Chapitre 8 : Jeux à deux joueurs

04 janvier

1 Jeux à deux joueurs

Dans ce chapitre, on s'intéresse uniquement à des jeux de réflexion à deux joueurs, à information totale, sans aléatoire, avec alternance des joueurs. Il existe de nombreuses variantes du modèle de jeux.
L'objet de ce chapitre est de concevoir des algorithmes permettant de jouer raisonnablement bien, voire de manière optimale à un jeu donné.

Définition 1 *Un **jeu à deux joueurs** est un quadruplet $J = (G, s_0, T_1, T_2)$ où :*

- *$G = (S, A)$ est un graphe orienté et biparti, c'est-à-dire tel que $S = S_1 \sqcup S_2$ et $A \subset S_1 \times S_2 \cup S_2 \times S_1$;*
- *$s_0 \in S$ est un sommet appelé **état initial** ;*
- *T_1 (resp. T_2) est une partie de S contenant des sommets qui sont des puits (degré sortant nul) appelés **états gagnants** pour le joueur 1 (resp. le joueur 2).*

*Les sommets de S_1 sont appelés **états du joueur 1** (c'est-à-dire les configurations du jeu où c'est au joueur 1 de jouer) et ceux de S_2 **états du joueur 2**.*

*Les puits de G sont appelés **états finaux** (ou terminaux). Les états finaux qui ne sont pas des éléments de T_1 ou T_2 sont appelés **états nuls**.*

Une arête (s, t) du graphe signifie qu'après un coup du joueur dont s est un état, le jeu passe de la configuration s à la configuration t.

Définition 2 *Une **partie** est un chemin depuis l'état initial vers un état final.*

*Une **stratégie** pour le joueur i est une fonction $f : S_i \to S_{3-i}$ telle que pour tout $s \in S_i$ non final, $(s, f(s)) \in A$, à savoir une fonction qui, pour chaque état du joueur i, indique le coup à jouer, c'est-à-dire le sommet suivant dans la partie. On parle ici de stratégie sans mémoire car la fonction de stratégie ne dépend que de la configuration courante.*

*On dit qu'une stratégie f pour le joueur i est **gagnante** pour i depuis un sommet s si et seulement si quelle que soit la stratégie g pour le joueur $3 - i$, il existe un entier $n \in \mathbb{N}$ tel que*

- *Si $s \in S_i$: $f \circ (g \circ f)^n(s)$ ou $(g \circ f)^n(s)$ est bien défini et est un état final gagnant pour le joueur i.*
- *Sinon, si $s \in S_{3-i}$: $(f \circ g)^n(s)$ ou $g \circ (f \circ g)^n(s)$ est bien défini et est un état final gagnant pour le joueur i.*

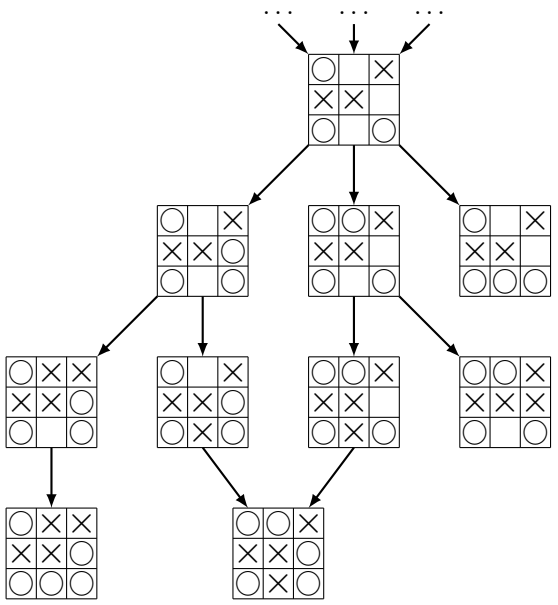
*Un sommet $s \in S$ est appelé **position gagnante** pour le joueur i s'il existe une stratégie gagnante pour i dans le jeu G depuis le sommet s. L'ensemble des positions gagnantes pour i est appelé **attracteur** pour i.*

Exemple 1.1

> Le jeu du tic-tac-toe (ou morpion) est un jeux à deux joueurs. Il consiste en une grille carrée de 9 cases (3×3). Deux joueurs s'affrontent en jouant tour à tour. À son tour, un joueur doit remplir une case vide avec son symbole (cercle ou croix). Le premier à aligner 3 de ses symboles, en ligne, colonne ou diagonale, remporte la partie. La partie est un match nul si toutes les cases sont remplies sans

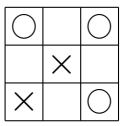
vainqueur. Le schéma suivant contient deux états gagnants pour le joueur O, un état gagnant pour le joueur X, donc perdant pour le joueur O, et un match nul.

Le schéma suivant illustre une partie du graphe des configurations du jeu.



Remarque 1.2

Une position gagnante pour i n'est pas nécessairement un sommet de S_i. Par exemple, la configuration suivante :



est une position gagnante pour le joueur O, même si c'est au joueur X de jouer. En effet, quel que soit le coup de X, le joueur O pourra aligner trois cercles au coup suivant.

2 Détermination d'une stratégie optimale à l'aide des attracteurs.

2.1 Calcul des attracteurs

Le calcul des attracteurs se fait à partir du constat suivant :

- si $s \in S_i$ et qu'il existe $(s,t) \in A$ tel que t est une position gagnante pour i, alors s est une position gagnante pour i (le joueur i peut choisir un coup qui l'emmène vers la victoire) ;
- si $s \in S_{3-i}$ et que pour tout $(s,t) \in A$, t est une position gagnante pour i, alors s est une position gagnante pour i (quel que soit le coup du joueur $3-i$, il ne peut pas empêcher i de gagner).

Définition 3 *On définit par induction la suite d'ensembles $A_n(i)$, pour $i \in \{1,2\}$ et $n \in \mathbb{N}$ par :*

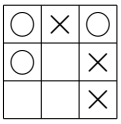
- $A_0(i) = T_i$ *(l'ensemble des états finaux gagnants pour i) ;*
- *pour $n \in \mathbb{N}$,* $A_{n+1}(i) = A_n(i) \cup \{s \in S_i \mid \exists t \in A_n(i), (s,t) \in A\}$
  $\cup \{s \in S_{3-i} \setminus \mathit{Puits} \mid \forall (s,t) \in A, t \in A_n(i)\}$

Proposition 2.1

$A_n(i)$ converge vers un ensemble $A(i)$ qui est l'attracteur du joueur i.

Remarque 2.2

On remarque que certains états ne sont dans aucun des attracteurs comme par exemple l'état suivant où on suppose que c'est au joueur $\times$ de jouer :



2.2 Stratégie optimale

On peut construire une stratégie **optimale** f pour le joueur i en distinguant les cas pour un sommet $s \in S_i$:

- si $s \in A(i)$, alors il existe $t \in A(i)$ tel que $(s,t) \in A$. On pose $f(s) = t$ avec t un tel sommet ;
- si $s \in A(3-i)$, alors tous les sommets t tels que $(s,t) \in A$ vérifient $t \in A(3-i)$. On pose $f(s) = t$ avec un tel t choisi arbitrairement ;
- si $s \in S \setminus (A(i) \cup A(3-i))$, alors les sommets t tels que $(s,t) \in A$ sont soit des sommets de $A(3-i)$, soit des sommets de $S \setminus (A(i) \cup A(3-i))$. On pose $f(s) = t$ avec un t choisi arbitrairement dans $S \setminus (A(i) \cup A(3-i))$.

Proposition 2.3

Dans le cas où toutes les parties se terminent on a le résultat suivant : une stratégie optimale pour i est gagnante pour i depuis s si $s \in A(i)$.

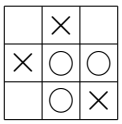
Remarque 2.4

S'il existe des cycles dans le graphe mais qu'un sommet s de S_i est dans $A(i)$ alors on peut tout de même trouver un t pour obtenir une stratégie qui permettra de gagner à partir de la configuration s. Contrairement à ce qui est décrit précédemment, on ne peut pas choisir n'importe quel $t \in A(i)$ tel $(s,t) \in A$ cependant il existe n tel que $s \in A_{n+1}(i)$ et $s \notin A_n(i)$ on sait alors qu'il existe $t \in A_n(i)$ tel que $(s,t) \in A$ par construction de $A(i)$ et c'est un tel t qu'il faut considérer et non pas un $t \in A(i)$ quelconque.

On peut illustrer cette remarque sur le jeu suivant :

Exercice 1

Déterminer si l'état suivant est une position gagnante pour O, pour X ou une position nulle, selon que c'est à O ou X de jouer.

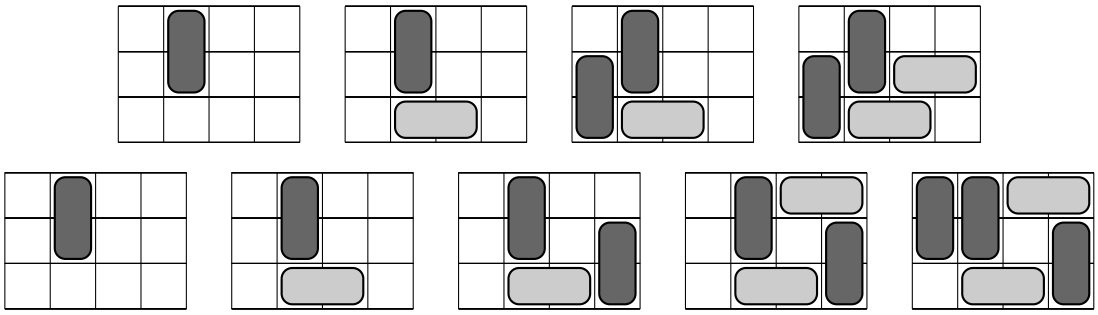


Remarque 2.5

Certains jeux ne présentent pas la possibilité de faire un match nul. Dans ce cas, chaque état du graphe des configurations est une position gagnante pour l'un des deux joueurs.

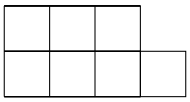
Exemple 2.6

Le jeu du domineering se joue sur un plateau avec des cases carrées contigües. À tour de rôle, chaque joueur place un domino sur le plateau ; le joueur 1 place ses dominos verticalement et le joueur 2 place ses dominos horizontalement. Le premier joueur qui ne peut plus jouer perd la partie. Les figures suivantes représentent respectivement une partie gagnée par le joueur 2 (gris clair) et par le joueur 1 (gris foncé).



Exercice 2

En dessinant les graphes des configurations, déterminer pour quel joueur le plateau suivant est gagnant, selon le joueur qui commence.



3 Algorithme Min-Max et heuristique

3.1 Min-Max

On peut formaliser la construction d'une stratégie optimale de la manière suivante : on attribue un **score** à chaque état du graphe des configurations :

- 1 pour une position gagnante pour le joueur 1 ;
- -1 pour une position gagnante pour le joueur 2 ;
- 0 pour une position nulle.

Dès lors, déterminer si une position est gagnante pour le joueur 1 ou non revient à calculer le score de cet état. Cela peut se faire récursivement de la manière suivante :

- si s est final, renvoyer 1, -1 ou 0 selon que l'état soit gagnant pour 1, pour 2 ou nul ;
- sinon, si $s \in S_1$, renvoyer le **maximum** des scores des voisins de s ;
- sinon, si $s \in S_2$, renvoyer le **minimum** des scores des voisins de s.

Pour rendre efficace un tel calcul, il est important de **mémoïser** les scores déjà calculés pour éviter d'avoir à les calculer à nouveau.

Remarque 3.1

S'il n'est pas implémenté correctement, l'algorithme décrit précédemment peut ne pas terminer si le graphe contient des cycles. Cela ne peut pas se produire dans un jeu comme le tic-tac-toe, mais est possible pour de nombreux jeux, comme les échecs par exemple.

3.2 Heuristique

Lorsque le graphe des configurations est de taille raisonnable, il est possible de calculer l'attracteur de chacun des joueurs et donc de déterminer des stratégies optimales. Cependant, dans la majorité des cas, les possibilités de jeu sont trop grandes pour permettre une exploration exhaustive.

Exemple 3.2

Le jeu du tic-tac-toe contient 5478 états. Le jeu du domineering contient 4632 états pour une grille 4×4, environ un million pour une grille 5×5, et environ 40 billiards (4×10^{16}) pour le jeu classique 8×8.

Lorsque c'est le cas, on peut attribuer à chaque état du jeu une valeur représentant la situation. Cette valeur, entière ou réelle, indique quel joueur est **a priori** en meilleure position pour gagner. Les rôles étant symétriques, on choisira généralement d'attribuer une valeur positive à un état favorable au joueur 1 et une valeur négative à un état favorable au joueur 2.

Un telle valeur sera calculée par une **heuristique**, et comme pour l'heuristique étudiée pour l'algorithme A^* :

- l'heuristique donne une estimation de la situation ;
- l'heuristique doit être facile à calculer.

Exemple 3.3

Dans le jeu du domineering, on peut choisir comme heuristique la fonction qui à un état du jeu associe :

- le nombre de coups possibles pour le joueur 1 moins le nombre de coups possibles pour le joueur 2 si la partie n'est pas terminée ;
- $+\infty$ (ou une valeur très grande) si l'état est gagnant pour le joueur 1 et $-\infty$ s'il est gagnant pour le joueur 2.

Dès lors, pour déterminer le coup à jouer pour le joueur 1 à partir d'un état s, on procède de la manière suivante :

— on choisit une profondeur d'exploration p ;

— on considère le sous-graphe des sommets à distance au plus p de s ;

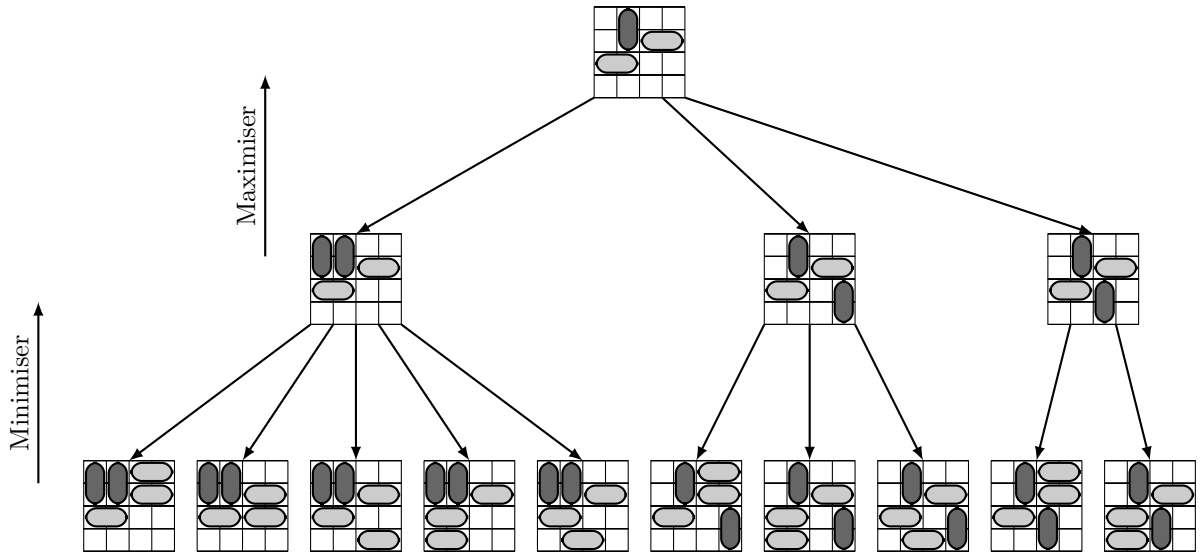
— on attribue un **score** à chaque sommet t de ce sous-graphe selon le principe du Min-Max :

- si t est un puit de ce sous-graphe, le score de t est donné par l'heuristique ;
- sinon, si $t \in S_1$, le score de t est le **maximum** des scores des voisins de t ;
- sinon, si $t \in S_2$, le score de t le **minimum** des scores des voisins de t.

— on choisit comme coup à jouer le voisin de s qui maximise le score.

Exemple 3.4

Voici une représentation d'un sous-graphe des configurations à profondeur 2 depuis un état donné :



Exercice 3

- Déterminer le score de chaque sommet du sous-graphe précédent en utilisant l'heuristique décrite dans l'exemple 3.3.
- En déduire le coup que jouerait le joueur 1 dans cette situation selon l'algorithme du Min-Max avec cette heuristique.
- Pour chaque sommet du sous-graphe précédent, déterminer s'il s'agit d'une position gagnante pour le joueur 1 ou le joueur 2. Justifier.

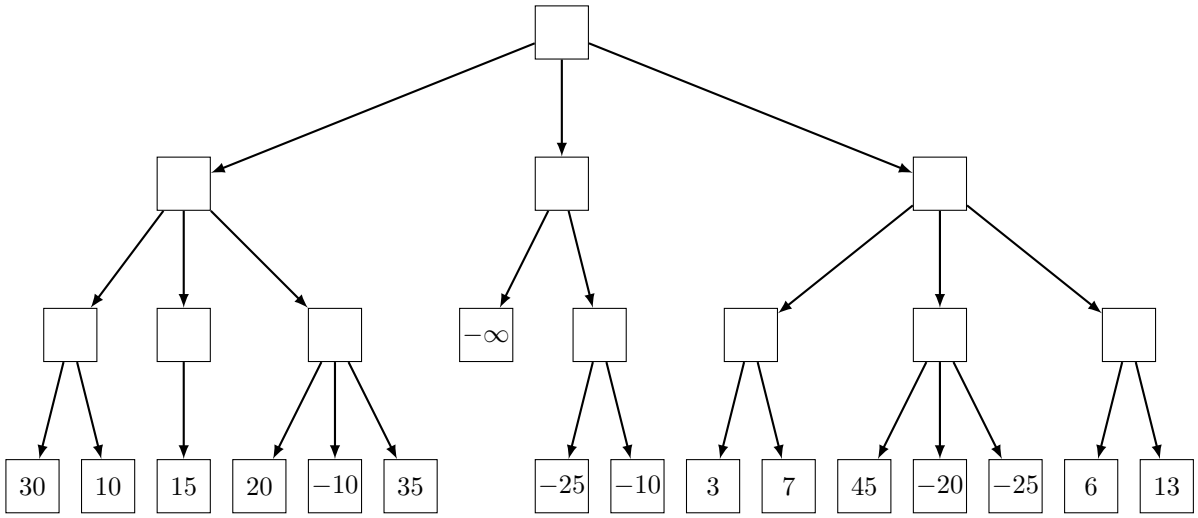
Remarque 3.5

Bien entendu, comme une telle exploration n'est pas exhaustive, rien ne garantit qu'elle prédira correctement une victoire. La qualité d'un tel algorithme dépend de la qualité de l'heuristique, ainsi que de la profondeur d'exploration.

Exercice 4

On considère un sous-graphe des configurations donné par l'arbre suivant. La valeur de l'heuristique pour chaque puit est indiquée dans les feuilles de l'arbre. Compléter l'arbre avec les scores de chaque nœud :

- en supposant que c'est au joueur 1 de jouer à la racine ;
- en supposant que c'est au joueur 2 de jouer à la racine.



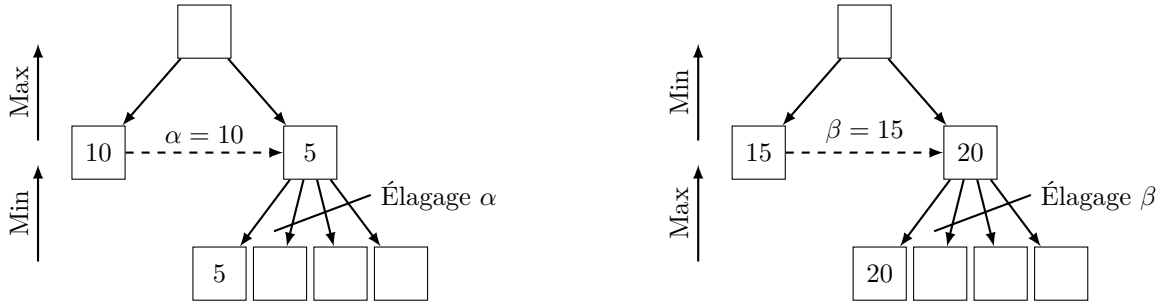
4 Élagage Alpha-Beta

Comme un algorithme de retour sur trace, l'algorithme du Min-Max explore en profondeur l'intégralité de l'arbre construit à partir d'un état donné. Cependant, dans certaines situations, il est possible de couper certaines branches de l'arbre et d'arriver à une conclusion sans tout explorer.

L'idée est qu'on peut garder en mémoire des bornes $[\alpha, \beta]$ indiquant la plage de valeurs dans laquelle se situe le score d'un nœud de l'arbre. Si on a la garantie que le score d'un fils de ce nœud sort de ces bornes, on peut arrêter complètement l'exploration de ce fils, et couper les branches restantes de ce fils.

Exemple 4.1

L'élagage α permet de couper des branches lorsqu'on cherche à calculer le score d'un nœud appartenant au joueur 1 (maximisation). L'élagage β permet de couper des branches lorsqu'on cherche à calculer le score d'un nœud appartenant au joueur 2 (minimisation).



Remarque 4.2

Les scores des nœuds intermédiaires peuvent différer lorsqu'on applique l'élagage Alpha-Beta par rapport aux scores obtenus avec uniquement l'algorithme du Min-Max. Ce n'est pas un problème, car l'élagage effectué garantit que le score de la racine sera le même.

Exercice 5

Représenter graphiquement l'arbre obtenu et les scores des différents nœuds restant après élagage Alpha-Beta dans l'arbre de l'exercice 4, en supposant que c'est au joueur 1 de jouer.