Théorie des jeux – Algorithme min_max

Problème

Nous avons vu comment faire jouer de manière parfaite un algorithme en utilisant la notion d'attracteur.
Cependant cette approche n'est pas envisageable pour des jeux plus complexes comme la plupart des jeux « classiques ». Par exemple, le nombre n des sommets du graphe de jeu est de l'ordre de 10^{32} pour le jeu de dames, entre 10^{43} pour les échecs et 10^{100} pour le jeu de Go.

Nous allons donc étudier une méthode permettant à un programme de jouer à un jeu, sans exploration complète du graphe, et donc de manière non parfaite.

Algorithme min-max

Jeu à somme nulle

On représente un jeu à somme nulle comme une structure $\mathcal{G} = (G, S_1, S_2, g)$ avec $G = (S, A)$, où g est une fonction qui à toute partie π fait correspondre le gain pour le joueur J_1 à l'issue de π ; le gain pour le joueur J_2 est alors $-\text{gain}(\pi)$.

Théorème du minimax et algorithme du min-max

A chaque sommet s est associée une valeur $v(s)$ calculée de la façon suivante à partir de la fonction de gain g.
$A(s)$ désigne l'ensemble des successeurs du sommet s.

$$\forall s \in S \qquad v(s) = \begin{cases} g(s) & \text{si } s \text{ est un sommet terminal} \\ \max\limits_{s' \in A(s)} v(s') & \text{si } s \text{ n'est pas terminal et } s \in S_1 \\ \min\limits_{s' \in A(s)} v(s') & \text{si } s \text{ n'est pas terminal et } s \in S_2 \end{cases}$$

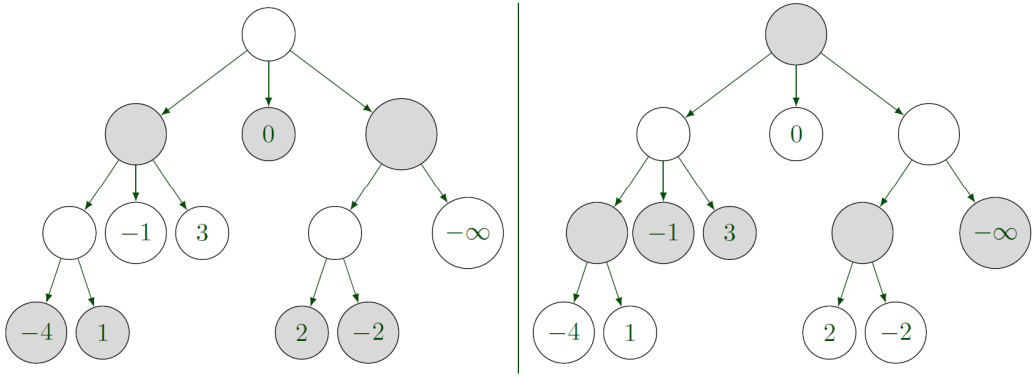
Le principe de l'algorithme du min-max est le suivant : étant donnée une position s, si s est contrôlée par le joueur J_1 celui-ci cherche le coup qui lui permet de maximiser son gain ; si s est contrôlée par le joueur J_2 celui-ci cherche le coup qui lui permet de minimiser le gain de l'adversaire ou, de manière équivalente, de maximiser son gain (le gain de J_2 est une valeur négative du point de vue de J_1).

Remarque : un problème difficile réside dans le choix de la fonction de gain g.

Une façon d'implémenter l'algorithme du min-max qui consiste à déterminer les valeurs des sommets du graphe de jeu à partir des sommets terminaux (cf. exemple 1).
L'algorithme de calcul des attracteurs dans les jeux d'accessibilité procède de façon analogue, l'algorithme min-max pose donc, sous cette forme, les mêmes problèmes.

Exemple 1 – Algorithme min-max « bottom-up »

Déterminer les valeurs des sommets du graphe de jeu ci-dessous ; les valeurs indiquées dans les sommets terminaux représentent le gain pour le joueur J_1.
Dans le premier exemple (à gauche), le sommet en haut est contrôlé par J_1 ; dans le deuxième, le sommet en haut est contrôlé par J_2.



Algorithme du min-max avec heuristique

Une autre approche de l'algorithme du min-max est utilisée pour traiter des jeux plus complexes, sans exploration complète du graphe et donc de manière imparfaite.
Pour cela, on se donne une ***fonction d'évaluation heuristique*** (qui remplace la fonction g) qui est susceptible d'attribuer une estimation numérique de la valeur de chaque position du jeu.
Formellement, il s'agit d'une fonction h définie sur l'ensemble des sommets du graphe de jeu, à valeurs dans $[-\infty, +\infty]$.
Une telle fonction est souvent construite de façon expérimentale ou par apprentissage automatique.

Dans le jeu d'échecs, une heuristique très simpliste s'obtient en associant à chaque position la différence du nombre de pièces du joueur J_1 et de ceux du joueur J_2. Une amélioration consiste à affecter les différentes pièces d'un coefficient (barème standard : pion 1 ; cavalier et fou 3 ; tour 5 ; dame 9). On peut aussi tenir compte de la position des pièces sur l'échiquier.

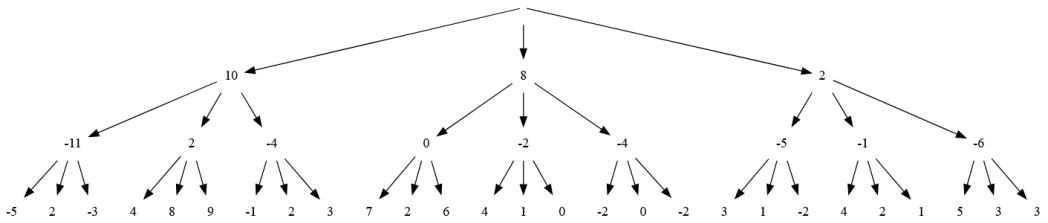
Dans le cas d'un sommet terminal, l'heuristique renvoie la vraie valeur du gain pour J_1 (ou par exemple $\pm\infty$ pour une position de victoire dans un jeu d'accessibilité, 0 pour un match nul).

L'algorithme du min-max avec une heuristique consiste à fixer une profondeur maximale d'exploration du graphe de jeu à partir de la position donnée et à procéder de façon récursive : à chaque appel récursif la profondeur est décrémentée de 1, et quand elle est égale à 0 (ou quand le sommet est terminal) la fonction renvoie la valeur fournie par l'heuristique.

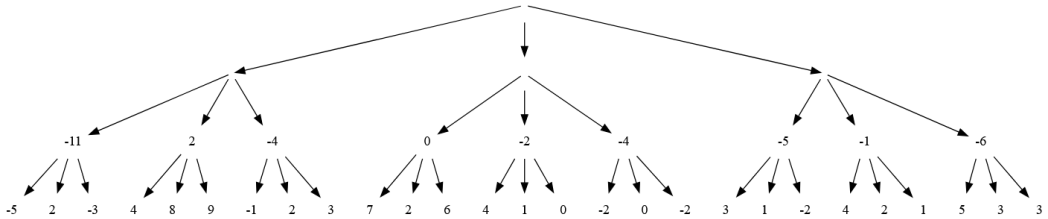
Exemple 2 – Algorithme min-max avec heuristique et profondeur fixée

Calculer la valeur du sommet en haut, contrôlé par J_1, donnée par l'algorithme min-max appliqué au graphe ci-dessous, avec les profondeurs spécifiées.
Les valeurs affichées dans chaque sommet sont celles fournies par la fonction d'évaluation heuristique.

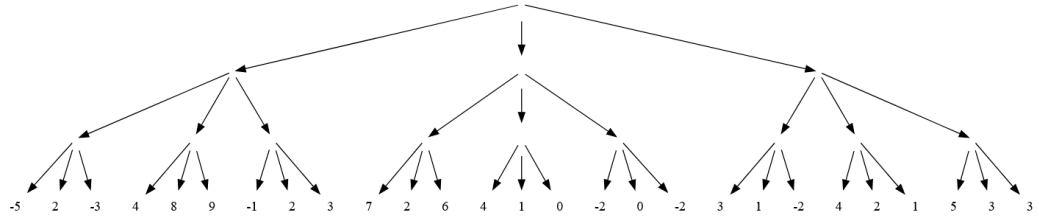
a) Profondeur = 1



b) Profondeur = 2



c) Profondeur = 3



Algorithme

Dans l'implémentation, les joueurs sont notés 0 et 1 (on associe le joueur noté J_0 à la valeur 0 et J_1 à la valeur 1), l'intérêt de cette notation réside dans la constatation suivante : si *j* est l'un quelconque des deux joueurs, alors 1-*j* est son adversaire.

La fonction récursive *minimax*(*pos*, *joueur*, *succ*, *h*, *p*) prend comme arguments :

- un objet *pos* qui représente la position courante (le sommet courant) ;
- le *joueur* (0 ou 1) qui la contrôle ;
- une fonction *succ* telle que *succ*(*pos*, *joueur*) renvoie la liste des successeurs de la position *pos* contrôlée par *joueur* ;
- la fonction d'évaluation heuristique *h* ; on suppose que *h*(*pos*, *joueur*) renvoie la valeur de l'heuristique en *pos* ;
- un entier *p* qui spécifie la profondeur d'exploration.

La fonction calcule la valeur de la position *pos* pour le joueur J_0 en utilisant l'algorithme du min-max.

```
def minimax(pos, joueur, succ, h, p):
    """
    Paramètres :
        pos : position (noeud ou sommet)
        joueur : 0 ou 1
        succ : succ(pos) renvoie le successeur du sommet pos
        h : heuristique h(pos) renvoie une valeur si elle existe
        p : entier (profondeur)
    Renvoie :
        valeur associée à pos passé en paramètre pour le joueur 0
    """
    # Critère d'arrêt : profondeur max atteinte (p décroît au cours des appels)
    if p == 0:
        return h(pos)    # h(pos) doit être définie à cette profondeur
    # Appels récursifs sur tous les successeurs du sommet pos
    else:
        # Liste des valeurs obtenues récursivement
        vsucc = [minimax(s, 1-joueur, succ, h, p-1) for s in succ(pos)]
                # Inversion du joueur et diminution de p
        if len(vsucc) == 0:     # Aucun successeur
            return h(pos)       # h(pos) doit être définie à cette profondeur
        elif joueur == 0:
            return max(vsucc)   # J0 maximise son gain
        else:
            return min(vsucc)   # J1 maximise son gain (en valeur absolue)
```

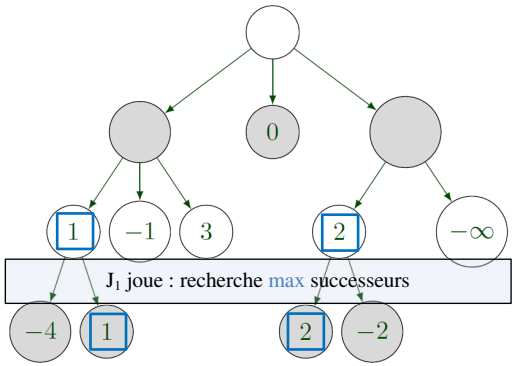
Un notebook « Théorie des jeux – Algorithme min-max » permettant de tester l'algorithme nim-max et de tracer des graphes est disponible, l'implémentation est fournie.

Réponses

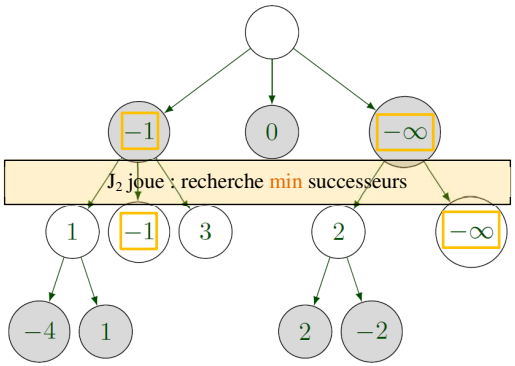
Exemple 1 – Algorithme min-max « bottom-up »

Déterminer les valeurs des sommets du graphe de jeu ci-dessous ; les valeurs indiquées dans les sommets terminaux représentent le gain pour le joueur J_1.
Dans le premier exemple, le sommet en haut est contrôlé par J_1.

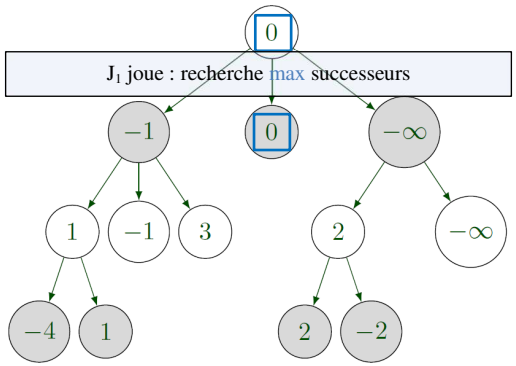
Principe : on part des sommets terminaux et on remonte dans le graphe en appliquant l'algorithme. Raisonnement détaillé pour le 1er cas ci-dessous.



Etape 1

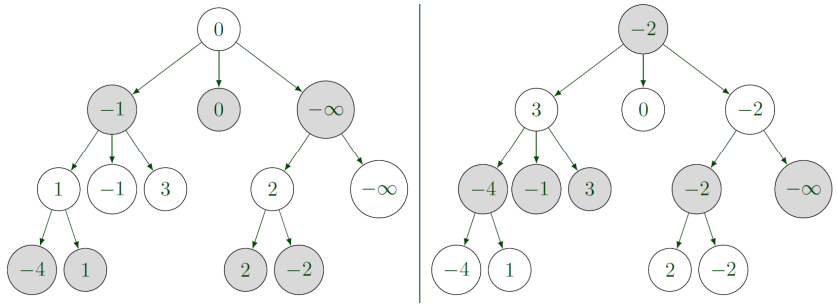


Etape 2



Etape 3

Solutions dans les deux cas proposés



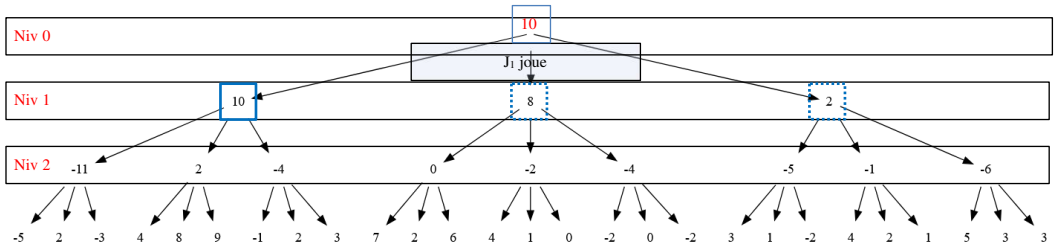
J_1 maximise son gain (0 mais tout autre choix est pire) et J_2 maximise également son gain (-2).

Exemple 2 – Algorithme min-max « top-down »

Calculer la valeur du sommet en haut, contrôlé par J_1, donnée par l'algorithme min-max appliqué au graphe ci-dessous, avec les profondeurs spécifiées.
Les valeurs affichées dans chaque sommet sont celles fournies par la fonction d'évaluation heuristique.

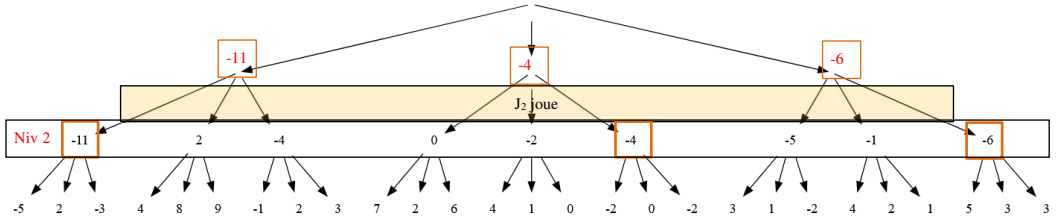
d) Profondeur = 1

J_1 joue puisqu'il contrôle le sommet et choisit donc la valeur max parmi les successeurs immédiats de sa position (niveau 0) dans le niveau 1.

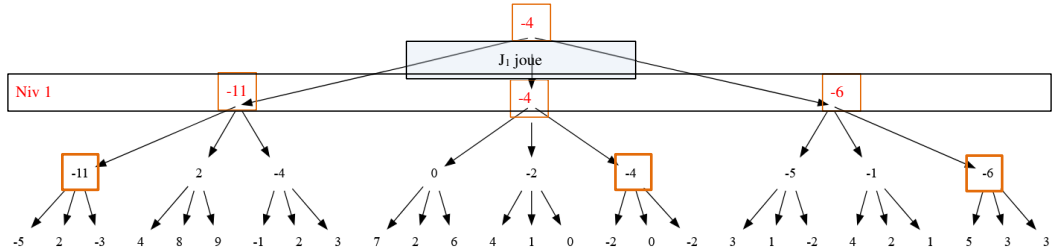


e) Profondeur = 2

Du niveau 1 au niveau 2, c'est J_2 qui joue et choisit la valeur min parmi les successeurs du niveau 1 donc dans le niveau 2 : -11.



Puis J_1 joue et choisit donc la valeur max parmi les successeurs de sa position (niveau 0) dans le niveau 1 : -4.



f) Profondeur = 3