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

Radu Ciucanu
radu.ciucanu@insa-cvl.fr

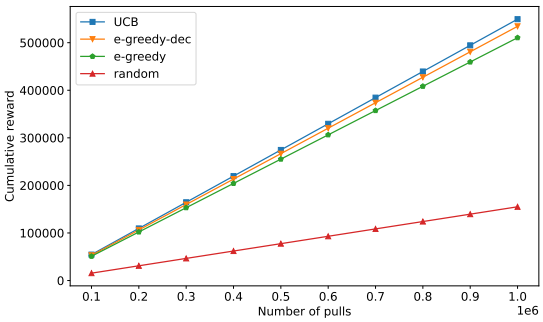
Travail à faire. Le sujet a deux parties :

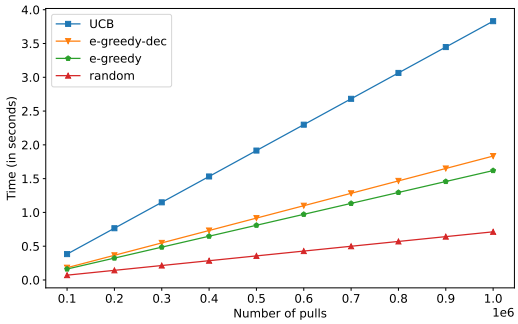
1. Une implémentation et comparaison de quelques algorithmes standard de bandits à partir de pseudocode donné, sans sécurité pour l'instant. Il suffit d'avoir parfaitement fait cette partie pour avoir 14/20.
2. Une partie plus ouverte pour éveiller la curiosité sur la proposition d'un nouveau protocole sécurisé. Cette partie vous permettrait d'améliorer votre note, en fonction de votre créativité et persévérance.

Rendu sur Celene. Votre rendu sur Celene sera une archive (par exemple `zip`) qui inclut : (i) votre code source (qui doit fonctionner sur un système d'exploitation de type Unix), et (ii) un compte rendu en format `pdf` qui explique comment utiliser votre code pour vérifier la qualité de votre travail, ainsi que les réponses aux différentes questions. Pour la préparation du compte rendu, je vous conseille d'utiliser Latex [1].

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.
2. Faites un petit benchmark pour comparer vos algorithmes, par rapport aux critères *cumulative reward* et temps d'exécution et générez des figures comme celles ci-dessous :





Quelques remarques :

— Quand vous faites des courbes, faites la moyenne de plusieurs runs afin d'avoir des courbes lisses. Par exemple, j'ai utilisé 100 runs pour générer les figures ci-dessus.

— Utilisez `matplotlib.pyplot` pour générer des figures en Python.

3. Dans le compte rendu, ajoutez au moins 2 couples de figures comme dans le point précédent, pour 2 scénarios de bandits $(K, \mu_1, \ldots, \mu_K)$ 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). Dans tous les cas, UCB est censé prendre plus de temps que les autres car ses calculs sont plus lourds. Discutez pourquoi vous pensez que vos figures ont du sens.

1. https://en.wikipedia.org/wiki/LaTeX

A bandit algorithm takes as **input** the budget N and the number of arms K, and gives as **output** the sum of observed rewards for all arms. The **unknown environment** of the bandit algorithm consists of K distributions associated to the K arms. We consider Bernoulli distributions with expected values $\mu_1, \ldots, \mu_K$ unknown to the learning agent. The agent has access to a reward function $pull(.)$ that can be called N times. For a chosen arm i, a call to the function $pull(i)$ randomly returns 0 or 1 according to the associated Bernoulli distribution, i.e., the probability of returning 1 is μ_i and the probability of returning 0 is $1-\mu_i$. The agent sequentially selects the N arms to be pulled with the goal of maximizing the sum of rewards. Given integers x and y such that $x \leqslant y$, by $[\![x, y]\!]$ we denote the list $[x, x+1, \ldots, y]$. By $[\![x]\!]$ we denote the list $[1, \ldots, x]$.

```
/* Initialization : pull each arm once & initialize variables */
for i ∈ [[K]]
    r ← pull(i)                                   /* Random reward for arm i */
    s_i ← r                                       /* Sum of observed rewards for arm i */
    n_i ← 1                                       /* Number of pulls of arm i */
/* Exploration-exploitation : pull one arm at each time step t */
for t ∈ [[K+1, N]]                                /* Only a budget of N − K is left */
    Choose i_m according to algorithm A
    r ← pull(i_m)
    s_{i_m} ← s_{i_m} + r
    n_{i_m} ← n_{i_m} + 1
return s_1 + ... + s_K
```

FIGURE 1 – Generic cumulative reward maximization with bandit algorithm $\mathcal{A}$.

Algorithm $\mathcal{A}$	*Strategy for choosing the arm at time t*
Random	**return** a random arm
ε-**greedy** with fixed or decreasing ε e.g., $\varepsilon = 0.1$ or $\varepsilon = \frac{1}{\ln(t^2)}$	**return** $\begin{cases} arg\,max_{i \in [\![K]\!]} \hat{\mu}_i \text{ , with probability } 1-\varepsilon & \textit{(exploit)} \\ \text{a random arm, with probability } \varepsilon & \textit{(explore)} \end{cases}$
UCB	**return** $arg\,max_{i \in [\![K]\!]} (\hat{\mu}_i + \sqrt{\frac{2\ln(t)}{n_i}})$

FIGURE 2 – Instantiations of Figure 1 for some cumulative reward maximization algorithms. We recall that for each arm $i \in [\![K]\!]$ at time t, n_i is the number of pulls, s_i is the sum of rewards, and $\hat{\mu}_i = \frac{s_i}{n_i}$ is the empirical mean.

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 [2] et en tant que démo à ICDE'22, dont une vidéo est disponible sur Youtube [3].
2. En s'inspirant de SAMBA et du survey [4] (notamment les sections sur la sécurité), imaginez un nouveau protocole sécurisé qui peut se différencier de SAMBA par exemple par rapport au modèle de sécurité, aux hypothèses de distribution de données entre les participants, aux applications des algorithmes de bandits, etc. Si vous pensez que vous avez trouvé un protocole qui tient la route, vous pouvez inclure dans le compte rendu des figures avec les messages échangés entre les participants, une courte description textuelle, du pseudocode, etc.

2. https://jair.org/index.php/jair/article/view/13163
3. https://www.youtube.com/watch?v=CSyVVmrhGH4
4. https://arxiv.org/pdf/1912.04977.pdf