Master MVA
Reinforcement Learning

TP 2: The Exploration-Exploitation Dilemma

Victor Busa
victor.busa@ens-paris-saclay.fr

December 1, 2017

1 Stochastic Multi-Armed Bandits on Simulated Data

1.1 Bernoulli bandit models

Comparison between UCB1 and NAIVE I used two different Multi-Armed Bandit problems. One complex problem with complexity $C_1(p) = 273.69$ and an easy problem whose complexity is $C_2(p) = 1.90$. All the arms are drawn from a Bernoulli distributions and the mean parameter of each arm is detailed in Table 1 and Table 2 below.

	Arm1	**Arm2**	**Arm3**	**Arm4**	**Arm5**	**Arm6**
Mean	0.2	0.65	**0.797**	0.794	0.795	**0.797**

Table 1: Mean parameter of each Bernoulli arms for the complex problem: $C_1(p) = 273.69$

	Arm1	**Arm2**	**Arm3**
Mean	0.3	**0.8**	0.4

Table 2: Mean parameter of each Bernoulli arms for the easy problem: $C_2(p) = 1.90$

Figure 1.1 and Figure 1.2 depict the observed cumulated regret averaged over 50 simulations of 60000 episodes each for `UCB1` with different values of α and the `NAIVE` algorithm for respectively the complex and the easy problem. We can see that, on average `UCB1` outperforms the `NAIVE` implementation on the easy problem as its regret is lower than the naive regret. Moreover, if we tune correctly the parameter α, `UCB1` highly outperforms the `NAIVE` algorithm on the easy problem as we can see for $\alpha = 0.5$.

For the complex problem we can see that `UCB1` still outperforms the `NAIVE` implementation and achieve the best results for low values of α. For $\alpha = 2$ the performance of `UCB1` looks very similar to the performance of the `NAIVE` implementation. This is due to the fact that the parameter α control the rate of exploration. Hence a higher α will favors exploration over exploitation. Large value of α will then decrease the performance of the `UCB1` algorithm.

Note: Intuitively, we might think that for the easy problem the `NAIVE` implementation and the `UCB1` implementation should behave in the same manner. Actually the bad performance of the `NAIVE` algorithm comes from the fact that if at the initialization we draw $[1, 0, 1]$, i.e a reward of 1 for Arm_1, 0 for Arm_2 and 1 for Arm_3 then we will subsequently not drawn the **second Arm** while it is the best in our example as $mean(Arm_2) = 0.8$. `UCB1` doesn't have this drawback as it balances between exploitation and exploration through the use of the α parameter.

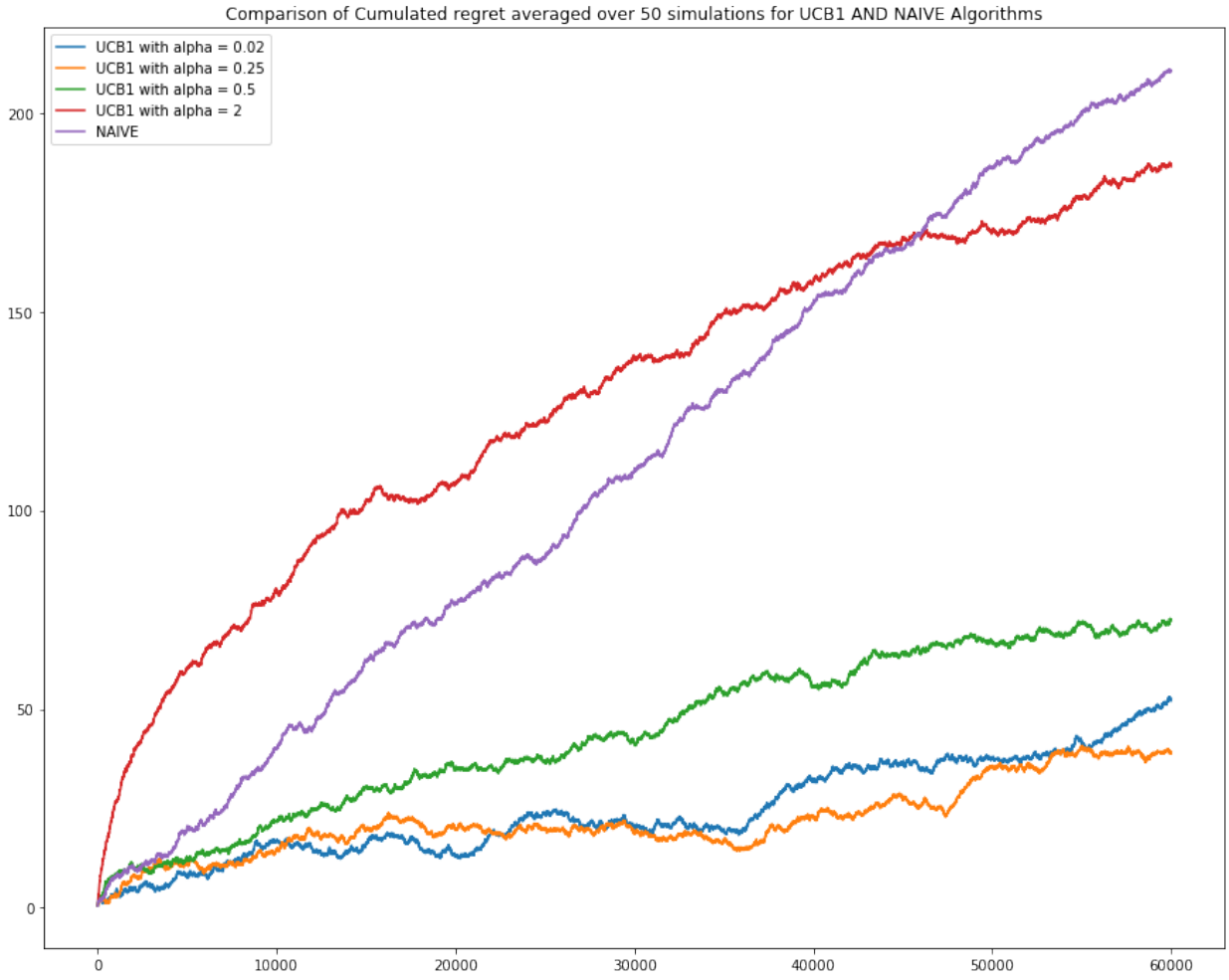


Figure 1.1: Cumulated regret averaged over 50 simulations of 60 000 episodes each for `UCB1` and `NAIVE` for different values of α on the complex problem

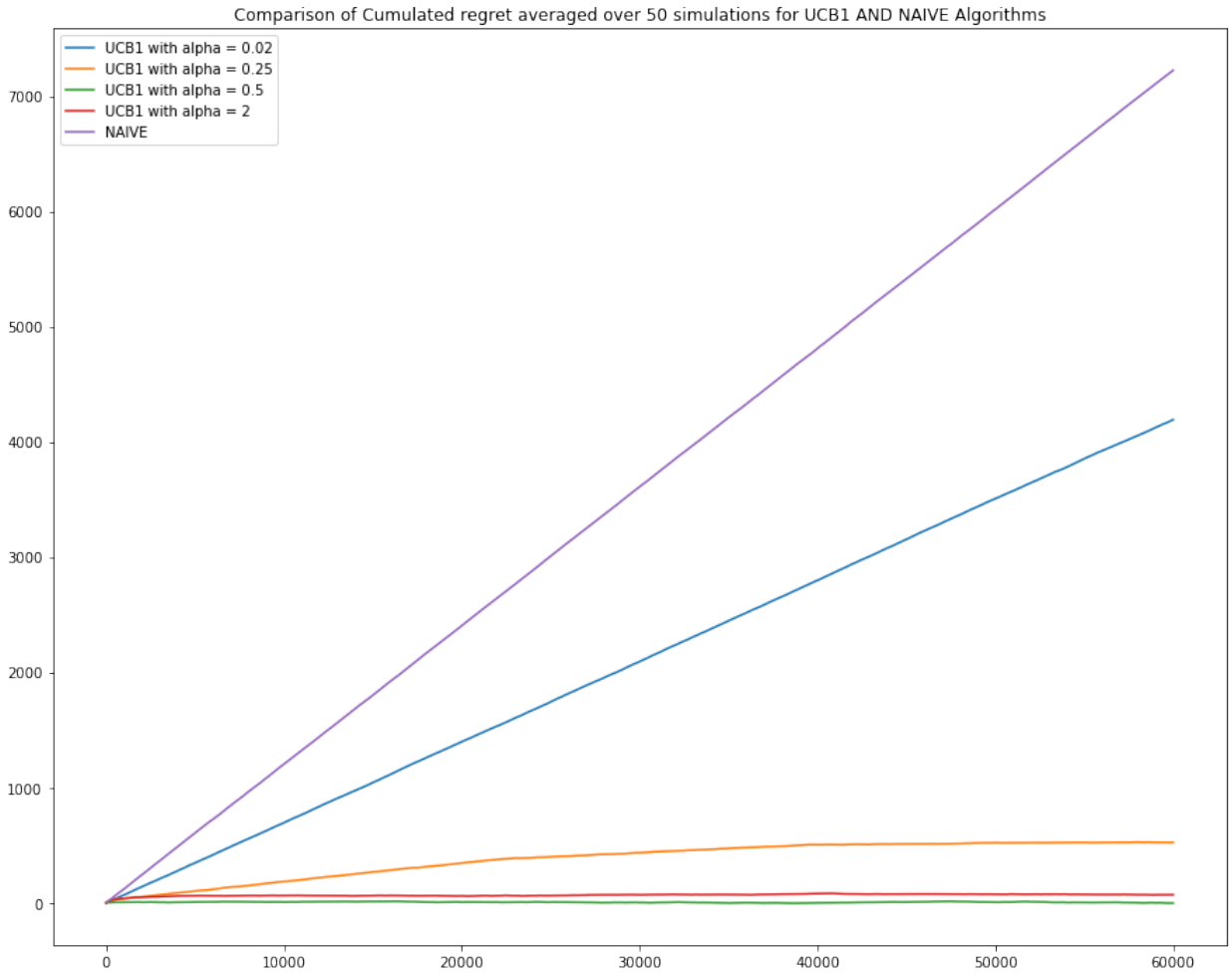
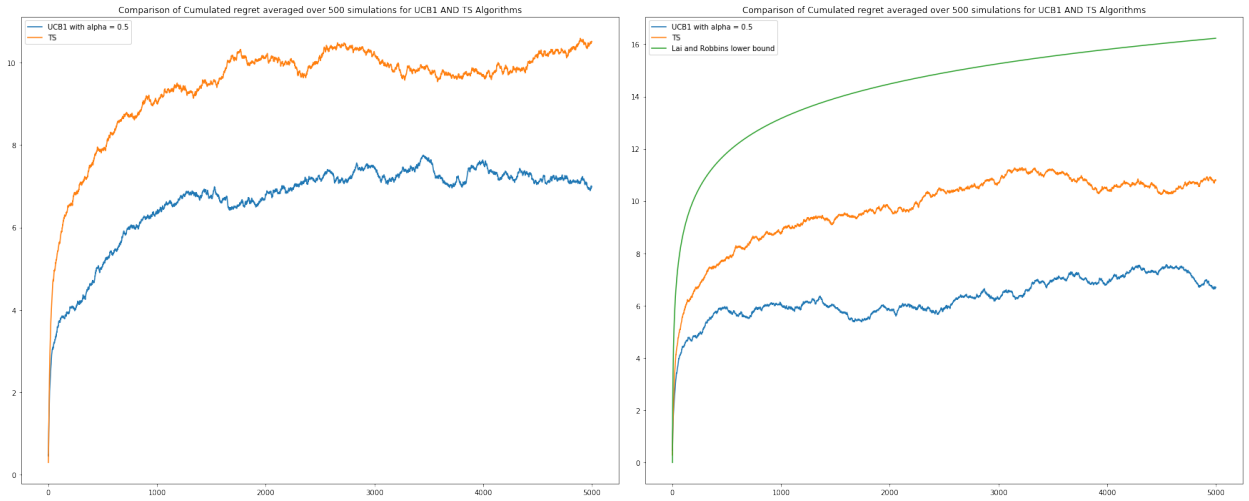


Figure 1.2: Cumulated regret averaged over 50 simulations of 60 000 episodes each for `UCB1` and `NAIVE` for different values of α on the easy problem

Q1: Comparison Between Thompson Sampling and UCB1
Problem 1 : 3 Bernoulli with parameters 0.3, 0.8 and 0.4 and complexity $C = 1.91$
Problem 2 : 4 Bernoulli with parameters 0.33, 0.34, 0.36, 0.35 and complexity $C = 83.9$

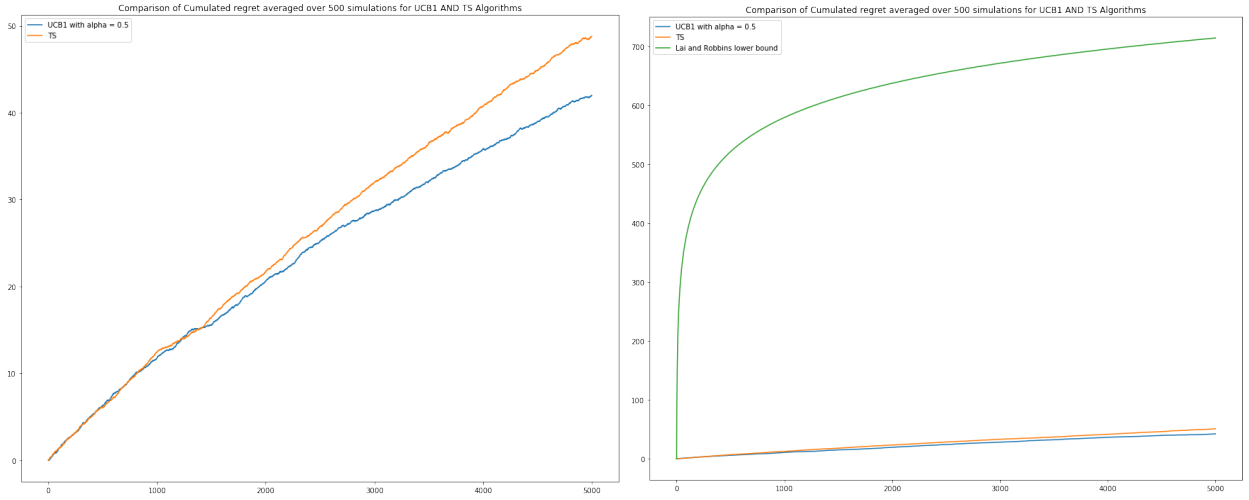
For easy Multi-Armed Bandit problems and for a well-suited value of α, on average, `UCB1` algorithm outperforms the `Thompson Sampling` algorithm. The Figure 1.3 depicts the averaged cumulated regret computed over 500 simulations of 5000 episodes with $\alpha = 0.5$ for **Problem 1**.



(a) First run without Lai Robbins bound (b) Second run with Lai Robbins bound

Figure 1.3: Averaged cumulated reward over 500 simulations for `UCB1` (Blue curve) and `Thompson Sampling` (Orange curve) on Problem 1

For complex problems such as **Problem 2**, `UCB1` still slightly performs better than `Thompson Sampling` for well-suited value of α. What is striking is the fact that the Lai & Robbins lower bound appears to be above both curves. This can be explained by the fact that the Lai & Robbins bound is an **asymptotic bound** and by the fact that the bound doesn't hold for **multiparametric** distributions.



(a) First run without Lai Robbins bound (b) Second run with Lai Robbins bound

Figure 1.4: Averaged cumulated regret over 500 simulations for `UCB1` (Blue curve) and `Thomson Sampling` (Orange curve) on Problem 2

1.2 Non-parametric bandits (bounded rewards)

Q2 Adaptation of Thompson Sampling

We can adapt Thompson Sampling algorithm so that it works for general stochastic bandits. The idea is to performs a Bernoulli trial with success probability $\widetilde{r}_t$ where $\widetilde{r}_t$ is the reward observed at time t from pulling the arm a_t. From this Bernoulli trial we observe $r_t \in \{0, 1\}$ and we then proceed by updating N and S based on the traditional Thompson Sampling algorithm.

For this question we consider the model in table 3

Arm	1	2	3	4	5	6	7	8	9
Type	exp(2)	exp(3)	exp(4)	exp(5)	beta(3,9)	beta(4,8)	beta(5,7)	beta(6,6)	Ber(0.8)
mean	0.57	0.35	0.25	0.20	0.25	0.33	0.42	0.5	**0.65**

Table 3: Multi-Armed bandit model

For this model we can see on Figure 1.5 that UCB1 outperforms Thompson sampling algorithm for $\alpha = 0.5$.

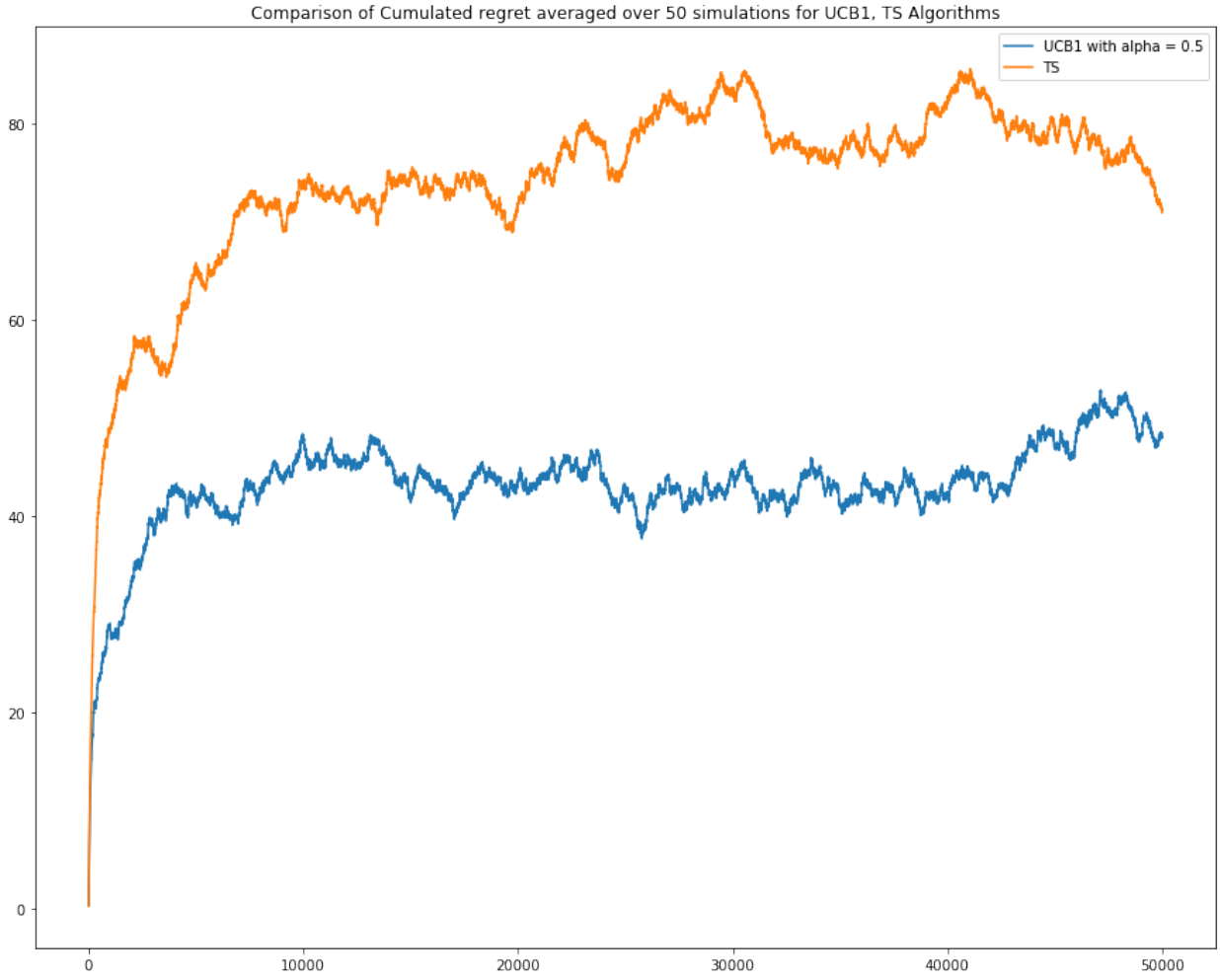


Figure 1.5: Cumulated regret averaged over 50 simulations of 50 000 episodes each for `UCB1` and `Thompson Sampling` for $\alpha = 0.5$

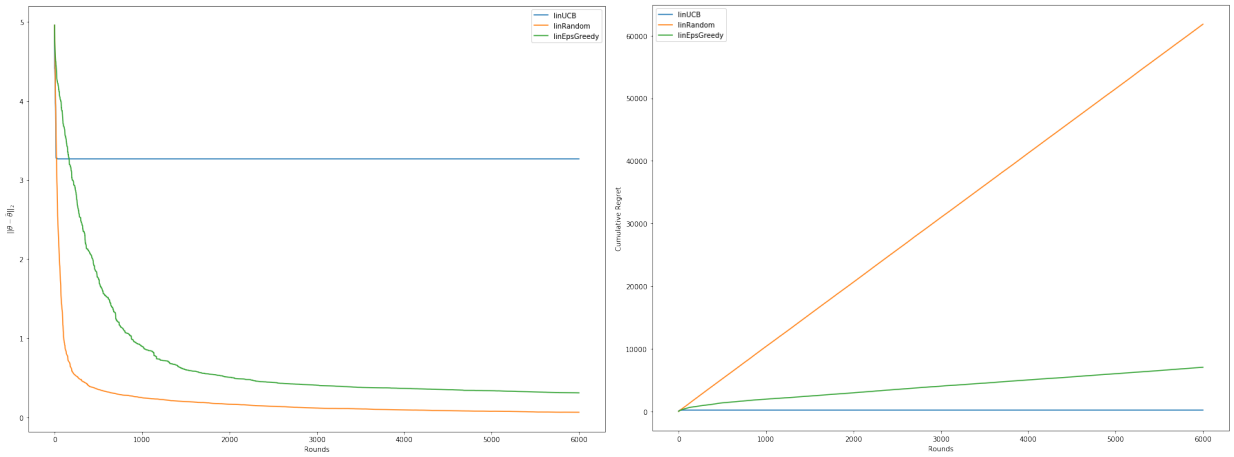
2 Linear Bandit on Real Data

I implemented two optimized version of `linUCB` algorithm (see Python code). The `linUCB2` implementation is slightly faster but prevent me from tracing the curve $\left\|\widehat{\theta}_t - \theta^\star\right\|_2$ in function of the number of iterations. That is why I choose to use the `linUCB` implementation in the remaining part.

I used the following formula to compute α:

$$\alpha = \sqrt{\frac{1}{2} log\ \frac{(2.T.K)}{\delta}}$$

where I choose $\delta = 10^{-3}$, $T = 6000$ (number of episode) and $K = 207$ (number of arms of the problem). Numerically I have $\alpha = 3.29$. In the remaining experiment I also set $\lambda = 2$ (regularization hyperparameter), $\epsilon = 0.1$ (for the greedy epsilon algorithm) and I averaged the result over **20 simulations**. Figure 2.1a and 2.1b show the result I obtained for the **ColdStartMovieLensModel**.



(a) $\left\|\widehat{\theta}_t - \theta^\star\right\|_2$ function of the number of iterations

(b) Cumulative averaged regret function of the number of iterations

Figure 2.1: ℓ_2 norm of $\widehat{\theta}$ w.r.t the true θ and Averaged cumulated regret over 20 simulations for `linUCB` (Blue curve) `linEpsGreedy` (Green curve) and `linRandom` (Orange curve) on the ColdStartMovieLensModel model

I've noticed that for certain value of α and for λ fixed, the linUCB algorithm can sometimes give really bad performances. The Table 4 inventories how many times we pull the 5 most pulled arms (computed using model.real_theta) for different values of α and different values of ϵ.

linUCB	$\alpha = 3.29$	(1: 5985)	(8: 1)	(14: 1)	(29: 1)	(42: 1)
	$\alpha = 0.2$	(29: 5994)	(42: 1)	(166: 1)	(171: 1)	(188: 1)
	$\alpha = 0.5$	(14: 5959)	(29: 32)	(24: 1)	(29:1)	(42: 1)
linEpsGreedy	$\epsilon = 0.1$	(1: 5425)	(93: 7)	(102: 7)	(196: 1)	(110: 6)
	$\epsilon = 0.3$	(1: 4240)	(120: 16)	(141: 16)	(164: 16)	(87: 15)
	$\epsilon = 0.01$	(1: 5708)	(42: 234)	(28: 2)	(50: 2)	(154: 2)

Table 4: (x^{th} best arm, number of pulls) for the 5 most pulled arms for different value of α and ϵ for one simulation of 6000 episodes

As we can see on the above table, the linUCB algorithm doesn't work at all for really bad value of α. For example we can see that for $\alpha = 0.5$ the most pulled arm is the 14^{th} best-arm with 5959 pulls and for $\alpha = 0.2$, the most pulled arm is the 29^{th} best-arm with 5994 pulls. We can also clearly see that linEpsGreedy performs the best for low values of ϵ which is understandable as the higher ϵ the more the algorithm will explore. But after a certain number of iterations it will know which arm(s) is (are) the best to pick.

According to Table 4, $\alpha = 3.29$, $\epsilon = 0.01$ and $\lambda = 2$ gives us good performance. Taking even higher value for α will give us slightly better performance.

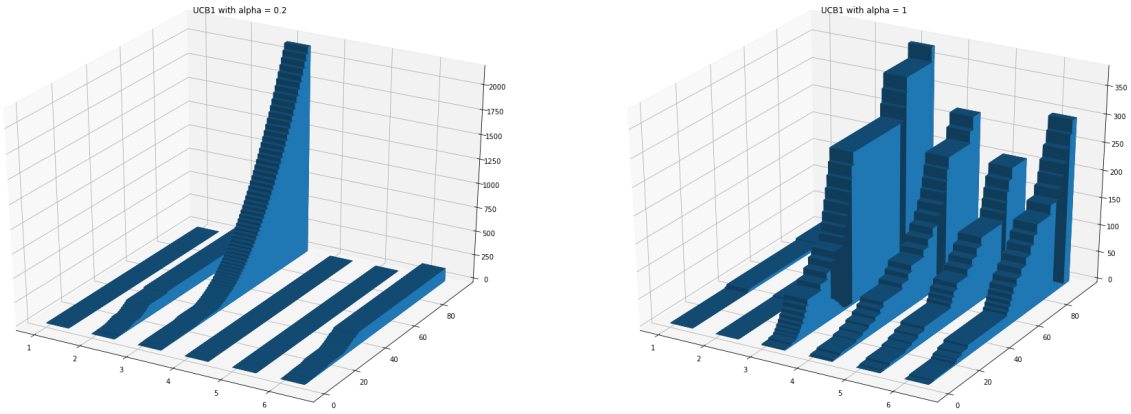
3 Annexe

histograms plot for UCB1
I wanted to see what the 3D histograms of cumulated rewards looks like. To do so I used the following complex model ($C = 273.69$):

Bernoulli arm	1	2	3	4	5	6
mean	0.2	0.65	0.797	0.794	0.795	0.797

Table 5: Multi-Armed bandit model

It appears that `UBC1` performs the best when α is in the range $[0.2, 0.5]$, while, when α is too low the algorithm will tends to have the same behaviour as the Naive implementation as it can be shown mathematically ($\alpha = 0$is equivalent to the `NAIVE` algorithm). When α is large `UCB1` will favor exploration over exploitation and hence the overall performance of `UCB1` can deplete a lot. The Figure 3.1 depicts the 3D histograms for $\alpha = 0.2$ and $\alpha = 1$



(a) $\alpha = 0.2$ (b) $\alpha = 1$

Figure 3.1: Cumulated reward over 100 episodes for `UCB1`

α	**Arm1**	**Arm2**	**Arm3**	**Arm4**	**Arm5**	**Arm6**
0.2	1	16	**65**	1	1	16
1	4	2	27	23	20	24

Table 6: Number of times each arm is drawn for different value of α

We can see that for $\alpha = 0.2$ (good choice of α) the algorithm quickly detects which arm is the better (Arm3 here) while when $\alpha = 1$ for the same number of iterations the algorithm still struggle to distinguish the best Arm between Arm_3, Arm_4, Arm_5 and Arm_6 and for a very good reason their means are really close to each other and the algorithm will tend to explore a lot more.