Master M2 MVA 2015/2016 - Reinforcement Learning - TP 2

The stochastic multi-armed bandit problem

By: Lilian Besson (`lilian.besson at ens-cachan.fr`). Attached programs: The programs for this TP are included in the `zip` archive I sent, and are regular **MATLAB/Octave** programs[1].

---

1 Building a MAB problem

I did not create any other `arm` class, but in the examples below I used at least each distribution once.

For this first part, we considered this MAB problem, for which the best arm is the first one (with mean 0.5):

- `Arm1` = Bernoulli(0.5) (mean: 0.5),
- `Arm2` = Beta(3, 7) (mean: 0.3),
- `Arm3` = Exp(3) (mean: 0.31674),
- `Arm4` = Finite([0.10.30.70.8], [0.20.40.10.3]) (mean: 0.45),

2 The UCB algorithm

See the file `naive.m` and `UCB.m` (they are well commented) for the implementation, and the main file `mainTP2.m` for the demo.

Below in Figure 1 is plotted the observed regret on *one* simulation, comparing the naive strategy with the UCB algorithm (with two different values of $\alpha = 0.02, 0.5$):

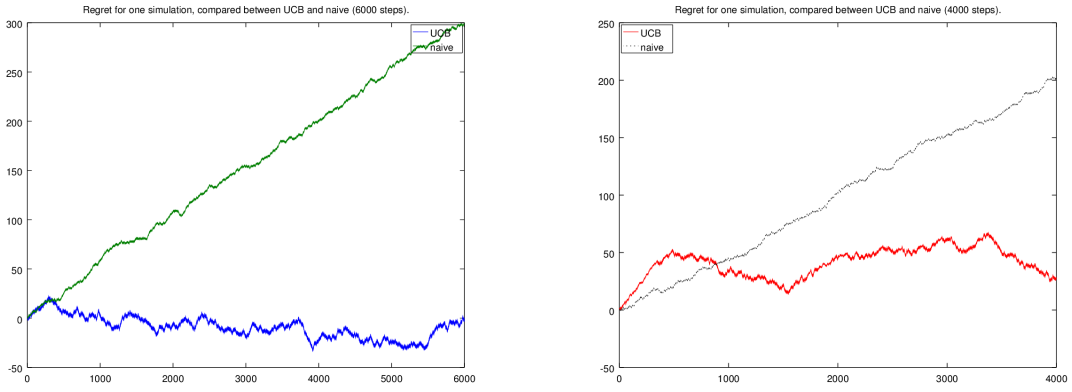

Figure 1: Regret for one simulation, naive vs UCB (left $\alpha = 0.02$, right $\alpha = 0.5$).

[1] Note: I only tested my programs with GNU Octave, and version `v4` at least is required – the `classdef` feature is new in Octave version 4.0.

As expected, even without trying to use the "best" value for α, the naive strategy is performing badly compared to UCB (ie. naive regret is bigger).

Question 1

For the same MAB problem, instead of comparing two strategies on just one simulation, we consider 4 strategies (naive, and 3 values of α for UCB), and we compare them based on their empirical mean regret (average on `nbSimulation` simulations, 50 in our test, to not be too time consuming).
Below is showed the key part of that simulation (for the 3rd evaluated strategy, a UCB with $\alpha = 0.25$):

```
T = 4000; alpha = 0.25;
all_rews3 = zeros(nbSimulation,T);
% Simulate a lot of times one (random) run of UCB on this MAB
for i = 1:nbSimulation
    [rews3,draws3] = UCB(T,alpha,MAB);
    % Compute the estimated regret of this ith simulation
    all_rews3(i,:) = maxmu * (1:T) - cumsum(rews3);
end;
% At the end, compute the average of the estimated regret
avg_rews3 = mean(all_rews3, 1);
```

Below in Figure 2 is plotted the observed mean regret (on 40 simulations), comparing the naive strategy with the UCB algorithm (with three different values of $\alpha = 0.02, 0.5$):

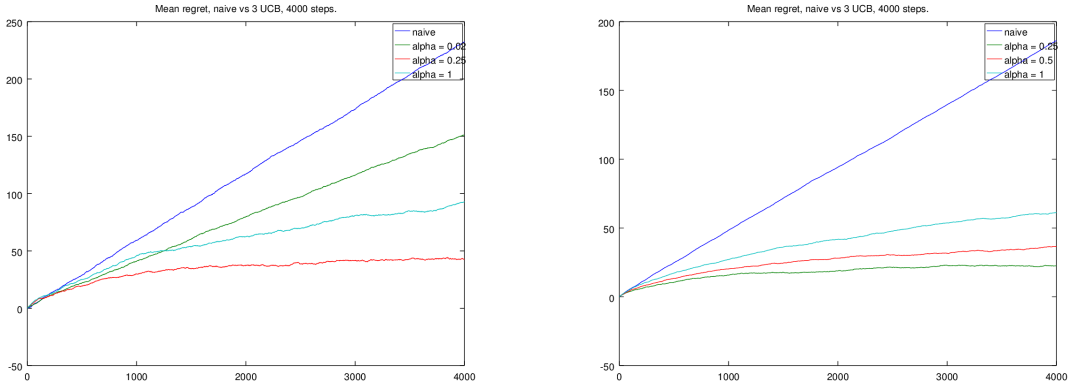

Figure 2: Two comparisons of naive vs 3 UCB, based on the average regrets.

The best value for α seemed to be 0.02, which is coherent with the one predicted by the course. In the lecture, we saw that the optimal value for ρ was about 0.2, and $\rho\sqrt{\frac{\log t}{2N}} = \sqrt{\frac{\alpha \log t}{N}}$ so $\alpha = \frac{\rho^2}{2}$, and so if $\rho^* \simeq 0.2$, $\alpha^* \simeq 0.02$.

Another observation is that all the UCB performed better than the naive strategy, because we chose valid values of α. For some badly chosen α, UCB will perform worse.

3 Complexity of a bandit problem

In order to plot the lower bound when comparing the **UCB** and **Thompson sampling** algorithms, we implemented the complexity for a MAB problem with K Bernoulli arms in the file `complexity.m`. That bound comes from Example 2 in a paper by Lai and Robbins in 1985, and we can compute it concretely, thanks to the fact that the Kullback-Leibler divergence between two Bernoulli distributions is easy to compute (with a formula).
Note: computing the bound requires to have access to the parameters of the Bernoulli (their means), and not just to be able to observe events: it is not simulation-based.

4 A Bayesian idea for Bernoulli bandit problems

1. In the `Thompson.m` file we implemented the **Thompson Sampling** algorithm, as described in the assignment. It's a basic randomized algorithm, which selects at each step the arm to pull based on a Beta distribution (the posterior distribution). Cf. the file for more details, it's also well commented.

Below is included the key part of that simulation, showing what happens at each time step:

```
% Beta distribution used to chose the action
theta_t = betarnd(S + 1, N - S + 1);
% Pick the next arm A_t : Bayesian bandit algorithms choose an action
% based on the current posterior (Beta) distributions over the parameters of the arms
[~, A_t] = max(theta_t);
% Update rewards and nb of visits
rew_t  = MAB{A_t}.play(); % This reward *is* random!
N(A_t) = N(A_t) + 1;      % One more visit of A_t
S(A_t) = S(A_t) + rew_t;  % More reward!
% Append the chosen arm and obtained reward
rews  = [rews rew_t];
draws = [draws A_t];
```

2. I don't think we can call the **Thompson Sampling** algorithm "optimistic" anymore, because it does not build confidence intervals at all. UCB is optimistic in the sense that it keeps an estimation of the average and confidence on each arms, picking the one that has the biggest Upper Confidence Bound (average + confidence). Thompson Sampling simply keeps in memory and slowly improves an estimation of the underlying hidden parameters of the random arms, but it does not have any idea of how trustworthy the estimations are.

Question 2

To conclude, we wanted to compare the two algorithms UCB vs Thompson Sampling. The second one only works for Bernoulli MAB, so we first chose a "simple" Bernoulli MAB (where UCB will

be able to quickly identify the best arm), and an "harder" one when we guess that Thompson Sampling will really be useful.

- "Easy" Bernoulli MAB, 3 arms of parameters $0.5, 0.7, 0.4$. Best one has mean 0.7 (far from the other arms). Complexity was computed to be 7.8184.
- "Hard" Bernoulli MAB, 4 arms of parameters $0.43, 0.45, 0.5, 0.4$. Best one has mean 0.5 (less far from the other arms). Complexity was computed to be 133.58.

On the plot below (Figure 3), we compare the regrets of **Thompson Sampling** with the regrets of **UCB**, for *one* simulation. We observe what we were expecting: UCB can work better than Thompson (if it is lucky) on the easy MAB, but Thompson is really better on the harder MAB problem:

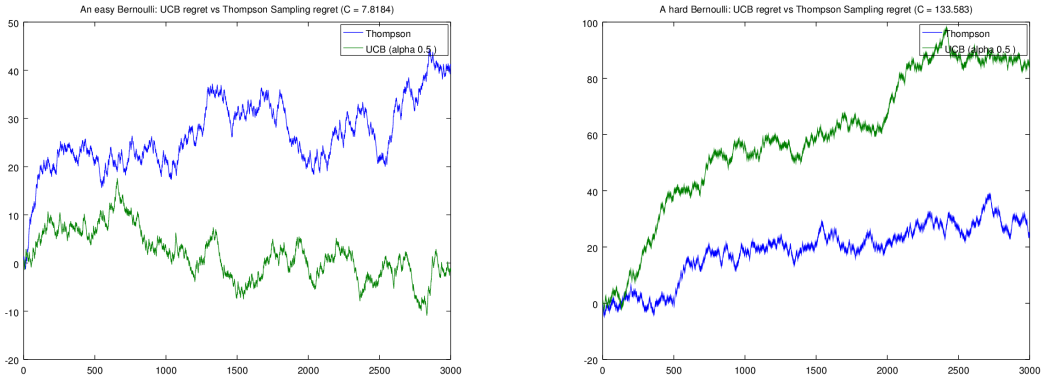

Figure 3: Comparing UCB vs Thompson Sampling, "easy" (left), and "hard" (right) Bernoulli MAB.

And the next plot (Figure 4) compare the average regrets (for 20 simulation, all this was really slow so it would have taken a lot of time to do more). We observe that the previous plot came from a lucky run of UCB: in average, Thompson is better for the easy MAB. It is harder to interpret the result of the plot on the right for the harder MAB, where both algorithms perform as well.

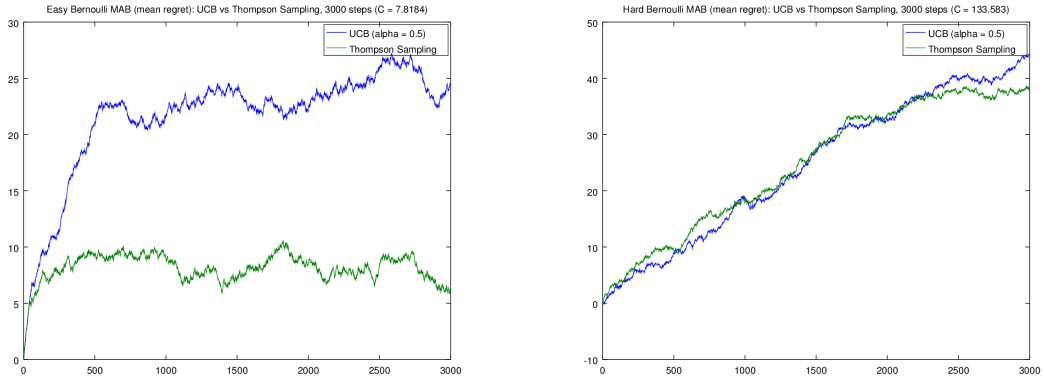

Figure 4: Comparing mean regrets of UCB vs Thompson Sampling, "easy" (left), and "hard" (right) MAB.

Finally, we can compare these average regrets of UCB and Thompson with the Lai and Robbins's bound:

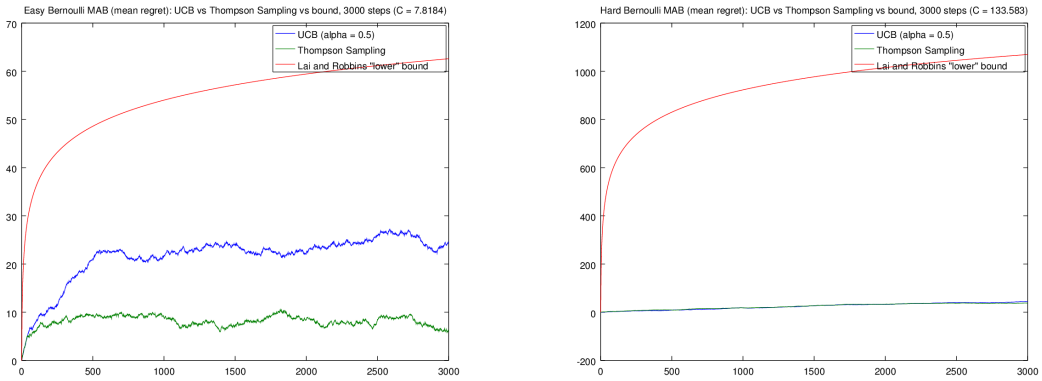

Figure 5: Comparing mean regrets of UCB vs Thompson Sampling vs Lai and Robbins's bound.

These last plots show that the theoretical *lower* bound is in fact *above* the other curves, which would be really weird if it was not an *asymptotic* bound. In our limited-time simulation, the horizon was finite and quite small ($T = 3000$), so we should be not surprised to observe this behavior: for "small" values of T, the bound is very pessimistic. And additionally, for more complex MAB problems, the bound is observed to be really less accurate.

Conclusion

The TP was interesting, thanks.

I did a little extra for the **UCB** and **naive** implementation, the two functions will draw and keep updated a plot showing their current estimated means for each arms (just change `doPlot = false` to `doPlot = true` in `main.m`). It was interesting to see how evolve the means during the iteration of **UCB**. I also try to plot the confidence intervals for **UCB**, but it turned out to be pretty slow (I should have done the same plot optimization trick with `HandlePlot` but ran out of time to finish it).