COMPTE RENDU

TP OST – Décision dans l'incertain & Sécurité
Introduction aux algorithmes de bandit

LE Doan Phuoc
DAU Nguyen Minh Tuan
STI 4A TP5

Le 19 Mars 2024

Partie 1 : Implémentation et comparaison d'algorithmes de bandits

1. Implémentez en Python les algorithmes de la Figure 2 instanciés dans le cadre générique de la Figure 1.
   - Nous avons implémenté des codes sur Python et des fonctions d'algorithmes de bandits ci-dessous:

   + Fonction **Random:** Le rôle de la fonction est de sélectionner au hasard un bras (action) que l'agent doit tirer (exécuter) sans tenir compte des récompenses passées ou du nombre de fois où chaque bras a été tiré.

```python
def random_choice(Q, N):
    return np.random.randint(K)
```

   + Fonction **Epsilon_greedy:** En définissant un paramètre ε (EPSILON), l'algorithme explore aléatoirement d'autres options avec une probabilité ε pour éviter de rester coincé dans un optimum local, tout en exploitant l'option la plus connue avec une probabilité de 1-ε. Cette double approche permet à l'algorithme de continuer à apprendre sur l'environnement tout au long de son fonctionnement, améliorant ainsi ses estimations au fil du temps tout en tirant parti des connaissances accumulées pour obtenir de meilleures performances globales.

```python
def epsilon_greedy(Q, N):
    if np.random.rand() < EPSILON:
        return np.random.randint(K)
    else:
        return np.argmax(Q)
```

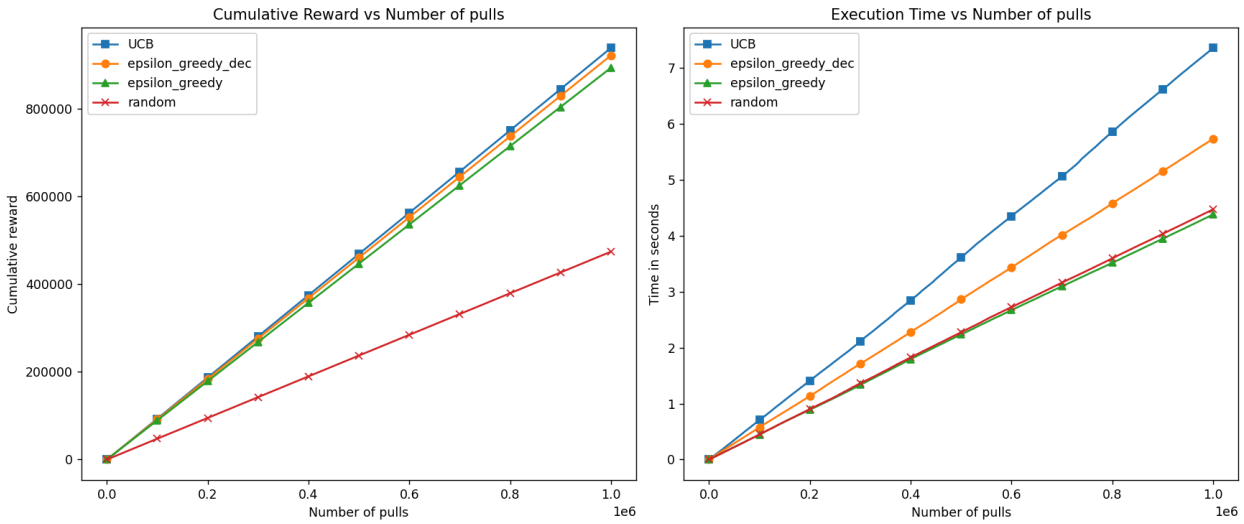
   + Fonction **Epsilon_greedy_decaying:** epsilon_greedy_decaying se présente comme une approche qui adapte le rythme d'exploration dans le temps et propose un équilibre qui passe de l'exploration à l'exploitation de manière contrôlée

```python
def epsilon_greedy_decaying(Q, N, t):
    epsilon_t = 1 / np.log(t + 2)
    if np.random.rand() < epsilon_t:
        return np.random.randint(K)
    else:
        return np.argmax(Q)
```

+ Fonction **UCB:** La fonction UCB adopte une approche empirique de la sélection des actions en considérant à la fois les récompenses estimées de chaque bras (codées en Q) et l'incertitude ou la variance associée à ces estimations.

```python
def UCB(Q, N, t):
    confidence = np.sqrt(2 * np.log(t + 1) / (N + 1e-5))
    return np.argmax(Q + confidence)
```

2. Faites un petit benchmark pour comparer vos algorithmes, par rapport aux critères *cumulative reward* et *temps d'exécution* et générez des figures.
   - Pour générer le figure, on execute le fichier **bandit.py** et nous avons defini des constants comme :
     + **Number of pulls :** 1.000.000
     + **Number of runs :** 10
   - Les figures :



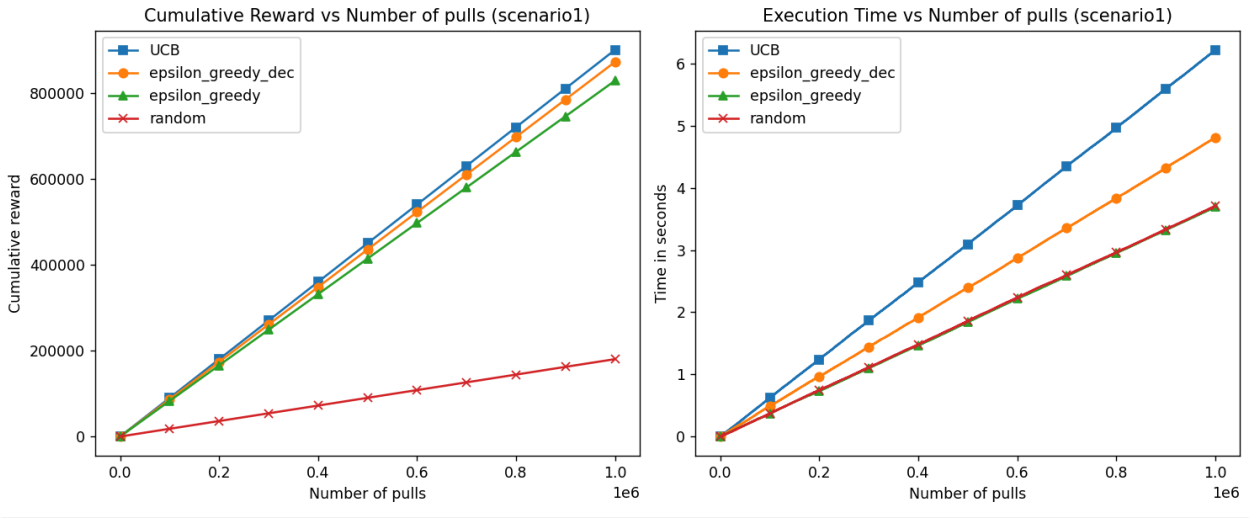
3. Ajoutez au moins 2 couples de figures comme dans le point précédent, pour 2 scénarios de bandits (K, μ1, . . . , μK) différents : un pour lequel UCB donne un cumulative reward un peu meilleur que les autres (comme dans mon exemple) et un autre pour lequel tous les algos sont plus ou moins pareils (voire les algos basés sur ε sont un peu meilleurs).

- Pour voir les deux scenarios, on execute le fichier **test2scenario.py**

- **Scénario 1** : UCB donne un cumulative reward un peu meilleur que les autres. J'ai fixé des rewards plus élevées pour quelques branches afin de potentiellement bénéficier à la stratégie d'exploration d'UCB.

```
# Scenario 1: UCB outperforms other algorithms
true_rewards_scenario1 = np.array([0.9] * 1 + [0.1] * (K - 1))  # Higher rewards closer to 1 to favor UCB
```

Ensuite, j'ai fait un benchmark pour comparer vos algorithmes, par rapport aux critères cumulative reward et temps d'exécution et générez des figures :



+ Dans ce scénario, la récompense (reward) cumulée d'UCB est légèrement meilleure que celle des autres algorithmes, ce qui correspond à la conception de l'algorithme d'UCB. UCB excelle lorsqu'il existe des différences significatives entre les récompenses des différentes armes, car elle équilibre efficacement l'exploration et l'exploitation.

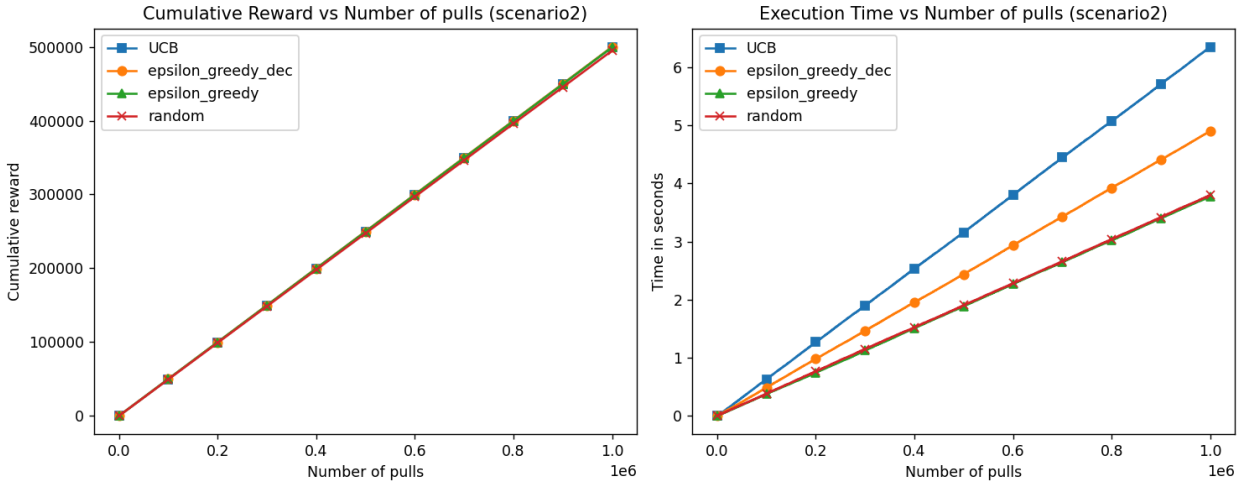
+ Les chiffres montrent qu'UCB est initialement plus exploratoire, ce qui peut conduire à une récompense cumulée plus élevée au fil du temps, car elle est plus susceptible de trouver et d'exploiter la meilleure option.

+ Les performances d'UCB se font au détriment du temps de calcul. Puisqu'il calcule des limites de confiance, ce qui implique des opérations logarithmiques, il est intrinsèquement plus lent. Ceci est confirmé par le graphique du temps d'exécution, où UCB prend systématiquement plus de temps.

- **Scénario 2 :** tous les algos sont plus ou moins pareils (voire les algos basés sur ε sont un peu meilleurs). J'ai modifié des rewards afin que les avantages de l'exploration soient moindres.

```
# Scenario 2: All algorithms perform similarly or ε-greedy algorithms are slightly better
true_rewards_scenario2 = np.array([0.5] * 9 + [0.45] * (K - 9))  # More uniform rewards to favor ε-greedy
```

Ensuite, j'ai fait un benchmark pour comparer vos algorithmes, par rapport aux critères cumulative reward et temps d'exécution et générez des figures :



+ Dans ce scénario, les algorithmes ε-gourmands fonctionnent de manière comparable ou légèrement meilleure qu'UCB, ce qui pourrait être dû au fait que les récompenses sont plus uniformes d'une branche à l'autre. Avec moins de variance dans les récompenses, le bénéfice de l'exploration d'UCB diminue et la simplicité de l'exploration de ε-gourmand devient plus efficace.

+ Les chiffres des récompenses cumulées reflètent cela en montrant que l'ε-gourmand et ses variations correspondent étroitement ou dépassent les performances d'UCB.

+ Le temps d'exécution pour UCB est encore plus long, comme prévu, car le calcul des limites de confiance ne dépend pas de la distribution des récompenses et reste constant dans sa complexité.

- **Conclusion :**

+ Les chiffres ont du sens car ils illustrent le compromis entre la profondeur d'exploration (UCB) et la vitesse d'exécution (ε-gourmande). Ils soulignent également comment différentes distributions de récompenses peuvent influencer l'efficacité d'un algorithme particulier.

+ Dans un environnement où les actions optimales ne sont pas claires et où il est important de trouver la meilleure option, l'exploration approfondie d'UCB est avantageuse, quoique plus lente.

+ En revanche, lorsque les récompenses sont similaires ou si la définition du problème permet une exploration plus rapide mais moins approfondie, les algorithmes ε-gourmands peuvent fonctionner aussi bien, voire mieux, en raison de leur simplicité et de leur exécution plus rapide.

Partie 2 : Proposition d'un nouveau protocole sécurisé

1. Avec vos propres mots, décrivez en quelques phrases le fonctionnement du protocole Samba publié dans le journal JAIR'22 et en tant que démo à ICDE'22, dont une vidéo est disponible sur Youtube.

- SAMBA est décrit comme un cadre d'apprentissage par renforcement qui facilite l'apprentissage collaboratif de manière sécurisée et distribuée. Il est conçu pour les situations dans lesquelles plusieurs propriétaires de données peuvent conserver leurs données stockées localement et collaborer via un serveur central sans compromettre la sécurité de leurs données.
- Le protocole utilise des méthodes cryptographiques pour garantir que même si chaque participant est capable d'exécuter des algorithmes de bandit multi-armés sur ses propres données, le processus global reste sécurisé et la récompense cumulée est optimisée.
- SAMBA maintient les niveaux de performances des algorithmes non sécurisés tout en offrant des garanties de sécurité.

2. Proposez un nouveau protocole sécurisé qui peut se différencier de SAMBA

Protocole sécurisé proposee : PvANet

Modèle de sécurité :

- *Confidentialité différentielle locale* : Chaque participant ajoute un bruit contrôlé à ses données avant de les partager, ce qui garantit que les informations individuelles ne peuvent être distinguées statistiquement.
- *Signatures numériques et chaîne de blocs* : Pour assurer l'intégrité des données et l'authentification des participants, chaque transaction dans le réseau est signée numériquement et enregistrée sur une chaîne de blocs décentralisée.

Hypothèses de distribution de données :

- *Réseau pair-à-pair (P2P)* : Les participants forment un réseau décentralisé sans autorité centrale. Les décisions et l'apprentissage sont effectués localement, avec des échanges de messages chiffrés directement entre les pairs.
- *Stockage distribué :* Les données sont répliquées et stockées à travers le réseau, en utilisant des techniques de sharding pour améliorer la disponibilité et la résilience.

Application des algorithmes de bandits :

- Les participants utilisent des algorithmes de bandits locaux qui sont adaptés par des mises à jour globales partagées via le réseau P2P. Ces mises à jour utilisent des techniques de transfert d'apprentissage pour améliorer l'apprentissage local sans compromettre la confidentialité des données.

Échanges de messages :

- *Demande de mise à jour globale :* Un participant envoie une demande chiffrée à travers le réseau P2P pour obtenir des mises à jour globales sans révéler ses propres données.
- *Partage de mise à jour globale :* Les mises à jour, sous forme de modèles ou de paramètres partiellement formés, sont partagées après application d'un bruit pour la confidentialité différentielle.

- Ce protocole vise à améliorer la sécurité, la flexibilité et l'efficacité dans la distribution et l'accès aux données dans des environnements réseau hétérogènes et dynamiques.