Intelligence artificielle et jeux

Plan

I	**Algorithme d'intelligence artificielle (IA)**	**2**
A	Présentation générale	2
B	Apprentissage supervisé	2
C	Apprentissage non supervisé	5
II	**Théorie des jeux**	**7**
A	Contexte	7
B	Attracteurs	8
C	Algorithme minimax	10

Éléments de cours

- Intelligence artificielle ☐
- Apprentissage automatique (Machine learning) ☐
- Apprentissage supervisé (Supervised learning) ☐
- ← Exemple ☐
- Algorithmes de classification / de régression ☐
- Matrice de confusion ☐
- Apprentissage non supervisé (Unsupervised learning) ☐
- ← Exemple ☐
- Jeu et graphe ☐
- Graphe biparti ☐
- Stratégie gagnante ☐
- Position gagnante ☐
- Méthode des attracteurs ☐
- Fonction heuristique (ou fonction d'évaluation) ☐
- Algorithme minimax ☐

Capacités exigibles

- Algorithme des k plus proches voisins avec distance euclidienne. Matrice de confusion. Lien avec l'apprentissage supervisé.
- Algorithme des k-moyennes. Lien avec l'apprentissage non-supervisé. On observe des convergences vers des minima locaux.
- Jeux d'accessibilité à deux joueurs sur un graphe. Graphe biparti. Stratégie gagnante. Position gagnante.
- Détermination des positions gagnantes par le calcul des attracteurs. Construction de stratégies gagnantes.
- Notion d'heuristique. Algorithme min-max avec une heuristique.

I Algorithme d'intelligence artificielle (IA)

A Présentation générale

Définition : Intelligence artificielle

La notion d'**intelligence artificielle** désigne les techniques informatiques visant à imiter l'intelligence humaine.

Définition : Apprentissage automatique (Machine learning)

Parmi ces algorithmes, un des domaines principal est l'**apprentissage automatique**. Il s'agit pour la machine d'apprendre par elle-même à résoudre un problème, sans avoir à programmer explicitement son fonctionnement.

B Apprentissage supervisé

Définition : Apprentissage supervisé (Supervised learning)

Dans un programme d'**apprentissage supervisé**, on entraîne la machine sur de nombreux exemples d'entrées et de sorties fournies par l'opérateur.

Exemple

Algorithme des k plus proches voisins :

Les iris sont des fleurs dont on peut distinguer au moins trois espèces : les iris setosa, les iris versicolores et les iris de Virginie.

Iris setosa

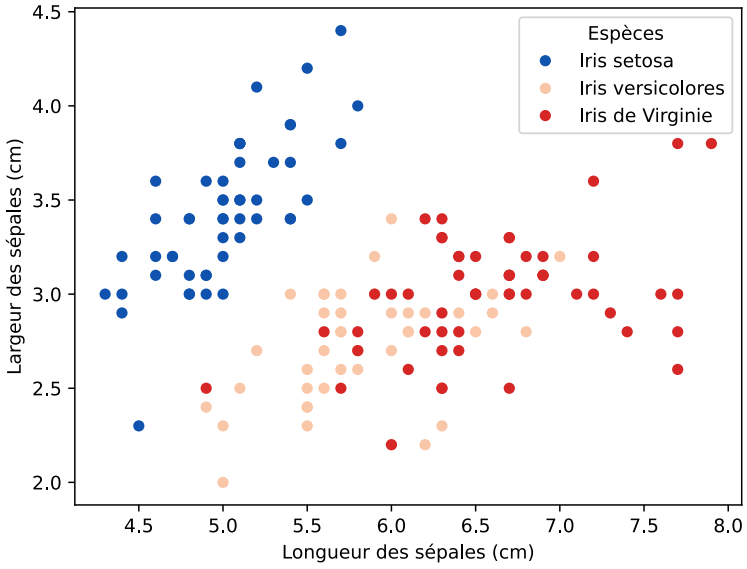
Iris versicolore

Iris de Virginie

Il est difficile de les différencier à l'œil nu, mais on peut extraire deux critères importants pour les déterminer : la longueur des sépales et leur largeur.

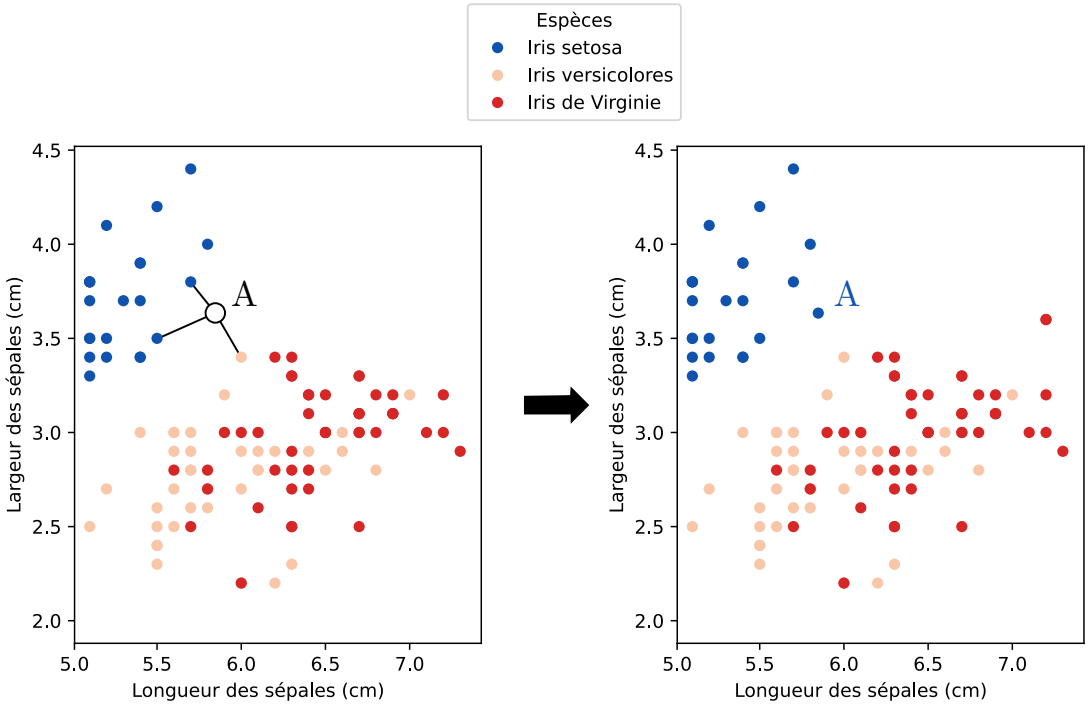
① **Phase d'apprentissage :**

En identifiant chacun de ces paramètres pour des individus donnés, on peut créer l'espace suivant :



② **Phase d'utilisation :**

Une fois que l'on dispose de ces données, pour déterminer l'espèce d'un nouvel individu (A), on place ce dernier dans l'espace créé et on s'intéresse à ses k plus proches voisins (par exemple $k = 3$). On en déduit la catégorie de cette iris à partir de l'espèce majoritaire de ces voisins.



Ici les 3 plus proches voisins de l'iris inconnue A sont majoritairement des iris setosa. On attribue donc cette espèce à A.

Remarque

Les algorithmes supervisés fonctionnent de la même manière que les méthodes d'étalonnage en TP :

	Algorithme supervisé	↔	**Mesure par étalonnage**
①	Phase d'apprentissage	↔	Construction de la courbe d'étalonnage
②	Phase d'utilisation	↔	Lecture de la mesure sur la courbe

Définition : Algorithmes de classification / de régression

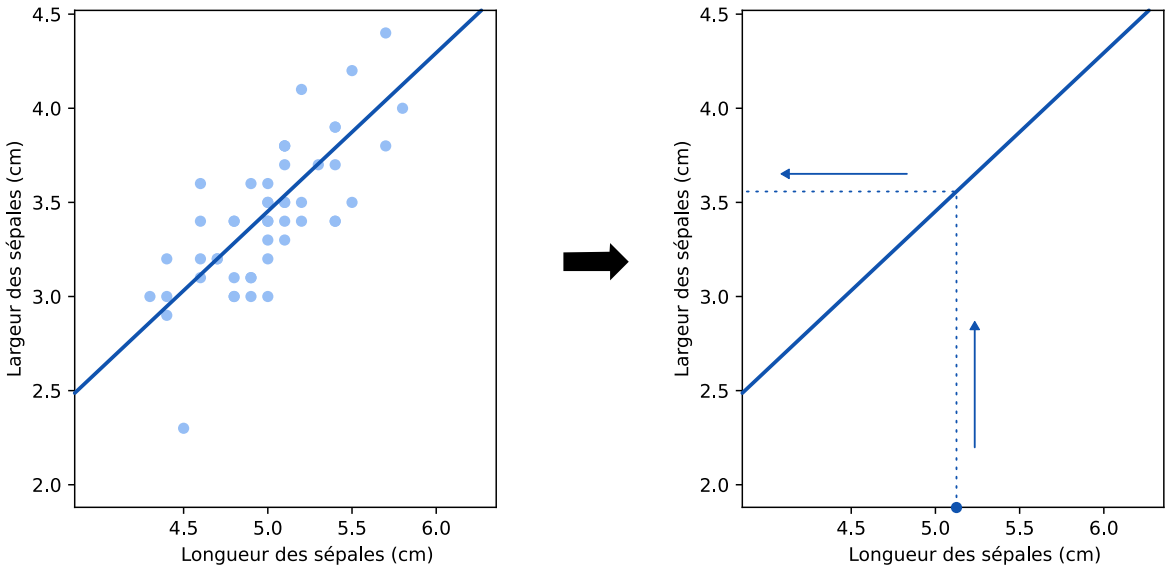
Le but d'un algorithme supervisé peut être :

- de classer des objets en catégories précédemment définies par l'utilisateur (on parle de **classification**) ;
- d'attribuer un nombre à un objet (on parle de **régression**).

Exemple

Classification : Comme présenté précédemment, on cherche à obtenir l'espèce d'une iris.

Régression : On peut chercher à anticiper la largeur des sépales d'une iris setosa à partir de sa longueur. Durant la phase d'apprentissage, on a accumulé des données nous permettant de tracer une droite d'étalonnage entre les deux paramètres.



Apprentissage
(Etablir la courbe d'étalonnage)

Utilisation
(Lecture de la mesure)

Remarque

La classification est une forme de régression, dont l'espace des nombres renvoyés est discret.

Matrice de confusion

Un algorithme supervisé ne sera jamais parfaitement efficace. Il peut donc être intéressant au moins d'arriver à estimer les erreurs commises. Dans le cas d'une méthode de classification, on utilise une **matrice de confusion**, construite après la phase d'apprentissage, permettant de résumer l'efficacité de la machine.

Il s'agit d'une matrice carrée de taille $n \times n$, où n est le nombre de catégories dans lesquelles on peut classer nos entrées :

chaque ligne `i` correspond à une classe réelle des entrées ;

chaque colonne `j` correspond à une classe estimée des entrées ;

chaque cellule `(i, j)` indique le nombre de cas détectés comme appartenant à la classe `j`, mais étant de fait de la classe `i`.

Exemple

On considère un test de détection de présence d'un virus dans une population. Les issues possibles d'une évaluation sont au nombre de 2 : "POSITIF" ou "NÉGATIF". La matrice de confusion sera donc de taille 2×2. On imagine par exemple les résultats suivants :

		Classe estimée	
		POSITIF	NEGATIF
Classe réelle	POSITIF	97	3
	NEGATIF	9	91

Dans cet exemple, on a donné en entraînement 200 entrées : $97 + 3 = 100$ cas réellement positifs et $9 + 91 = 100$ négatifs. Cette matrice permet de capturer les informations suivantes :

$$\begin{pmatrix} \textbf{Vrais positifs :} & 97 & \textbf{Faux négatifs :} & 3 \\ \textbf{Faux positifs :} & 9 & \textbf{Vrais négatifs :} & 91 \end{pmatrix}$$

On voit par exemple que sur 200 entrées, il y a eu $3 + 9 = 12$ erreurs, le taux d'erreur est donc de

$$\text{erreur} = \frac{12}{200} = 6\%$$

Mais ce nombre de capture pas toute la subtilité des résultats : il est bien plus grave d'avoir un faux test négatif qu'un faux positif. La matrice de confusion permet d'évaluer ces différents taux d'erreurs dont il est important de comprendre la différence :

Précision : Probabilité qu'un test positif corresponde effectivement à un individu infecté :

$$\text{précision} = \frac{97}{97 + 9} = 91.5\%$$

Sensibilité : Probabilité que le test évalue correctement un individu infecté :

$$\text{précision} = \frac{97}{97 + 3} = 97.0\%$$

Spécificité : Probabilité que le test évalue correctement un individu non-infecté :

$$\text{précision} = \frac{91}{9 + 91} = 91.0\%$$

C Apprentissage non supervisé

Définition : Apprentissage non supervisé (Unsupervised learning)

Dans un programme d'**apprentissage non supervisé**, la machine cherche elle-même comment regrouper les entrées en catégories non-fournies par l'opérateur.

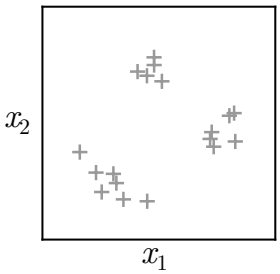
Remarque

C'est dans cette catégorie que l'on trouve les **réseaux de neurones**. Il s'agit de systèmes inspirés du fonctionnement neuronal du cerveau humain, capables d'apprendre en autonomie à résoudre des tâches complexes. Ce sous-domaine n'est pas étudié dans le cadre du programme.

Exemple

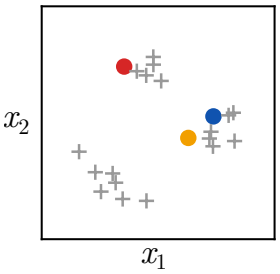
Algorithme des k moyennes :

Imaginons que l'on dispose d'un ensemble d'objet déterminés par deux paramètres x_1 et x_2. On représente cet ensemble dans le graphe ci-contre.

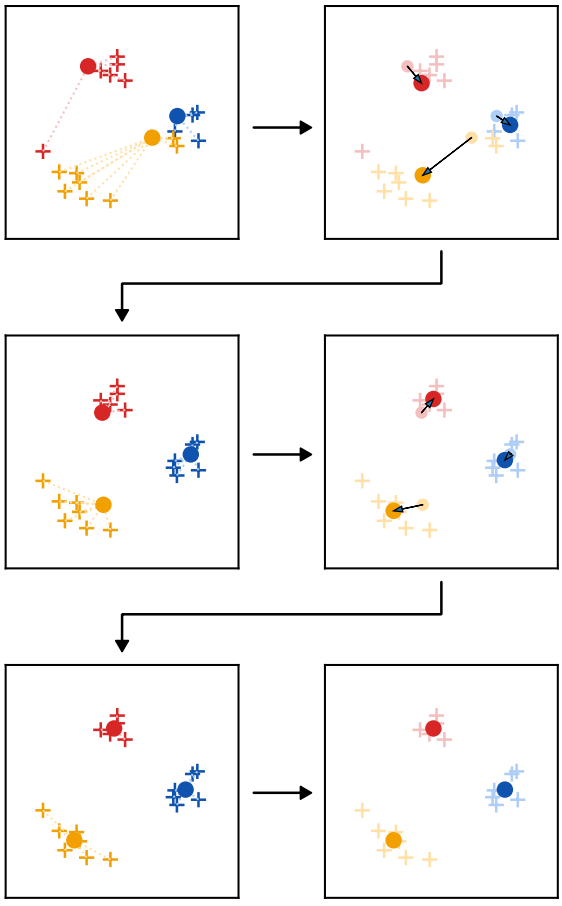


Il semble apparaître trois groupes distincts. Le but de notre algorithme va être d'identifier ces catégories. On résume ci-dessous le comportement de la machine :

1. Choisir trois nouveaux points aléatoires dans l'espace (x_1, x_2). On les appellera **barycentres**.



2. Répéter les opérations suivantes autant que nécessaire :
   Assignation Relier chaque point au barycentre le plus proche.
   Calcul des barycentres Pour chaque groupe ainsi formé, recalculer son barycentre.



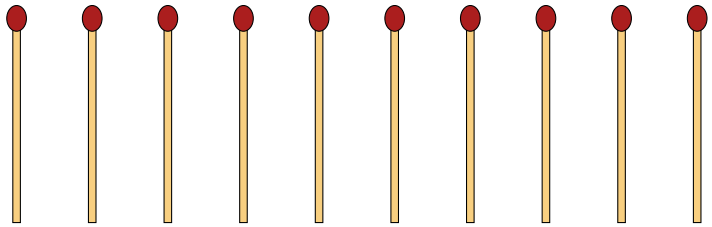
Au bout d'un certain nombre d'itérations, l'algorithme devrait converger vers une catégorisation donnée.

II Théorie des jeux

A Contexte

✔ Exemple

Durant toute notre étude des jeux, on prendra comme exemple le "jeu des bâtonnets", cas particulier du "jeu de Nim". Il s'agit d'un jeu à deux joueur · euses, dans lequel on dispose n bâtonnets positionnés sur la table :



Exemple d'une partie avec 10 bâtonnets.

Le jeu fonctionne ainsi :

Déroulé : Chaque joueur · euse prend à tour de rôle 1, 2 ou 3 bâtonnets et les retire du jeu.

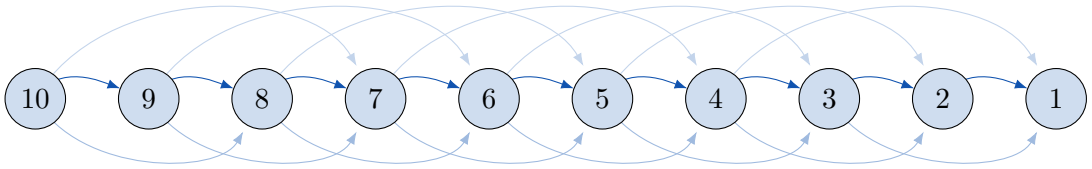
But : La personne qui prend le dernier bâtonnet perd (et son adversaire gagne).

La question naturelle qui se pose alors est :

Existe-t-il une stratégie permettant d'accéder à coup sûr à la victoire ?

Alice et Bob s'affrontent et c'est Alice qui commence. On peut résumer le jeu par un graphe

- dont chaque sommet représente une issue possible (nombre de bâtonnets restants) ;
- et dont les sommets sont reliés par de flèches indiquant les coups possibles.

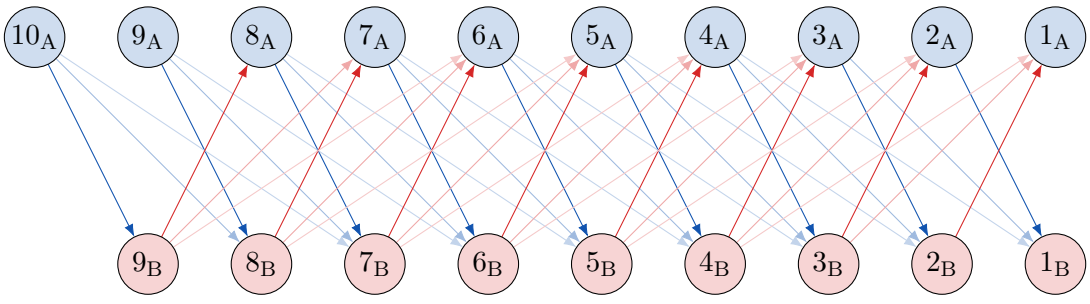


En partant de 10 bâtonnets, on peut Alice peut amener la prochaine situation à 9, 8 ou 7 bâtonnets.

Le problème de cette représentation, est qu'elle ne prend pas en compte qui joue. Par exemple s'il reste 8 bâtonnets, il y a deux configurations possibles :

- soit Alice a retiré 2 bâtonnets et donc c'est à **Bob** de jouer ;
- soit Alice puis Bob ont retiré chacun · e 1 bâtonnet et c'est donc à **Alice** de jouer.

Pour vraiment prendre en compte toutes les configurations possibles, il faut donc différencier les précédents sommets selon qui doit jouer :



Propriété : Jeu et graphe

Un jeu à n joueur · euses peut se représenter sous la forme d'un graphe où chaque joueur · euse contrôle certains sommets.

Définition : Graphe biparti

Dans un jeu à deux joueur · euses jouant à tout de rôle, aucune flèche ne peut relier deux sommets contrôlés par la même personne. On dit alors que le graphe est **biparti**.

✔ Exemple

Dans le jeu des bâtonnets, on a déjà visualisé le graphe du jeu. Ce graphe est bien biparti car chaque sommet bleu (contrôlé par Alice) n'est relié qu'à des rouges (contrôlés par Bob). Ceci est évident puisque que ni Alice, ni Bob ne peut jouer deux fois de suite.

Définition : Stratégie gagnante

On dit d'une stratégie qu'elle est **gagnante pour le joueur J à partir d'un état S** (sommet du graphe de jeu), si elle permet à J de gagner à coup sûr en partant de S.

Définition : Position gagnante

On dit d'un sommet S que c'est une **position gagnante** s'il existe une stratégie gagnante partant de S.

✔ Exemple

Dans le jeu des bâtonnets, un · e joueur · euse perd s'il ne reste qu'un bâtonnet à son tour. Donc le sommet 1_A fait perdre Alice (resp. 1_B fait perdre Bob). Ainsi par exemple :

- les positions 1_A 2_A et 3_A sont gagnantes pour Alice puisqu'elle peut diriger la partie vers 1_B ;
- les positions 1_B 2_B et 3_B sont gagnantes pour Bob puisqu'il peut diriger la partie vers 1_A ;

B Attracteurs

Méthode des attracteurs ♥

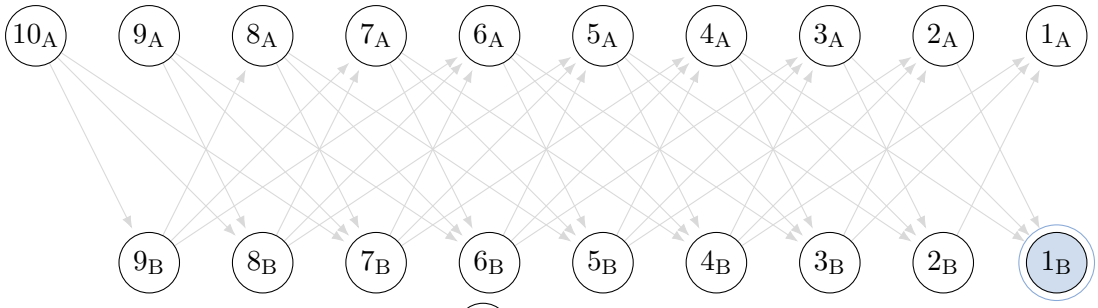
Pour déterminer une stratégie gagnante, on peut utiliser la **méthode des attracteurs**. L'idée est de partir des positions gagnantes triviales et de *remonter le temps* pour trouver les sommets qui permettent à coup sûr de s'y ramener, à condition de respecter une stratégie adaptée.

Dans un jeu où Alice et Bob s'affrontent, on construit récursivement des ensembles (appelés **attracteurs d'Alice**) de la manière suivante :

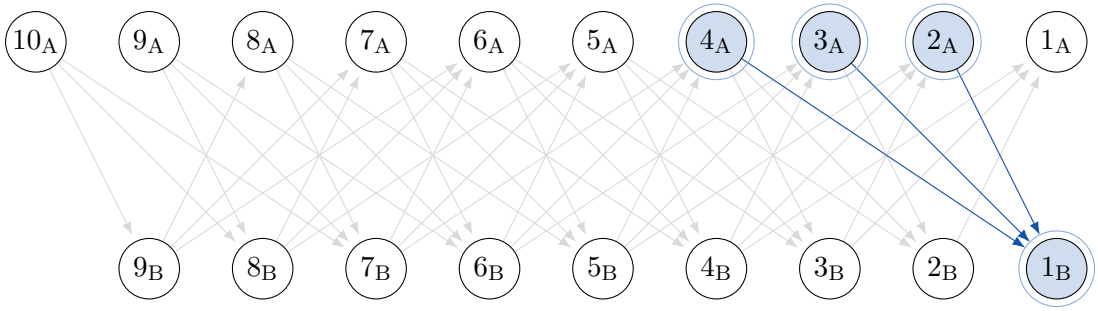
$$\begin{aligned} \mathcal{A}_0 &= \{ \text{ positions gagnantes pour Alice en zéro coup } \} \\ \forall n \in \mathbb{N}, \mathcal{A}_{n+1} = \mathcal{A}_n &\cup \{ \text{ sommets d'Alice permettant d'aller vers } \mathcal{A}_n \} \\ &\cup \{ \text{ sommets de Bob allant nécessairement vers } \mathcal{A}_n \} \end{aligned}$$

Ainsi on arrive petit à petit à identifier toutes les positions gagnantes pour Alice, jusqu'au sommet initial.

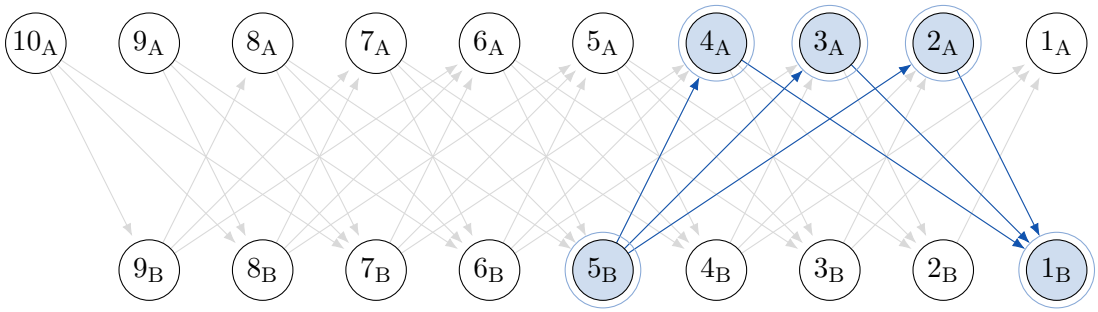
Exemple



$\mathcal{A}_0$ inclue simplement (1_B), faisant directement perdre Bob.

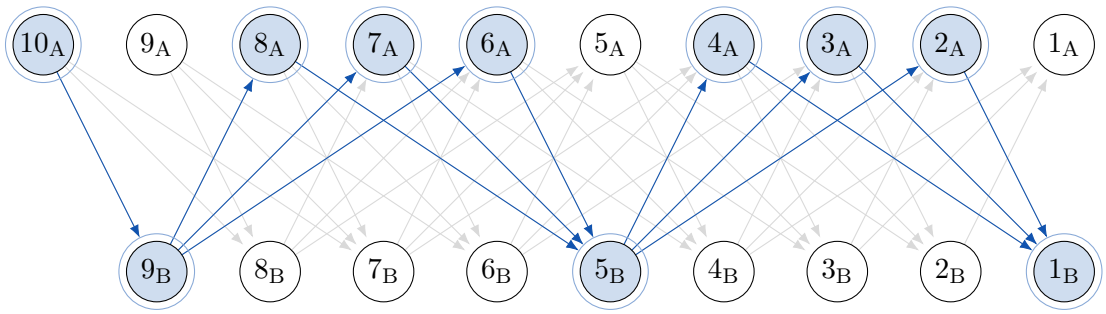


Dans $\mathcal{A}_1$, on ajoute les trois sommets qui permettent à Alice de revenir dans $\mathcal{A}_0$. Il n'y a aucun sommet de Bob qui ne dirige la partie directement vers $\mathcal{A}_0$.



Pour $\mathcal{A}_2$, il n'y aucune position d'Alice pouvant emmener dans $\mathcal{A}_1$. En revanche, on remarque que le sommet (5_B) de Bob, ne permet que d'aller en des positions de $\mathcal{A}_1$. On ajoute donc ce nœud à $\mathcal{A}_2$.

On remarque alors que la situation est similaire au cas initial. Par récursion, on peut compléter les attracteurs sur tout le graphe :



Les sommets mis en valeur si dessus sont inclus dans $\mathcal{A}_5$, l'attracteur final contenant toutes les positions gagnantes d'Alice.

Pour gagner la partie, Alice doit faire en sorte de toujours jouer des coups de manière à se déplacer dans l'attracteur le plus grand (ici $\mathcal{A}_5$).

C Algorithme minimax

Remarque

La méthode des attracteurs peut devenir très lente puisqu'il faut itérer à chaque fois sur tout le graphe restant pour tester les sommets n'étant pas encore dans un attracteur. Elle devient totalement inadaptée pour des jeux plus complexes. Les échecs par exemple peuvent être représentés par un graphe d'environ 10^{50} nœuds. Ce nombre monte à 10^{100} pour le jeu de go !

Les intelligences artificielles comme Stockfish (aux échecs) ou AlphaGo (jeu de go) fonctionnent par anticipation des prochains coups envisageables. Même si elles peuvent imaginer les issues vraisemblables de nombreux tours en avance, il leur toujours impossible d'être exhaustif quant aux situations possibles.

Définition : Fonction heuristique (ou fonction d'évaluation)

Pour un · e joueur · euse J, il peut être utile d'évaluer l'avantage que lui donne une certaine position p par un nombre réel. On va donc avoir besoin d'une fonction :

$$h : \begin{array}{c} \{\text{positions sur le graphe}\} \longrightarrow \mathbb{R} \\ p \longrightarrow h(p) \end{array}$$

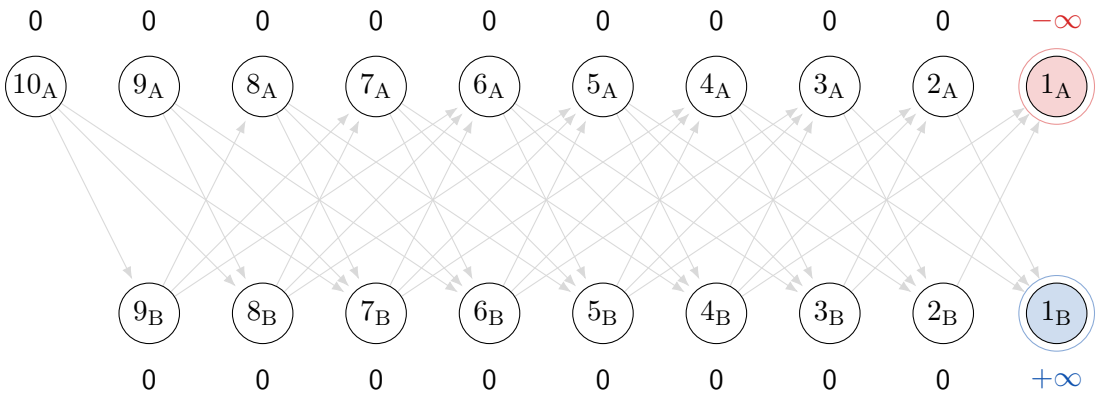
de sorte que plus $h(p)$ est grand, plus la position p est avantageuse pour J. Une telle fonction est dite **heuristique**, ou d'**évaluation**.

Remarque

- Une position gagnante pour J devrait renvoyer $+\infty$, puisque ce nœud est infiniment intéressant (permet de gagner à coup sûr).
- Dans un jeu à deux joueur · euses, une position gagnante pour l'adversaire doit renvoyer $-\infty$.

Exemple

Oublions la construction des positions gagnantes avec la méthode des attracteurs puisque l'objectif est justement de créer un nouvel algorithme de résolution du jeu. On peut directement assigner aux positions (1_A) et (1_B), les valeurs $\pm\infty$ puisqu'il s'agit des positions gagnantes triviales :



Le reste est *a priori* indéterminé, donc associe donc le score de 0 à chaque position.

Algorithme minimax ♥

Lorsqu'Alice doit jouer, elle construit un arbre d'évolution du jeu, à partir du nœud auquel elle se trouve. Elle itère cela jusqu'à une profondeur k (nombre de coups d'avance).

Ensuite elle procède en partant de la dernière ligne de l'arbre et en remontant petit à petit, tout en évaluant chacun des nœuds de la manière suivante :

- une position contrôlée par Bob prend comme valeur le **minimum** des scores lui succédant ;
- une position contrôlée par Alice prend comme valeur le **maximum** des scores lui succédant ;
- s'il n'y a pas de nœud succédant, garder la valeur donnée par l'heuristique ;

Une fois en haut de l'arbre, Alice joue le coup ayant le plus grand score.

Remarque

- Le nom de cet algorithme est dû au fait que les scores sont évalués alternativement en prenant un minimum puis un maximum.
- Cet algorithme anticipe les coups de l'adversaire, en supposant que celui-ci joue son meilleur coup disponible.
- Évidemment, plus la profondeur k est grande, meilleure est la prédiction. Cependant l'arbre croît de manière exponentielle avec k, donc pour des questions de complexité, on ne peut pas choisir une profondeur trop élevée.

Exemple

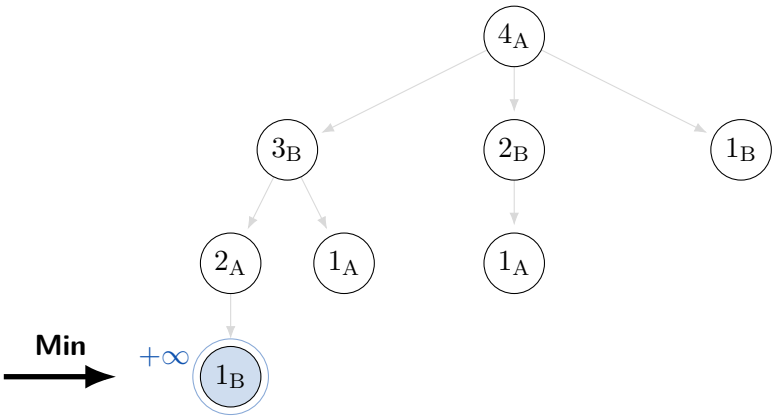
Par souci pédagogique, étudions le cas très simplifié où Alice commence à jouer avec 4 bâtonnets.

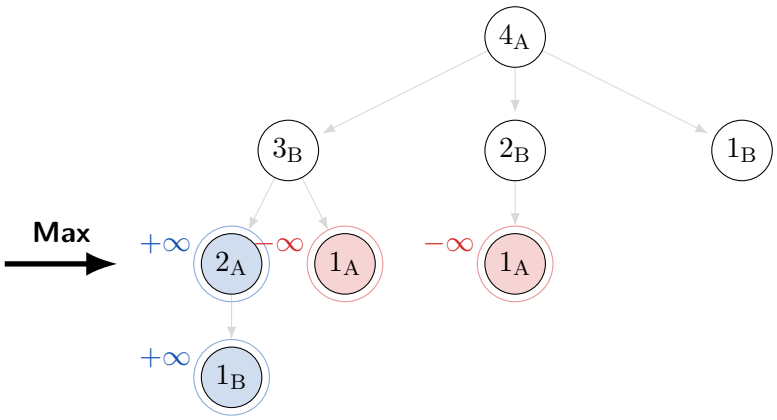
Construisons l'arbre d'évolution de la partie pour une profondeur $k = 3$ et remplissons les lignes une à une. On rappelle le comportement de notre fonction heuristique :

$$\forall p,\ h(p) = \begin{cases} +\infty & \text{si } p = (1_{\mathrm{B}}) \\ -\infty & \text{si } p = (1_{\mathrm{A}}) \\ 0 & \text{sinon} \end{cases}$$

Ligne 1 :

Il n'y a qu'un seul nœud tout en bas, celui-ci n'ayant pas de successeur, on lui attribue le score donné par la fonction heuristique.



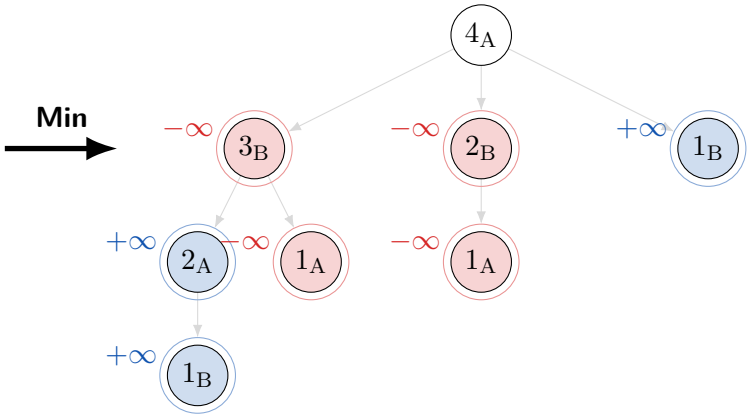


Ligne 2 :

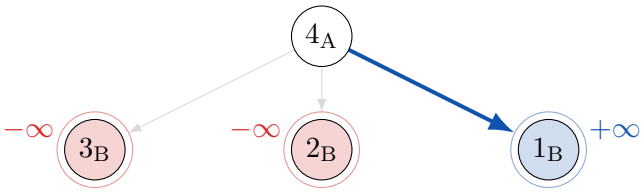
Sur cette ligne, les nœuds 1_A n'ont aucun successeur, donc on leur attribue leur score donné par l'heuristique. En revanche 2_A est contrôlé par Alice, on lui donne donc le score maximum de ses successeurs (il n'y en a qu'un).

Ligne 3 :

Cette fois-ci, les nœuds sont contrôlés par Bob. Il faut donc leur attribuer les scores minimaux de leurs successeurs. Sauf 1_B qui n'en n'a pas, et se voit donc assigner sa valeur heuristique.



Ainsi, Alice a trois options dont deux perdantes (score $-\infty$) et une gagnante (score $+\infty$). Elle fait donc le choix optimal maximisant la valeur :



On retrouve le même résultat qu'avec la méthode des attracteurs : 4_A est une position gagnante pour Alice (score $+\infty$) puisqu'il lui suffit d'enlever 3 bâtonnets pour mettre Bob en échec.

Remarque

- L'exemple ci-dessous est extrêmement simplifié. L'utilisation de la méthode minimax par rapport à celle des attracteurs techniquement pas justifiée. Cet algorithme manque d'intérêt dans ce jeu, où chaque nœud est soit gagnant, soit perdant.
- On voit bien que l'utilisation des fonctions **Min** et **Max** permet de simuler une partie où chaque joueur · euse joue toujours son coup optimal.
- On remarque une certaine redondance dans les racines de l'arbre. L'utilisation de la programmation dynamique (**Chapitre :** *Dictionnaires et programmation dynamique*) est tout à fait bienvenue ici !