Introduction à la théorie des jeux

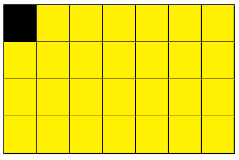
Introduction

Dans ce chapitre, on donne une introduction à l'étude de la théorie des jeux. Les jeux que l'on va étudier sont des jeux à *information complète*, ce qui exclut la plupart des jeux de cartes (chaque joueur n'a pas connaissance du jeu adverse et éventuellement du reste du paquet), et sans hasard. On ne s'intéresse qu'à des *jeux d'accessibilité* à deux joueurs, traditionnellement nommés Adam et Eve, qui peuvent être modélisés par des graphes orientés, jouer un coup consiste à suivre un arc.
Ces prénoms ont été choisis car on cherche si quel que soit le coup joué par Adam($\forall$, A à l'envers), il existe ($\exists$, E en miroir vertical) une stratégie gagnant pour Eve.
Dans un premier temps, on discute de jeux qui sont suffisamment simples pour pouvoir être résolus complètement : l'espace des états possibles n'est pas trop grand, et on pourra déterminer une stratégie gagnante. Dans un second temps, on aborde la notion de stratégie avec heuristique , qui peut s'appliquer aux jeux qu'il est impossible (à l'heure actuelle, au vu des contraintes matérielles) de résoudre complètement, comme le jeu d'échecs ou le jeu de go.

1 Jeux sur un graphe

1.1 Le jeu de Chomp

Le premier jeu auquel nous allons nous intéresser se joue à l'aide d'une tablette de chocolat rectangulaire dont le coin supérieur gauche est empoisonné : chaque joueur choisit à tour de rôle un carré, et le mange, ainsi que tous les morceaux situés à la droite et en dessous du carré choisi. Bien évidemment, le joueur qui n'a plus d'autre choix que de manger le carré empoisonné a perdu.



On dessinera ci-dessous un exemple de partie perdue par Adam, qui a commencé à jouer en premier.
Tablette de 3 lignes et 4 colonnes, carrés numérotés comme sur une matrice.
1er coup joué par Adam : il mange le carré 3,3.
Eve mange le carré 2,4.
Adam mange le carré 1,4
Eve mange le carré 2,2
Adam mange le carré 2,1
Eve mange le carré 1,2
Il ne reste que le carré 1,1....

Modélisation

A ce type de jeu est associé un graphe orienté (S, A) appelé *arène*. Chaque sommet de S représente une configuration du jeu (une *position*), l'une d'entre elles étant la position de départ. Une arête $a = (s_1, s_2) \in A$ reliant deux sommets indique la possibilité pour un joueur de passer de la position s_1 à la position s_2 en un coup.

Par exemple, l'arène du jeu de Chomp pour une tablette à 2 lignes et 3 colonnes est représentée ci-dessous.

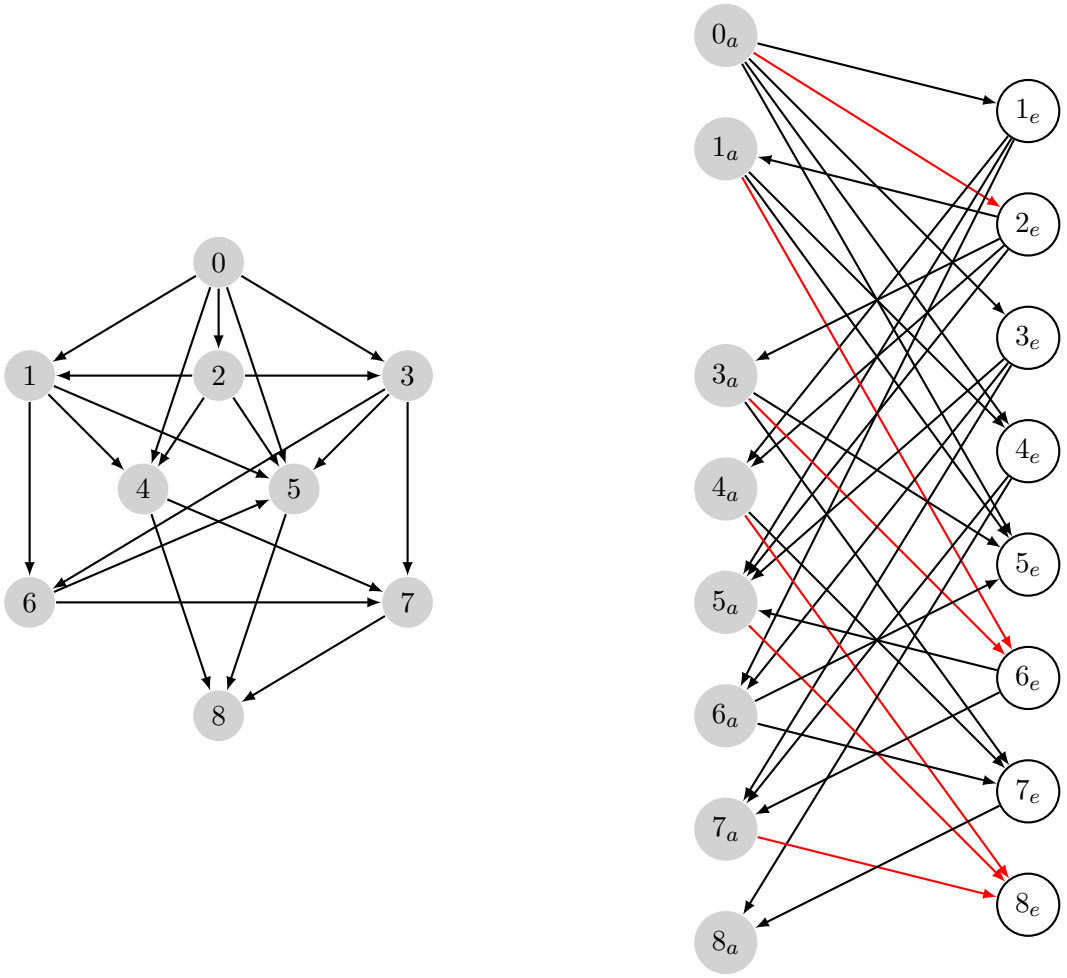
Pour visualiser une partie de Chomp, il suffit d'imaginer un jeton initialement posé sur la position de départ s_0. A tour de rôle, chaque joueur le déplace le long d'une arête issue de la position courante s et le pose sur un successeur de s. Une partie est donc un chemin d'origine s_0 dans l'arène.

Ce jeu fait partie des jeux d'*accessibilité* : le graphe associé ne comporte pas de cycle (ce qui assure que toute partie est finie), et il est déterminé par un ensemble de positions (les *cibles*) qui sont sans successeurs.

Ainsi dans le jeu de Chomp, le sommet du graphe associé à la tablette où il ne reste que le carré empoisonné est la seule cible du jeu, et l'atteindre signifie la fin de la partie (et la victoire pour celui qui l'atteint).

Une fois fixé le joueur qui commence (Adam par exemple) , on peut nommer les sommets du graphe (ici on les numérote de 0 à 8).

Et on peut, en doublant chacun des sommets qu'on indexera par le nom du joueur, créer un nouveau graphe dont les sommets seront partitionnés en deux sous-ensembles $S = S_a \cup S_e$ avec $S_a \cap S_e = \emptyset$, et S_i est l'ensemble des positions à partir desquelles le joueur i jouera. Un tel graphe est appelé biparti.



Graphe et graphe biparti associés au jeu de Chomp (2,3)

Stratégies gagnantes

Une *stratégie* pour Adam est une application $f : S'_a \subset S_a \to S_e$ tel que pour $s \in S'_a$, $f(s)$ est une arête du graphe biparti. Ainsi de manière informelle, une stratégie consiste à déterminer , pour chaque position de S'_a, le mouvement suivant à jouer. Une partie $s_0 \to s_1 \to \cdots \to s_n$ est dite *jouée suivant* f lorsque pour tout $k \in [\![0, n-1]\!]$, si $s_k \in S'_a$, alors $s_{k+1} = f(s_k)$. Une stratégie est dite *gagnante* pour Adam si toute partie jouée en suivant cette stratégie est gagnante pour Adam. On définit bien évidemment de manière analogue les stratégies et les stratégies gagnantes pour Eve.

Sur le graphe biparti de la figure précédente, les arcs correspondant à une stratégie gagnante pour Adam sont en rouge (à faire vous même sur le polycopié, photocopies en noir et blanc..) . Adam doit commencer par poser le jeton sur le sommet 2, puis :

- si Eve joue son coup suivant sur 1 ou 3, poursuivre sur 6. Eve ne peut alors que suivre sur 5 ou 7 , et Adam peut conclure en jouant sur 8.
- Si Eve joue son coup suivant sur 4 ou 5, alors poursuivre (et terminer) sur 8

.

Eve possède aussi une stratégie gagnante définie par $f(1_e) = f(3_e) = 6_a$ et $f(4_e) = f(5_e) = f(7_e) = 8_a$, mais comme elle ne joue pas en premier, il est nécessaire que Adam ne joue pas sur le sommet 2 au début de la partie pour qu'Eve puisse suivre cette stratégie.

Positions gagnantes

Une position s est dite *gagnante* lorsqu'il existe une stratégie gagnante pour une partie débutant au sommet s. Par exemple, pour le jeu de Chomp (2, 3), les positions 0, 1, 3, 4, 5, 7 sont gagnantes, les positions 2, 6 et 8 sont perdantes.

Théorème : Dans le jeu de Chomp (p, q) , il existe une stratégie gagnante pour le premier joueur dès que $(p, q) \neq (1, 1)$

Démonstration :
Si $p = 1$ ou $q = 1$, un coup suffit à Adam pour atteindre la position finale.
Supposons maintenant $p > 1$ et $q > 1$, et faisons choisir à Adam le carré en bas à droite. Supposons que ce coup soit perdant ; dans ce cas , Eve possède un coup gagnant. mais tout mouvement choisi par Eve à cette étape du jeu aurait pu être choisi par Adam pour entamer sa partie. Cela signifie que ou bien ce premier mouvement est gagnant, ou bien il en existe un autre qui soit gagnant. Dans tous les cas, Adam possède donc un coup gagnant. Ce type de stratégie est appelé un *vol de stratégie*, mais c'est une preuve non constructive, savoir qu'une stratégie est gagnante existe ne suffit pas à nous apprendre comment la trouver. C'est ce à quoi nous allons nous employer maintenant.

1.2 Détermination des positions gagnantes

Considérons un graphe orienté acyclique (S, A) associé à un jeu d'accessibilité.

Lemme Dans un graphe acyclique, il existe des sommets sans successeurs.

Démonstration : supposons qu'il n'existe pas de sommets sans successeurs. partant d'un sommet s_0 quelconque, on pourrait alors construire une suite (s_n) de sommets tels que pour tout $n \in \mathbb{N}$, $(s_n, s_{n+1}) \in A$. Mais S est de cardinal fini, donc il existe $p < q$ tel que $s_p = s_q$, ce qui signifie que $(s_p, s_{p+1}, \cdots, s_q)$ est un cycle, ce qui est absurde.

Définitions :

- Un sous-ensemble S' de sommets est dit *stable* si tout sommet de S' n'a aucun successeur dans S'.
- Un sous-ensemble S' de sommets est dit *absorbant* si tout sommet n'appartenant pas à S' possède au moins un successeur dans S'.
- Un sous-ensemble S' de sommets est un *noyau* s'il est à la fois stable et absorbant.

Par exemple dans le jeu de Chomp (2, 3), les sommets (2, 6, 8) forment un noyau du graphe.

Théorème : Tout graphe acyclique et orienté possède un unique noyau.

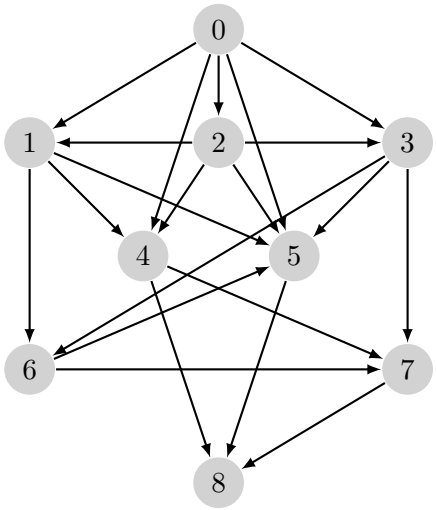
Démonstration : raisonnons par récurrence sur le nombre n de sommets.

- si $n = 1$, l'unique sommet est aussi l'unique noyau du graphe.
- si $n > 1$, supposons le résultat acquis jusqu'au rang $n - 1$. Par hypothèse, il existe au moins un sommet s sans successeur. Nécessairement, ce sommet doit appartenir à un éventuel noyau (car un noyau est absorbant). Considérons le graphe (S', A') obtenu en supprimant de (S, A) le sommet s ainsi que ses prédécesseurs.
  - si (S', A') est le graphe vide, c'est que $\{s\}$ est un noyau de (S, A), et c'est le seul (tous les autres sommets admettent s comme successeur)
  - dans le cas contraire, observons que (S', A') reste acyclique. par hypothèse de récurrence, il possède un unique noyau N'. Et dans ce cas, $N = N' \cup \{s\}$ est un noyau de (S, A)
    Pour justifier qu'il est unique, considérons un noyau quelconque N de (S, A) . Il doit contenir s , et $N \setminus \{s\}$ est un noyau de (S', A'), donc est égal à N'.

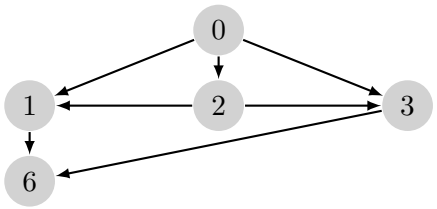
Cette preuve a le mérite d'être constructive : pour calculer le noyau de (S, A), il suffit de :

- chercher un sommet s de (S, A) sans successeur
- supprimer de (S, A) s et ses prédécesseurs
- recommencer tant qu'il reste des sommets

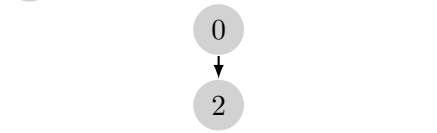
Voici par exemple comment on calcule le noyau du jeu de Chomp (2,3) :



8 n'a pas de successeur : il appartient au noyau,
on le supprime ainsi que tous ses prédécesseurs.



6 n'a pas de successeur : il appartient au noyau,
on le supprime ainsi que tous ses prédécesseurs.



2 n'a pas de successeur : il appartient au noyau,
on le supprime ainsi que tous ses prédécesseurs.

Le noyau est donc (2,6,8).

Remarque : dans le cas du jeu de Chomp, les éléments du noyau sont les positions perdantes, et les autres sommets les positions gagnantes. La stratégie gagnante consiste, pour chaque sommet qui n'est pas dans le noyau, à jouer un coup qui s'y ramène.

Algorithme du calcul du noyau

On considère un graphe acyclique (S, A). Il est représenté par un dictionnaire d dont les clés sont les sommets et les valeurs les successeurs des clefs (sous forme de liste) ;
Par exemple, le jeu de Chomp (2,3) est représenté en mémoire par le dictionnaire :

```
d={
  0:[1,2,3,4,5],
  1:[4,5,6],
  2:[1,3,4,5],
  3:[5,6,7],
  4:[7,8],
  5:[8],
  6:[5,7],
  7:[8],
  8:[]
  }
```

Exercice 1 :

1. Rédiger une fonction sansSuccesseurs(d) qui renvoie un sommet sans successeur
2. Rédiger une fonction predecesseurs(d,j) qui renvoie la liste de tous les prédécesseurs du sommet j.

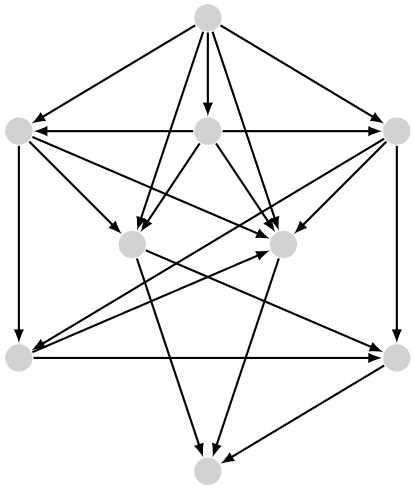
3. Rédiger une fonction supprimeSommets(d,j) qui supprime un sommet du graphe (et donc toutes les arêtes qui sont liées à ce sommet)
4. En déduire une fonction noyau(d) qui renvoie la liste des sommets constituant le noyau de (S, A)

L'algorithme précédent a une complexité en $O(n^3)$ si n est le nombre de sommets du graphe. sachant que le jeu de Chomp (p, q) possède $\binom{p+q}{p} - 1$ sommets, on conçoit que le calcul du sommet se révèle rapidement d'un coût rédhibitoire dès que p et q deviennent trop grands.
Pour $p = q$, on a $n \sim \binom{2p}{p} \sim \frac{4^p}{\sqrt{\pi p}}$, donc la complexité croit exponentiellement avec p.

Fonction de Sprague-Grundy

On considère un graphe orienté acyclique (S, A) associé à un jeu impartial dans lequel le perdant est celui qui ne peut plus jouer. On définit le *nimber n(s)* d'un sommet $s \in S$ de la façon suivante :
— si s est une position gagnante(une position sans successeur), $n(s) = 0$
— sinon, $n(s)$ est le plus petit entier positif ou nul n'apparaissant pas dans la liste des nimbers des successeurs de s
La fonction n ainsi définie s'appelle aussi la fonction de Sprague-Grundy.

Exercice 2
Reporter sur le graphe ci-dessous (associé au jeu de Chomp(2,3)) le nimber de chacun de ses sommets :



Théorème : le noyau du graphe correspond aux sommets s vérifiant $n(s) = 0$.

Démonstration :Posons $S' = \{s \in S | n(s) = 0\}$. Par définition, S est stable (un sommet de S' ne peut avoir parmi ses successeurs un autre sommet de nimber nul).
De plus, pour tout sommet $s \in S \setminus S'$, $n(s) > 0$, donc s possède un successeur de nimber nul, autrement dit un successeur appartenant à S', donc S' est absorbant.

Exercice 3
On représente un graphe par un dictionnaire d dont les clés sont les sommets et les valeurs les successeurs des clefs (sous forme de liste). le but de cet exercice est de construire un dictionnaire n dont les clefs sont les sommets et les valeurs les nimbers des sommets.

1. Rédiger une fonction $sansNimber(d, n)$ qui renvoie un sommet s ne possédant pas encore de nimber mais dont tous les successeurs en possèdent. cette fonction renverra None dans le cas où un tel sommet n'existe pas.
2. En déduire une fonction $nimber(d)$ qui renvoie le dictionnaire des nimbers des différents sommets composant ce graphe.

2 Algorithme min-max

Dans le cas d'un jeu deux joueurs plus complexe, le calcul du noyau n'est pas possible. L'algorithme que nous avons décrit a une complexité en $O(n^3)$, où n est le nombre de sommets du graphe, autrement dit le nombre de positions que l'on peut rencontrer lors d'une partie. Cependant cet entier n est souvent extrêmement grand : il est estimé de l'ordre de 10^{32} pour les dames, entre 10^{43} et 10^{50} pour les échecs et de l'ordre de 10^{100} pour le jeu de go. Il devient donc nécessaire de s'appuyer non plus sur une évaluation exacte de la position, mais sur une estimation de la valeur de la position atteinte.

2.1 Heuristique

Dans la suite de cette section, nous supposerons posséder une fonction h, qui à toute position légale p du jeu associe une valeur dans $\mathbb{R}$ de sorte que :

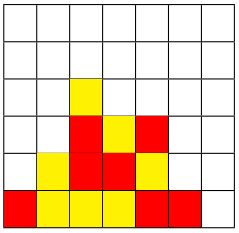
— plus $h(p)$ est grand, meilleure est la position pour Adam
— plus $h(p)$ est petit, meilleure est la position pour Eve

Une telle fonction est appelée une *heuristique*.

Jeu de Puissance 4

.

Pour illustrer cette section, nous allons prendre l'exemple du jeu de Puissance 4 : le but du jeu est d'aligner une suite de quatre pions de même couleur sur une grille comprenant six rangées et sept colonnes. tour à tour, les deux joueurs placent un pion dans la colonne de leur choix, le pion coulisse alors jusqu'à la position la plus basse possible dans la dite colonne, à la suite de quoi, c'est à l'adversaire de jouer.
Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) consécutif d'au moins quatre pions de sa couleur.
Si , alors que toutes les cases de la grille de jeu sont remplies, aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle.



Une position de puissance 4

Une heuristique simple consiste à attribuer à chaque case une valeur, par exemple le nombre d'alignements potentiels de quatre pions lorsqu'on place un pion à cet emplacement , puis à sommer les cases occupées (positivement pour les pions d'Adam, négativement pour ceux d'Eve).

3	4	5	7	5	4	3
4	6	8	10	8	6	4
5	8	11	13	11	8	5
5	8	11	13	11	8	5
4	6	8	10	8	6	4
3	4	5	7	5	4	3

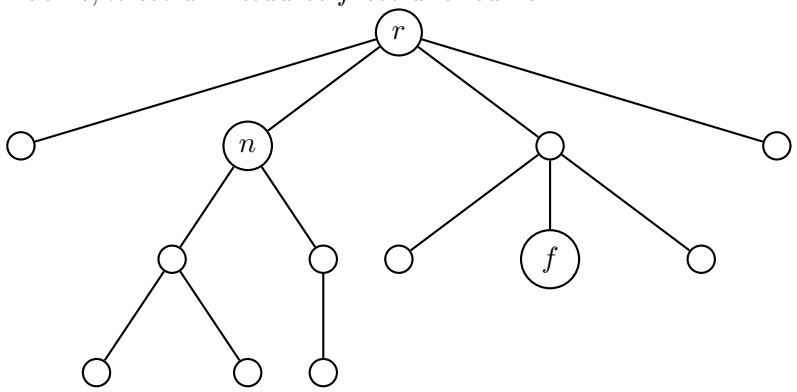
Les valeurs de chaque case pour l'heuristique utilisée

Par exemple , si on convient que les pions jaunes sont ceux d'Adam, la valeur de l'heuristique de la position présentée ci dessus est $4+5+7+6+8+13+11-3-5-4-8-10-11-11=2$
Evidemment , l'heuristique sera égale à $+\infty$ pour une position gagnante pour Adam et à $-\infty$ pour une position gagnante pour Eve.

2.2 Représentation par un arbre

Un jeu comme les jeux évoqués peut se représenter sous forme arborescente. Formellement, un arbre est un graphe connexe acyclique, dont on choisit un sommet particulier (la racine) , qui oriente l'arbre.
En informatique, les arbres poussent du haut vers le bas : la racine est donc située en haut.
La profondeur d'un nœud est sa distance à la racine.
Pour la représentation graphique, tous les nœuds situés à une même profondeur sont représentés sur une même ligne horizontale.
Il existe un unique chemin simple de la racine r à un nœud n quelconque du graphe. Le long de ce chemin, il y a une relation de parenté entre nœuds successifs : si p précède q, alors p est le *père* de q et q est un *fils* de p.
Un nœud sans fils est appelé une *feuille*, sinon c'est un nœud interne.

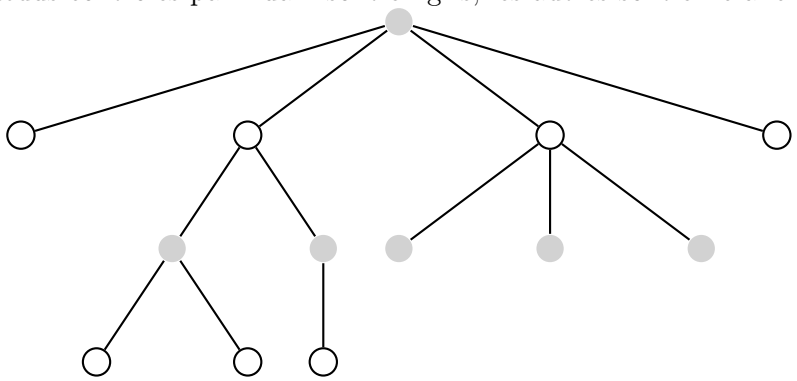
Sur la figure suivante, r est la racine, n est un nœud et f est une feuille.



Lorsqu'on dessine un arbre pour un jeu Min-Max, les nœuds représentent des positions du jeu. Puisque les joueurs jouent alternativement, les nœuds d'un même niveau sont alternativement contrôlés par un même joueur. Par symétrie, on peut supposer que :

- la racine est contrôlée par le joueur Adam (ou Max)
- les nœuds à profondeur 1 sont contrôlés par le joueur Eve (ou Min)
- les nœuds à profondeur 2 sont contrôlés par le joueur Adam (ou Max)
- etc...

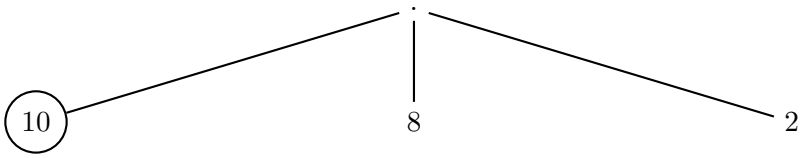
Dans la figure suivante, les nœuds contrôlés par Adam sont en gris, les autres sont en blanc.



2.3 Min-Max

Au moment où l'un des deux joueurs doit déposer un pion, plusieurs possibilités s'offrent à lui (entre une et sept pour le jeu du Puissance 4, car il y a 7 colonnes, dont certaines peuvent être déjà remplies).
Une solution simple pour choisir le coup à jouer consiste à calculer l'heuristique correspondant à chacune des configurations atteignables et à jouer celle d'heuristique maximale si c'est Adam qui joue, ou minimale si c'est Eve qui joue.

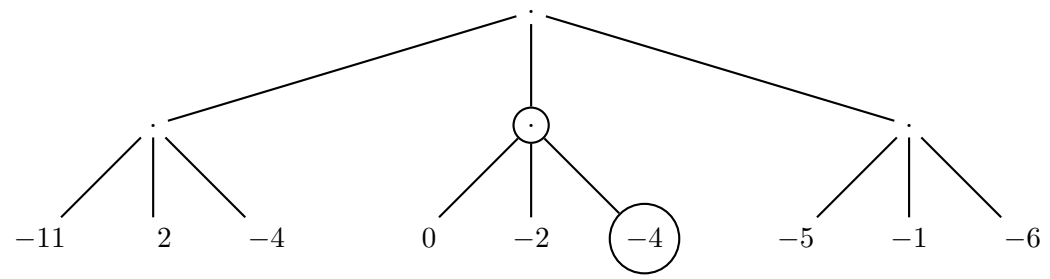
Première figure : c'est à Adam de jouer, trois coup sont possibles.



Adam choisira le coup le plus à gauche, correspondant à une évaluation égale à 10, qui est le maximum sur les trois coups possibles.

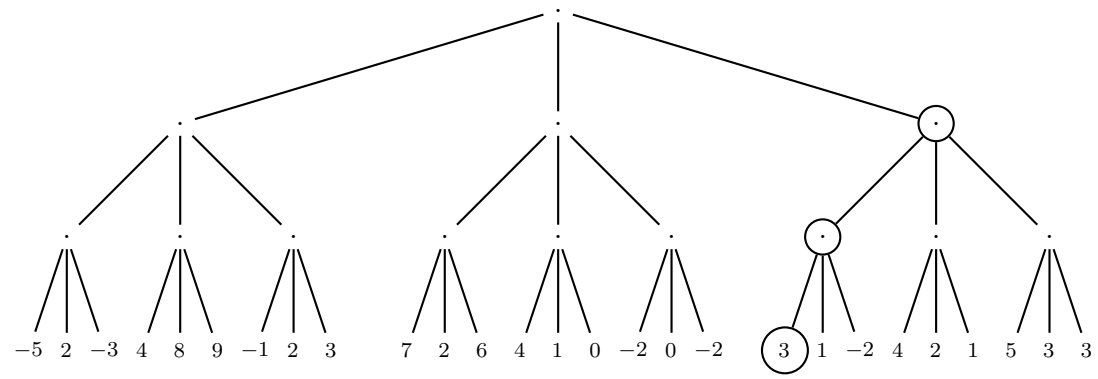
Mais Adam peut aussi tenir compte du coup que va jouer Eve ensuite, et donc calculer l'heuristique de chacune des positions que Eve pourra atteindre. Si on observe la figure suivante, on constate qu'il vaut mieux pour Adam jouer le coup central, en partant du principe qu'Eve joue au mieux (donc qu'elle cherche à minimiser l'heuristique de la position qu'elle va choisir).

Deuxième figure : c'est à Adam de jouer, en tenant compte du coup suivant de Eve.

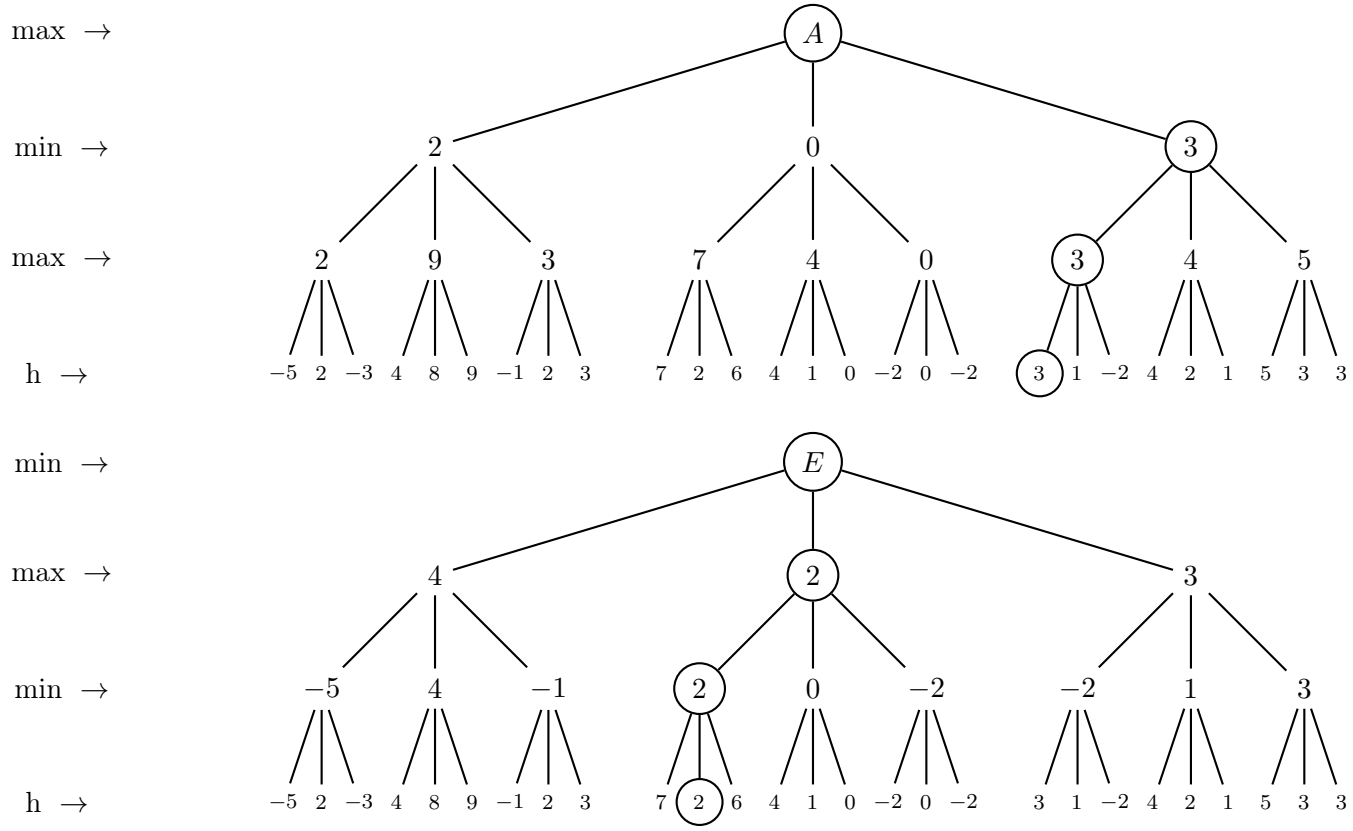


Ici on voit qu'il vaut mieux pour Adam choisir le coup central.

Bien évidemment, on peut réitérer ce raisonnement, et tenir compte du coup suivant joué cette fois par Adam. La figure suivante montre qu'en tenant compte des deux coups suivants, Adam a en fait intérêt à jouer le coup le plus à droite.



On peut répéter ce raisonnement, mais le nombre de configurations ayant tendance à croître exponentiellement , il est nécessaire de limiter la profondeur de la recherche.



Cette figure est le résultat de l'algorithme min-max à une profondeur 2, premier arbre si c'est à Adam de jouer, deuxième arbre si c'est à Eve de jouer.

Nous allons écrire deux fonctions :

- maximin(p, n) (destinée à Adam) va chercher à maximiser l'heuristique après n coups en partant de la position p, en supposant que son adversaire joue au mieux.
- minimax(p, n) (destinée à Eve) va chercher à minimiser l'heuristique après n coups en partant de la position p, en supposant que son adversaire joue au mieux.

Ces deux fonctions sont mutuellement récursives : pour calculer maximin(p, n), on calcule pour chaque position $p_1, p_2, \cdots, p_k$ atteignable en un coup à partir de p la valeur de l'heuristique des positions minimax$(p_i, n-1)$ avant de choisir la position conduisant à la valeur maximale.
De manière symétrique, pour calculer minimax(p, n), on calcule pour chaque position $p_1, p_2, \cdots, p_k$ atteignable en un coup à partir de p la valeur de l'heuristique des positions maximin$(p_i, n-1)$ avant de choisir la position conduisant à la valeur minimale.
On suppose que la fonction h d'argument p une position de jeu et renvoie la valeur de son heuristique a été définie, ainsi que la fonction successeurs d'argument p, qui renvoie la liste des positions atteignables à partir de la position p.

```
def minimax(p,n):
    if n==0 or successeurs(p)==[]:
        return h(p)
    mini=np.inf
    for pk in successeurs(p):
        s=maximin(pk,n-1)
        if s<mini:
            mini=s
    return mini
```

```
def maximin(p,n):
    if n==0 or successeurs(p)==[]:
        return h(p)
    maxi=-np.inf
    for pk in successeurs(p):
        s=minimax(pk,n-1)
        if s>maxi:
            maxi=s
    return maxi
```

Exercice Calculer la valeur de l'heuristique de la position associée à l'arbre ci-dessous , dans le cas où c'est le joueur qui cherche à maximiser l'heuristique qui doit jouer.