A Comparative Study of Influence Maximization algorithms on Dynamic Networks

ANNEM KHYATHI REDDY*, B18CSE005
ADITYA KUMAR VERMA†, B18CSE004
VINIT RAMESH GORE‡, M20CS064

Abstract: Influence maximization problem has gained interest of many researchers in the past decade because of its wide umbrella of applications and challenges. Influence maximization has number of applications in real world even. Few of it's applications are viral marketing, epidemic spread analysis, exemplar-based clustering and many more.But most of the research in this field is done on the static networks but not on the dynamic networks. In this project we intend to compare various existing algorithms for influence maximization on the dynamic networks.

Keywords: *Influence Maximization, Information Diffusion, Dynamic networks, Social Networks, Submodular Maximization*

1 INTRODUCTION

Influence Maximization :- IM is a task of identifying the subset of k nodes from the nodes in given network such that activating the can influence maximum number of nodes in the network. IM has a vast range of applications. It is used widely in viral marketing. Selecting the nodes which can spread the product to maximum number of users is very crucial in advertising. IM is very useful in this case. IM is mostly used in epidemic spread analysis. In this case in helps in identifying the nodes which spread to disease more and control them to reduce the spread of the epidemic.

Dynamic Networks :- Dynamic networks are networks that generally vary from time to time. The vertices of dynamic networks are not binary, instead represent a probability for having link between two nodes. They are also known as temporal networks. Dynamic networks may be of different types. One of them is Incremental model where new users join overtime and new links are being created with time. Another one is fully dynamic model in which users can join and leave in the network in the due course of the time.

Applying IM on dynamic networks is much more challenging. Network might be given as a stream or a batch. In real-time scenarios determining the network at time t is difficult. This also requires approximation algorithms. IM on dynamic networks is convertible into 'Submodular Maximisation under cardinality constraint' problem.

1.1 Submodular Maximisation under cardinality constraint

The problem of maximising a function f under a cardinality constraint k can be framed as selecting Seed set $S \subseteq V$ with $|S| \leqslant k$ so as to maximise f(S).

2 PROBLEM STATEMENT

Influence maximization problem (IM) has wide-range applications such as viral marketing, epidemic spread analysis and exemplar-based clustering. Its parent problem of submodular maximization finds application in various machine learning tasks from Gaussian-process regression to active learning, sparse reconstruction to data summarization to name a few. With the explosion of data, the requirement for processing large data efficiently becomes extremely important and less research has been done in this direction. The problem of influence maximization on dynamic graphs is one of these less explored problems.

---

*Added proposal and details for Introduction details, Greedy Algorithm, MaxG probing algorithm details
†Added references and details for Fully dynamic algorithm
‡Implemented code for all experiments and added the details about the Sieve streaming and IC Maintenance algorithms in the report.

3 RELATED WORK

Influence Maximization on dynamic networks comes with a set of more challenges compared to static networks as the network is not the same at all the time stamps in the case of dynamic networks. This requires to obtain the network(G) at time t and then find the seed set from the graph at given time t.

The greedy algorithm proposed by Kempe et. al[8] is the most cited work in classical IM literature that gives $(1 - \frac{1}{e})$- approximate solution with strong theoretical guarantees. This is so far the best approximation in submodular maximization problems with cardinality constraints. However, it cannot be directly applied to the dynamic domain where complete graph is not available due to its large scale and updates to the graph are frequently occurring (for eg. over 9.68K tweets are sent per second[1]). Many previous approaches ([9] [10] [12] [11] [6]) in IM for dynamic graphs try to adapt Greedy algorithm to the dynamic domain. Others like ([14][5][13]) use heuristics. We would be exploring some of these papers in our work.

There are some other approaches that try to extend the existing algorithms originally devised for static graphs to the dynamic domain by applying heuristics and without provable approximation guarantees [7].

The definition of dynamic networks varies amongst previous works. Some papers consider a static network with changing links. Some consider a fully dynamic scenario of insertion/deletion of nodes as well as edges. We will consider this difference of definitions exclusively.

4 MAIN SECTION

Social influence is the phenomenon on which influence maximisation depends on mostly. Social influence is the phenomenon by which a node influences it's neighbours to behave similarly to it. We considered following algorithms for our study. In the subsequent sections we will give a brief overview of the algorithms under study in this paper. All the algorithms are based on the influence maximisation in dynamic networks.

4.1 Greedy Algorithm

As we all know IM problem is NP-hard. It cannot be solved in polynomial time. Thus greedy approach is incorporated in order to get the approximated solutions. However the classic greedy algorithm gives a nice approximation. It provides approximate solution of 1-1/e. Even though it provides a good approximation it is not that efficient in dynamic scenario.

Algorithm 1 Greedy Algorithm

Input: k; Influence function $\sigma(.)$
Output: S: Seed Set
1: S ← empty list
2: **for** $i \leftarrow 1...k$ **do**
3: $u^* \leftarrow argmax_{u \in V-S}(\sigma(S \cup u) - \sigma(S))$
4: $S \leftarrow S \cup u^*$
5: **end for**
6: return S

[1] https://www.internetlivestats.com/one-second/

4.2 MaxG probing Algorithm

4.3 Mathematical definition

Given a network G^0 at time t=0. Network keeps changing as the network is dynamic. Network G^t at time t can be approximated using probing. We should find set S of cardinality k at time t, such that influence spread $\sigma(S)$ is maximised.

4.3.1 Probing. We are provided with the graph at time t=0 itself. The graph at time t is unknown for us. Therefore we work on an observed network $\hat{G}^t$ at time t. For every time t>0, we keep on updating the network with the edges of the probed nodes on the observed network $\hat{G}(t-1)$ which is the observed network at time t-1. For time t=0, $\hat{G}^0$ will be same as G^0. The aim of our probing algorithm should be including nodes that bring most change in the network, so that output will approach the output of real solution. Mathematically we define $\hat{\sigma}_v(S)$ as the influence spread of the set S.

Let S_0 is the considered seed set before probing v. $S_0^{'}$ is the seed set containing node v along with the S_0. The goal is to maximise $\hat{\sigma}_v(S_0^{'})$ - $\hat{\sigma}_v(S_0)$ which is known as performance gap. ϵ is considered as tolerance. We should find performance gap for each node. $\beta(v)$ is the maximum performance gap by probing v. Maximum performance gap is the highest value satisfying $\mathrm{P}[\hat{\sigma}_v(S_0^{'})$ - $\hat{\sigma}_v(S_0) \geq \beta(v)\ ] \leq \epsilon$. This algorithm tries to maximise $\beta(v)$.

5 SIEVE STREAMING

The Sieve Streaming algorithm [4] is one of the most accepted algorithm in the domain of using streaming algorithms for submodular maximization problems. It provides $1/2 - \epsilon$-approximation guarantee to the optimum solution by passing through all the edges (connections between users) of the network coming in the form of a stream only once.

The algorithm requires $O(logk/\epsilon)$ update time per element and stores at most $O(\frac{nlogk}{\epsilon})$ elements. The total computation cost of algorithm is $O(\frac{nlogk}{\epsilon})$, which is much lesser than the classical greedy algorithm ($O(nk)$).

6 IC MAINTENANCE

The IC Maintenance algorithm by Wang et. al use the Sieve Streaming algorithm as the checkpoint oracle in their algorithm. The algorithm works on a sliding window streaming model. Since in the streaming setting, one edge is considered at a time, the approach followed by them is to include last N actions in the current window. Then for the next consequent action, the previous Nth action is removed from the window. The checkpoint oracle values for this new edge is calculated and stored. The checkpoint oracle values for the deleted action are deleted. The checkpoint oracle values for the remaining actions are copied from the previous window.

Algorithm 2 Sieve Streaming

Initialization

sieves ← empty list
for each γ in $logspace(m, 2 * k * m, 1 + \epsilon)$ **do**
    Add a $SingleThresholdSieve(f, k, \gamma)$ instance to sieves
end for

Insert

Input: element
sieves ← empty list
for sieve in sieves **do**
    sieve.process(element)
end for

Get Solution Value

Output: return best value from all sieves // *Function value of f*

SingleThresholdSieve.process

Input: element
$threshold = \gamma/2k$
if $sieve.solution.size < k$ **then**
    if marginal increase of element to f is greater than $threshold$ **then**
        add to solution
    end if
end if
Output: best value from all sieves

Algorithm 3 IC Maintenance

Insert

Input: *element*
Delete Δ_{t-1}. Create Δ_t. // Δ *is the checkpoint oracle*
for all $\Delta_{t-1}[i]$ **do**
    $\Delta_t[i-1] \leftarrow \Delta_{t-1}[i]$
end for
for all $\Delta_t[i]$ **do**
    $\Delta_t[i]$.process(element)
end for

Get Solution Set

Output: the solution set of $\Delta_t[1]$ // *Most-influential seed set*

7 FULLY DYNAMIC ALGORITHM[9]

Algorithm 4 Fully Dynamic

Initialization

$R \leftarrow \log(1 + Q1(2k))$
$T_i \leftarrow Y(1+Q1) < i; \forall_i$ s.t. $i \in [0, R]$
$T \leftarrow \log n$
$A_{i,l} \leftarrow \phi\ \forall_{i,l}$ s.t. $i \in [1, R], l \in [0, T]$
$S_{i,l} \leftarrow \phi; \forall i, l$ s.t. $i \in [1, R], l \in [0, T]$
$B_l \leftarrow \phi; \forall_L$ s.t. $L \in [0, T]$

Bucket-Construct

Input: i, l
loop
  $A_{i,l} = e \in A_{i,l}$ s.t. $T_i \leq f(e|Spread(i,l)) \leq T_{i-1}$
  if $|Ai, l| \geq 2T - l$ and $|Spread(R,T)| < k$ **then**
    $S_{i,l} \leftarrow S_{i,l} \cup Peeling(A_i, l, T_i, f)$
  end if
  until $|A_{i,l}| < 2T - l$ or $|Spread(R,T)| \geq k$
end loop

Insertion

Input: e
$Bl \leftarrow Bl \cup e; \forall_l$ s.t. $l \in [0, T]$
$V \leftarrow V \cup e$
if $\exists_l$ s.t. $|B_l| \geq 2T - l$ **then**
  Let $l*$ be the smallest such index
  $S_{i,l} \leftarrow \phi; \forall_{l*,i}$ s.t. $l' \in [0, T], i' \in [1, R]$
  $B_l \leftarrow \phi; \forall_l$ s.t. $l \in [1, T]$
  $Level - Construct(l*)$
end if

Deletion

Input: e
$A_{i,l} \leftarrow A_{i,l'} \backslash e; \forall_{i,l}$ s.t. $i \in [1, R], l \in [0, T]$
B_l ¸$B_{l'} \backslash e; \forall_l$ s.t. $l \in [0, T]$
$V \leftarrow V' \backslash e$
if $e \in Spread(R,T)$ **then**
  Let $S_{i,l}$ be the set containing e
  Remove e from $S_{i,l}$
  if the size of $S_{i,l}$ has reduced by ϵ fraction since it was constructed **then**
    $S_{i,l} \leftarrow \phi; \forall_{l,i}$ s.t. $l' \in [1, T], i \in [0, R]$
    $Level - Construct(l)$
  end if
end if

Level-Construct(l)

$B_l \leftarrow \phi$
for $i \leftarrow 1...R$ **do**
  if $l > 0$ **then**
    $A_{i,l} \leftarrow B_l - 1 \cup SRj = 0Aj, l - 1$
  else $A_{i,l} \leftarrow V$
  end if
  $Bucket - Construct(i, l)$
end for
if $|Spread(R,T)| \geq k$ **then**
  $A_{i,l} \leftarrow \phi$; ∀l < l <= T ∀1<=i<=R
end if
if $l < T$ and $|Spread(R,T)| < k$ **then**
  $Level - Construct(l + 1)$

7.1 Submodular Maximization

Submodularity is a property of set functions, i.e., functions f : $2^V \rightarrow$ R that assign each subset S $\subset$ V a value f(S). Hereby V is a finite set, commonly called the ground set. In our example, V may refer to the locations where sensors can be placed, and f(S) the utility (e.g., detection performance) obtained when placing sensors at locations S. In the following, we will also assume that f(ϕ) = 0, i.e., the empty set carries no value. Submodularity has two equivalent definitions, which we will now describe. The first definition relies on a notion of discrete derivative, often also called the marginal gain.

7.2 The Algorithm

This algorithm maintains a structured data structure that uses a number of families of element sets. For an index R, the algorithm maintains a sequence of zero thresholds.For index i, Ai,l is a set of items with marginal value. For index l, we define Ai,l as the set of items with minimal value. To include items from Ai,l, we need to select a random subset of the set.The algorithm used in this logic is called Bucket-Construct. Its goal is to find a high-quality random subset of Ai,l.The procedure for this problem is called the Peeling procedure. It takes as input a set N, which is equal to a set S, and selects a set S of sizes t uniformly at random. It avoids generating a lot of oracle queries.The batch insertion logic may require recomputed A-sets to maintain its operation. Fortunately, we can do this by introducing buffer sets Bl for each level. These sets have a capacity of 2Tl1.If a set exceeds its capacity, we pick the first one. We then reconstruct the set with the smallest l.

8 IMPLEMENTATION

8.1 Data Collection

The Email-Enron dataset[1] (Nodes-36692, Edges-183831) was downloaded from the SNAP repository[3]. For testing on small dataset, A Barabasi-Albert graph (Nodes-5000, Edges-10824), referred as SynBA here, was generated using Networkx[2].

8.2 Data Preprocessing

The data was read into a Networkx graph.

8.3 Experimental Setup

The Google Colaboratory environment was used for all experiments.

9 RESULTS

The algorithms were evaluated on the basis of two performance metrics: 1. Function value (solution value) - Spread of the function over time. Greater the magnitude, more good is the solution and,
2. Number of oracle calls - metric for measuring time of the algorithm in terms of function calls. The results for Greedy algorithm on synthetic graph SynBA are as follows:

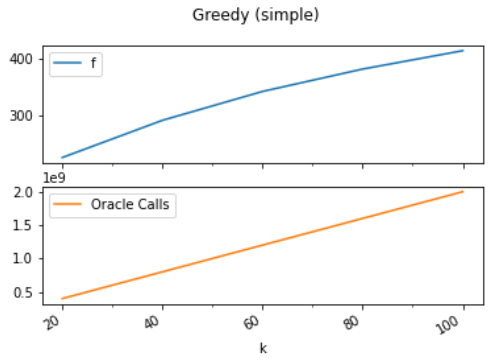

Fig. 1. Results of the Greedy Algorithm on SynBA graph

The results for Random algorithm on synthetic graph (BA) are as follows:

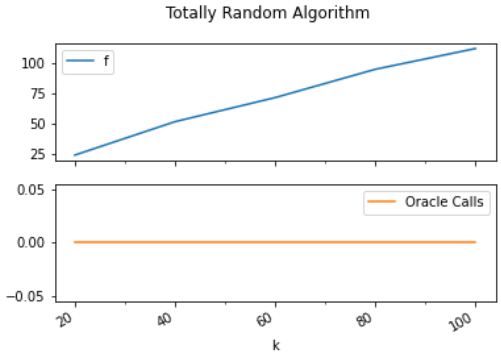

Fig. 2. Results of Totally Random Algorithm on SynBA graph

The results for Sieve Streaming algorithm on synthetic graph (BA) are as follows:

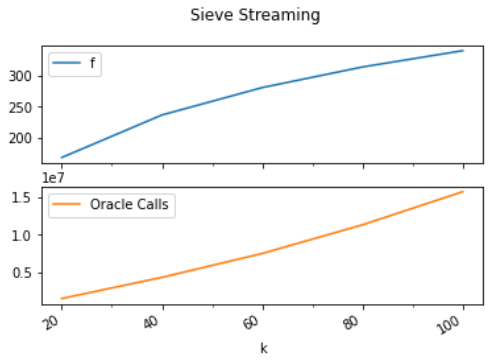

Fig. 3. Results of Sieve Streaming Algorithm on SynBA graph

The results for Random algorithm on synthetic graph (BA) are as follows:

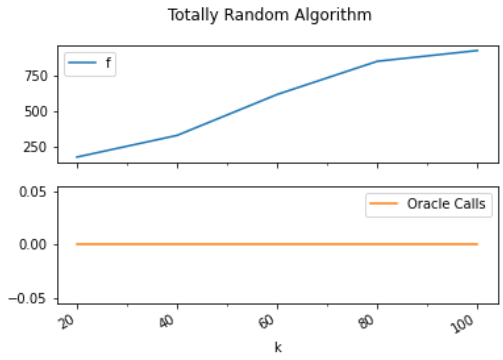

Fig. 4. Results of Totally Random Algorithm on Enron dataset

The results for Sieve Streaming algorithm on synthetic graph (BA) are as follows:

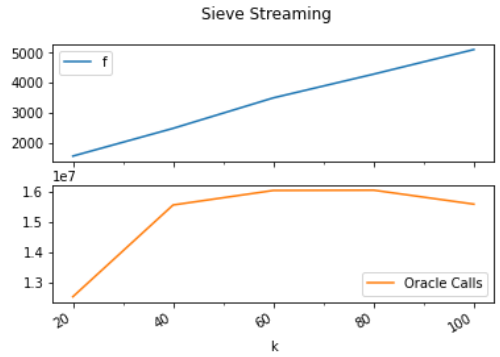

Fig. 5. Results of Sieve Streaming Algorithm on Enron dataset

10 CODE

The code of this project is available at https://colab.research.google.com/drive/1CXzSV5dv8rV-m_RoYdNAZQxgiN8vpSGd.

REFERENCES

[1] [n. d.]. Email-Enron dataset. http://snap.stanford.edu/data/email-Enron.html.
[2] [n. d.]. Networkx. https://networkx.org/. Open source package to work with networks on Python..
[3] [n. d.]. SNAP Repository. http://snap.stanford.edu/data/index.html. Large networks repository.
[4] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. 2014. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 671–680.
[5] Wei Chen, Chi Wang, and Yajun Wang. 2010. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1029–1038.

[6] Xiaodong Chen, Guojie Song, Xinran He, and Kunqing Xie. 2015. On Influential Nodes Tracking in Dynamic Social Networks. In *Proceedings of the 2015 SIAM International Conference on Data Mining, Vancouver, BC, Canada, April 30 - May 2, 2015*, Suresh Venkatasubramanian and Jieping Ye (Eds.). SIAM, 613–621. https://doi.org/10.1137/1.9781611974010.69

[7] Kyomin Jung, Wooram Heo, and Wei Chen. 2012. IRIE: Scalable and Robust Influence Maximization in Social Networks. In *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*, Mohammed Javeed Zaki, Arno Siebes, Jeffrey Xu Yu, Bart Goethals, Geoffrey I. Webb, and Xindong Wu (Eds.). IEEE Computer Society, 918–923. https://doi.org/10.1109/ICDM.2012.79

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

[9] Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Jakub Tarnawski, and Morteza Zadimoghaddam. 2020. Fully Dynamic Algorithm for Constrained Submodular Optimization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/9715d04413f296eaf3c30c47cec3daa6-Abstract.html

[10] Morteza Monemizadeh. 2020. Dynamic Submodular Maximization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/6fbd841e2e4b2938351a4f9b68f12e6b-Abstract.html

[11] Naoto Ohsaka, Takuya Akiba, Yuichi Yoshida, and Ken-ichi Kawarabayashi. 2016. Dynamic Influence Analysis in Evolving Networks. *Proc. VLDB Endow.* 9, 12 (2016), 1077–1088. https://doi.org/10.14778/2994509.2994525

[12] Yanhao Wang, Qi Fan, Yuchen Li, and Kian-Lee Tan. 2017. Real-time influence maximization on dynamic social streams. *arXiv preprint arXiv:1702.01586* (2017).

[13] Yu Yang, Zhefeng Wang, Jian Pei, and Enhong Chen. 2017. Tracking Influential Individuals in Dynamic Networks. *IEEE Trans. Knowl. Data Eng.* 29, 11 (2017), 2615–2628. https://doi.org/10.1109/TKDE.2017.2734667

[14] Honglei Zhuang, Yihan Sun, Jie Tang, Jialin Zhang, and Xiaoming Sun. 2013. Influence Maximization in Dynamic Social Networks. In *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*, Hui Xiong, George Karypis, Bhavani M. Thuraisingham, Diane J. Cook, and Xindong Wu (Eds.). IEEE Computer Society, 1313–1318. https://doi.org/10.1109/ICDM.2013.145