

Jeux d'accessibilité à deux joueurs

I Jeu d'accessibilité sur un graphe

1. Vocabulaire

Dans ce chapitre on s'intéresse à des jeux entre deux joueurs J_1 et J_2 qui satisfont les propriétés suivantes :

- chaque joueur joue à tour de rôle.
- le jeu est à *information complète* : chaque joueur voit la situation actuelle dans son ensemble ; ceci exclut la majorité des jeux de cartes pour lesquels on n'a en général pas la vision du jeu de son adversaire, ni des cartes dans une éventuelle pioche.
- le jeu est *sans mémoire* : la décision prise par le joueur qui doit jouer ne dépend que de la situation actuelle et pas des situations précédentes.
- le jeu est *sans hasard* : la décision prise par un joueur amène toujours à la même nouvelle situation ; ceci exclut les jeux avec des dés, pour lesquels la nouvelle situation va dépendre du résultat du lancer de dé.

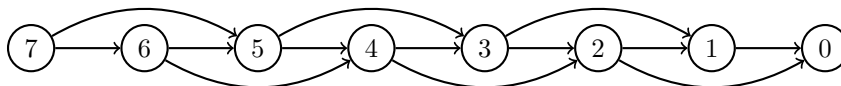
Un exemple d'un tel jeu est le *jeu de Nim* :

- ★ on dispose de n tas contenant chacun des jetons identiques : le tas $i \in \llbracket 1, n \rrbracket$ contient au départ $k_i \geq 1$ jetons.
- ★ chaque joueur joue à tour de rôle de la façon suivante : il choisit un des tas sur lequel il peut prendre entre 1 et p jetons, tous sur le même tas, mais pas forcément le même nombre à chaque fois.
- ★ le joueur qui prend le dernier jeton (de tous les tas) est déclaré perdant (et l'autre vainqueur).

2. Graphe associé à un jeu

Pour représenter un tel jeu, on introduit un graphe orienté $G = (V, E)$ dans lequel les sommets (les points de V) sont toutes les situations possibles au cours du jeu (ou les positions du jeu) et les arrêtes (les éléments de E) joignent deux sommets (donc deux positions) lorsqu'on peut passer de l'un à l'autre en une seule action.

Si on considère une version simpliste du jeu de Nim avec un seul tas, contenant 7 jetons et pour lequel on peut prendre 1 ou 2 jetons à chaque fois, le graphe associé est le suivant :

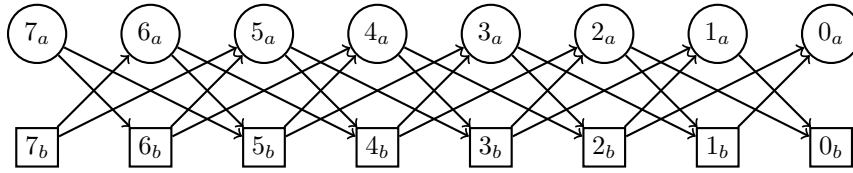


Une partie peut donc être modélisée de la façon suivante : chaque joueur déplace, à tour de rôle, un pion sur le graphe. Celui qui amène le pion sur le sommet 0 perd la partie.

De façon plus générale, la partie se termine lorsque le pion est sur un sommet sans successeur. Selon les règles du jeu, une telle position peut être une position gagnante (pour J_1 ou J_2 , donc perdante pour l'autre joueur) ou une position de match nul (qui n'existe pas dans le jeu de Nim).

3. Graphe biparti

De façon à préciser les positions jouées par le joueur J_1 et par le joueur J_2 (pour éviter de devoir préciser quel joueur joue), on va dupliquer le graphe du jeu de la façon suivante (dans la version simplifiée du jeu de Nim précédente) :



Dans cette représentation, les situations jouées par le joueur J_1 sont représentées par un cercle, celles jouées par J_2 par un carré. Une partie consiste donc à déplacer un pion sur les arrêtes du graphe, d'un rond vers un carré pour J_1 , d'un carré vers un rond pour J_2 .

Un tel graphe est appelé **graphe biparti** : l'ensemble des sommets V peut être partitionné en deux sous ensembles V_1 et V_2 (ie $V = V_1 \cup V_2$ et $V_1 \cap V_2 = \emptyset$) de sorte que toute arrête issue d'un sommet de V_1 pointe vers un sommet de V_2 et inversement ; il n'est donc pas possible de joindre deux sommets du même sous ensemble V_i par une seule arrête.

Les sommets de V_1 (les ronds) sont les sommets *contrôlés* par J_1 , ceux de V_2 (les carrés) sont ceux *contrôlés* par J_2 .

On peut remarquer que dans une telle situation, si c'est J_1 qui commence à jouer, le sommet 7_b n'est pas atteignable. Le joueur J_1 gagne si J_2 amène le pion sur le sommet 0_a , si c'est J_1 qui l'amène sur le sommet 0_b , c'est J_2 qui gagne.

II Stratégie gagnante

1. Stratégie

Dans un jeu d'accessibilité à deux joueurs, modélisé par un graphe biparti $G = (V, E)$, une **stratégie** pour le joueur J_i est une application $\sigma : V_i \rightarrow V_{3-i}$ telle que $\forall s \in V_i, (s, \sigma(s)) \in E$. On dit que le joueur J_i respecte la stratégie σ si pour toute partie jouée par J_i dont les positions (de J_i) successives sont $s_0, s_1, \dots, s_n, \dots$ et tout $k \in \mathbb{N}$, on a $s_{k+1} = \sigma(s_k)$.

Un joueur d'un tel jeu remporte la partie si une des positions qu'il atteint appartient à un sous ensemble F de V .

Dans le jeu de Nim à un tas précédemment introduit, pour le joueur J_1 , on peut prendre $F = \{1_b\}$: si le joueur J_1 arrive à déplacer le pion jusque sur le sommet 1_b , à l'étape d'après, le joueur J_2 sera obligé de prendre le dernier jeton donc perd la partie (et c'est donc J_1 qui gagne).

Une stratégie σ est dite *gagnante* pour le joueur J_i si pour toute partie jouée, et si le joueur J_i respecte la stratégie σ , indépendamment des coups joués par J_{3-i} , le joueur J_i remporte la partie. De façon plus générale, une position est dite *gagnante* pour J_i , s'il existe une stratégie gagnante pour J_i de puis cette position.

Résoudre un jeu consiste à déterminer les positions gagnantes pour un joueur et une stratégie permettant de gagner depuis ces positions.

2. Attracteur

On va introduire un algorithme permettant de déterminer les positions gagnantes pour un des joueurs.

On suppose le jeu modélisé par son graphe biparti $G = (V, E)$ (avec V partitionné en $V = V_1 \cup V_2$) comme précédemment et on note F_i l'ensemble des positions à atteindre pour le joueur J_i : dans le jeu de Nim précédent, $F_1 = \{1_b\}$ et $F_2 = \{1_a\}$ par exemple.

On définit alors les ensembles suivant :

$$A_0 = F_1$$

$$A_{k+1} = A_k \cup \{s_1 \in V_1 \mid \exists s \in A_k, (s_1, s) \in E\} \cup \{s_2 \in V_2 \mid \forall s' \in V_1, (s_2, s') \in E \Rightarrow s' \in A_k\}$$

On peut alors vérifier que A_k est l'ensemble des positions gagnantes pour J_1 en moins de k coups :

- A_0 est l'ensemble des positions à atteindre pour J_1 : toute position de A_0 est donc gagnante pour J_1 (en 0 coups).
- si on suppose que toute position de A_k est gagnante pour J_1 en moins de k coups et si $s \in A_{k+1}$, on a
 - ★ soit $s \in A_k$ donc s est gagnante pour J_1 en moins de k coups.
 - ★ soit $s \in \{s_1 \in V_1 \mid \exists s \in A_k, (s_1, s) \in E\}$ donc $s \in V_1$ est contrôlée par J_1 (donc c'est à J_1 de jouer) et il existe une arrête qui mène le pion à une position de A_k , qui est gagnante pour J_1 en moins de k coups.
 - ★ soit $s \in \{s_2 \in V_2 \mid \forall s' \in V_1, (s_2, s') \in E \Rightarrow s' \in A_k\}$ donc $s \in V_2$ est contrôlée par J_2 (donc c'est à J_2 de jouer) et tout coup jouable par J_2 ramène à une position de A_k , qui est gagnante pour J_1 en moins de k coups.

Dans tous les cas, s est donc gagnante pour J_1 en moins de $k+1$ coups.

On définit alors l'**attracteur** pour J_1 :

$$A = \bigcup_{k \in \mathbb{N}} A_k$$

qui est donc l'ensemble des positions gagnantes pour J_1 .

Remarque(s) :

- (II.1) Les ensembles A_k constituent une suite croissante pour l'inclusion donc $A_k = \bigcup_{0 \leq i \leq k} A_i$. De plus $A_k \subset V$ est donc stationnaire : la réunion définissant l'attracteur est donc finie, avec au plus n termes (n étant le nombre de sommets de G)

- (II.2) Dans un jeu sans match nul, comme le jeu de Nim, la détermination de l'attracteur de J_1 suffit : l'ensemble des positions gagnantes pour J_2 est alors le complémentaire de A .
- (II.3) Dans un jeu où des matchs nuls sont possibles, on introduit deux attracteurs : $A(J_1)$, défini comme précédemment, et $A(J_2)$ obtenu en échangeant les indices 1 et 2. Reste alors le complémentaire de $A(J_1) \cup A(J_2)$ qui est l'ensemble des positions conduisant à un match nul si J_1 et J_2 jouent tous les deux de manière optimale, ie de manière à ne jamais passer par une position de l'attracteur de leur adversaire.

3. Algorithme de détermination de l'attracteur

On va essentiellement partir des sommets de F_i et remonter le graphe pour déterminer les positions qui permettent d'arriver aux sommets de F_i en un coup pour déterminer A_1 , puis de même avec $A_2, \dots$

Pour cela on va commencer par déterminer le graphe transposé de G : c'est le graphe qui a les mêmes sommets que G et les mêmes arrêtes que G mais qui sont dirigées dans l'autre sens. On supposera que le graphe G est défini par son dictionnaire d'adjacence : les clés sont les sommets de G et $G[s]$ est la liste des sommets joignables depuis le sommet s .

```
def transp(G) :
    TG = {}
    for s in G :
        for t in G[s] :
            TG[t].append(s)
    return TG
```

La complexité de cette fonction est $O(n^2)$ si n est le nombre de sommets de G .

On peut alors déterminer l'attracteur du joueur J_i , $i \in \{1, 2\}$. Dans ce code, on adopte les notations suivantes :

- G est le graphe biparti du jeu.
- F l'ensemble des positions à atteindre pour le joueur considéré, représenté par la liste de ses sommets.

```
def attracteur(G, F) :
    TG = transp(G)
    A = {}
    test = {}
    for s in F :
        test[s] = True
    for k in range(1, len(G)) :
        Ak = []
        if k%2 == 1 :
            for s in A :
                for t in TG[s] :
                    if not test[t] :
                        test[t] = True
                        Ak.append(t)
            else :
                for s in A :
                    for t in TG[s] :
                        ok = True
                        for u in G[t] :
                            ok = ok and test[u]
                        if ok and not test[t] :
                            test[t] = True
                            Ak.append(t)
        A = Ak
    return [s for s in test if test[s]]
```

A_0
pour mémoriser l'attracteur
initialisation
les nouvelles positions à ajouter
détermination du joueur concerné
elles n'y sont pas déjà
on les rajoute dans l'attracteur
et dans Ak pour l'étape suivante
autre joueur
tester toutes les arrêtes sortantes
on vérifie u dans l'attracteur
pas encore dans l'attracteur
on l'y rajoute
et dans Ak pour l'étape suivante
pour l'étape suivante, on recommence
avec ces nouvelles positions
la liste de l'attracteur

Remarque(s) :

- (II.4) On utilise un dictionnaire `test` pour mémoriser les éléments qui sont dans les ensembles A_i ; la structure de dictionnaire est plus efficace qu'une liste pour tester l'appartenance à cet ensemble. Au départ, seuls les positions de F sont dans A_0 . À la fin (dernière ligne), il faut alors reconstruire la liste des positions gagnantes à partir de ce dictionnaire.
- (II.5) L'entier k sert à choisir le joueur concerné par le mouvement (et donc à savoir quels positions il contrôle) : si on cherche par exemple l'attracteur pour le joueur J_1

- l'ensemble F contient les positions gagnantes pour J_1 , donc des sommets contrôlés par J_2 , ie dans V_2 .
- les sommets que l'on rajoute pour passer de A_0 à A_1 sont alors des sommets de V_1 , contrôlés par J_1 , puisque le dernier coup qui permet d'attendre les sommets de F sont joués par J_1 .
- les sommets que l'on rajoute pour passer de A_1 à A_2 sont alors des sommets de V_2 puisque le coup précédent est joué par J_2 , ...

Lorsque le test `k%2 == 1` est réalisé, c'est le joueur dont on cherche l'attracteur qui joue, sinon c'est l'autre joueur.

- (II.6) La variable `Ak` contient donc ces nouvelles positions à rajouter à l'ensemble A_{k-1} pour obtenir A_k . On ne peut pas travailler directement avec la liste `A` car la boucle `for s in A` poserait problème si on fait évoluer la liste `A` elle-même au cours de la boucle.
- (II.7) Lorsque `k%2 == 1`, tous les sommets qui atteignent A_{k-1} sont bons.
- (II.8) Lorsque `k%2 == 0`, il ne faut garder que les sommets dont toutes les arrêtes sortantes tombent dans A_{k-1} (ce qui est vérifié par la variable `ok`)
- (II.9) La complexité de cette fonction est $O(n^3)$: il y a au pire 4 boucles `for` imbriquées mais, comme les ensembles `Ak` sont disjoints par construction, les boucles imbriquées `for k in range(1,len(G))` et `for s in A` ont une complexité globale en $O(n)$.