Research Project of Reinforcement Learning

A movie recommendation system based on

Multi-action bandits

Simon Bussy & Antoine Recanati
simon.bussy@ens-cachan.fr , antoine.recanati@ens-cachan.fr

supervised by
Emilie Kaufmann , **emilie.kaufmann@inria.fr**

January 2015

1. Introduction

In the context of a movie recommendation system, one can imagine that instead of recommending just one movie to a user, we recommend a bunch of movies, and qualify our recommendation as good if the user clicks on one of the films. This situation can be modeled by a multi-armed bandit models in which at each round, a certain number of arms, let say m, are drawn simultaneously.
This problem is a particular case of what is sometimes called in the literature a combinatorial bandit problem. The goal of this project was to understand this recent set up and then to adapt the classical stochastic MAB algorithms (UCB and Thompson Sampling) to this context.
We will present in the next sections the way we tried to address this problem and the different types of feedback we tried, depending on the full or partial observation of the rewards, and two different reward functions.

Table of contents

1. Introduction **1**

2. Multi-action bandits **3**
2.1. CUCB Algorithm . . . 3
2.2. Thompson sampling . . . 3
2.3. Complexity . . . 4

3. Reward : sum of rewards **4**
3.1. Oracle . . . 4
3.2. Full-observations set up . . . 4
Thompson sampling algorithm . . . 4
CUCB algorithm . . . 5
3.3. Only global reward observed set up . . . 5
Thompson sampling algorithm . . . 5

4. Reward : maximum of rewards **6**
4.1. Oracle . . . 6
4.2. Full-observations set up . . . 6
4.3. Only global reward observed set up . . . 6
5. Results and discussion . . . 7

6. Conclusion **11**

References **11**

7. Appendices **11**
7.1. Derivation of a theoretical upper bound on regret . . . 11
Figures . . . 12

2. Multi-action bandits

In a combinatorial multi-armed bandit (CMAB) problem, subsets of base arms with unknown distributions form super arms. In each round, a super arm is played and the base arms contained in the super arm are played. We will discribe in this section how we can adapt the classical UCB and Thomson sampling algorithm to this framework. In the next sections, we will discuss the cases where the outcomes of all base arms are observed or not, and where the reward obtained by pulling a super arm is first the sum of the base arms composing the super arm, or the maximum of base arms rewards.

2.1. CUCB Algorithm

We present here the CUCB algorithm, which is the adapted UCB in the framework introduced. The combinatorial multi-armed bandit problem consists of m base arms. Let denote $\mu = (\mu_1, ..., \mu_m)$ the vector of expectations of all base arms. In our applications, we will take m=10 and each base arm will follow a Bernoulli distribution modeling whether or not a user clicks on the movie representing by the arm.
Let $\mathscr{S}$ denote the set of all possible super arms that can be played in the CMAB problem. For example, $\mathscr{S}$ could be the set of all subsets of base arms containing 3 base arms. In each round, one of the super arms $S \in \mathscr{S}$ is selected and played.
The CUCB algorithm first initialize all variables in m rounds and then maintain an empirical mean for each arm as well as the total number of times each arm is played. Then we just play the super arm returned by an oracle, and we will further explain how to compute the oracle associated with the different setups we considered.

Then, we could describe the algorithm as follows :

Algorithm 1 CUCB algorithm

```
1: For each arm $i$, maintain (1) $T_i$ the total number of times arm $i$ is played so far, (2) $S_i$ the sum of arm $i$ rewards observed so far.
2: $t = 0$
3: For each arm $i$, play an arbitrary super arm $S \in \mathscr{S}$ such that $i \in S$ and update variables $T_j$ and $S_j$ for all $j \in S$ (including $i$).
4: $t \leftarrow m$
5: while true do
6:     $t \leftarrow t + 1$
7:     For each arm $i$, set $\overline{\mu_i} = \hat{\mu_i} + \sqrt{\frac{\alpha ln(t)}{T_i}}$
8:     S = Oracle($\overline{\mu_1}, ..., \overline{\mu_m}$)
9:     Play S update $T_i(t+1)$ and $S_i(t+1)$
10: end while
```

2.2. Thompson sampling

Thompson sampling (also known as probability matching) is a simple, randomized, bayesian algorithm. The (bayesian) idea is to consider, for each arm, the posterior probability $P(\theta_i | x_{i,1}, ..., x_{i,T_i(t)})$ on the mean θ_i of its (Bernouilli, in our case) distribution, given the rewards $x_{i,1}, ..., x_{i,T_i(t)}$ observed so far for this arm ($T_i(t)$ is the number of times arm a has been pulled at time t). The idea for a classical MAB is to pull, at each step, the arm with highest probability of being optimal. To do so, the (randomization) heuristic is to draw a sample $\theta_i(t)$ from the posterior distribution, and pull the arm which has the highest sample $\theta_i(t)$ at time t. When an arm has been pulled few times, its posterior is widespread and therefore the sampling is

Name	nArms	nSuperArms	nArms/SuperArm	Complexity term
VeryEasy	20	10	3	7
Easy	10	$\binom{10}{3}$	3	80
Hard	10	$\binom{10}{3}$	3	558
VeryHard	10	$\binom{10}{3}$	3	833

TABLE 1: arms of different complexity

likely to end up pulling this arm again, which will narrow the posterior at the next step. When an arm has been pulled enough times, its posterior is peaked around its actual mean. This enables this algorithm to deal efficiently with the exploration-exploitation dilemma.
Choosing a uniform prior, $\theta_i \sim \mathscr{U}([0,1])$, results in a beta distribution for the posterior

$$\theta_i | x_{i,1}, ..., x_{i,T_i(t)} \sim Beta(S_i(t) + 1, T_i(t) - S_i(t) + 1) \tag{1}$$

where $S_i(t) = \sum_{s=1}^{T_i(t)} x_{i,s}$ is the cumulative reward of this arm at time t.

The Thompson sampling algorithm adapted to the CMAB framework in the different cases will be presented in detail in the next sections.

2.3. Complexity

Let us recall the complexity term introduced by Lai and Robbins for a Multi-Arm Bandit,

$$\sum_{i \neq i^*} \frac{(\mu^* - \mu_i)}{KL(\nu_i, \nu^*)} \tag{2}$$

where ν_i are the distributions of the arms of the bandit, μ_i, their means, $KL(\nu_i, \nu^*)$ the Kullback-Leibler divergence between the distributions ν_i and ν^*, and the $*$ refers to the optimal arm.
For our multi-action bandit, we defined its complexity as the complexity of the naive problem where each super arm is considered as a simple arm in a classical MAB. We tried the algorithms against problems of different complexity, whose characteristics are given in table 1. VeryEasy is a multi-action bandit with 1 arm significantly higher than the others ($\mu^* = 0.8$, $\mu_{other} \simeq 0.2$, and only one super arm among the 10 super arms to which this arm belongs. For all other problems, all super arms of size 3 are considered. In the Easy problem, 3 arms are significantly higher than the others. In the Hard problem, 1 arm is significantly higher than the others, and in the VeryHard problem, which is probably the most common for applications, all arms are within the same range ($\mu \sim 0.1 + 0.15 * \mathscr{U}([0,1])$.

3. Reward : sum of rewards

3.1. Oracle

The oracle, *i.e.* the function of whom the *argmax* is chosen as the best super arm, is the estimation of the reward of the super arm :

$$r_\mu(S) = \sum_{i \in S} \mu_i \tag{3}$$

3.2. Full-observations set up

Thompson sampling algorithm

Let us come back to our problem where at each round, one has to chose one among several super arms to pull, each composed of m arms. The case where, at each round, the individual rewards of all arms in the super arm pulled are observed, and the reward of the superarm is the sum of those individual rewards [1] is the case described in [2]. Thompson sampling is easy to adapt to this full-observation case, yielding the algorithm described in Algorithm 2. If the sets considered are all possible sets of size m, the algorithm boils down to Thompson sampling for a classical Bernouilli MAB, except that at each round, the m arms with highest samples θ_i are simultaneously pulled.

Algorithm 2 Thompson sampling, full-obs., sum of rewards set up

```
1: For each arm $i$, maintain (1) $T_i$ the total number of times arm $i$ is played so far, (2) $S_i$ the sum of arm
   $i$ rewards observed so far.
2: $t = 0$
3: For each arm $i$, play an arbitrary super arm $\mathscr{M} \in \mathscr{S}$ such that $i \in \mathscr{M}$ and update variables $T_j$ and $S_j$
   for all $j \in \mathscr{M}$ (including $i$).
4: $t \leftarrow m$
5: for all arms $i$ do
6:     sample $\theta_i \sim Beta(1 + S_i(t), 1 + T_i(t) - S_i(t))$
7: end for
8: while true do
9:     $t \leftarrow t + 1$
10:    play $\mathscr{M}(t) = argmax_{\mathscr{M} \in \mathscr{S}} \sum_{i \in \mathscr{M}} \theta_i$
11:    for all arms $i \in \mathscr{M}(t)$ do
12:        observe rewards $x_i$
13:        update $T_i(t + 1)$ and $S_i(t + 1)$
14:        sample $\theta_i \sim Beta(1 + S_i(t), 1 + T_i(t) - S_i(t))$
15:    end for
16: end while
```

CUCB algorithm

In this case, we will use the CUCB algorithm discribed in section 2 where the reward associated to the super arm S is the sum of outcomes of the base arms in S.

3.3. Only global reward observed set up

Thompson sampling algorithm

We adapted the algorithm by attributing the reward of the super arm pulled to all the arms of which it is composed. However, since we model the arms with Bernoulli laws, we restrict their mean to $[0, 1]$ and therefore attribute the mean of the rewards among the super arm to each arm, that is, the algorithm is the same than with max-of-rewards (see in next section, algorithm 3), except that the oracle is changed and the attribution of rewards becomes :

$$S_i(t + 1) \leftarrow S_i(t) + \frac{1}{M} \sum_{j \in \mathscr{M}} x_j \tag{4}$$

where M is the number of arms per super arm. This may seem more perilous than in the max-of-rewards set up. Indeed, in that other set up, the “bad arms” surrounded by good arms are overestimated, since they would receive a reward 1 when actually giving 0 if the others arms in a common super arm gave 1. The “good arms”, however, are little biased. Here, not only the “bad arms” may be overestimated, but the “good” ones may be underestimated, and the results may not be very different from the naive strategy where each super arm is modeled as just one simple arm. Still, the results are significantly better than with the naive strategy,

1. or if the optimal arm for the actual reward is the same than the one for the reward so defined (sum of individual rewards).

as we will see further, because it requires less exploration time.
CUCB is adapted in the same fashion.

4. Reward : maximum of rewards

4.1. Oracle

For a super arm S composed of M arms following a Bernoulli distribution of means μ_i, $i = 1, ..., M$, the probability to obtain a reward $y_S = 0$, is the probability that all the interior arms simultaneously get a reward $x_i = 0$, that is :

$$P(y_S = 0) = \prod_{i \in S}(1 - \mu_i)$$

Therefore, the probability of getting a reward $y_S = 1$ is :

$$r_\mu(S) = 1 - \prod_{i \in S}(1 - \mu_i) \tag{5}$$

This is therefore the function we use to select the super arm in this set up. A few remarks :
- this oracle is exact : if we have access to the actual values of μ_i, we know which arm is most likely to give the maximal reward.
- if $\mu_i\prime \geq \mu_i$ for all $i \in S$, then $r_{\mu\prime}(S) \geq r_\mu(S)$.
- if all possible super arms are considered, the optimal super arm is the one composed of the m best arms. Otherwise, it may not even include the best arm, *e.g.*, $SuperArm1 = \{0.5, 0.5, 0.5\}$ is better than $SuperArm2 = \{0.6, 0., 0.\}$.

4.2. Full-observations set up

In our set up where the reward of a super arm is the maximum reward among all arms (*i.e.*, in the movie recommendation context, has the user clicked on at least one of the m movies advertised), the algorithm is unchanged, except for the choice of the best super arm, which is no longer the *argmax* of the oracle $\sum_{i \in \mathscr{M}} \theta_i$, but of $Oracle(\mathscr{M}) = 1 - \Pi_{i \in \mathscr{M}}(1 - \theta_i)$.

4.3. Only global reward observed set up

This scenario is of most interest in the movie recommendation framework. The only information available is : has the user clicked on at least one of the movies in the bunch of movies, the goal being to find the best super arm. One way to do it is to naively apply UCB or Thompson sampling to the super arms, as if they were simple arms from which we can observe the reward. However, the knowledge of the composition of the super arms enables us to improve the results, by assessing the quality of an arm through the rewards of the super arms to which it belongs, and then reassessing the quality of the super arms according to the quality of the arms they are made of.
We used this additional knowledge by updating the posterior of each arm in the super arm pulled as if they all had the reward of the super arm. That is, if all arms give 0, the super arm's reward is 0 and each arm's posterior is updated accordingly. In this special case, we know from the super arm's reward that the actual arms' rewards are all 0. If one or more arms give 1, then we update the arms posterior as if they all had given 1. This may overestimate the distribution of bad arms often surrounded by good arms, but this may not be such a bad issue since we only are interested in the super arms rewards. The resulting algorithm is given in Algorithm 3.

Algorithm 3 Thompson sampling, partial-obs., max of rewards set up

```
1: For each arm $i$, maintain (1) $T_i$ the total number of times arm $i$ is played so far, (2) $S_i$ the sum of
   rewards observed so far in all super arms containing $i$.
2: $t = 0$
3: For each arm $i$, play an arbitrary super arm $\mathscr{M} \in \mathscr{S}$ such that $i \in \mathscr{M}$, observe its reward and update
   variables $T_j$ and $S_j$ for all $j \in \mathscr{M}$ (including $i$).
4: $t \leftarrow m$
5: for all arms $i$ do
6:     sample $\theta_i \sim Beta(1 + S_i(t), 1 + T_i(t) - S_i(t))$
7: end for
8: while true do
9:     $t \leftarrow t + 1$
10:    play $\mathscr{M}(t) = argmax_{\mathscr{M} \in \mathscr{S}} 1 - \prod_{i \in \mathscr{M}}(1 - \theta_i)$
11:    observe $r(\mathscr{M}(t))$
12:    for all arms $i \in \mathscr{M}(t)$ do
13:        $T_i(t + 1) \leftarrow T_i(t) + 1$
14:        $S_i(t + 1) \leftarrow S_i(t + 1) + r(\mathscr{M}(t))$
15:        sample $\theta_i \sim Beta(1 + S_i(t), 1 + T_i(t) - S_i(t))$
16:    end for
17: end while
```

5. Results and discussion

In figure 1 are shown the regret curves for the VeryEasy and the VeryHard problems, for the naive (dotted curves that just appear like lines with bigger width), partial-observation and full-observation set up, for Thompson sampling and CUCB with different values of α, all of this when the reward is the sum of individual rewards among base arms. All curves are displayed in larger in appendix. The "blind" term refers to the naive strategy.

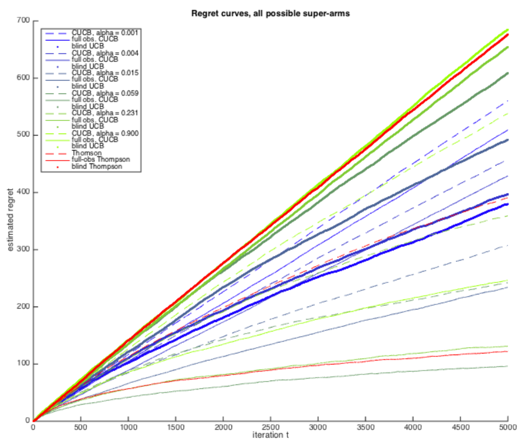
Figures 2 and 3 show similar results when the reward of a super arm is the maximum among the rewards of the base arms.

For complex problems, as expected, the full observation set up (all rewards in the set observed) yields better results than partial observation (only total reward of the set), which itself does better than the naive strategy. What differs is the speed of convergence, or in other words, the time needed for exploration. Let us take the case where all possible super arms are considered. There are $\binom{m}{M}$ super arms if M is the number of arms per super arm, and m the total number of arms. Therefore, the naive strategy needs a lot more time to explore all super arms.

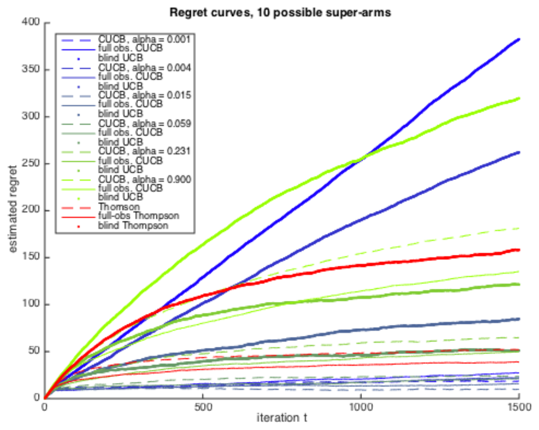
For our VeryEasy problem, however, the full-observation set up is sometimes less efficient than the partial-information one. This is counter- intuitive, but can be explained by the fact that for this problem, one super arm is high above the others, and therefore there is no need for much exploration which happens to be a waste of time. Still, for the rest of our test examples, the curves are as expected (see figure 2 for example, where the naive curves are not plotted for clarity).

The difference of regret between (naive and) partial-information and full-observation therefore seems to increase with complexity. Still, the difference may be more striking for intermediate cases, in between the (very easy) case where all algorithms do well, and the (very hard) case where they all have important regret. This is illustrated on figure 3, where we can see on the close up on the Easy problem how different the convergence of the different set up are (focus on the red curves (Thompson sampling), for example) compared to the VeryEasy and VeryHard problems.

Another point is that the naive strategy sometimes does better than CUCB for the worse values of α on complex problems. Thus, one should chose carefully the value of α. The results obtained from Thompson



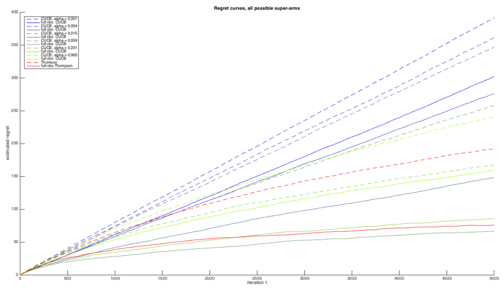
(a) VeryHard



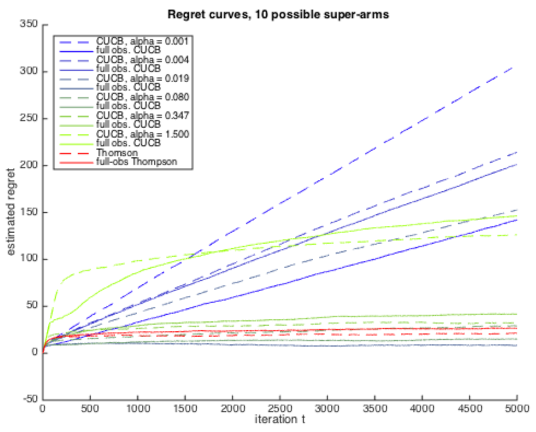
(b) VeryEasy

FIGURE 1: Reward : sum of rewards, $N_{simu} = 200$

sampling, however, are in agreement our expectations : the improvement is clear between naive and partial-observation strategies, and between partial-observation and full-observation in almost all cases (except a slight inversion between partial-observation and full-observation for the VeryEasy problem).
Thompson sampling overall does better than CUCB for most values of α, but in general, CUCB played with the best value of α yields the best result.

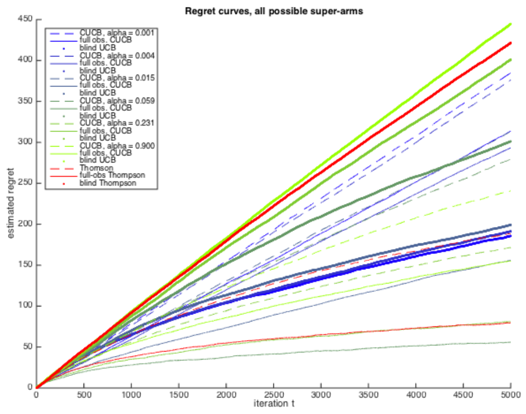


(a) VeryHard

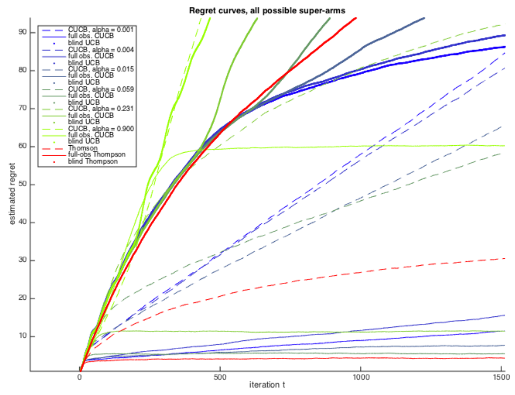


(b) VeryEasy

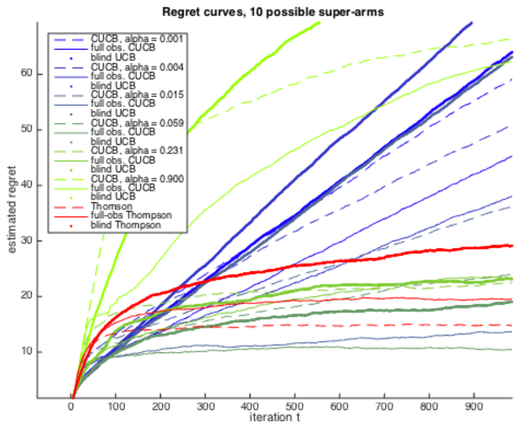
FIGURE 2: Reward = max of rewards, $N_{simu} = 200$



(a) VeryHard



(b) close up on Easy



(c) VeryEasy

FIGURE 3: Reward = max of rewards, $N_{simu} = 200$

6. Conclusion

We first explain the general combinatorial multi-armed bandit framework and then provide CUCB algorithm as well as Thomson sampling algorithm, that are adapted versions for this particular setup. These algorithms demonstrate their effectiveness to be used in a movie recommendation system in practice for all the different cases considered : for different reward functions (linear or not) and with all or partial informations available.
This research project was for us the occasion to explore one particular recent Bandit problem that already has various applications, including in the recommendation systems framework.

References

[1] Wei Chen, Yajun Wang and Yang Yuan, Combinatorial Multi-Armed Bandit : General Framework, Results and Applications.

[2] Audrey Durand and Christian Gagné, Thomson Samplning for Combinatorial Bandits and Its Application to Online Feature Selection.

7. Appendices

7.1. Derivation of a theoretical upper bound on regret

In [1] is derived the following upper bound on the regret for CUCB (with $\alpha = 2/3$) :

$$\left(\frac{6\log t}{(f^{-1}(\Delta_{min}))^2} + \frac{\pi^2}{3} + 1\right) m\Delta_{max} \tag{6}$$

It applies to our case where the oracle is exact and non-decreasing with μ, where Δ_{min} (resp. Δ_{max}) is the difference between the oracle for the best and for the worse (resp. for the second best) super arm, and the *bounded smoothness function* f is a strictly increasing function such that, for any two expectation vectors μ and $\mu\prime$ (means of all arms), we have $\mid r_{\mu(S)} - r_{\mu\prime(S)} \mid \leq f(\Lambda)$ if $\max_{i\in S} \mid \mu_i - \mu_i\prime \mid \leq \Lambda$. Let us prove that $f(\Lambda) = (1+\Lambda)^M - 1$ is a bounded smoothness function. It is strictly increasing. Let us consider a set S, μ_1 the vector of means of the arms of this set, and μ_2 another vector of means for this set, with $\mu_2 = \mu_1 + \Delta\mu$. From $r(\mu) = \prod_{i=1}^M (1-\mu_i)$, we have :

$$|r(\mu_1) - r(\mu_2)| = \left|\prod_i (1 - \mu_{2,i} + \Delta\mu) - \prod_i (1 - \mu_{2,i})\right|$$

But,

$$\prod_{i=1}^M [(1-\mu_{2,i}) + \Delta\mu] = \sum_{k=0}^M \sum_{\mathscr{P}_M(i)} \prod_{j\in\mathscr{P}_M(i)} (1-\mu_{2,i}) \prod_{l\in\{1,\dots,M\}\setminus\mathscr{P}_M(i)} \Delta\mu_l$$

where $\mathscr{P}_M(i)$ are the sets of i elements in $\{1, ..., M\}$. Therefore,

$$|r(\mu_1) - r(\mu_2)| = \left|\sum_{k=1}^M \sum_{\mathscr{P}_M(i)} \prod_{j\in\mathscr{P}_M(i)} (1-\mu_{2,i}) \prod_{l\in\{1,\dots,M\}\setminus\mathscr{P}_M(i)} \Delta\mu_l\right|$$

Using that the absolute value of the sum is lower than the sum of absolute values, and the upper bounds $\mid 1 - \mu_{2,j} \mid \leq 1$, and $\mid \Delta\mu_l \mid \leq \|\Delta\mu\|_\infty$, gives :

$$|r(\mu_1) - r(\mu_2)| \leq \sum_{k=1}^M \binom{M}{k} \|\Delta\mu\|_\infty^{M-i+1} = (1 + \|\Delta\mu\|_\infty)^M - 1 \tag{7}$$

Using this function f together with equation 6 provides the bound on regret :

$$\left(\frac{6 \log t}{[(1+\Delta_{min})^{1/M} - 1]^2} + \frac{\pi^2}{3} + 1 \right) m\Delta_{max} \tag{8}$$

This upper-bound is far above from the regrets obtained with the algorithms, especially for complex problems, so we did not plot it as it was not on a too larger scale than the actual regret.

Figures

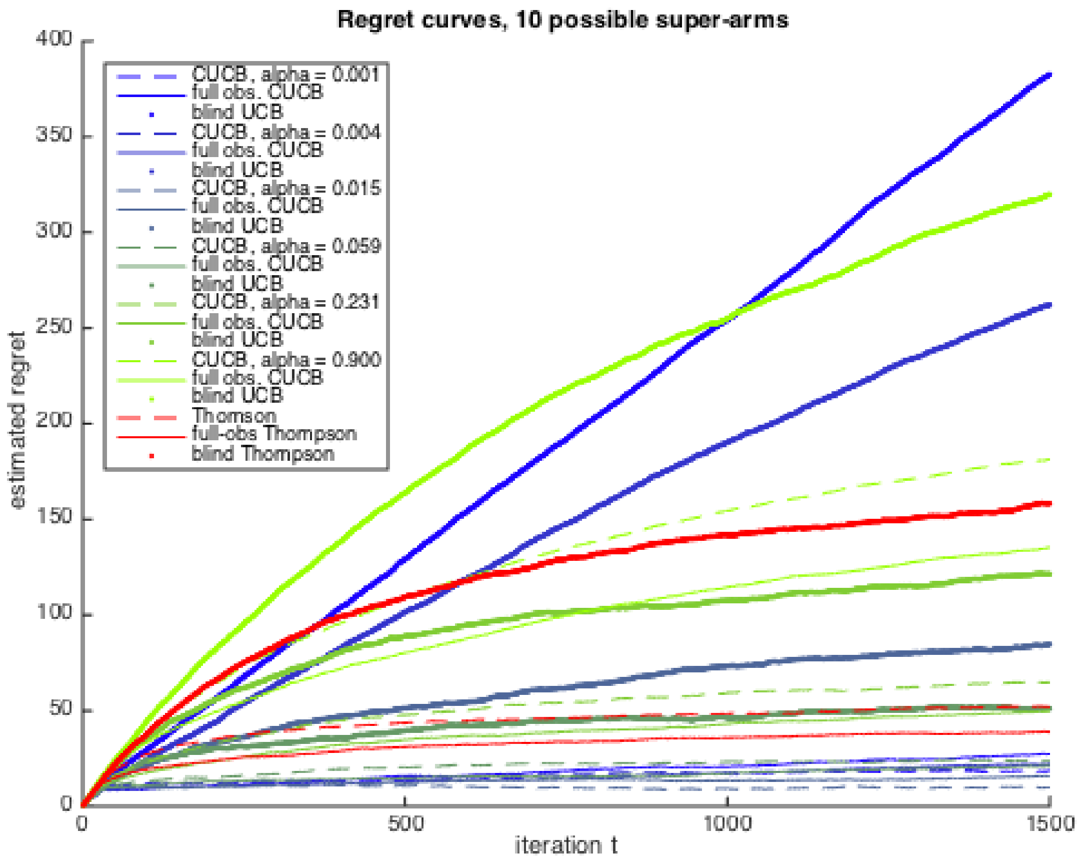


FIGURE 4: Reward = sum of rewards, VeryEasy, $N_{simu} = 200$

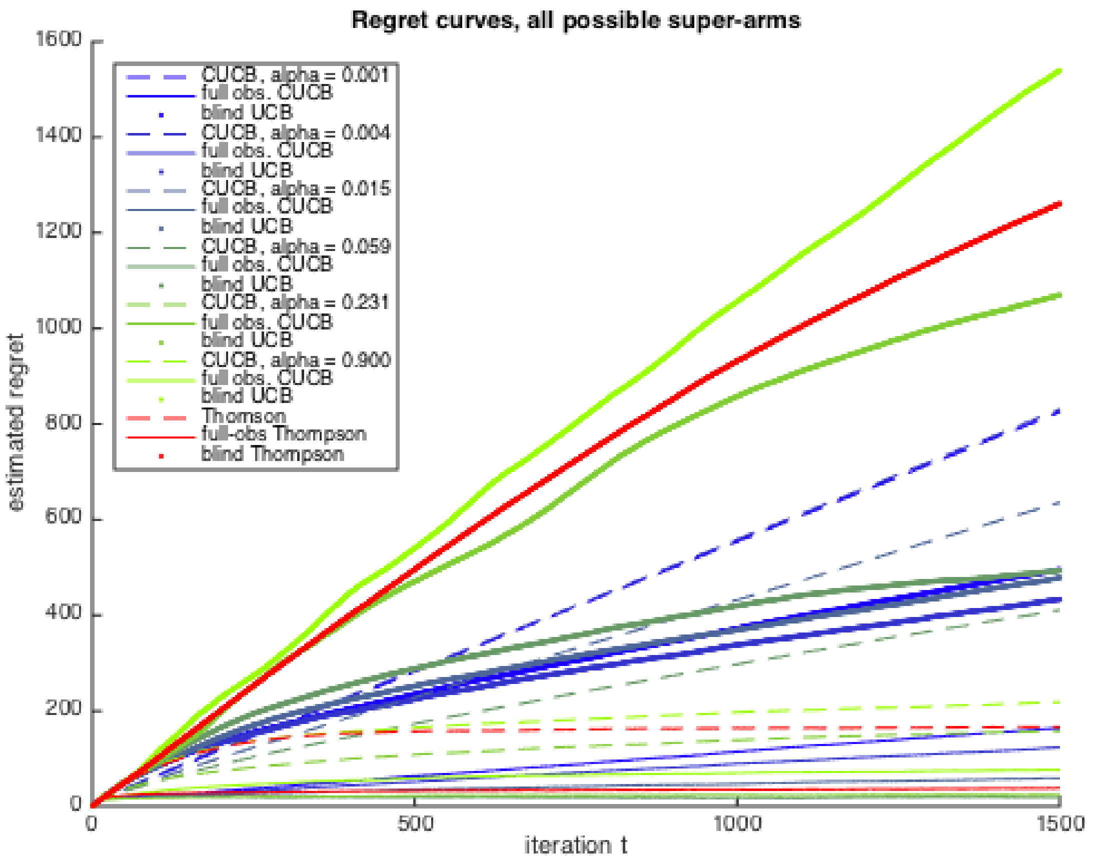


FIGURE 5: Reward = sum of rewards, Easy, $N_{simu} = 200$

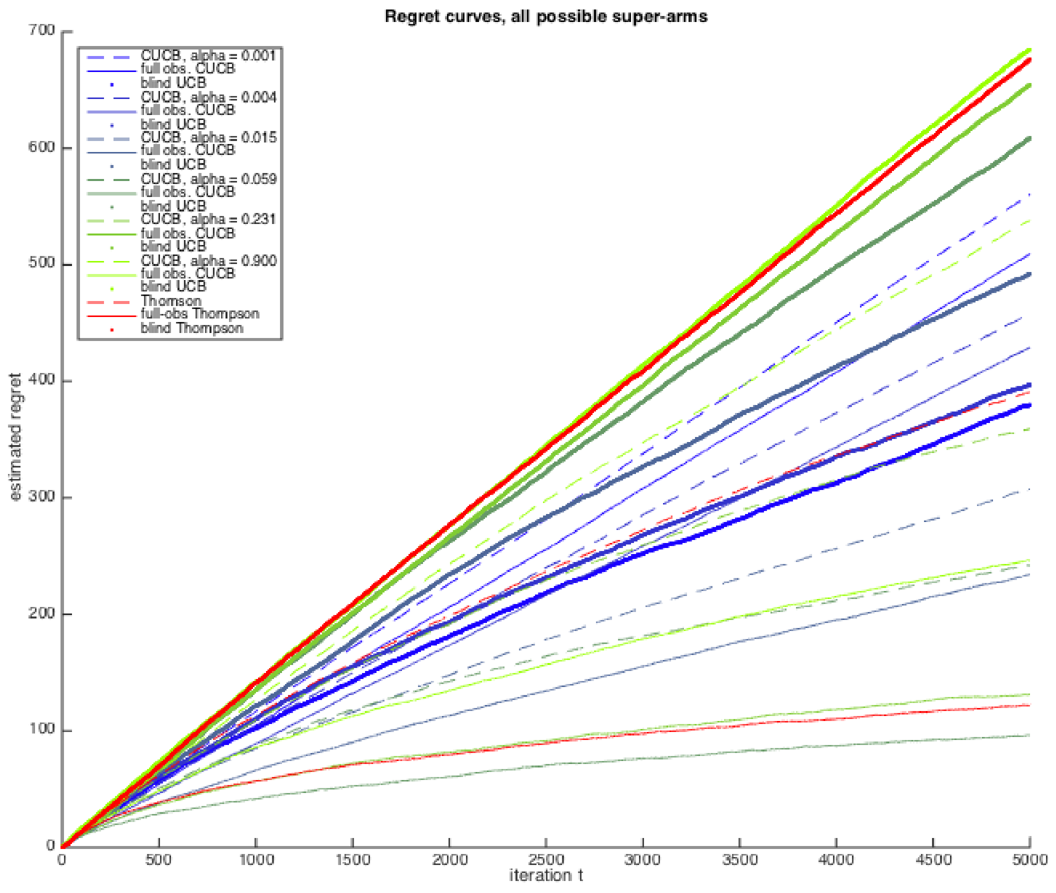


Figure 6: Reward = sum of rewards, VeryHard, $N_{simu} = 200$

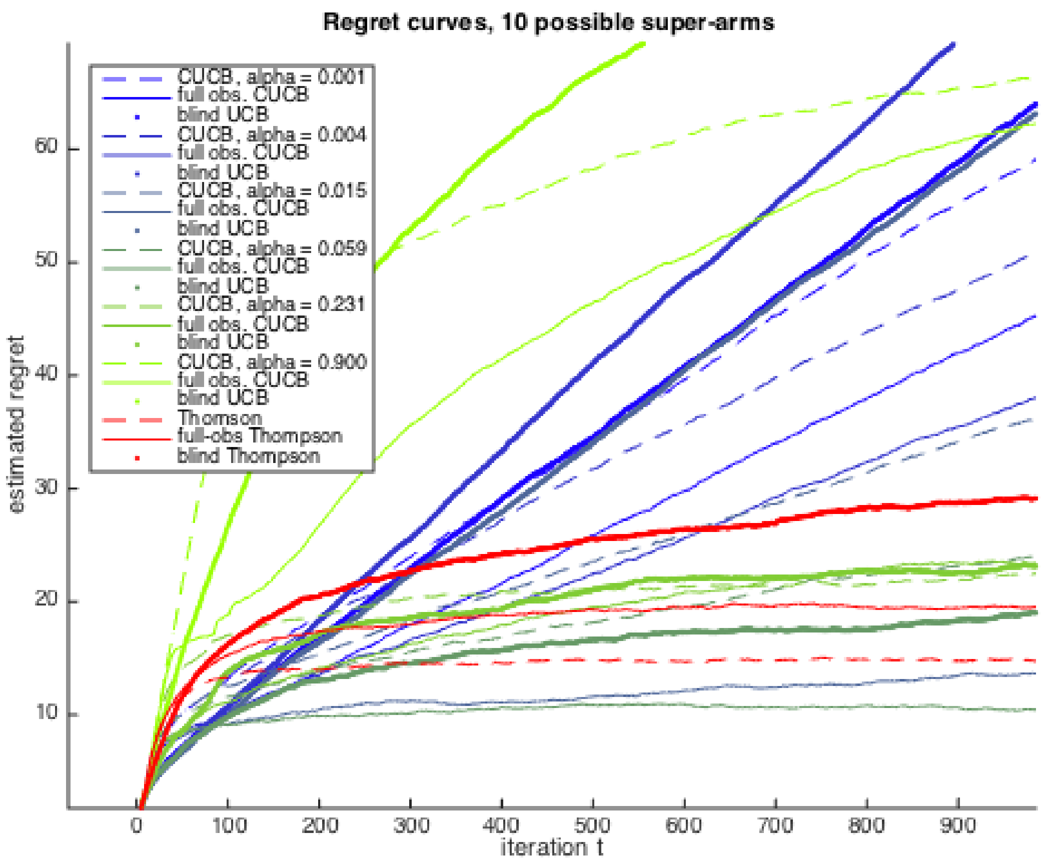


FIGURE 7: Reward = max of rewards, VeryEasy, $N_{simu} = 200$

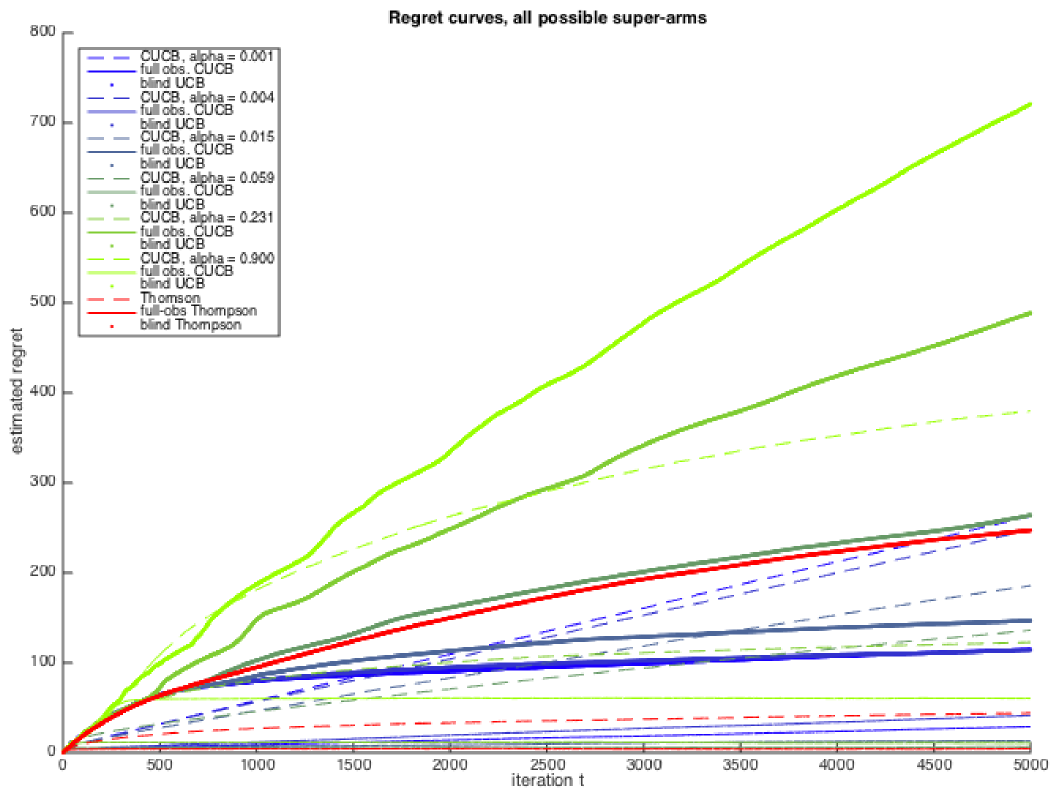


FIGURE 8: Reward = max of rewards, Easy, $N_{simu} = 200$

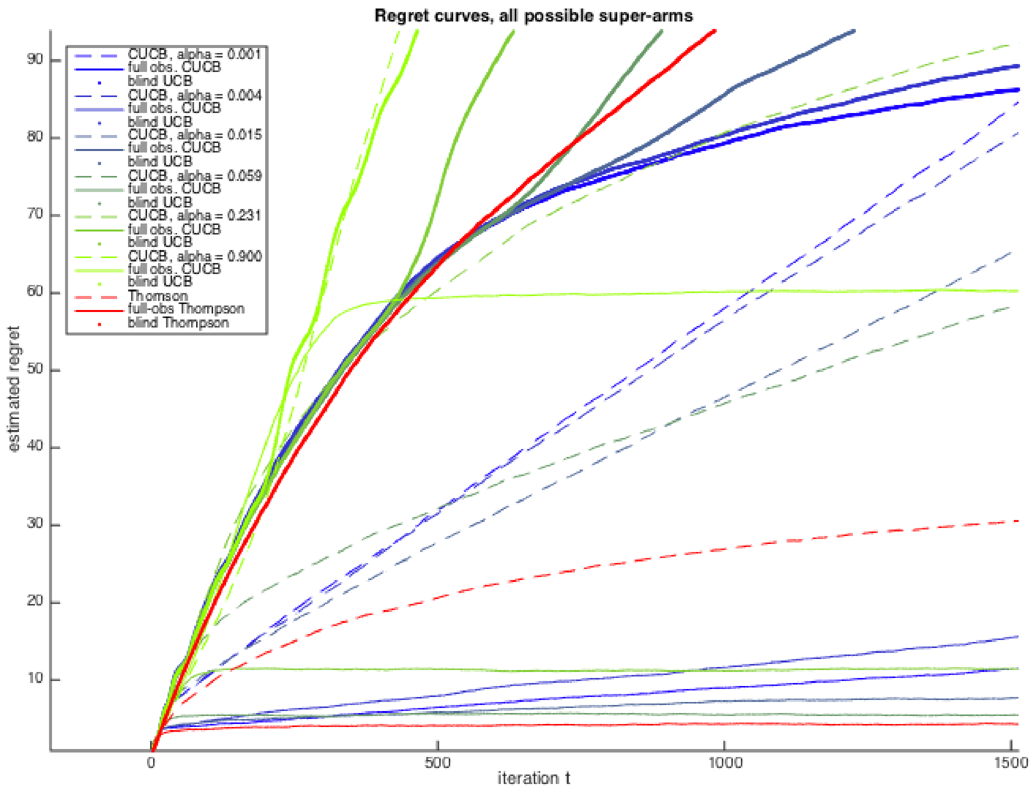


FIGURE 9: Reward = max of rewards, Easy - CLOSE UP, $N_{simu} = 200$

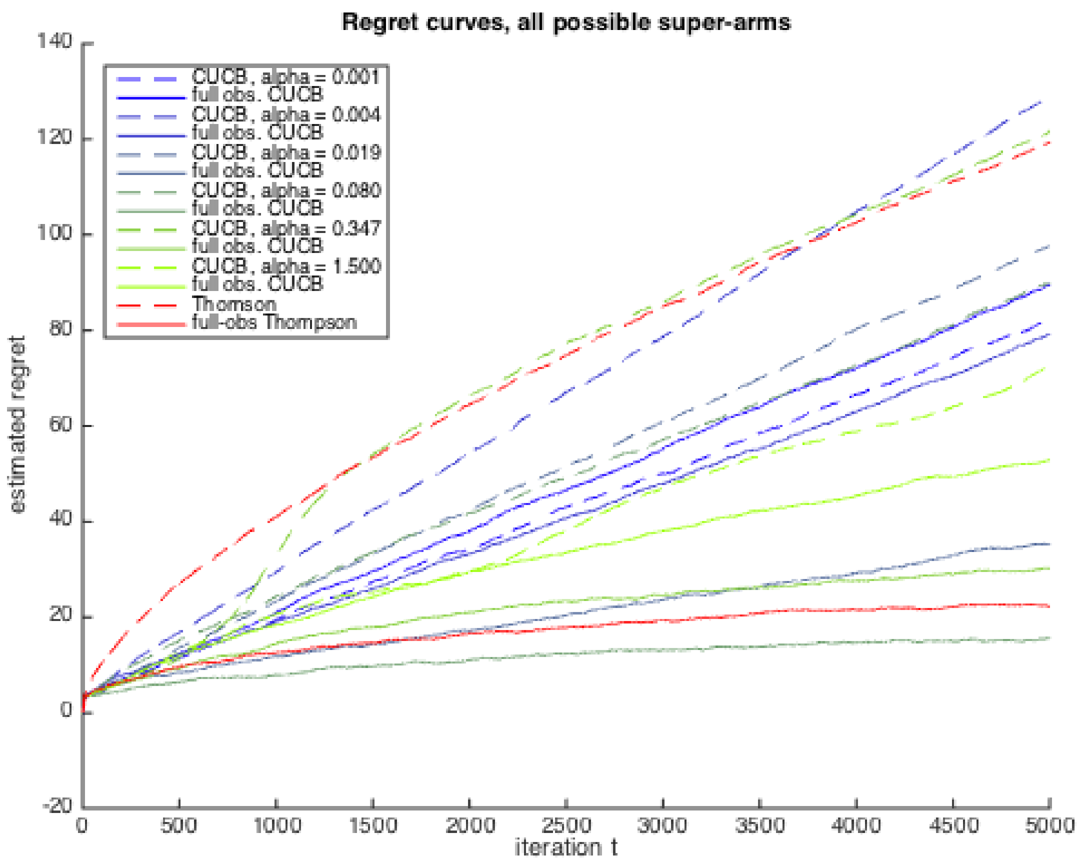


Figure 10: Reward = max of rewards, Hard (naive strategy not displayed), $N_{simu} = 200$

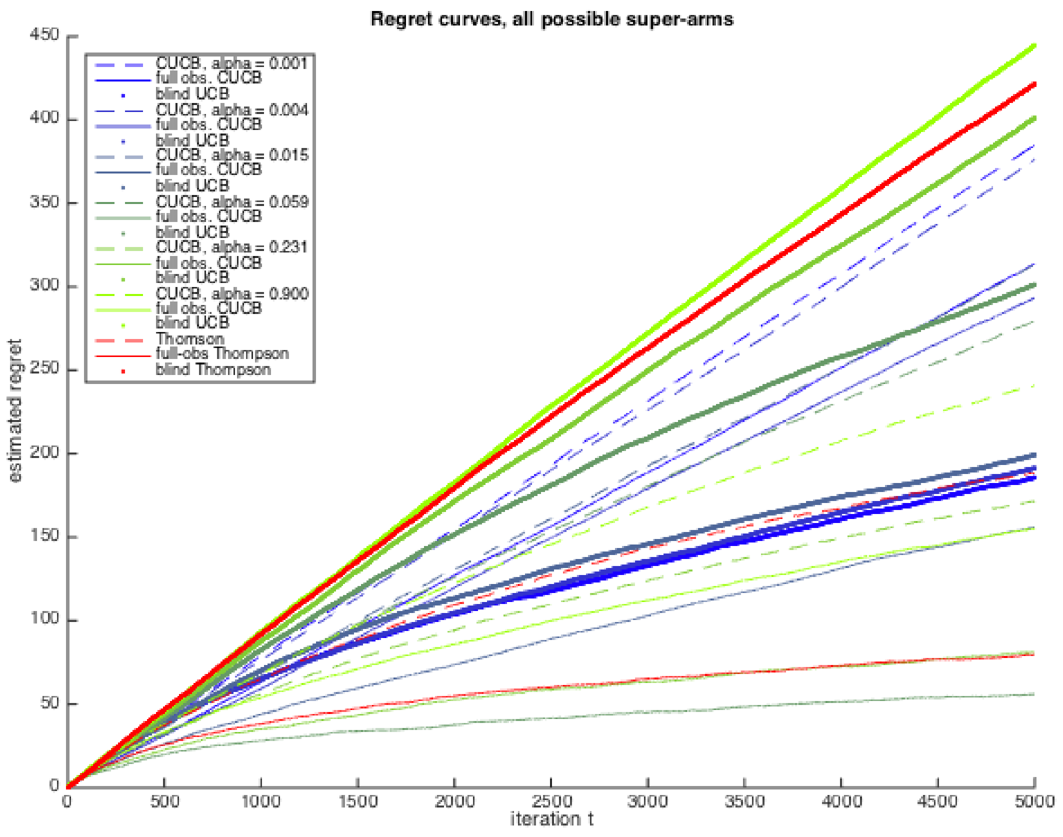


FIGURE 11: Reward = max of rewards, VeryHard, $N_{simu} = 200$