Compte rendu de TP bandit

Chaumette Noé
Sneed Jérémie

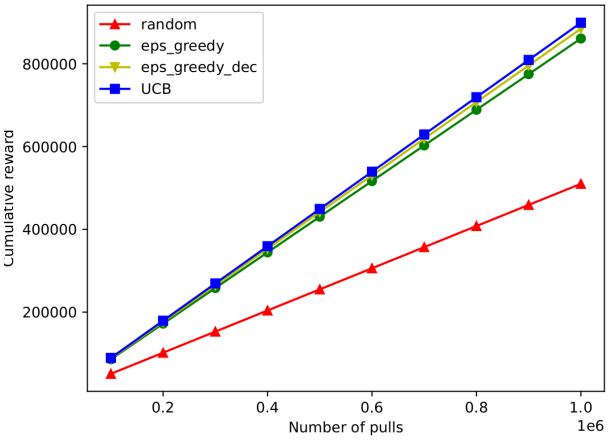
STI, 4A

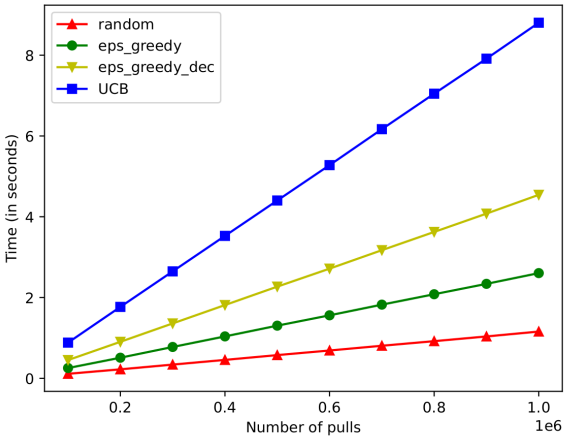
1 Implémentation et comparaison d'algorithmes bandits

Après avoir implémenté le pseudo code bandit en python, nous avons 2 expériences différentes afin d'illustrer les points forts et faibles de nos algorithmes. Les 2 expériences ont été réalisées en ayant lancé l'algorithme 100 fois afin d'avoir des résultats représentatifs. Notre première expérience a été effectuée avec $K = 10$:

$\mu_1 = 0.4, \mu_2 = 0.8, \mu_3 = 0.1, \mu_4 = 0.2, \mu_5 = 0.2, \mu_6 = 0.5, \mu_7 = 0.7, \mu_8 = 0.9, \mu_9 = 0.7, \mu_{10} = 0.6$

Nous avons obtenu les courbes suivantes :





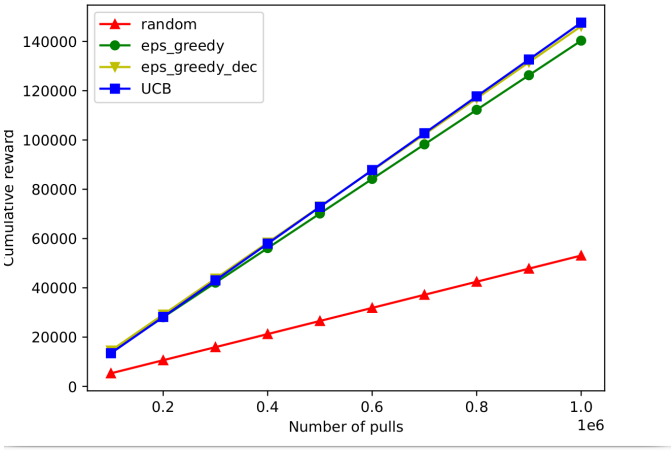
Notre résultat est cohérent car, pour les récompenses, l'algorithme *UCB* est le plus efficace car le plus rapide à trouver les branches intéressantes afin de les exploiter correctement. En effet, l'algorithme *random*, par exemple, est le plus mauvais car sa stratégie est complétement aléatoire, et, si l'on regarde la moyenne des μ pour 1000000 d'essais, on a 510000, ce qui correspond au graphique. Les algorithmes ϵ et ϵ_{dec} sont plus efficaces que *random* car ils ont une stratégie d'éxploitation qui leur permet d'optimiser l'appel des branches.

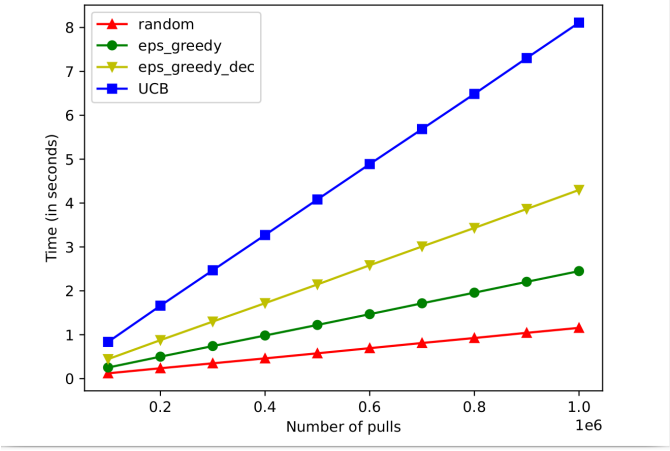
ϵ_{dec} est meilleur que ϵ car plus on avance dans les tentatives, plus il va avoir une stratégie d'éxploitation plutôt que d'éxploration, il va donc affiner son utilisation des branches. *UCB* est le plus optimal car il va essayer d'éxploiter au maximum, peut-être en négligeant un peu trop l'éxploration. Nous allons voir ça dans notre second cas.

Pour notre deuxième expérience, nous avons toujours K = 10 mais des μ différents :

$\mu_1 = 0.1, \mu_2 = 0.05, \mu_3 = 0.05, \mu_4 = 0.01, \mu_5 = 0, \mu_6 = 0.1, \mu_7 = 0.001, \mu_8 = 0.05, \mu_9 = 0.15, \mu_{10} = 0.02$

Nous avons obtenu les courbes suivantes :





Pour ce résultat, l'algorithme *random* est le moins efficace pour les même raison que précédemment. Par contre, l'algorithme *UCB* sera moins efficace car l'utilisation de valeurs très petites et très proches peut désavantager ce dernier par rapport à ϵ_{dec} ou ϵ.
Bien que, globalement, UCB est plus efficace sur la majorité des stratégie par rapport à ϵ_{dec}.
Pour le graphique du temps il est tout à fait logique qu'il soit similaire à celui de la précédente stratégie car les valeurs de μ ne jouent pas sur la compléxité de l'algorithme et donc sur le temps d'éxécution.

2 Proposition d'un nouveau protocole sécurisé

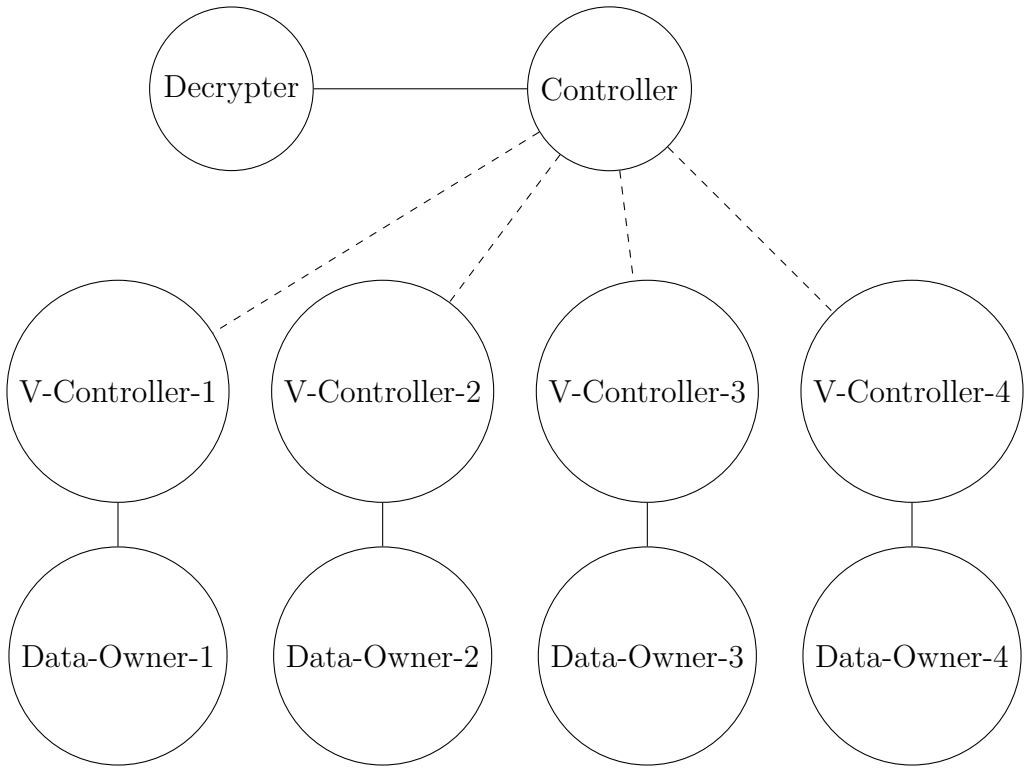
2.1 Présentation de Samba

Le protocole Sambda est un framework générique pour sécuriser les $multi-armed\ bandits$. Le principe de base est simple, chaque $data-owner$ possède sa propre distribution des récompenses qui en plus est inconnue des autres.
De plus, seul lui peut voir sa récompense cumulée, les autres n'ont accès à aucune information sur lui. Chacun ne peut connaître des informations que sur eux même.
Le protocole Samba est très utile car il peut utiliser de nombreux algorithmes bandits standatds (comme UCB par exemple) et c'est un protocole sécurisé à l'aide de cryptographie (AES en général mais cela peut varier).
Pour plus de détails, le score va être encrypté par le $data-owner$ et être envoyé au

controlleur. Ce controlleur va créer une nouvelle liste avec le texte encrypté et va envoyer la liste permutée à *comp*. Comp sera capable de décrypter chaque text et va créer une liste remplie de 0, sauf à la case i qui devra être *pull*. Il va envoyer cette liste au controlleur qui va la dépermuter et donner chaque récompenses aux bonnes personnes.

2.2 Nouveau protocole sécurisé

Nous pouvons imaginer un nouveau protocole basé sur de l'isolation (en utilisant la technologique de μ-noyau par exemple).
En effet, ce protocole aurait un *controlleur* qui pourrait créer un nouveau *controlleur* pour chaque $data - owner$. Chaque *controlleur* associé aurait la distribution personnalisé du $data - owner$ un peu comme pour le Samba.
Ainsi, les $data - owners$ ne seront pas du tout en contact avec les controlleur des autres, ils seront donc isolés.
De plus, pour protéger des packets sniffer, les données seront encryptés en utilisant un algorithme performant comme AES par exemple. Seul le controlleur originel sera en lien avec un decrypteur qui lui permettra d'analyser les résultats et de continuer l'algorithme.
Pour plus de détails voici un schéma explicatif :



Nous avons donc les V-Controller qui sont les controlleurs virtuels séparés par le μ-noyau afin d'avoir des processus complétement isolés pour chaque data owner. Et la partie de decryptage avec le controlleur originel afin qu'il puisse décrypter ce qu'il reçoit afin de l'analyser. Petite précision, nous ne connaissons pas la validité exacte de notre système et s'il est réellement viable. Mais nous espérons qu'il puisse l'être.