Intelligence Artificielle

Recherche Adverse
(Jeux)

Stephane Marchand-Maillet

Contenu

- Contexte: recherche adverse
- Algorithme MIN-MAX
- Elagage α-β

Jeux

- Prototype de l'Intelligence:

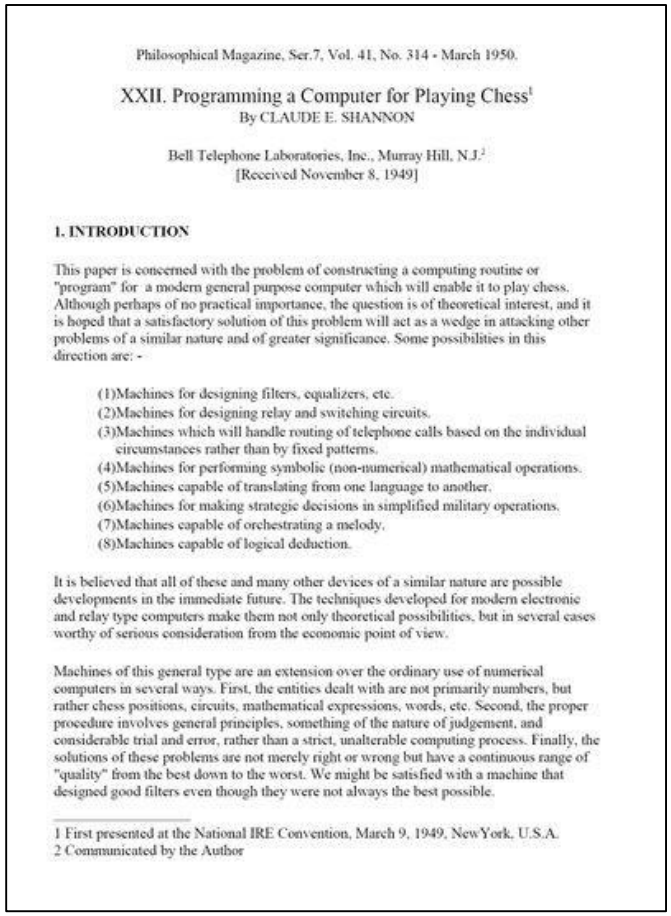
  Savoir jouer contre/battre un adversaire (intelligent)

- ont toujours été un support au développement de l'IA
  - Cohérents avec le Test de Turing
- Premier article sur l'IA (1950):

Programming a Computer for Playing Chess.
CLAUDE E. SHANNON Philosophical Magazine, Ser.7, Vol. 41, No. 314 – March 1950

This paper is concerned with the problem of constructing a computing routine or "program" for a modern general purpose computer which will enable it to play chess. Although perhaps of no practical importance, the question is of theoretical interest, and it is hoped that a satisfactory solution of this problem will act as a wedge in attacking other problems of a similar nature and of greater significance. Some possibilities in this direction are: -

(1)Machines for designing filters, equalizers, etc.
(2)Machines for designing relay and switching circuits.
(3)Machines which will handle routing of telephone calls based on the individual circumstances rather than by fixed patterns.
(4)Machines for performing symbolic (non-numerical) mathematical operations.
(5)Machines capable of translating from one language to another.
(6)Machines for making strategic decisions in simplified military operations.
(7)Machines capable of orchestrating a melody.
(8)Machines capable of logical deduction.



Philosophical Magazine, Ser.7, Vol. 41, No. 314 - March 1950.

XXII. Programming a Computer for Playing Chess[1]
By CLAUDE E. SHANNON

Bell Telephone Laboratories, Inc., Murray Hill, N.J.[2]
[Received November 8, 1949]

1. INTRODUCTION

This paper is concerned with the problem of constructing a computing routine or "program" for a modern general purpose computer which will enable it to play chess. Although perhaps of no practical importance, the question is of theoretical interest, and it is hoped that a satisfactory solution of this problem will act as a wedge in attacking other problems of a similar nature and of greater significance. Some possibilities in this direction are: -

(1)Machines for designing filters, equalizers, etc.
(2)Machines for designing relay and switching circuits.
(3)Machines which will handle routing of telephone calls based on the individual circumstances rather than by fixed patterns.
(4)Machines for performing symbolic (non-numerical) mathematical operations.
(5)Machines capable of translating from one language to another.
(6)Machines for making strategic decisions in simplified military operations.
(7)Machines capable of orchestrating a melody.
(8)Machines capable of logical deduction.

It is believed that all of these and many other devices of a similar nature are possible developments in the immediate future. The techniques developed for modern electronic and relay type computers make them not only theoretical possibilities, but in several cases worthy of serious consideration from the economic point of view.

Machines of this general type are an extension over the ordinary use of numerical computers in several ways. First, the entities dealt with are not primarily numbers, but rather chess positions, circuits, mathematical expressions, words, etc. Second, the proper procedure involves general principles, something of the nature of judgement, and considerable trial and error, rather than a strict, unalterable computing process. Finally, the solutions of these problems are not merely right or wrong but have a continuous range of "quality" from the best down to the worst. We might be satisfied with a machine that designed good filters even though they were not always the best possible.

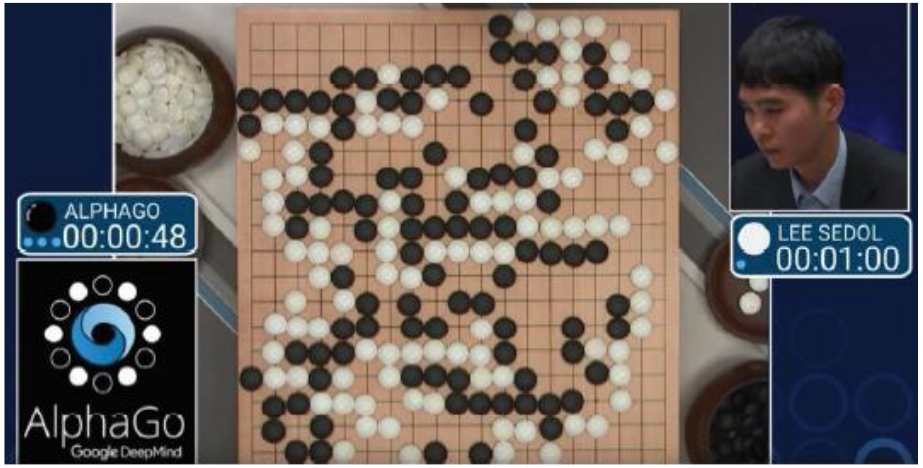
1 First presented at the National IRE Convention, March 9, 1949, NewYork, U.S.A.
2 Communicated by the Author

https://vision.unipv.it/IA1/aa2009-2010/ProgrammingaComputerforPlayingChess.pdf

Historique

- 1945 : Turing présente les échecs comme ce que peuvent faire les ordinateurs.
- 1946 : Turing parle d'intelligence des machines, en relation avec les échecs.
- 1950 : Turing écrit le premier programme de jeux.
- 1950 : Shannon écrit le premier article sur les jeux.
- 1957 : Prédiction de Simon : dans 10 ans, le champion du monde d'échecs sera un ordinateur.
- 1963 : Le programme de jeu de dames de Samuel gagne contre le champion du monde
- 1989 : Création du World Man Machine Championship.

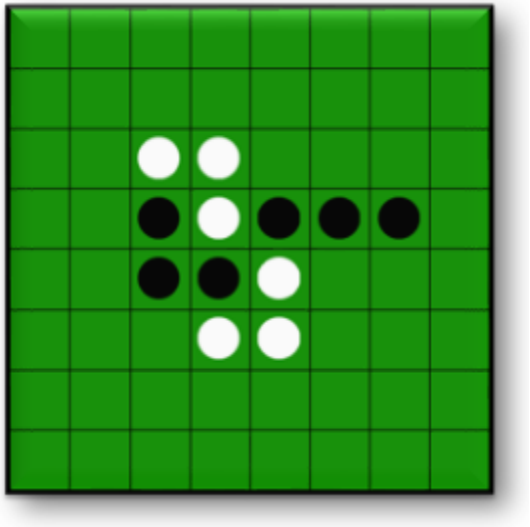
1997: Deep Blue *vs* Kasparov



2016: AlphaGo *vs* Lee Sedol

Exemple

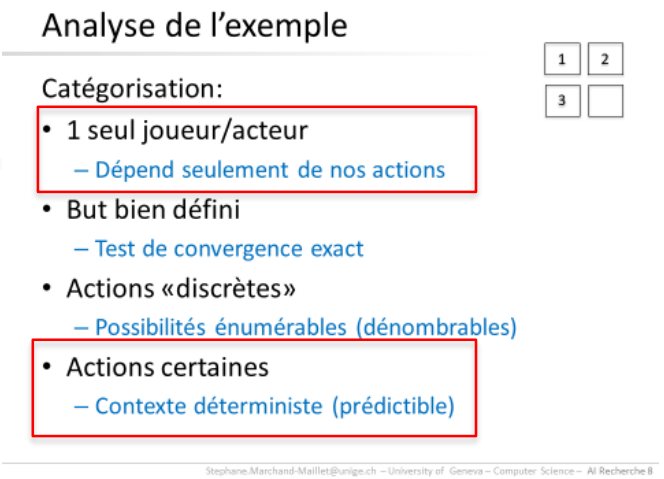
Reversi (Othello)



But: placer le plus de pions de sa couleur

Jeu: encadrer des pions adverses pour les « renverser »

Analyse de l'exemple



Catégorisation:

- 2 joueurs
  - – Ne dépend pas seulement de nos actions
- But bien défini
  - – Gagner la partie
- Actions «discrètes»
  - – Possibilités énumérables (dénombrables)
- Actions
  - – Certaines: contexte déterministe (prédictible)
  - – Incertaines (adversaire)

Autres exemples

- Morpion
- Jeux de plateau
  - Othello
  - Dames (*Chinook*)
  - Echecs (*DeepBlue*)
  - Go (AlphaGO)
- …

Adversaire

On résout l'incertitude sur les choix de l'adversaire en lui attribuant une stratégie de jeu

On fait les hypothèses :

- L'adversaire peut observer les mêmes états que nous (pas d'information supplémentaire)
- L'adversaire joue en alternance avec nous
- L'adversaire veut gagner la partie (pire des cas)
- L'adversaire est cohérent dans sa stratégie

Formulation: Jeux

On formule le problème comme une exploration de l'espace des configurations

- Etats: Configurations de jeu
- Transition: choix d'une nouvelle configuration contraint par les règles du jeu
- Etat initial: Configuration de départ et joueur initial
- Etat final: égalité ou victoire d'un des joueurs

Fonction d'utilité et recherche

Jeux à somme nulle:

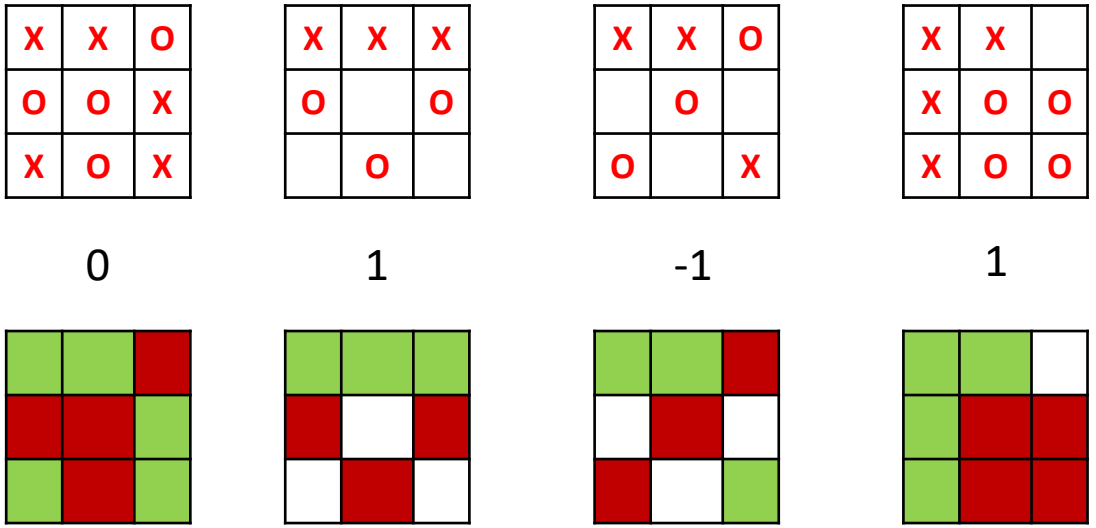
- La fonction d'utilité mesure le résultat du jeu
  - Gagnant +1
  - Egalité: 0
  - Perdant: -1

La recherche estime le meilleur coup à jouer (au tour de notre joueur) pour maximiser l'utilité

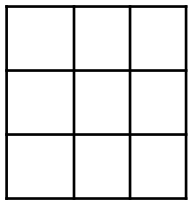
Exemple

Jeu du Morpion:

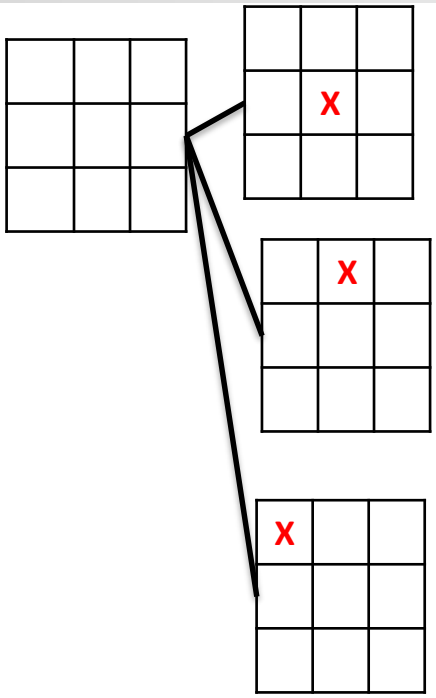
- Symboles **X** et **O** placés alternativement
- But: aligner 3 symboles identiques en ligne, colonne ou diagonale



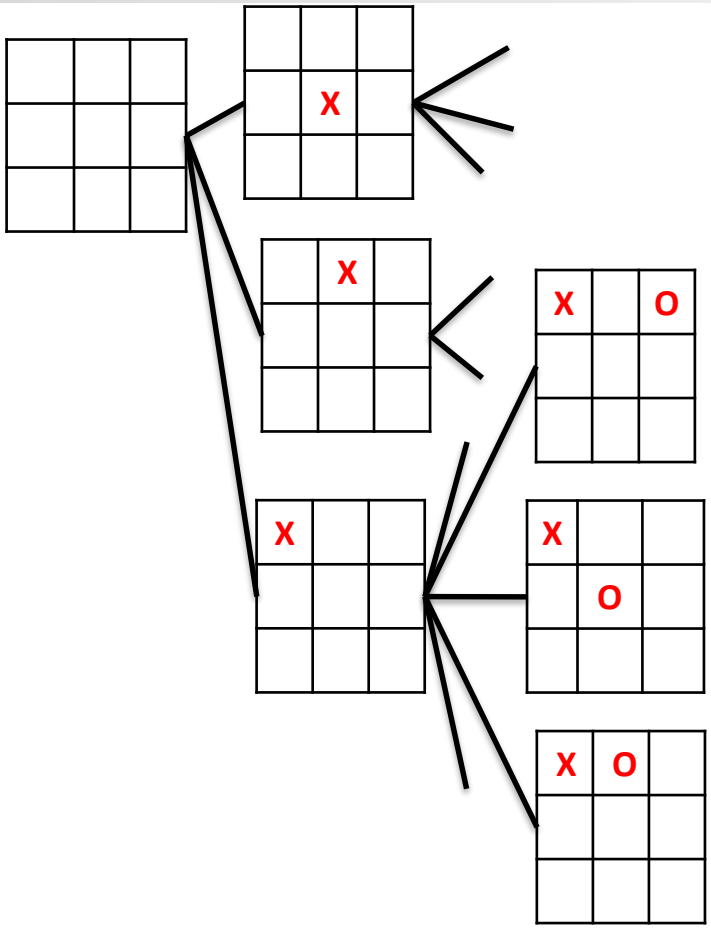
Arbre de jeu (incomplet)



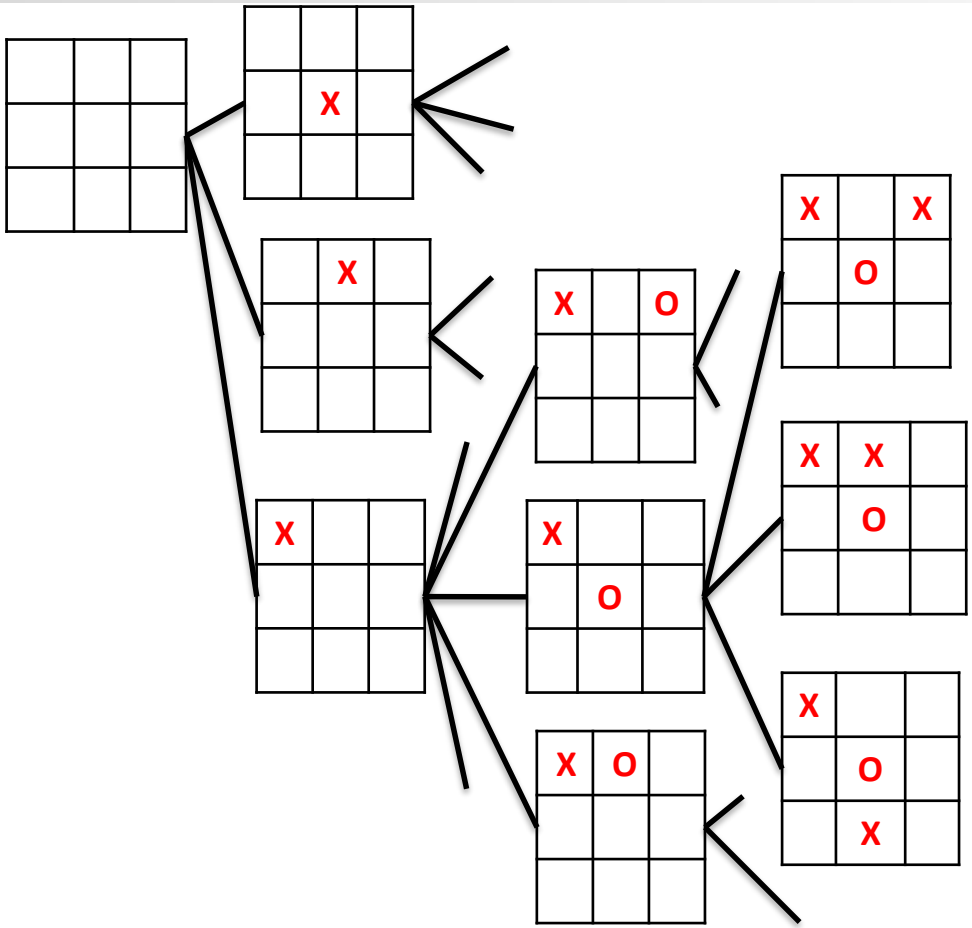
Arbre de jeu (incomplet)



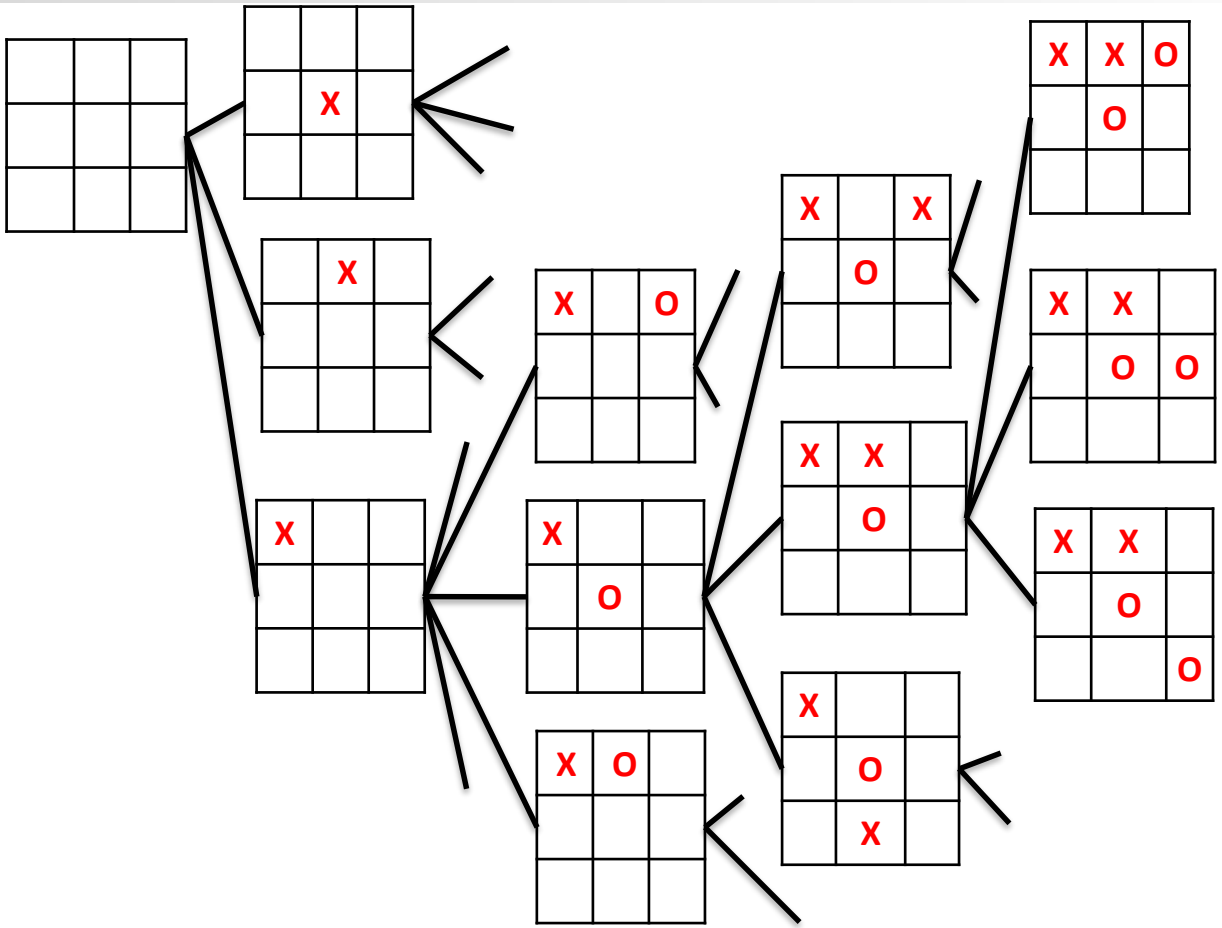
Arbre de jeu (incomplet)



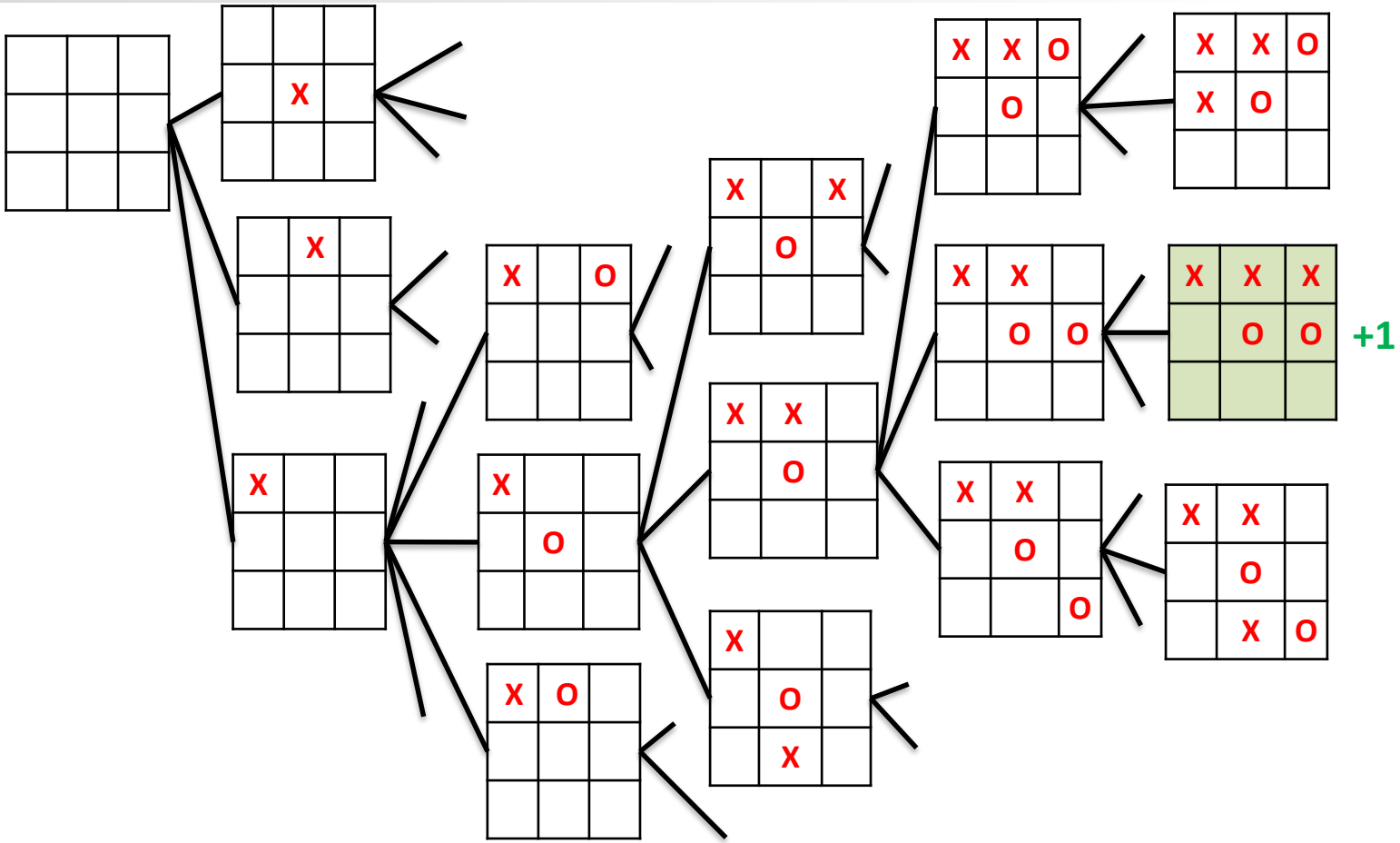
Arbre de jeu (incomplet)



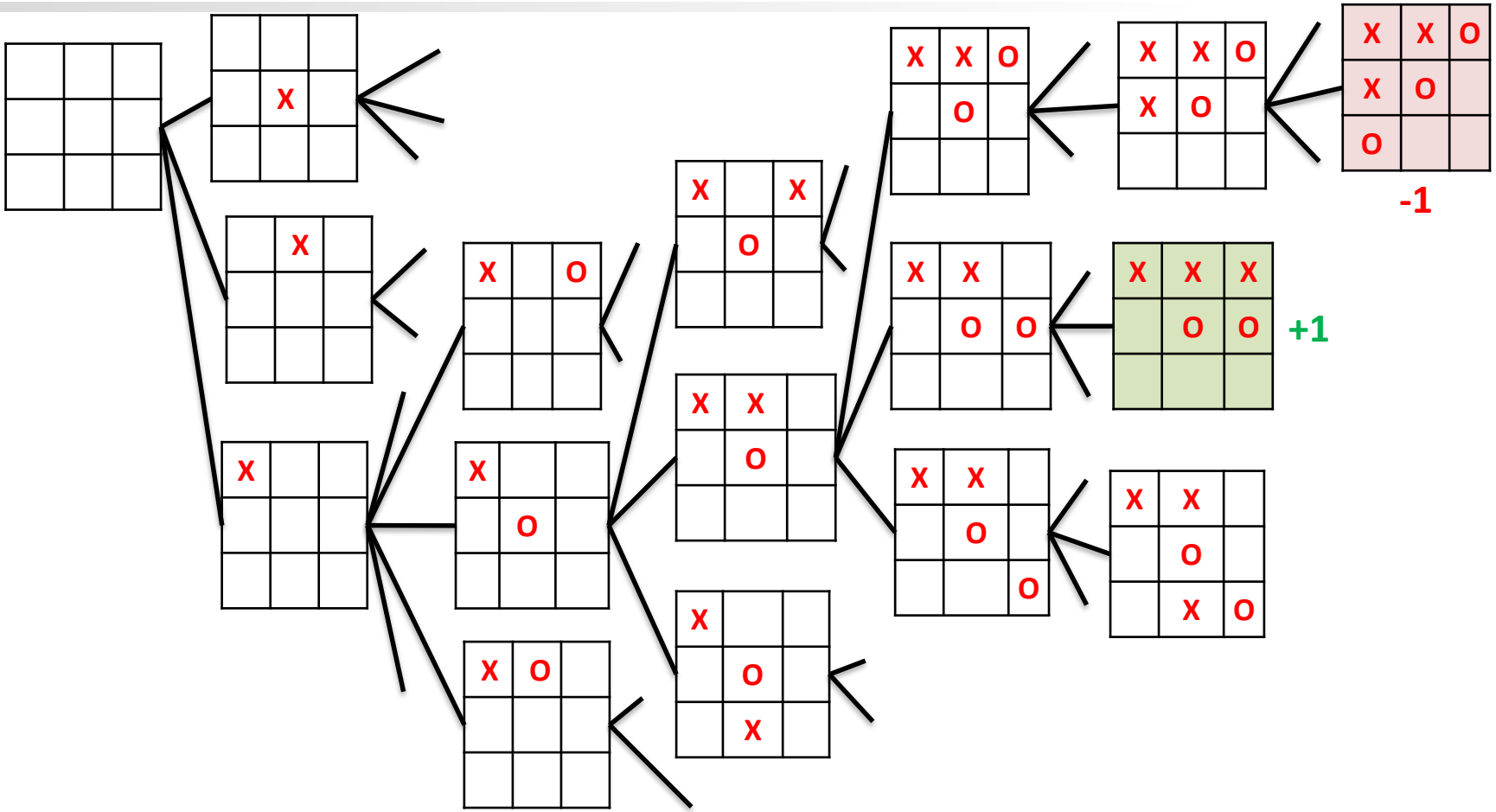
Arbre de jeu (incomplet)



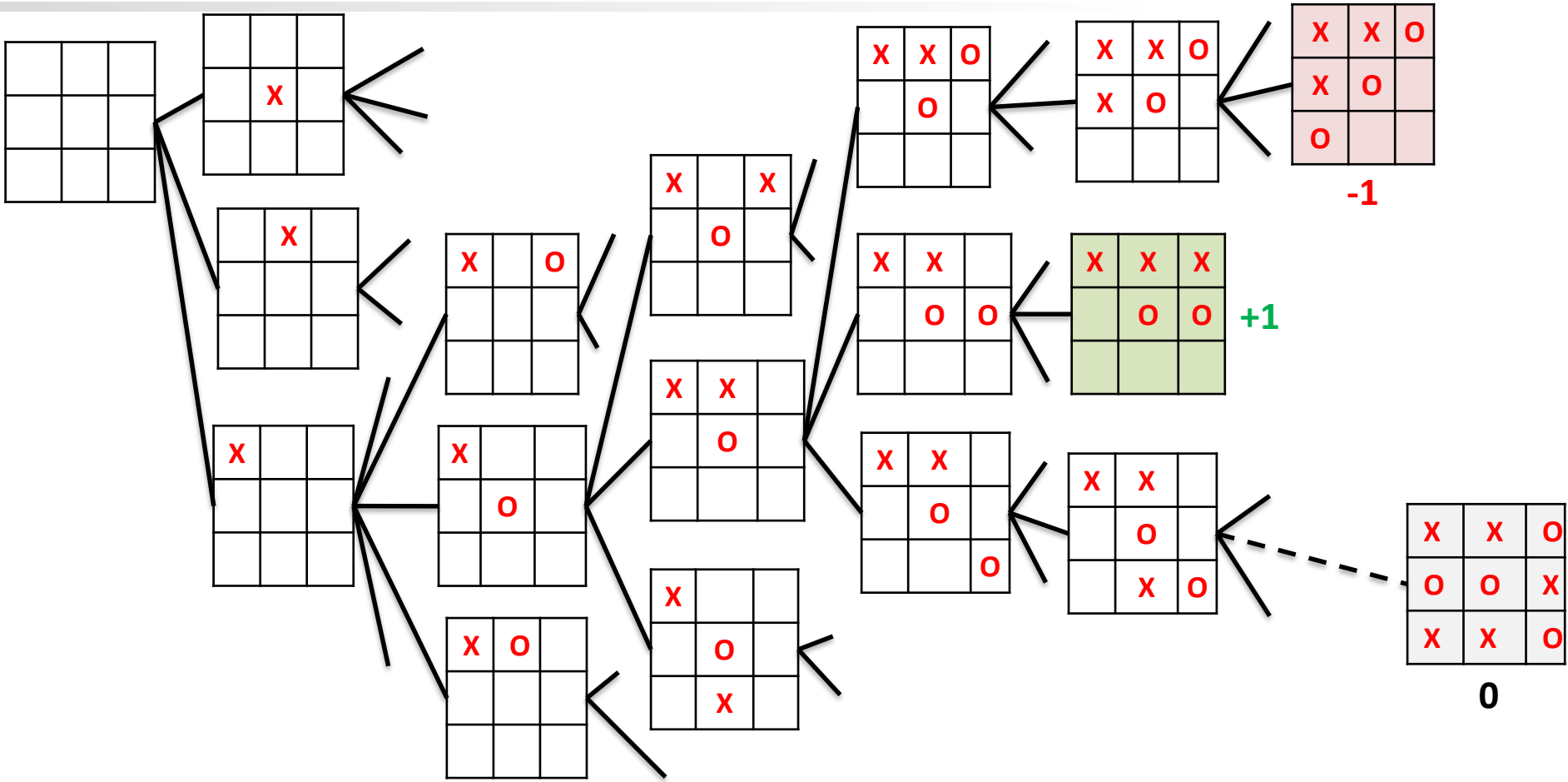
Arbre de jeu (incomplet)



Arbre de jeu (incomplet)



Arbre de jeu (incomplet)



Impossible à développer en entier pour des jeux non triviaux

Fonction d'évaluation

Permet de guider notre stratégie pour le choix des coups à jouer:

$$f(v): V \rightarrow \mathbb{R}$$

→ Estimation (numérique) du potentiel d'une configuration (état) *s*=state(*v*) à mener à la victoire (relativement aux autres états)

En général, une fonction des caractéristiques du jeu (valeurs des pièces, nombres, position, …)

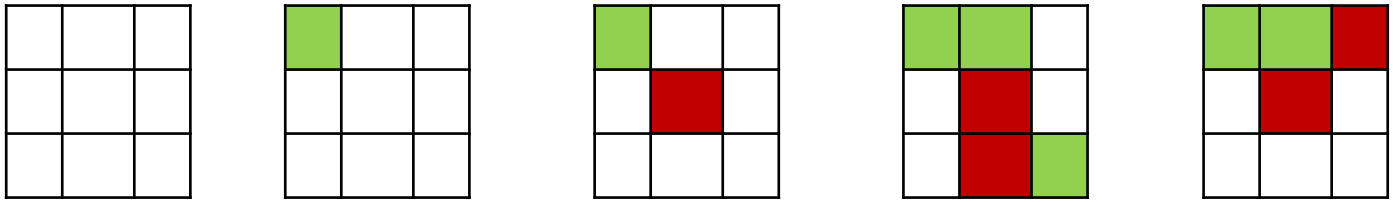
→On cherche à MAXimiser *f*
→L'adversaire à MINimiser *f*

} Joueurs: MAX *vs* MIN

→On peut ne pas développer l'arbre jusqu'à la conclusion du jeu

Exemple

Jeu du Morpion:



Fonction d'évaluation:

$f(v)$ = (# lignes/colonnes/diagonales ouvertes pour MAX)

- (# lignes/colonnes/diagonales ouvertes pour MIN)

Recherche avec horizon

On développe l'arbre avec un horizon de profondeur limité (profondeur *M*):

→Pour satisfaire des contraintes de mémoire, ou de temps de recherche

→Les feuilles de l'arbre ne sont pas décisives (gagnant/perdant/égalité)

→On calcule leur fonction d'évaluation $f(v)$ et on propage ces valeurs vers le haut

→Algorithme MiniMax

Algorithme MiniMax

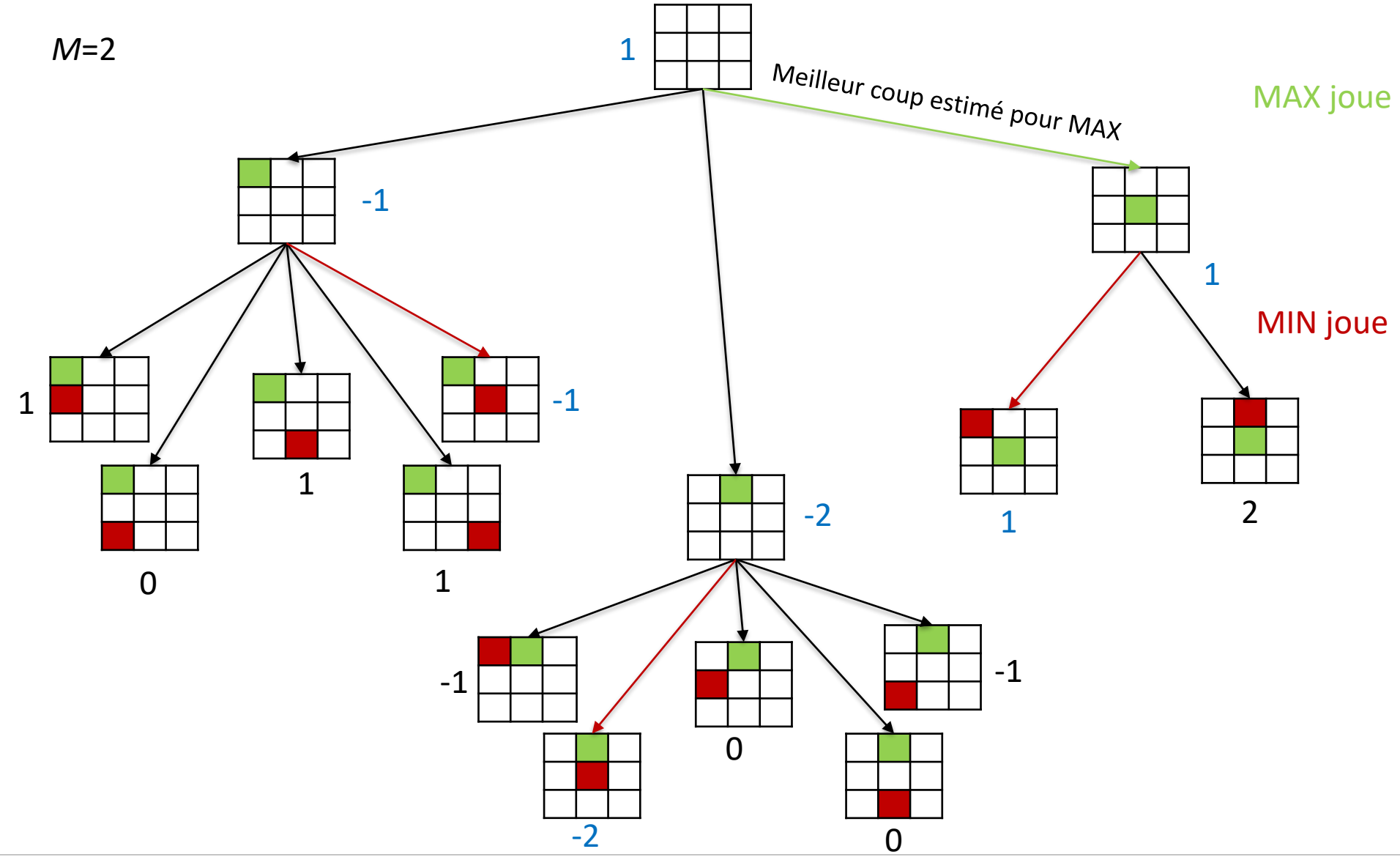
On ne remonte pas simplement les valeurs le long du chemin, on doit prendre en compte le fait que l'on joue contre un adversaire:

- MAX cherche à maximiser la fonction d'évaluation
  →Il fera le choix du coup menant au maximum des valeurs
- MIN cherche à minimiser la fonction d'évaluation
  →Il fera le choix du coup menant au minimum des valeurs

→On rétro-propage selon cette logique

Exemple

(modulo symétries)



Algorithme MiniMax

En partant d'un nœud où MAX doit jouer:

1. Développer l'arbre jusqu'à la profondeur fixée
2. Evaluer $f(v)$ pour toutes les feuilles v de cet arbre
3. Rétro-propager ces valeurs selon:
   a. Un nœud où MAX doit jouer reçoit le maximum des valeurs de ses enfants
   b. Un nœud où MIN doit jouer reçoit le minimum des valeurs de ses enfants
4. Jouer selon le chemin de la valeur remontée

Si f est une bonne estimation, on maximise nos chances de victoire

On itère jusque ce qu'un nœud terminal (décisif) soit accessible avec la profondeur fixée

Propriétés

- Complet: oui si l'arbre de jeu est fini

- Optimal: oui si l'adversaire est aussi optimal

- Complexité en temps: $O(b^M)$
  - branchement moyen b
  - Profondeur maximale M
- Complexité en espace: O(b.M)
  - Recherche en profondeur

Exemple: Echecs
- b=35, novice: M=4, expert M=8
- Fonction d'évaluation complexe à déterminer

→ On veut M=20 => 2.10^{30} coups à développer

→ On doit réduire l'expansion de l'arbre

Elagage α-β

Principe: ne plus développer les branches de l'arbre correspondant aux coups sans intérêt (au sens MiniMax)

→ Algorithme exact: trouve la même solution

On affecte:

- Une valeur α aux nœuds MAX (où MAX doit jouer)
- Une valeur β aux nœuds MIN (où MIN doit jouer)

On maintient:

- α = valeur du meilleur successeur jusqu'ici ($-\infty$ au départ)
- β = valeur du plus faible successeur jusqu'ici ($+\infty$ au départ)

Donc:

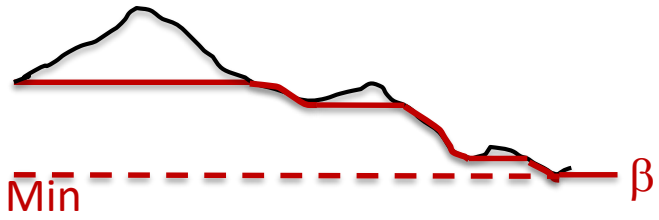
- α = max → ne peut que croître, ne peut pas décroître
  - borne inférieure de la valeur finale
- β = min → ne peut que décroître, ne peut pas croître
  - borne supérieure de la valeur finale

Elagage α-β

Principe: ne plus développer les branches de l'arbre correspondant aux coups sans intérêt (au sens MiniMax)

→ Algorithme exact: trouve la même solution

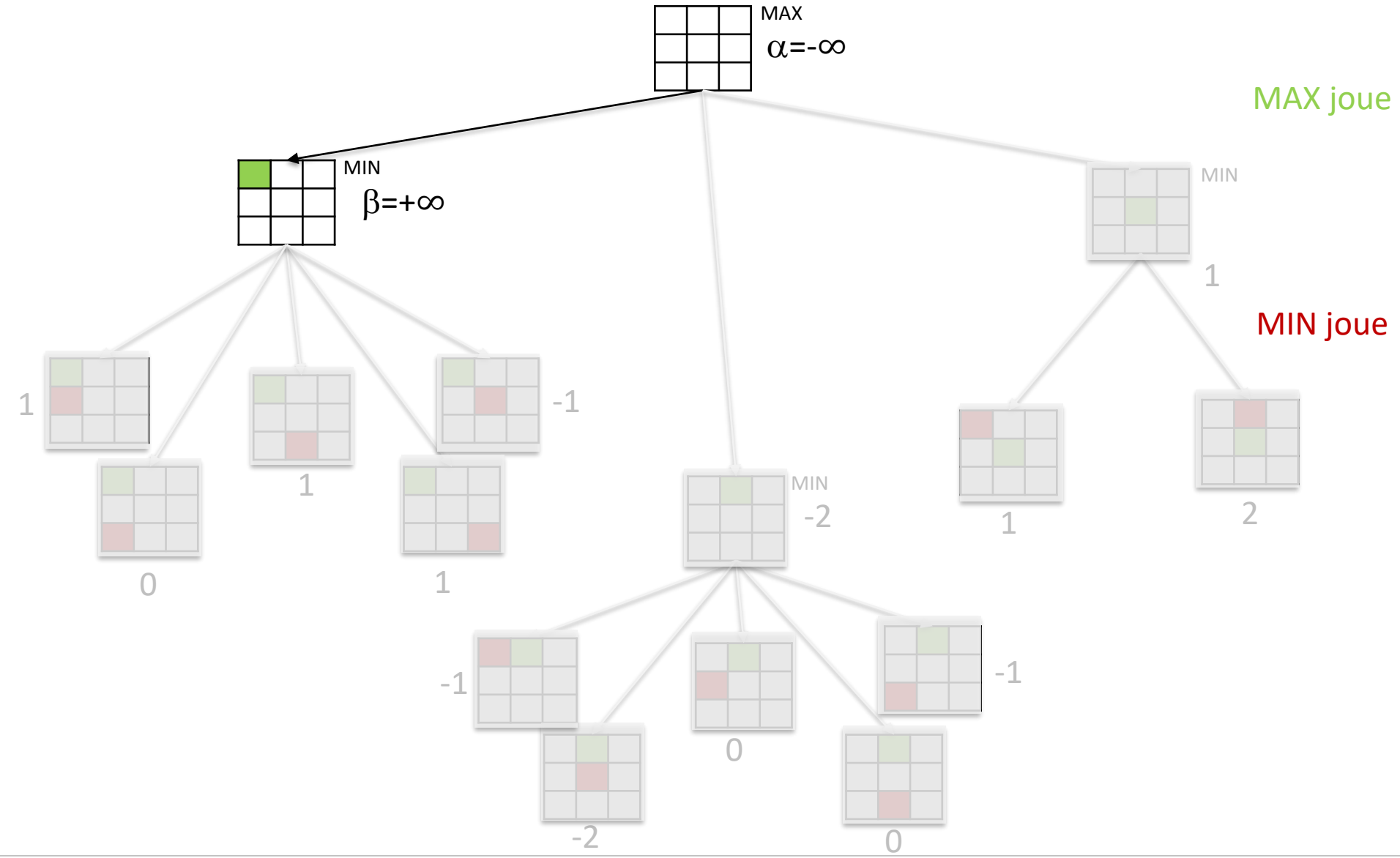
- α = max → ne peut que croître, ne peut pas décroître
  - borne inférieure de la valeur finale
- β = min → ne peut que décroître, ne peut pas croître
  - borne supérieure de la valeur finale



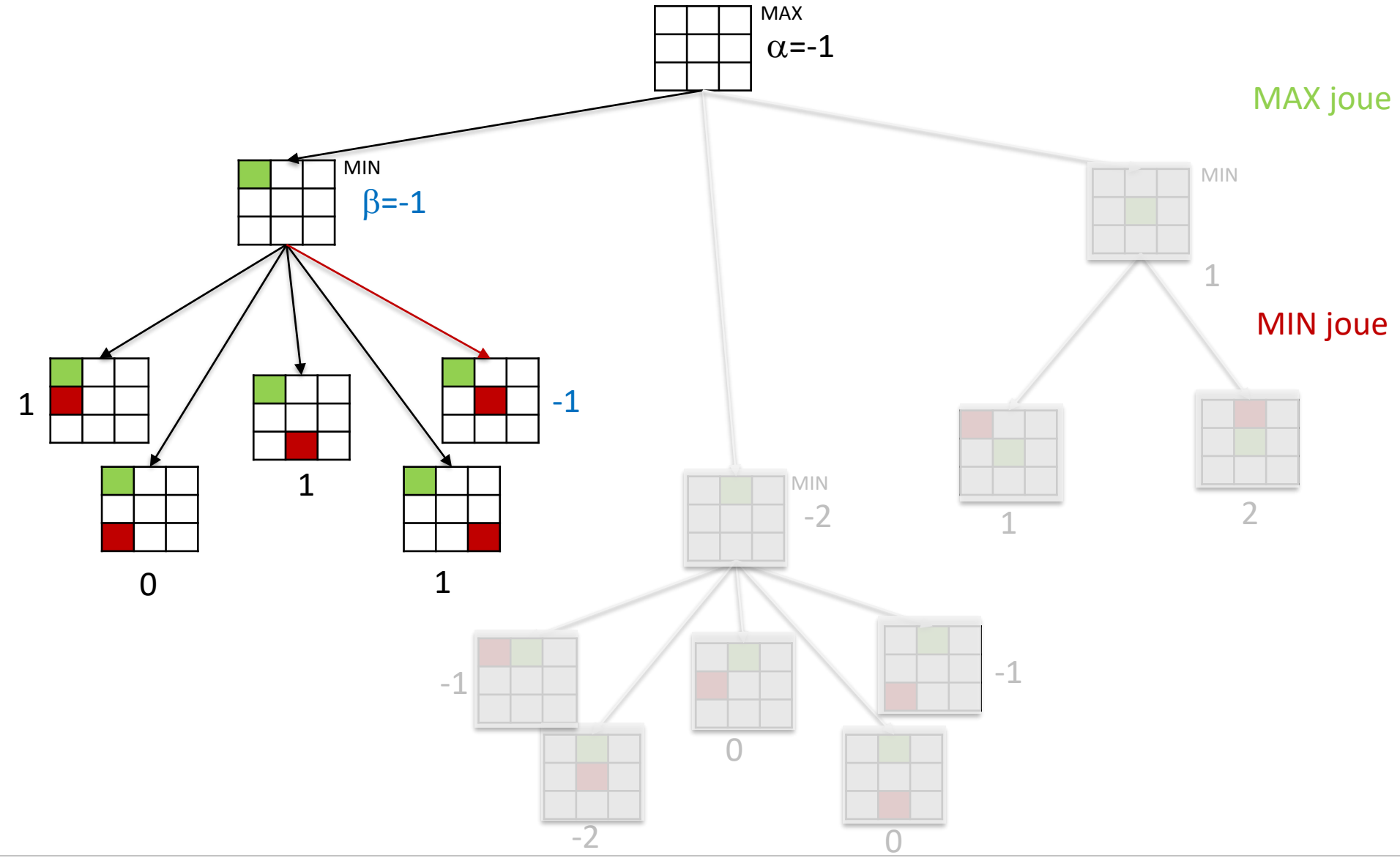
On interrompt la recherche au-dessous de *v* si:

- $\alpha(v) \geq \beta(v')$ avec *v* nœud MAX et *v'* ancêtre de *v*
- $\beta(v) \leq \alpha(v')$ avec *v* nœud MIN et *v'* ancêtre de *v*

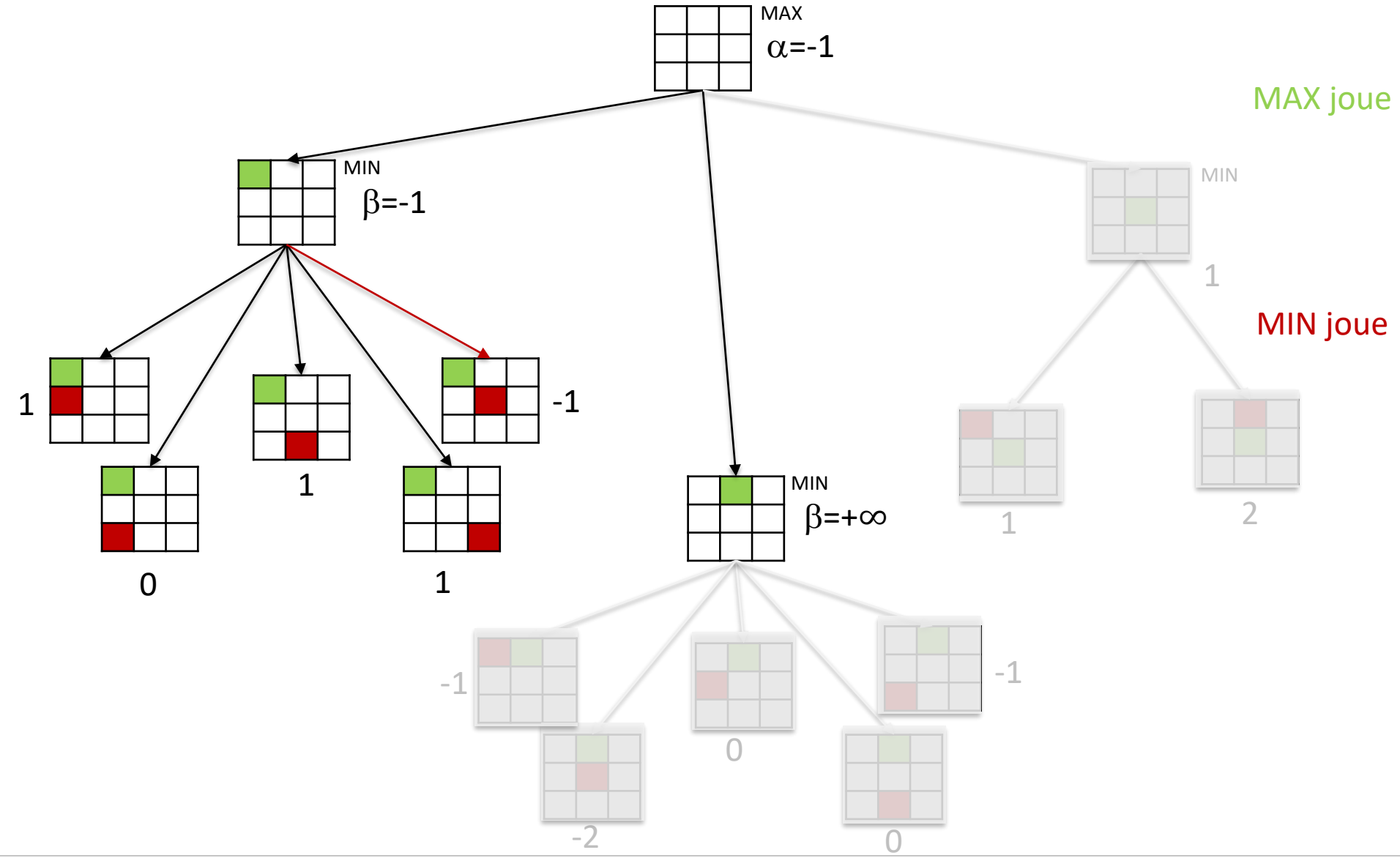
Exemple



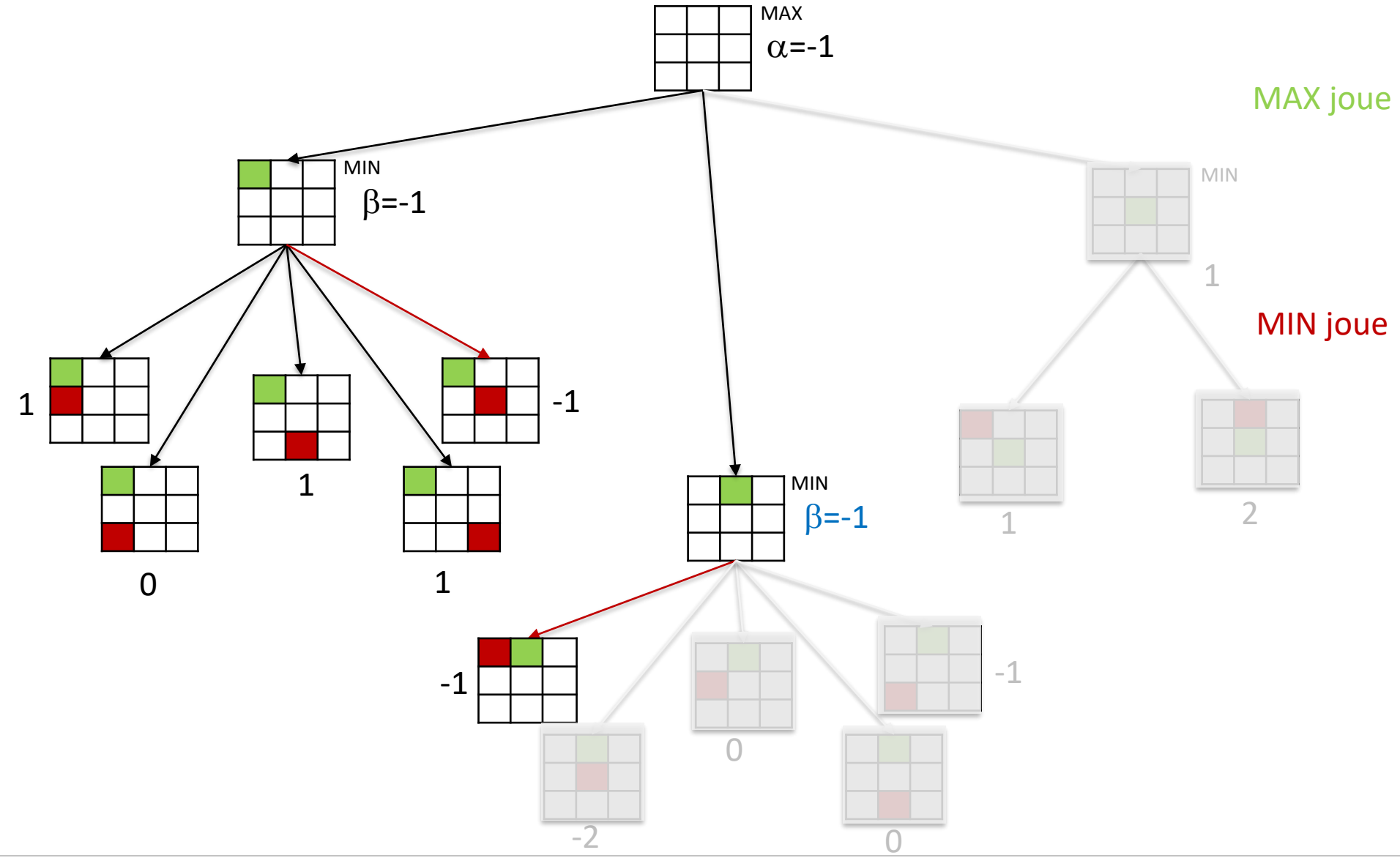
Exemple



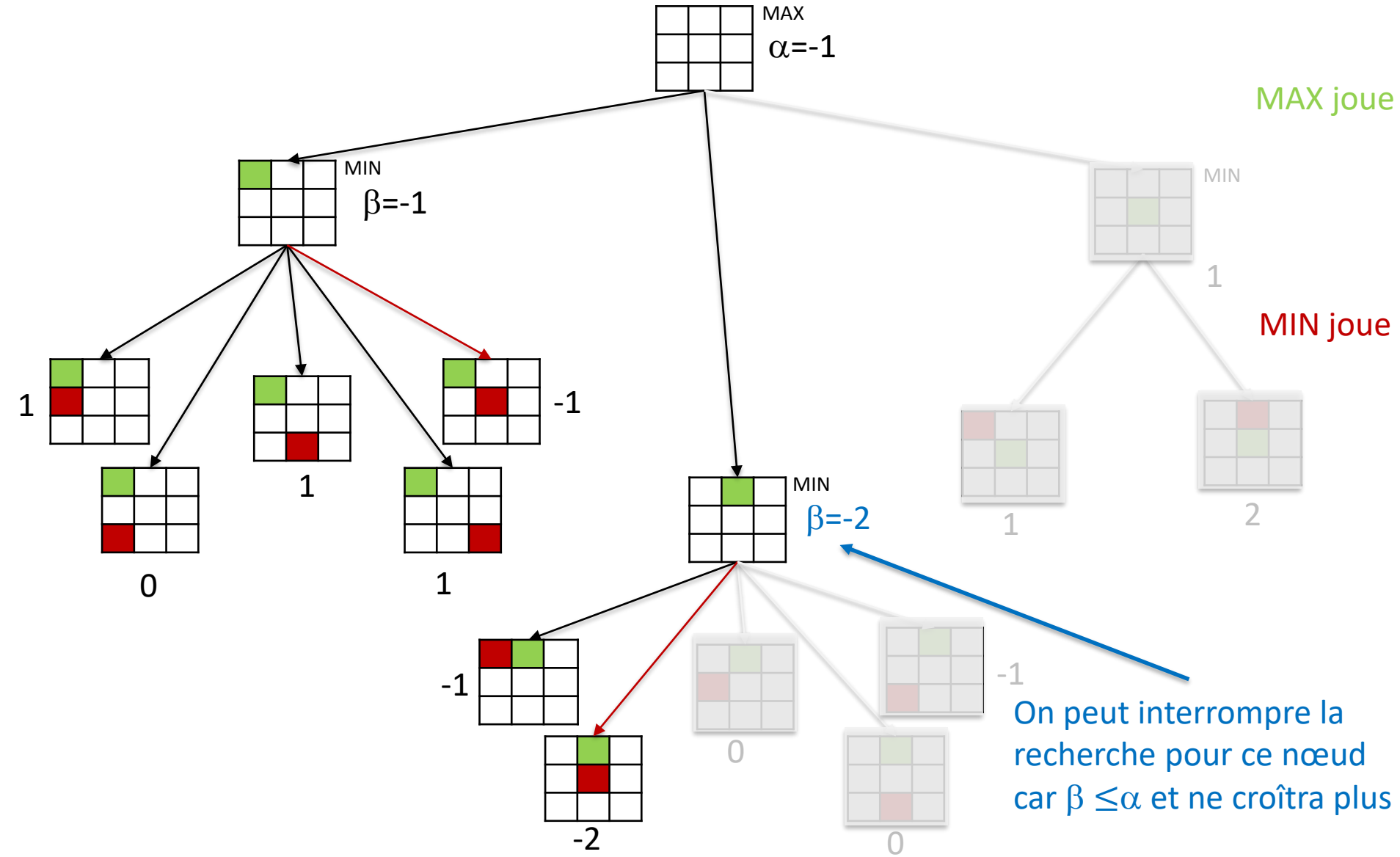
Exemple



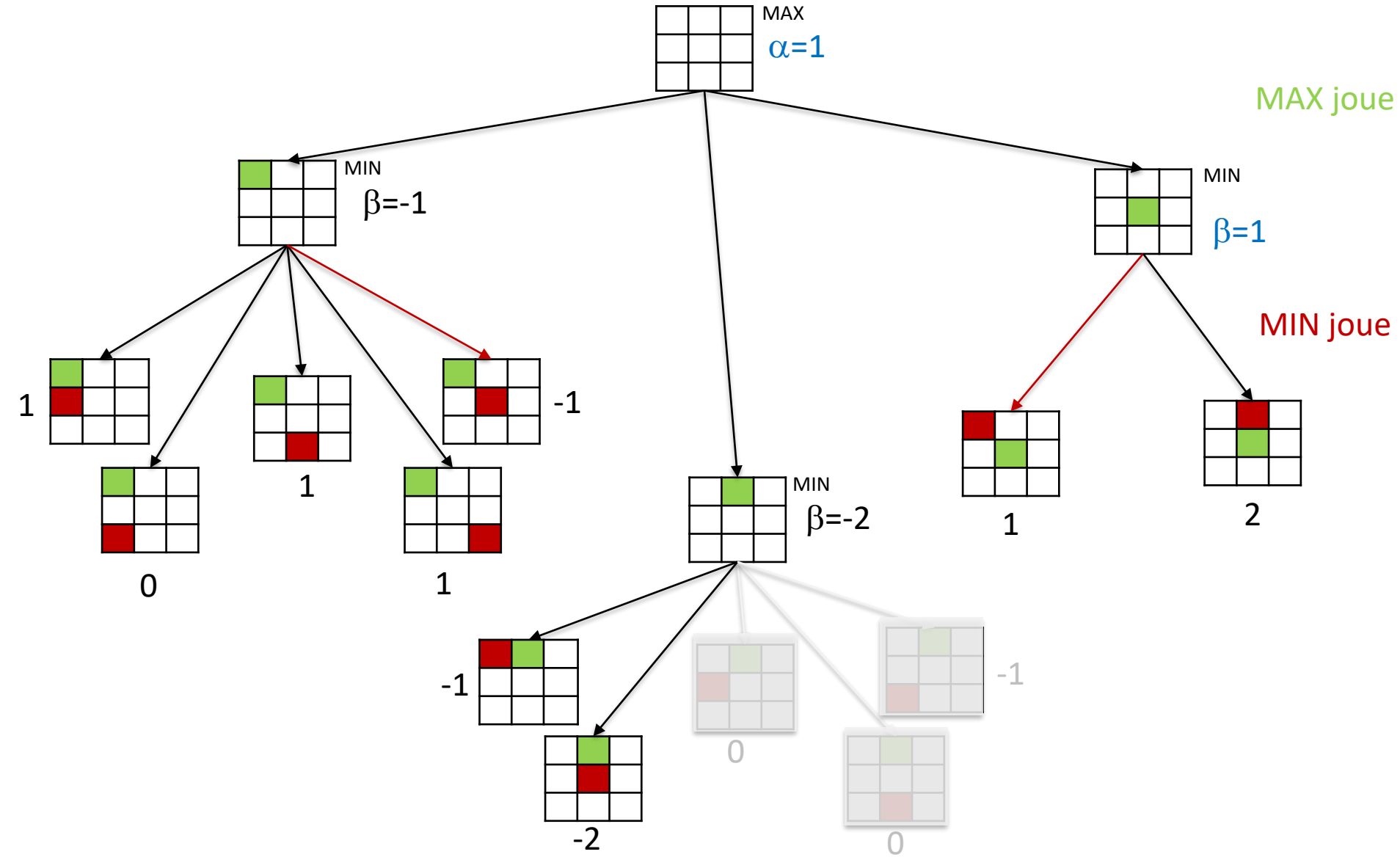
Exemple



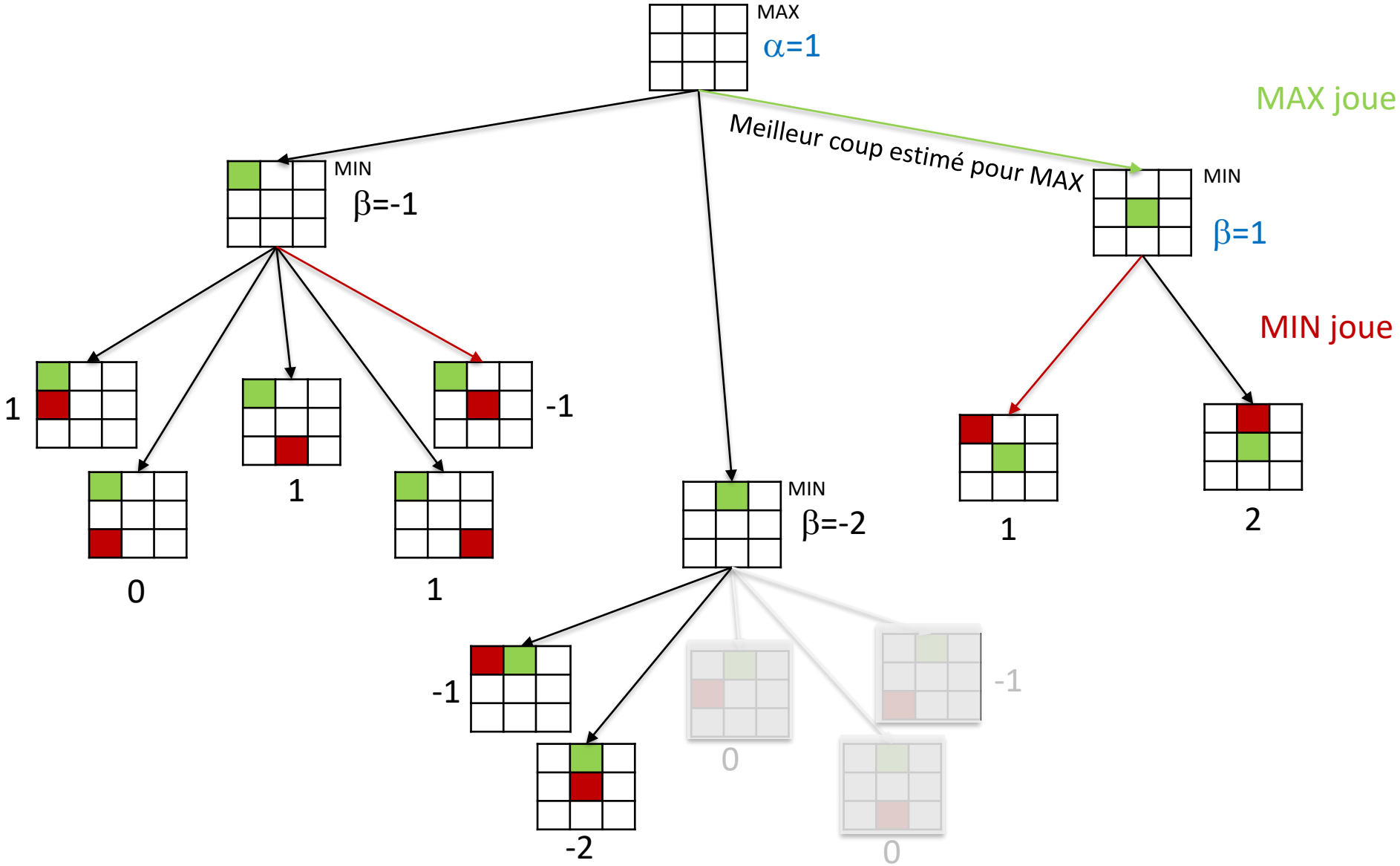
Exemple


MAX
α=-1
MAX joue
MIN
β=-1
MIN
1
MIN joue
1
-1
1
1
MIN
β=-2
0
1
2
-1
0
-1
0
-2
On peut interrompre la recherche pour ce nœud car β ≤α et ne croîtra plus

Exemple

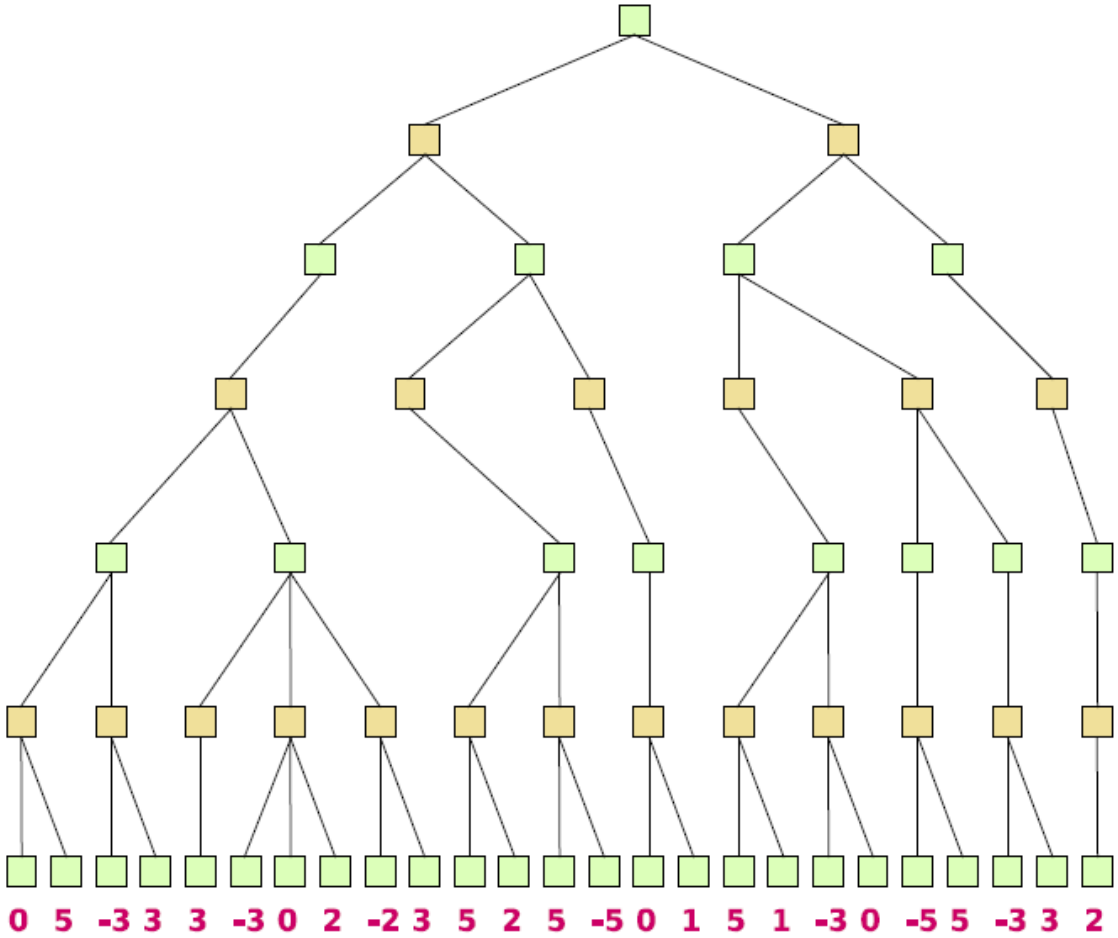


Exemple


MAX
α=1
MAX joue
Meilleur coup estimé pour MAX
MIN
β=-1
MIN
β=1
MIN joue
1
-1
1
1
0
1
MIN
β=-2
2
-1
-1
0
0
-2

Elagage α-β

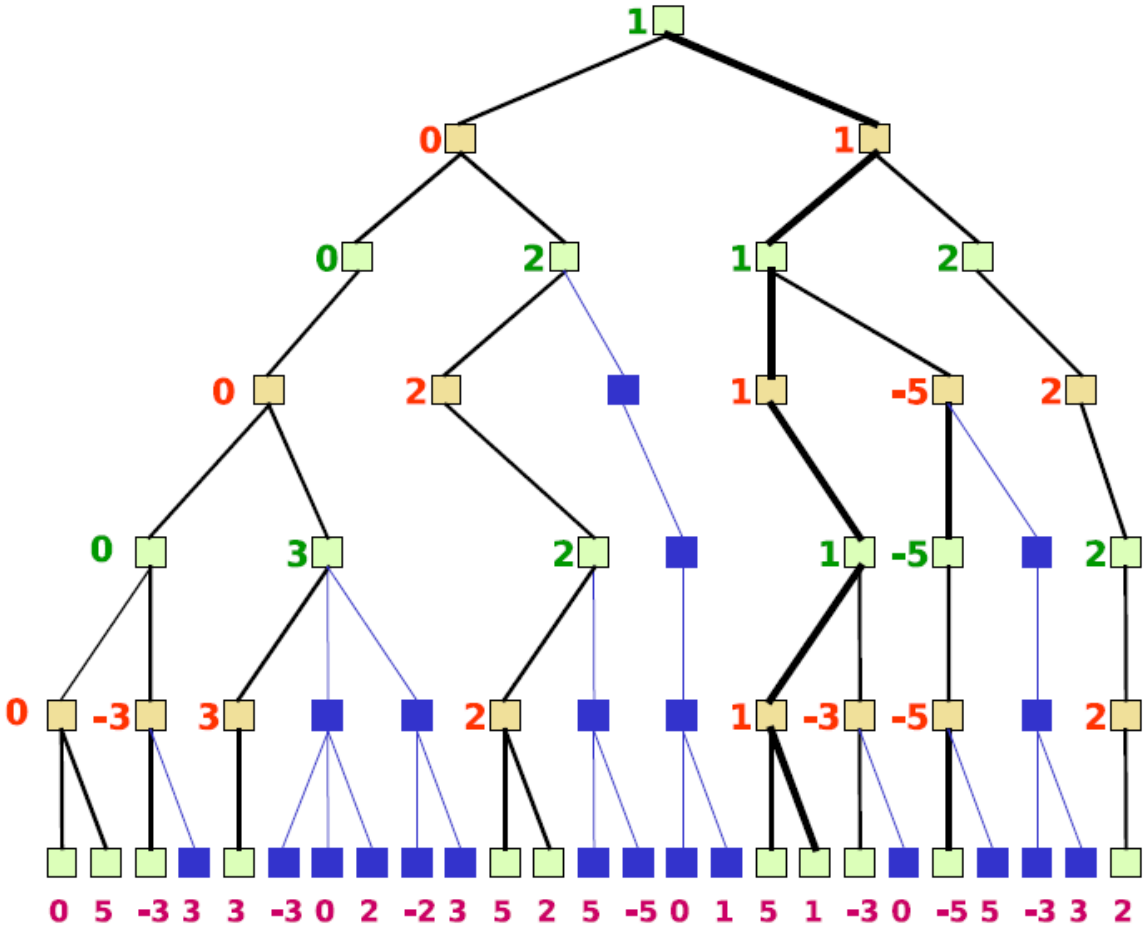
Exercice: Marquer l'élagage (slide suivant)



On interrompt la recherche au-dessous de *v* si:

- $\alpha(v) \geq \beta(v')$ avec *v* nœud MAX et *v'* ancêtre de *v*
- $\beta(v) \leq \alpha(v')$ avec *v* nœud MIN et *v'* ancêtre de *v*

Elagage α-β



On interrompt la recherche au-dessous de *v* si:

- $\alpha(v) \geq \beta(v')$ avec *v* nœud MAX et *v'* ancêtre de *v*
- $\beta(v) \leq \alpha(v')$ avec *v* nœud MIN et *v'* ancêtre de *v*

Propriétés de l'élagage

Algorithme exact: on trouve la même solution que si on developpait l'arbre complet

Le gain est maximum pour l'élagage a-b quand:

- Les enfants d'un nœud MAX sont ordonnés selon leur valeurs décroissantes
- Les enfants d'un nœud MIN sont ordonnés selon leur valeurs croissantes

Dans ce cas on a une complexité en temps $O(b^{M/2})$ et un facteur de branchement $\sqrt{b}$

Si l'ordre est quelconque on a un gain moyen et une complexité $O(b^{3M/4})$

→ Permet d'augmenter M: si MiniMax est faisable pour M=10 alors l'élagage permet M=13

Alternatives & évolutions

- NegaMax: pas d'alternance des fonctions min et max. Simplement une fonction évaluée à un niveau dont on prend l'opposé au niveau suivant

- ExpectMiniMax: permet de modéliser les jeux dans lesquels un aléa est présent (exple: backgammon)

AlphaGO (b=360)

Stratégie différente: *Reinforcement Learning*

- Apprentissage d'une stratégie (*Policy*) qui lie les actions a (coups à jouer) en fonction de l'état s (configuration de jeu courante)

→Pas d'exploration systématique

→Modélisation locale du contexte $a=\phi(s)$

- Utilise des réseaux de neurones entraînés sur des parties jouées contre un humain et contre lui-même

Résumé

- Jeu à 2 joueurs à somme nulle

→Recherche adverse

→Prise en compte de la stratégie adverse

→Algorithme MiniMax

→Elagage α–β

- Autres contextes pour la prise en compte de l'aléa
- Alternative actuelles par apprentissage