COURS : ÉTUDE DES JEUX

I) JEUX À DEUX JOUEURS 2
I.1. Introduction et définitions 2
I.1.1. Graphe de jeu 2
I.1.2. Partie et partie partielle 3
I.1.3. Condition de gain, état final 4
I.1.4. Stratégie avec mémoire et stratégie sans mémoire 5
I.1.5. Stratégie gagnante et position gagnante 7
I.2. Arbre d'un jeu 8
II) JEUX D'ACCESSIBILITÉ 9
II.1. Définition, condition d'accessibilité 9
II.2. Attracteur et piège de l'ensemble des sommets finaux 9
II.3. Principe de l'algorithme de calcul de l'attracteur 11
II.4. Algorithme de calcul de l'attracteur 13
II.5. Complexité de l'algorithme de calcul de l'attracteur 14
III) CONSTRUCTION D'UNE STRATÉGIE GAGNANTE PAR ATTRACTEURS 17
III.1. Jeux sans nuls 18
III.2. Jeux avec nuls 19
IV) ALGORITHME DU MINIMAX 20
IV.1. Principe de l'algorithme 20
IV.2. Algorithme et complexité 21
IV.3. Exemple 22
V) ALGORITHME DU MINIMAX AVEC HEURISTIQUE 23

I) JEUX À DEUX JOUEURS

I.1. Introduction et définitions

I.1.1. Graphe de jeu

Un jeu à deux joueurs J_0 et J_1 est modélisé par un graphe orienté biparti $G = (S, A)$, où S est partitionné en deux parties S_0 et S_1 telles que toute arête de A admette une extrémité dans S_0 et une extrémité dans S_1 : la partie S_i s'appelle l'ensemble des **sommets du joueur J_i** (l'ensemble des **états contrôlés par J_i**).

Concrètement, S_i est l'ensemble des sommets pour lesquels c'est au joueur J_i de jouer. Le graphe G muni de cette partition s'appelle un **graphe de jeu** (une **arène**).

Exemple de la Figure 1: les sommets sont séparés en deux colonnes S_0 (états où c'est à J_0 de jouer – ronds bleus doublement cerclés) et S_1 (états où c'est à J_1 de jouer – en vert)

- $S_0 = \{s_0, s_2, s_4, s_6, s_8\}$: sommets contrôlés par J_0
- $S_1 = \{s_1, s_3, s_5, s_7\}$: sommets contrôlés par J_1

Le graphe étant biparti, toute arête va d'une colonne à l'autre (donc les tours alternent).

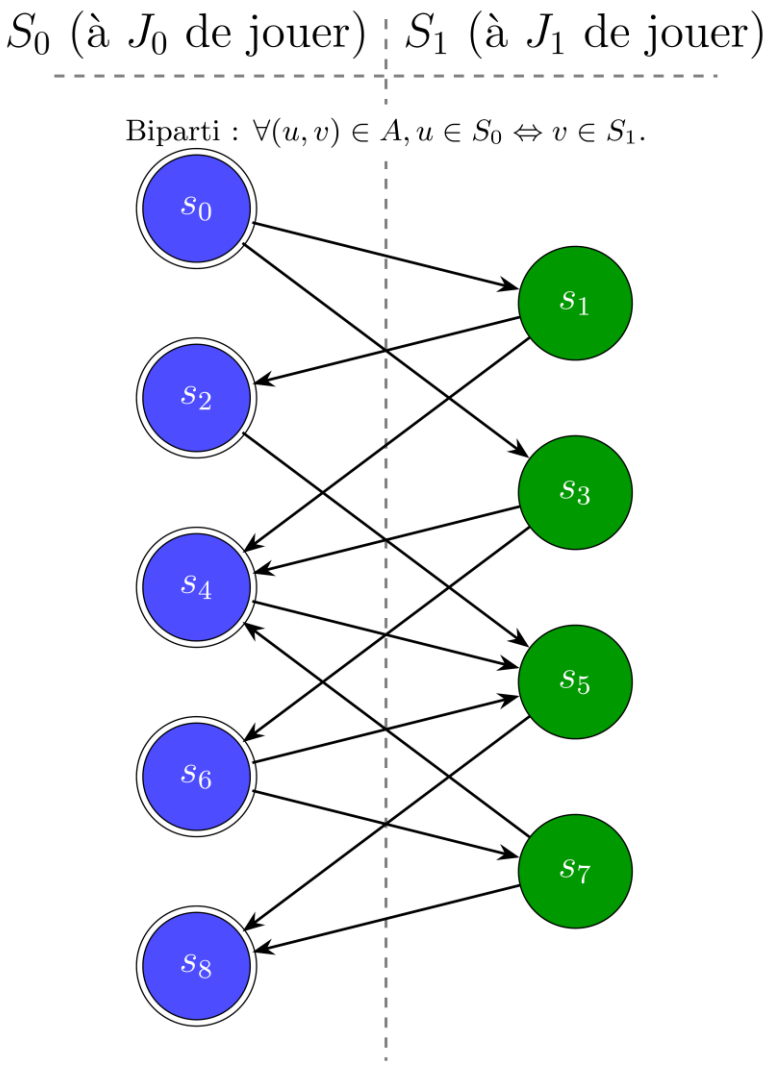


Figure 1: Illustration d'un graphe de jeu (arène)

I.1.2. Partie et partie partielle

Une **partie** P est un chemin maximal dans G (il ne peut pas être prolongé). Elle est construite de la façon suivante :

- Un sommet $s_0 \in S$, appelé position de départ, est d'abord fixé. L'appartenance de s_0 à S_i signifie que le joueur J_i commence la partie.
- Le joueur J_i tel que $s_0 \in S_i$, choisit alors s_1 tel que $(s_0, s_1) \in A$.
- L'autre joueur, J_j, tel que $s_1 \in S_j$, choisit alors s_2 tel que $(s_1, s_2) \in A$.
- Et ainsi de suite.

On note $\mathcal{P}$ l'ensemble de toutes les parties et l'on appelle **partie partielle** de $\mathcal{P}$ tout n-uplet formé des n premiers termes de la partie P.

Sur la Figure 2, on a choisi $s_0 \in S_0$ et c'est au joueur J_0 de commencer. Deux parties sont illustrées :

- $P_1 = (s_0, s_3, s_6, s_5, s_8, s_7)$ (en traits pleins rouges),
- $P_2 = (s_0, s_1, s_4, s_7)$ (en traits pointillés bleus).

Ce sont des chemins maximaux car s_7 n'a aucune flèche sortante et donc le chemin ne peut pas être prolongé depuis ce sommet.

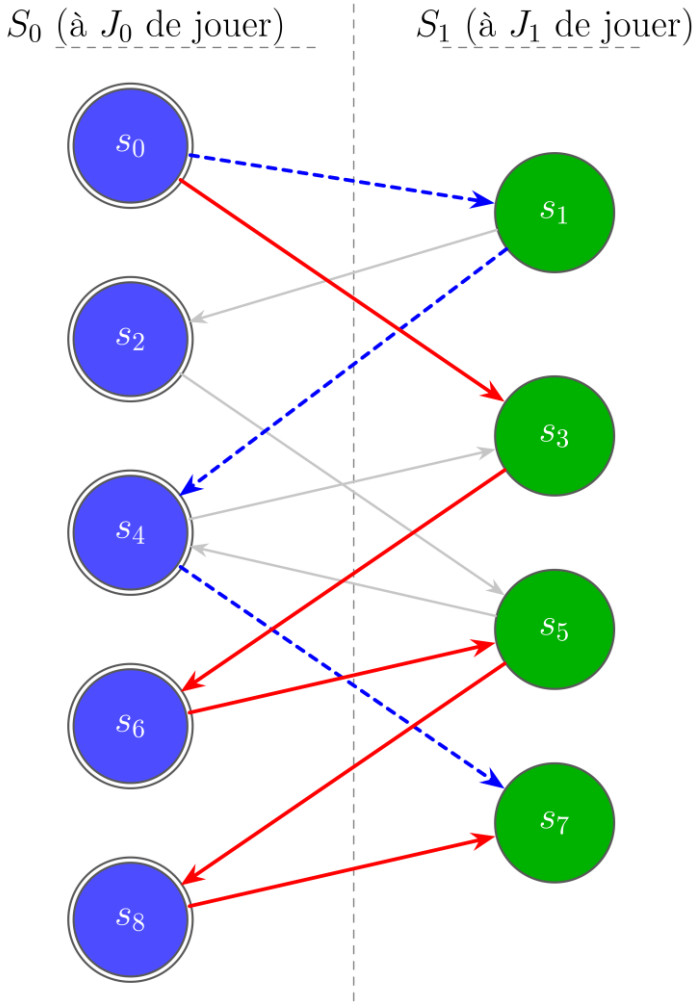


Figure 2 : Illustration de deux parties (chemin maximal)

I.1.3. Condition de gain, état final

Une **condition de gain** pour le joueur J_i est un sous-ensemble C_i de $\mathcal{P}$, tel que le joueur J_i gagne la partie P si $P \in C_i$.

Ainsi, un jeu à deux joueurs sur G peut être défini comme un triplet (G, C_0, C_1), où C_i est une condition de gain pour J_i (avec l'hypothèse que $C_0 \cap C_1 = \emptyset$).

Les sommets du graphe biparti dont aucune arête ne sort sont appelés **états finaux** : un état final peut être gagnant pour J_0, gagnant pour J_1 ou de match nul.

Exemple : sur la Figure 3, on a choisi $s_0 \in S_0$ et c'est au joueur J_0 de commencer la partie. L'ensemble des parties possibles est $\mathcal{P} = \{P_1, P_2, P_3, P_4\}$: $P_1 = (s_0, s_1, s_2)$, $P_2 = (s_0, s_1, s_4)$, $P_3 = (s_0, s_3, s_6)$ et $P_4 = (s_0, s_3, s_2)$.

Dans cet exemple :

- s_2 est un état final gagnant pour le joueur J_0 ;
- s_4 est un état gagnant pour le joueur J_1 ;
- s_6 est un état de match nul.

On a $C_0 = \{P_1, P_4\}$ et $C_1 = \{P_2\}$ et l'ensemble des matchs nuls $\mathcal{M} = \mathcal{P} \backslash (C_0 \cup C_1) = \{P_3\}$.

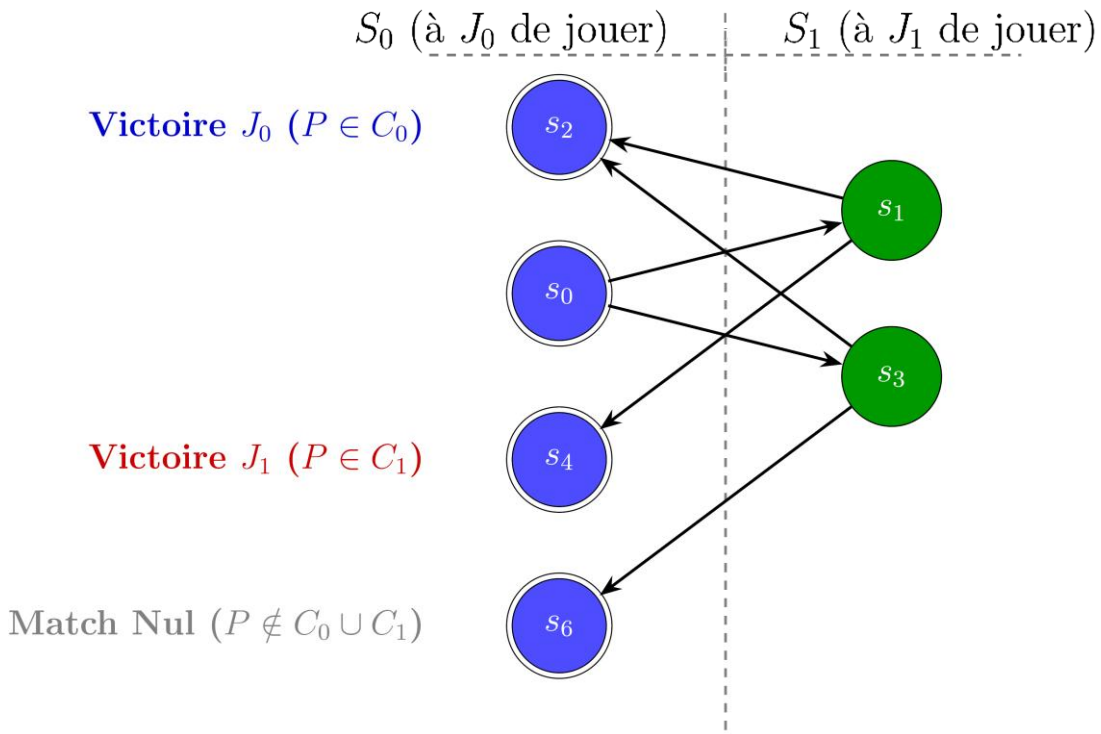


Figure 3 : Exemple de graphe à 4 parties

I.1.4. Stratégie avec mémoire et stratégie sans mémoire

Notons $\mathcal{P}_i$ l'ensemble des parties partielles dont le dernier sommet appartient à S_i. Une partie partielle s'écrit alors sous la forme $(s_0, \ldots, s_{p-1})$. Puisque $s_{p-1} \in S_i$, c'est au joueur J_i de jouer.

Une **stratégie avec mémoire** pour le joueur J_i est une application $f : \mathcal{P}_i \to S_{1-i}$ qui, à toute partie partielle $h = (s_0, \ldots, s_{p-1})$, fait correspondre un sommet s_p tel que (s_{p-1}, s_p) soit une arête de G.

On dit que le joueur J_i **suit la stratégie f** lors de la partie $P = (s_n)_{n\in\mathbb{N}}$ si pour tout $n \in \mathbb{N}$:

$$(s_n \in S_i \Rightarrow s_{n+1} = f(s_0, \ldots, s_n))$$

Conceptuellement, une stratégie avec mémoire est simplement un manuel d'instructions complet. Avant même que la partie ne commence, le joueur J_i rédige un manuel qui dit : « Si la partie arrive sur tel sommet en ayant suivi tel chemin spécifique, voilà ce que je jouerai ». Il doit y avoir une instruction pour chaque situation possible, même celles qui n'arriveront peut-être jamais lors d'une partie donnée (car cela dépend aussi des choix de l'autre joueur J_{1-i}).

Prenons l'exemple de la Figure 4 : les sommets contrôlés par J_0 sont $S_0 = \{s_0, s_1\}$ et ceux contrôlés par J_1 sont $S_1 = \{s_A, s_B, s_{Victoire}, s_{Défaite}\}$. Une fois sur s_1, le joueur J_0 a le choix entre $s_{Victoire}$ (état gagnant pour J_0) et $s_{Défaite}$ (état gagnant pour J_1) :

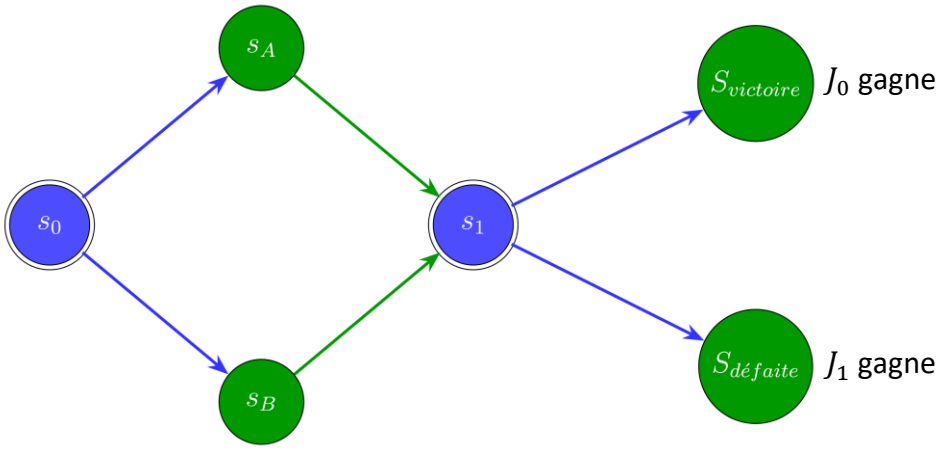


Figure 4 : Graphe d'illustration

Dans ce graphe, pour le sommet de décision s_1, il y a deux parties partielles possibles :

$$h_1 = (s_0, s_A, s_1) \text{ et } h_2 = (s_0, s_B, s_1).$$

Imaginons que J_0 veuille un comportement complexe : il décide qu'il veut aller sur $s_{Victoire}$ si le jeton est passé par s_A, mais qu'il préfère aller sur $s_{Défaite}$ si le jeton est passé par s_B. On définit alors deux stratégies avec mémoire : $f(h_1) = s_{Victoire}$ et $f(h_2) = s_{Défaite}$.

Si on définit également f telle que $f(s_0) = s_A$, alors f est une **stratégie gagnante pour J_0** : la partie jouée par le joueur J_0 sera $P = (s_0, s_A, s_1, s_{Victoire})$. Une stratégie plus robuste pour f serait de définir $f(h_1) = f(h_2) = s_{Victoire}$. Cette stratégie serait en effet gagnante même en définissant f telle que $f(s_0) = s_B$.

Dans l'exemple précédent, on remarque que $f(h_1) \neq f(h_2)$, même si les deux parties partielles se terminent sur le même sommet s_1.

Dans ce cours, on ne considèrera que des **stratégies sans mémoire**, c'est-à-dire des stratégies f telles que pour tout couple (h, h') de parties partielles ayant même sommet final, on a $f(h) = f(h')$. Concrètement, $f(h)$ ne dépend pas de la partie partielle h mais seulement de son sommet final. Une stratégie sans mémoire peut donc être vue comme une application $f : S_i \to S_{1-i}$.

Avec une stratégie sans mémoire, dans l'exemple précédent (Figure 4), pour le sommet s_1 le joueur J_0 ne peut choisir qu'une unique arête sortante. La stratégie gagnante devra donc être définie de sorte que $f(s_1) = s_{\text{Victoire}}$.

Dans le cadre des jeux sur graphes finis avec des sommets terminaux et une condition de gain qui dépend uniquement du sommet final, la mémoire est superflue : si une stratégie gagnante existe, elle ne dépend que du sommet courant, parce que le passé n'ajoute aucune information pertinente pour décider « quoi faire depuis s ».

En revanche, on est obligé d'utiliser de la mémoire dès que la condition de gain dépend de l'historique de façon non réductible au seul sommet courant (ou qu'on n'a pas l'information parfaite).

Reprenons sur la Figure 5 ci-dessous notre exemple précédent. La situation est plus simple que dans l'exemple précédent puisque depuis la position initiale, il n'existe qu'une seule façon d'atteindre s_1, à savoir la partie partielle $h_1 = (s_0, s_1)$, et une seule façon d'atteindre s_3, à savoir $h_2 = (s_0, s_3)$. Ainsi, deux parties partielles distinctes ne peuvent jamais se terminer sur un même sommet. Par conséquent, la condition $f(h) = f(h')$ est donc toujours vraie pour tout couple de parties partielles ayant même sommet final. Dès lors, une stratégie ne peut pas faire dépendre son choix du passé : connaître le sommet courant suffit entièrement pour décider du coup à jouer. Ainsi, dans cet exemple, toute stratégie est nécessairement une stratégie sans mémoire.

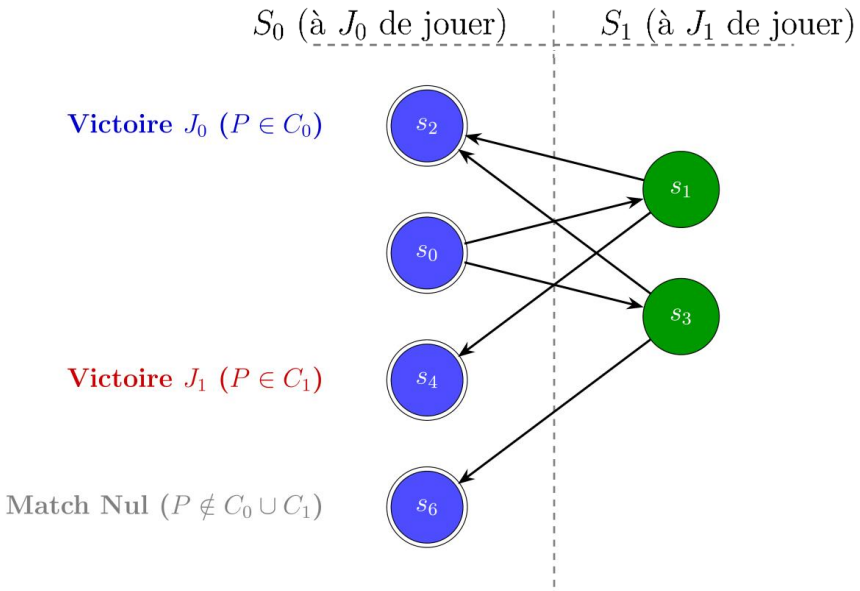


Figure 5 : Graphe d'Illustration à 4 parties

I.1.5. Stratégie gagnante et position gagnante

Nous avons vu quelques exemples de **stratégies gagnantes**. Voici maintenant une définition plus formelle :

Une stratégie f est gagnante pour le joueur J_i à partir de $s_0 \in S$ si toute partie débutant en s_0 dans laquelle J_i respecte f, est gagnée par J_i. Une **position gagnante** pour le joueur J_i est un sommet s à partir duquel il existe une stratégie gagnante pour lui.

Reprenons encore notre exemple (Figure 6 ci-dessous) :

- Pour le joueur J_0 :
  - Si J_0 choisit la stratégie $f(s_0) = s_1$, la partie peut se terminer en s_4 (défaite) si J_1 joue la stratégie $f(s_1) = s_4$. Donc J_0 ne gagne pas « toutes » les parties.
  - Si J_0 choisit la stratégie $f(s_0) = s_3$, la partie peut se terminer en s_6 (match nul) si J_1 joue la stratégie $f(s_3) = s_6$. Là non plus, J_0 ne gagne pas « toutes » les parties.
    - Puisqu'aucune stratégie ne garantit la victoire à J_0, s_0 n'est pas une position gagnante pour J_0.
    - Il n'existe pas de stratégie gagnante pour J_0 à partir de s_0.

- Pour le joueur J_1 :
  - Si la partie arrive sur s_3, J_1 ne gagnera jamais.
  - Si la partie arrive sur $s_1$et si J_1 définit la stratégie $f(s_1) = s_4$, alors J_1 gagne à tous les coups.
    - s_1 est une position gagnante pour J_1.
    - Mais il n'existe pas de stratégie gagnante pour J_1 à partir de s_0 car J_0 peut éviter s_1.

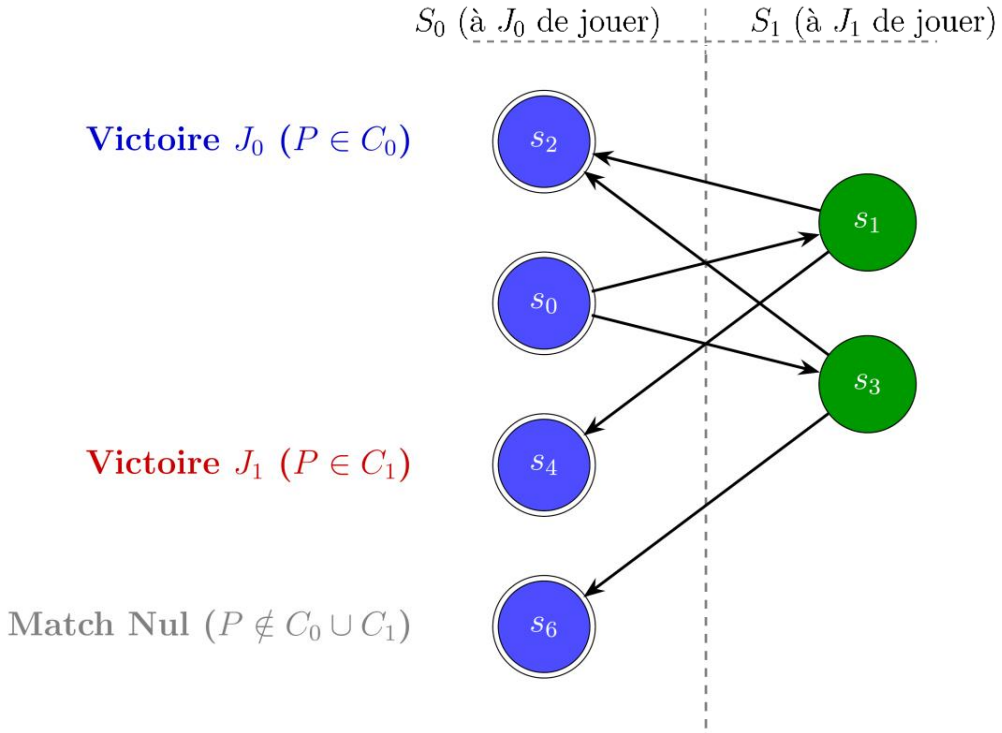


Figure 6 : Graphe d'illustration à 4 parties

I.2. Arbre d'un jeu

Pour étudier un jeu, il est souvent utile de le représenter par un arbre dans lequel la racine correspond à la situation de début du jeu et où les branches correspondent aux choix qui s'offrent au joueur qui commence la partie.

Prenons comme exemple le jeu de Nim (voir TD1) : dans ce jeu, il y a initialement n allumettes sur la table. Deux joueurs, J_0 et J_1, en retirent tour à tour 1, 2 ou 3. Le joueur qui enlève la dernière allumette perd la partie. Pour $n = 4$ allumettes, **l'arbre du jeu** est celui de la Figure 7, où les sommets doublement cerclés indiquent le tour de J_0 :

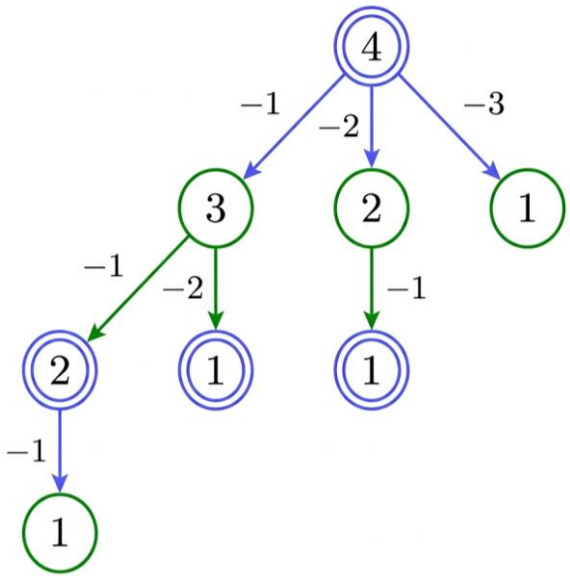


Figure 7 : Arbre du jeu de Nim (pour n = 4 allumettes)

II) JEUX D'ACCESSIBILITÉ

II.1. Définition, condition d'accessibilité

Un **jeu d'accessibilité** est un jeu tel qu'il existe $i \in \{0,1\}$ et un ensemble de sommets $F \subseteq S$ (sommets finaux), tels que C_i est l'ensemble des parties qui se terminent en un sommet de F et tels que $C_{1-i} = \mathcal{P} \setminus F$ dans les cas sans nuls.

Ainsi, dans un tel jeu, le joueur J_i souhaite atteindre l'un des sommets de F alors que son adversaire souhaite les éviter. Une condition de gain s'appelle ici une **condition d'accessibilité**.

Dans la Figure 8, le but de J_0 est de pousser le jeton dans la zone de l'ensemble des sommets finaux $F = \{f_1, f_2\}$, celui de J_1 est de dévier le jeton vers $s_{\text{piège}}$. Les conditions d'accessibilité sont $C_0 = \{(s_0, s_1, s_3, f_1), (s_0, s_1, s_3, f_2)\}$ et $C_1 = \{(s_0, s_1, s_{\text{piège}}), (s_0, s_2, s_{\text{piège}})\}$.

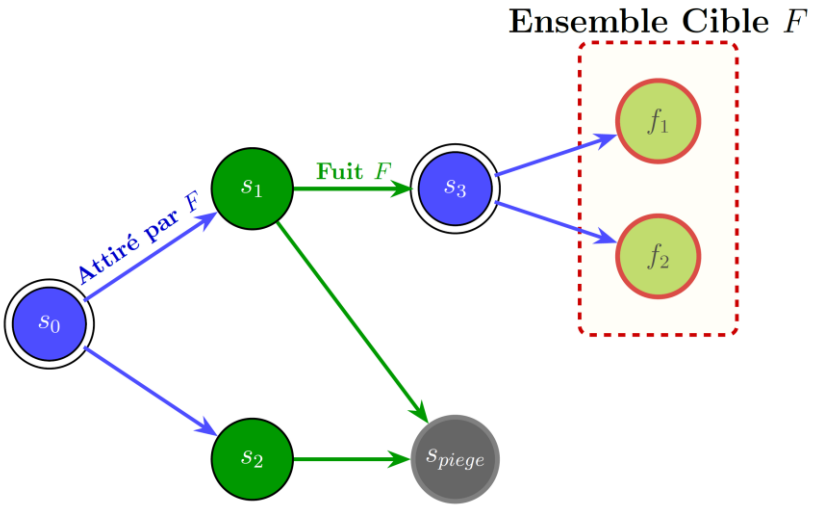


Figure 8 : Illustration d'un jeu d'accessibilité

Une condition d'accessibilité est un cas particulier d'une condition de gain (définie en I.1.3).

Dans le cas général, une condition de gain C_i est simplement un sous-ensemble arbitraire de $\mathcal{P}$ (l'ensemble de toutes les parties possibles). Cette définition est extrêmement vaste et pourrait par exemple définir des règles comme « Je gagne si la partie passe par le sommet A, puis ne touche jamais le sommet B, et finit sur un sommet de numéro pair. », ou encore « Je gagne si la partie (infinie) passe une infinité de fois par le sommet A. ».

Dans la condition d'accessibilité, on ne s'autorise plus à inventer des règles complexes. La condition d'accessibilité C_i a l'obligation d'être formulée d'une seule et unique manière : atterrir dans l'ensemble F. Un jeu d'accessibilité pour J_0 s'écrit :

$$J_0\ gagne \ \Leftrightarrow\ \exists n, s_n \in F$$

II.2. Attracteur et piège de l'ensemble des sommets finaux

Définissons une suite $\omega_n^{(i)}(F)$ avec $n \in \mathbb{N}$, comme étant l'ensemble des sommets de S à partir desquels le joueur J_i peut atteindre un sommet de F en au plus n coups.

Sur la Figure 9 en bas de page, la zone en rouge est le terme $\omega_0^{(0)}(\mathrm{F}) = \mathrm{F} = \{\mathrm{f}\}$: c'est la cible du joueur J_0.

La zone orange est le terme $\omega_1^{(0)}(\mathrm{F})$: le joueur J_0 peut gagner en au plus un coup s'il est déjà sur $\omega_0^{(0)}(\mathrm{F})$, ou si c'est à lui de jouer et qu'il peut choisir un coup qui le mène en $\omega_0^{(0)}(\mathrm{F})$ (c'est le cas ici depuis le sommet a). On a donc :

$$\omega_1^{(0)}(F) = \omega_0^{(0)}(F) \cup \{a\}$$

La zone en jaune est le terme $\omega_2^{(0)}(\mathrm{F})$: comme précédemment, le joueur J_0 peut gagner en au plus deux coups s'il est déjà sur $\omega_1^{(0)}(\mathrm{F})$ ou si c'est à lui de jouer et qu'il peut choisir un coup qui le mène en $\omega_1^{(0)}(\mathrm{F})$ (ce n'est pas le cas ici). Mais il peut également gagner si c'est le tour de son adversaire et que tous ses coups le mènent en $\omega_1^{(0)}(\mathrm{F})$ (c'est le cas ici depuis le sommet b mais pas depuis le sommet c car l'arête (c, d) ne mène pas en $\omega_1^{(0)}(\mathrm{F})$) :

$$\omega_2^{(0)}(\mathrm{F}) = \omega_1^{(0)}(\mathrm{F}) \cup \{\mathrm{b}\}$$

D'une manière générale, on peut donc définir la suite par :

- $\omega_0^{(\mathrm{i})}(\mathrm{F}) = \mathrm{F}$
- $\forall\, \mathrm{n} \in \mathbb{N},\ \omega_{\mathrm{n}+1}^{(\mathrm{i})}(\mathrm{F}) = \omega_{\mathrm{n}}^{(\mathrm{i})}(\mathrm{F}) \cup \left\{\mathrm{s} \in \mathrm{S_i} \mid \exists\, \mathrm{s}' \in \omega_{\mathrm{n}}^{(\mathrm{i})}(\mathrm{F}) \mid (\mathrm{s}, \mathrm{s}') \in \mathrm{A}\right\}$
  $\cup \left\{\mathrm{s} \in \mathrm{S_{1\text{-}i}} \mid \forall\, \mathrm{s}' \in \mathrm{S}, \left((\mathrm{s}, \mathrm{s}') \in \mathrm{A} \Rightarrow \mathrm{s}' \in \omega_{\mathrm{n}}^{(\mathrm{i})}(\mathrm{F})\right)\right\}$

Concrètement, le joueur $\mathrm{J_i}$ peut gagner en au plus n coup si et seulement s'il est déjà en $\omega_{\mathrm{n}-1}^{(\mathrm{i})}(\mathrm{F})$, ou si c'est à lui de jouer et qu'il peut choisir un coup qui le mène en $\omega_{\mathrm{n}-1}^{(\mathrm{i})}(\mathrm{F})$, ou si c'est le tour de son adversaire et que tous ses coups le mènent en $\omega_{\mathrm{n}-1}^{(\mathrm{i})}(\mathrm{F})$.

Puisque plus rien ne peut entrer dans $\omega_2^{(0)}(\mathrm{F})$, on a $\omega_3^{(0)}(\mathrm{F}) = \omega_2^{(0)}(\mathrm{F})$: la suite est stationnaire au rang $\mathrm{n}_0 = 2$. On dit que $\omega_2^{(0)}(\mathrm{F})$ est **l'attracteur de F pour le joueur $\mathbf{J_0}$**, et on le note simplement $\omega^{(0)}(\mathrm{F})$: c'est l'ensemble des sommets à partir desquels le joueur J_0 est certain d'atteindre un sommet de F, quels que soient les coups joués par J_1. C'est donc l'ensemble des positions gagnantes pour J_0.

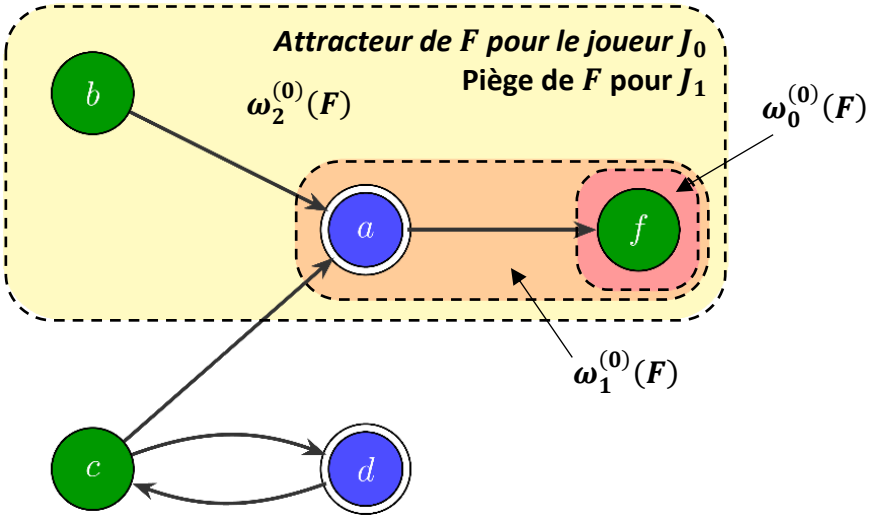


Figure 9: Évolution de la suite $\omega_n^{(0)}(F)$ et attracteur de F pour le joueur J_0

Plus formellement, la suite $\left(\omega_n^{(i)}(F)\right)_{n\in\mathbb{N}}$ étant croissante et à valeurs dans l'ensemble fini S, elle est stationnaire. Il existe ainsi $n_0 \in \mathbb{N}$ tel que pour tout $n \geq n_0$, $\omega_n^{(i)}(F) = \omega_{n_0}^{(i)}(F)$. L'ensemble $\omega_{n_0}^{(i)}(F)$, noté simplement $\omega^{(i)}(F)$, s'appelle **l'attracteur de F pour le joueur J_i** et le **piège de F pour le joueur J_{1-i}**.

La fonction **rang** est alors définie de S dans $\mathbb{N}$ par $rg(s) = \min\left\{n \in \mathbb{N} \mid s \in \omega_n^{(i)}(F)\right\}$ si $\left\{n \in \mathbb{N} \mid s \in \omega_n^{(i)}(F)\right\}$ est non vide, et $rg(s) = +\infty$ sinon. Ainsi, le rang de s est le nombre minimal de coups pour lequel le joueur J_i est sûr d'atteindre un point de F à partir de s.

II.3. Principe de l'algorithme de calcul de l'attracteur

On peut calculer explicitement l'attracteur de F pour le joueur J_i . L'algorithme procède à rebours, en partant de l'ensemble F des sommets finaux favorables au joueur J_i. Il détermine alors progressivement tous les sommets à partir desquels J_i peut forcer l'atteinte de F.

Pour être efficace, l'algorithme ne teste pas en permanence tous les sommets du graphe. Il prépare deux informations lors de son initialisation pour chaque sommet s du graphe :

- Son nombre de successeurs deg[s];
- La liste de ses prédécesseurs pred[s];

Au départ, tous les sommets de F sont déjà gagnants pour J_i, puisqu'une partie située en un tel sommet a déjà atteint l'objectif.

Prenons l'exemple du graphe de la Figure 10, dans lequel l'ensemble cible pour le joueur J_0 est défini par $F = \{f_1, f_2\}$. Les compteurs et l'attracteur $\omega^{(0)}(F)$ sont initialisés avec en particulier pour les sommets s_3 et s_4 : $\deg[s_4] = 2$ et $\deg[s_3] = 2$.

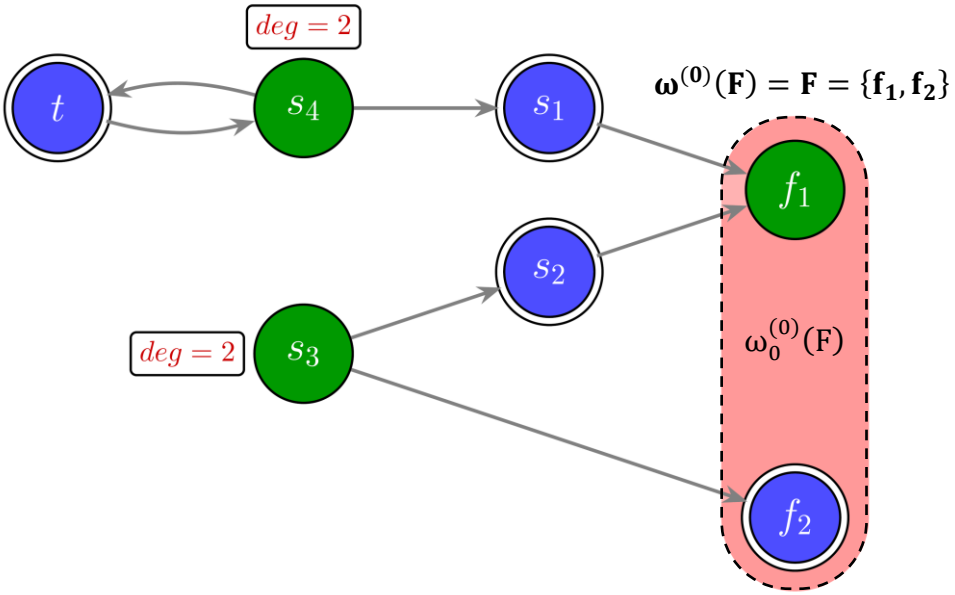


Figure 10 : Algorithme de calcul de l'attracteur (étape 1 : initialisation)

Ensuite, pour chaque sommet s appartenant à l'attracteur, on examine ses prédécesseurs $p \in pred\,[s]$:

- Si le prédécesseur p est contrôlé par le joueur J_i ($p \in S_i$) alors on ajoute ce prédécesseur à l'attracteur. En effet, si $p \in S_i$ possède au moins un successeur dans l'attracteur, il suffit pour J_i de choisir ce coup pour atterrir dans l'attracteur.
- Si en revanche le prédécesseur p est contrôlé par l'adversaire ($p \in S_{1-i}$), alors il ne peut être ajouté à l'attracteur que si tous ses successeurs y appartiennent. Pour savoir si tous les successeurs de ce prédécesseur appartiennent à l'attracteur, on décrémente tout d'abord le compteur deg[p] qui lui est associé puis si ce compteur tombe à zéro, on ajoute p à l'attracteur.

Dans l'exemple de la Figure 11, les prédécesseurs du sommet f_1 sont contrôlés par le joueur J_0. Ils sont donc ajoutés à l'attracteur. En revanche, comme l'unique prédécesseur du sommet f_2 n'appartient pas à S_0, son degré est décrémenté et passe de $deg[s_3] = 2 \rightarrow 1$:

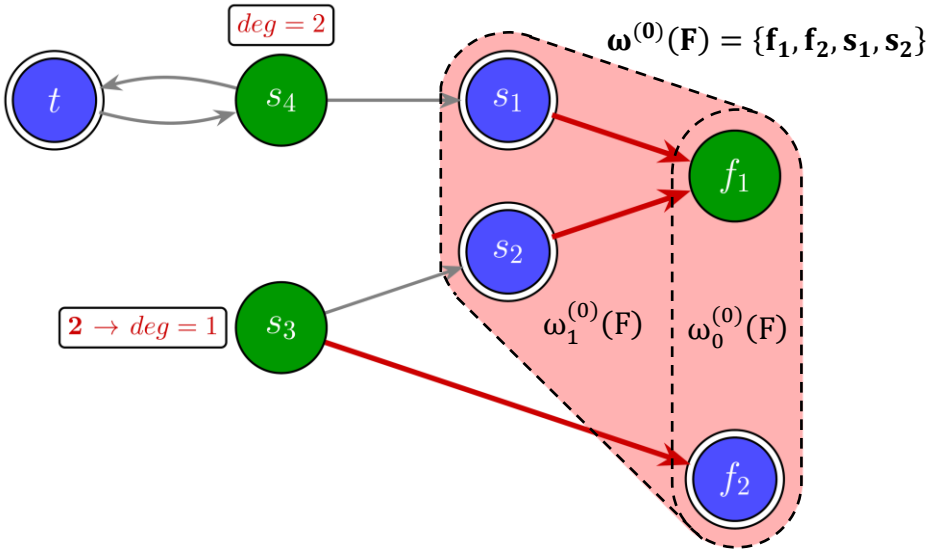


Figure 11 : Algorithme de calcul de l'attracteur (étape 2 : analyse des prédécesseurs de f_1 et f_2)

Ensuite, il faut analyser les prédécesseurs des sommets s_1 et s_2. Puisque le prédécesseur du sommet s_2 n'appartient pas à S_0, son degré est décrémenté. Puisque qu'il tombe à zéro, on ajoute ce prédécesseur à l'attracteur :

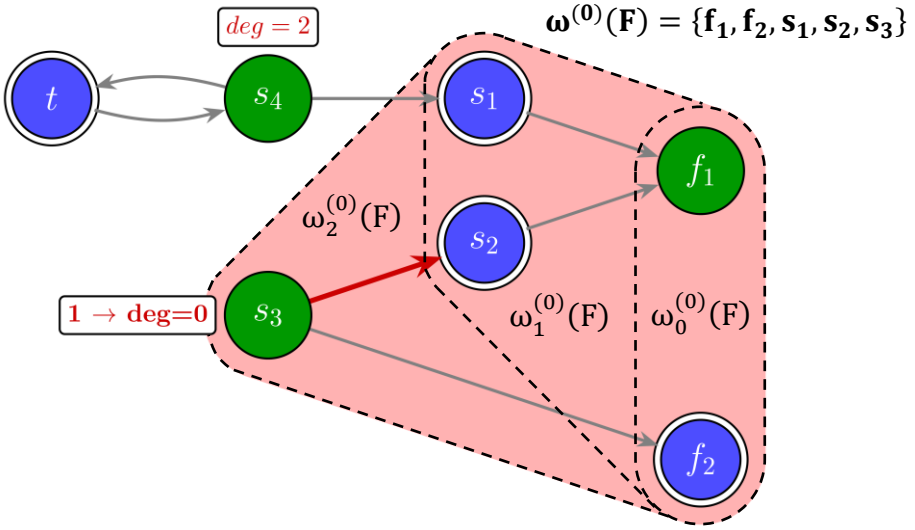


Figure 12 : Algorithme de calcul de l'attracteur (étape 3 : analyse des prédécesseurs de s_2)

Les sommets appartenant à l'attracteur qui n'ont pas été analysés sont maintenant s_1 et s_3. Puisque le sommet s_3 n'a aucun prédécesseur, ce sommet ne compte pas. Concernant l'unique prédécesseur du sommet s_1, puisqu'il est contrôlé par J_1, on décrémente son degré qui passe de $\deg[s_4] = 2 \rightarrow 1$.

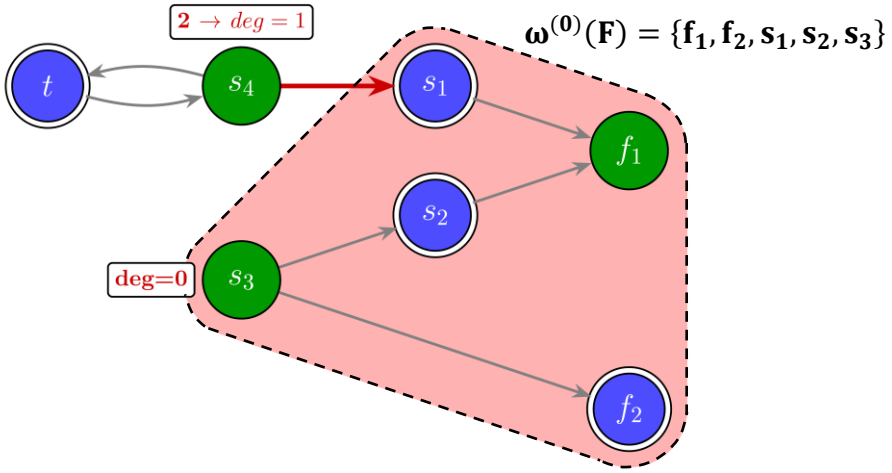


Figure 13 : Algorithme de calcul de l'attracteur (étape 4 : analyse des prédécesseurs de s_1 et s_3)

La construction de l'attracteur est alors terminée puisque tous les prédécesseurs des sommets contenus dans l'attracteur ont été analysés. En conclusion, si au cours d'une partie le joueur J_0 se retrouve en s_1 ou en s_2, ou que le joueur J_1 se retrouve en s_3, alors le joueurJ_0 est certain de gagner la partie.

II.4. Algorithme de calcul de l'attracteur

L'algorithme de calcul de l'attracteur peut être vu comme une adaptation de l'algorithme de recherche en largeur (*Breadth First Search - BFS*).

Dans un BFS classique, on part d'un sommet initial, puis on explore progressivement tous les sommets atteignables en suivant les arêtes dans leur sens direct. Une file sert à traiter les sommets dans l'ordre de leur découverte. Voici pour rappel l'algorithme du BFS :

Algorithme BFS

Entrée : G : dictionnaire d'adjacence du graphe (G[x] : ensemble des successeurs de x)
s_0 : sommet de départ
Sortie : Ω : ensemble des sommets atteignables depuis s_0

```
BFS (G, s0) :
    Ω := {s0}                    # Ensemble des sommets atteignables initialisé avec s0
    file := {s0}                 # File d'attente initialisée avec le sommet s0
    Tant que file est non vide :
        s := défiler(file)

        Pour tout t ∈ G[s] :
            Si t ∉ Ω :
                Ω := Ω ∪ {t}     # Dès qu'un nouveau sommet est découvert, il est ajouté
                Enfiler t dans la file
    Retourner Ω
```

Dans l'algorithme de calcul de l'attracteur $\omega^{(i)}(F)$, on procède au contraire à rebours : on part de l'ensemble cible F, puis on remonte le graphe en examinant les prédécesseurs des sommets déjà reconnus comme appartenant à l'attracteur.

Cependant, contrairement au BFS classique, un prédécesseur d'un sommet déjà reconnu comme appartenant à l'attracteur est ajouté seulement s'il vérifie une condition correspondant aux règles du jeu :

- Si le sommet appartient au joueur J_i, il suffit qu'il possède un successeur déjà dans l'attracteur ;
- S'il appartient à l'adversaire J_{1-i}, il faut que tous ses successeurs soient déjà dans l'attracteur (ce qui est le cas si son degré est nul après avoir été décrémenté).

Ainsi, l'algorithme de l'attracteur est une exploration en largeur rétrograde et sélective : la file joue le même rôle d'organisation que dans le BFS, mais l'entrée d'un sommet dans l'ensemble découvert dépend ici non seulement de l'existence d'une arête, mais aussi de la structure du jeu.

Voici donc les modifications à faire pour passer du BFS au calcul de l'attracteur :

- Changer de point de départ : puisqu'on part de l'ensemble F et non du sommet de départ s_0, la file initiale et l'ensemble $\omega^{(i)}(F)$ contiennent tous les sommets de F.
- Parcourir les prédécesseurs et non les successeurs : on veut savoir quels sommets peuvent mener à un sommet déjà dans $\omega^{(i)}(F)$. Il faut donc disposer, pour chaque sommet s, l'ensemble de ses prédécesseurs $\mathrm{pred}[s]$.
- Ne pas ajouter automatiquement un sommet découvert : un prédécesseur p n'est ajouté que s'il devient gagnant pour J_i.
- Introduire un compteur pour les sommets adverses : pour les sommets de l'adversaire J_{1-i}, on maintient un compteur $\deg[p]$ égal au nombre de successeurs de p qui ne sont pas encore connus dans l'attracteur. Chaque fois qu'un successeur de p entre dans $\omega^{(i)}(F)$, on décrémente ce compteur.

L'algorithme final, rédigé dans l'esprit d'un BFS modifié est donné en page suivante. Une autre version utilisant une fonction auxiliaire est également donnée ensuite. Dans la version avec file d'attente, la propagation est faite en largeur et dans la version avec une fonction auxiliaire, elle est faite en profondeur.

II.5. Complexité de l'algorithme de calcul de l'attracteur

Dans les deux algorithmes, le pré-calcul des prédécesseurs et des degrés sortants est en $O(|S| + |A|)$.

Dans le cas de l'algorithme avec une fonction auxiliaire, le nombre d'appels non triviaux à la fonction ETENDRE est majoré par $|S|$ (chaque sommet est ajouté à Ω au plus une fois). De plus, la boucle intérieure sur pred[s] traite chaque arête (p, s) au plus une fois, ce qui donne un total de $O(|A|)$ opérations élémentaires. La complexité totale est donc de $O(|S| + |A|)$.

La complexité totale est la même dans le cas de l'algorithme avec utilisation d'une file. En effet, chaque sommet est enfilé ou défilé au plus une fois (le test $s \notin \Omega$ l'assure). La boucle interne sur pred[s] traite chaque arête (p, s) au plus une fois. La complexité totale est donc $O(|S| + |A|)$.

Algorithme de calcul de l'attracteur (avec une file – propagation en largeur)

Entrée : G : dictionnaire du graphe $G(S, A)$ (G[x] : ensemble des successeurs de x)
S_i : Ensemble des sommets contrôlés par J_i
F : Sommets cibles pour le joueur J_i
Sortie : Ω : L'attracteur de F pour le joueur J_i

```
ATTRACTEUR (G, Si, F) :
    Ω := F          # Positions gagnantes trouvées pour le joueur Ji
    file := F       # File d'attente initialisée avec les sommets de F

    # Initialisation : calcul des prédécesseurs et des degrés sortants de chaque sommet
    Pour tout s ∈ S :
    |   deg[s] := 0              # Degré sortant de s
    |   pred[s] := ∅             # Liste des prédécesseurs de s
    Pour tout s ∈ S :
    |   # Pour chaque successeur s+ du sommet courant s
    |   Pour tout s+ ∈ G[s] :
    |   |   deg[s] := deg[s] + 1            # Incrémente le degré de s
    |   |   pred[s+] := pred[s+] ∪ {s}      # Ajoute s aux prédécesseurs du sommet s+

    # Boucle principale : propagation rétrograde de l'attracteur
    Tant que file est non vide :
    |   # Récupère le sommet dans la file
    |   s := défiler(file)
    |
    |   # Analyse des prédécesseurs de s
    |   Pour tout p ∈ pred[s] :
    |   |   # Si le prédécesseur n'est pas dans l'attracteur
    |   |   Si p ∉ Ω :
    |   |   |   # et s'il est contrôlé par le joueur Si
    |   |   |   Si p ∈ Si :
    |   |   |   |   Ω := Ω ∪ {p}       # Alors on l'ajoute à l'attracteur
    |   |   |   |   Enfiler(p)         # et à la liste des prochains sommets à analyser
    |   |   |   # Sinon, si c'est un prédécesseur contrôlé par l'adversaire
    |   |   |   Sinon :
    |   |   |   |   # On décrémente son degré
    |   |   |   |   deg[p] := deg[p] - 1
    |   |   |   |   # Si son degré tombe à 0, on l'ajoute à l'attracteur et à la file
    |   |   |   |   Si deg[p] == 0 :
    |   |   |   |   |   Ω := Ω ∪ {p}
    |   |   |   |   |   Enfiler(p)
    Retourner Ω
```

L'autre algorithme ci-dessous repose sur le même principe que la version utilisant une file : on part des sommets cibles F, puis on propage l'attracteur à rebours en examinant les prédécesseurs des sommets déjà reconnus comme favorables au joueur J_i. La différence essentielle tient à la manière de parcourir le graphe.

Ici, cette file est remplacée par une fonction récursive ÉTENDRE, qui propage immédiatement l'information vers les prédécesseurs admissibles : on obtient ainsi une exploration en profondeur.

Algorithme de l'attracteur (avec une fonction auxiliaire – propagation en profondeur)

Entrée : G : dictionnaire du graphe $G(S, A)$ (G[x] : ensemble des successeurs de x)
S_i : Ensemble des sommets contrôlés par J_i
F : Sommets cibles pour le joueur J_i

Sortie : Ω : L'attracteur de F pour le joueur J_i

ATTRACTEUR (G, S_i, F) :

```
Ω := F          # Positions gagnantes trouvées pour le joueur J_i

# Initialisation : calcul des prédécesseurs et des degrés sortants de chaque sommet
Pour tout s ∈ S :
    deg[s] := 0                 # Degré sortant de s
    pred[s] := ∅                # Liste des prédécesseurs de s
Pour tout s ∈ S :
    # Pour chaque successeur s+ du sommet courant s
    Pour tout s+ ∈ G[s] :
        deg[s] := deg[s] + 1            # Incrémente le degré de s
        pred[s+] := pred[s+] ∪ {s}      # Ajoute s aux prédécesseurs du sommet s+

# Procédure auxiliaire : étendre l'attracteur depuis un sommet s
ÉTENDRE(s) :
    Pour tout p ∈ pred[s] :             # Analyse des prédécesseurs de s
        Si p ∉ Ω :                      # Si le prédécesseur n'est pas dans l'attracteur
            Si p ∈ S_i :                # et s'il est contrôlé par le joueur S_i
                Ω := Ω ∪ {p}            # Alors on l'ajoute à l'attracteur
                ETENDRE(p)              # et on étend l'attracteur à partir de p
            # Sinon, si c'est un prédécesseur contrôlé par l'adversaire
            Sinon :
                deg[p] := deg[p] - 1        # On décrémente son degré
                Si deg[p] == 0 :            # Si son degré tombe à 0
                    Ω := Ω ∪ {p}            # On l'ajoute à l'attracteur
                    ETENDRE(p)              # et on étend l'attracteur
# Calcul de l'attracteur : initialisation depuis les sommets finaux
Pour tout s ∈ F :
    ÉTENDRE(s)
Retourner(Ω)
```

III) CONSTRUCTION D'UNE STRATÉGIE GAGNANTE PAR ATTRACTEURS

Pour rappel, le rang $\mathrm{rg}(s)$ d'un sommet est le nombre minimal de coups garantissant au joueur J_i d'atteindre un sommet dans la cible F. La Figure 14 illustre la propriété suivante.

À l'intérieur de l'attracteur de F pour le joueur J_i, c'est-à-dire pour tout $s \in \omega^{(i)}(F) \setminus F$, le terrain est en « pente descendante » vers F :

- Si $s \in S_i$ (c'est à J_i de choisir), alors s admet un successeur $s' \in \omega^{(i)}(F)$ tel que $\mathrm{rg}(s') < \mathrm{rg}(s)$: il existe au moins un choix pour le joueur J_i qui fait « baisser » le rang et le rapproche de la victoire ;
- Si $s \in S_{1-i}$ (c'est à J_{1-i} de choisir), alors pour tout successeur s' de s, $s' \in \omega^{(i)}(F)$ et $\mathrm{rg}(s') < \mathrm{rg}(s)$: tous les choix de J_{1-i} font « baisser » le rang et rapprochent J_i de la victoire.

À l'extérieur de l'attracteur, pour tout $s \in S \setminus \omega^{(i)}(F)$, le terrain est de rang infini :

- Si $s \in S_i$ (c'est à J_i de choisir), alors pour tout successeur s' de s, on a $s' \notin \omega^{(i)}(F)$ et $\mathrm{rg}(s') = +\infty$: tous les choix que prend le joueur J_i le maintiennent hors de l'attracteur ;
- Si $s \in S_{1-i}$ (c'est à J_{1-i} de choisir), alors s admet au moins un successeur $s' \notin \omega^{(i)}(F)$ tel que $\mathrm{rg}(s') = +\infty$: le joueur J_{1-i} possède au moins un choix pour maintenir J_i hors de l'attracteur et l'empêcher de gagner.

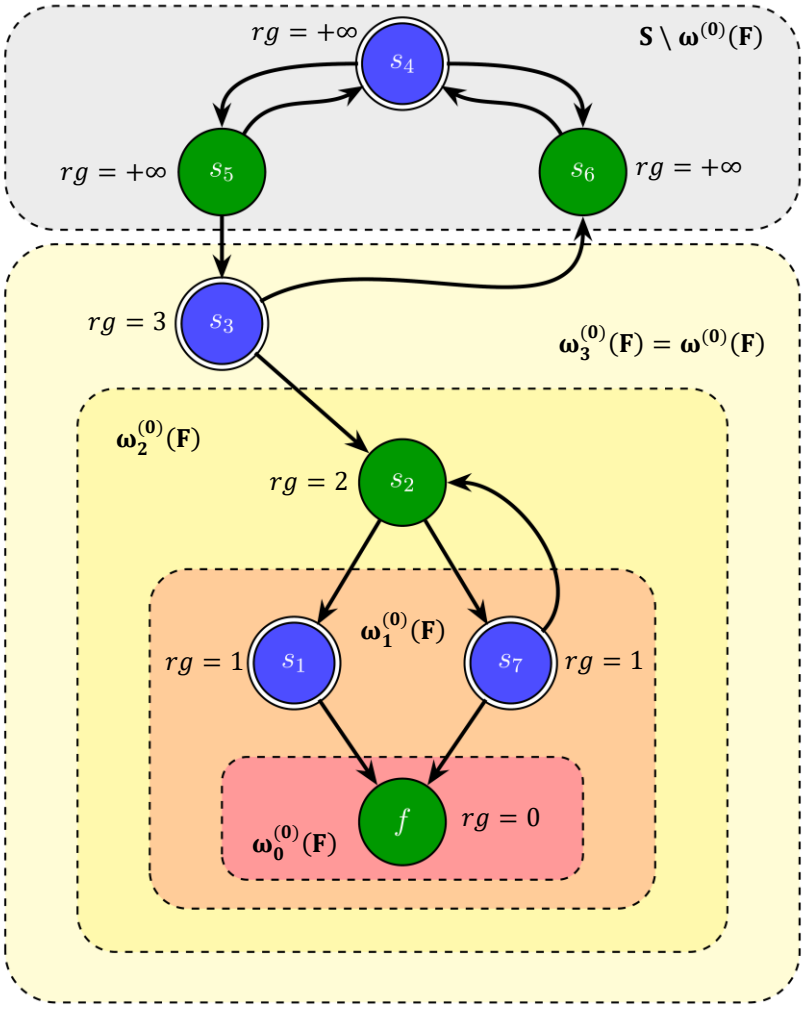


Figure 14 : Décroissance stricte du rang et mécanique d'une stratégie gagnante

III.1. Jeux sans nuls

La **construction d'une stratégie gagnante par attracteurs** répond à une question qualitative : « depuis un état courant, quel joueur peut forcer la victoire ? ». Elle découpe tout l'espace des états en régions bien définies : gagnante pour J_i ou gagnante pour J_{1-i}.

Construction d'une stratégie gagnante $f : S \to S$ pour J_i

Pour tout $s \in \omega^{(i)}(F)$:

- Si $s \in S_i$ (c'est à J_i de choisir), on pose $f(s) = s'$ tel que $rg(s') < rg(s)$: on choisit le successeur s' qui fait strictement baisser le rang ;
- Si $s \in S_{1-i}$ (c'est à J_{1-i} de choisir), on pose $f(s) = s'$ où s' est un successeur quelconque de s : la propriété précédente garantit que quel que soit le choix de l'adversaire, le rang va baisser de toute façon.

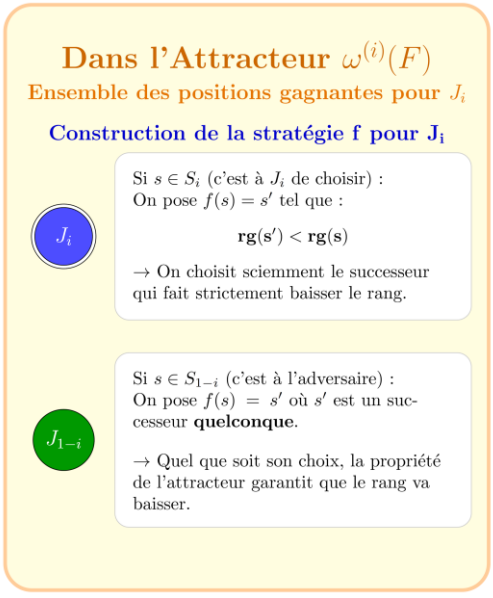
Construction d'une stratégie gagnante $f' : S \to S$ pour J_{1-i}

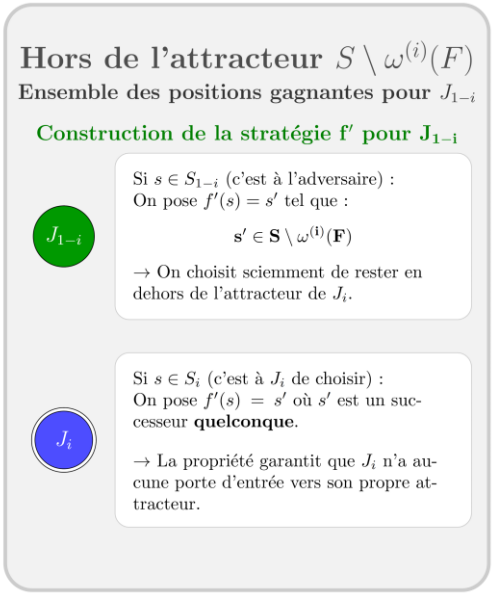
Pour tout $s \in S \setminus \omega^{(i)}(F)$:

- Si $s \in S_{1-i}$ (c'est à J_{1-i} de choisir), on pose $f'(s) = s'$ où s' est un successeur de s dans $S \setminus \omega^{(i)}(F)$: on choisit sciemment le successeur s' qui reste en dehors de l'attracteur de F pour le joueur J_i.
- Si $s \in S_i$ (c'est à J_i de choisir) on pose $f'(s) = s'$ où s' est un successeur quelconque de s : la propriété précédente garantit que J_i n'a aucune porte d'entrée vers son propre attracteur ;

Le graphe se divise donc en deux territoires parfaitement étanches :

- L'ensemble des positions gagnantes pour J_i est $\omega^{(i)}(F)$.
- L'ensemble des positions gagnantes pour J_{1-i} est $S \setminus \omega^{(i)}(F)$.





Dans son attracteur, un joueur choisit un successeur qui y reste, et si l'on a défini un rang, il peut même choisir un successeur de rang strictement plus petit pour garantir une victoire en temps fini. Si c'est à l'adversaire de jouer, n'importe lequel de ses coups conserve malgré tout la propriété. Hors de l'attracteur, la situation s'inverse : si c'est à J_{1-i} de jouer, il choisit un successeur qui reste hors de l'attracteur ; si c'est à J_1 de jouer, quel que soit son choix, il ne peut pas rejoindre son attracteur.

III.2. Jeux avec nuls

Dans cette section, on se place dans le cadre suivant : le graphe de jeu est fini, toute partie maximale se termine sur un sommet terminal, et chaque sommet terminal est étiqueté soit « victoire de J_i », soit « victoire de $J_{1\text{-}i}$ », soit « nul ».

On note :

- F_i l'ensemble des sommets terminaux gagnants pour J_i ;
- $F_{1\text{-}i}$ l'ensemble des sommets terminaux gagnants pour $J_{1\text{-}i}$;
- F_N l'ensemble des sommets terminaux nuls.

On définit alors :

$$A_i = \omega^{(i)}(F_i) \quad \text{et} \quad A_{1\text{-}i} = \omega^{(1\text{-}i)}(F_{1\text{-}i})$$

Par définition de l'attracteur :

- tout sommet de A_i est une position gagnante pour J_i ;
- tout sommet de $A_{1\text{-}i}$ est une position gagnante pour $J_{1\text{-}i}$.

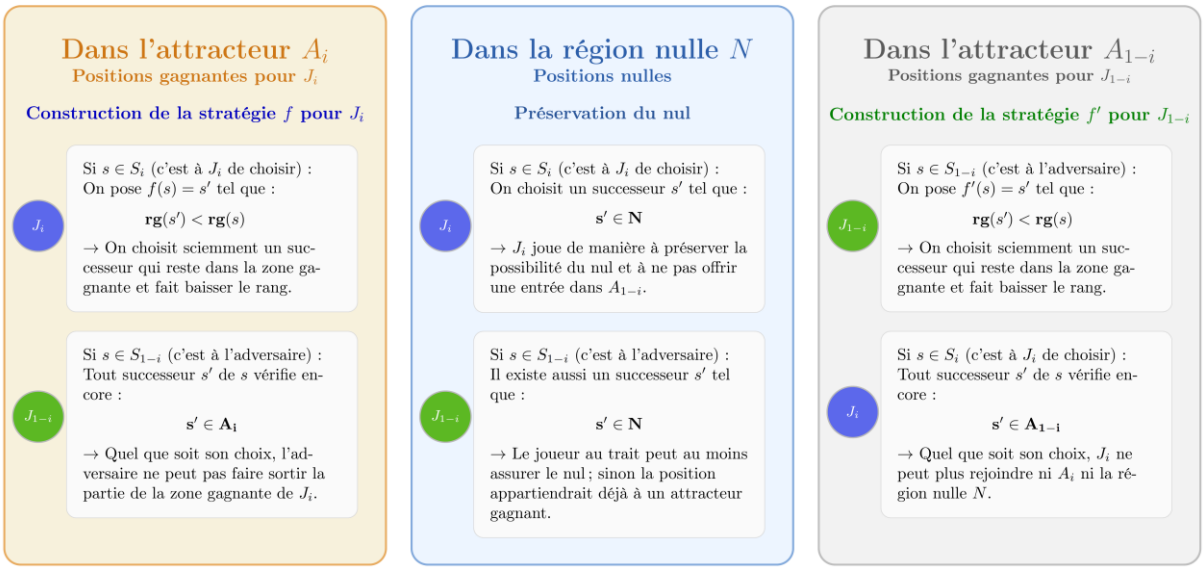
Les sommets qui n'appartiennent ni à A_i ni à $A_{1\text{-}i}$ sont donc des positions depuis lesquelles aucun des deux joueurs ne peut forcer, à lui seul, l'atteinte de son propre ensemble terminal gagnant. Dans ce cadre, on appelle **région nulle** l'ensemble :

$$N = S \setminus (A_i \cup A_{1-i})$$

Ainsi, le graphe se divise donc en trois territoires parfaitement étanches :

- L'ensemble des positions gagnantes pour J_i est A_i (attracteur de J_i) ;
- L'ensemble des positions gagnantes pour J_{1-i} est A_{1-i} (attracteur de J_{1-i}) ;
- L'ensemble des positions nulles est $N = S \setminus (A_i \cup A_{1-i})$

Dans ce jeu avec nuls, la stratégie optimale suit la règle de priorité $A_i \succ N \succ A_{1-i}$: on cherche d'abord un coup qui mène dans sa région gagnante ; à défaut, on choisit un coup qui préserve le nul ; et ce n'est qu'en dernier ressort que l'on subit une position perdante.



IV) ALGORITHME DU MINIMAX

On considère dans ce paragraphe des jeux à somme nulle (tout gain d'un joueur est exactement la perte de l'autre) et à information parfaite (chaque joueur a, à tout moment, une vision complète de l'état du jeu – ce qui n'est pas le cas général dans les jeux de cartes).

L'algorithme du minimax (algorithme du min-max) permet de sélectionner une suite de meilleurs successeurs allant de la racine de l'arbre de jeu à une feuille. Il peut servir à déterminer l'existence d'une stratégie gagnante pour un joueur, mais aussi à jouer selon cette stratégie.

IV.1. Principe de l'algorithme

Le joueur qui commence une partie (profondeur 0 de l'arbre – niveau pair) choisit la branche de l'arbre qui lui assure un gain maximal. Quant à son adversaire (profondeur 1 – niveau impair), il cherchera, pour minimiser sa perte, à minimiser le gain maximum du premier. Les joueurs jouant alternativement, les branches de valeur maximale sont à rechercher aux niveaux de profondeur paire, celles de valeur minimale aux niveaux de profondeur impaire :

- Niveaux pairs (0, 2, 4, …) : tour du joueur initial → Recherche du MAX.
- Niveau impairs (1, 3, 5, …) : tour de l'adversaire → Recherche du MIN.

Pour pouvoir s'exécuter, l'algorithme n'a besoin que de trois éléments concrets :

- L'arbre de jeu : la structure qui contient tous les états possibles et les coups permis ;
- L'identifiant du joueur initial : pour savoir de quel point de vue on calcule les scores (qui cherche le MAX et qui cherche le MIN) ;
- Une fonction d'évaluation : une règle qui ne s'applique qu'à la toute fin du jeu (sur les « feuilles » de l'arbre, les sommets finaux). Elle dit simplement qui a gagné la partie une fois qu'elle est terminée (par exemple : 1 si J_i gagne, -1 s'il perd, 0 pour un nul).

La Figure 15 illustre le principe de l'algorithme minimax :

- La descente : l'algorithme effectue un parcours en profondeur de l'arbre en explorant tous les coups possibles, sans avoir aucune idée de la stratégie à adopter, jusqu'à arriver à des positions où la partie est terminée (les feuilles).
- L'évaluation : avec la fonction d'évaluation, il étiquette les feuilles avec 1, -1 ou 0.
- La remontée : pour évaluer les sommets situés au-dessus, il regarde les valeurs des coups suivants en dessous de lui, et prend le MAX (si c'est le tour de J_i), ou le MIN (si c'est le tour de l'adversaire). Il fait ainsi remonter les valeurs jusqu'à la racine en alternant les calculs MAX et MIN à chaque étape.

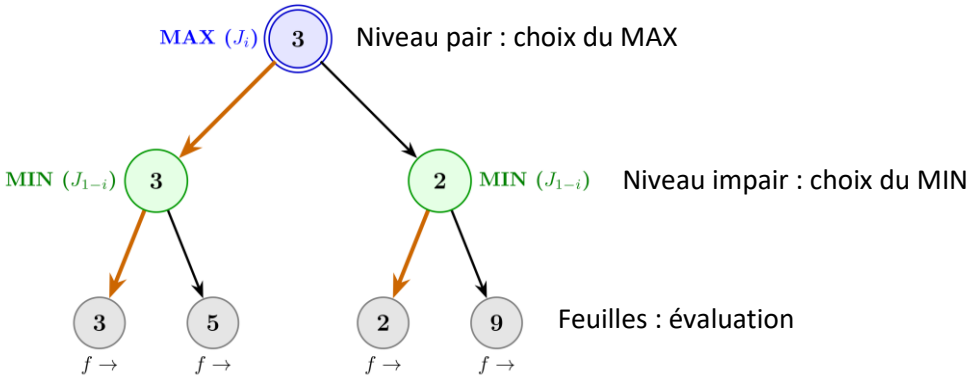


Figure 15 : Principe de l'algorithme Minimax

IV.2. Algorithme et complexité

Dans la version récursive naïve, chaque fois qu'un sommet est atteint par plusieurs chemins différents, toute la sous-exploration située sous ce sommet est refaite intégralement.

Algorithme MINIMAX

Entrée : G : dictionnaire du graphe $G(S, A)$ (G[x] : ensemble des successeurs de x)
i : Indice su joueur J_i
f : Fonction d'évaluation des sommets terminaux (feuilles)
s : Sommet actuel (racine de l'exploration)
Sortie : $v(s)$: Valeur minimax du sommet s, du point de vue du joueur J_i

MINIMAX (G, i, f, s) :
Si s est un état final (feuille), on évalue la feuille
Si G[s] == $\emptyset$:
  Retourner f(s)
Si c'est au joueur J_i de jouer, il maximise
Si $s \in S_i$:
  Retourner $\max_{s^+ \in G[s]} \{\text{MINIMAX}(G, i, f, s^+)\}$
Si c'est au joueur J_{1-i} de jouer, il cherche à minimiser
Sinon :
  Retourner $\min_{s^+ \in G[s]} \{\text{MINIMAX}(G, i, f, s^+)\}$

Pour évaluer un sommet de hauteur h, l'algorithme doit évaluer ses p successeurs, chacun étant racine d'un sous-arbre de hauteur $h-1$. Cela représente un coût total $pC(h-1)$. Une fois ces p valeurs calculées, il faut encore déterminer leur maximum ou leur minimum, ce qui ajoute un coût proportionnel à p. Ainsi, la complexité de l'algorithme $C(h)$ vérifie la formule de récurrence $C(h) = pC(h-1) + p$, avec comme condition initiale $C(0) = 1$ (si l'évaluation d'une feuille est supposée constante).

La récurrence est :

$$\begin{aligned} C(h) &= pC(h-1) + p = p(pC(h-2) + p) + p = p^2C(h-2) + p^2 + p \\ &= p^2(pC(h-3) + p) + p^2 + p = p^3C(h-3) + p^3 + p^2 + p \\ &= p^hC(0) + \sum_{k=1}^{h} p^k = p^h + p\frac{p^h - 1}{p-1} \\ &= \left(1 + \frac{p}{p-1}\right)p^h - \frac{p}{p-1} \end{aligned}$$

$C(h)$ est de la forme $Ap^h - B$, avec A et B constants. Il existe donc une constante $K > 0$ telle que, pour tout h, $C(h) \leq Kp^h$. La complexité est donc en $O(p^h)$.

Par exemple, pour un arbre de hauteur 3, où à chaque étape, 3 coups (successeurs) sont possibles, la complexité vaut 27 comparaisons. Notons qu'elle croît très vite : elle est de l'ordre de 3.5 milliards de comparaisons pour une hauteur de 20.

Une version avec mémorisation consiste à stocker, après le premier calcul, la valeur minimax d'un sommet s. Si l'on rencontre de nouveau ce sommet plus tard, on ne relance pas toute la récursion : on lit simplement la valeur déjà calculée. Chaque sommet est donc évalué au plus une fois, puis chaque arête est examinée un nombre borné de fois. Dans un graphe orienté acyclique, on tombe alors sur une complexité de l'ordre de $O(|S| + |A|)$ si l'évaluation terminale est en temps constant.

Algorithme MINIMAX (avec mémorisation)

MINIMAX (G, i, f, s) :
- Si s ∈ memo :
  - Retourner memo[s]
- Si G[s] == ∅ :
  - memo[s] := f(s)
  - Retourner memo[s]
- Si s ∈ S_i :
  - memo[s] := $\max_{s^+ \in G[s]} \{\text{MINIMAX}(G, i, f, s^+)\}$
- Sinon :
  - memo[s] := $\min_{s^+ \in G[s]} \{\text{MINIMAX}(G, i, f, s^+)\}$
- Retourner memo[s]

IV.3. Exemple

Dans cet exemple, J_0 commence la partie (nœuds doublement cerclés) et cherche à maximiser son gain. L'adversaire cherche à minimiser ce gain.

Les valeurs des feuilles tout en bas de l'arbre (au niveau 3) ayant été évaluées, les valeurs maximales sont remontées au niveau 2. Les valeurs minimales du niveau 2 sont ensuite remontées au niveau 1. Enfin, la racine (niveau 0) prend la valeur maximale du niveau 1.

La stratégie gagnante pour J_0 consiste alors à jouer les coups MAX à chaque sommet sur lequel il se trouve, et celle du joueur J_1 à jouer les coups MIN.

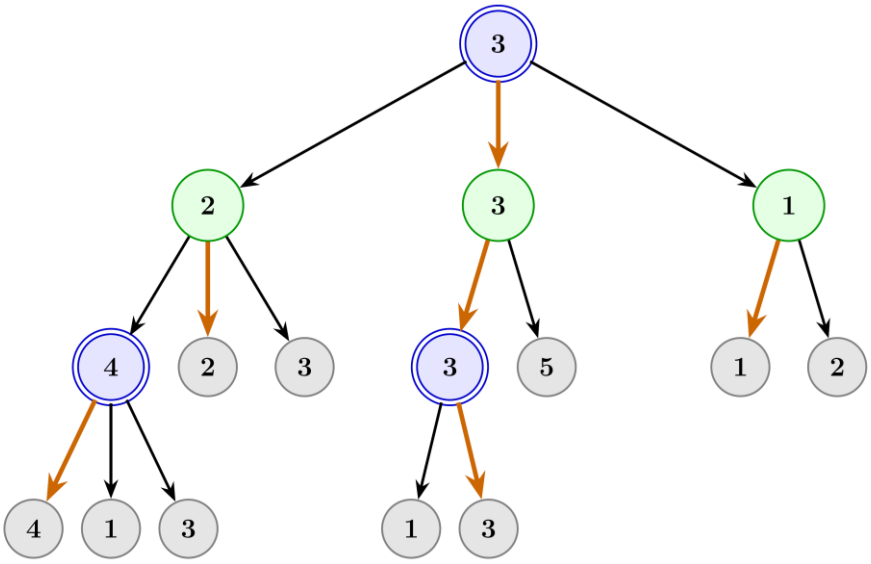


Figure 16 : Exemple Minimax

V) ALGORITHME DU MINIMAX AVEC HEURISTIQUE

Quand l'arbre de jeu a une faible profondeur, il est tout à fait possible d'appliquer l'algorithme du minimax en descendant jusqu'aux feuilles, que l'on peut évaluer, puis en faisant remonter l'information jusqu'à la racine.

Dans le cas contraire, l'arbre étant trop grand pour être parcouru entièrement, on utilise une **heuristique** qui attribue une valeur aux nœuds sans avoir à descendre jusqu'aux feuilles, mais en se limitant à une profondeur fixée à l'avance. La valeur attribuée est d'autant plus grande que le nœud correspond à une situation favorable pour le joueur considéré.

On obtient l'algorithme MINIMAX_HEUR en remplaçant la fonction d'évaluation f de l'algorithme MINIMAX par une fonction heuristique h. La variable p représente la profondeur maximale d'exploration de l'arbre. Un sommet s_1 de l'arbre est meilleur pour le joueur $\mathrm{J_j}$ qu'un sommet s_2 si $\mathrm{h}(\mathrm{s}_1) > \mathrm{h}(\mathrm{s}_2)$.

Algorithme MINIMAX avec heuristique

Entrée : G : dictionnaire du graphe $\mathrm{G}(\mathrm{S}, \mathrm{A})$ (G[x] : ensemble des successeurs de x)
i : Indice su joueur $\mathrm{J_i}$
h: Fonction d'évaluation heuristique
s : Sommet actuel (racine de l'exploration)
p : Profondeur maximale d'exploration de l'arbre
Sortie : $\mathrm{v}(\mathrm{s})$: Valeur minimax du sommet s, du point de vue du joueur $\mathrm{J_i}$

MINIMAX_HEUR (G, i, h, s, p) :
Si s est un état final (feuille), ou p == 0
Si G[s] == $\emptyset$ ou p == 0:
  Retourner h(s)
Si c'est au joueur $\mathrm{J_i}$ de jouer, il maximise
Si $\mathrm{s} \in \mathrm{S_i}$:
  Retourner $\max_{\mathrm{s}^+ \in \mathrm{G[s]}} \{\mathrm{MINIMAX\_HEUR}(\mathrm{G}, \mathrm{i}, \mathrm{h}, \mathrm{s}^+, \mathrm{p}-1)\}$
Si c'est au joueur $\mathrm{J_{1-i}}$ de jouer, il cherche à minimiser
Sinon :
  Retourner $\min_{\mathrm{s}^+ \in \mathrm{G[s]}} \{\mathrm{MINIMAX\_HEUR}(\mathrm{G}, \mathrm{i}, \mathrm{h}, \mathrm{s}^+, \mathrm{p}-1)\}$

Cela revient à définir récursivement la valeur minimax tronquée à profondeur p par :

$$V_p(s) = \begin{cases} h(s) & si\ Succ(s) = \emptyset\ ou\ p = 0 \\ \max\limits_{s' \in Succ(s)} V_{p-1}(s') & si\ s \in S_i \\ \min\limits_{s' \in Succ(s)} V_{p-1}(s') & si\ s \in S_{1-i} \end{cases}$$