3I005
TME 2–4 : Projet Exploration / Exploitation

Ariana Carnielli
Yasmine Ikhelif

Table des matières

1 Introduction

Ce mini-projet s'intéresse à la problématique de l'exploitation *vs* exploration, qui consiste à choisir, parmi une quantité de ressources limitées, combien de ces ressources vont être utilisées pour explorer le problème, afin d'avoir plus de chances de trouver la meilleure solution possible, et combien seront utilisées pour exploiter cette meilleure solution trouvée. Si beaucoup de ressources sont dépensées pour l'exploration, on aura plus de chances de trouver une solution proche de l'optimale, mais moins de ressources disponibles pour son exploitation. De l'autre côté, arrêter l'exploration trop tôt peut conduire au choix d'une action sous-optimale pendant la phase d'exploitation, ce qui conduit à un gain plus petit à long terme.

Dans ce mini-projet, on illustre la problématique de l'exploitation *vs* exploration dans trois situations. La première, décrite dans la Section 2, s'intéresse à l'exemple des bandits-manchots. Il s'agit de considérer une machine à sous à N leviers, chacun ayant une probabilité de victoire différente inconnue du joueur. L'objectif est de maximiser le gain, ce qui nécessite un bon équilibre entre l'exploration des différents leviers afin d'avoir une bonne estimée de la probabilité de gain de chacun et l'exploitation du meilleur levier. L'algorithme principal permettant cet équilibre est l'algorithme UCB, dont les détails sont donnés dans la Section 2.

La deuxième situation considérée, décrite dans la Section 3, 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.

Finalement, ces mêmes algorithmes, ayant été codés de façon généraliste, sont appliqués à un jeu légèrement plus complexe que morpion, le jeu puissance 4. Les détails de l'implémentation et les résultats obtenus sont donnés dans la Section 4.

2 Bandits-manchots

2.1 Description

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 μ^i constant en temps.

On dispose de T coups pour jouer à la machine à sous. À chaque coup $t \in \{0,\dots,T-1\}$, on choisit une *action* $a_t \in \{0,\dots,N-1\}$, qui représente le levier choisi pour ce coup, et on accumule le gain obtenu avec ce levier, noté par r_t. Ainsi, r_t est une variable aléatoire suivant une loi de Bernoulli de paramètre μ^{a_t}. L'objectif est de maximiser le gain total G_T obtenu au bout de T coups, $G_T = \sum_{t=0}^{T-1} r_t$. Comme G_T est une variable aléatoire, on maximise son espérance, qui vaut

$$\mathbb{E}(G_T) = \mathbb{E}\left(\sum_{t=0}^{T-1} r_t\right) = \sum_{t=0}^{T-1} \mathbb{E}(r_t) = \sum_{t=0}^{T-1} \mu^{a_t}.$$

Ainsi, si les $\mu^0,\dots,\mu^{N-1}$ étaient connus, la stratégie maximisant son espérance serait de choisir

$$a_t = \operatorname*{argmax}_{i\in\{0,\dots,N-1\}} \mu^i$$

pour tout $t \in \{0,\dots,T-1\}$. Si on note μ^* la valeur maximale de $\{\mu_0,\dots,\mu_{N-1}\}$ et G_T^* la variable aléatoire G_T avec la stratégie a_t définie ci-dessus, le maximum de l'espérance de G_T vaut $\mathbb{E}(G_T^*) = \sum_{t=0}^{T-1} \mu^* = T\mu^*$.

Comme les μ^i ne sont pas connus, la stratégie ci-dessus ne peut pas être utilisée en pratique. Pour une autre stratégie, on s'intéresse au regret L_T du joueur au bout de T coups, défini comme la différence entre l'espérance du gain obtenu avec la stratégie optimale ci-dessus et son gain réel, $L_T = \mathbb{E}(G_T^*) - G_T = T\mu^* - \sum_{t=0}^{T-1} r_t$. Cette définition est légèrement différente de celle donnée dans l'énoncé du projet car on utilise l'espérance de G_T^* au lieu d'un G_T^* aléatoire. Ce choix, arbitraire, a été fait pour assurer que, lorsque deux résultats identiques sont obtenus par deux joueurs, leur regret sera aussi identique.

On est alors confronté à un problème du type exploitation *vs* exploration, où l'exploration consiste à tester les leviers afin d'estimer leurs paramètres μ^i et l'exploitation consiste à jouer le levier avec le plus grand paramètre μ^i estimé.

2.2 Implémentation du jeu

L'implémentation de la machine à sous a été faite en stockant les leviers d'une machine comme un tableau de taille N contenant les paramètres $\mu^0, \ldots, \mu^{N-1}$ des N machines. La fonction `cree_machine` permet de créer un tel tableau avec des paramètres choisis de façon aléatoire uniforme dans $[0,1]^N$. Un coup du jeu est simulée par la fonction `jouer`, qui prend en argument une machine et l'action / levier choisi et rend le gain correspondant au coup joué. Une stratégie pour le jeu est une fonction quelconque prenant en argument un tableau `mu` donnant la proportion de gains de chaque levier et un deuxième tableau `Na` contenant la quantité de coups jouées dans chaque levier, et qui renvoie un coup à jouer à la prochaine étape. Les T coups d'une partie sont simulés par la fonction `run`, qui prend en argument la machine, la stratégie et T et qui utilise les fonctions précédentes pour simuler les T coups, renvoyant à la fin les dernières versions des tableaux `mu` et `Na` et un tableau de taille T avec le gain cumulé à chaque $t \in \{0, \ldots, T-1\}$. La fonction `run` utilise la fonction auxiliaire `mise_a_jour` qui met à jour les tableaux `mu` et `Na` en fonction du levier choisi et du résultat obtenu.

2.3 Stratégies

Dans ce projet, quatre stratégies ont été implémentées.

- **Algorithme aléatoire.** Dans cet algorithme, le levier à jouer dans le coup suivant est choisi de façon aléatoire uniforme parmi les N leviers disponibles. Il s'agit d'un algorithme qui fait uniquement une exploration des N leviers sans utiliser aucune information obtenue pour exploiter le meilleur levier observé.
- **Algorithme glouton.** L'idée de cet algorithme est de décomposer le jeu en deux phases bien distinctes, l'une avec uniquement de l'exploration et l'autre avec uniquement de l'exploitation. Dans la première, les N leviers sont joués chacun la même quantité de fois, passée en argument à la stratégie, afin d'estimer les paramètres μ^i de chaque levier par la proportion de gains obtenue expérimentalement sur ce levier stockée dans le tableau `mu`. Remarquons que cette estimée est justifiée par la loi des grands nombres, qui garantit que la proportion des gains obtenue expérimentalement avec le levier i converge vers le paramètre μ^i. Dans la deuxième phase, l'algorithme glouton exploite les informations obtenues en jouant toujours au levier correspondant au maximum du tableau `mu`.
- **Algorithme ε-glouton.** L'inconvénient de l'algorithme glouton est que le tableau `mu` contient uniquement des estimées de μ^i, et pas leurs vraies valeurs, qui sont inaccessibles. Ainsi, si ces estimées sont mauvaises, ce qui peut arriver de façon aléatoire, on risque de passer toute la phase d'exploration en jouant un levier qui n'est pas l'optimal. Pour y remédier, l'algorithme ε-glouton modifie l'algorithme glouton en introduisant un paramètre ε petit. À chaque coup, on a une probabilité ε de choisir un levier au hasard, de façon uniforme parmi les N leviers, et, avec une probabilité

$1-\varepsilon$, on joue comme dans l'algorithme glouton. Cela garantit que, même après la fin de la première phase de l'algorithme glouton, on continue à avoir un peu d'exploration pour essayer d'améliorer les estimées de `mu` et ainsi éviter de rester bloqué sur une action sous-optimale.

- **Algorithme UCB.** L'algorithme ε-glouton permet de garder un peu d'exploration pendant la phase d'exploitation, mais cette exploration est faite de façon purement aléatoire. L'algorithme UCB cherche à faire une exploration de façon plus intelligente avec les informations disponibles. L'idée de cet algorithme est de choisir la prochaine action a_t par la formule

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

où $\hat{\mu}_t^i$ est l'estimée de μ^i disponible à l'instant t et $N_i(t)$ est la quantité de fois que le levier i a été actionné jusqu'à l'instant t. Le terme $\sqrt{\frac{2\log(t)}{N_i(t)}}$ est celui responsable pour l'exploration ; sans ce terme, cette formule est exactement celle utilisée dans la phase d'exploitation de l'algorithme glouton. La division par $N_i(t)$ indique que, plus on a joué sur le levier i, plus notre estimée $\hat{\mu}_t^i$ est proche de μ^i. Le terme en $\log(t)$ garantit qu'on ne néglige pas complètement un levier : même si son estimée de $\hat{\mu}_t^i$ est trop petit, s'il a été trop peu joué, le terme $\sqrt{\frac{2\log(t)}{N_i(t)}}$ augmente au cours du temps et il sera éventuellement rejoué, afin d'améliorer encore l'estimée de $\hat{\mu}_t^i$. Comme la croissance de $\log(t)$ est assez lente, ces explorations de leviers avec une estimée petite n'arrivent pas très souvent pour ne pas trop nuire à l'exploration. Remarquons aussi que, comme il y a une division par $N_i(t)$, il est nécessaire de joueur chaque levier au moins une fois avant de pouvoir appliquer cet algorithme, ce qui impose une phase d'exploration pendant au moins les N premiers coups.

Ces quatre stratégies ont été implémentées comme quatre fonctions, `algo_alea`, `algo_glouton`, `algo_glouton_e` et `algo_UCB`, prenant toutes en argument les tableaux `mu` et `Na` comme décrit dans la Section 2.2. L'algorithme glouton prend aussi en argument facultatif la quantité de fois que chaque levier est joué pendant la phase d'exploitation et l'algorithme ε-glouton prend en argument facultatif la valeur de ε.

3 Morpion

4 Puissance 4

5 Conclusion

Ce rapport a permis, dans la Section 2, de mettre en évidence la problématique de l'exploitation *vs* exploration à travers l'exemple des bandits-manchots, qui illustre bien l'intérêt d'un algorithme équilibrant exploration et exploitation comme l'algorithme UCB. On a également pu remarquer, dans les Sections 3 et 4, que l'utilisation de cet

algorithme dans la stratégie Monte Carlo Tree Search permet d'obtenir des joueurs très performants.