Social Influence Maximization

Gabriele Ghiringhelli (914499), Leonardo Febbo (905180),
Alberto Frattini (898440), Alberto Floris (905900).

December 2019

1 Assignments

1. *Imagine three social networks with, respectively,* 10^2, 10^3 *and* 10^4 *nodes. Every edge can be characterised by 4 features and the probability of the edge is given by a linear combination of the features. Assign a cost of every node and define a reasonable budget for the social influencer. We suggest using 3 different values of the budget for the 3 social networks.*

2. *Apply, to the three scenarios, the greedy algorithm to solve approximately the influence maximization problem in the case in which the probabilities of the edges are known. In doing that, use Monte Carlo simulations and show how the value of the objective function varies for different values of the approximation bound.*

3. *Assume that the probabilities of the edges are not known. Apply: combinatorial bandit algorithms (TS and UCB1-like) in the case the linear structure of the probabilities is not exploited (and therefore the probability of every edge is estimated by using only samples related to that edge), the UBC1-like algorithm exploiting the linear structure of the probabilities of the edge. Show how the expected regret varies as the number repetitions of the problem increases.*

4. *Modify the social networks to assure that every node has at most 15 neighbors. Apply a UCB1-like algorithm when edges cannot be observed and have to be estimated from data. Show how the expected regret varies as the number repetitions of the problem increases.*

2 Introduction

This document aims to discuss the Social Influence Maximization on different Social Networks. We introduce the problem of influence maximization as it was formulated for the first time in 2003 in [1, Section 2.1].

The goal of the influence maximization problem is to maximize the number of individuals that are reached by some information. Another fundamental element is the way in which the influence spread in the network, moving from one node to another one. These ways are called diffusion models and we have chosen to use the Influence Cascade (IC) model.

People in the network are usually divided into three main categories: active users (alive), who have been reached by the information, inactive users (dead), who are not aware of the information yet, but there is a positive probability they may be influenced, and susceptible users, who can be influenced. Active users can influence inactive users, changing their status and activating them. One possible way for a node to be activated is because one of her neighbors has been activated and thus, in the next time window, it will try to influence her. Once a node has been influenced, it will remain active until the end of the process and, unless differently specified, each node tries to influence its neighbors only in the time instant after its activation.

Formally, the social network is modeled as a directed graph G = (V,E), where each node $v \in V$ represent an individual in the network and each edge e represents the connections between agents. We denote $|V|$ = n and $|E|$ = m. Each edge is characterized by some probability: p(e) : E $\rightarrow$ [0, 1], indicating the propagation probability along edge e. In other words, p(e) represents the strength of the relationship involving the agents. There is a set S of k $< |V|$ nodes, called seeds from which the diffusion starts. The term seed has been adopted since they are the starting point from which the information is generated. Finally, according to some diffusion model M, an influence function $\sigma(S)$: V $\rightarrow$ N is defined. $\sigma(S)$ computes the expected number of nodes that will be reached by the information if the seeds are nodes in S. Our goal is finding the optimal set of seeds.

To do so we will use the Greedy algorithm combined with Monte-Carlo sampling over a graph adopting the IC model. A node can get activated only if a randomly generated probability at each round of the IC is greater or equal than the probability assigned at that edge. The process stops when no new nodes can be activated.

We will exploit three different scenarios, each of them has a different Budget constraint fixed, in terms of number of nodes that can be influenced. Each node in the graph has a unitary cost, equal for each node. Probabilities over the edges are generated by a linear distribution of the features and the parameters of the features θ. The three scenarios that we have defined are the reality that we want to inspect. They are described as follow:

- **Classroom Mates**
  The first Social Network describes the scenario of a classroom of 10^2 students, using a Facebook group. Each student will be a node in the graph. The edges of the graph are characterized by a linear distribution over four features, that are: pleasantness, kindness, culture, and sincerity.
  Budget: 20.

- **Kinder Advertising**
  The Second Social Network is characterised by 10^3 of users seeing the new Kinder advertising campaign of a new product of chocolate bar with coconut, in which features are: Pathos generated by the advertise, the link with old products, tastiness and how much coconut is liked.
  Budget: 50.

- **Fashion Influencer**
  The Last Social Network composed by 10^4 of Instagram users, that could be influenced by some fashion influencer. Indeed the features will be the following arguments: Dress, Shoes, Makeup and Nightlife events.
  Budget: 100.

3 Greedy and Monte Carlo Algorithms

In this section, we want to show, thanks to the use of the Greedy Algorithm combined with the Monte Carlo(MC), how the value of the objective function, that has been defined as the number of nodes infected, varies for different values of the approximation bound, that is the number of iterations of MC.

We used MC simulations, integrated into the greedy framework, for estimating influence function $\sigma(S)$ so as to preserve model generality.
The Greedy framework iteratively selects a node u, that is not a seed, into S, if u provides the maximum marginal gain and it employs MC simulations for estimating influence spread $\sigma(S \cup u)$ for each user set S ∪ u in G. In particular, an instance of MC simulation, always starts from S ∪ u in G, simulates the activation process generating a set of live edge graphs related to the IC model and outputs the number of activated users denoted by I(S ∪ u). The influence function $\sigma(S)$ follows from the sub modularity property:

$$\forall X \subseteq Y, x \notin Y : f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$$

This means that, adding a seed to a set of seeds gives a marginal increase in the utility function, larger than adding the same node to a strictly larger set of seeds.
For each S ∪ u, the algorithm runs r rounds of MC simulations and takes the average I(S ∪ u) as an estimation of influence spread $\sigma(S \cup u)$. The number of repetitions R of the MC is given by:

With probability of at least

$$1 - \delta$$

the estimated activation probability of every node is subject to an additive error of

$$\pm \epsilon n$$

$$R = O\left(\frac{1}{\epsilon^2} \log\Big(|S|\Big) \log\left(\frac{1}{\delta}\right)\right)$$

Where n is the number of nodes and S is the set of Seeds.

As Output, after the execution of the Greedy Algorithm, we have the set of optimal Seeds that have to be activated in order to obtain the optimal influence maximization. In Figure 1 is shown the pseudo code of the Greedy working with MC:

Algorithm 1: $\boldsymbol{Greedy}(\mathcal{G}, \sigma)$

```
initialize S ← ∅;
while the size of S is smaller than k do
    find u ← MC samples
    S ← S ∪ {u};
return S;
```

Figure 1: Greedy Pseudo Code

Here is reported our code interpretation of the MC definition:

```
def monte_Carlo_Sampling(self, n_iterations_mc, temp_seeds):

        nodes = np.random.binomial(0, 1.0, size=(n_nodes))
        for _ in range(n_iterations_mc):
            nodes_live_graph = self.influence_cascade(self.n_steps_
            max_live_graph, temp_seeds)
            nodes = np.array(nodes + nodes_live_graph)
        nodes_by_iteration = nodes / n_iterations_mc
        return np.sum(nodes_by_iteration)
```

The main scope of this section, is the one of obtaining the most suitable objective function, indeed, in this first case, we have obtained the most efficient set of seeds, related to the fact that we assume to know which are the probabilities of the edges in each of the scenario analyzed.

(In [2], Leskovec et al. present an optimization in selecting new seeds, which is referred to as the "Cost-Effective Lazy Forward" (CELF) scheme. The CELF optimization uses the submodularity property of the influence maximization objective to greatly reduce the number of evaluations on the influence spread of vertices. Their experimental results demonstrate that CELF optimization could achieve as much as 700 times speedup in selecting seed vertices, which is a very impressive result.)

3.1 Results of Greedy's and MC's analysis

We show, in the following subsections, thanks to some graph (Figure 2 to 10), the results of the objective function, that we have obtained in each scenario, at the variation of Approximation Bound.

They report how the value of the objective function varies for different values of the approximation bound for each scenario that we have analyzed.
In each of the three subsection are reported three graphs:

1. The **First graph** shows how many nodes have been reached by keeping epsilon fixed among delta variation. There are four functions of different color, with a different value of epsilon each. The optimal, the one with $\epsilon = 0.1$ can be visibly noticed in the graph.

2. In the **Second graph**, there are four functions, each with a different color and with a different delta. ϵ has the optimal value fixed. Thanks also to a graphic analysis, we have obtained that, the most suitable δ that we will use is a value between 0.2 and 0.3.

3. The **Third graph** reports, again with four functions of different color, but with different ϵ, how, actually, the optimal value for ϵ is 0.1, in order to have a more accurate optimal objective function.
   Other values, that grow up too quickly, may give inaccurate results.

3.1.1 Scenario with 10^2 nodes

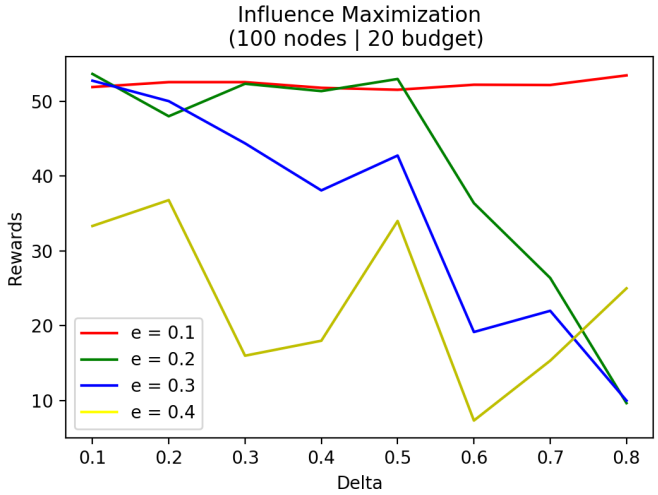

Figure 2: First Graph

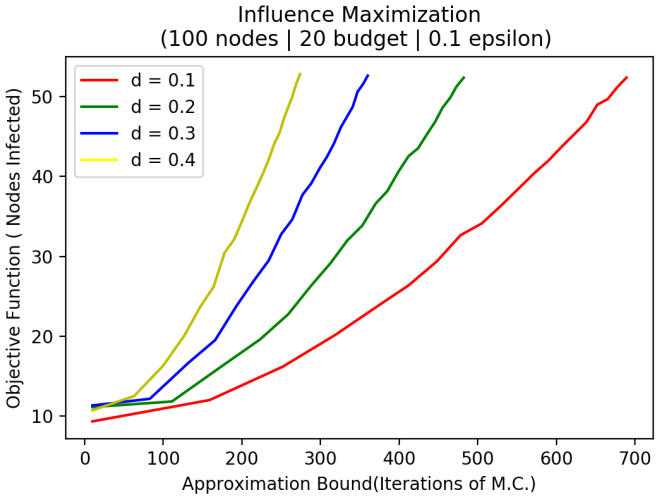

Figure 3: Second Graph

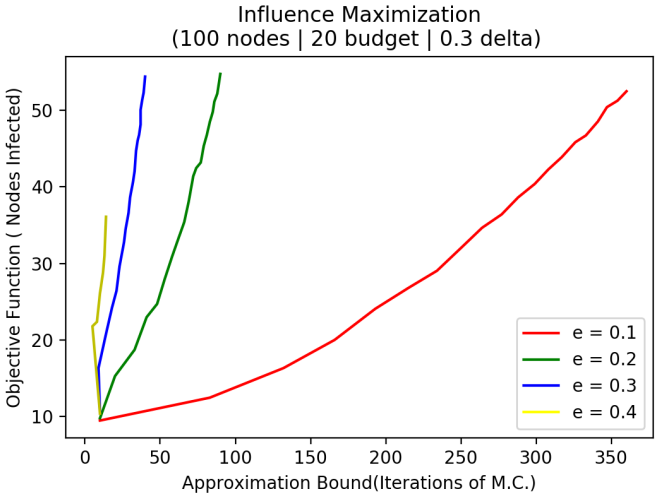

Figure 4: Third Graph

3.1.2 Scenario with 10^3 nodes

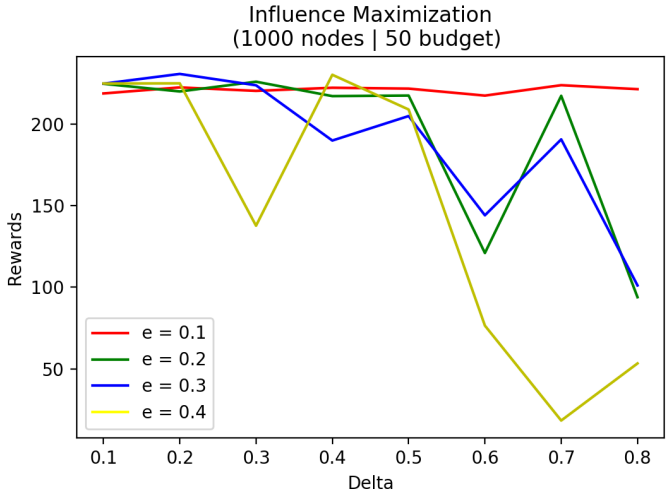

Figure 5: First Graph

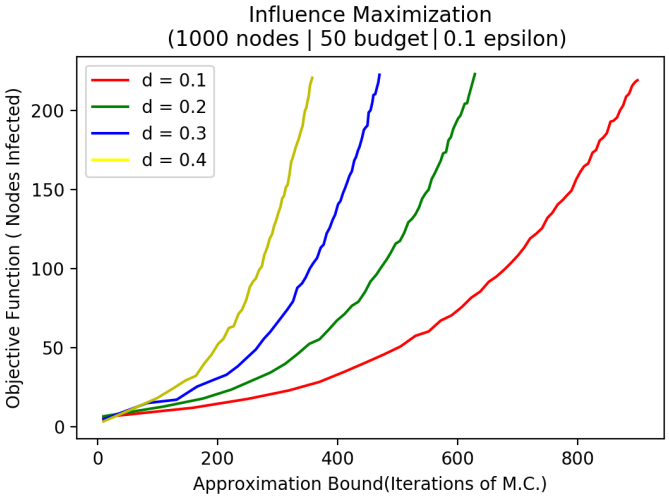

Figure 6: Second Graph

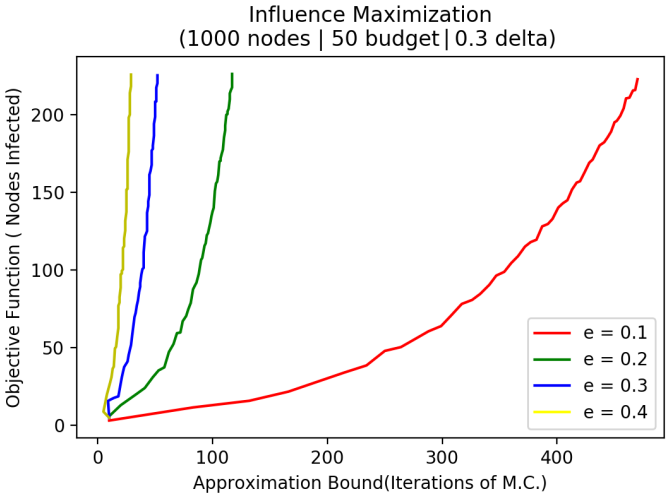

Figure 7: Third Graph

3.1.3 Scenario with 10^4 nodes

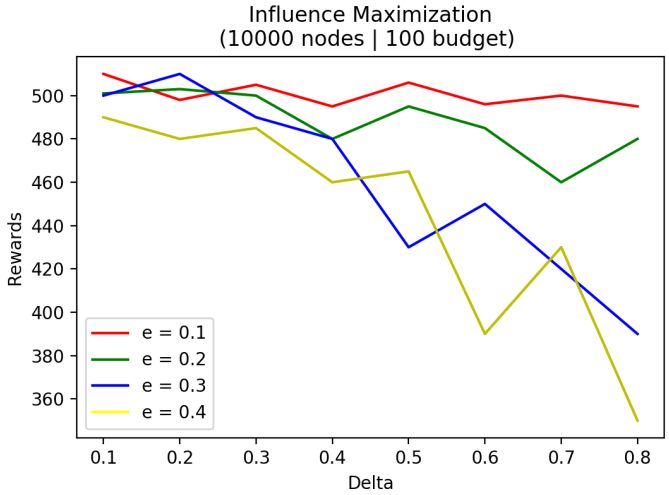

Figure 8: First Graph

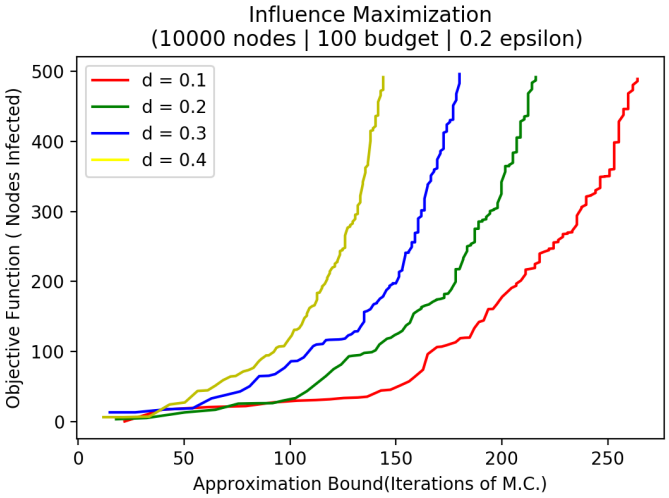

Figure 9: Second Graph

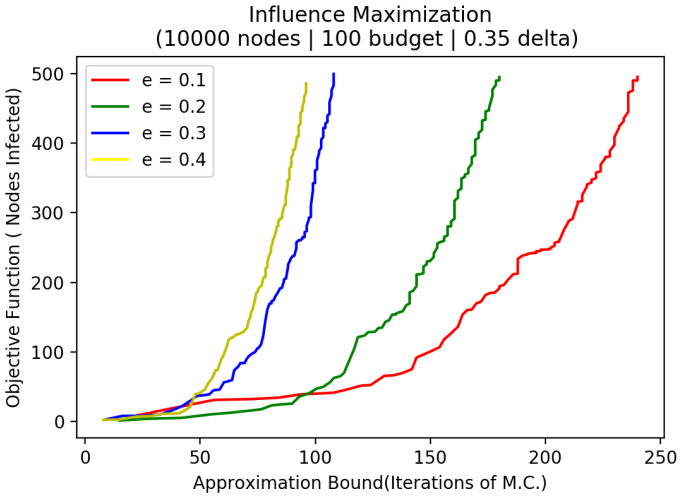

Figure 10: Third Graph

4 Unknown Social Problems

Here is presented the Influence Maximization Problem (IMP) without knowing the structure of the social network.

Most of existing algorithms assume that the entire topology of the network is given and so the goals are only: scalability, for the exact approaches, and the quality of the solution, for the approximated ones. However, in these settings, an algorithm to compute the best placements for the seeds should also explore the network in an intelligent way in order to reconstruct the presence of the various edges.

In [3], the authors assume that the influence probabilities are not known and it is not possible to estimate them just by data. This is why they propose to learn them online, employing CMAB (Combinatorial Multi Armed Bandit). The authors study the problem adopting the IC, even though, the provided framework is valid for any discrete time diffusion model. The actual spread is the number of nodes reachable from the selected seed nodes in the true possible world and we denote it by $\sigma(S)$. The mapping between CMAB and IMP is the following:

- associate an arm i to each edge $e \in$ E;
- reward $X_{i,t}$ represents the status of the edge, which can be either live, i.e., an activation attempt along it succeeded, or dead;
- mean μ_i is the weight of the edge corresponding to arm i;
- superarm A is associated to the union of outgoing edges ES from the seeds $s \in$ S;
- reward r_t in round t corresponds to the spread σ in the t-th diffusion attempt.

In each round, the regret minimization algorithm selects a seed set S with cardinality k and plays the corresponding superarm ES.

4.1 Online Influence Maximization, observable edges

As we have said previously, in the real World it is very difficult, if not impossible, being able to access this kind of information or being accurate in quantifying it. To avoid this assumption, we can use an approach based on a CMAB paradigm on two different possible cases, estimating the probabilities on the edges round by round, selecting a different seed set each time.

In CMAB framework there are m arms, each associated with a random variable, denoted as $X_{i,t}$, indicating the reward of triggering arm i on round t. $X_{i,t}$ is defined on [0, 1] and is independently and identically distributed according to Bernoulli distribution in the first case and a linear distribution in the second one, with mean μ_i.

At each round $t \in T$, a set of arms, called superarm and denoted by A, is played, triggering all the arms in the set. Moreover, some other arms may be probabilistically triggered. Let $p_i(A)$ be the triggering probability of arm i if the superarm A is played (of course, if $i \in A$, $p_i(A) = 1$). The reward obtained at each round t is a function of the rewards of the arms that have been triggered in t.

Each time an arm i is triggered, we observe the rewards and update its mean estimate μ. The superarm that is expected to give the highest reward is selected in each round with TS and UCB1 in the first case and with a Linear-UCB1 in the second one, which takes as input the current mean estimates and outputs an appropriate superarm A.

The goal is to minimize the accumulated regret incurred by choosing sub optimal seed sets over multiple rounds.
We have studied, in 4.2 and 4.3, the two different cases that we can met under edge level feedback, which are:

- **Fully Observable Edges**
- **Partially Observable Edges**

4.2 Fully Observable Edges

In this case, it is allowed to observe how the influence spread among different edges throughout the network. Here, for each scenario, we want to run the IC in the reality defined in the first point.

At each observation, we can draw a sample from a Beta distribution, adopting a Thompson Sampling (TS) approach, for every edge in the fake reality that we have generated to try recreating the reality that we can just observe. In this copy of the reality, initially, probabilities over edges are assigned thanks to a Bernoulli random variable whose expected value is unknown.

At the end of one IC's iteration, we have obtained a fake reality with updated probabilities, thanks to the regret minimization algorithm, and now we give it in input to the Greedy and MC algorithm, created in the previous section, to obtain a set of optimal Seeds. At the beginning of each next iteration, the set of optimal seed, found in the previous one, is passed to IC. We want to repeat this operation a H number of repetitions in order to obtain a set of seeds step by step closer to the optimal case.

We have adopted the TS, as defined in [4, Section V.A] and its pseudo code is reported in Figure 11:

Algorithm 1 Combinatorial Thompson Sampling (CTS).

1: For each base arm i, let $a_i = 1$, $b_i = 1$
2: **for** $t = 1, 2, \ldots$ **do**
3: For each base arm i, draw a sample $\theta_i(t)$ from Beta distribution $\beta(a_i, b_i)$; let $\boldsymbol{\theta}(t) := (\theta_1(t), \ldots, \theta_m(t))$
4: Select super arm $S(t) = \text{Oracle}(\boldsymbol{\theta}(t))$, get the observation $Q(t)$
5: **for all** $(i, X_i) \in Q(t)$ **do**
6: $Y_i \leftarrow 1$ with probability X_i, 0 with probability $1 - X_i$
7: $a_i \leftarrow a_i + Y_i$
8: $b_i \leftarrow b_i + (1 - Y_i)$
9: **end for**
10: **end for**

Figure 11: TS Pseudo Code

In addition to the use of TS, we have also used the upper confidence bound (UCB1) to obtain, with a different algorithm, the probabilities over the edges. Then we repeat the same procedure as before.
Follows our code of the UCB Learner:

```
class UCB_Learner:

    def __init__(self, n_nodes):
        self.n = n_nodes
        self.cumsum_realizations = np.zeros((n_nodes, n_nodes))

    def update(self, realizations):
        self.cumsum_realizations += realizations

    def create_estimate_graph(self, t):
        emp_mean = (self.cumsum_realizations / t)
        low_bound = -np.sqrt((2 * np.log(t)) / (t-1))
        high_bound = np.sqrt((2 * np.log(t)) / (t-1))
        bound = np.random.uniform(low=low_bound,
        high=high_bound, size=(self.n,self.n))
        matrix = emp_mean + bound
        matrix[matrix > 1.] = 1.
        matrix[matrix < 0] = 0.
        return matrix
```

In the following subsections is shown the fact that as the number of repetitions increase, regret decreases.
In our results, reported in the graphs (from Figure 12 to 23), are presented, per each Scenario, the analysis of Regret and Reward.
In the top row, there are graphs generated with the use of TS, while in bottom one, there are ones generated thanks to UCB1.
In our inspection, we have considered that Repetitions are the number of learner's update, and that Experiments means the number generated from *stimulations* $*$ *experiments of MC(with* ϵ *and* δ *fixed).*
We have given more importance to the TS, because as shown in our results and as we have learned from literature, empirically, the TS is the best in guarantee the Regret minimization.

4.2.1 Scenario with 10^2 nodes

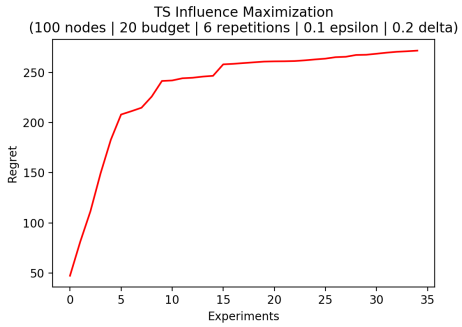

Figure 12: Regret TS

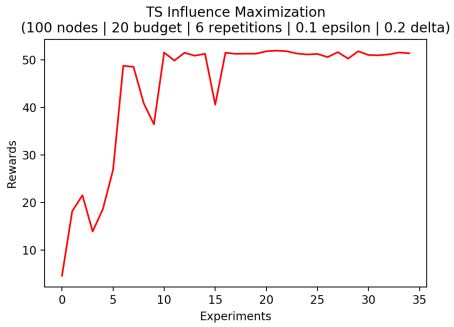

Figure 13: Reward TS

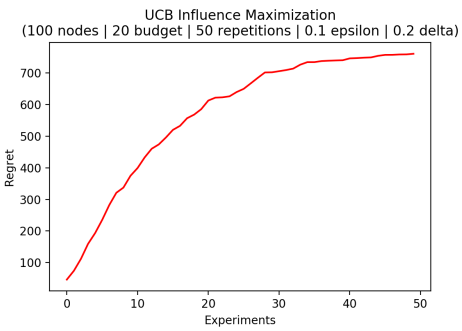

Figure 14: Regret UCB1

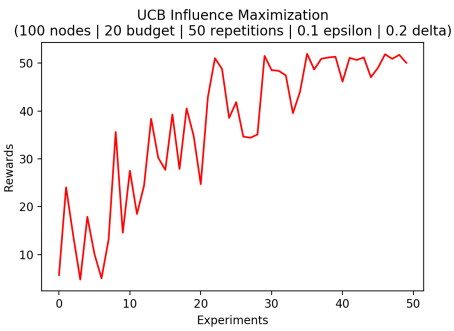

Figure 15: Reward UCB1

4.2.2 Scenario with 10^3 nodes

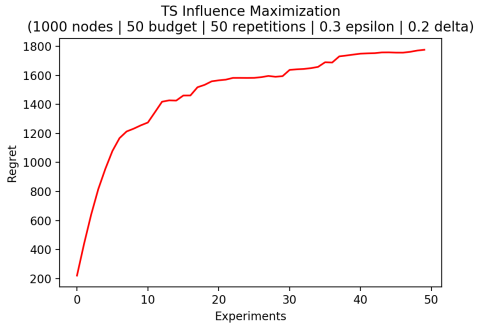

Figure 16: Regret TS

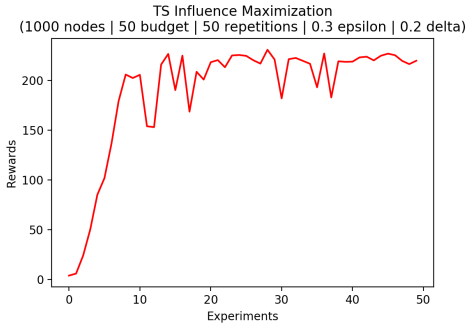

Figure 17: Reward TS

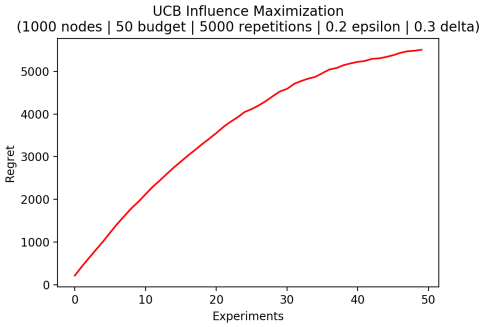

Figure 18: Regret UCB1

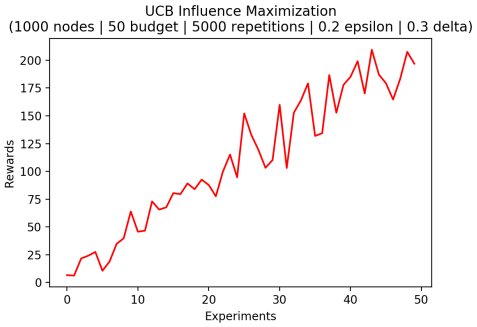

Figure 19: Reward UCB1

4.2.3 Scenario with 10^4 nodes

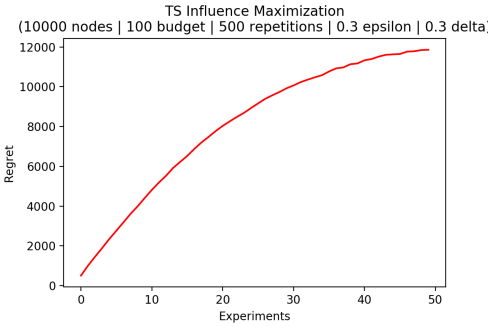

Figure 20: Regret TS

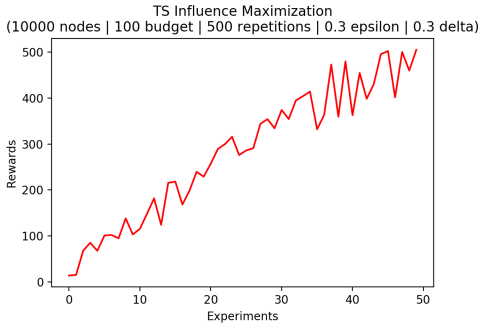

Figure 21: Reward TS

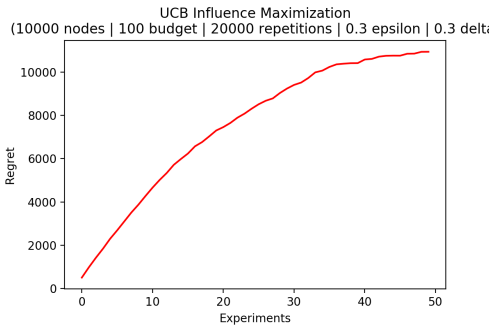

Figure 22: Regret UCB1

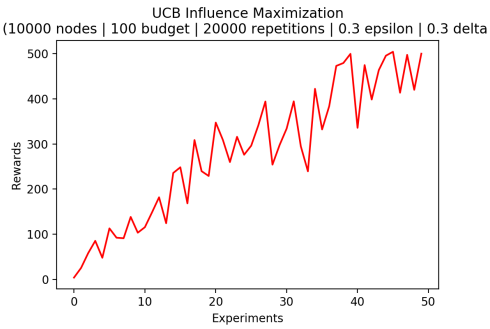

Figure 23: Reward UCB1

4.3 Partially Observable Edges:

Here we assume that, in each reality scenario, the probabilities over the edges are unknown as in the previous case, but only a small number of probes is allowed to obtain the topology.

In this case we want to run the IC in the reality and to update the upper confidence bound of edges analyzed thanks to the LinUCB1 learner. We want to define the problem setting as it has been defined in [5, page 2].

So let's now assume that T is the number of rounds and K the number of possible actions. Let $r_{t,a} \in [0,1]$ be the reward of action a on round t. On each round t for each action a the learner observes K feature vector $x_{t,a} \in R^d$, with $\| x_{t,a} \| \leq 1$. After observing the feature vectors the learner then selects an action a_t and receives reward r_{t,a_t}. We operate under the linear realizability assumption; that is, there exists an unknown weight vector $\theta^* \in R^d$ with $\| \theta^* \| \leq 1$ so that:

$$\mathbf{E}[r_{t,a} \mid x_{t,a}] = x_{t,a}^{\top} \theta^*$$

for all t and a. Hence, we assume that the $r_{t,a}$ are independent random variables with expectation $x_{t,a}^T \theta^*$.

Let $x_{t,a}$ be the feature vector of action a at step t, and define the regret of an algorithm A to be:

$$\sum_{t=1}^{T} r_{t,a_t^*} - \sum_{t=1}^{T} r_{t,a_t}$$

where $a_t^* = argmax_a x_{t,a}^T \theta^*$ is the best action at step t according to θ^* and a_t is the action selected by A at step t. We note that in our setting, the context vectors $x_{t,a}$ can be chosen arbitrarily by an oblivious adversary as long as the rewards $r_{t,a}$ are independent given the context.

To compute the expected reward of each arm by finding a linear combination of the previous rewards of the arm, the LinUCB decomposes the feature vector of the current round into a linear combination of feature vectors seen on previous rounds and uses the computed coefficients and rewards on previous rounds to compute the expected reward on the current round.
The pseudo code of the Algorithm is shown in figure 24.

Algorithm 1 LinUCB: UCB with Linear Hypotheses

Inputs: $\alpha \in \mathbb{R}_+, K, d \in \mathbb{N}$
$A \leftarrow I_d$ {The d-by-d identity matirx}
$b \leftarrow \mathbf{0}_d$
for $t = 1, 2, 3, \ldots, T$ **do**
  $\theta_t \leftarrow A^{-1} b$
  Observe K features, $x_{t,1}, x_{t,2}, \cdots, x_{t,K} \in \mathbb{R}^d$
  for $a = 1, 2, \ldots, K$ **do**
    $p_{t,a} \leftarrow \theta_t^{\top} x_{t,a} + \alpha \sqrt{x_{t,a}^{\top} A^{-1} x_{t,a}}$ {Computes upper confidence bound}
  end for
  Choose action $a_t = \arg\max_a p_{t,a}$ with ties broken arbitrarily
  Observe payoff $r_t \in \{0, 1\}$
  $A \leftarrow A + x_{t,a_t} x_{t,a_t}^{\top}$
  $b \leftarrow b + x_{t,a_t} r_t$
end for

Figure 24: LinUCB Pseudo Code

So when an iteration of the IC is finished, we obtain an update of probabilities over edges. Now we want to do as in the previous case in which Greedy and MC are run in order to obtain a set of optimal seeds. This operation has to be repeated a H number of times.

Regret bound, with probability $1-\delta$ is calculated per each scenario, for T rounds, K actions, and d dimensional feature vectors, as:

$$O\left(\sqrt{Td\ln^3(KT\ln(T)/\delta)}\right)$$

We chose to show only the regret generated by the graph of 10^3 nodes, because we have seen, through experimental results on the other two scenario, that results are not so different different from the one shown in figure 25.

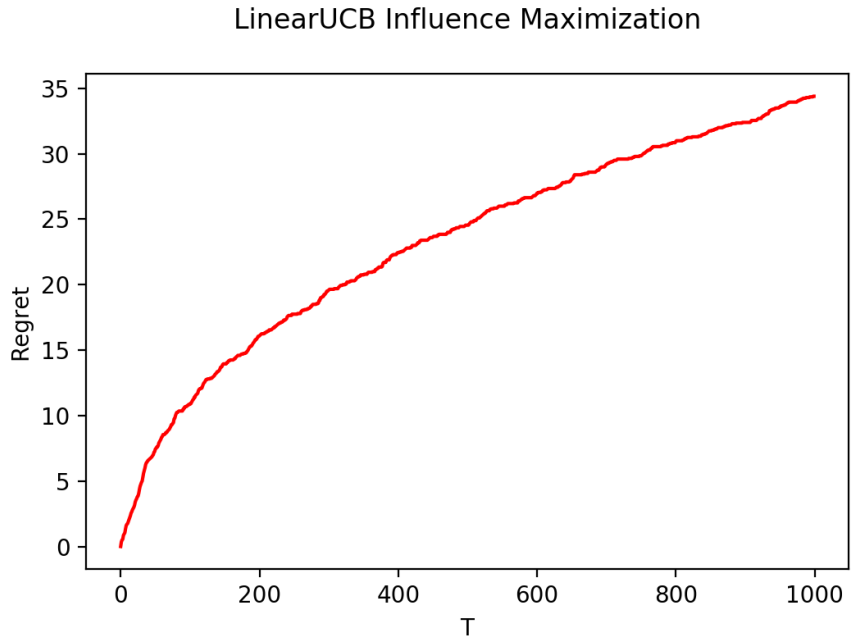

Figure 25: Regret LinUCB 10^3 nodes graph

5 Online Influence Maximization, non observable edges

In this last case, we have to consider that edges are not observable and so, we can only observe the activation of nodes trough the IC run over the reality. In order to do that, we have firstly adapted each Scenario with the constraint of the 15 neighbours. Then, because we can just see the activation of nodes at each round of the IC, we need to discover which edges activated it at the previous round. This is because we do not know which node, active at $t-1$, activated the node at time t or when it was activated.

Under edge level feedback, we assume to know the status of each edge and use it to update mean estimates.
Under node level feedback, any of the active parents may be responsible for activating a node u and we want to discover which one it is. In order to do that, the probabilities over the edges can be found through a Frequentist approach or with the Maximum Likelihood Estimation (MLE).

In the next subsections are described how they work. Once we have discovered which edges have most likely activated a node at time t, we have to repeat the procedure similar to the one that has been exposed in edge level feedback, using UCB1 like algorithm. Then the Greedy and MC algorithm to find the set of Seeds.

5.1 MLE

In general, MLE is a method of estimating the parameters of a probability distribution by maximizing a likelihood function, so that under the assumed statistical model the observed data is most probable.

The point in the parameter space that maximizes the likelihood function is called maximum likelihood estimate. The logic of maximum likelihood is both intuitive and flexible, and as such the method has become a dominant means of statistical inference. From a statistical standpoint, a given set of observations are a random sample from an unknown population.

The goal of maximum likelihood estimation is to make inferences about the population that is most likely to have generated the sample, specifically the joint probability distribution of the random variables y_1, y_2, ..., not necessarily independent and identically distributed. Associated with each probability distribution is a unique vector $\theta = [\theta_1, \theta_2, ..., \theta_k]^T$ of parameters that index the probability distribution within a parametric family. Evaluating the joint density at the observed data sample y $= (y_1, y_2, ..., y_n)$ gives a real valued function:

$$L_n(\theta) = L_n(\theta; \mathbf{y}) = f_n(\mathbf{y}; \theta)$$

which is called the likelihood function.
For independent and identically distributed random variable f_n will be the product of univariate density functions. The goal of maximum likelihood estimation is to find the values of the model parameters that maximize the likelihood function over the parameter space, that is

$$L_n(\hat{\theta}; \mathbf{y}) = \sup_{\theta \in \Theta} L_n(\theta; \mathbf{y})$$

Intuitively, this selects the parameter values that make the observed data most probable. The specific value

$$\hat{\theta} = \hat{\theta}_n(\mathbf{y}) \in \Theta$$

that maximizes the likelihood function L_n is called the maximum likelihood estimate.

There are two kind of Maximum Likelihood Estimation, one offline and one online. The first one has a complexity that doesn't scale to networks with large number of edges, so we have analyzed at theoretical level the online one as it has been defined in [3, Section 4.2].

5.2 Frequentist Approach

We chose to adopt this approach and so, we propose a scheme whereby we choose one of the active neighbours of v, say u_i, uniformly at random, and assign the credit of activating v to u_i.
The probability of assigning credit to any one of K active parents $1/K$. That is, edge $(u_i,\ v)$ is giving a reward of 1 whereas edge $(u_i,\ v)$ corresponding to other active parents u_i, $j \neq i$, are assigned a zero reward. We then follow the same update formula as described for edge level feedback model.

In the next subsections are displayed(Figure 26 to 31), in each scenario, how this procedure influence the regret and reward calculation. Here again we use the notation Experiments, with ϵ and δ fixed.

5.2.1 Scenario with 10^2 nodes

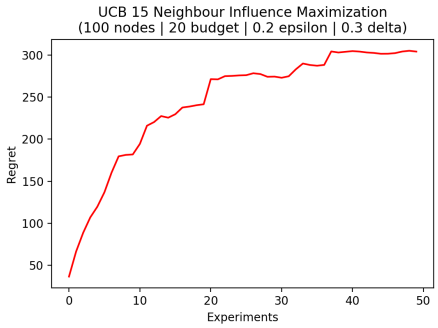

Figure 26: Regret 15 neighbours

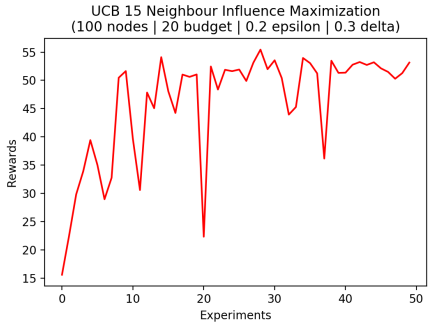

Figure 27: Reward 15 neighbours

5.2.2 Scenario with 10^3 nodes

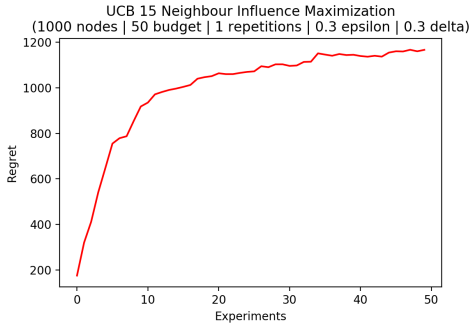

Figure 28: Regret 15 neighbours

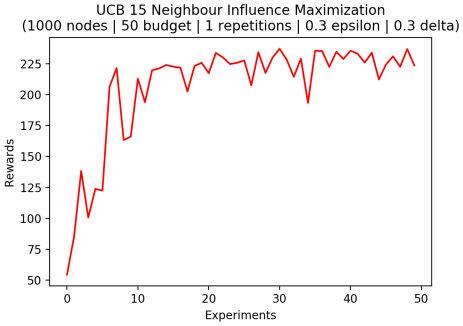

Figure 29: Reward 15 neighbours

5.2.3 Scenario with 10^4 nodes

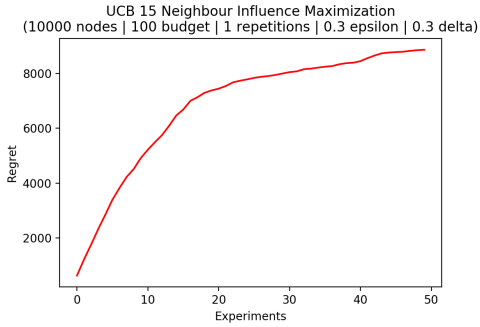

Figure 30: Regret 15 neighbours

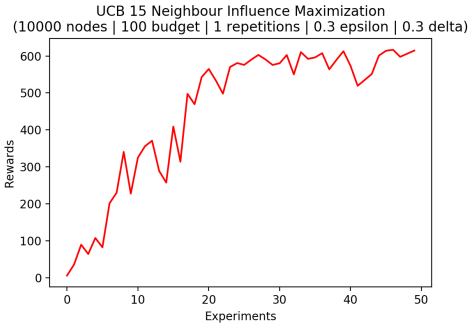

Figure 31: Reward 15 neighbours

6 REFERENCES

We have used, in order to complete the assignments, knowledge from Teaching Material of Data Intelligence Application Course and from the following papers.

- [0] Giuseppe De Nittis and Nicola Gatti. How to Maximize the Spread of Social Influence: A Survey.
- [1] David Kempe, Jon Kleinberg, and Eva Tardos. Maximizing the Spread of Influence through a Social Network. In ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 137–146, 2003.
- [2] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance. Cost effective outbreak detection in networks. In Proceedings of the 13th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2007.
- [3]Sharan Vaswani, Laks V.S. Lakshmanan, Mark Schmidt. Influence Maximization with Bandits.
- [4]Alihan Hüyük, Cem Tekin. Thompson Sampling for Combinatorial Network Optimization in Unknown Environments
- [5] Wei Chu, Lihong Li, Lev Reyzin, Robert E. Schampire. Contextual Bandits with Linear Payoff Functions, page 2