Master's Thesis

Online Influence Maximization in Temporal Networks

Oleksii Kyrylchuk

18M38268

Graduate Major in Artificial Intelligence
School of Computing
Tokyo Institute of Technology

Supervisor: Murata Tsuyoshi

July, 2020

Abstract

Influence Maximization (IM) problem of finding the most influential nodes in a network is a significant problem in network analysis. It is used in both disease spreading simulation and online marketing. Until now, researchers have focused on solving IM for cases where we know specific information about the network, like the likelihood of a person telling about something to their friend (Offline Influence Maximization), and where the network is static. But in reality, we usually do not know different parameters about the network (Online Influence Maximization) and the network itself changes over time, which makes online IM applied to temporal networks a relevant problem to solve. This work solves it by taking one of the best online IM algorithms, IMLinUCB, and adapting it to temporal networks. The resulting algorithm, TIMLinUCB, is then tested on relevant social networking datasets like Facebook and Digg and shown to be more efficient than the competing IM algorithm RSB.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research background

1.1.1 Networks and Influence Maximization (IM)

A network is a collection of nodes joined by edges. Networks are extremely useful tools when it comes to modeling complicated real-world phenomena like social interactions, power grids, transportation, protein structures et cetera.

While the amount of problems that can be modeled with their help is extremely large, this thesis is focusing on a particular problem in network science called the Influence Maximization (IM) problem [14]. The definition of the problem is as follows: "find a set of nodes that can maximize the spread of information in the network". For example, if you were a company owner trying to promote their product on social media, you would first run an Influence Maximization algorithm on the network and then contact the people found by the algorithm and ask them to promote your products - if they agree to do so, you will know that you maximized the number of potential customers you are reaching out to and consequentially getting the most out of the money you spent on advertising.

The set of most influential nodes that you can use to advertise the product is also called the *seed set*, and the spread of information is usually simulated by using a certain *diffusion model* that determines whether a node propagates the information to their neighbors or not. Influence Maximization problem is NP-hard, making exact solutions very computationally expensive. However, it is still possible to obtain an approximate solution [2].

1.1.2 Diffusion models

Below we describe diffusion models that are the most actively used in modern research [2] [20] - Independent Cascade (IC) [28], Linear Threshold (LT) [28], Triggering (TR) [28], and Shortest Path (SP) [16].

Independent Cascade (IC) model

Independent Cascade model is a well-studied model, popularized by Kempe et. al in [15]. The model's algorithm is presented below (Algorithm 1). In Independent Cascade, every node in a network can be either active (influenced) or not. At the beginning of the simulation, every node except for the seed ones is inactive. At every iteration of the simulation, every active node a might activate their neighbor u with a certain probability $p_{a,u}$, which is called the *activation probability.* An inactive node can become active but not vise versa, and each node gets only one chance to activate their neighbors. The simulation ends when no new activated nodes in an iteration.

Algorithm 1: Independent Cascade (IC) model

```
Input: Graph $G$
       Seed set $S$
       Activation probabilities $p$
Output: $I$, the set of influenced nodes
Initialization: Set seed sets nodes to be currently active $A \leftarrow S$
1  for $t = 0, 1, 2...T$ do
2      for every active node $a_i \in A_t$ do
3          for every neighbor $u_k$ of $a_i$ do
4              Add $u_k$ to $A_{t+1}$ with the activation probability $p_{a_i,u_k}$
5          end
6          Add $a_i$ to the influenced nodes $I = I \cup a_i$
7          Remove $a_i$ from the current nodes $A = A \setminus a_i$
8      end
9      if $A \in \emptyset$ then
10         return $I$
11     end
12 end
```

Linear Threshold (LT) model

Linear Threshold is another seminal diffusion model, introduced by Granovetter in [9]. As opposed to the IC model introduced above, LT activates the node if a "sufficient" amount of neighbors are active.

In order to achieve this, it introduces two values - one associated with nodes, called threshold $\theta \in [0, 1]$, and the edge weights $b \in [0, 1]$. A node u will thus become active in the iteration t of the simulation if and only if the sum of the edge weights of all of its active neighbors is larger than its threshold $\sum_{v \in Activeneighbors(u)} b_{v,u} > \theta_u$. The simulation starts with the seed set nodes set to active and terminates when no new nodes are activated in an iteration.

Algorithm 2: Linear Threshold (LT) model

```
Input: Graph $G$
       Seed set $S$
       Edge weights $b$
       Thresholds $u$
Output: $A$, the set of influenced nodes
Initialization: Set seed sets nodes to be active $A \leftarrow S$
1  for $t = 0, 1, 2...T$ do
2      for every inactive node $u_i \notin A_t$ do
3          for every neighbor $v_k$ of $u_i$ do
4              $A_{t+1} = A_{t+1} \cup u_i$ if $\sum_{v \in neighbors(u) \cap A_t} b_{v,u} > \theta_u$
5          end
6      end
7      if no new nodes were activated then
8          return $A$
9      end
10 end
```

According to the survey done by Li, et al. [20], most researchers select the edge weights b

by choosing them from a set $\{0.1, 0.01, 0.001\}$ at random. They could not find an approach that would set the edge weights using some data obtained from a real-world dataset.

Triggering (TR) model

The triggering model proposed by Kempe et al. in [15] is an attempt to combine and generalize the Independent Cascade and Linear Threshold models.

The core idea of this model is to choose a subset of the node's neighbors that can activate it for every node u, called the "triggering set". The triggering set is then used to activate the node - if any of the neighbors of u are active and in the triggering set, the node is activated.

Similarly to IC and TR, we use the number of activated nodes as our metric. The triggering set is chosen based on a probability $P_{trig}(u)$ that is provided for every node u in the network.

Algorithm 3: Triggering (TR) model

```
Input: Graph $G$
       Seed set $S$
       Triggering probabilities $P_{trig}$
Output: $A$, the set of influenced nodes
Initialization: Set seed sets nodes to be currently active $A \leftarrow S$
1  for $t = 0, 1, 2 ... T$ do
2      for every inactive node $u_i \notin A$ do
3          Choose a triggering set $Trig_{u_i}$ based on $P_{trig}(u_i)$
4          if $\exists$ neighbor $v$ of $u_i \in Trig_{u_i}$ then
5              Activate $u_i$: $A = A \cup u_i$
6          end
7      end
8      if No new nodes were activated then
9          return $A$
10     end
11 end
```

Shortest Path (SP) model

Kimura et al. proposed the Shortest Path model in [16], the algorithm (Algorithm 4) for which you can find below. It is a variation of the IC model, only allowing a node to activate its neighbor at the iteration t equal to the distance between that node and the initial active set $t = d(A, v)$. A variation of the SP model, SP1 extends the time slot to $t = d(A, v)$ *or* $d(A, v) + 1$. The Shortest Path model was proposed to reduce the computational complexity and speed up the processing of the IC model, especially in large networks.

While it is computationally faster, it approximates the influence generated by the IC model - meaning that if you want to be more precise in your calculations it is better to use Independent Cascade.

Algorithm 4: Shortest path (SP) model

```
Input: Graph $G$
       Seed set $S$
       Activation probabilities $p$
Output: $I$, the set of influenced nodes
Initialization: Set seed sets nodes to be currently active $A \leftarrow S$
1  for $t = 0, 1, 2...T$ do
2      for every active node $a_i \in A_t$ do
3          for every neighbor $u_k$ of $a_i$ do
4              if $t = d(u_k, A_0)$ then
5                  Add $u_k$ to $A_{t+1}$ with the activation probability $p_{a_i,u_k}$
6              end
7          end
8          Add $a_i$ to the influenced nodes $I = I \cup a_i$
9          Remove $a_i$ from the current nodes $A = A \setminus a_i$
10     end
11     if $A \in \emptyset$ then
12         return $I$
13     end
14 end
```

1.1.3 Offline Influence Maximization

Even after choosing a diffusion model, there are multiple ways of approaching Influence Maximization.

As you can see from the models above, the probability of a node influencing its neighbors (activation probability in IC, edge weight in LT) plays a large role in simulating the propagation and obtaining the overall influence that a seed set can provide. *Offline Influence Maximization* problem is defined as Influence Maximization problem where the edge activation probabilities are given [19].

Offline IM is well-researched as shown by the works of Kempe [15], Jiang [13], Wang [38], and Zhou [43] and remained the main way that people solved the Influence Maximization problem until Lei et al. introduced Online Influence Maximization in 2015 [19].

TIM algorithm

TIM is a state of the art Offline IM algorithm introduced by Tang, et al. in "Influence Maximization: Near-Optimal Time Complexity Meets Practical Efficiency" [32].

In order to understand the algorithm, we first have to define a *Reverse Reachable Set* - a key insight that enabled a whole family of Reverse Influence Samplings (RIS) algorithms [17] [5]. A *Reverse Reachable Set* (RR set) is defined as follows: Let v be a node in G, and g be a graph that is obtained by removing each edge e in G with the probability $1 - p(e)$, where $p(e)$ is that edge's activation probability. The RR set for v in g is the set of nodes in g that can reach v. You can see a visual explanation of a reverse reachable set in the figure 1.1.

The TIM algorithm itself consists of two phases:

1. **Parameter Estimation**, where TIM derives the parameter θ needed for node selection.

2. **Node selection**, where TIM samples θ random RR sets from G and then finds a

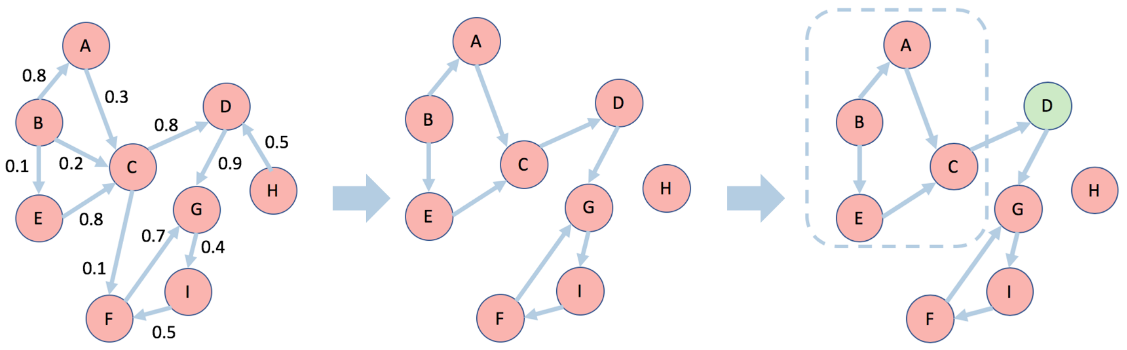


Figure 1.1: RR set of a node C (Source: [17])

seed node set S_k^* containing nodes that cover[1] a large number of the generated RR sets.

Node selection During node selection, the algorithm samples θ random RR sets from G and finds k nodes that are included in most RR sets. The k-sized seed set S_k^* of those nodes is then returned as the final result. You can see the pseudocode for node selection in the algorithm 5. Please note that the *for loop* itself is a standard greedy approach of solving the maximum coverage problem, a problem of finding k subsets of a set that, when joined together, include the largest amount of items from the original set (when compared to choosing k other subsets).

Algorithm 5: TIM - Node selection

```
Input: Graph $G$
       Seed set cardinality $k$
       Parameter $\theta$
Output: Seed set $S_k^*$
Initialization: Initialize a set $R = \emptyset$
                Initialize node set $S_k^* = \emptyset$
1 Generate $\theta$ random RR sets and insert them into R
2 for $j = 1...k$ do
3     Identify the node $v_j$ that covers the largest amount of RR sets in $R$
4     Add $v_j$ to $S_k^*$
5     Remove from $R$ all RR sets that are covered by $v_j$
6 end
7 return $S_k^*$
```

Parameter estimation The main complexity and the goal of parameter estimation of the TIM algorithm lies in choosing a appropriate θ - a number of RR sets to run node selection against. In [32], θ is defined as follows: $\theta = \lambda / OPT$, where λ is a parameter based on the Chernoff bounds [21] (equation 1.1) and OPT is the maximal influence for any size-k seed node set S.

Since OPT is essentially the optimal result of Influence Maximization, we can not directly calculate it, which creates the necessity to estimate it instead. The approximation

[1] A node v covers a set of nodes S if $n \in S$

is achieved by introducing parameters KPT and KPT^*. KPT is defined as the mean of the expected spread of the seed set S^*, generated by taking k samples from the set of all nodes (Note: there might be less than k nodes in S^* since the duplicates are eliminated). KPT^*, on the other hand, is a value that has a high probability to be in range $KPT^* \in [KPT/4, OPT]$ and is a replacement for OPT that TIM is using ($\theta = \lambda/KPT^*$). The algorithm 6 shows the steps used for calculating KPT^*. If you want to know more about how KPT^* and λ were derived, please refer to [32].

$$\lambda = (8 + 2\epsilon)n \cdot (l \log n + \log \binom{n}{k} + \log 2) \cdot \epsilon^{-2} \tag{1.1}$$

Algorithm 6: TIM - KPT estimation

```
Input: Graph $G$
       Seed set cardinality $k$
Output: Approximation of the maximal influence that can be obtained from
        using a seed set with $k$ seeds, $KPT^*$
1  for $i = 1 \ldots \log_2 n - 1$ do
2      Let $c_i = (6l \log n + 6 \log(\log_2 n)) \cdot 2^i$
3      Let $sum = 0$
4      for $j = 1 \ldots c_i$ do
5          Generate a random RR set R
6          $\kappa(R) = 1 - (1 - \frac{w(R)}{m})^k$
7          $sum = sum + \kappa(R)$
8      end
9      if $sum/c_i > 1/2^i$ then
10         return $KPT^* = n \cdot sum/(2 \cdot c_i)$
11     end
12 end
13 return $KPT^* = 1$
```

While the structure of TIM is somewhat complicated, the payoff is worth it - TIM has a complexity of $O((k + l)(n + m) \log n/\epsilon^2)$ (where k is the number of seed nodes, n is the number of nodes, m is the number of edges and ϵ and l are algorithm parameters) and returns a solution of an accuracy $(1 - 1/e - \epsilon)$ with the probability of at least $1 - n^{-l}$. Keeping in mind that the lower bound for Offline IM complexity is $\omega(m + n)$, the TIM's complexity is quite impressive since it only differs from the optimal one by the factor of $\log n$. It also supports the Triggering model covered above, which means that it also supports the IC and LT models. A notable point that we will return to later in the paper is that both the algorithm's accuracy and performance are heavily influenced by the parameter ϵ.

1.1.4 Online Influence Maximization

The problem that arises when dealing with Offline IM is that in real-world datasets we usually don't have the luxury of knowing the exact activation probabilities. This creates another problem of finding the activation probabilities before we are able to solve the IM problem, which led to creation of the Online Influence Maximization.

Online Influence Maximization problem is defined as the version of the Influence Maximization problem where the activation probabilities are either unknown or incomplete [19].

Since the problem of finding the influential nodes in the network in which we know the activation probabilities can be delegated to various Offline IM algorithms, the main issue in Online IM becomes finding activation probabilities that, when coupled with an Offline algorithm, produce the seed set S that generates the most influence. In order to solve this problem, Online IM algorithms usually employ either a *heuristic-based* strategy or an *Explore-Exploit* strategy that depends on running the Offline IM algorithm multiple times.

Examples of heuristic-based strategies are the *Random* heuristic, in which we select the seed nodes randomly or the *MaxDegree* heuristic, in which the activation probability of a node depends on its out-degree.

Explore-Exploit strategies work in the following way: we *exploit* by executing an Offline IM algorithm with certain activation probabilities and obtaining the seed nodes. These seed nodes may not be optimal for the network since we do not know how close our activation probabilities are to the real-world ones - we alleviate this problem by *exploring* in order to improve those activation probabilities. One of the main problems that appear when using the Explore-Exploit strategy is balancing the amount of exploration and exploitation - and it is also one of the core issues in *Multi-Armed Bandit* literature [19] [31] [26].

Two Exploit-Explore algorithms are described below: the Confidence-Bound algorithm (Algorithm 7), which was proposed alongside Online IM in [19] and IMLinUCB (Algorithm 8), which was developed by Adobe and is currently considered to be a state-of-the-art [39] [6] [41] algorithm for Online Influence Maximization.

Confidence-Bound algorithm

Algorithm 7: Confidence-Bound (CB) algorithm using an Exploit/Explore strategy

```
Input: Graph $G$
       Seed set cardinality $k$
Output: Seed set $S$ that maximizes the influence in $G$
Initialization: Set seed sets nodes to be currently active $A \leftarrow S$
1 for edge $e \in G$ do
2     $\mu_{uv} \leftarrow \frac{\alpha_{uv}}{\alpha_{uv}+\beta_{uv}}$
3     $\sigma uv \leftarrow \frac{1}{\alpha_{uv}+\beta_{uv}}\sqrt{\frac{\alpha_{uv}\beta_{uv}}{\alpha_{uv}+\beta_{uv}+1}}$
4     $p_{uv} \leftarrow \mu_{uv} + \theta\sigma_{uv}$
5 end
6 $G' \leftarrow G$ with activation probabilities $p_{u,v} \forall (u,v) \in E$
7 $S \leftarrow \text{OfflineIM}(G', k)$
8 return $S$
```

In the algorithm above, the activation probability of an edge from node u to node v is modeled as a Beta distribution with the probability density function you can see in the equation 1.2. In the equation, B is the Beta function that ensures that the total probability mass is 1 and α_{ij} and β_{ij} are the distribution parameters that represent our prior knowledge of the graph and in the absence of a good prior are set to $\alpha = \beta = 1$.

$$f_{P_{u,v}}(x) = \frac{x^{\alpha_{u,v}-1}(1-x)^{\beta_{u,v}-1}}{B(\alpha_{uv}, \beta_{uv})} \tag{1.2}$$

We then use a combination of the mean and standard deviation of P_{uv} to represent $p_{u,v}$ of a specific edge. The mean $p_{u,v} = \mathbb{E}[P_{uv}]$ is used during exploitation, when we run an

Offline IM algorithm on the network. Since the distribution P_{uv} can be highly uncertain (with the default parameters, it is a uniform distribution $B(1, 1)$, which means that any value between 0 and 1 can be the activation probability), we give the nodes more chances to be activated during exploration, by estimating them not just as a mean but a sum of a mean and the standard variance of P_{uv}, $p_{u,v} = \mathbb{E}[P_{uv}] + \sigma_{uv}$.

In a Confidence-Bound algorithm, we use the parameter θ to control the balance of exploration and exploitation of the activation probability $p_{u,v} = \mathbb{E}[P_{uv}] + \theta\sigma_{uv}$. A global parameter is chosen in favor of assigning a new θ to every edge to reduce the number of parameters needed to optimize the algorithm, making it simpler and more efficient.

If $\theta = 0$ then the algorithm only exploits the current nodes, and if $\theta = 1$ then CB only explores. In general, if $\theta > 0$ then CB treats the activation probabilities as undervalued and tries to make them larger and if $\theta < 0$ then it tries to reduce them.

θ can be either set by a user of the algorithm or determined automatically using feedback obtained when running the algorithm. For more details on how to set it automatically please refer to Chapter 6 of Lei, et al.'s work [19].

IMLinUCB algorithm

IMLinUCB [39] is an Online IM algorithm designed to maximize the number of nodes influenced by the chosen seed set under the IC model. This is also equivalent to minimizing the difference in affected nodes between its results and a result of an Offline IM algorithm, since it is assumed that the Offline IM algorithm will usually outperform the Online one. In order to achieve this goal, it creates a linear model for estimating the activation probabilities of edges $\overline{w}$. They associate each edge $e \in E$ with a known feature vector $x_e \in \mathbb{R}^d$ (d is the dimension of the feature vector and assume that there exists an unknown coefficient vector $\theta^* \in \mathbb{R}^d$ such, that $\overline{w}_e$ is well-approximated by $x_e^\intercal \theta^*$. They formally define being well-approximated as the difference $\rho \triangleq \max_{e \in E} |\overline{w}_e - x_e^\intercal \theta^*|$ being small.

The IMLinUCB paper showcased two methods for obtaining features x_e - the first one is to use a column associated with the edge's id in the identity matrix $I \in \mathbb{R}^{|E|x|E|}$ and the second method involves using an embedding algorithm like *node2vec* [10]. Since node2vec only generates node embeddings, the edge embeddings are generated by multiplying the features for two nodes that are being joined by a particular edge. If the network is large, using an embedding algorithm is preferred since it can produce fewer features than using the identity matrix, leading to much faster computations (even though it can lead us to not obtaining the most optimal solution).

The algorithm itself (Algorithm 8) starts out by initializing the variables B and M that are used in order to save the results of past observations. Specifically, if we combine the feature vectors of all observed edges in the iteration i into a matrix X_i (a row of a matrix corresponds to the feature vector of one edge) and combine their outcomes (activated or not) into a vector Y_i, $M_i = I + \sigma^{-2} X_i^\intercal X_i$ and $B_i = X_t^\intercal Y_i$. After the initialization is complete, it enters the learning loop, that consists of three steps. First, it computes an Upper Confidence Bound $U_i(e)$ for all edges. This is the "ceiling" of our estimation of the activation probability, with $x_e^\intercal \theta^*$ serving as the mean. Second, the algorithm chooses a set of source nodes based on an Offline IM algorithm under the IC model and $U_i(e)$ that we generated in our last step. The algorithm also receives the total number of nodes influenced in this iteration as well as the edge semi-bandit feedback, or the edges that connect active nodes to the nodes they activated in this iteration. Finally, the feedback is used to update B and M in order to improve the performance of the next iteration of the algorithm.

Algorithm 8: IMLinUCB algorithm

```
Input: Graph $G$
       Seed set cardinality $K$
       Oracle $oracle$
       Feature vector $x_e$
       Algorithm parameters $\sigma, c$
Output: Source node set $S$
Initialization: $B_0 \leftarrow 0 \in R^d, M \leftarrow I \in R^{d \times d}$
1  for $i = 0, 1, ..., n$ do
2      $\bar{\theta}_{t-1} \leftarrow \sigma^{-2} M_{t-1}^{-1} B_{t-1}$
3      $U_i(e) \leftarrow Proj_{[0,1]} \left( x_e^\intercal \bar{\theta}_{t-1} + c\sqrt{x_e^\intercal M_{t-1}^{-1} x_e} \right)$
4      Choose $S_i \in OfflineIM(G, K, U)$ and observe the edge-level semi-bandit
         feedback
5      Initialize $M_t \leftarrow M_{t-1}$ and $B_t \leftarrow B_{t-1}$
6      for all observed edges $e \in E$ do
7          $M_t \leftarrow M_t + \sigma^{-2} x_e x_e^\intercal$ and $B_t \leftarrow B_t + x_e w_t^e$
8      end
9  end
10 return the seed set $S$, influencing the largest amount of nodes
```

1.1.5 IM in Static and Temporal Networks

While the Offline/Online separation of IM algorithms is one way to look at them, another distinguishing factor is the structure of the network itself - is it a *static* network or a *temporal* one? A *static* network is a representation of the real-world network at one specific point in time - its nodes and edges do not change. A *temporal* network, as opposed to the static one, can change over time - both its nodes and edges can be present or absent from the network at a specific time point.

While the temporal network models some real-world activities more precisely than a static one would, researching temporal network is a relatively young field [?], meaning that there are less algorithms designed specifically for temporal networks.

	Offline	**Online**
Static	"Maximizing the spread of influence through a social network" [15] "Community-based greedy algorithm" [38] "Upper Bound based Lazy Forward" [43] "Simulated annealing based influence maximization" [13]	"Online influence maximization with semi-bandit feedback" [39] "Model-independent online learning for influence maximization" [35] "Influence Maximization with Bandits" [36] "Factorization Bandits for Online Influence Maximization" [42]
Temporal	"On influential node discovery in dynamic social networks" [1] "Diffusion maximization in evolving social networks" [7] "Dynamic influence analysis in evolving networks" [23] "Upper Bound Interchange Greedy algorithm" [29]	"Randomized Multi-Armed Bandit" [3]

Table 1.1: IM Algorithm classification

As of the writing of this paper, IM in Static networks was heavily researched by many different specialists [2] [30], with various algorithms covering both Offline [15] [38] [43] [13] and Online cases [39] [35] [36] [42].

The research on IM in Temporal networks started relatively recently, with most algorithms focusing on solving the Offline IM problem for temporal networks [1] [7] [23] [29]. In fact, the survey conducted by Li et al. in 2018 [20] even adds the Online IM algorithm used from [19] to the "Dynamic" category with other Temporal IM algorithms, since it assumes that the activation probabilities are unknown and their estimations will change while iterating through the algorithm (as you can see in the algorithm 7 above). The other algorithms presented in the survey like [24] or adaptations of the Static network algorithms like the ones presented in [22] all assume that the activation probabilities (susceptibility in [22]) are known beforehand.

Online IM in a temporal network - RSB

The algorithm *RSB*, proposed by Bao et al. in [3] claims to be the first IM algorithm operating on temporal networks without assuming that the edge activation probability is known beforehand. Because of this lack of research in the field, they compare RSB with an algorithm that chooses seed nodes randomly and an Online IM algorithm designed for static networks. RSB uses the Exploit-Explore strategy (explained above) in order to achieve the Online Influence Maximization at every time step - a more detailed explanation of the algorithm can be found below.

Metrics The paper introducing RSB proposes multiple metrics for measuring the efficacy of the algorithm. One of them is the total number of nodes activated by a seed set S_t at every time step t in the network. It is also equivalent to minimizing *regret*, or the difference between RSB and an Offline IM algorithm that knows everything about the network. Another one is *weak regret*, or the difference in activated nodes between the reward obtained by RSB and a reward obtained by selecting one seed set by using an Offline IM algorithm and using it as our seed set at every time step.

They design their algorithm to minimize weak regret (assuming that the Offline IM algorithm will perform better than RSB). By doing it they also minimize regret as well as maximize the number of activated nodes in the network, since the larger the amount of nodes that are activated by seed sets chosen by RSB leads to reducing the gap between a result of the Offline IM algorithm and the Online IM algorithm.

Algorithm overview In RSB, a node u under a certain subset S of the network is called an arm and denoted $u|S$. The main idea of the algorithm is to select a seed set S_t in every time step based on a time-varying weight w_t associated with arms that serve as candidates of being added to the seed set.

Exploitation is performed by using the weights from the last time step in order to calculate the current ones, and exploration - by adding a constant $\frac{\gamma}{N}$ (where $\gamma \in (0, 1]$ is a user-set value to every weight) to every weight, giving a chance to the arms that were never tried before.

You can see how the above strategy is implemented in the algorithm 9. Every seed of the seed set S_t is selected sequentially, with the arm weights w_t starting out equal and then being adjusted based on the parameter v - an auxiliary parameter used to persist the past reward information of the arm $n|S_t^{(1:k-1)}$. After calculating the weights, the algorithm normalizes them (line 7) and uses them to choose the arm $\alpha|S^{1:k-1}$ as the k_{th} node of the seed set (line 9). It then (line 11) runs a simulation that calculates the reward of adding α to the seed set - the amount of new nodes that were activated by it. The results of the simulation are used to update the parameter v_t of every arm (line 17), with $\exp \frac{\gamma \hat{r}_t^k(n|S_t^{(1:k-1)})}{NC}$ being the unbiased estimation of the reward of the arm $n|S_t^{(1:k-1)}$. The algorithm also tries to equalize (line 12) the rewards by dividing the actual number of influenced nodes by the probability of selecting that arm, which helps improve the algorithm in cases where the probability of choosing the arm α is low.

The complexity of running RSB at one time stage t is shown to be $O(KN)$ in its paper [3], which means that we iterate through every node once for every possible seed set "slot" that we have. In turn, this makes the overall complexity of the algorithm $O(tKN)$, where K is the number of seeds we have to select and N - the number of nodes in the network.

Parameters The algorithm relies on two hyperparameters: γ that serves as the ratio between exploration and exploitation and was explained above, and C, which is related to the largest spread brought in by a seed. If RSB is used for regret analysis, C is required to be $C \geq \frac{\gamma r_t^k n|S)}{N q_t^k(n|S)}$ but the paper authors state that when running the algorithm C is set empirically and might not follow the above constraint.

Diffusion models In order to run their experiments, RSB introduces three time-varying diffusion models, described below. Please note that these models are all using the same base idea as the *Independent Cascade* model described in the section 1.1.2, applied to a temporal graph, with the main difference between the models being the way of generating the activation probabilities of the edges in the network. The authors also claim that RSB can support any model as long as the information spread from an active node can be modeled as a random variable.

The first model they introduce is called the *Weighted Cascade (WC)* model. The activation probability of an edge (u, v) at time t is set to the inverse of the indegree of the node v_t: $p_{u,v}^t = \frac{1}{d_m^t}$. As such, the node's probability varies over time - it will be low if it has a lot of inbound edges and high if it has few inbound edges.

The second model they introduce is the *Trivalency (TR) model*. In this model, the influence probability of an edge is randomly chosen from a set of $\{0.1, 0.01, 0.001\}$ at every time step. Those three values correspond to the three social tie types - strong, medium and weak.

The third and last model they introduce is the *Fluctuating Reward (FR)* model. In it, the activation probabilities evolve over time in a fashion similar to a sinusoidal wave. At time t_0, the probabilities are sampled randomly from a uniform distribution $[0, 0.1]$ and then they increase or decrease at a constant rate $\frac{0.3}{T}$ (T is the number of time steps) until they either reach the maximal value 0.1 or the minimal value 0.

Algorithm 9: RSB algorithm

```
Input: Set of nodes N
       Seed set cardinality $K$
       Algorithm parameters $\gamma, c$
Output: T seed sets $S_t^{(1:K)}$, one for every time step t
Initialization: Set $v_1^{n|S_1^{(1:k-1)}} = 1, \forall n \in N, k = 1, ..., K$ and $S \leftarrow \emptyset$
1  for $t = 1, 2, ..., T$ do
2      for $k = 1, 2, ..., K$ do
3          for each node $n \in N$ do
4              Set $w_t^{n|S_t^{(1:k-1)}} = (1-\gamma)\frac{v_t^{n|S_t^{(1:k-1)}}}{\sum_{n' \in N} v_t^{n'|S_t^{(1:k-1)}}} + \frac{\gamma}{N}$
5          end
6          for each node $n \in N \setminus S_t^{(1:k-1)}$ do
7              $q_t^{n|S_t^{(1:k-1)}} = \frac{w_t^{n|S_t^{(1:k-1)}}}{\sum_{n' \in N \setminus S_t^{(1:k-1)}} w_t^{n'|S_t^{(1:k-1)}}}$
8          end
9          Draw an arm $\alpha|S_t^{(1:k-1)}$ according to the distribution
           $\{q_t^{n|S_t^{(1:k-1)}}\}_{n \in N \setminus S_t^{(1:k-1)}}$
10         Receive a reward $r_t^k(\alpha|S_t^{(1:k-1)})$
11         Set $S_t^{(1:k)} = S_t^{(1:k-1)} \cup \{\alpha\}$
12         Set $\hat{r}_t^k(\alpha|S_t^{(1:k-1)}) = \frac{r_t^k(\alpha|S_t^{(1:k-1)})}{q_t^{\alpha|S_t^{(1:k-1)}}}$
13         for $n \in N \setminus \{\alpha\}$ do
14             Set $\hat{r}_t^k(n|S_t^{(1:k-1)}) = 0$
15         end
16         for each arm $n|S_t^{(1:k-1)}$, $\forall n \in N$ do
17             Update $v_{t+1}^{n|S_{t+1}^{(1:k-1)}} = v_t^{n|S_{t+1}^{(1:k-1)}} \exp \frac{\gamma \hat{r}_t^k(n|S_t^{(1:k-1)})}{NC}$
18         end
19     end
20     Store $S_t$ to return it later: $S = S \cup S_t$
21 end
22 return $S$
```

1.2 Research goals

As described in the section 1.1.5, currently there exists a lack of Online IM algorithms for Temporal networks. This thesis aims at filling in the hole of absent Online IM algorithms applied to Temporal networks by taking a state-of-the-art Online IM algorithm IMLinUCB [39] described above (Algorithm 8) and adapting it into an algorithm that can be applied to a Temporal Network. In order to simulate the propagation, Influence Cascade model will be applied at every time step of the temporal network.

It is then compared to its direct competitor, RSB [3] using real-world social networking datasets such as Digg and Facebook to determine the efficacy and efficiency of the algorithm. An offline algorithm TIM will be used on every time step of the network as well - this should produce a close-to-optimal result that will be used as a baseline for the comparison.

The main metric that will be used in the comparison is the efficacy of the algorithm, or the amount of influence it will be able to gather with the chosen seed nodes, measured by the number of activated nodes after simulating the influence spread with the IC model. The second metric used is efficiency, or the amount of time it took to run the algorithm.

Chapter 2

TIMLinUCB

2.1 Theory

The main idea of the adaptation is to run IMLinUCB on every time step t, optionally preserving the parameters of the algorithm in between time steps. It's important to note that saving the parameters from last time step is not always beneficial since the network could have changed tremendously in a small time period, making our old parameters influence the result in a negative way.

Algorithm 10: TIMLinUCB algorithm

```
Input: Temporal graph $G$
       Seed set cardinality $K$
       Edge feature matrix $F$
       Algorithm parameters $\sigma, c, \epsilon > 0$
       Boolean $persist\_parameters$
Output: A set $S$, containing T source node sets $S_t$, one for every time step
Initialization: $B_0 \leftarrow 0 \in R^d, M \leftarrow I \in R^{d \times d}, S \leftarrow \emptyset$
1 for $t = 0, 1, ..., T$ do
2     if $persist\_parameters$ and $t > 0$ then
3         Populate $M_t$ and $B_t$ based on the last time step
4     end
5     $S_t$ = IMLinUCB($G_t$, $K$, $TIM$, $F$, $\sigma$, $c$, $\epsilon$, $M_t$, $B_t$)
6     $S = S \cup S_t$
7 end
8 return $S$
```

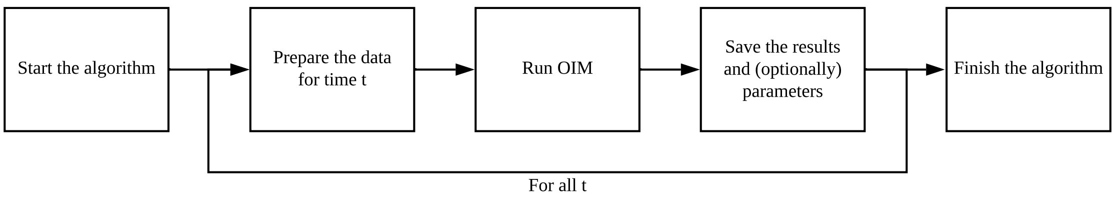

Start the algorithm
Prepare the data for time t
Run OIM
Save the results and (optionally) parameters
Finish the algorithm
For all t

Figure 2.1: TIMLinUCB (high-level view)

Performance and Parallelism

This structure of the algorithm makes its complexity $\mathcal{O}(T * \mathcal{O}(IMLinUCB))$, which, when expanded, becomes $\mathcal{O}(Tmd^2\mathcal{O}(OfflineIM))$, where T is the number of time steps in the temporal network, m is the number of edges in the network, d is the number of edge features, and $\mathcal{O}(OfflineIM)$ is the complexity of the chosen Offline IM algorithm.

Since TIMLinUCB is using TIM as the Offline IM algorithm, the complexity becomes $\mathcal{O}(Tmd^2(k+l)(n+m)\log n/\epsilon^2)$, where n is the number of nodes, and l and ϵ are TIM parameters. Since in the TIM implementation used in TIMLinUCB, $l = 0.5$ is constant, the complexity thus becomes $\mathcal{O}(Tmd^2k(n+m)\log n/\epsilon^2)$, meaning that it scales linearly with the number of time steps in the network t and the number of seeds k, $n \log n$ with the number of nodes n, quadratically with the number of edge features d and the number of edges m and exponentially with ϵ.

We can lower the execution time of the algorithm without changing the complexity by parallelizing the IMLinUCB executions. As every iteration of IMLinUCB depends on the previous one, parallelizing the IMLinUCB itself is not feasible, but we can run multiple IMLinUCB instances in parallel. While this approach removes the ability for TIMLinUCB to persist features (since with feature persistence, every time step depends on the previous one), it can be used to drastically reduce the runtime of the algorithm.

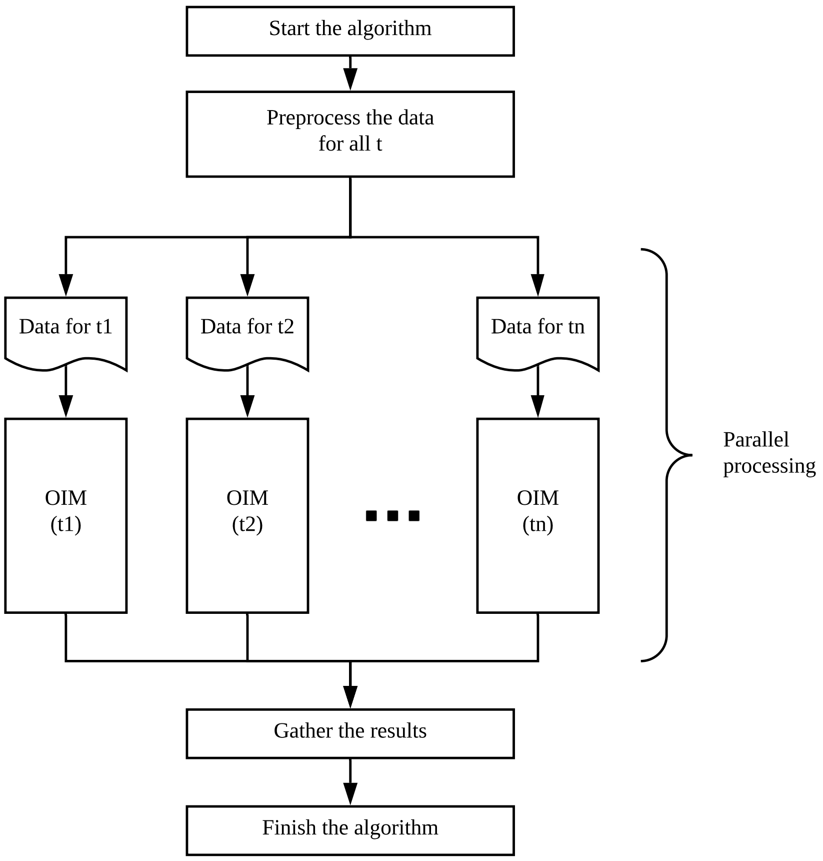


Figure 2.2: Parallel processing of time steps within TIMLinUCB

We can also run multiple TIMLinUCB instances in parallel, which is useful when we are looking for the best parameters for the algorithm - you can just queue multiple TIMLinUCB versions to run and then select and stick to the best-performing version.

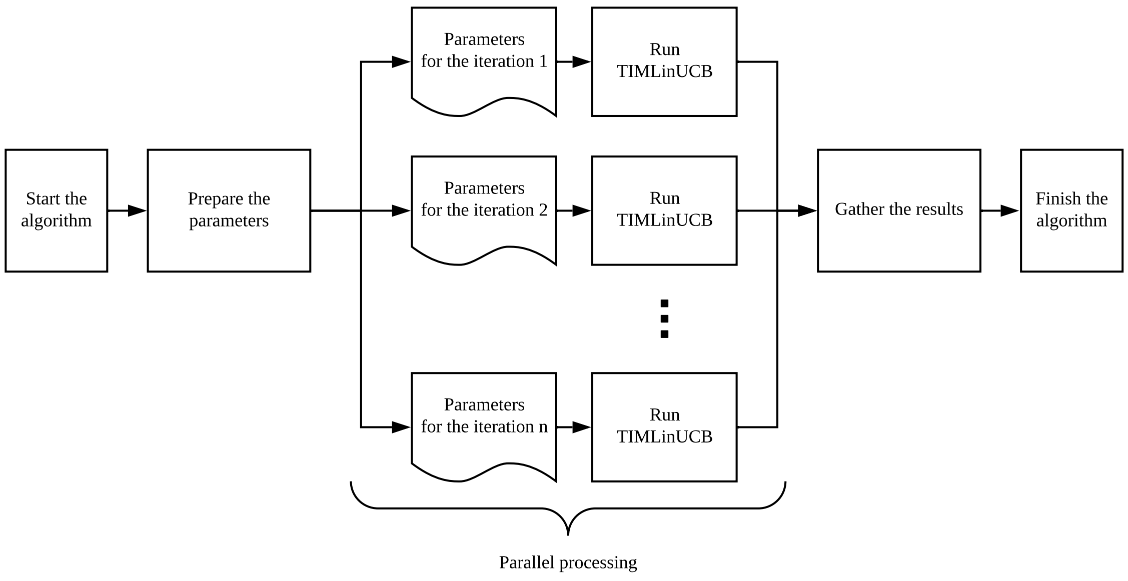


Figure 2.3: Parallel processing of TIMLinUCB

2.2 Implementation

2.2.1 Experiment details

Programming Language and Packages

The following algorithms were implemented in Python 3.7 for this thesis:

- TIMLinUCB
- IMLinUCB
- RSB
- Influence propagation under the IC model

The main packages used were *pandas* [25] [40] and *numpy* [34]. The parallelism was achieved with the help of the package *joblib* and the plots were generated with the help of *matplotlib* [12]. Algorithms TIM [32] [33] and node2vec [10] were used by calling their respective binaries. The TIMLinUCB code is available on GitHub [18].

Features for IMLinUCB

The original IMLinUCB paper suggests two approaches for generating edge features x_e. The first one is to create an identity matrix $I^{|E|\times|E|}$, that would represent every edge by one of its rows. The second one is to use a node embedding algorithm node2vec [10] to create node features, and then convert them to edge features by multiplying the features of the source and target nodes $f_{edge} = f_{source\ node} * f_{target\ node}$.

Since an identity matrix of that size would be quite large and take a lot of resources to process, we chose to use the second approach and generated node2vec features for every dataset before running the TIMLinUCB algorithm.

Datasets

The datasets used in this paper were taken from the Network Repository [27]. They were chosen based on the notion that in order to analyze and compare Online IM algorithms for Temporal networks, the networks would have to be as close to the real-world ones as possible.

Both Facebook [37] and Digg [11] datasets were generated based on real social networking websites, which means that the performance of the algorithm when using these datasets could reflect the performance when finding the most influential nodes in a social network for marketing or quarantine purposes. The original datasets are really large, with Facebook containing information about $60,290$ users and $1,545,686$ friendships gathered from the New Orleans regional network and Digg - about $279,630$ users and $1,731,653$ connections between them.

Dataset		$\boldsymbol{t_0}$	$\boldsymbol{t_{50}}$	$\boldsymbol{t_{100}}$
Facebook	Nodes	185	5861	8450
	Edges	118	13685	28511
Digg	Nodes	1276	3463	4824
	Edges	1670	5815	7848

Table 2.1: Temporal characteristics of the datasets

In order to compare TIMLinUCB and RSB, we will use the data from the first 100 days of the networks. This is mirroring the performance evaluation of RSB [3] algorithm, where they use 100 time steps of Tencent Weibo traces and an artificially generated network to evaluate the performance of the algorithm. Unline RSB, however, that used a 7-day period from the Tencent Weibo dataset and repeated it until they had 100 days, the datasets in this paper are large enough to not have to resort to such measures. You can find the temporal network characteristics in the table 2.1 - as you can see, both networks steadily gain connections and nodes over time as users are communicating and new users either join the social network or are scanned by the crawlers. Please also note that the Facebook algorithm is more sparse than Digg at first (there are fewer edges in it than there are nodes) but it becomes dense rather quickly - by the time t_{100} it has 3.3 times more edges than nodes while Digg's edge:node ratio is 1.63 : 1.

Diffusion model

This paper uses the *Independent Cascade* diffusion model introduced in the section 1.1.2 above. Since both Facebook and Digg datasets introduced above do not contain any information about the activation probabilties of the edges, which we need to be able to evaluate the Online IM algorithm that we created, we generate those probabilities for each edge by sampling the Uniform distribution $U(0, 0.1)$. The choice of the $[0, 0.1]$ range was guided by empirical evidence as shown in [4] [8].

There were multiple reasons for choosing the Independent Cascade model as the diffusion model for this algorithm. The first and most important reason is that both IMLinUCB and TIM, which our algorithm is based on, use the IC model in their experiments. They rely on it because it is one of the "[...] best known and studied models" [39] used by many other researchers [20] [2]. The diffusion models used by the competitor algorithm, RSB, being special cases of the IC model applied to a temporal network also contributed to this decision.

While the Shortest Path model (introduced in the section 1.1.2) could be used as well, it approximates the results produced by the IC model and we prefer the better efficacy provided by IC. We decided to not use the LT model because of the problem of selecting realistic edge weights. We think that extending the algorithm to support the Triggering model might be a worthwhile pursuit in the future, and state so in the "Future work" part of the Conclusion (Chapter 3).

2.2.2 Finding the best parameters

Grid search - Epsilon, C and Sigma

In order to get good results out of TIMLinUCB, we have to make sure that IMLinUCB and TIM produce good results as well - and hyperparameters that we pass into them play a large role in that.

In order to select the best parameters, this paper implements a grid search for them. You can find the results of iterating through multiple options for ϵ, σ and c in both datasets in the figures 2.4 and 2.5 below. In them, one heatmap represents a choice of a particular epsilon, with sigma being shown on the y-axis and c on the x-axis. The higher the average number of nodes influenced by a seed set generated by TIMLinUCB with the given ϵ, σ, and c, the lighter the corresponding area of the chart is and vise versa. The experiments were done by running IMLinUCB with 5 iterations on the first 20 time steps of every dataset. The figures 2.4 and 2.5 contain the results of searching for the 5 most influential seed nodes in the network - the results of finding 20 and 100 seed nodes were put in the appendix A.2 since they offer similar insights.

Of course, the optimal parameters will change over time as the network evolves, but it is not feasible to run this many iterations of TIMLinUCB over the entire network. A better approach is to narrow the results down to a small range of ϵ, σ and c and then test them over the entire network.

Facebook dataset By analyzing the heatmaps of the Facebook dataset (Figure 2.4), it becomes clear that lower ϵ leads to the better overall efficacy of the algorithm, which confirms our belief that TIM's accuracy depends on ϵ (as stated in the section 1.1.3, TIM returns a solution of an accuracy $(1 - 1/e - \epsilon)$). Smaller σ, the reverse learning rate of IMLinUCB, also generates better results - but it is likely that due to a rather small number of IMLinUCB iterations that smaller learning rate algorithms just did not train enough yet.

We can also see that having a small c ($c \approx 0.01$) and a large σ (and thus a smaller learning rate) generates worse results than the other parameter couplings, likely due to the linear parameter $c\sqrt{x_e^\intercal M_{t-1}^{-1} x_e}$ from the equation $U_i(e) \leftarrow Proj_{[0,1]}\left(x_e^\intercal \bar{\theta}_{t-1} + c\sqrt{x_e^\intercal M_{t-1}^{-1} x_e}\right)$ not being large enough to influence the U_e.

Digg dataset The results for the Digg dataset (figure 2.5) appear to be quite different from the Facebook one at first sight, but they are not. The notion of smaller ϵ producing better results is confirmed once again, as well as the notions of having low c and a large σ making the learning process too slow and thus producing worse results. We can see that c plays a larger role in this case, however, with algorithms that used $c = 0.01$ or $c = 0.1$ performing very poorly.

The reward is also lower than the one in the Facebook dataset due to the fact that the Digg dataset is more sparse at the beginning.

It also must be said that running TIMLinUCB with a low ϵ takes much more time than running it with a high one. You can see that demonstrated on the figure 2.6 that was generated by running TIM (algorithm relying on the ϵ parameter) using the data from the Facebook dataset (September 2006 - July 2007).

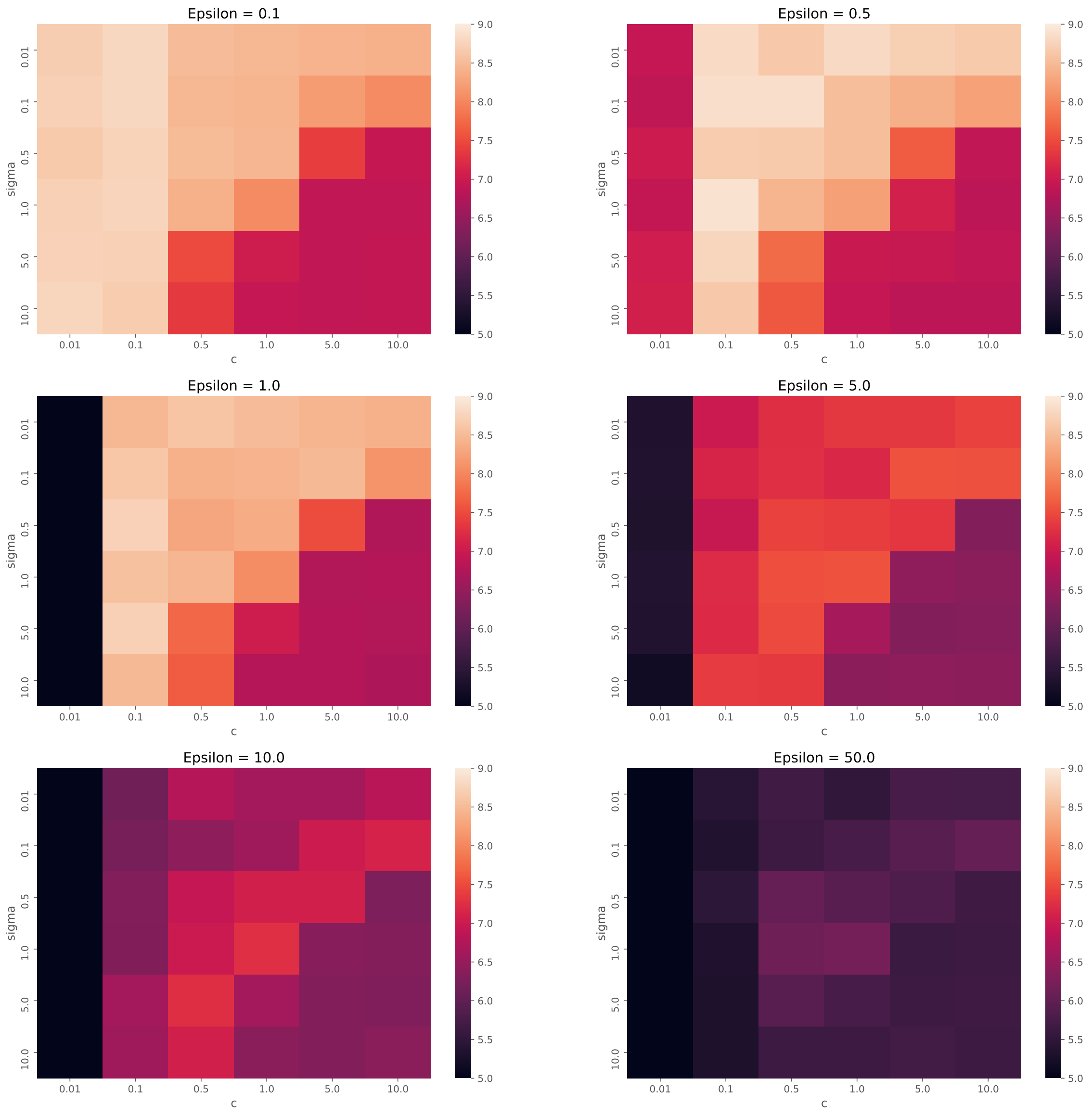


Figure 2.4: TIMLinUCB: Grid search for the best parameters (Facebook dataset, 5 seed nodes)

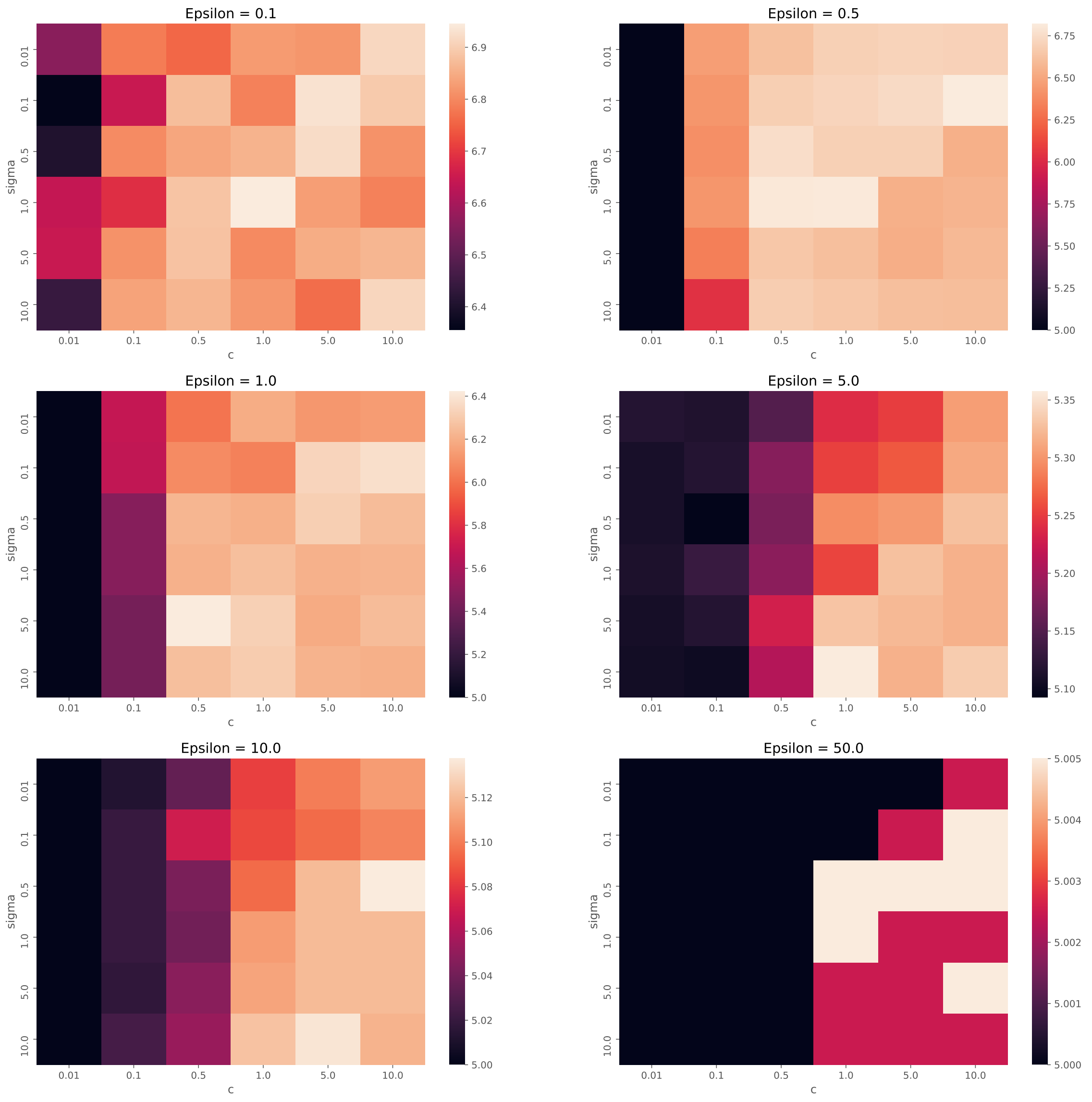
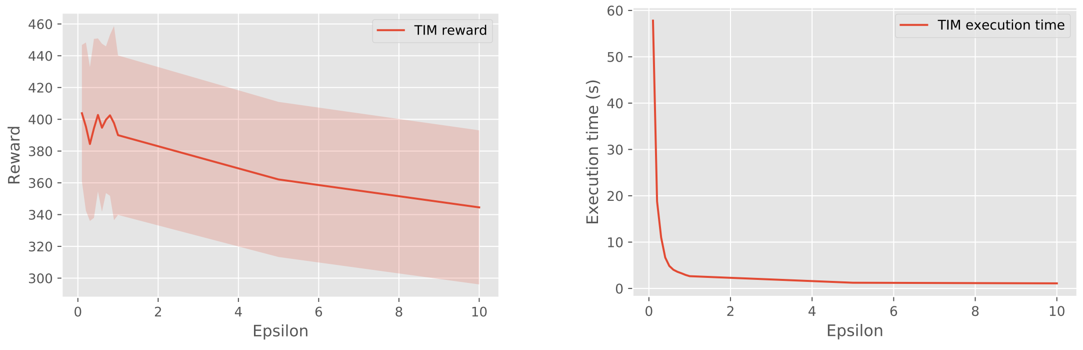


Figure 2.5: TIMLinUCB: Grid search for the best parameters (Digg dataset, 5 seed nodes)

With ϵ decreasing, the reward is increasing linearly while the time spent on the algorithm is increasing exponentially, which is also shown in the complexity measurement of the algorithm 8. This means that as our choice for ϵ becomes smaller and smaller we will gain fewer benefits for spending much more time on the problem. In the end, choosing larger or smaller ϵ will depend on the purpose of the experiment you are conducting as well as your circumstances - if you have a lot of time and a very powerful server to run the algorithm on, choosing a very small ϵ might be worth it.



(a) Reward per ϵ (b) Time of execution per ϵ

Figure 2.6: TIM performance

Number of IMLinUCB iterations

Another important parameter of TIMLinUCB is the number of IMLinUCB iterations. As you can see from the figure 2.7 below, even though it might seem like the difference is rather small at first, the gap between running IMLinUCB with 1 iteration and 30 grows over time.

This is because the number of IMLinUCB iterations is essentially the number of learning steps that the Online IM algorithm will take to learn the network and find the best approximations of the influence probabilities (or, in case of IMLinUCB, their upper bounds). In the figure 2.8, we can see that while the algorithm was learning and improving a lot in the first few steps, there wasn't much increase in the number of influenced nodes after running the algorithm for 15 iterations - we assume that the algorithm is close to a local minimum at that point, so all subsequent iterations influence the result less.

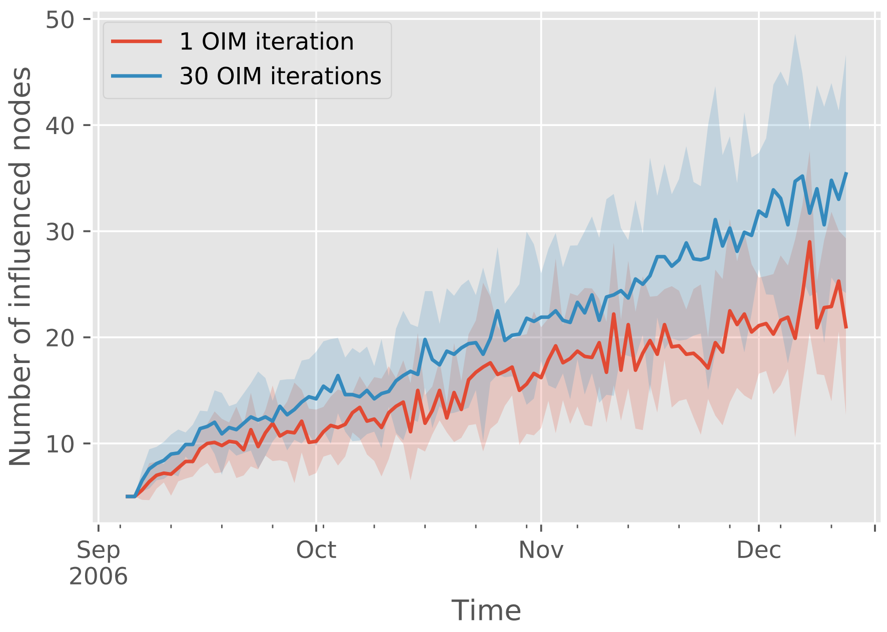


Figure 2.7: Number of influenced nodes over time with a different amount of OIM iterations

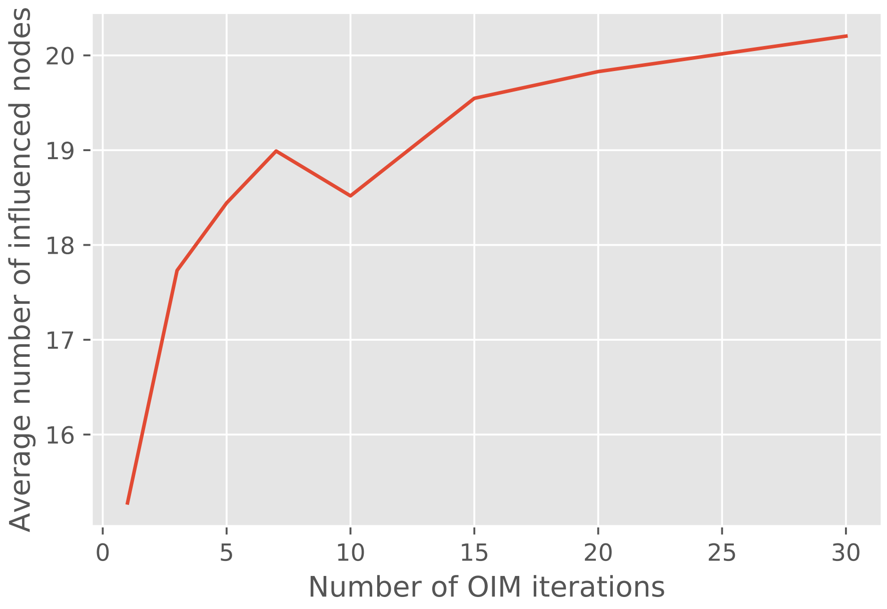


Figure 2.8: Mean number of nodes influenced by a seed set chosen with a different amount of OIM iterations

Even though a large number of Online IM iterations increases the influence generated by the resulting seed set, at the same time it slows down the processing time. That said, unlike the epsilon in the last section, this parameter influences the execution time in a linear way, which is not as bad for the algorithm's performance.

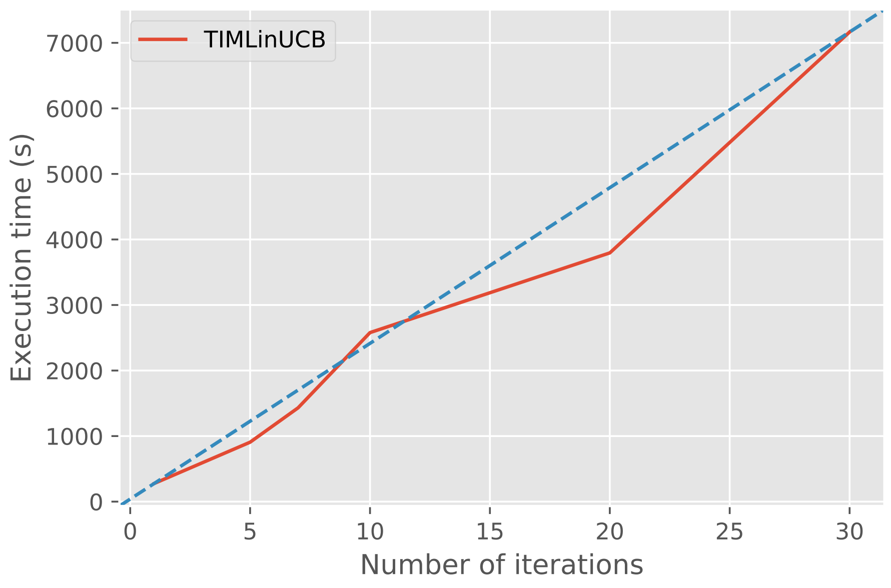


Figure 2.9: Time required to run TIMLinUCB vs the number of OIM iterations

2.3 Comparison with other algorithms

This paper compares TIMLinUCB to two other algorithms - RSB and an offline IM algorithm, TIM, being run on every time step. TIM is used as a benchmark Offline algorithm.

2.3.1 RSB

You can find the theoretical explanation of RSB in the section 1.1.5 above.

As for the practical implementation, the authors of RSB algorithm kindly agreed to provide us with the C++ code of their algorithm. The code was translated into Python for the purpose of comparison, since it did not support the datasets in this paper.

Performance and parallelization The RSB algorithm has a complexity of $O(kTn)$, where k is the amount of seed nodes, T is the number of time steps in the network, and n is the number of nodes in the network. It is lower than TIMLinUCB's complexity $\mathcal{O}(Tmd^2k(n+m)\log n/\epsilon^2)$, which means that at least theoretically RSB should outperform TIMLinUCB in the execution speed. One problematic point, however, is that due to its design it can not be parallelized. Since every time step in the algorithm 9 relies on the results of the previous time step, RSB must be evaluated sequentially which may impair the possibilities of the algorithm's application.

Grid search Since RSB takes two parameters, γ and C, we ran the same grid search as we did with TIMLinUCB to determine the optimal parameters.

As we can see from the figures 2.10 - 2.12, the results of running the algorithm with its best parameters on the Facebook dataset are barely higher than the actual number of seed nodes, mostly due to the sparseness of the networks at those time steps. While the heatmap generated by the search for 5 seeds does not show any obvious dependencies between γ (which balances out exploration and exploitation) and C, we can observe patterns forming in the searches for 20 and 100 seed nodes. There are ranges of γ for each C that produce equivalent results, with some values of C generating more influential seed nodes than others. This means that when looking for the best parameters, a user of the RSB algorithm would benefit from looking through a wide range of possible C's and a smaller range of

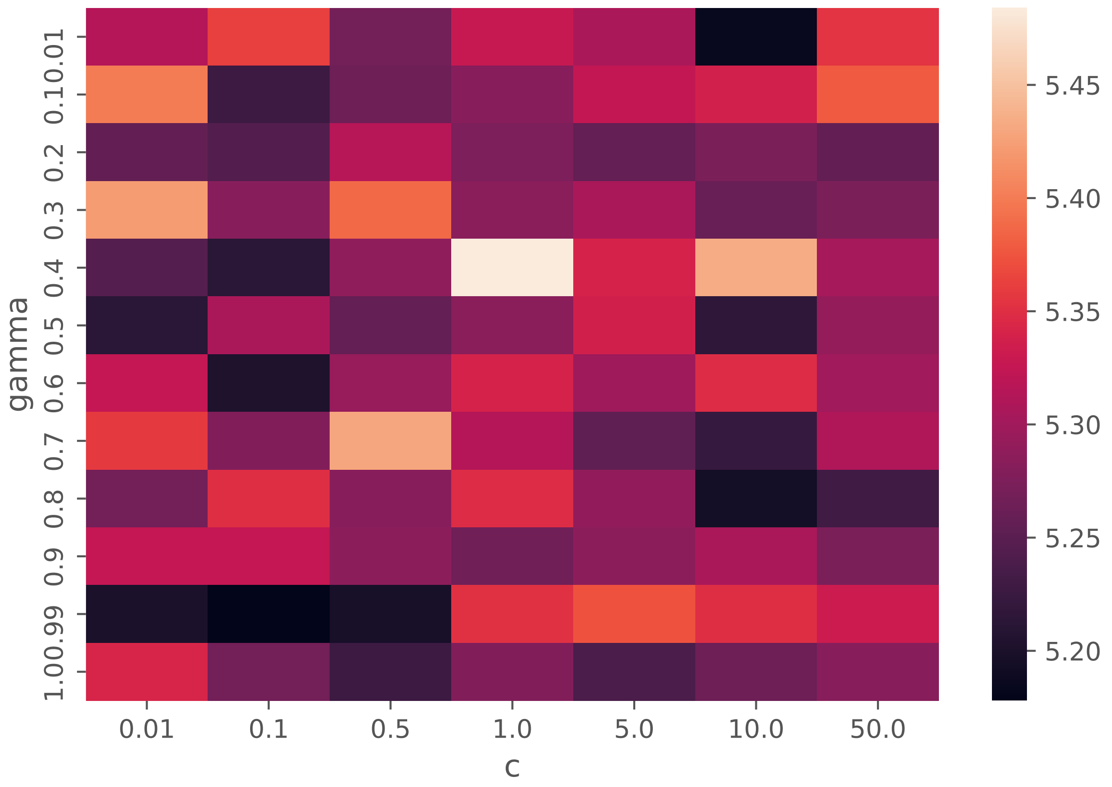


Figure 2.10: RSB - Grid search for the best parameters (Facebook dataset, 5 seed nodes)

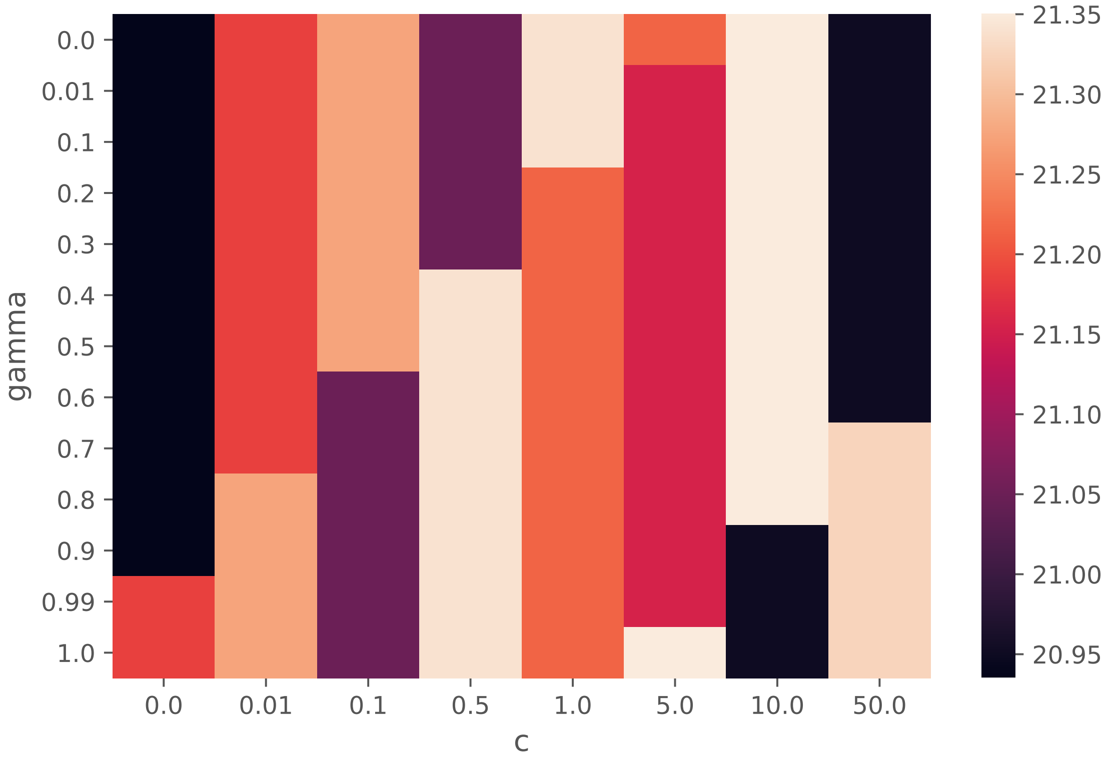


Figure 2.11: RSB - Grid search for the best parameters (Facebook dataset, 20 seed nodes)

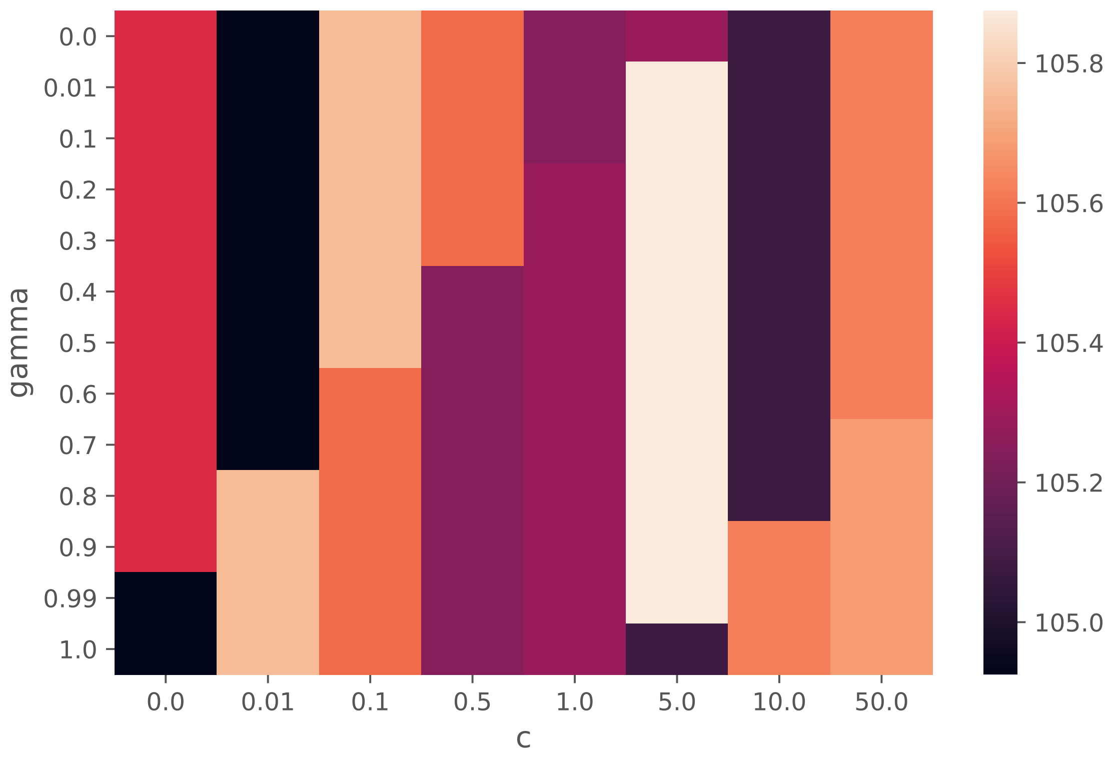


Figure 2.12: RSB - Grid search for the best parameters (Facebook dataset, 100 seed nodes)

possible γ's. Please note, that the results from the Digg dataset were put in the appendix A.2 as they offer similar insights to the Facebook one..

2.3.2 Comparison

Experiment setup

In order to compare TIMLinUCB and RSB, we ran both algorithms asking them to find the most influential 5, 20 and 100 nodes at every time step t. The experiments were done on the first 100 days of Facebook and Digg datasets (the reasoning for it is explained in the section 2.2.1).

The algorithms were run using the optimal parameters obtaining by using the grid search (shown in the sections 2.3.1 and 2.2.2). The information diffusion using the obtained seed sets was then simulated by using the IC model on every time step. The efficacy was measured in terms of the average number of nodes that were influenced after running the IC model 10 times.

Please note that despite the author of RSB not using the IC in their experiments, it claimed that RSB can be run under any model as long as we can model the information spread as a random variable, which is possible in IC.

Also, even though the RSB was modeled to minimize "regret", the regret is a difference between the number of influenced nodes generated by the source set obtained by running an Offline IM algorithm and one obtained by running RSB - and since we assume Offline IM to perform better than Online IM on average, the algorithm should also produce high influenced node count, since generating a large amount of influence is the main purpose of the Influence Maximization problem.

Furthermore, the TIM_T algorithm was run with a smaller ϵ than TIMLinUCB (0.2 in TIM vs 0.5 in TIMLinUCB). Since TIM's accuracy is essentially $1-1/number\_of\_edges-\epsilon$,

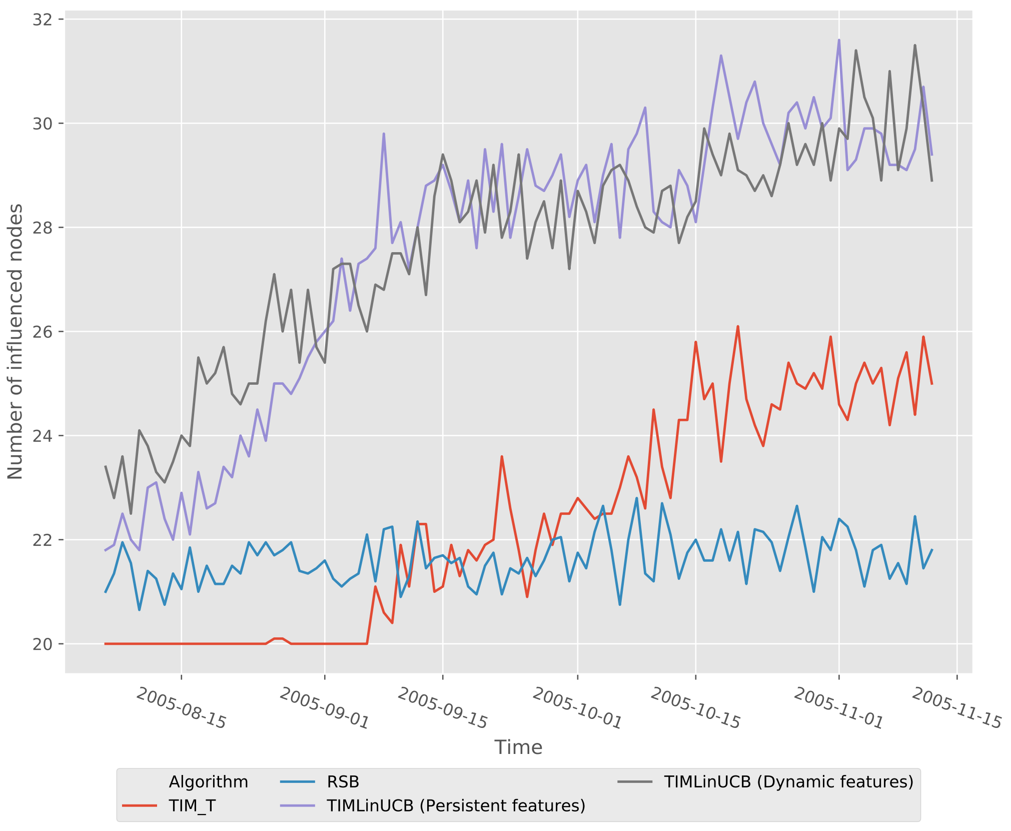


Figure 2.13: The performance of TIM with a large ϵ

running it with an epsilon that's too high defeats the purpose of using it as a control Offline IM algorithm that is more efficient than the Online ones. If we look at the figure 2.13 below that shows the results of running both TIM_T and TIMLinUCB on the first 100 days of the Digg dataset (the seed node set k was set to 20), we can see that TIMLinUCB can output good results even if the ϵ is rather large. This is likely due to running TIM multiple times in IMLinUCB, which helps us find better nodes even if the accuracy of the algorithm is not as good.

Results

As we can see from the figures 2.16 - 2.21 and table 2.2 ($|I|_t$ is the number of nodes influenced by the algorithm at the time t) below, TIMLinUCB consistently produces better results than RSB for every size of the seed set $k \in \{5, 20, 100\}$.

Persistent vs Dynamic parameters An interesting point to notice is that the version of TIMLinUCB with dynamic features generally outperforms the one with persistent ones - this indicates that the network is changing too much for the features from the last time step to be useful for the IMLinUCB algorithm.

Note, that for the seed size $k = 5$ and the Digg dataset, persisting the parameters throughout the network was actually more benefitial than not doing it - which means that a small number of well-connected nodes in there remained relatively stable while the rest of the network was changing rapidly (since it is no longer a benefitial choice when we are selecting $k = 20$ seeds).

The impact of seed set size As we can see from the table 2.2, the higher seed set size usually equals a larger amount of influence gained. But with the sizes of the seed set increasing, you would want for the number of influenced nodes to increase accordingly - after all, a seed set with $k = 1,000,000$ nodes generating an influence of $|I| = 1,000,100$ is pretty bad but if we just look at "the seed set S_k has influenced 100 people" it is neither good nor bad since we do not have a metric to compare it to.

In the figure 2.14 below, you can see how large is the set of newly activated nodes when compared to the original seed set. As we can see, the IM in Digg dataset started out by activating ≈ 1.7 for every original seed node but then the ratio dropped to ≈ 1.2. The Facebook dataset's ratio also dropped, but not as drastically - it went from being ≈ 3.5 to being ≈ 2.

Why did the ratio drop? There are multiple reasons for it. The main one is that the efficacy of new nodes in the seed node set goes down as we increase the size of the set. For example, if we were doing a marketing campaign then our efficacy would go down after we reached out to every major TV and Internet personality, simply because an average person does not have enough friends and audience to be able to influence as many people as a professional entertainer.

Another reason lies in the network structure - since Digg is not as dense as Facebook is, it is harder to find well-connected nodes that can activate multiple other nodes. To reuse the example from the above paragraph, the Digg dataset contains a smaller amount of popular *influencers* and a larger amount of normal people that are not generating as much influence.

Offline vs Online IM - TIM_T and TIMLinUCB While TIM_T was supposed to serve as the "golden standard" algorithm that TIMLinUCB and RSB were supposed to catch up with, it surprisingly underperformed despite knowing the true activation probabilities of the network and operating under a smaller ϵ which improves the accuracy of the algorithm. There are multiple potential reasons for this.

The first one is that while ϵ increases potential accuracy, TIM is still just approximating the best seed nodes so the underperformance might be because of a random chance.

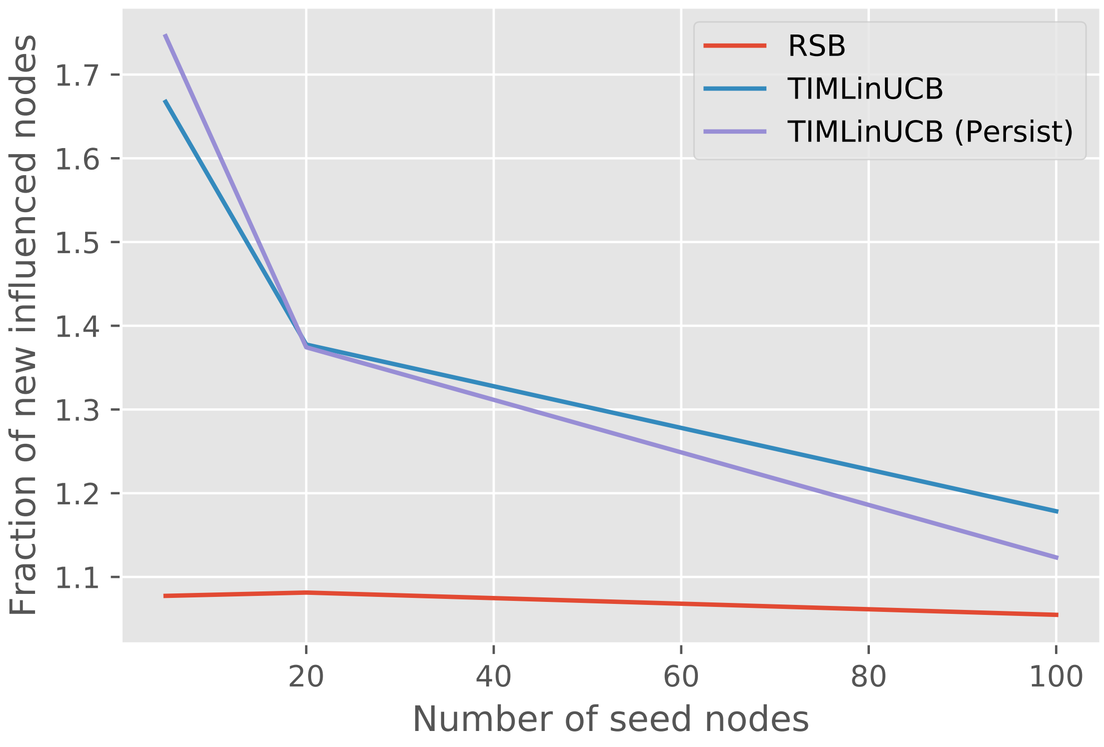
The second reason is that since the networks are somewhat sparse at first, it is harder for TIM to find the best nodes - this is supported by the evidence that TIM performs rather well in the last time step of the network, especially in t_{100} when it has to select 100 seed nodes.

The third reason would be that TIM_T is inherently at a disadvantage since it does not know about the network at the last time step and thus can not save its parameters like RSB or TIMLinUCB (Persist) - but as we saw in the paragraph "Persistent vs Dynamic parameters" above, the networks are changing so much that not saving the parameters from the last time step can be a very good choice.

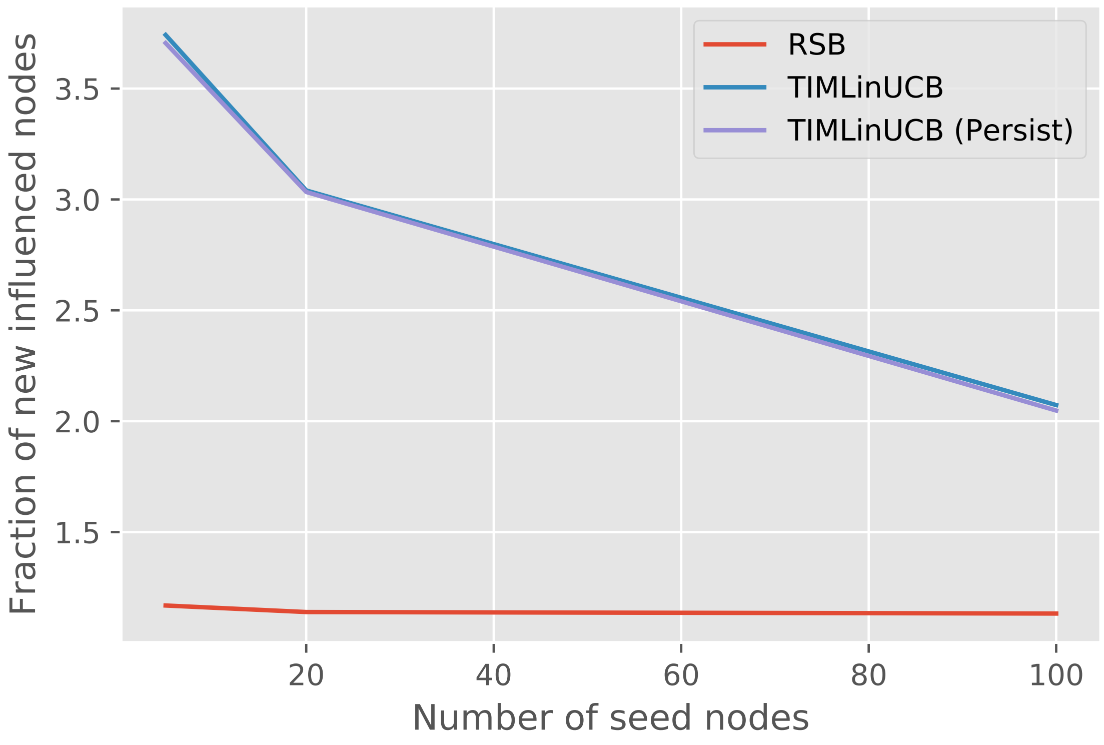
Lastly, TIMLinUCB might just be outperforming TIM_T due to the fact that it runs TIM multiple times and optimized its parameters. In order to compete with that, the benchmark TIM algorithm would have to set it epsilon much lower than it is now - around 0.1 or even 0.01.

Execution time Despite losing in efficacy, RSB does outperform TIMLinUCB (as demonstrated in table 2.3) in its execution speed. Even though using the parallel version of the TIMLinUCB speeds up the algorithm 4.1 times on average when compared to the normal one, RSB outperforms both, especially when applied to smaller seed sets.

The main reason for this difference is the design of RSB. As shown before in algorithm 9, RSB does not have to run a computationally expensive Offline IM algorithm multiple times at every time step in order to find the activation probabilities which increases the



(a) Digg dataset



(b) Facebook dataset

Figure 2.14: Comparison of the fraction of newly introduced nodes $\frac{|I|}{k}$ when compared to the number of seed nodes k

Table 2.2: Comparison between RSB and TIMLinUCB (Efficacy)

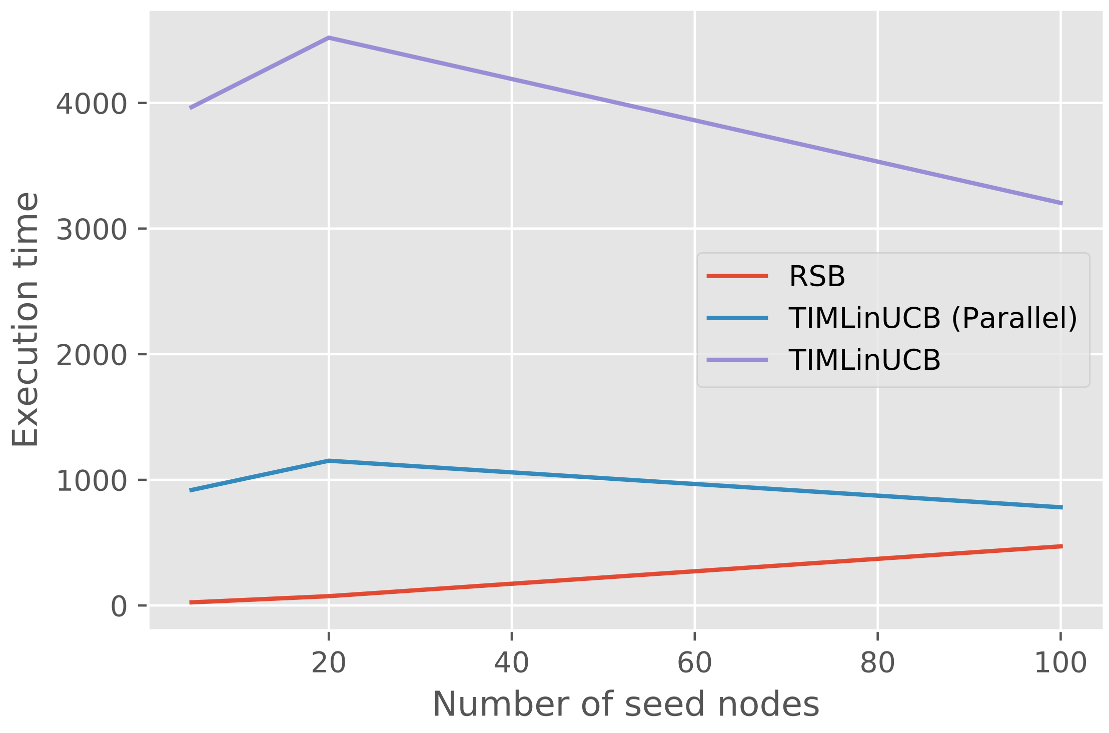
Algorithm	$\boldsymbol{|I|_{t_1}}$	$\boldsymbol{|I|_{t_{50}}}$	$\boldsymbol{|I|_{t_{100}}}$	$\boldsymbol{|I|_{mean}}$
Digg dataset - 5 seed nodes				
RSB	5.45 ± 0.67	6.0 ± 0.77	5.10 ± 0.30	5.39
TIMLinUCB	6.70 ± 1.35	**8.4 ± 1.91**	9.50 ± 2.69	8.34
TIMLinUCB (Persist)	**7.30 ± 1.00**	8.4 ± 1.20	**9.90 ± 1.81**	**8.73**
TIM_T (Offline)	5.1 ± 0.3	8.2 ± 1.47	9.0 ± 1.90	7.87
Digg dataset - 20 seed nodes				
RSB	22.15 ± 1.49	21.5 ± 0.59	21.50 ± 1.32	21.63
TIMLinUCB	**23.00 ± 1.67**	**28.3 ± 3.00**	28.90 ± 3.27	**27.55**
TIMLinUCB (Persist)	22.70 ± 1.35	27.8 ± 1.99	**29.40 ± 2.50**	27.49
TIM (Offline)	20.0 ± 0.0	26.7 ± 2.00	29.1 ± 2.34	26.02
Digg dataset - 100 seed nodes				
RSB	103.25 ± 1.55	106.3 ± 1.35	107.25 ± 2.26	105.48
TIMLinUCB	105.80 ± 2.04	**118.4 ± 4.98**	**126.30 ± 3.29**	**117.85**
TIMLinUCB (Persist)	**106.60 ± 1.74**	113.5 ± 2.84	121.50 ± 2.42	112.33
TIM_T (Offline)	100.0 ± 0.0	108.4 ± 3.23	115.0 ± 3.16	108.34
Facebook dataset - 5 seed nodes				
RSB	**5.3 ± 0.46**	5.50 ± 0.67	5.5 ± 0.50	5.84
TIMLinUCB	5.0 ± 0.00	17.70 ± 4.29	**34.2 ± 6.43**	**18.71**
TIMLinUCB (Persist)	5.0 ± 0.00	**19.00 ± 2.79**	33.5 ± 5.84	18.53
TIM_T (Offline)	5.0 ± 0.0	15.0 ± 2.51	30.0 ± 4.66	17.50
Facebook dataset - 20 seed nodes				
RSB	**20.2 ± 0.40**	24.40 ± 2.65	25.1 ± 2.74	22.79
TIMLinUCB	20.0 ± 0.00	**61.80 ± 8.94**	96.1 ± 7.57	**59.61**
TIMLinUCB (Persist)	20.0 ± 0.00	59.60 ± 8.30	**102.3 ± 16.01**	59.08
TIM_T (Offline)	20.0 ± 0.0	55.2 ± 1.47	96.4 ± 10.29	59.36
Facebook dataset - 100 seed nodes				
RSB	**101.4 ± 0.92**	113.30 ± 6.23	121.3 ± 6.72	113.27
TIMLinUCB	100.0 ± 0.00	212.10 ± 10.74	**297.5 ± 19.36**	**207.28**
TIMLinUCB (Persist)	100.0 ± 0.00	**215.60 ± 12.17**	292.2 ± 29.41	204.81
TIM_T (Offline)	100.0 ± 0.0	221.5 ± 12.17	317.0 ± 16.46	214.54

Dataset	Number of seed nodes	Algorithm name	Computational time (s)
Digg	5	RSB	**24.80**
		TIMLinUCB	918.99
		TIMLinUCB (Persist)	3966.10
	20	RSB	**73.95**
		TIMLinUCB	1152.06
		TIMLinUCB (Persist)	4519.14
	100	RSB	**470.50**
		TIMLinUCB	781.52
		TIMLinUCB (Persist)	3203.93
Facebook	5	RSB	**76.18**
		TIMLinUCB	919.36
		TIMLinUCB (Persist)	3299.37
	20	RSB	**246.04**
		TIMLinUCB	1813.79
		TIMLinUCB (Persist)	8257.83
	100	RSB	**405.06**
		TIMLinUCB	1155.43
		TIMLinUCB (Persist)	4061.77

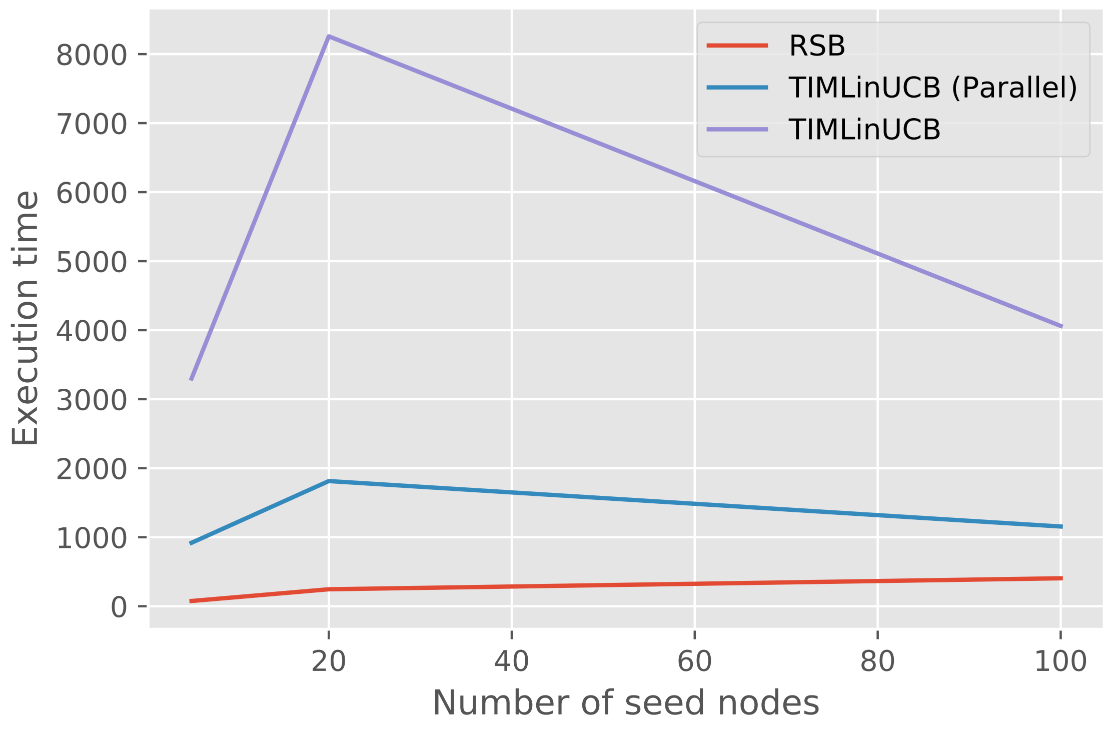
Table 2.3: Comparison between RSB and TIMLinUCB (Time spent)

performance. At the same time, TIMLinUCB having all of those extra steps allows it to produce better results than RSB.

An interesting observation that one can make from the figure 2.15 is that the execution time of RSB grows with the increase of the seed set size, while the time of TIMLinUCB remains relatively constant (if we exclude the abnormal jump at 20 seeds in the Facebook dataset). This means that TIMLinUCB might be more efficient at handling large (≥ 100) seed sets, and might be more useful during large marketing campaigns, especially when this is coupled with the fact that TIMLinUCB's seeds generate more influence overall.



(a) Digg dataset



(b) Facebook dataset

Figure 2.15: Execution time comparison

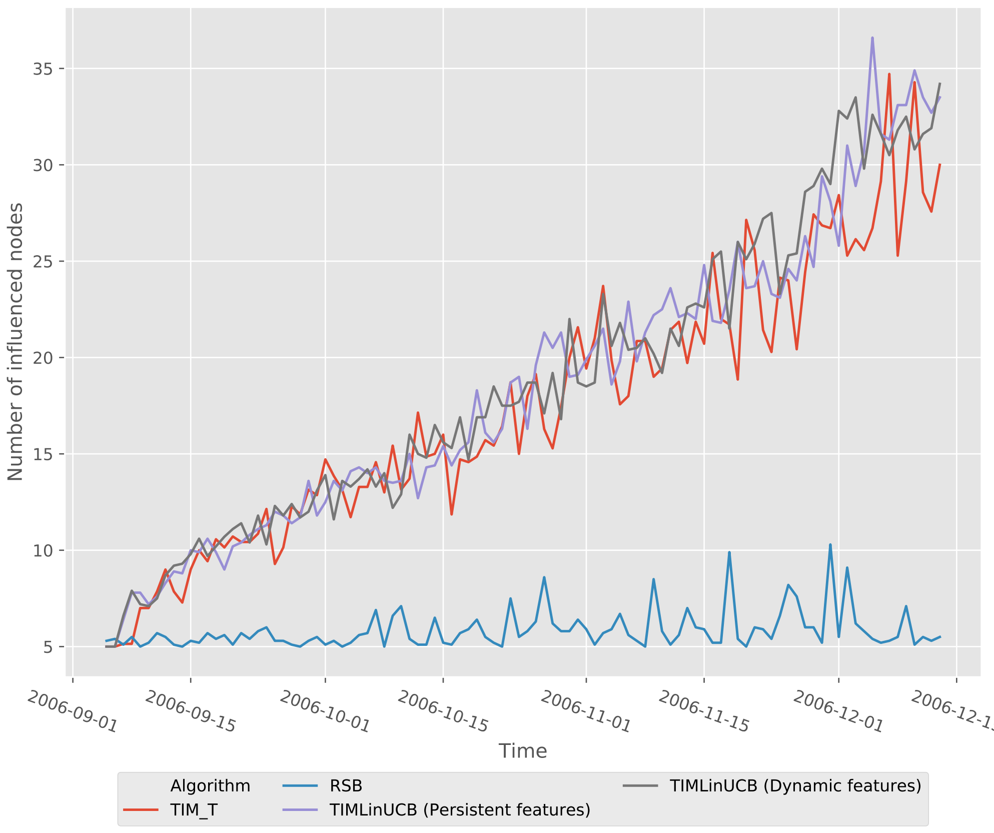


Figure 2.16: Comparison between TIMLinUCB and RSB (Facebook dataset, 5 seed nodes)

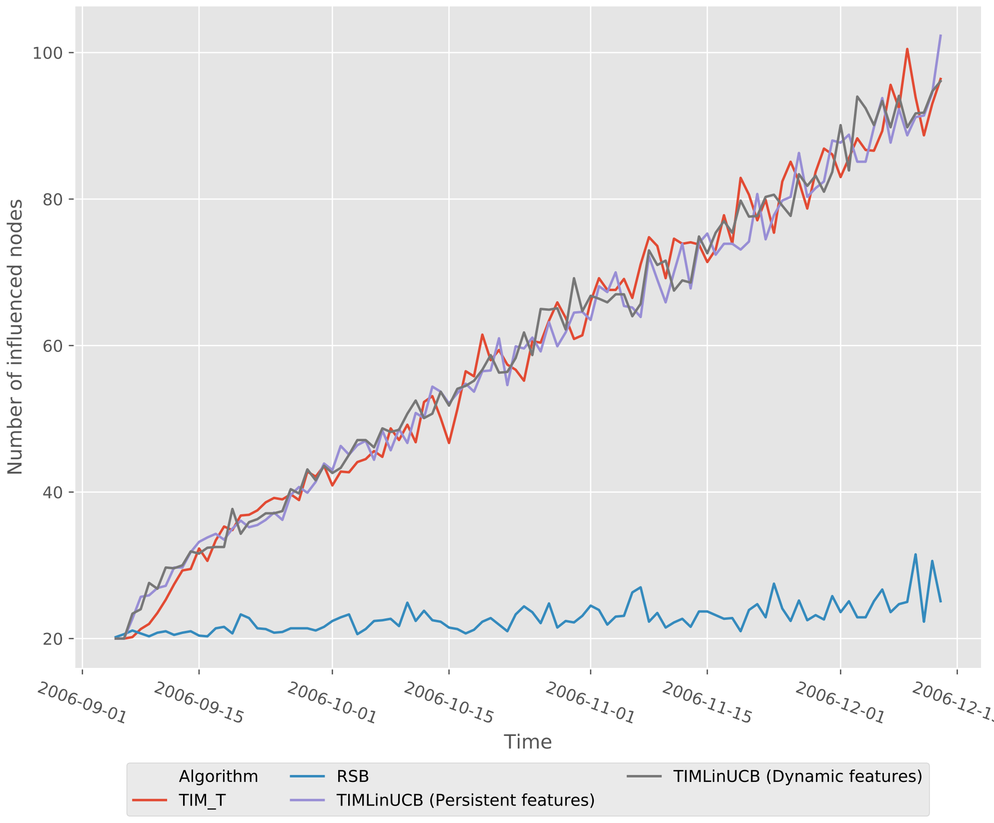


Figure 2.17: Comparison between TIMLinUCB and RSB (Facebook dataset, 20 seed nodes)

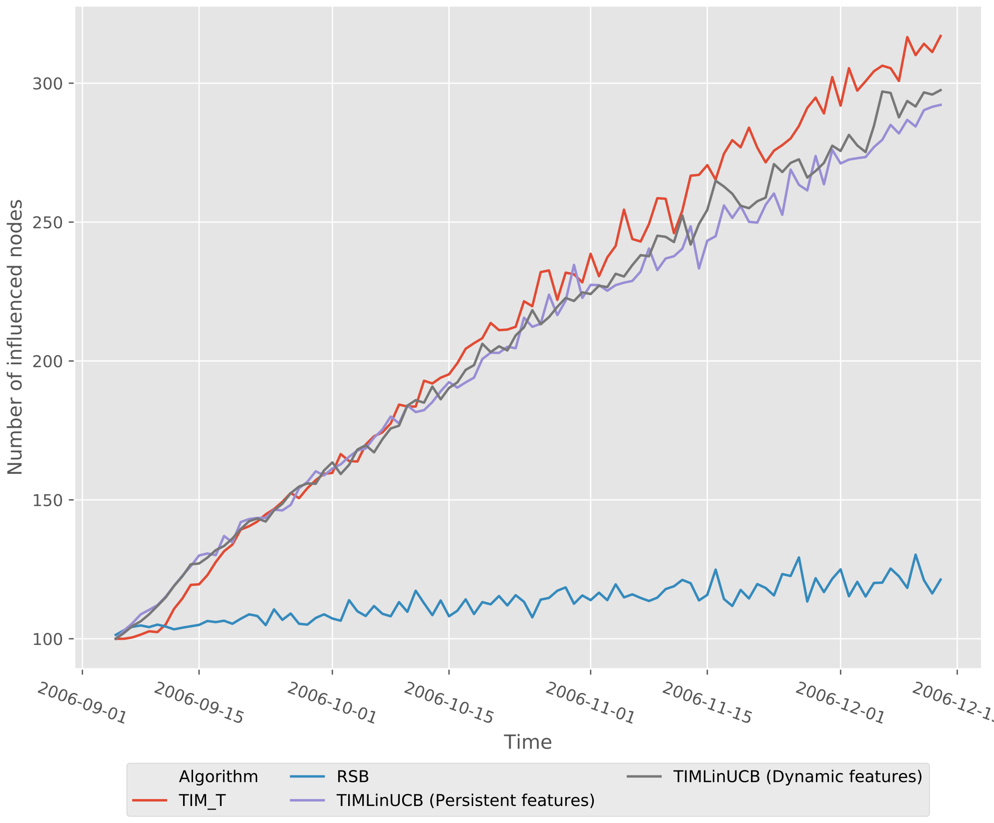


Figure 2.18: Comparison between TIMLinUCB and RSB (Facebook dataset, 100 seed nodes)

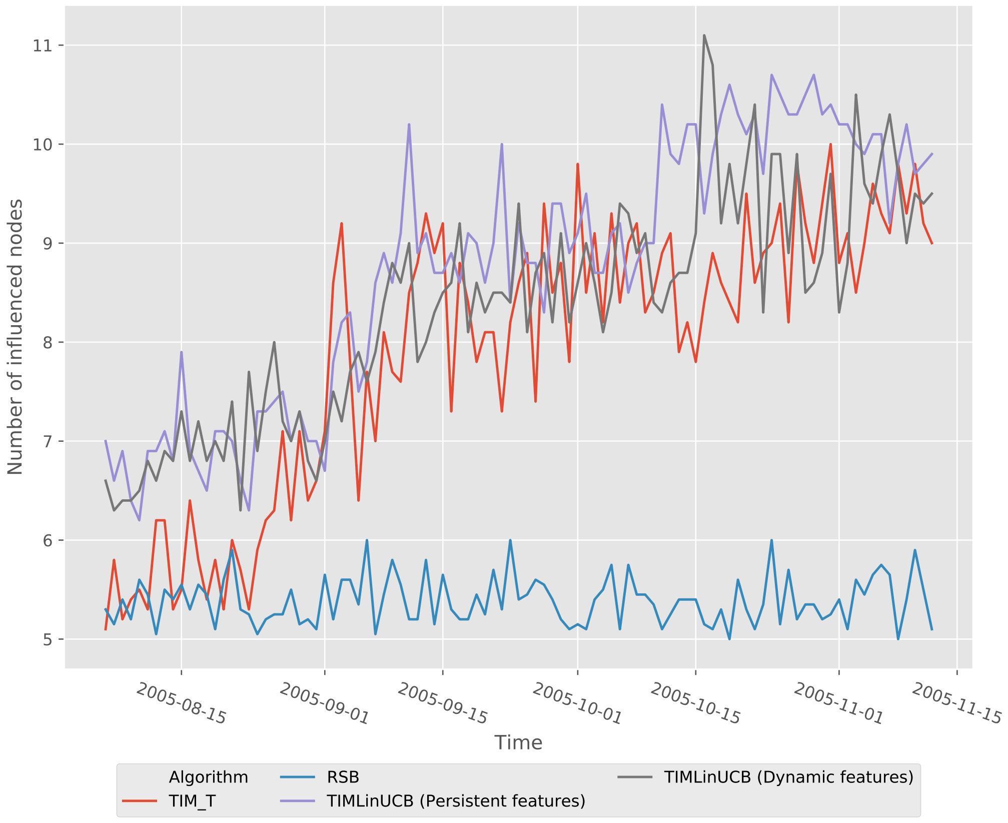


Figure 2.19: Comparison between TIMLinUCB and RSB (Digg dataset, 5 seed nodes)

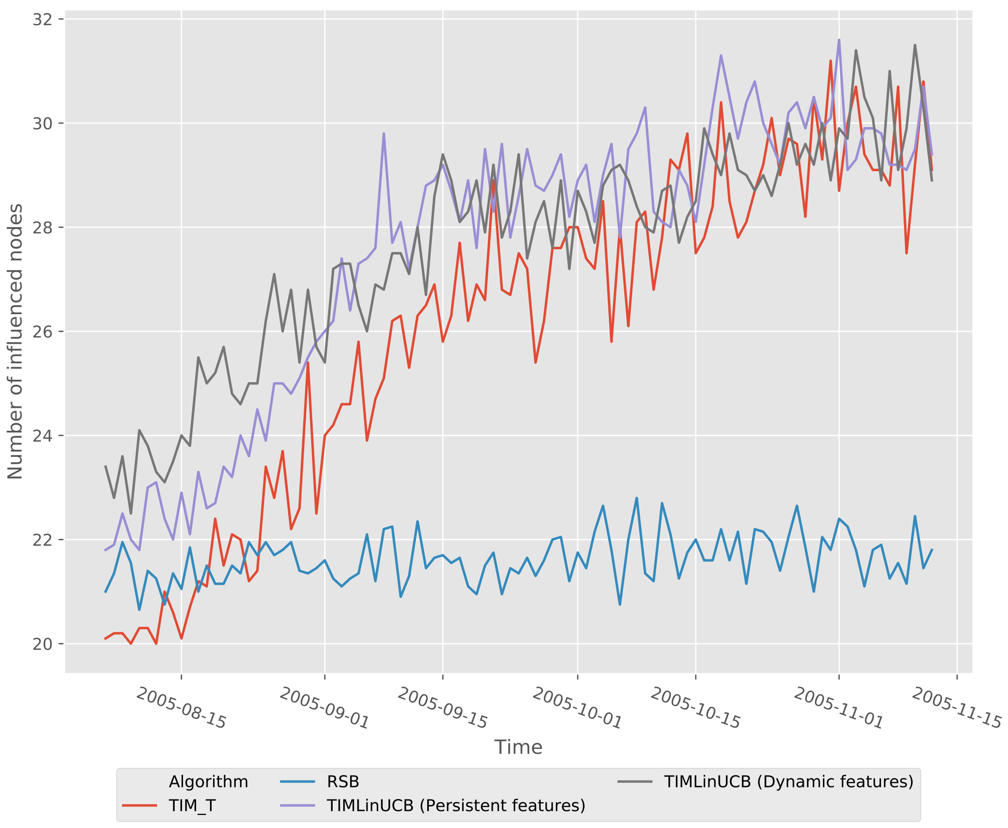


Figure 2.20: Comparison between TIMLinUCB and RSB (Digg dataset, 20 seed nodes)

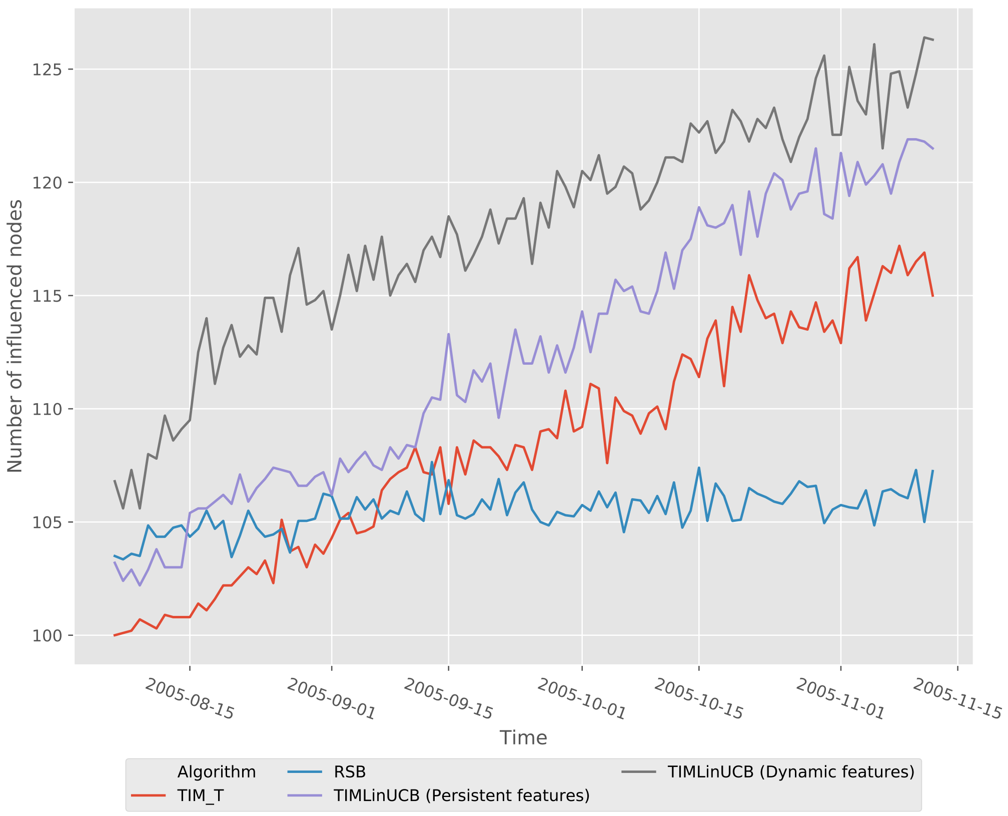


Figure 2.21: Comparison between TIMLinUCB and RSB (Digg dataset, 100 seed nodes)

Chapter 3

Conclusion

This paper presented a brief overview of the research landscape of the Influence Maximization problem, including an overview of currently used diffusion models and explanations of strategies and algorithms for both Offline and Online Influence Maximization. We have found that due to its newness and complexity of the Online Influence Maximization problem, there is a lack of algorithms that can solve the Online Influence Maximization problem in Temporal networks.

We argued that this problem is worth solving, since in real-world applications of the Influence Maximization problem, especially social marketing, the activation probabilities are often not known and social networks themselves change over time. Therefore, the main goal of this paper was to create an algorithm that would contribute to the research of Online Influence Maximization in Temporal Networks. In order to achieve this, we took a state-of-the-art Online IM algorithm, IMLinUCB and made it applicable to Temporal networks.

The resulting algorithm, TIMLinUCB, was shown to be more effective at solving the Online Influence Maximization problem on Temporal Networks than its competitor RSB, albeit it takes more time to find the solution. To combat the performance issue, this thesis includes multiple ways of speeding up the algorithm by parallelizing it. We also showed how one could go about finding the best parameters to feed into the algorithm, and analyzed the impact of every parameter on the algorithm's performance and efficacy.

Future work

In terms of future work, one way to improve the performance of TIMLinUCB would be to rewrite the algorithm in a more low-level language like C or C++. While this would not improve the complexity of the algorithm, this change could influence the empirical runtime of the algorithm.

We could also improve the performance of the algorithm would be to sacrifice some of the results obtained by IMLinUCB in order to approximate its output without going through every iteration.

Another way to improve the current algorithm is to make it compatible with more diffusion models, namely the Triggering one introduced in section 1.1.2.

While we can optimize the current algorithm by either rewriting the program in a more efficient language or approximating the results of IMLinUCB, one could also find an Online IM algorithm that is better and faster than IMLinUCB. With Online Influence Maximization still being actively researched, a release of a better algorithm is rather likely, and it would improve both performance and efficacy of the algorithm.

Yet another way to further the research would be to develop an algorithm that could perform Online IM on Temporal networks in a distributed way - the datasets are getting larger and larger nowadays, and not being able to fit them into a computer's memory might be a problem.

Appendix A

Additional charts

A.1 Grid search for the best parameters - TIMLinUCB

This section of the appendix contains the grid search results for the best parameters of TIMLinUCB used for finding the most influential 20 and 100 seed nodes in the network ($\epsilon = 0.1$ was excluded from the comparison due to it being inefficient to calculate, as explained in the section 2.2.2 of chapter 2, “Finding the best parameters”). The insights that can be obtained from these charts are the same as described in the section 2.2.2 of chapter 2.

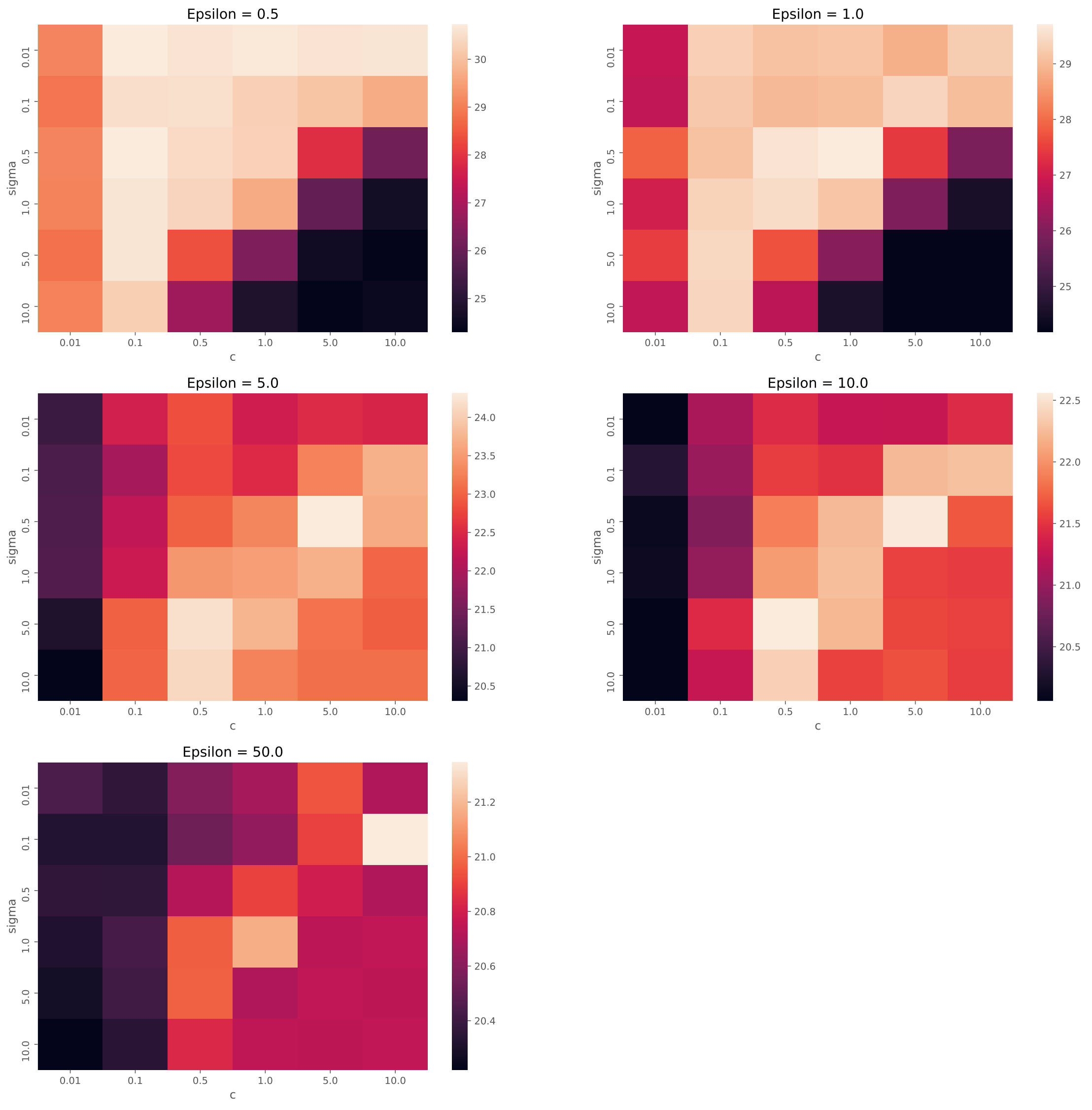


Figure A.1: TIMLinUCB - Grid search for the best parameters (Facebook dataset, 20 seed nodes)

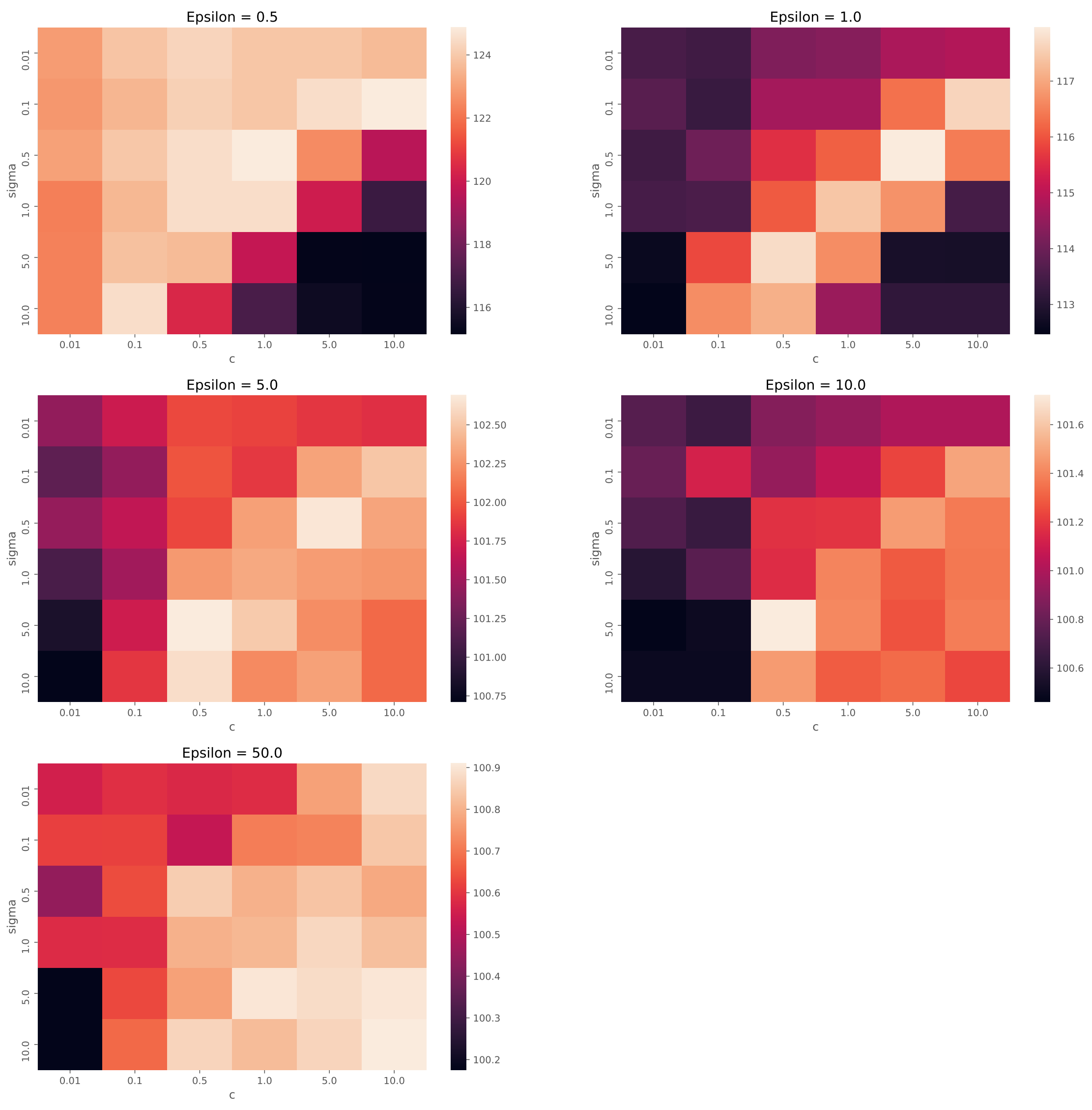


Figure A.2: TIMLinUCB - Grid search for the best parameters (Facebook dataset, 100 seed nodes)

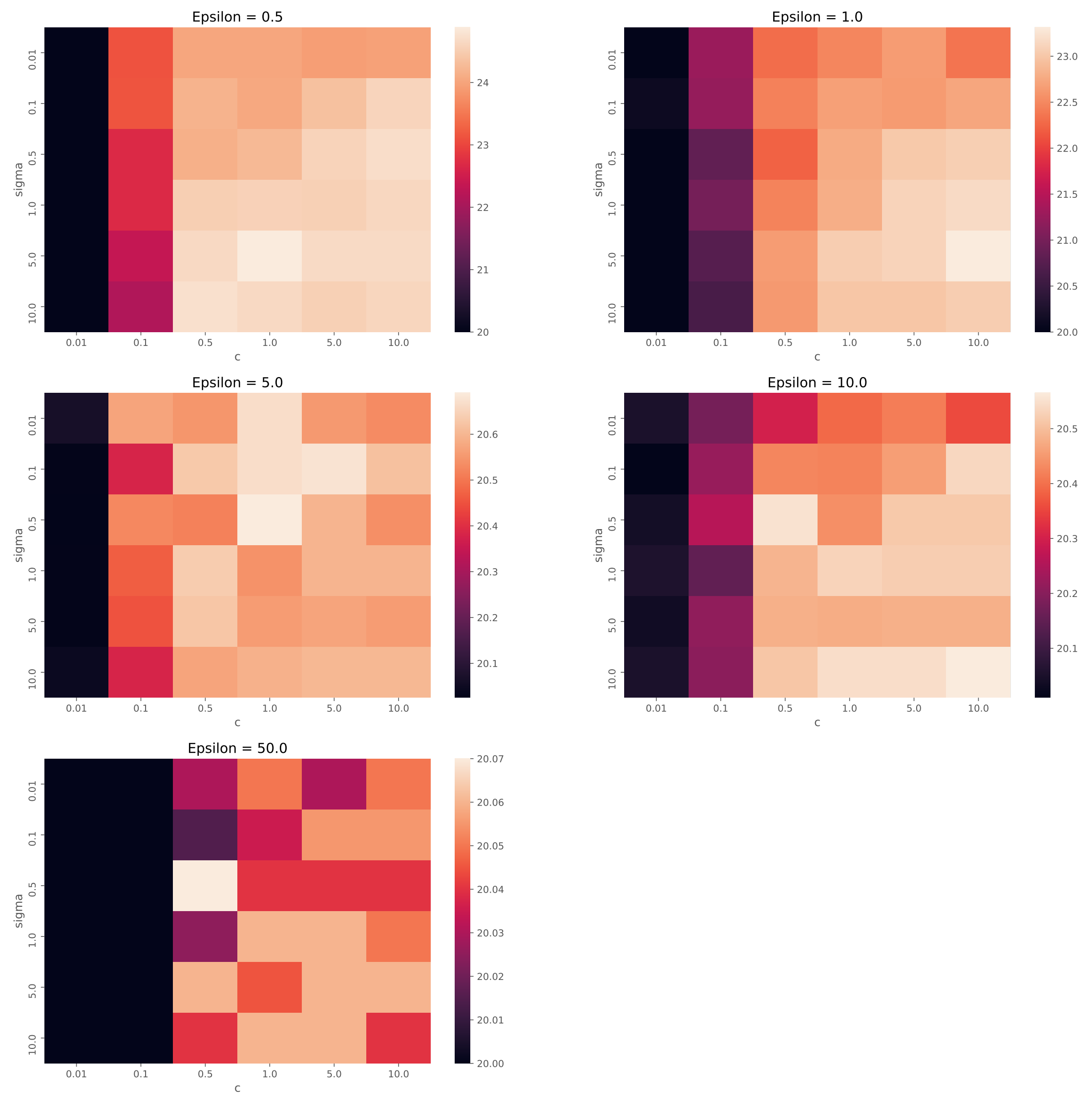


Figure A.3: TIMLinUCB - Grid search for the best parameters (Digg dataset, 20 seed nodes)

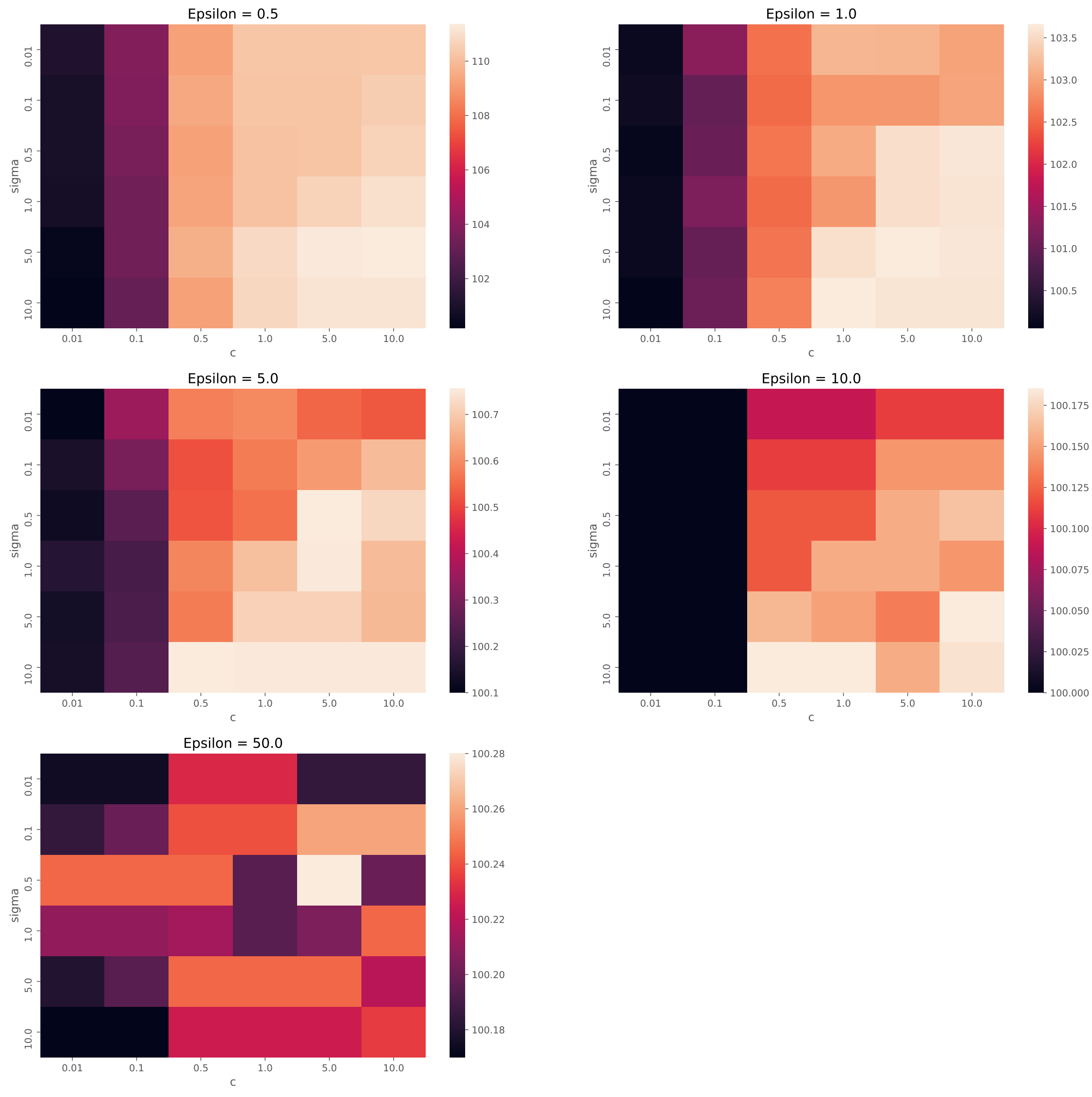


Figure A.4: TIMLinUCB - Grid search for the best parameters (Digg dataset, 100 seed nodes)

A.2 Grid search for the best parameters - RSB

This section of the appendix contains the grid search results for the best parameters of the RSB algorithm, executed on the Digg dataset. The insights that can be obtained from the charts are the same as the ones described in the section 2.3.1 of chapter 2 based on the Facebook dataset.

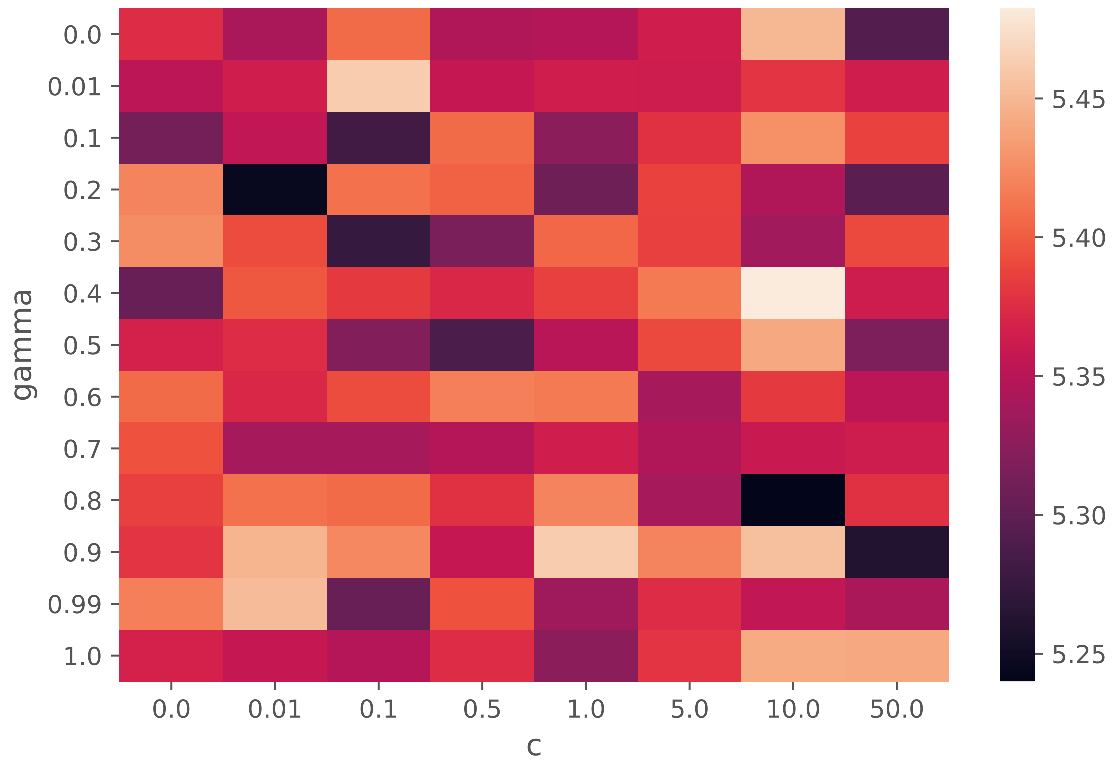


Figure A.5: RSB - Grid search for the best parameters (Digg dataset, 5 seed nodes)

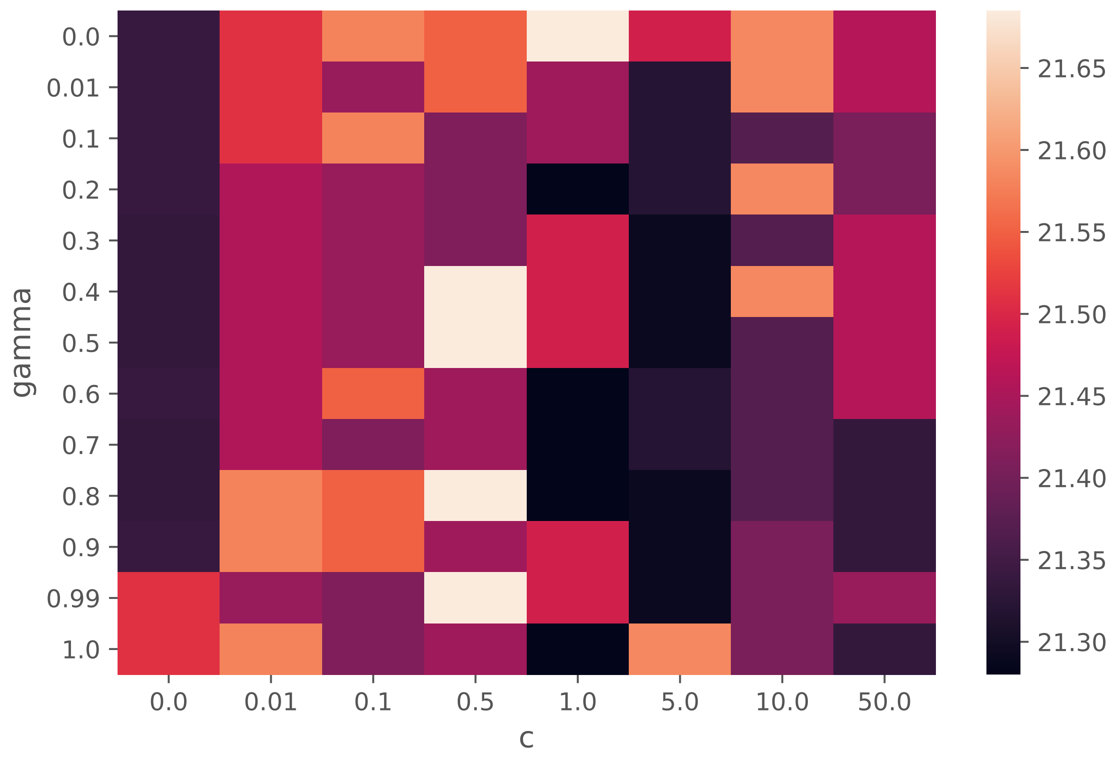


Figure A.6: RSB - Grid search for the best parameters (Digg dataset, 20 seed nodes)

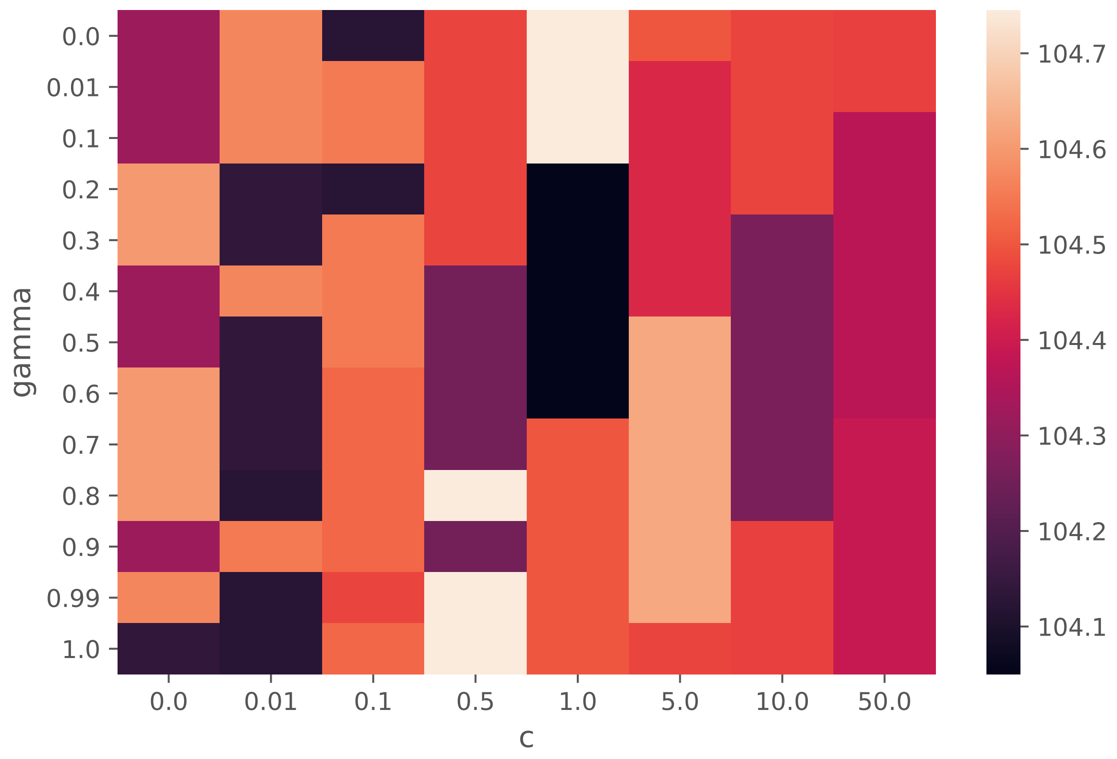


Figure A.7: RSB - Grid search for the best parameters (Digg dataset, 100 seed nodes)

A.3 TIMLinUCB and RSB comparison

The comparison charts in papers introducing RSB [3], IMLinUCB [39] and TIM [32] all use either one value or a mean of the variable when plotting it. When calculating the influence, however, there is some variance present even if we are simulating the diffusion spread using the same seed nodes - after all, even if an edge's activation probability is high it does not guarantee that the edge will activate. In order to show the variance present in the number of influenced nodes obtained by using the seed sets from RSB and IMLinUCB, we add the charts displaying the variance to this appendix.

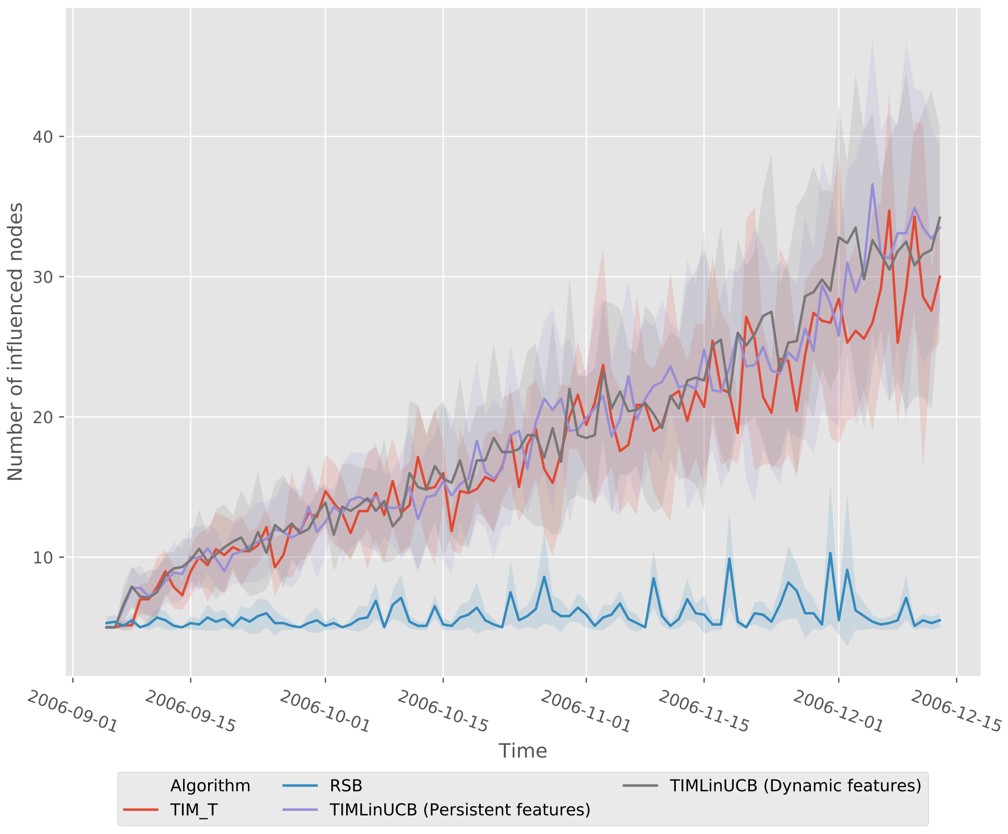


Figure A.8: Comparison between TIMLinUCB and RSB (Facebook dataset, 5 seed nodes)

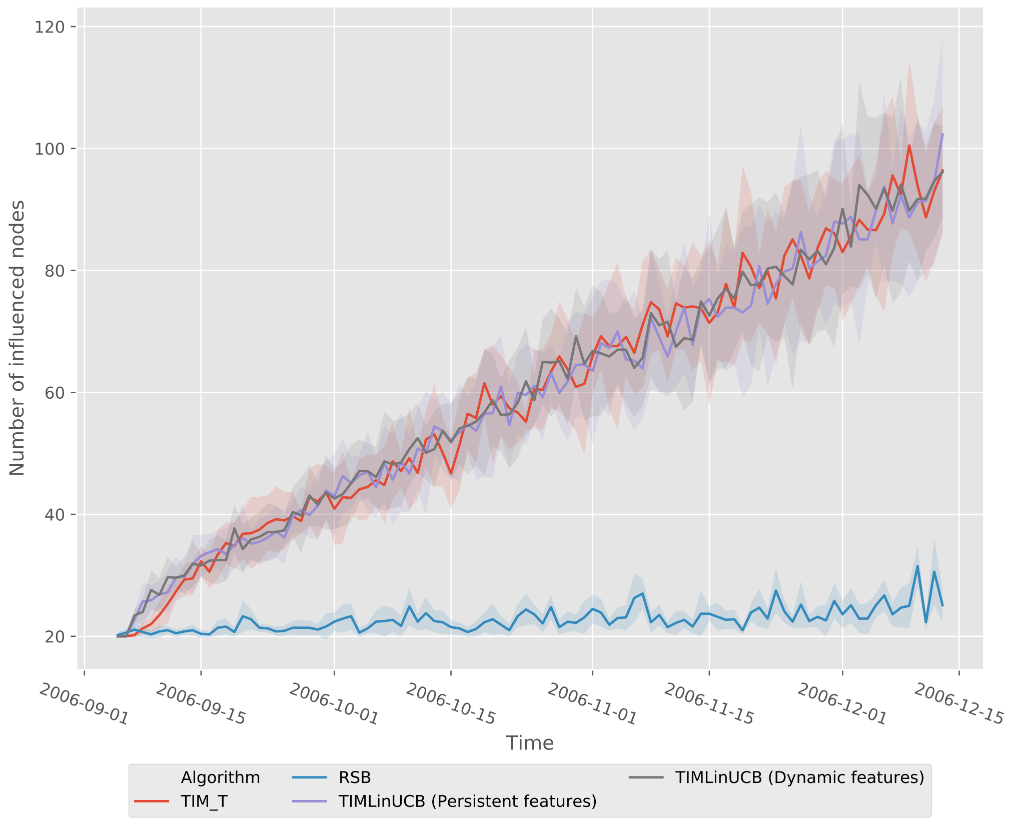


Figure A.9: Comparison between TIMLinUCB and RSB (Facebook dataset, 20 seed nodes)

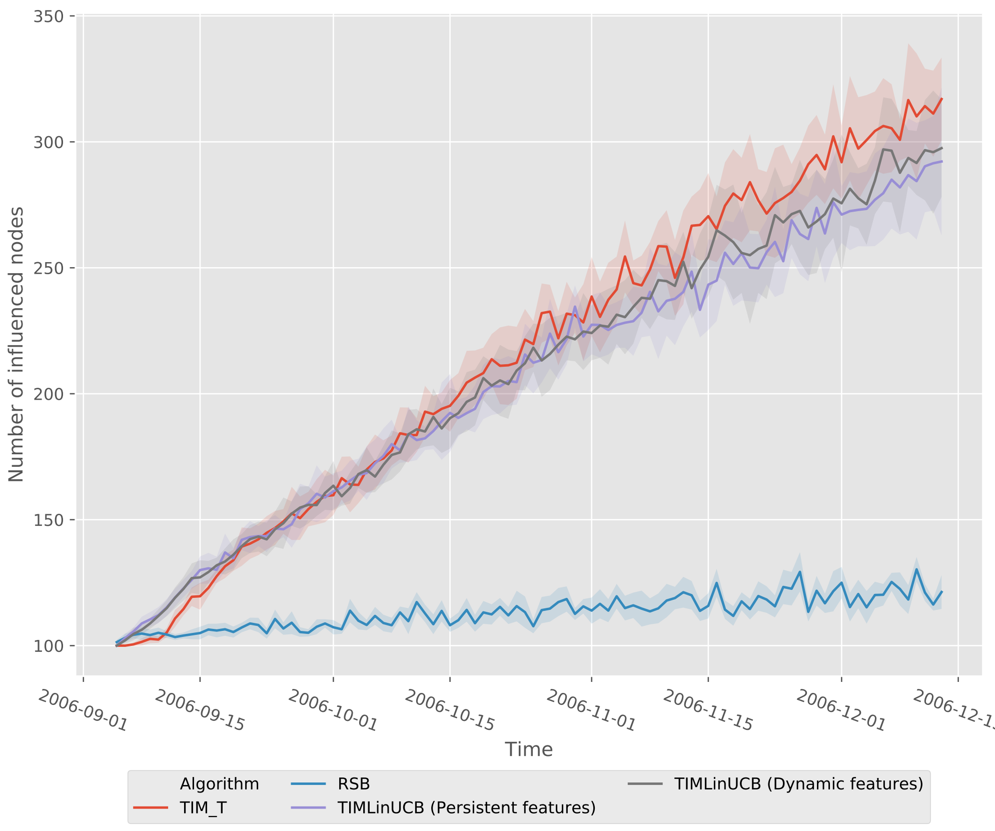


Figure A.10: Comparison between TIMLinUCB and RSB (Facebook dataset, 100 seed nodes)

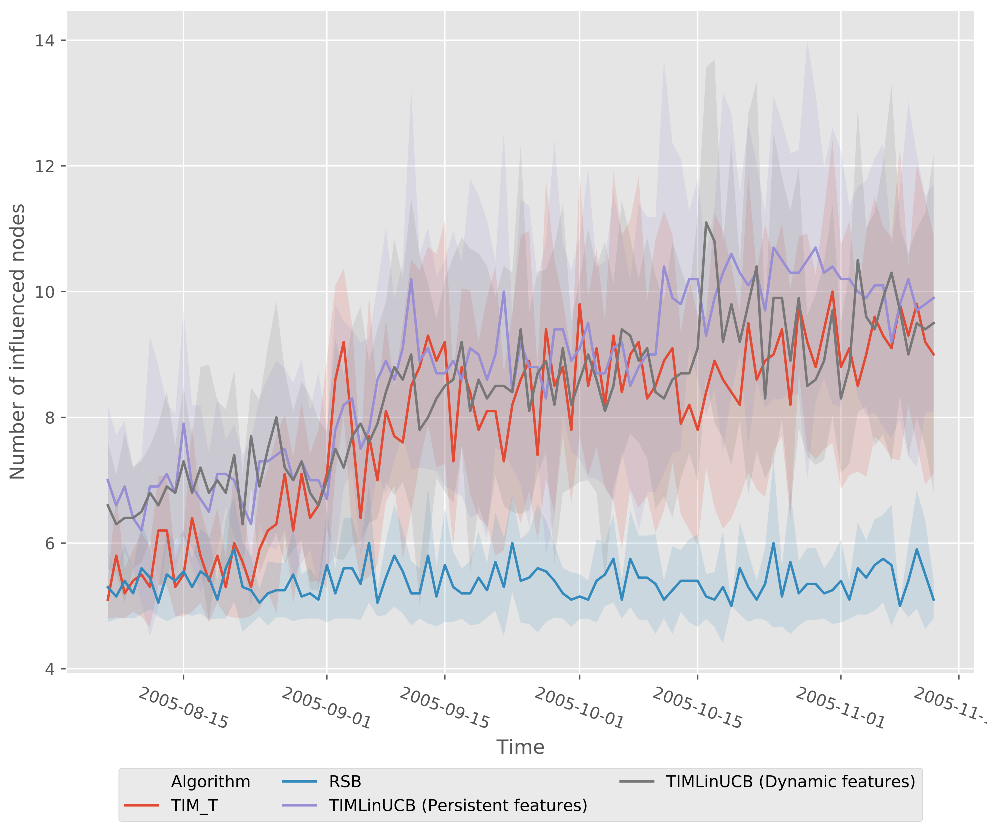


Figure A.11: Comparison between TIMLinUCB and RSB (Digg dataset, 5 seed nodes)

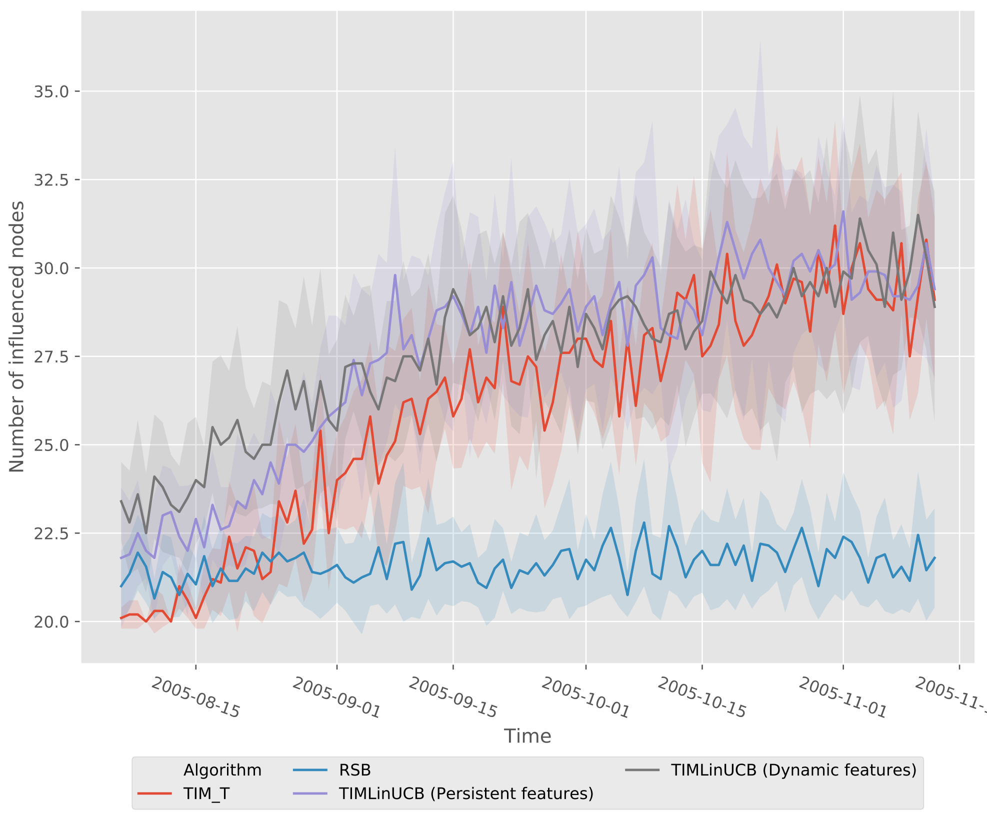


Figure A.12: Comparison between TIMLinUCB and RSB (Digg dataset, 20 seed nodes)

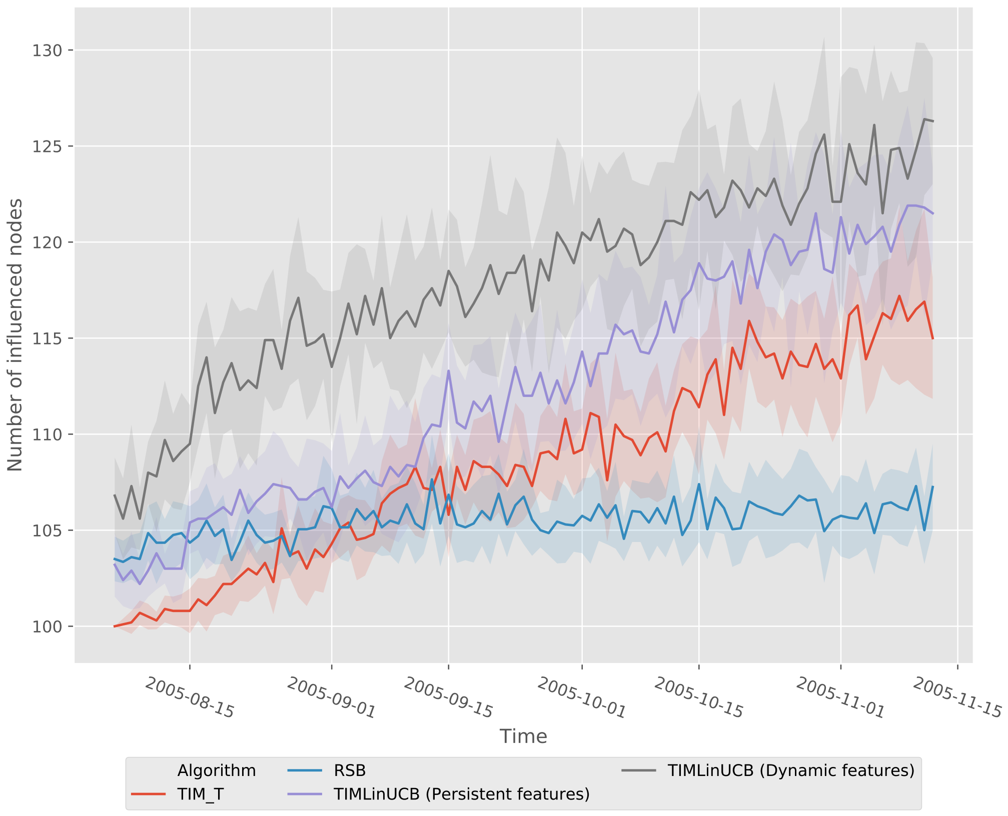


Figure A.13: Comparison between TIMLinUCB and RSB (Digg dataset, 100 seed nodes)

Acknowledgment

I would like to thank everyone that supported me on my journey of writing this thesis and discovering Tokyo Tech.
To my professor, Tsuyoshi Murata, that gave me a chance and supported me.
The members of my laboratory that were always kind and helpful.
The Hult Prize Tokyo Tech team that taught me so much.
The Tokyo Tech International Student Association, that showed me a different perspective on international people living in Japan.
A Capella circle Ajiwai, that showed me how Japanese people view the world.
Kind and helpful people at HUB-ICS.
My close friends and family that were always there. Tarek, Mitski, Karthik, Genki, Fynn, Ridwan, Chihiro - you guys changed my life. Mom, Dad - I love you.
To mentors I found.
To Aish, that I will not forget.
And you, if you have read this thesis!
Thank you!

References

[1] Charu C Aggarwal, Shuyang Lin, and Philip S Yu. On influential node discovery in dynamic social networks. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 636–647. SIAM, 2012.

[2] Suman Banerjee, Mamata Jenamani, and Dilip Kumar Pratihar. A survey on influence maximization in a social network. *Knowledge and Information Systems*, pages 1–39, 2020.

[3] Yixin Bao, Xiaoke Wang, Zhi Wang, Chuan Wu, and Francis CM Lau. Online influence maximization in non-stationary social networks. In *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*, pages 1–6. IEEE, 2016.

[4] Nicola Barbieri, Francesco Bonchi, and Giuseppe Manco. Topic-aware social influence propagation models. *Knowledge and information systems*, 37(3):555–584, 2013.

[5] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. Maximizing social influence in nearly optimal time. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 946–957. SIAM, 2014.

[6] Djallel Bouneffouf and Irina Rish. A survey on practical applications of multi-armed and contextual bandits. *arXiv preprint arXiv:1904.10040*, 2019.

[7] Nathalie TH Gayraud, Evaggelia Pitoura, and Panayiotis Tsaparas. Diffusion maximization in evolving social networks. In *Proceedings of the 2015 acm on conference on online social networks*, pages 125–135, 2015.

[8] Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. Learning influence probabilities in social networks. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 241–250, 2010.

[9] Mark Granovetter. Threshold models of collective behavior. *American journal of sociology*, 83(6):1420–1443, 1978.

[10] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[11] Tad Hogg and Kristina Lerman. Social dynamics of digg. *EPJ Data Science*, 1(1):5, 2012.

[12] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[13] Qingye Jiang, Guojie Song, Cong Gao, Yu Wang, Wenjun Si, and Kunqing Xie. Simulated annealing based influence maximization in social networks. In *Twenty-fifth AAAI conference on artificial intelligence*, 2011.

[14] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, 2003.

[15] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, 2003.

[16] Masahiro Kimura and Kazumi Saito. Tractable models for information diffusion in social networks. In *European conference on principles of data mining and knowledge discovery*, pages 259–271. Springer, 2006.

[17] Hautahi Kingi. *Reverse Influence Sampling in Python*, 2018 (accessed July 20, 2020). `https://web.archive.org/web/20200720050209/https://hautahi.com/im_ris`.

[18] Oleksii Kyrylchuk. Timlinucb: First release, July 2020 (Accessed July 14, 2020). DOI: *10.5281/zenodo.3945230*. Available at `https://doi.org/10.5281/zenodo.3945230`.

[19] Siyu Lei, Silviu Maniu, Luyi Mo, Reynold Cheng, and Pierre Senellart. Online influence maximization. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 645–654, 2015.

[20] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1852–1872, 2018.

[21] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge university press, 1995.

[22] Tsuyoshi Murata and Hokuto Koga. Extended methods for influence maximization in dynamic networks. *Computational social networks*, 5(1):8, 2018.

[23] Naoto Ohsaka, Takuya Akiba, Yuichi Yoshida, and Ken-ichi Kawarabayashi. Dynamic influence analysis in evolving networks. *Proceedings of the VLDB Endowment*, 9(12):1077–1088, 2016.

[24] Naoto Ohsaka, Takuya Akiba, Yuichi Yoshida, and Ken-ichi Kawarabayashi. Dynamic influence analysis in evolving networks. *Proceedings of the VLDB Endowment*, 9(12):1077–1088, 2016.

[25] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[26] Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[27] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.

[28] Paulo Shakarian, Abhinav Bhatnagar, Ashkan Aleali, Elham Shaabani, and Ruocheng Guo. The independent cascade and linear threshold models. In *Diffusion in Social Networks*, pages 35–48. Springer, 2015.

[29] Guojie Song, Yuanhao Li, Xiaodong Chen, Xinran He, and Jie Tang. Influential node tracking on dynamic social network: An interchange greedy approach. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):359–372, 2016.

[30] Jimeng Sun and Jie Tang. A survey of models and algorithms for social influence analysis. In *Social network data analytics*, pages 177–214. Springer, 2011.

[31] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[32] Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. *CoRR*, abs/1404.0900, 2014.

[33] Youze Tang, Xiaokui Xiao, and Yanchen Shi. Tim plus, July 2017 (accessed July 21, 2020). Available at `https://web.archive.org/web/20200721120737/https://sourceforge.net/projects/timplus/`.

[34] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[35] Sharan Vaswani, Branislav Kveton, Zheng Wen, Mohammad Ghavamzadeh, Laks Lakshmanan, and Mark Schmidt. Model-independent online learning for influence maximization. *arXiv preprint arXiv:1703.00557*, 2017.

[36] Sharan Vaswani, Laks Lakshmanan, Mark Schmidt, et al. Influence maximization with bandits. *arXiv preprint arXiv:1503.00024*, 2015.

[37] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the evolution of user interaction in facebook. In *WOSN*, pages 37–42, 2009.

[38] Yu Wang, Gao Cong, Guojie Song, and Kunqing Xie. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1039–1048, 2010.

[39] Zheng Wen, Branislav Kveton, Michal Valko, and Sharan Vaswani. Online influence maximization under independent cascade model with semi-bandit feedback. In *Advances in neural information processing systems*, pages 3022–3032, 2017.

[40] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

[41] Qingyun Wu, Zhige Li, Huazheng Wang, Wei Chen, and Hongning Wang. Factorization bandits for online influence maximization. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 636–646, 2019.

[42] Qingyun Wu, Zhige Li, Huazheng Wang, Wei Chen, and Hongning Wang. Factorization bandits for online influence maximization. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 636–646, 2019.

[43] Chuan Zhou, Peng Zhang, Jing Guo, Xingquan Zhu, and Li Guo. Ublf: An upper bound based approach to discover influential nodes in social networks. In *2013 IEEE 13th International Conference on Data Mining*, pages 907–916. IEEE, 2013.