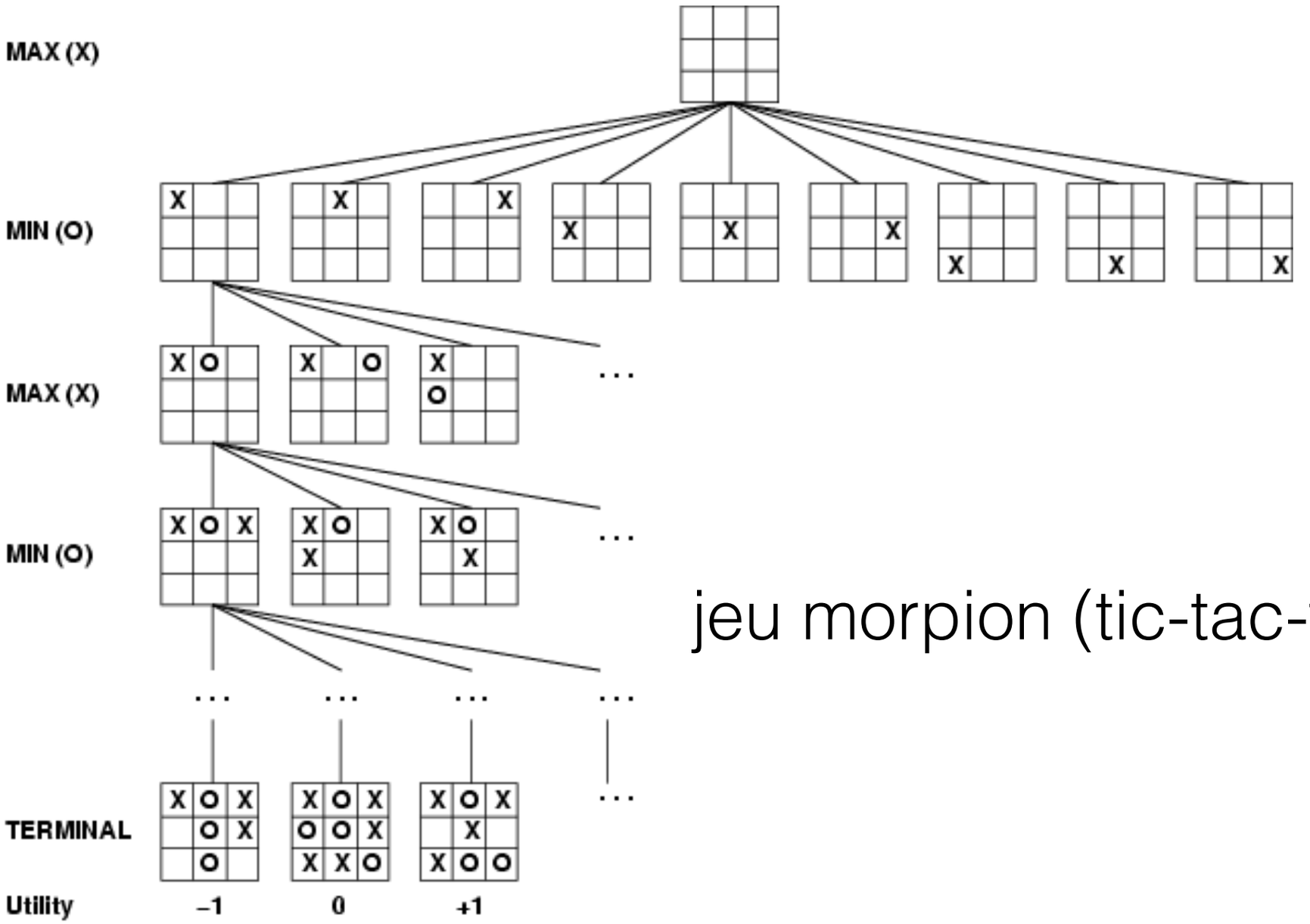
- Deux joueurs : MAX et MIN
- ce qu'un joueur gagne l'autre perd :

si o le résultat de jeu et u_{MAX} et u_{MIN} les fonctions de paiement (les fonction d'utilité) de deux joueur alors pour chaque o,

$$u_{MIN}(o) + u_{MAX}(o) = 0$$

Dans ce cas il suffit de spécifier $u(o) = u_{MAX}(o)$, et on appelle u la fonction d'utilité.

L'arbre de jeu (2 joueurs jouant à tour de rôle)



jeu morpion (tic-tac-toe)

Description d'un jeu

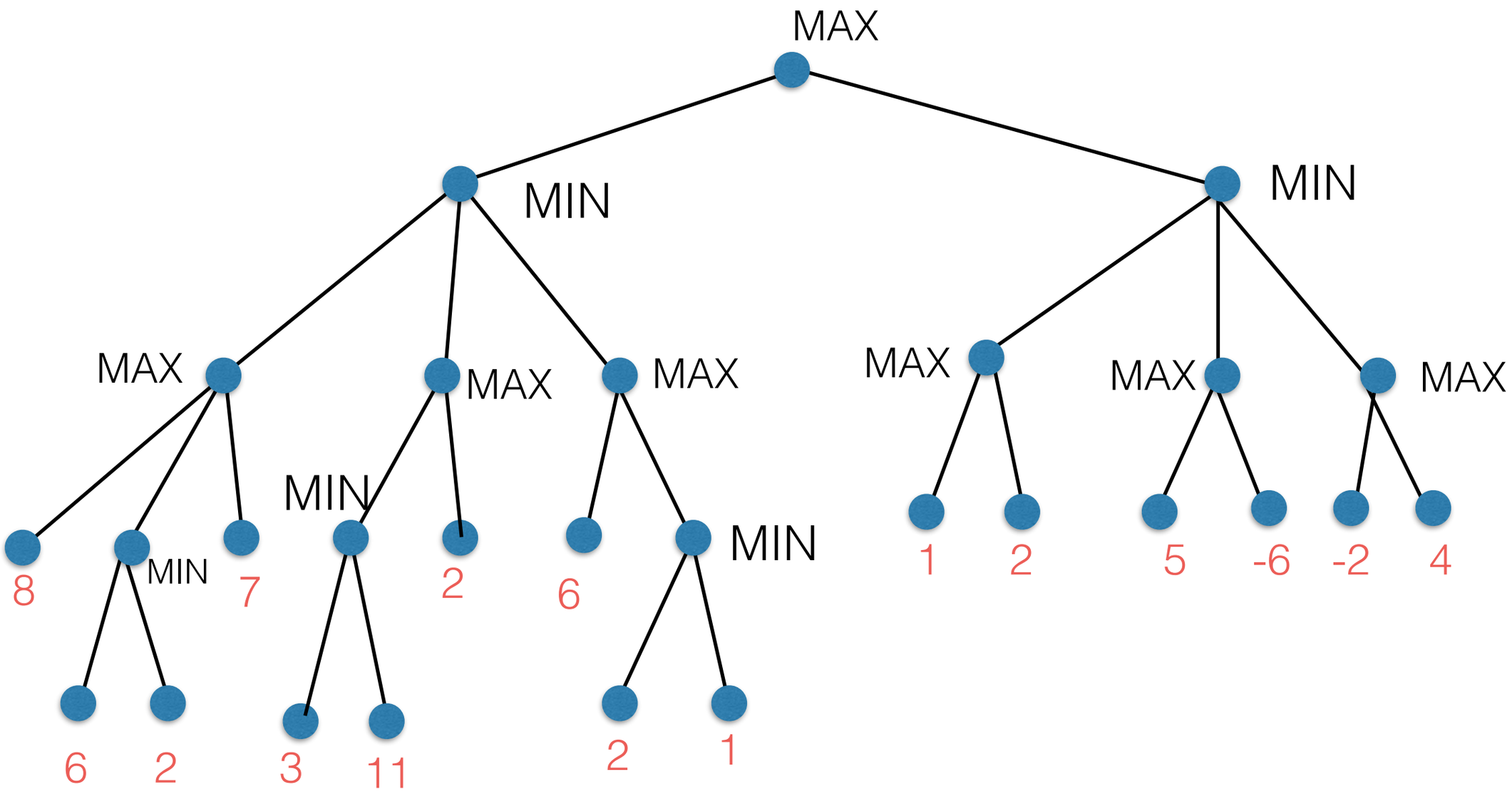
- **S**0 l'état initial
- PLAYER(s) le joueur qui joue dans l'état s
- ACTIONS(s) l'ensemble d'actions disponibles dans l'état s
- RESULTAT(s,a) transition, l'état obtenu quand on exécute l'action a dans s
- TERMINAL(s) un test booléen pour déterminer si s est un état final
- UTILITE(s) la fonction d'utilité définie pour chaque état final s. On suppose que UTILITE est la fonction d'utilité du joueur MAX, en changeant le signe on obtient l'utilité pour le joueur MIN.

Difficulté de jeux

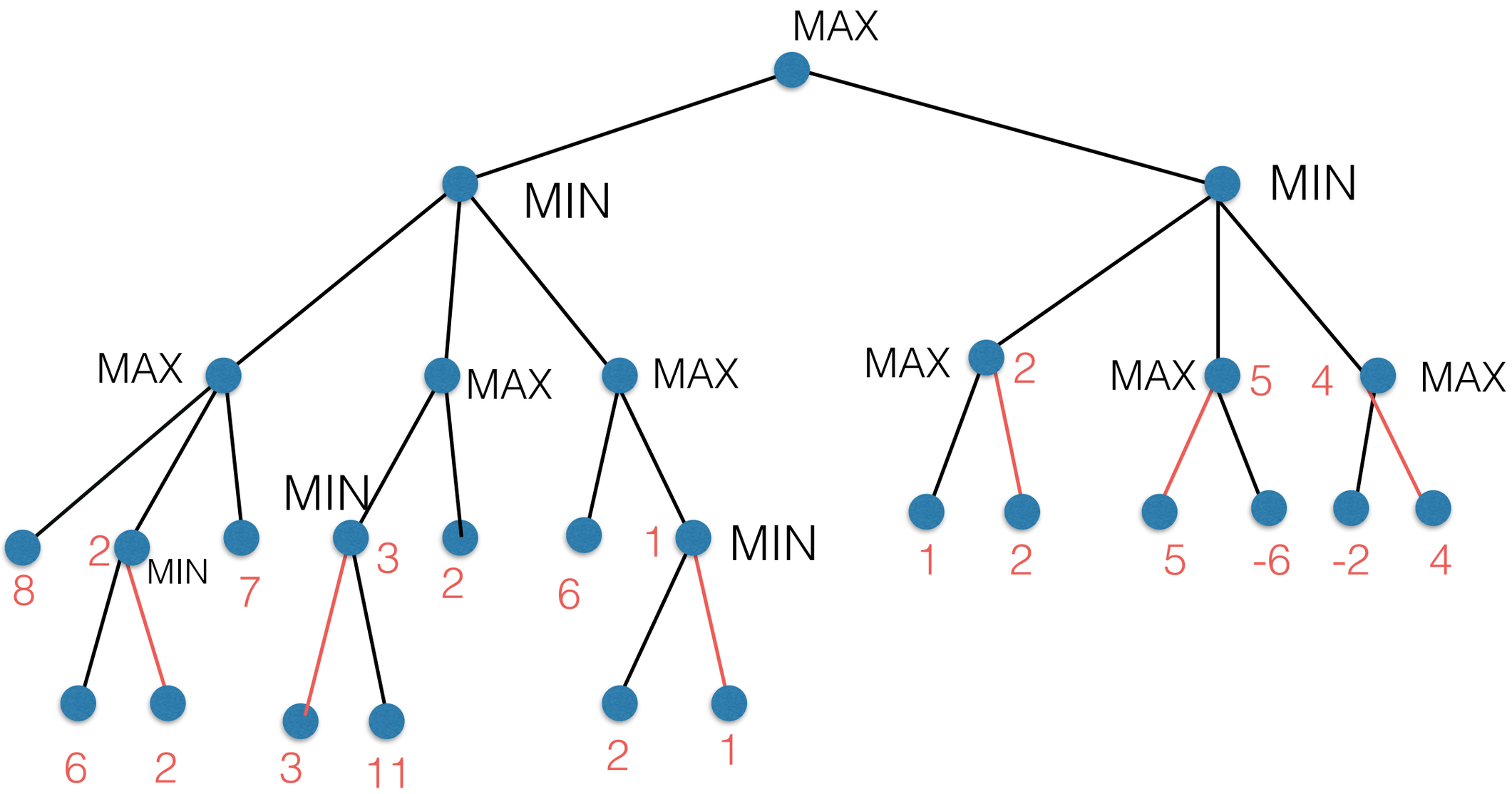
Grand nombre d'états : jeu d'échec :

- facteur de branchement (en moyenne) 35 (facteur de branchement = le nombre d'actions disponibles dans un état)
- nombre de coup par joueur 50
- l'arbre de jeu 35^{100} de noeuds

Minmax

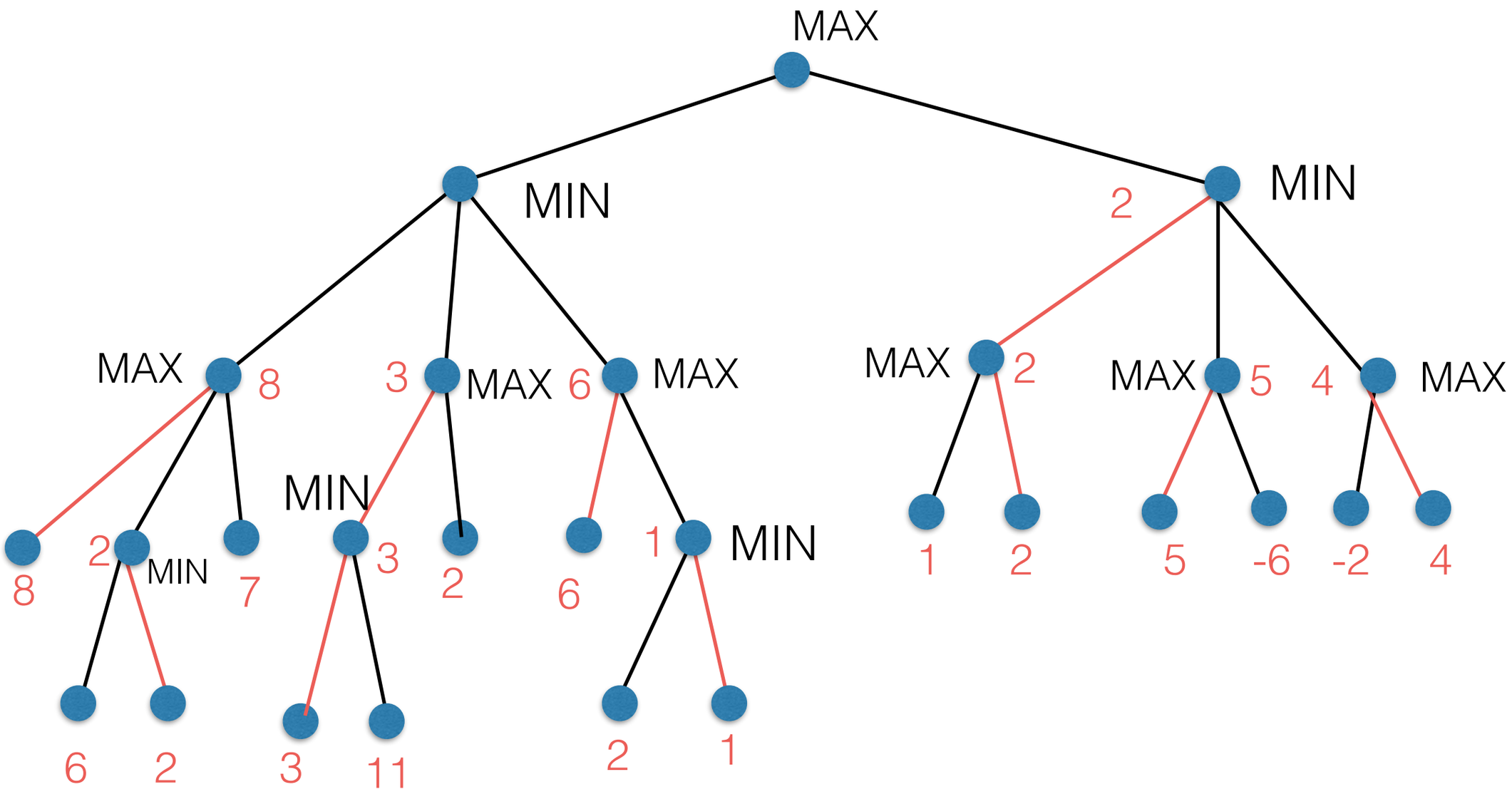


Minmax



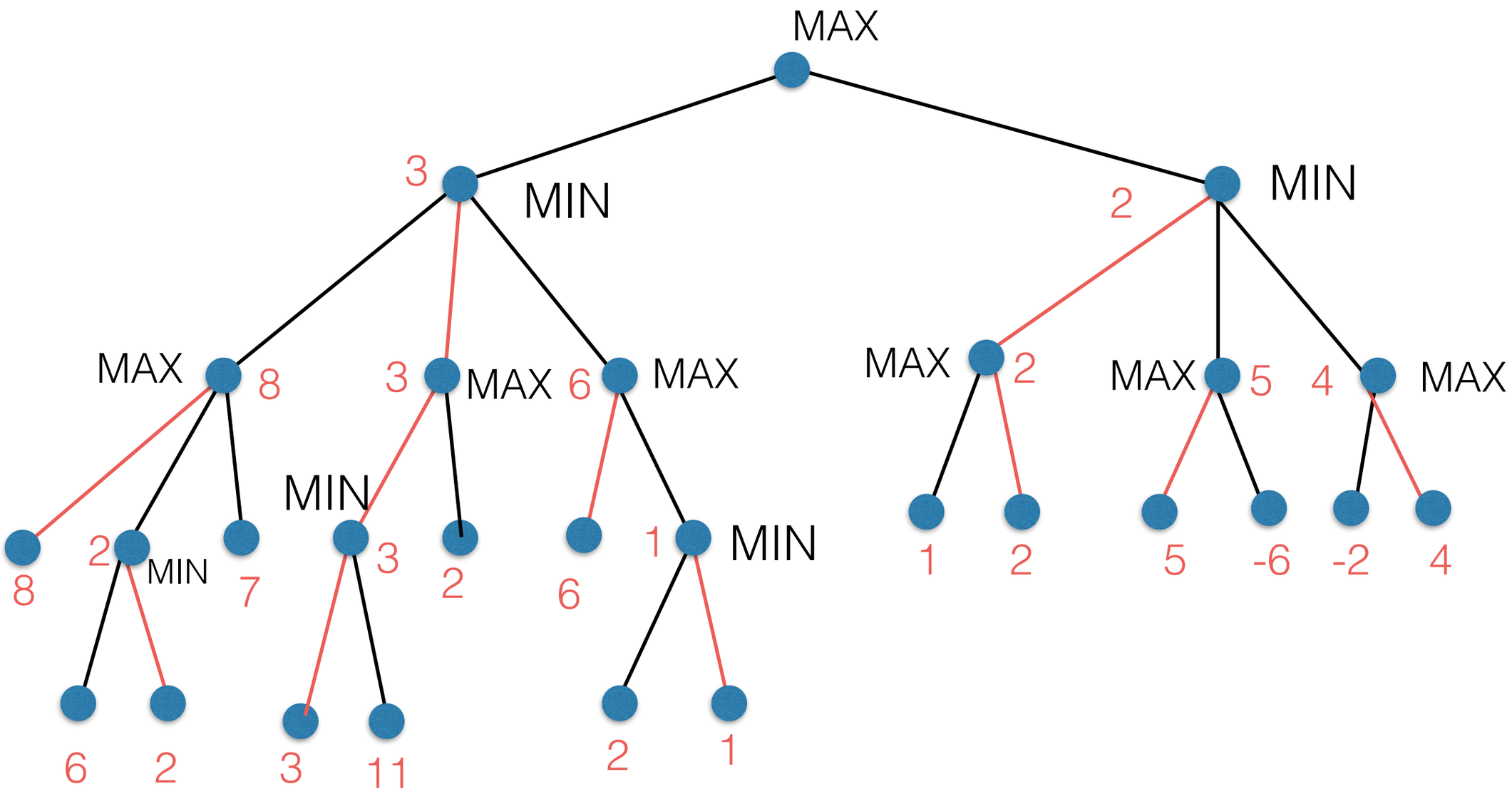
Calculer minmax de chaque sommet en partant de feuilles.

Minmax



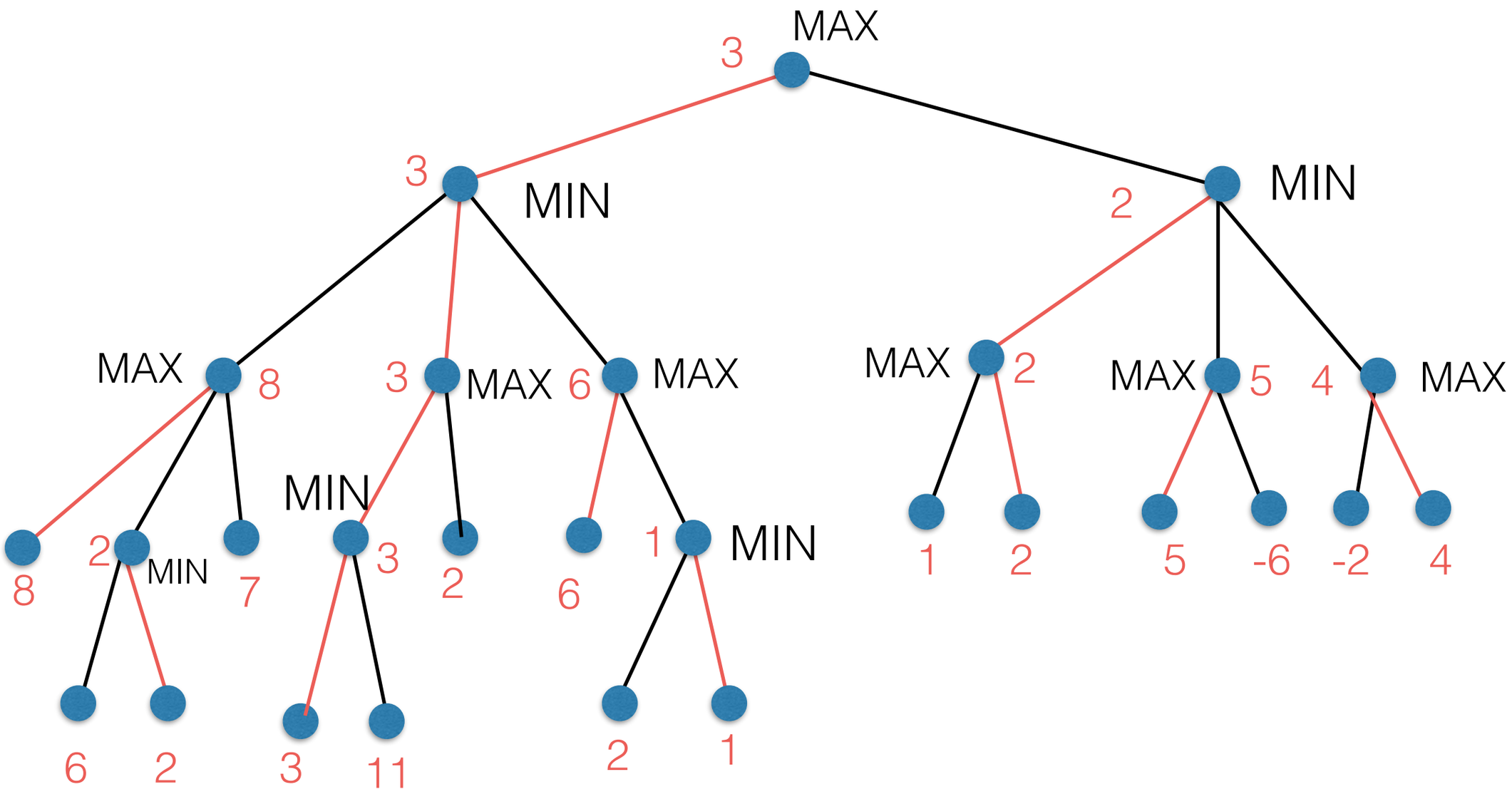
Calculer minmax de chaque sommet en partant de feuilles.

Minmax



Calculer minmax de chaque sommet en partant de feuilles.

Minmax



Calculer minmax de chaque sommet en partant de feuilles.

Algorithme Minimax

```
function MinMaxDecision(état s) returns action
   if Player(s) == Max then
        return a in Actions(s)
        telle que MinMax(Resultat(s,a)) est maximal
   else
        return a in Actions(s)
        telle que MinMax(Resultat(s,a)) est minimal
```

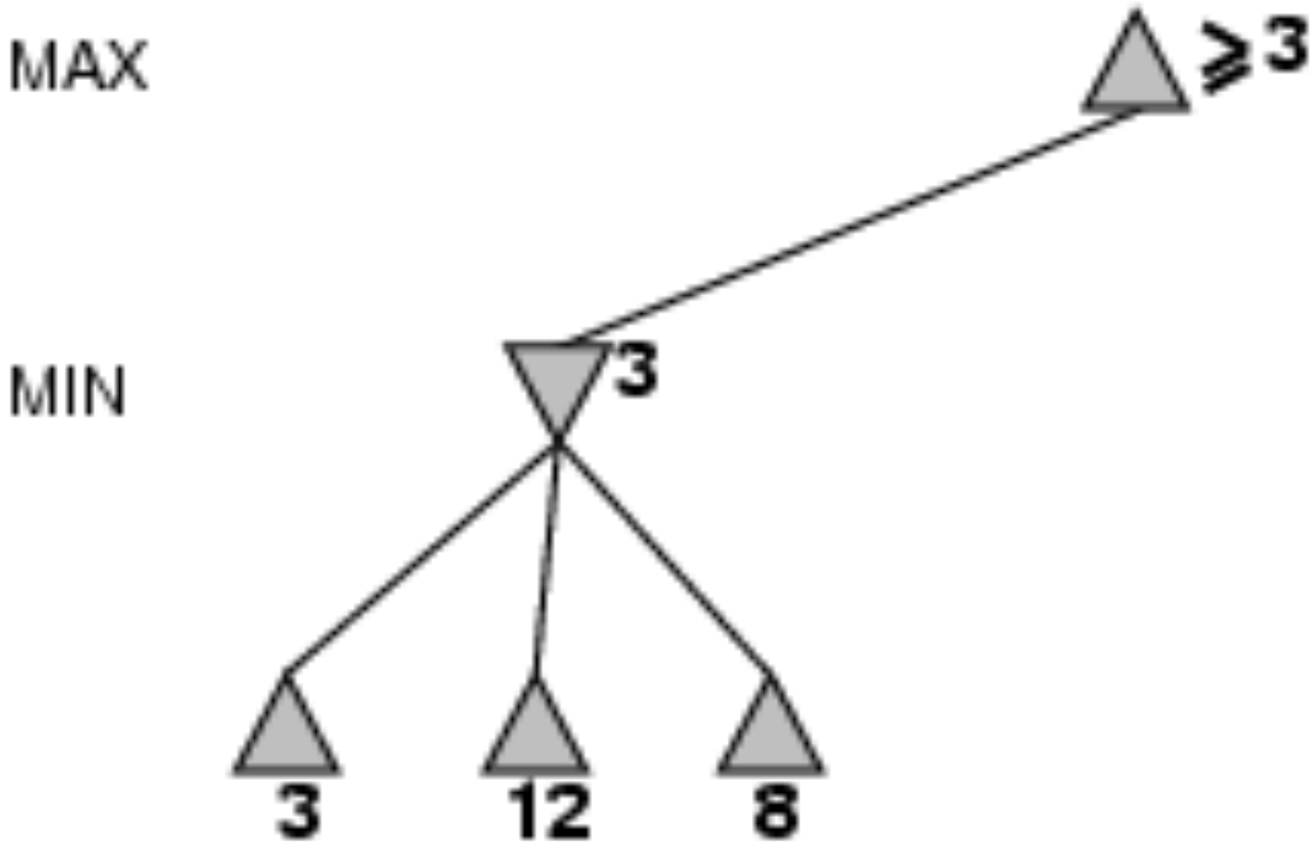
```
function MinMax(état s) returns valeur
   if Terminal(s) then return Utilité(s)
   if Player(s) == Max then
        v = −∞
        for a in Actions(s) do
            v = max(v, MinMax(Resultat(s,a)))
   else
        v = +∞
        for a in Actions(s) do
            v = min(v, MinMax(Resultat(s,a)))
   return v
```

Propriétés de l'algorithme Minmax

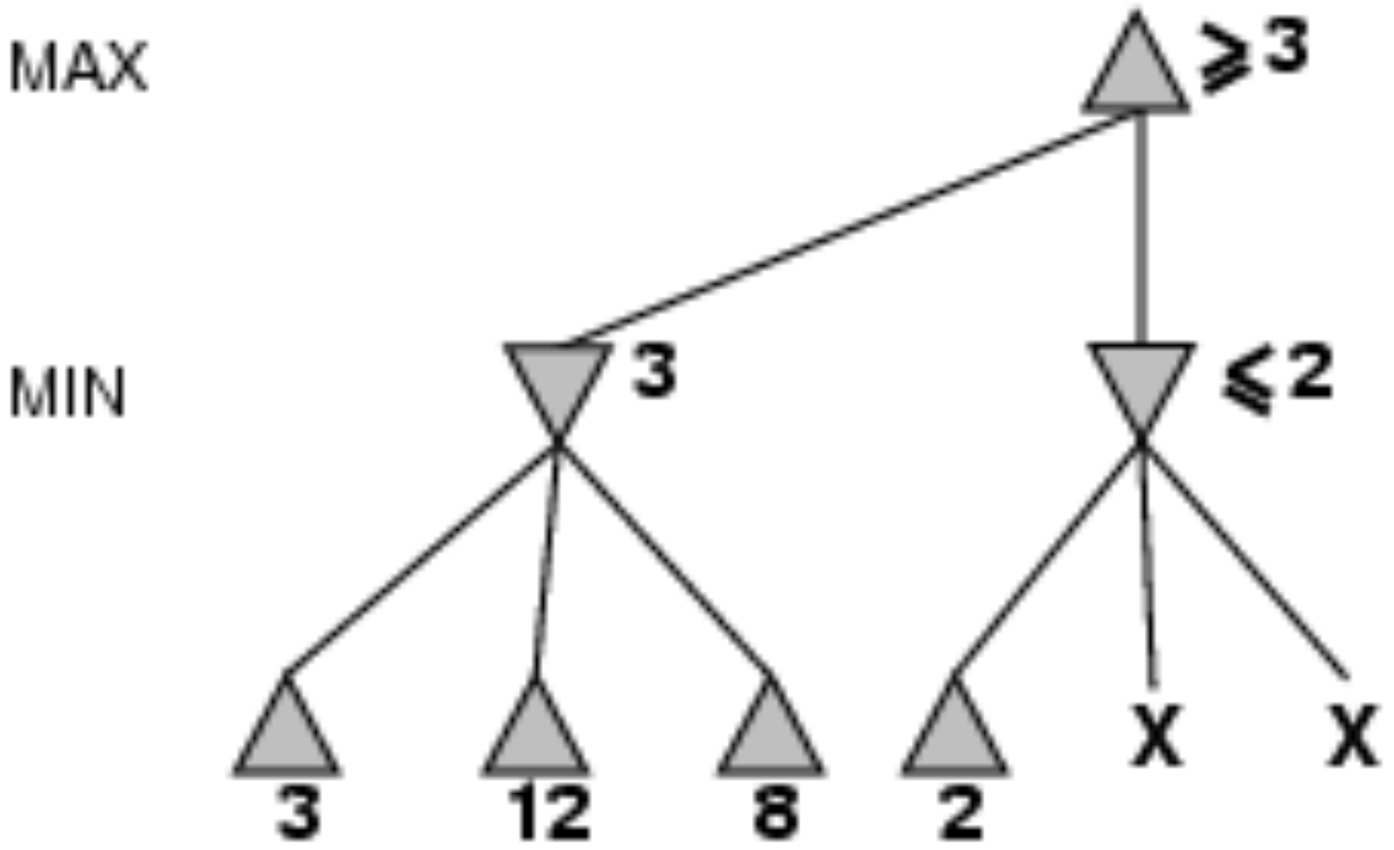
- complet ? oui si l'arbre fini
- optimal ? oui si l'arbre fini (et l'adversaire optimal)
- complexité en temps $O(b^m)$ ou b le facteur de branchement et $ la profondeur de l'aigre de jeu
- complexité en espace O(bm)

Irréaliste pour le jeu d'échecs.

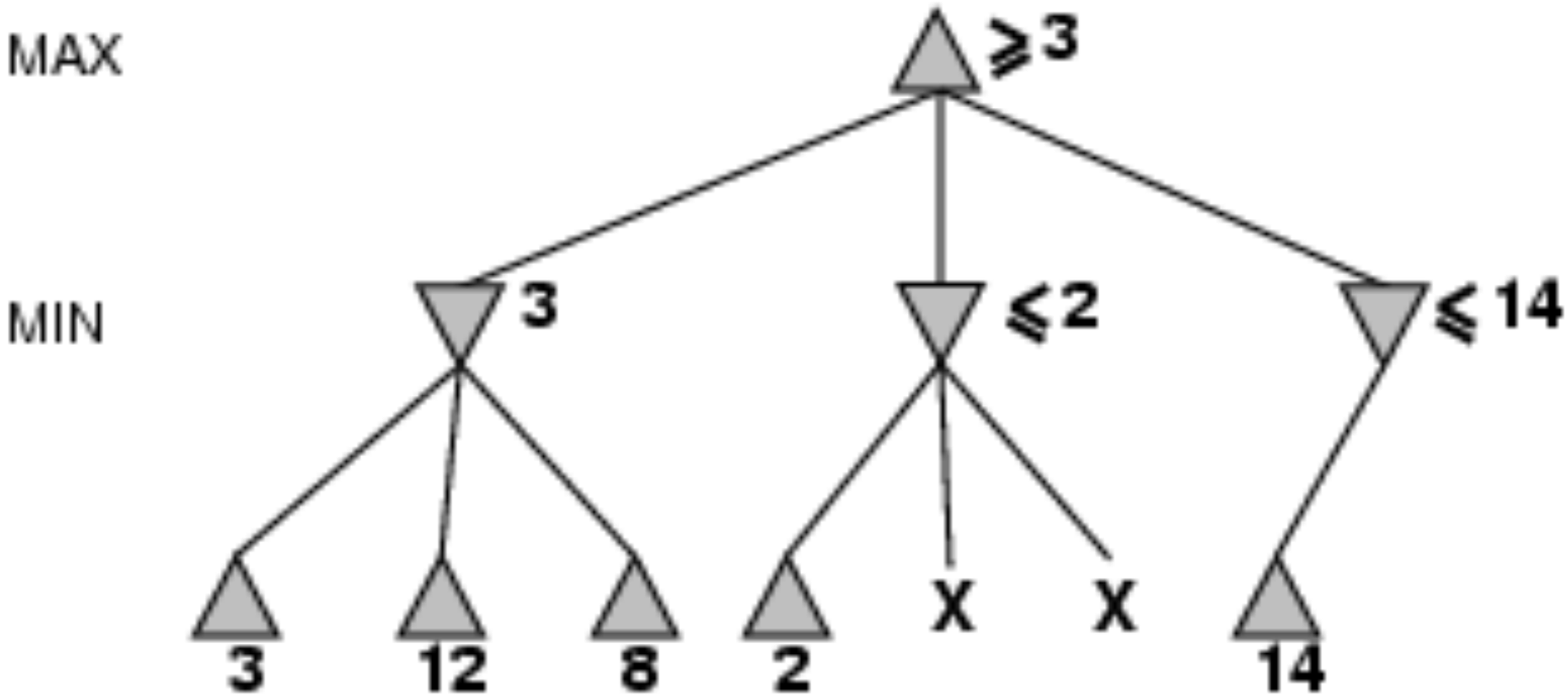
exemple d'élagage α-β



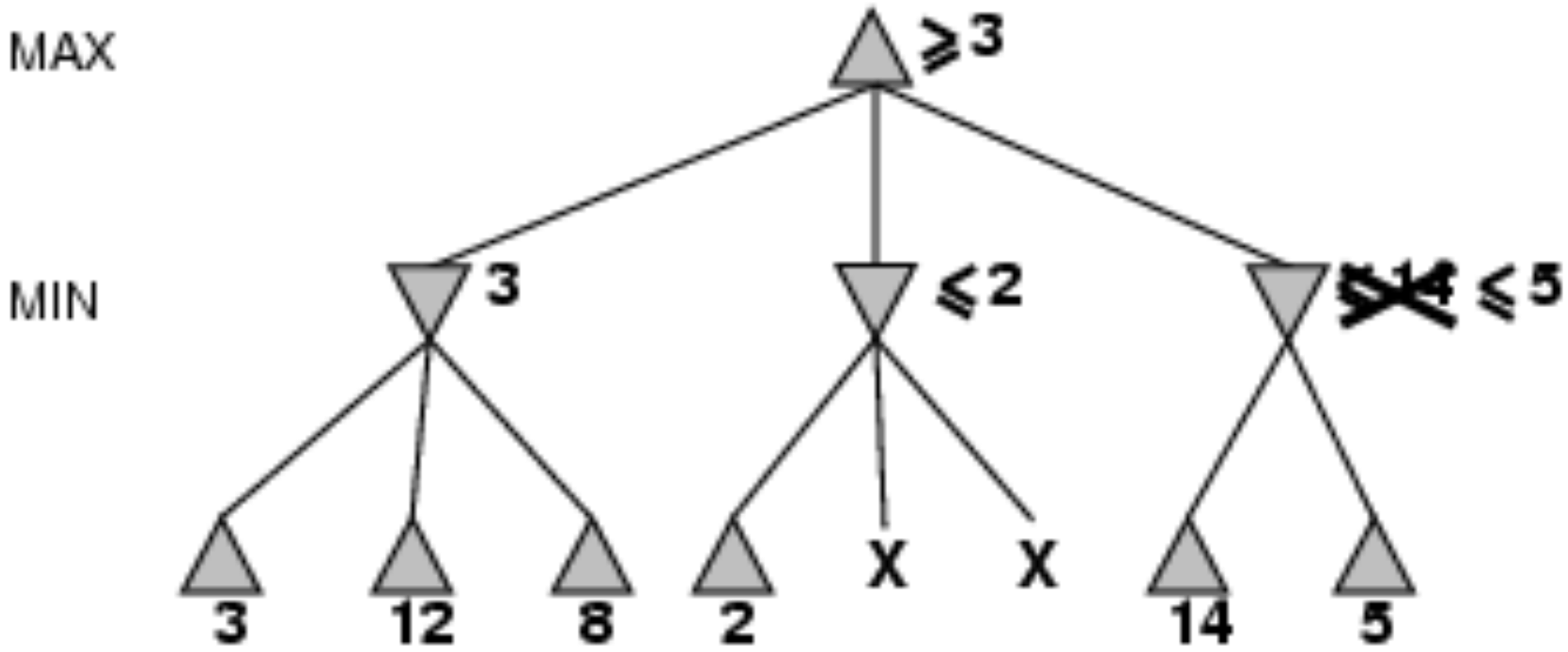
exemple d'élagage α-β



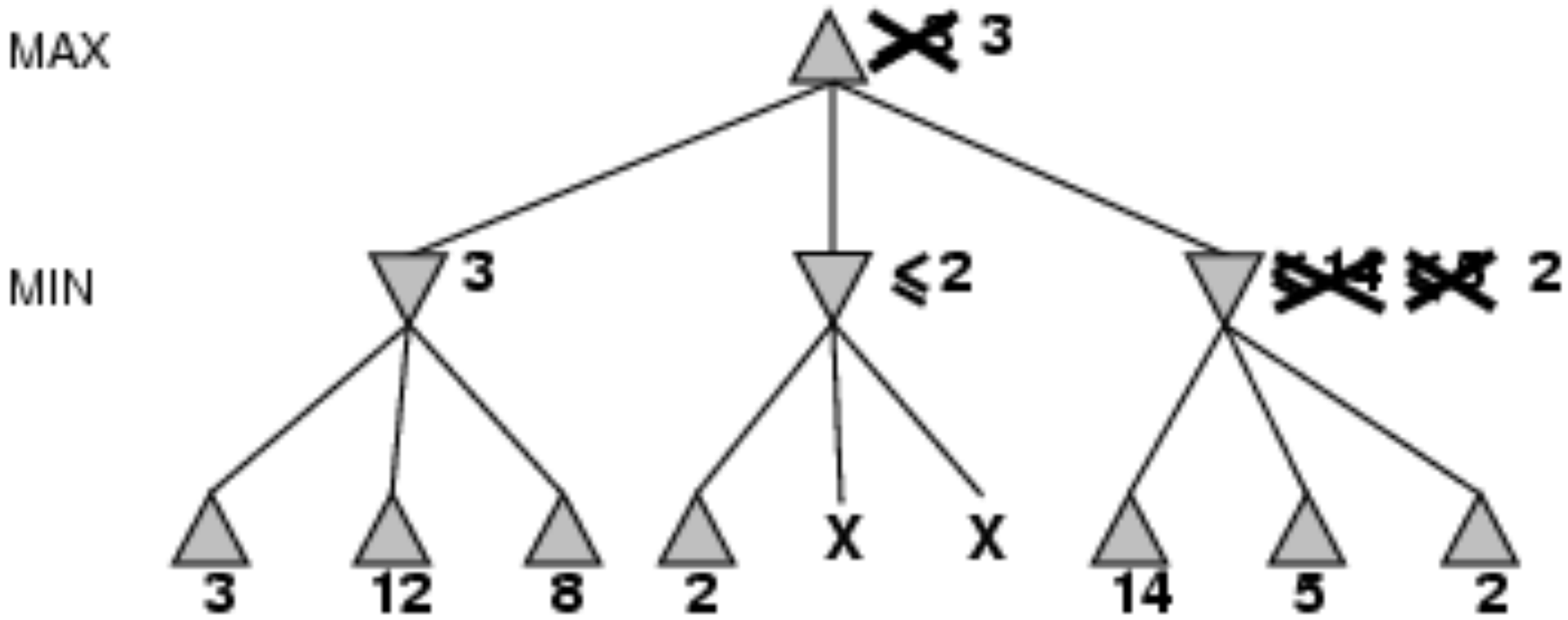
exemple d'élagage α-β



exemple élagage α-β



exemple élagage α-β

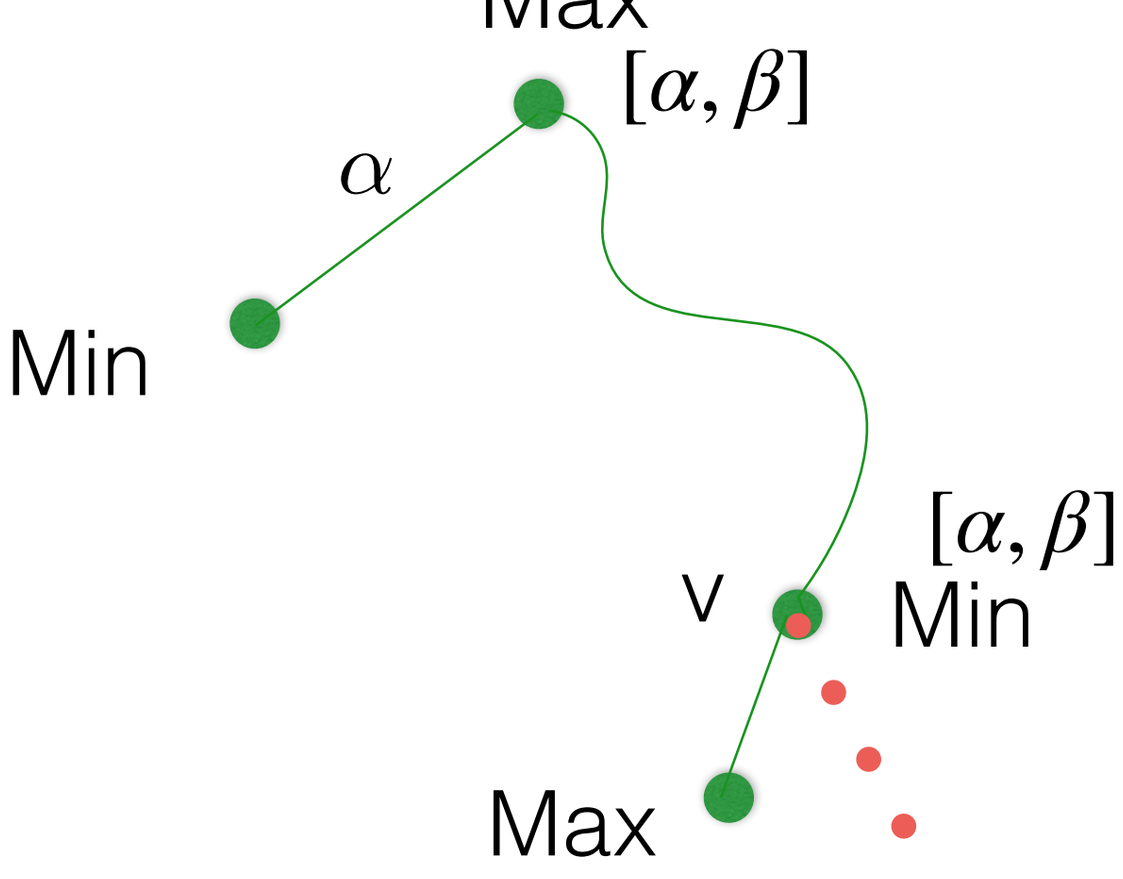


Propriété d'élagage $\alpha - \beta$

- Elagage n'affecte pas le résultat
- L'ordre de visite de enfants est important et influence la complexité (qui dépend de la taille de parties élaguées).
- Avec l'ordre de visite optimal la complexité en temps passe de $O(b^m)$ à $O(b^{m/2})$, donc on peut doubler la hauteur de l'arbre visité.

Le rôle de paramètres alpha-beta élagage α

- α - Max possède une stratégie qui lui garantie au moins α
- β - Min possède une stratégie qui lui garantie de ne pas perdre plus que β

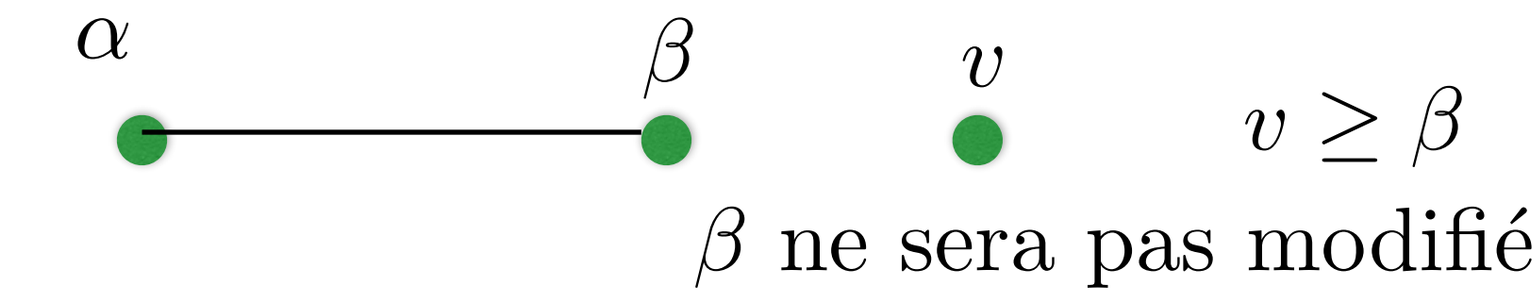


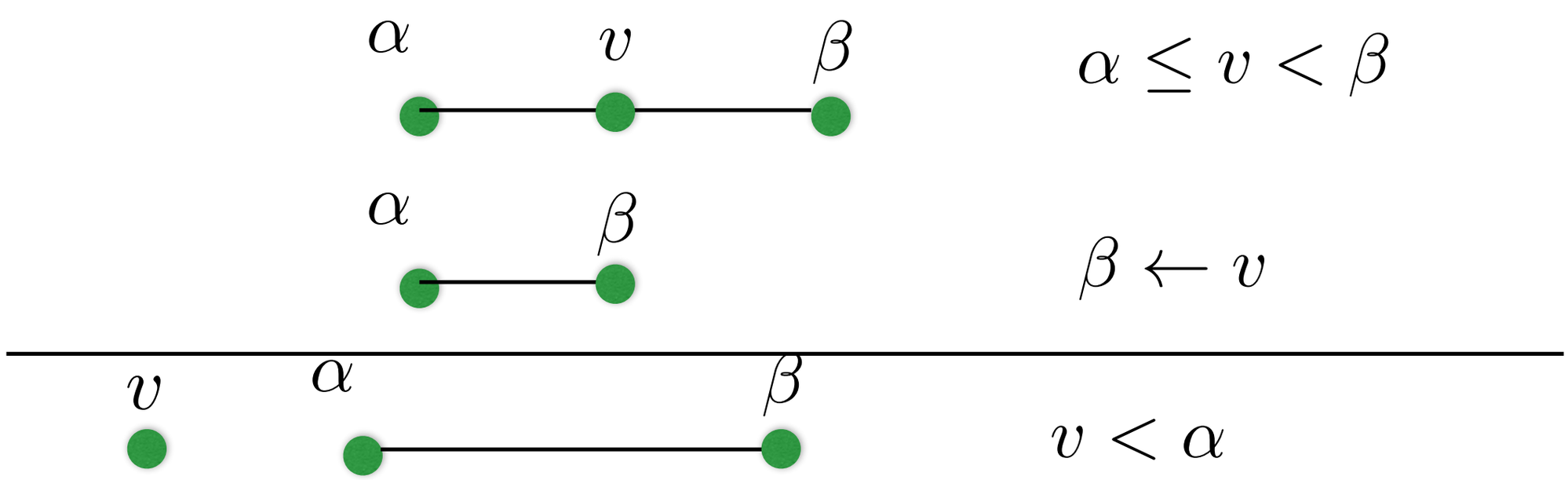
le joueur Min constate qu'il peut limiter la valeur à v et $v \leq \alpha$

Joueur Min contrôle le sommet

$[\alpha, \beta]$ l'intervalle de sommet courant contrôlé par Min

v – une valeur retournée par un fils du sommet courant



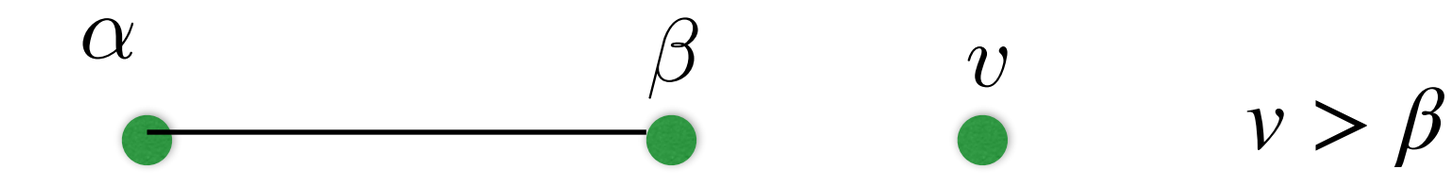


élaguer

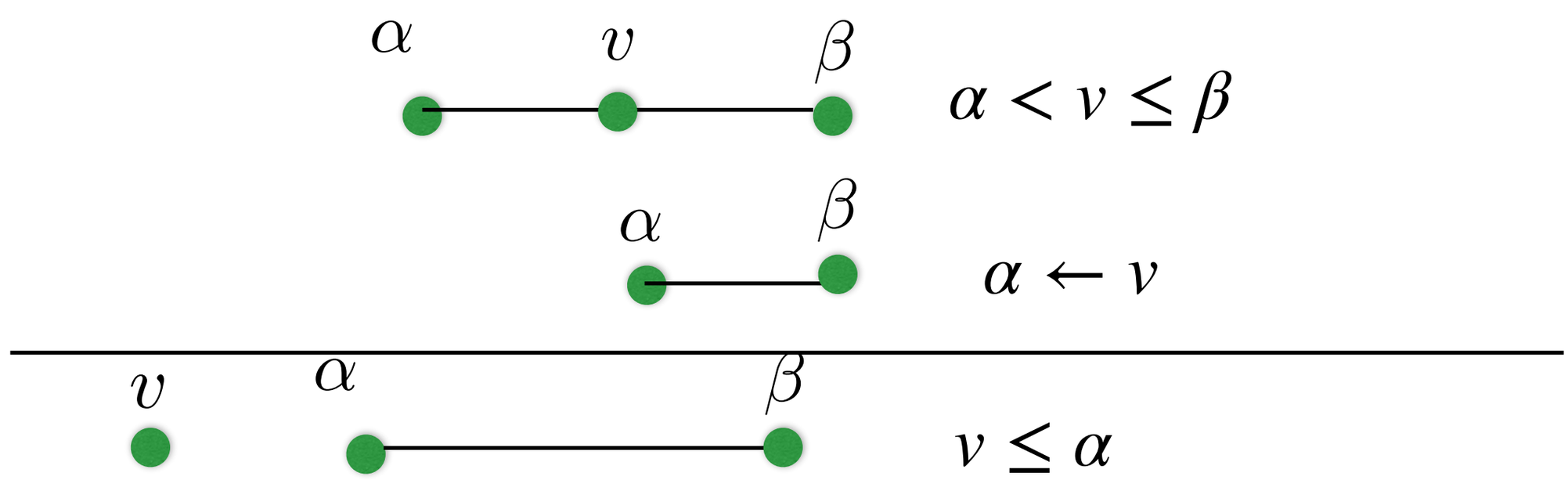
Joueur Max contrôle le sommet

$[\alpha, \beta]$ l'intervalle de sommet courant contrôlé par Max

v – une valeur retournée par un fils du sommet courant



élaguer



pas de modification de α

Algorithme d'élagage α-β

function Alpha-Beta-Search(state) **returns** an action

inputs : *state* - current state in the game

$v \leftarrow \mathrm{MAX}-\mathrm{VALUE}(state,-\infty,+\infty)$ if *state* controlled by Max

$v \leftarrow \mathrm{MIN}-\mathrm{VALUE}(state,-\infty,+\infty)$ if *state* controlled by MIN

return the action in SUCCESSORS(*state*) with value v

Algorithme d'élagage α-β

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*

inputs: *state*, current state in game

α, the value of the best alternative for MAX along the path to *state*

β, the value of the best alternative for MIN along the path to *state*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow -\infty$

for a, s in SUCCESSORS(*state*) **do**

  $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))

  if $v \geq \beta$ **then return** v

  $\alpha \leftarrow$ MAX(α, v)

return v

Algorithme d'élagage α-β

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*

inputs: *state*, current state in game

α, the value of the best alternative for MAX along the path to *state*

β, the value of the best alternative for MIN along the path to *state*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow +\infty$

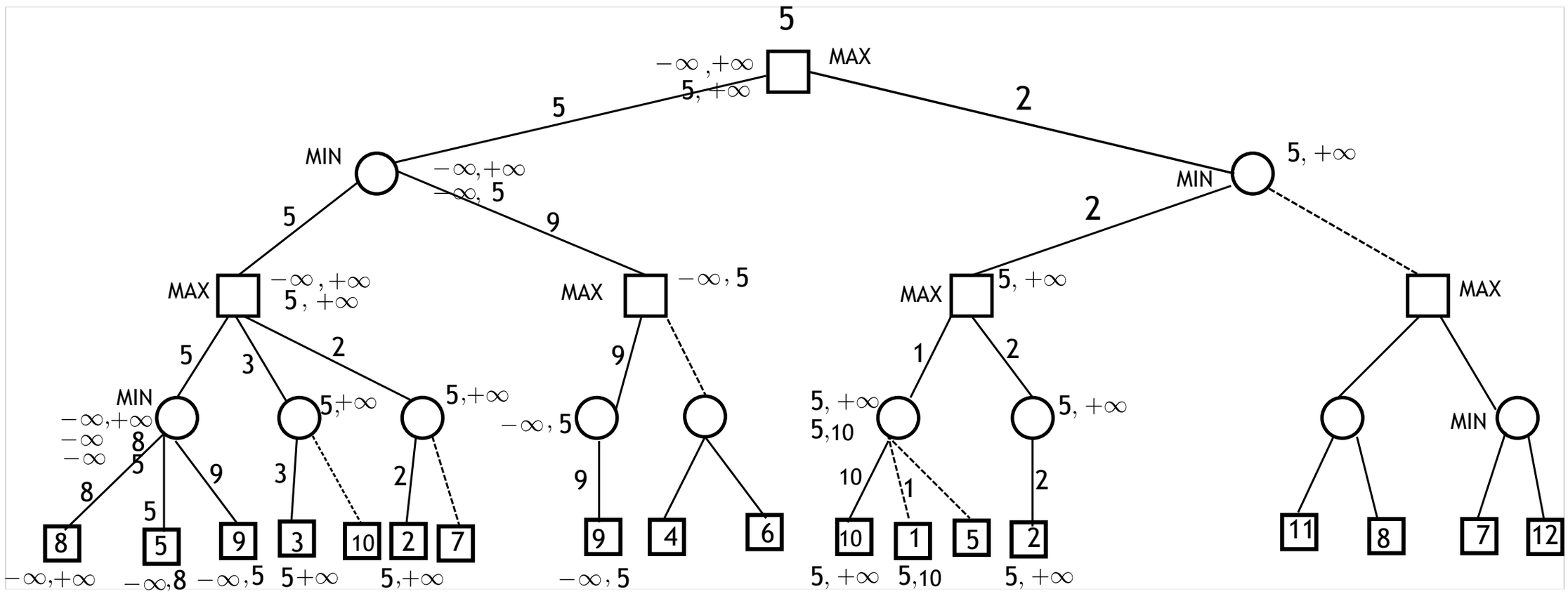
for a, s in SUCCESSORS(*state*) **do**

  $v \leftarrow$ MIN(v, MAX-VALUE(s, α, β))

  if $v \leq \alpha$ **then return** v

  $\beta \leftarrow$ MIN(β, v)

return v


5
MAX
$-\infty, +\infty$
$5, +\infty$
5
2
MIN
$-\infty, +\infty$
$-\infty, 5$
5
9
MIN
$5, +\infty$
MAX
$-\infty, +\infty$
$5, +\infty$
MAX
$-\infty, 5$
2
MAX
$5, +\infty$
MAX
5
3
2
9
1
2
MIN
$-\infty, +\infty$
$-\infty$
$-\infty$
8
5
5,+∞
5,+∞
$-\infty, 5$
$5, +\infty$
5,10
$5, +\infty$
MIN
8
5
9
3
2
9
10
1
2
8
5
9
3
10
2
7
9
4
6
10
1
5
2
11
8
7
12
$-\infty, +\infty$
$-\infty, 8$
$-\infty, 5$
5+∞
5,+∞
$-\infty, 5$
$5, +\infty$
5,10
$5, +\infty$

La fonction d'évaluation

Si la profondeur de l'arbre trop grande on coupe à un niveau d et on applique une heuristique pour évaluer les positions à ce niveau (une estimation de valeur de sommets).

La plupart de fonctions d'évaluation utilisent un certain nombre de propriétés (features) et ensuite on applique une somme pondérée pour évaluer la position :

$$Eval(s) = w_1 f_1(s) + \ldots + w_n f_n(s)$$

par exemple pour évaluer une position dans le jeu d'échec : $w_1 = 9$ et

$f_1(s) =$ le nombre de reines blanches - le nombre de reines noires

Recherche versus consultation

Les échecs : bases de données pour les ouvertures.

Pour les finales de jeu d'échecs : toutes les finales avec jusqu'à 5 pièces et beaucoup avec 6 pièces ont été complètement résolues dans les années 90.

En particulier on a découvert une finale qui termine avec l'échec et mat après 262 pas. Mais les règles du jeu demandent que soit une pièce soit capturée soit un pion avance pendant les derniers 50 pas.

En 2006 toutes les finales à 6 pièces sans pion et certains finales à 7 pièces ont été résolues. Il y a une finale à 7 pièces qui demande au moins 517 pas avant l'échec et mat.

jeux avec les coups stochastiques

Backgammon - dans l'arbre du jeu il y a de noeuds de chances.

La valeur du noeud de chance est égale à l'espérance de valeurs de noeuds fils. Une variante de alpha-beta est possible (quel critère pour élaguer dans un noeud de chance?)

jeux partiellement observables

Kriegspiel chess - variante de jeu d'échec.

Le joueur voit seulement ses propres figures.

Etat d'art

- Jeu de dames. Depuis 2007 le programme CHINOOK joue parfaitement. Utilise alpha-beta élagage et une base de données de 39×10^{12} de jeux finals.

- Echecs - Deep Blue a vaincu Garry Kasparov en 1997. Une fonction d'évaluation sophistiquée qui utilisait 8000 caractéristiques. DB la recherche jusqu'au profondeur 14 mais parfois capable d'examiner les positions jusqu'au profondeur 40 (un bon joueur - profondeur 8, un grand maître — profondeur 12). Large base de donnée d'ouvertures et de finales.

- Null move heuristic : évaluation d'une position (sous-estimation) on donnant deux pas à l'adversaire au début et ensuite méthode classique jusqu'à un faible profondeur. Bons résultats pour les échecs.

- Hydra (successeur de Deep Blue), un cluster de 64 processeurs, 1G par processus, 200 millions d'évaluation par seconde, la recherche jusqu'au profondeur 18. Le meilleurs programme RYBKA mais la fonction d'évaluation inconnue.

- Othello (Riversi) - 1997 le programme Logistello a battu le champion mondial.