3I005 Projet Exploration/Exploitation

BOUKHALFI Mourad BESSOUL Amine

Table des matières

1 Introduction

Ce mini-projet s'intéresse à la problématique de l'exploitation vs exploration que l'on retrouve dans plusieurs domaine de l'intelligence artificielle, en particulier en Machine Learning. Cette problématique consiste à choisir, parmi un certain nombre de choix possibles, s'il vaut mieux exploiter la connaissance acquise et arrêter l'exploration trop sous peine d'avoir à choisir une action sous-optimale ou s'il faut au contraire s'il faut utiliser le maximum de ressource pour l'exploration pour avoir plus de chance de trouver une solution optimale et donc avoir moins de ressources disponibles pour son exploitation.
Dans ce mini-projet, cette problématique est mise en avant par trois situations.
La première s'intéresse à l'exemple des Bandits-manchots. Celle-ci repose sur un principe assez simple: pour une mise, on a le droit d'actionner un levier qui fait tourner des rouleaux, et en fonction de la combinaison obtenue sur les rouleaux, une récompense est attribué au joueur,
La deuxième situation est celle du jeu de morpion, où trois stratégies de jeu sont implémentées. La première consiste dans une stratégie purement aléatoire, la deuxième est une stratégie de Monte Carlo, qui explore les actions possibles de façon aléatoire et choisit la meilleure et la dernière, une stratégie de Monte Carlo Tree Search, qui utilise l'algorithme UCB pour optimiser l'exploration des actions possibles.

1.1 Préambule technique

Le projet a été développé en python 3 et utilise les modules suivant :
-scipy.sparse.linalg
-pyAgrum
-numpy, matplotlib.pyplot, math, time, random, …

2 Bandits-manchots

2.1 Description

Le but de cette partie est de tester différents algorithmes du dilemme d'exploration vs exploitation pour des IAs de jeu.
On considère une machine à sous avec N leviers, numérotés par les entiers de 0 à N 1.
Chaque levier, lorsqu'il est actionné, peut donner une récompense de 0 ou 1 de façon aléatoire.
On suppose que tous les leviers sont indépendants, que deux actionnements différents du même levier sont aussi indépendantes, et que la récompense du levier i suit une loi de Bernoulli de paramètre i constant en temps.
Le gain du joueur est ensuite calculé en faisant la somme de ses récompenses au cours de toutes ses parties.

On va, pour modéliser cela implémenter quatre algorithmes:

• *Algorithme aléatoire:* Dans cet algorithme, le choix du levier que le joueur va utiliser est choisit aléatoirement. Les actions sont donc choisis uniformément.

• *Algorithme greedy:* L'idée de cet algorithme est de décomposer le jeu en deux phases bien distinctes. La première est une phase d'exploration qui s'apparente à ce qui s'est fait dans l'algorithme aléatoire. Dans la deuxième phase l'algorithme exploite les informations obtenues (phase d'exploitation) en jouant toujours au levier correspondant au maximum de gain.

• *Algorithme ε-greedy:* Contrairement aux deux premiers cités plus haut, cette algorithme est censé s'améliorer au fur et à mesure qu'il est lancé car il possède un inconvénient majeur. En effet si le levier choisi est mauvais, on jouera toute les parties avec celui-ci . Dans cet algorithme on choisit avec une probabilité aléatoire ε de faire appel à l'algorithme aléatoire afin d'améliorer l'exploration.

• *Algorithme UCB:* Cet algorithme s'apparente au précédent, cependant il cherche à faire une exploration de façon plus intelligente en utilisant la formule suivante:

$$a_t = \underset{i \in \{0,\dots,N-1\}}{\operatorname{argmax}} \left(\hat{\mu}_t^i + \sqrt{\frac{2\log(t)}{N_i(t)}} \right)$$

Le regret est définit comme la différence entre le gain maximal espéré et le gain réel. Ainsi plus le regret sera faible, meilleur sera l'algorithme.

2.2 Résultats

On prends une machine respectant ces conditions de gain:[0.4, 0.5, 0.25, 0.1, 0.4, 0.2]
On réalise une simulation sur 1000 parties et on observe les résultats suivants:

Pour l'*Algorithme aléatoire, on obtient un gain de 296 et on observe la courbe suivante*

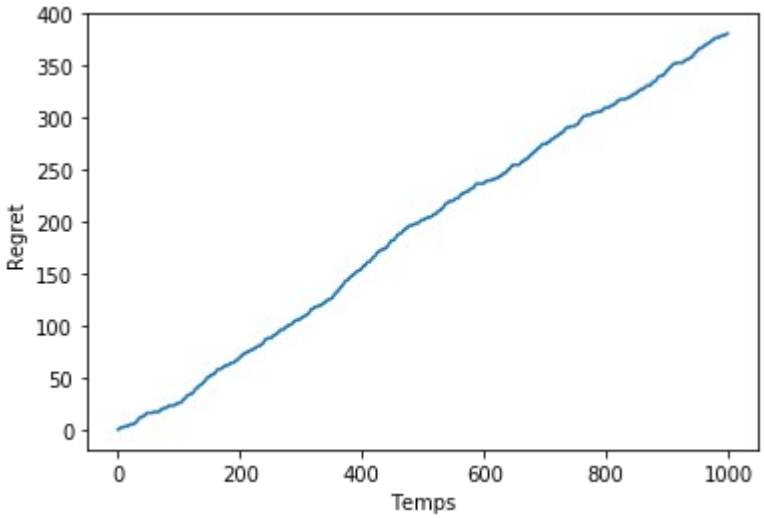


Figure 1- Gain en fonction du temps

Pour l'*Algorithme greedy, on obtient un gain de 369 et on observe la courbe suivante*

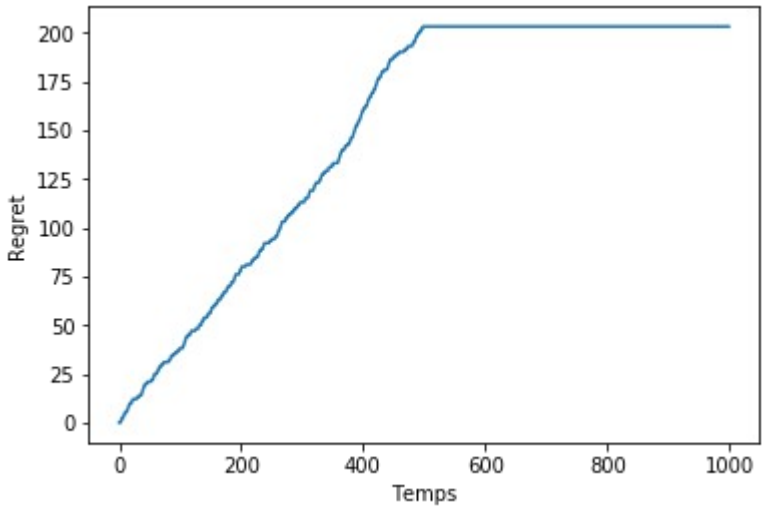


Figure 2- Gain en fonction du temps

Pour l'*Algorithme ε-greedy, on obtient un gain de 368 et on observe la courbe suivante*

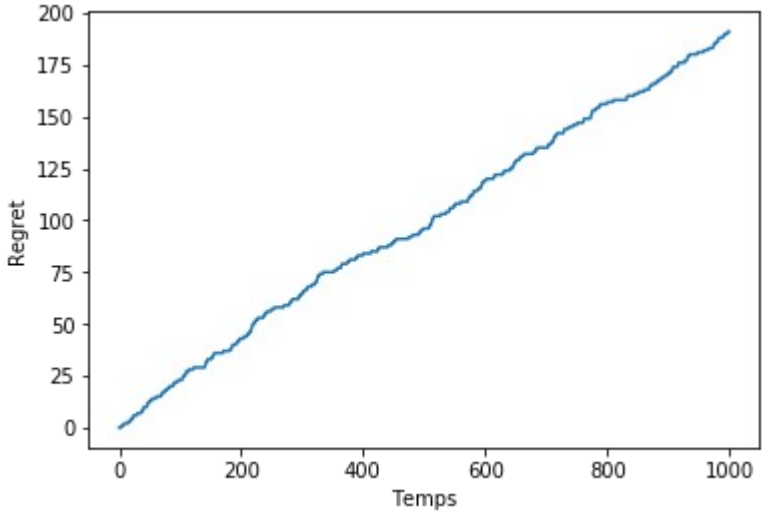


Figure 3-Gain en fonction du temps

Pour l'*Algorithme UCB, on obtient un gain de 390 et on observe la courbe suivante*

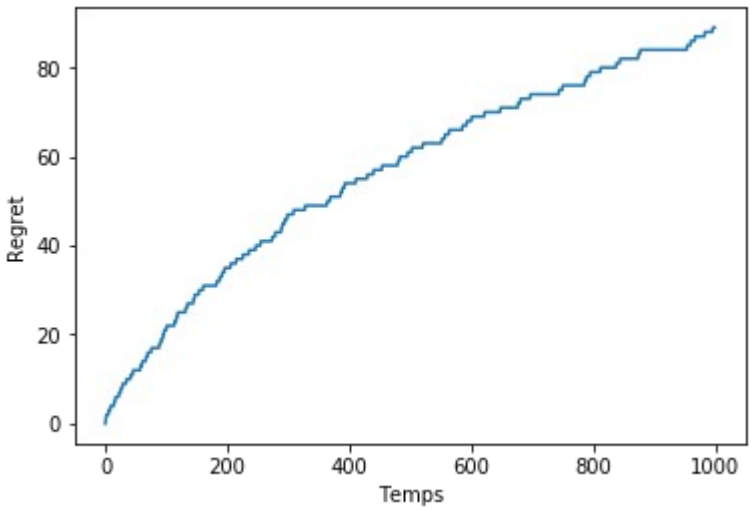


Figure 4- Gain en fonction du temps

Nous avons réalisé plusieurs simulations avec d'autres valeurs et on constate de manière générale que l'algorithme UCB est celui qui possède le plus petit regret.
L'algorithme aléatoire est quant à lui bien évidemment le plus mauvais.
Les deux autres algorithmes greedy et *ε-greedy sont plus au moins efficaces en fonction de leurs paramètres.*
Cela dit, lorsqu'on diminue la valeur de ε dans l'algorithme ε-greedy, son efficacité se rapproche de celle de l'algorithme UCB.

3 Morpion

2.1 Description

Dans cette partie, on s'intéresse à différents algorithmes permettant de jouer au jeu du morpion. Dans ce mini-projet, une implémentation de ce jeu nous à été fournie permettant de représenter l'état générique d'un jeu de plateau à deux joueurs, une simulation du jeu générique, une représentation de l'état d'un jeu de morpion ainsi qu'une représentation du comportement d'un agent.

2.1 Les algorithmes

Ce projet implémente trois algorithmes différents pour jouer au morpion: une stratégie aléatoire, une stratégie de Monte Carlo ainsi qu'une stratégie Monte Carlo Tree Search.

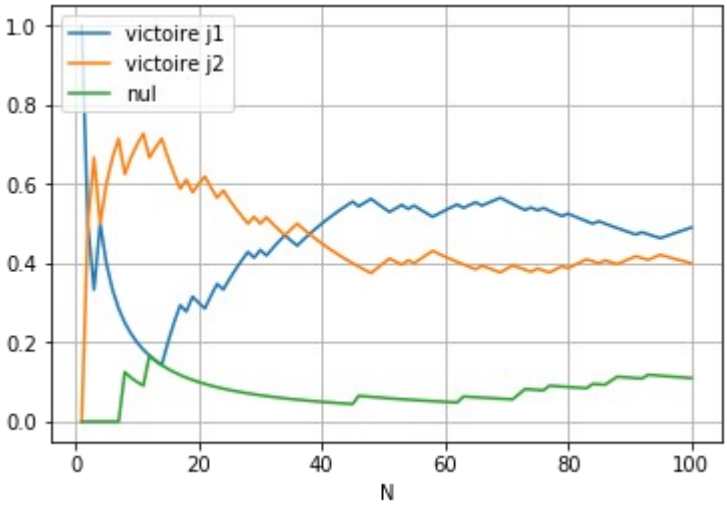
- *Stratégie aléatoire:* Comme pour le problème des bandits-manchots, la stratégie aléatoire n'utilise aucune information sur l'état du jeu et réalise des choix aléatoirement.

- *Stratégie Monte Carlo: Cet algorithme consiste à donner une approximation d'un résultat trop complexe à calculer: il s'agit d'échantillonner aléatoirement et de manière uniforme l'espace de possibilités et de rendre comme résultat la moyenne des expériences.Pour chaque action possible, la stratégie de Monte Carlo simule plusieurs parties aléatoire et calcule le taux de victoire de chaque action, choisissant celle avec le plus grand taux.*

2.1 Résultats

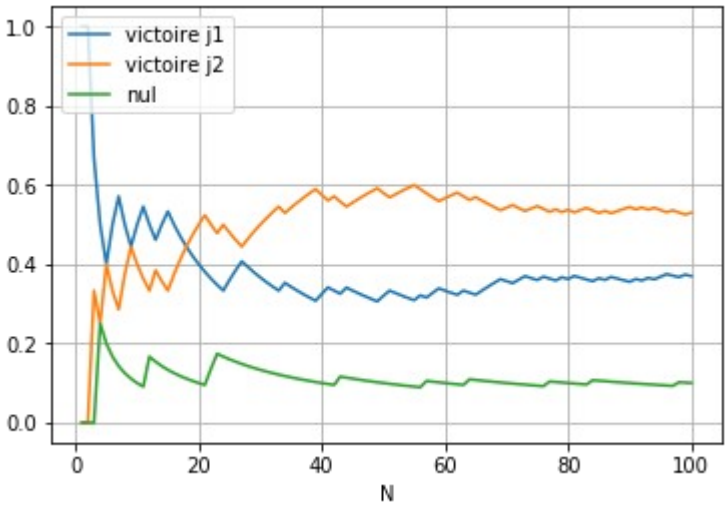
Nous avons réaliser différentes simulations entre des joueurs aléatoire et des joueurs Monte Carlo.

Joueur aléatoire contre joueur aléatoire



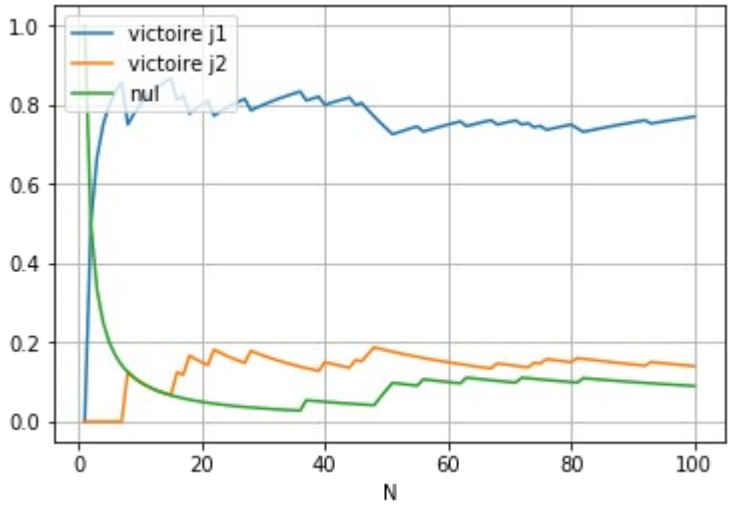
probabilité j1: 0.45970107148341205
probabilité j2: 0.4650378359177765
probabilité match nul: 0.07526109259881134

Joueur aléatoire contre joueur Monte Carlo



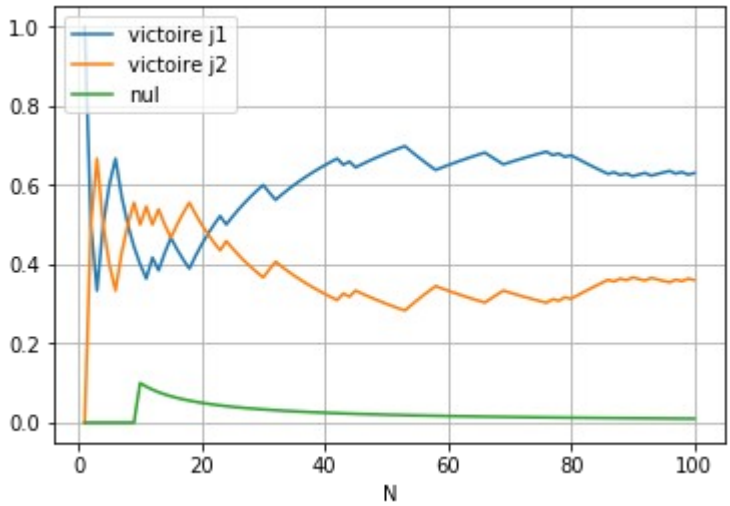
probabilité j1: 0.3872331740242882
probabilité j2: 0.5033242234337901
probabilité match nul: 0.10944260254192169

Joueur Monte Carlo contre joueur aléatoire



probabilité j1: 0.7643633488156334
probabilité j2: 0.136197858305947
probabilité match nul: 0.09943879287841946

Joueur Monte Carlo contre joueur Monte Carlo



probabilité j1: 0.6048639731431056
probabilité j2: 0.37155193422018085
probabilité match nul: 0.023584092636713655

5 Conclusion

Ce projet nous a permis de mieux comprendre la problématique de l'exploitation vs exploration via des exemples bien précis, notamment via l'exemple des bandits-manchots, qui illustre un certain équilibre entre exploration et exploitation à l'aide du très efficace algorithme UCB.