Chapitre 4

Théorie des jeux (partie 1)

Dans cette partie, nous allons étudier un certain type de jeux, le but étant de trouver un moyen de gagner.

Table des matières

1 Un exemple : le jeu de Nim 2

2 Modélisation d'une partie 3

3 Calcul des attracteurs 4

4 Construction d'une stratégie gagnante 5

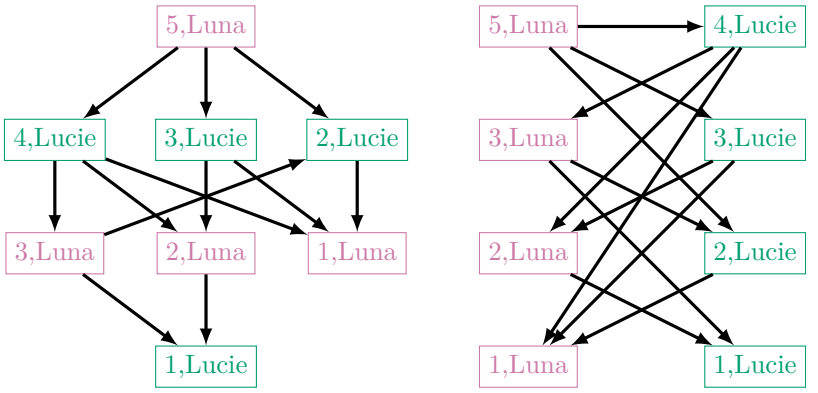
1 Un exemple : le jeu de Nim

Imaginons deux joueurs et un tas de N allumettes. Chacun à son tour, les joueurs jouent en respectant deux règles :

- Ils enlèvent une, deux ou trois allumettes au choix
- Mais ils doivent laisser au moins une allumette dans le tas.

Celui qui doit jouer alors qu'il reste exactement une allumette a perdu car il ne peut pas jouer en respectant les deux règles :

Prenons un exemple, où Luna et Lucie jouent avec 5 allumettes. Luna commence. On peut faire un graphe de toutes les possibilités :



(a) Tentative de classer de haut en bas les coups joués dans l'ordre

(b) Les sommets où Luna joue sont à gauche, ceux où Lucie joue à droite

FIGURE 1 – Deux représentation du graphe

Le sommet 5,Luna veut dire qu'il y a actuellement cinq allumettes et que c'est à Luna de jouer. Le sommet 1,Lucie veut dire qu'il reste 1 allumette et que c'est à Lucie de jouer, elle a donc perdu. De même 1,Luna veut dire que Luna a perdu. Ainsi, une des parties possibles serait :

$$5,\text{Luna} \underset{\text{Luna}}{\longrightarrow} 4,\text{Lucie} \underset{\text{Lucie}}{\longrightarrow} 2,\text{Luna} \underset{\text{Luna}}{\longrightarrow} 1,\text{Lucie}$$

Ainsi, Lucie perd cette partie. Remarquons, qu'il était possible pour Lucie de gagner cette partie quoique fasse Luna, en effet, après le premier coup de Luna, il reste, 4, 3 ou 2 allumettes dans le tas, ainsi Lucie peut toujours enlever 3, 2 ou 1 allumettes pour que Luna se retrouve dans la case 1,Luna et perde donc la partie. Ainsi, Lucie avait une stratégie gagnante.

Remarquons que ce graphe a une particularité : chaque sommet est contrôlé par exactement une des joueuses. On peut donc séparer les sommets en deux paquets, ainsi on préférera la représentation à droite.

Cherchons maintenant à implémenter ce graphe en Python. Rappelons un peu de vocabulaire sur les graphes : Si (s_1, s_2) est un arc [1], on dit que s_2 est un **successeur** de s_1 et que s_1 est un **prédécesseur** de s_2. On note $d_+(s_1)$ le nombre de successeurs de s_1, $d_+(s_1)$ est appelé **degré sortant** de s_1 et $d_-(s_2)$ le nombre de prédécesseurs de s_2, $d_-(s_1)$ est appelé **degré entrant** de s_2. Rappelons également qu'il existe plusieurs façons de coder des graphes :

- Avec un couple `(D,M)` où `D` est un dictionnaire dont les clés sont les numéros de sommets, et les valeurs sont le numéro du sommet et `M` la matrice d'adjacence : `M[i,j]=1` s'il existe un arc reliant `i` à `j` si le graphe est non valué, 0 sinon [2].
- Avec un couple `G=(S,A)` où `S` est une liste de sommets et `A` est la liste des arêtes.
- `G` un dictionnaire dont les clés sont les sommets et la valeur associée à un sommet est la liste de successeurs de ce sommet. Ainsi, `G[s]=[a,b]` signifie que `s` est un sommet et qu'il y a un arc reliant `s` à `a` et un autre reliant `s` à `b`.

C'est de cette dernière façon que nous allons utiliser les graphes.

Procédons par récursivité : en partant d'un sommet de la forme `(n,j)` on va aller visiter les trois successeurs de ce sommet : `(n-1,o)`,`(n-2,o)`,`(n-3,o)` où `o` est le nom de l'opposant du joueur, mais attention, il faut garder en tête que le nombre d'allumettes doit être supérieure ou égale à 1.

1. Dans le cas d'un graphe non orienté, on note $d(s)$, le **degré** de s, c'est-à-dire le nombre de voisins de s.
2. `M[i,j]` vaudra la valeur de l'arc si le graphe est valué.

```
def GrapheNim(N,joueur):
    """Renvoie le graphe du jeu de Nim partant de N allumettes avec joueur commence"""
    if (N,joueur) not in G:#Si on n'a pas encore rajouté ce sommet
        G[(N,joueur)]=[(k,Opp[joueur]) for k in range(max(N-3,1),N)]#Liste des successeurs de (N,joueur)
        for (n,o) in G[(N,joueur)]:
            GrapheNim(n,o)#on va visiter ce successeur

Opp={"Luna":"Lucie","Lucie":"Luna"}# dictionnaire des opposantes
G={}#Dictionnaire du graphe: les clés sont les sommets, G[s]=la liste des succeeurs de s
GrapheNim(8,"Luna")
```

On remarque que cette fonction n'est rien de plus qu'un parcours en profondeur récursif. On peut d'ailleurs reprogrammer cette fonction avec un parcours en profondeur avec une pile ou avec un parcours en largeur avec une file :

```
def GrapheNimP(N,joueur):
    """Renvoie le graphe du jeu de Nim partant de N allumettes avec joueur commence
    (parcours en profondeur par pile)"""
    Pile=[(N,joueur)]#Pile avec le sommet initial
    while len(Pile)>0:#Tant que toute la pile n'est pas finie (tant qu'il y a du boulot à faire on bosse)
        (N,joueur)=Pile.pop()#On dépile#le dernier arrivé est servi (c'est un peu dégueu)
        if (N,joueur) not in GP:#S'il n'a pas déjà été traitée
            GP[(N,joueur)]=[(k,Opp[joueur]) for k in range(max(N-3,1),N)]#Liste des successeurs de (N,joueur)
            Pile=Pile+[(n,o) for (n,o) in GP[(N,joueur)] if (n,o) not in GP]
            #On rajoute les successeurs à la pile

def GrapheNimL(N,joueur):
    """Renvoie le graphe du jeu de Nim partant de N allumettes avec joueur commence
     (parcours en largeur par file)"""
    File=[(N,joueur)]#File avec le sommet initial
    while len(File)>0:#Tant que toute la pile n'est pas finie (tant qu'il y a du boulot à faire on bosse)
        (N,joueur)=File.pop(0)#On défile: le premier arrivé est servi (c'est plus juste)
        if (N,joueur) not in GL:
            GL[(N,joueur)]=[(k,Opp[joueur]) for k in range(max(N-3,1),N)]#Liste des successeurs
            #de (N,joueur)
            File=File+[(n,o) for (n,o) in GP[(N,joueur)] if (n,o) not in GL]

GP={}#Graphe du parcours en profondeur avec pile
GrapheNimP(8,"Luna")
GL={}#Graphe du parcours en largeur avec une file
GrapheNimL(8,"Luna")
print(G==GP and G=GL)#On teste si les trois graphes obtenus par trois méthodes sont bien identiques
#Of course, ils le sont, le contraire aurait été inquiétant
```

2 Modélisation d'une partie

De manière générale, on peut imaginer un jeu se jouant sur un **graphe orienté non valué** appelé **arène** avec deux joueurs J_1 et J_2. Chaque sommet désigne **une configuration du jeu**. L'un de ces sommets est la position de départ.

Si on a une arête qui relie s_1 à s_2 veut dire que l'un des joueurs à la possibilité de passer de s_1 à s_2. Le graphe possède une propriété d'être **bipartie** : l'ensemble des sommets S vérifie[3] $S = S_{J_1} \cup S_{J_2}$ avec $S_{J_1} \cap S_{J_2} = \varnothing$, S_{J_1} (resp. S_{J_2}) désigne l'ensemble des sommets à partir duquel J_1 (resp. J_2) joue. Si $s \in S_{J_1}$, on dit que c'est un sommet contrôlé par J_1. De plus, les arcs ne peuvent relier que l'un des sommets de S_{j_1} vers un sommet de S_{J_2} et inversement[4].

Un sommet sans successeur est appelé **sommet terminal** (le jeu s'arrête donc), un sommet sans prédécesseur est appelé **sommet initial**.

On suppose le graphe fini et **acyclique** (il n'y a pas de cycle), ainsi partant d'un sommet initial s_0, on est obligé de finir sur un sommet n'ayant pas de successeur et donc le jeu s'arrête. Notons `T` la liste des sommets terminaux, alors `T=G1∪G2∪N`.

3. Mathématiquement, S_{J_1} et S_{J_2} forment une partition de l'ensemble des sommets.
4. Cela traduit qu'une fois que J_1 a joué, c'est à J_2 à jouer, on évite a priori donc les jeux, où J_1 peut jouer plusieurs fois ou J_1 et J_2 jouent simultanément.

Les sommets de `G1` sont synonymes de victoires pour J_1, les sommets de `G2` sont synonymes de victoires pour J_2 et les sommets de `N` de parties nulles.

Ainsi, une partie où J_1 commence est un chemin de la forme

$$s_0 \xrightarrow[J_1]{} s_1 \xrightarrow[J_2]{} s_2 \xrightarrow[J_1]{} s_3 \longrightarrow \dots \longrightarrow s_n$$

Où $s_i s_{i+1}$ est un arc du graphe. Les sommets s_0, s_2, s_4 etc. sont contrôlés par J_1. Cela veut dire, qu'au début de la partie, c'est J_1 qui a choisi d'aller s_1 (parmi tous les successeurs de s_0), d'aller en s_3 (parmi les successeurs de s_2). En revanche, les sommets s_1, s_3, s_5 etc. sont contrôlés par J_2. Cela veut dire que quand on était en s_1 c'est J_2 qui a choisi d'aller en s_2 (parmi tous les successeurs de s_1) etc. Le sommet s_n est un sommet terminal et donc la partie s'arrête, suivant que s_n appartiennent à `G1`, à `G2`, ou à `N`, on a le résultat de la partie.

Le graphe est connu des joueurs, il n'y a pas d'informations cachés, pas de hasard.

Les jeux dont le graphe est **bipartie**, que les parties s'arrêtent sur trois états (victoires de J_1, victoire de J_2 ou nul) s'appellent **des jeux d'accessibilité**.

Une **stratégie** du joueur J_1 est une fonction ϕ: $\begin{cases} S_{J_1} \longrightarrow S_{J_2} \\ s_1 \longmapsto s_2 \in G[s_1] \end{cases}$. Cela veut dire que si le joueur J_1 doit jouer alors qu'il est en position s_1, alors il jouera la position s_2 (un des successeurs de s_1). Conformément au programme, c'est une **stratégie sans mémoire** : le joueur joue seulement en fonction de la position actuelle s_1 et non en fonction de l'historique de la partie.

Une **stratégie** ϕ est dite **gagnante** pour J_1 depuis le sommet s_0 si toute partie jouée depuis s_0 en suivant ϕ est gagnante.

Une **position** est dite gagnante pour J_1 s'il existe une stratégie gagnante partant de cette position.

3 Calcul des attracteurs

Rappelons que `G1` désigne l'ensemble des sommets terminaux sur lesquels J_1 gagne la partie, on cherche à déterminer comment être sûr d'y arriver, pour cela on va essayer de remonter le temps. Posons :

$$\begin{array}{rrcl} & A_0 & = & \texttt{G1} \\ \forall n \in \mathbb{N} & A_{n+1} & = & A_n \cup \{s \in S_{J_1} \mid A_n \cap G[s] \neq \varnothing\} \cup \{s \in S_{J_2} \mid G[s] \subset A_n\} \end{array}$$

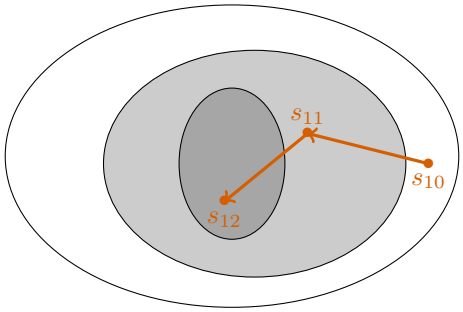


FIGURE 2 – Les premiers ensembles A_0 (en gris foncé), A_1 (union du gris foncé et gris clair) et A_2 (la grande ellipse) : Au 10-ième coup, on est au sommet appelé $s_{10} \in A_2$ si c'est à J_1 de jouer, il pourra choisir au moins un coup $s_{11} \in A_1$, si c'est à J_2 de jouer, tous les coups qu'il pourra faire le mèneront à un sommet dans $s_{11} \in A_1$. Encore une fois, si c'est à J_1 de jouer, il pourra jouer au moins un sommet $s_{12} \in A_0$ et gagner, si c'est à J_2 de jouer, quelque soit le coup qu'il peut jouer, ce sera un coup $s_{12} \in A_0$. Dans tous, les cas, J_1 gagne.

Ainsi, A_{n+1} est l'union de A_n et de l'ensemble des sommets contrôlés par J_1 qui permettent d'arriver à A_n et de l'ensemble des sommets contrôlés par J_2 ou quelque soit le coup joué par ce dernier, il sera obligé d'aller dans A_n.

Détaillons ce qui se passe sur les premiers A_n :

- Si on est dans une position qui est dans A_0, alors la partie est gagné par J_1, autrement dit la partie est déjà gagné donc (est gagné en 0 coup)
- Si on est dans une position qui est dans A_1 mais pas dans A_0, alors si c'est au joueur J_1 de jouer, il peut choisir d'aller en un coup dans une positon de A_0, si c'est au joueur adverse de jouer, quelque soit le coup qu'il peut jouer, il va tomber dans une positon de A_0. Dans les deux cas, J_1 va gagner en 1 coup.

- si on est dans une position qui est dans A_2 mais pas dans A_1, alors si c'est au joueur de J_1 de jouer, il peut choisir d'aller en A_1, ainsi par ce qui précède il va gagner en deux coups, si c'est au joueur J_2 il va être forcer d'aller en A_1, ainsi J_1 va gagner en deux coups.

En généralisant un peu (par une récurrence), on peut aboutir à la conclusion suivante : si on est dans une position de A_n, alors J_1 peut gagner en n coups ou moins. Ce qui justifie que ces ensembles A_n vont être de solides alliés pour gagner. Il faudrait donc être en mesure de calculer ces ensembles A_n. C'est ce que nous allons faire :

Soit X un ensemble et J un joueur, notons :

$$F(X,J) = (\{s \in S_J \mid \exists \text{ successeur } \in X \cap G[s]\} \cup \{s \notin S_J \mid \forall \text{ successeur } \in G[s] \quad \text{successeur} \in X\}) \setminus X$$

De sorte que $A_{n+1} = A_n \cup F(A_n, J_1)$. Pour coder la fonction `F`, on va s'aider, de deux fonctions auxiliaires `Existence(X,s)` et `PourTout(X,s)` la première renverra `True` si l'un des successeurs du sommet est dans X et la seconde renverra `True` si tous les successeurs de `s` sont dans X :

```
def Existence(X,s):
    """Renvoie True s'il existe un successeur de s dans X"""
    for succ in G[s]:
        if succ in X:#l'un des successeurs est dans X
            return True
    return False

def PourTout(X,s):
    """Renvoie True si tous les successeurs de s sont dans X"""
    for succ in G[s]:
        if succ not in X:#l'un des successeurs n'est pas dans X
            return False
    return True#Tous les successeurs sont dans X
```

On code alors la fonction `F(X,j)` en bouclant sur tous les sommets du graphe, on distingue les cas où le sommet est contrôlé par le joueur `j` ou par son adversaire :

```
def F(X,j):
    L=[]
    for s in G:
        if s not in X and s[1]==j and Existence(X,s):#s est contrôlé par j
            L.append(s)
        if s not in X and s[1]!=j and PourTout(X,s):#s est contrôlé par l'adversaire
            L.append(s)
    return L
```

Ainsi, pour tout $n \in \mathbb{N}$, $A_{n+1} = A_n \cup F(A_n, J_1)$ et $A_n \cap F(A_n) = \varnothing$, $A_n \subset A_{n+1}$. Notons qu'il existe $N \in \mathbb{N}$ tel que $A_N = A_{N+1}$. De plus, pour ce $N \in \mathbb{N}$, on a $A_N = \bigcup_{n\in\mathbb{N}} A_n$. Ainsi, A_N est l'ensemble des positions où J_1 est assuré de pouvoir gagner (s'il joue bien). On dit que A_N est **l'attracteur** pour J_1 : si on tombe dans l'attracteur alors on y reste (à condition que J_1 joue bien) quoique fasse J_2.

```
def Attracteur(joueur):
    X=[(1,Opp[joueur])]#Liste des sommets gagnants pour joueur: X=A0
    Y=F(X,joueur)
    while len(Y)>0:#tant qu'il y a des choses à rajouter
        X=X+Y#On actualise X, à la nième itération X=An
        Y=F(X,joueur)
    return X

print(Attracteur("Luna"))
```

4 Construction d'une stratégie gagnante

Avec le calcul d'attracteur fait précédemment, on peut construire une stratégie gagnante sur A_N :

Pour tout sommet s dans l'attracteur (i.e. $s \in A_N$ on définit le **rang** de s par :

$$r(s) = \min\{k \in \mathbb{N} \mid s \in A_k\}$$

Soit $s \in A_N$, supposons que ça soit au tour de J_1 de alors qu'il est en position s (en particulier $s \in S_{J_1}$). Si $r(s) = 0$, alors $s \in A_0 = G1$ et J_1 a donc gagné la partie, sinon, notons $n = r(s) - 1 \in \mathbb{N}$, $s \in A_{n+1}$ et $s \notin A_n$ Alors, parmi, les successeurs de s il existe $v \in A_n$, en particulier $r(v) \leqslant n < r(s)$, on pose alors, $\phi(s) = v$ et J_1 joue la position v. Maintenant, J_2 doit jouer la position, v, si $r(v) = 0$, alors $v \in A_0 = G1$ et J_1 a donc gagné la partie, sinon notons $m = r(v) - 1 \in \mathbb{N}$, $v \in A_{m+1}$ et $v \notin A_m$, ainsi n'importe quel coup c que peut jouer J_2 est dans A_m et vérifie donc $r(c) \leqslant m < r(v)$

Ainsi, partant d'un sommet de l'attracteur, J_1 peut toujours choisir un coup dont le rang est plus petit, et quelque soit le coup que J_2 joue, il atteindra une position dont le rang est encore plus petit. Comme le rang diminue, la victoire n'en est que plus proche. Ainsi, codons le rang d'un élément de l'attracteur, il suffit de tenir compte du nombre d'itérations que l'on a dû faire pour trouver le sommet, puis parmi tous les successeurs, il suffit d'en retourner un dont le rang est plus petit. La fonction ϕ ainsi construite est donc une stratégie gagnante si on part d'un point de l'attracteur.

```
def Strategie(joueur,sommet):
    n=0
    X=[(1,Opp[joueur])]#Liste des sommets gagnants pour joueur, X=A_0
    R={k:n for k in X}#dictionnaire des rangs, R[s] est le rang de s
    while sommet not in R:#tant qu'il y a des choses à rajouter
        n=n+1#on actualise n
        Y=F(X,joueur)#on regarde les nouveaux sommets obtenus
        for k in Y:
            R[k]=n#on déclare leur rang
        X=X+Y#X=f(X)=f(A_{n-1})=A_n
    for v in G[sommet]:#pour tous les successeurs de sommet
        if v in R and R[v]<R[sommet]:
            return v#on retourne un successeur de sommet dans l'attracteur et de rang plus petit

Strategie("Luna",(6,"Luna"))#Coup que peut jouer Luna dont le rang est plus petit que la position actuelle.
```

Que faire si on n'est pas sur un point de l'attracteur ? On choisit aléatoirement un sommet (qui n'est pas dans l'attracteur de l'adversaire si possible) et espérer que votre adversaire fasse une erreur pour que la suite de la partie tombe dans votre attracteur.