Multi-armed bandit learning on a graph

The G-UCB algorithm

Reinforcement Learning

Leonardo Di Nino (1919479)

Academic Year 2023/2024

SAPIENZA
UNIVERSITÀ DI ROMA

The purpose of my final project in the reinforcement learning course was to address MAB learning on graphs. The two main references are the following papers:

- *Multi-armed Bandit Learning on a Graph*, Tianpeng Zhangy, Kasper Johansson, Na Li, 2023
- *Cooperative Multi-Agent Graph Bandits: UCB Algorithm and Regret Analysis*, Phevos Paschalidis, Runyu Zhang, Na Li, 2024

Table of Contents

The MAB framework: quick recap

The Multi-armed bandits model is a popular model to deal with decision-making scenarios under uncertainty. The classical vanilla MAB consists of:

- n independent arms (encoding for the *actions*);
- Each arm has unknown reward distribution ν_i with mean $\mu_i = \mathbb{E}_{r\sim\nu_i}[r]$
- An independency and similarity sampling scheme within each arm: every time the agent pulls an arm it gets an IID reward

The agent want to maximize the total reward on a certain temporal horizon T: this corresponds to the regret minimization problem, being the regret the difference between the maximum expected reward and the ones got from the sequence of decisions.

$$T\mu^* - \sum_{t=1}^{T} \mathbb{E}[r_{I_t}] \tag{1}$$

The MAB framework: quick recap

This model is a standard one in introducing the (in)famouse trade off in reinforcement learning between:

- *exploration* to learn the unknown distributions;
- *exploitation* to maximize the reward leveraging the current knowledge.

The golden rule in dealing with this trade-off is the optimism principle based on Hoeffding's inequality: the Upper Confidence Bound (UCB) grants that the regret is bounded by $O(\sqrt{nT})$.
However the classic MAB is a very simple model that requires a strong assumption that is not always feasible: the agent must be able to access all the arms at each time step. It must be able in other words to switch between different arms in a "stateless" flavour.

The graph bandit problem

A graph bandit problem reintroduce back transition dynamics on subsequent actions like in a classic MDP leveraging the topology of a graph. Specifically we consider a graph $G(S, \mathcal{E})$ with self-loops where:

- The nodes set S encodes for both the *states* and the *actions*;
- The edges set $\mathcal{E}$ encodes for the *available actions* given a certain state; specifically, the available actions can be seen as the set of neighbours N_u of a given node u;

As in the MAB at each node corresponds an unknown probability distribution, but the agent knows the structure of the graph, so the whole decision process can be recasted as a *graph traversal*: every time the agent visits a node s, it receives a random IID reward from a probability distribution $P(s)$.

The graph bandit problem: notation and formulation

A graph bandit problem reintroduce back transition dynamics on subsequent actions like in a classic MDP leveraging the topology of a graph. Specifically we consider a graph $G(S, \mathcal{E})$ where:

- The nodes set S encodes for both the *states* and the *actions*;
- The edges set $\mathcal{E}$ encodes for the *available actions* given a certain state; specifically, the available actions can be seen as the set of neighbours N_u of a given node u;

As in the MAB at each node corresponds an unknown probability distribution, but the agent knows the structure of the graph, so the whole decision process can be recasted as a *graph traversal*: every time the agent visits a node s, it receives a random IID reward from a probability distribution $P(s)$.

Table of Contents

2 Offline planning under perfect information

Offline planning as a shortest path problem

The graph bandit problem is in a perfect information setting if the mean rewards $\{\mu_s : s \in S\}$ are known. In that case we can define the expected cost of visiting a node:

$$c_s := \mu^* - \mu_s = \mathbb{E}[\mu^* - r_s] \tag{2}$$

being μ^* the maximum expected reward collectible on the graph.
We can recast the total regret of a certain traversal as $\sum_{t=0}^{T} c_{s_t}$, so that if we define a policy $\pi : S \to S$ we have the following problem:

$$\begin{cases} \min_\pi R_\infty(\pi, s_0) = \sum_{t=0}^{\infty} c_{s_t} \\ s_{t+1} = \pi(s_t) \in N_{s_t}, \forall t \end{cases}$$

Offline planning as a shortest path problem

Algorithm 1 shows us an implementation under perfect information: the graph bandit problem reduces itself to a shortest path where the edges weights are the switching costs.

Algorithm 1 Offline SP planning algorithm

Input: Graph $G = (S, \mathcal{E})$, mean reward vector $\mu = (\mu_1, \mu_2, \ldots, \mu_{|S|})$

Output: Policy $\pi_{\mathbf{SP}} : S \to S$

1: $\mu^* \leftarrow \max(\mu_1, \ldots, \mu_{|s|})$,
2: $s^* \leftarrow \arg\max_s(\mu_1, \ldots, \mu_{|S|})$
3: Define distance $a_{ss'} := c_{s'} = \mu^* - \mu_{s'}$ for all $(s, s') \in \mathcal{E}$
4: Let $\hat{G} = (S, \hat{\mathcal{E}}, \mathcal{D})$ be a directed version of G where $\mathcal{D}$ stands for the distances for $(s, s') \in \hat{\mathcal{E}}$.
5: $\pi_{\mathbf{SP}} \leftarrow$ `Dijkstra`$(\hat{G}, s^*)$, or `Bellman-Ford`$(\hat{G}, s^*)$ where $\pi_{\mathbf{SP}}(s)$ is the node that leads to the path with the shortest total distance from s to s^*.

Table of Contents

G-UCB: the setting and the notation

3 The G-UCB algorithm

In this setting the graph bandit learning is structured as a single traversal but it is divided into episodes of growing length without reset of the environment.
The algorithm results as a combination of the offline planning and the principle of ottimism according to the following rule for the upper-confidence-bound:

$$U_{m-1}(s) = \bar{\mu}_{m-1}(s) + \sqrt{\frac{2\log(t_m)}{n_{m-1}(s)}} \tag{3}$$

where t_m is the total number of timesteps up to m-th episode; $\bar{\mu}_{m-1}(s)$ is the average reward collected at node s up to and $n_{m-1}(s)$ is the number of samples collected at node s.

G-UCB: the algorithm

3 The G-UCB algorithm

- The whole procedure is a heuristic based on an upper confidence bound as a surrogate for the true mean in the offline planning algorithm;
- The doubling schemes ensures the logarithmic bound on the regret (“*Near-optimal regret bounds for reinforcement learning*” T. Jaksch, R. Ortner, and P. Auer, Journal of Machine Learning Research, vol. 11, no. 51, pp. 1563–1600, 2010)

Algorithm 2 G–UCB: Graph-UCB Algorithm

Input: The initial node s_0. The offline SP planning algorithm **SP** in Algorithm 1 that computes the optimal policy defined given the graph G and a set of reward values $\{\hat{\mu}_s\}_{s\in S}$ for each node: $\hat{\pi} \leftarrow \mathbf{SP}(G, \{\hat{\mu}_s\}_{s\in S})$.

$m \leftarrow 0$
Follow any path that visits all nodes at least once.//`Initialize` U_0.
Place the agent at s_0. $s_{curr} \leftarrow s_0$.
while The agent hasn't received a stopping signal **do**
  $m \leftarrow m + 1$
  Calculate the UCB values $U_{m-1}(s)$ for all $s \in S$ according to (5).
  $\pi_m \leftarrow \mathbf{SP}(G, \{U_{m-1}(s)\}_{s\in S})$
  while $U_{m-1}(s_{curr}) < \max_s U_{m-1}(s)$ **do**
    Execute π_m for one step. $s_{curr} \leftarrow \pi_m(s_{curr})$
    Collect reward at s_{curr}.
  end while
  $dest_m \leftarrow s_{curr}$. Keep collecting rewards at node $dest_m$ until the number of its samples doubles compared to its sample number at the beginning of episode m. //`This operation ensures` $n_m(dest_m) = 2n_{m-1}(dest_m)$.
end while

G-UCB: stating the upper bound on the regret

Theorem 1

Let $T \geq 1$ be any positive integer. Than the regret of G-UCB after T timesteps after the initialization is bounded by:

$$R^* \leq O(\sqrt{|S|T\log(T)} + D|S|\log(T)) \tag{4}$$

being S the set of vertices on the graph and D its diameter.

Furthermore if $T \geq D^2|S|\log(t)$, the bound becomes:

$$R^* \leq O(\sqrt{|S|T\log(T)}) \tag{5}$$

Table of Contents

4 Benchmarks and numerical experiments

Some specifics about the numerical experiments

The simulations were meant to compare G-UCB with some different approaches:

- **Local-UCB**: the standard UCB algorithm with graph-constraints on the decision process;
- **Local-TS**: the standard Thompson sampling algorithm for bayesian bandits with graph-constraints on the decision process;
- **G-QL**: a Q-learning procedure designed over a graph;
- **QL-UCB**: an hybrid Q-learning algorithm that introduces a bonus term in an upper-confidence-bound fashion; the procedure is proposed in *"Is q-learning provably efficient?"* (C. Jin, Z. Allen-Zhu, S. Bubeck, and M. I. Jordan, in Advances in Neural Information Processing, vol. 31. Curran Associates, Inc., 2018).

Some specifics about the numerical experiments

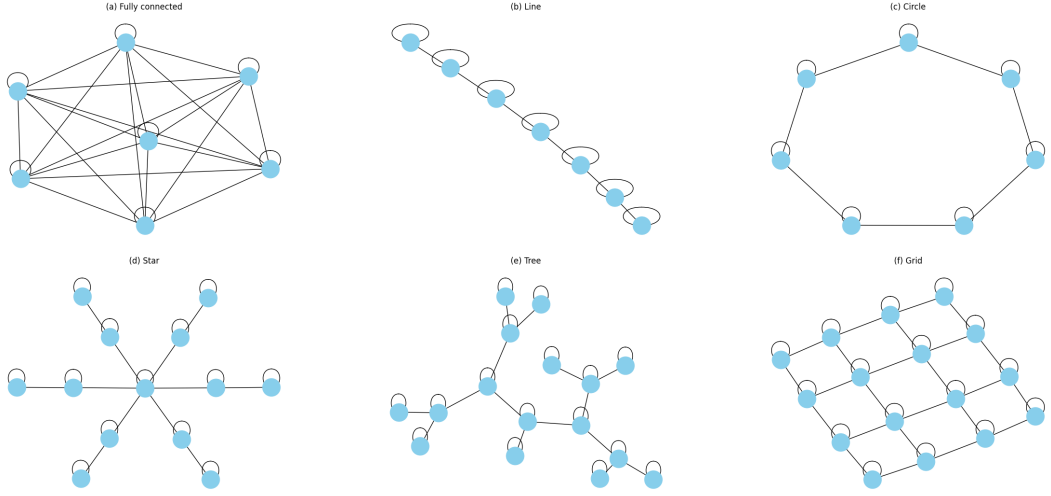
Every algorithm was tested with the same specifics:

- 100 simulations were run;
- $T = 2 \times 10^4$ was the hard upper bound on the number of timesteps;
- The random-reward graph was inizitialized randomly at the beginning of each simulation, sampling for each node the mean reward from $Unif(0.5, 9.5)$ for every topology, while for the fully connected one the means were initialized from $Unif(0.5, 1.5)$;
- The reward distribution for each node s was $Unif(\mu_s - 0.5, \mu_s + 0.5)$.

Then the average regret across the simulations was considered with 1 standard deviation for plotting a confidence band for the G-UCB.

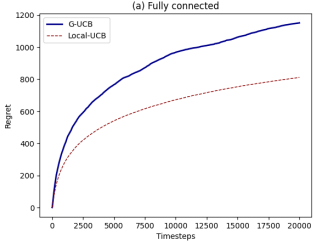
Topologies considered for testing

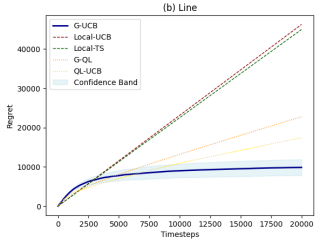
4 Benchmarks and numerical experiments

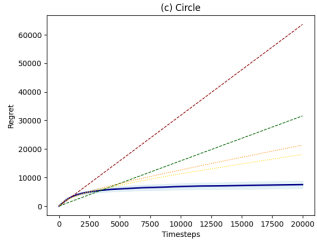


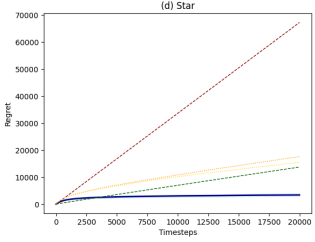
Topologies considered for testing

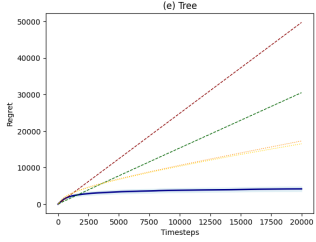
4 Benchmarks and numerical experiments

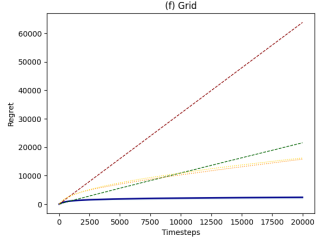








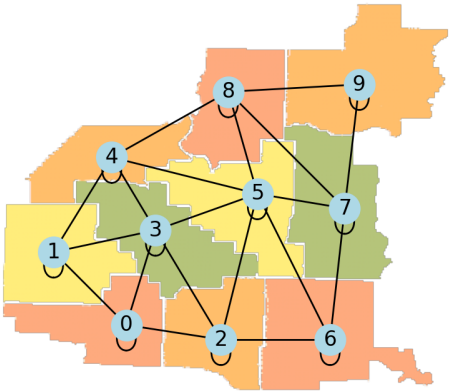




Robotic environment simulation

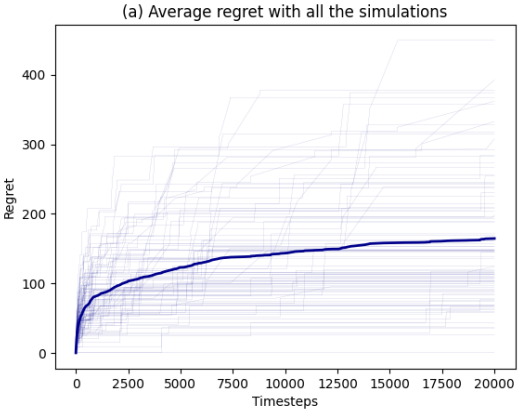
4 Benchmarks and numerical experiments

An additional simulation was performed on a synthetic robotic environment where a drone has to provide internet connection: the reward is the random amount of communication traffic carried in that node. The goal is to maximize the total reward on a traversal before being called back and recharged.



Robotic environment simulation

4 Benchmarks and numerical experiments



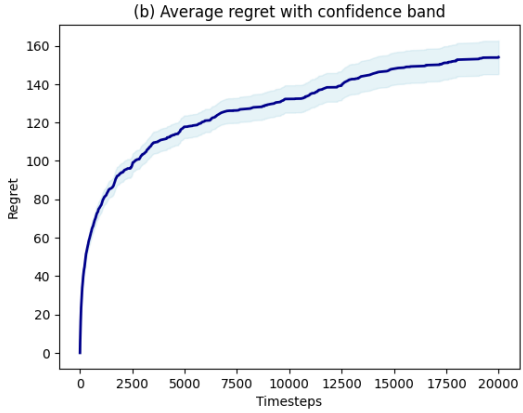


Table of Contents

5 The Cooperative Multi-Agent Graph Bandits: MultiG-UCB

MultiG-UCB: the main concepts

The MultiG-UCB algorithm is a generalization of G-UCB to a cooperative multi-agent scenario.
We consider again a connected graph with self loops $G = ([K], E)$, and for each arm k we have a related probability distribution P_k supported on $[0, 1]$ and mean μ_k.
We now have N agents coordinated by a central aggregator traversing the graph G through feasable paths. We need to introduce some concepts to generalize the previous framework to this one.

MultiG-UCB: the main concepts

We define the following:

- The vector $C_t = (c_{1,t}, ..., c_{K,t})$, being $c_{k,t}$ the number of agents allocated at node k at time-instant t;
- The non-decreasing concave function f_k such that $f_k(0) = 0, f_k(1) = 1$.

A reward $X_{k,t}$ is observed at time instant t in every node k where $c_{k,t} > 0$: the system observes a reward that is a weighted sum of the rewards observed at each node:

$$R_t = \sum_{k \in [K]} f_k(c_{k,t}) X_{k,t} \tag{6}$$

MultiG-UCB: the main concepts

In order to now recast the problem as a regret minimization we need further definitions. In particular given the following set of possible agents allocations:

$$\mathcal{C} = \{(c_1, ..., c_K) : \sum_{k \in [K]} c_k = N\} \tag{7}$$

We can define the optimal agents' allocation as:

$$C^* = \underset{C \in \mathcal{C}}{\operatorname{argmax}} \sum_{k \in [K]} f_k(c_k)\mu_k \tag{8}$$

And finally we have the following definition of expected regret:

$$\mathcal{R}_t = \mathbb{E}[\sum_{k \in [K]} (f_k(c_k^*)\mu_k - f_k(c_{k,t})X_{k,t})] \tag{9}$$

Offline planning as a minimum weighted matching on a complete bipartite graph

5 The Cooperative Multi-Agent Graph Bandits: MultiG-UCB

Again optimism principle is used to leverage UCB as a surrogate of the true mean reward to compute the optimal allocation of the agents and the switching costs. We then build a complete bipartite graph based on this partial outputs and use a minimum weighted matching to assign each agent to a destination node according to the optimal allocation. Differently from the single agent case this is a pseudo-solution with no proof of optimality.

Algorithm 2 `SPMatching`

Input: The current time t, the current position of the agents, $\{k_{i,t}\}_{i\in[N]}$, and a desired allocation of agents, $\hat{C}_e$.

1: Calculate the multiset of destination arms $\mathcal{S}_e = ([K], \hat{C}_e)$.
2: For each agent m calculate the regret shortest path between its current location and each of the destination arms.
3: Initialize a complete bipartite graph $G_B = ([N], \mathcal{S}_e, \mathcal{E}_B)$ where the weight of each edge $(i,k) \in \mathcal{E}_B$ is defined as the length of the regret shortest path between the current location of i and the destination arm k.
4: Calculate the minimum weight perfect matching of G_B.
5: Have each agent follow the regret shortest path to their assigned destination node.

MultiG-UCB: the main algorithm

5 The Cooperative Multi-Agent Graph Bandits: MultiG-UCB

The main program adapts the single agent case to the multi agent one leveraging the offline policy to dispatch the agent through minimal regret paths to the optimal allocation and again using a doubling scheme to ensure a logarithmic bound on the regret.

Algorithm 1 `Multi-G-UCB:`

Input: The initial nodes for each agent, $\{k_{i,0}\}_{i\in[N]}$. The offline planning algorithm `SPMatching` (see Section III-B) that computes the optimal policy given the graph G and a set of computed UCB values, $\{U_{k,t_e}\}_{k\in[K]}$, for each node.

$e \leftarrow 0$
Run the initialization algorithm to visit each vertex in G. See Section III-C.
while coordinator has not received a stopping signal **do**
    $e \leftarrow e+1$
    Calculate the UCB values $\{U_{k,t_e}\}_{k\in[K]}$.
    Solve the optimization problem in (3) for the desired allocation $\hat{C}_e$.
    Denote by k_{min} the arm with fewest observed samples of all arms k such that $\hat{c}_{k,e} > 0$ and denote by n_{min} its current number of samples ($n_{min} = n_{k_{min},t_e}$).
    Follow the offline planning policy `SPMatching` until agents are distributed according to $\hat{C}_e$.
    Have each agent i continue to collect rewards at its current node until $n_{k_{min},t} = 2n_{min}$.
end while

Some specifics about the numerical experiments

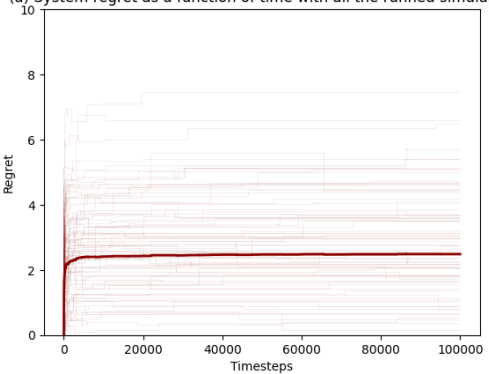
The algorithm was tested with these specifics:

- 100 simulations were run;
- $T = 10^5$ was the hard upper bound on the number of timesteps;
- The simulations were run only on the synthetic robotic environment because of the computational heavyness with 5 agents traversing the graph;
- The random-reward graph was inizitialized randomly at the beginning of each simulation, sampling for each node the mean reward from $Unif(0.25, 0.75)$;
- The reward distribution for each node s was $\mathcal{N}(\mu_s, 0.06)$.

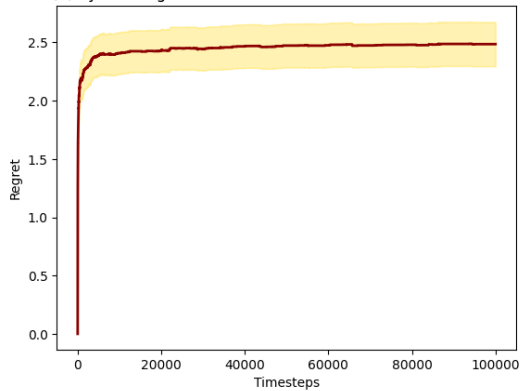
Robotic environment simulation for multi-agent scenario

5 The Cooperative Multi-Agent Graph Bandits: MultiG-UCB

(a) System regret as a function of time with all the runned simulation



(b) System regret as a function of time with confidence band



Multi-armed bandit learning on a graph

Thank you for listening!
Any questions?