Introduction à la théorie des jeux

Introduction

Nous nous intéressons dans ce cours à la modélisation de certains jeux à deux joueurs (jeu de Nim, tic-tac-toe, jeu d'échecs ...). L'objectif pour chacun de ces jeux est de déterminer (si possible) une stratégie gagnante, ou du moins optimale selon certains critères. Nous nous appuierons pour cela sur la représentation de ces jeux par des graphes.

1 Modèle général

Afin de dégager les notions générales, nous nous appuyons sur l'exemple du jeux de chomp. Le principe de ce jeu est le suivant : on dispose d'une tablette de chocolat de taille (n, p). Le carré en haut à gauche est empoisonné. Chacun des deux joueurs, à son tour, casse la tablette suivant une ligne verticale ou horizontale et mange la partie de la tablette qu'il a détachée de la partie principale (celle qui contient le carré empoisonné). Le joueur ne disposant plus que du carré empoisonné lorsque c'est son tour de jouer a perdu (et l'autre a gagné).
Un tel jeu est appelé jeu d'accessibilité à deux joueurs.

Question 1.1 *Représenter l'ensemble des configurations possibles et les joindre par un arc lorsque l'on peut passer de l'une à l'autre (on prendra* $(n, p) = (2, 3)$*).*

Pour tenir compte de la règle du jeu à deux joueurs, on va construire un autre graphe pour représenter ce jeu. On note respectivement J_1 et J_2 les deux joueurs, avec la convention que J_1 est le premier joueur à jouer. Le nouveau graphe a pour sommets l'ensemble des états possibles du jeu, un état étant constitué des informations suivantes : taille de la tablette à ce moment du jeu et joueur ayant la main. Par exemple, si le morceau de tablette restant est de taille $(2, 2)$ et que J_1 a la main, on notera $(2, 2, 1)$ l'état du jeu. Un arc relie deux sommets lorsque l'on peut passer de l'un à l'autre en une étape du jeu. Remarquons qu'un tel graphe a des sommets de deux catégories : ceux pour lesquels J_1 a la main et ceux pour lesquels J_2 a la main. Les arêtes relient nécessairement des sommets de catégories différentes. Un tel graphe est dit biparti.
Pour achever de représenter le jeux, il reste à faire figurer le sommet de départ ainsi que ceux où le jeu s'achève.

Question 1.2 *Représenter le graphe décrit ci dessus pour le jeu de chomp (on prendra toujours* $(n, p) = (2, 3)$*). On mettra les états pour lesquels un même joueur a la main sur une même ligne.*

L'exemple que nous avons pris est caractéristique des jeux d'accessibilité ayant les caractéristiques suivantes :

- le jeu est à information complète ;
- le jeu est déterministe ;
- les joueurs jouent à tour de rôle.

La formalisation d'un tel jeu se fait à travers la notion dégagée précédemment de graphe biparti. Plus précisément, on définit un jeu d'accessibilité à deux joueurs J_1 et J_2 comme étant un graphe orienté fini (S, A) appelé arène tel que :

- il existe une partition $S = S_1 \cup S_2$ et les arcs vont de S_i à S_j pour $i \neq j$;
- un sommet de départ s_0 est donné ;
- est également donnée une partition de l'ensemble F des états finaux : $F = G_1 \cup G_2 \cup N$ (états finaux gagnants pour J_1, J_2 et états nuls).

Une partie est un chemin de ce graphe qui commence en s_0 et qui finit en un sommet de F.

2 Une première stratégie de résolution

Question 2.1 *Décrire de manière informelle une stratégie gagnante pour le joueur* J_1 *dans le cas du jeu de chomp (*$(2, 3)$*).*

Étant donné une arène telle que décrite dans le paragraphe précédent (et en gardant les mêmes notations), une stratégie pour le joueur J_i est une application $f : S_i \to S$ telle que pour tout $s \in S_i$, $(s, f(s))$ appartienne à l'ensemble A des arcs.
Une stratégie est dite gagnante lorsque toute partie jouée selon cette stratégie est gagnante.

Question 2.2 *Décrire de manière plus formelle (en explicitant la fonction f) une stratégie gagnante pour le joueur J_1 dans le cas du jeu de champ ($(2,3)$).*

Afin de déterminer une stratégie gagnante dans le cas général, on définit la notion d'attracteur. Pour un sous-ensemble H de S, on appelle attracteur d'ordre n de H pour le joueur i l'ensemble des sommets à partir desquels le joueur i a une stratégie telle que la partie mène nécessairement à un sommet de H en au plus n coups.
Plus précisément, on définit cet attracteur At_i^n par récurrence de la manière suivante (on note j l'autre joueur) :

- $At_i^0(H) = H$;
- $At_i^{n+1}(H)$ est constitué :
  - des éléments de $At_i^n(H)$;
  - des sommets s de S_i tels qu'il existe une arête issue de s menant à $At_i^n(H)$;
  - des sommets s de S_j tels que toutes les arêtes issues de s mènent à $At_i^n(H)$.

Question 2.3 *Reprendre le graphe biparti représentant le jeu de champ ($(2,3)$) et construire les attracteurs de rangs successifs du sommet gagnant pour J_1.*

Cette suite d'attracteurs a les propriétés suivantes :

- Elle est croissante et stationnaire (à partir au plus du rang $Card(S)$) ; on note $At_i(H)$ leur réunion.
- Les positions gagnantes pour J_i sont les éléments de $At_i(G_i)$.
- Le joueur J_i possède une stratégie gagnante si et seulement si le sommet de départ s_0 est dans $At_i(G_i)$. Dans ce cas, la détermination de la suite des $At_i^n(G_i)$ permet de définir cette stratégie.

Question 2.4 *Nous allons maintenant écrire un algorithme permettant de déterminer l'ensemble des positions gagnantes pour un joueur (toujours dans le cas du jeu de chomp).*
Un graphe sera représenté par un dictionnaire : les clefs sont les sommets et la valeur associée à chaque clef s est la liste des sommets accessibles à partir de s.
Le fichier `chomp_ACompleter.py` *contient des trames pour les fonctions demandées ci-dessous.*

1. *On pourrait bien écrire ces dictionnaires de manière explicite pour le jeu de chomp($(2,3)$). On va plutôt écrire deux fonctions* `graphChomp(n,p)` *et* `graphBipartiChomp(n,p)` *renvoyant les dictionnaires représentant ces graphes.*
   Afin de tester ces fonctions, on pourra écrire tout d'abord ces dictionnaires à la main.

2. *On aura besoin de fonctions permettant d'une part de tester si on peut atteindre un sous-ensemble de sommets (représenté par une liste de sommets) à partir d'un sommet s et d'autre part si toutes les arêtes issues d'un sommet s mènent à un sous-ensemble donné. Compléter ces fonction dans le fichier .py.*

3. *Écrire une fonction* `attracteurChomp(n,p,joueur)` *permettant de déterminer l'attracteur du sommet gagnant pour le joueur* `joueur`.
   Tester sur l'exemple fait auparavant à la main.

4. *Écrire une fonction* `strategieChomp(n,p,joueur,pos)` *qui, étant donné une position* `pos` *dans laquelle se trouve* `joueur`*, doit renvoyer une position (de l'autre joueur) qui assurera le gain de la partie pour* `joueur` *(c'est à dire une position dans l'attracteur de la position gagnante de joueur).*

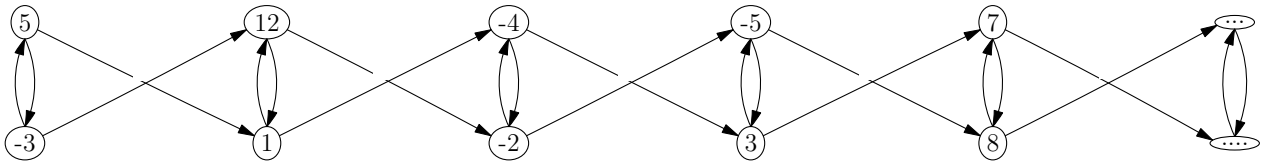
3 Algorithme du min-max

3.1 Introduction

On a abordé dans le paragraphe précédent une situation de jeu dans laquelle à l'aide d'un graphe on pouvait identifier les positions gagnantes et dégager une stratégie à partir de ces positions. Cela était possible car le graphe associé au jeu (l'arène) était très petite. Il est clair que dans de nombreux jeux (échecs, go, puissance quatre ...), on n'est pas dans cette situation : les positions possibles du jeu sont en très grand nombre et il est impossible de les traiter de manière exhaustive.
Dans ces situations, on met en œuvre une autre stratégie dont le principe est le suivant. On associe à chaque situation de jeu un score à travers une fonction appelée fonction d'utilité. Plus la position est bonne (pour un joueur), plus le score doit être élevé (pour ce joueur). Par exemple, aux échecs, une telle fonction pourrait tenir compte du nombre de pièces de chaque joueur, du nombre de déplacements possibles pour chacune de ces pièces, du nombre de cases contrôlées ... La construction d'une telle fonction demande bien sûr une connaissance fine du jeu et peut constamment être améliorée après des retours d'expérience. Une telle fonction est appelée heuristique.
Une fois cette heuristique définie, la première méthode qui peut être envisagée est, à chaque coup, d'essayer de maximiser cette heuristique. Mais cela n'est pas suffisant : il faut aussi s'assurer que l'autre joueur ne pourra pas, à partir de la position atteinte après cette maximisation, obtenir un score intéressant (pour lui, c'est à dire un score bas pour le premier joueur). On peut évidemment continuer ce raisonnement avec un horizon de plusieurs coups.

3.2 Exemple

Pour expliquer les détails de cet algorithme, on se base sur un exemple. Celui-ci ne découle pas d'un jeu connu : on part d'une arène.



La première ligne représente les positions de jeu du joueur J_1, la seconde celles du joueur J_2. Le contenu de chaque sommet est la valeur de la position (pour J_1). On suppose que la partie commence au sommet en haut à gauche. Les sommets seront repérés si besoin par un couple (numéro de ligne , numéro de colonne).

Première approche : horizon deux coups

Quel coup doit jouer J_1 s'il ne prend en compte que la valeur de sa position d'arrivée ?

Quel coup doit jouer J_1 s'il prend en compte le fait que J_2 (supposé connaître également l'heuristique) va chercher à minimiser le score lors de son coup suivant.

Pour représenter la prise en compte des possibilités du jeu à horizon deux coups, on représente dans un arbre toutes les possibilités du jeu à cet horizon :

On écrit sur les feuilles les valeurs des positions finales. Puis on remonte l'arbre en donnant à chaque nœud une valeur selon le principe suivant. Si ce nœud est une position de jeux de J_1, la valeur prise est la valeur maximale parmi celles des fils ; si ce nœud est une position de jeux de J_2, la valeur prise est la valeur minimale parmi celles des fils. Ce processus est à l'origine du nom de l'algorithme. Cette démarche donne également le coup le plus efficace (avec cette heuristique et cet horizon) pour J_1.
Remarque : on suppose bien sûr que J_1 et J_2 jouent à même horizon.

Stratégie à horizon 4 coups

Dessiner l'arbre représentant les possibilités du jeu à horizon 4 coups. Donner une valeur à chacun de ces sommets selon le principe expliqué ci-dessus. Identifier le coup optimal pour J_1.

3.3 Implémentation

On représente le graphe de façon usuelle par un dictionnaire. Un autre dictionnaire contient les valeurs de chaque position données par l'heuristique.
On écrit une fonction récursive `minMax(pos , nbCoups)` qui renvoie le score optimal obtenu à partir de la position `pos` en applicant la stratégie de l'algorithme min-max avec une profondeur de `nbCoups`. Le principe de la récursivité est le suivant :

- L'appel à la fonction `minMax(pos , nbCoups)` va s'appuyer sur des appels avec `nbCoups-1`, les positions étant celles des fils de `pos`.
- Il faut faire attention au fait que selon que `pos` est une position où J_1 doit jouer ou bien J_2 doit jouer, on doit choisir le maximum ou le minimum des scores obtenus par ses fils.

Compléter le fichier `Ex_minmax_Acompleter` qui contient la construction des dictionnaires ainsi qu'une trame de la fonction `minMax`.
Écrire ensuite une fonction `minMax2`, évolution de la précédente, qui renvoie en plus le coup optimal qui doit être joué.