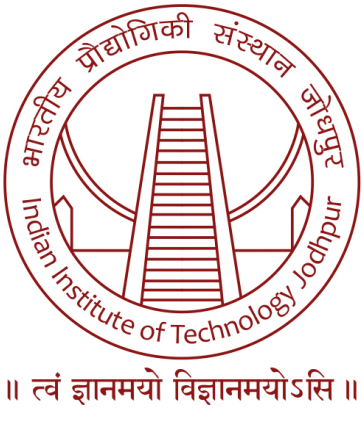
Estimating Diffusion Degree for Influence Maximization on graph streams

A Project Report Submitted by

Vinit Ramesh Gore

in partial fulfillment of the requirements for the award of the degree of

Master of Technology in Artificial Intelligence



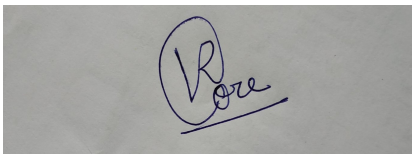
Indian Institute of Technology Jodhpur

Department of Computer Science and Engineering

June, 2022

Declaration

I hereby declare that the work presented in this Project Report titled Estimating Diffusion Degree for Influence Maximization on graph streams submitted to the Indian Institute of Technology Jodhpur in partial fulfilment of the requirements for the award of the degree of Master of Technology in Artificial Intelligence, is a bonafide record of the research work carried out under the supervision of Dr. Suman Kundu. The contents of this Project Report in full or in parts, have not been submitted to, and will not be submitted by me to, any other Institute or University in India or abroad for the award of any degree or diploma.



Signature

Vinit Ramesh Gore

M20CS064

Certificate

This is to certify that the Project Report titled Estimating Diffusion Degree for Influence Maximization on graph streams, submitted by Vinit Ramesh Gore(M20CS064) to the Indian Institute of Technology Jodhpur for the award of the degree of Master of Technology in Artificial Intelligence, is a bonafide record of the research work done by him under my supervision. To the best of my knowledge, the contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

suman kundu

Signature

Dr. Suman Kundu

Acknowledgements

I hereby thank my project advisor, Dr. Suman Kundu, for supporting my endeavour of coming up with a new contribution in the field of social network analysis. Without his frequent guidance, this work would not have been possible. I hereby offer my work to God first and then to the community at large.

Abstract

In recent years, solving social network problems like computation of centrality measures, community detection, event detection, influence maximization, network summarization, etc. on temporal networks has received significant attention. Temporal networks can be represented as a sequence of interactions and can be modelled as graph streams. However, less attention has been given to the computation of centrality measures in such a streaming setting. Diffusion degree centrality measure of a node exploits the neighborhood properties of the node during information diffusion. In this paper, we estimate the diffusion degree centrality measure on graph streams and then apply the estimates to solve the influence maximization problem - finding top-K influential nodes on real-world networks. Our proposed algorithm constrains memory usage to $O(n * q)$ space and requires $O(m * K)$ time.

Contents

List of Figures

List of Tables

Estimating Diffusion Degree for Influence Maximization on graph streams

1 Introduction and background

With the development in big data technologies, the appearance of large networks like social networks, collaboration networks, biological networks, travel and transportation networks, etc. has triggered their study and analysis. Consequently, these large networks are represented as graphs having nodes as the entities and edges as the relation or interaction between these entities. In any network, each node possess a unique characteristic of its own. These characteristics can be analysed and some numerical value could be provided to each node for the decided context. For example, the context could be analysing the degree of nodes. Consequently, nodes with high degrees will have higher values than nodes with lower degrees. Similarly, different contexts like determining how close each node is to all other nodes or how much important a node is to maintain the connectivity in the graph, etc. would assign different numerical values to every node. These values are also called as node centrality measures. Node centrality measures are useful to determine the importance of the node in a network for a given context. More importantly, centrality measures are useful in ranking the nodes for a given context [1]. In order to find the top-K influential nodes in a network, centrality measures like high degree or distance centrality [2] have been popular. Some works [3, 4] have also leveraged PageRank centrality to find the top-K influential nodes.

1.1 Diffusion Degree

Diffusion degree [5] is the node centrality measure that considers the degree of the node as well as the degree of its neighbors during the diffusion process. Here, diffusion process involves a cascading process starting from initial K nodes that formulates the spread of a new information over the edges over time. All edges are considered to have a common probability λ of transferring the information to another node called as diffusion probability. Mathematically, it is given by:

$$DD_u = \lambda * (d_u + \sum_{i \in \Gamma_u} d_i) \tag{1}$$

For every node u in the graph, Diffusion degree tends to build a BFS tree, containing the nodes that can be influenced by the node u. The first layer of this tree consists of neighbors of u, the second layer consists of the neighbors of these neighbors, and so on. For the diffusion process, the diffusion probability is usually in the order 10^{-2} to 10^{-1}. Therefore, the chance of activating the nodes in the third layer goes close to zero [6] and can be safely ignored. Thus, the number of nodes in the first two layers are used to decide the centrality value of the node. It was proposed to provide a practically efficient solution for the influence maximization problem on large graphs.

1.2 Temporal networks and graph streams

Traditionally, graph problems were limited to the static graph settings where the number of nodes and edges are fixed. In case of dynamic networks, only structural changes i.e. node addition/deletion or edge addition/deletion were modelled. Temporal networks are different from dynamic networks. These networks have the ability to collect and store large volumes of network data having fine granularity, with additional information associated to vertices/edges [7]. Although, the structure is not seen to be changing frequently, a lot of *interactions* take place *over time*. Storing and processing temporal networks gives rise to new problems and new computational challenges. Temporal networks can be represented as a sequence of interactions/edges without any loss of information.

Graph streams are inspired from data streams. In a data stream, data are presented as a sequence of data items (potentially infinite). For a small number of passes, typically constant or just one pass, and small memory compared to data size, the algorithm is expected to perform fast computation per data item processed, e.g., constant or poly-logarithmic. Specifically, a graph stream can be either a sequence of interactions (temporal network) or a sequence of edge addition/deletion (dynamic graphs). It is a computational model for representing temporal networks [7]. In this paper, we model temporal networks on graph streams. We work under the incremental setting [8] where only node/edge additions take place.

1.3 Computation of centrality measures in temporal graphs

This is an umbrella of well-studied problems and various approaches are proposed for different settings. Centrality measures of the nodes in a graph may quickly change over time. There are some works that represent the temporal graphs as a sequence of static graphs - they find the centrality measure over those static graphs. These approaches become impractical when large graphs are considered and real-time querying is required. Traditionally, the typical approach followed over large static graphs was to sample a subgraph from the large graph using sampling techniques like random walks, graph traversals, etc. The sketches used in the streaming settings are analogous to these sampling techniques. We define the problem as to find centrality measure of a given node x through *online queries* over a stream of graph interactions. Unlike many previous works, our definition provides centrality measures over a lossless graph model. In the proposed work, we estimate the diffusion degree of every node by sampling few neighbors of every node instead of storing the entire graph in the memory. We perform a single pass over the stream of graph edges and process every edge in $O(q)$ time ($q << n$). We utilize $O(n * q)$ space in the memory.

The paper is organized as follows: Related works are included in 3, Estimation of Diffusion degree estimates in 4.1, using them for influence maximization in 4.2, experiments and results are in 5, the discussions follow in 6.1, and finally the conclusions 6.2.

2 Problem definition and Objective

Name	Description
n	Total number of nodes in the graph
m	Total number of edges in the graph
d	Average degree of graph
d_{in}	Average in-degree of graph
d_u	Degree of node u
q	Number of neighbors to be stored in the summary
Γ_u	Set of neighbors of a node u
λ	Common diffusion probability
$S_q^{(u)}$	Set of q neighbors of a node u
DD_u	Diffusion degree of a node u
DDS_u	Estimated diffusion degree of a node u

Table 2.1: Important notations used in the paper

We define the problem as to find centrality measure of a given node x through *online queries* over a stream of graph interactions. A static graph is defined as $G = (V, E)$ while a stream graph is defined as $G = (T, V, W, E)$, where

- T : a time domain
- V : a set of nodes
- $W \subseteq T \times V$: a set of temporal nodes
- $E \subseteq T \times V \times V$: a set of links

s.t., $(t, u, v) \in E$ implies $(t, u) \in W$ and $(t, v) \in W$

The formal definition of Diffusion degree on static graphs is given by:

$$DD_u = \lambda * (d_u + \sum_{i \in \Gamma_u} d_i) \tag{2}$$

We aim to estimate Diffusion degree on graph streams using the following equation:

$$\widehat{DDS}_u = \lambda * (d_u + \frac{d_u}{q} * \sum_{j \in S_q^{(u)}} d_j) \tag{3}$$

The formal definition of the problem of influence maximization can be given as follows:

$$S^* = \underset{S \subseteq V, |S| \leq K}{\arg\max} f(S)$$

Given a monotone submodular function f, find the set S^* of size at most K, that maximizes the function f.

3 Literature survey

We classify the related works into three sections: influence maximization on static graphs and Influence Maximization on graph streams.

3.1 Influence Maximization

A social network is represented with a graph $G = (V, E)$ such that vertices or nodes (V) are the people and the edges (E) are their connections or relations. Given a graph G and a seed set size K, an influence maximization (IM) algorithm will output a seed set S of size K that has optimal influence spread. The influence spread ($f(S)$) is the number of users influenced in the entire network when the seed set S of nodes with a new behavior (e.g. adopted a new product) is considered. The nodes that have adopted the new behavior are considered as active nodes. Typically, $f(S)$ is the measure for evaluating an IM algorithm for accuracy. The function f is given by an information diffusion technique. The mechanisms by which information spreads over the network come under the umbrella of information diffusion techniques. The function f, given a seed set S, measures the spread of information over the diffusion process for certain number of discrete time steps and will return the number of active users at the end of the process. There are two popular information diffusion models in the literature: Independent cascade model and Linear threshold model. We will discuss and use the Independent cascade model in this paper. It was first proposed by Goldenberg, Libai and Muller [9][10]. As in any other diffusion model, the nodes are activated by their neighbors only indicating "word-of-mouth" interaction. Every edge is associated with certain probability which we call diffusion probability. At a given time step t, every active node gets only one chance to activate its neighbor. If it succeeds, the neighbor becomes active. A node may be activated by more than one of its neighbors, in which case the sequence of activation does not matter. In a progressive model where a node does not become inactive again after becoming active, this independent cascade process continues over time and the number of active nodes goes increasing until none of the active nodes succeed in activating its neighbor. That marks one *simulation* of the diffusion process. Since the process depends upon randomly generated values that are used to compare with edge probabilities, multiple such simulations are performed. Typically, 2000 or more simulations are performed to get a good expected value.

In [2], Kempe et al. formulated this problem as a discrete optimization problem and proposed a greedy algorithm that provided $O(1 - \frac{1}{e} - \epsilon)$ i.e. about 63% approximation. This is so far the best approximation attained. However, the Greedy algorithm is slow and often considered impractical. It takes hours to run over a large graph of 10^5 nodes. So the quest in the study of influence maximization has been to develop algorithms that reach the above approximation but have practical time complexities [6, 11, 12]. One of the methods to achieve this is to use heuristics. Centrality measures are the most popular heuristics used. These methods [1] are used to rank nodes based on their centrality values and the top-K nodes are selected as the seed set. These algorithms give output typically within a few seconds and the influence spread values are lesser but closer to the approximation guarantee of Greedy. However, these are heuristic

algorithms and no theoretical guarantees on the approximation can be given i.e. the approximation ratio may differ on different datasets. Our proposed method belongs to this classification. Another method to optimize Greedy algorithm is to use a summary of the graph to avoid time spent in running multiple simulations. These techniques maintain the approximation guarantee of the Greedy algorithm but reduce time by storing sketches. IMM algorithm [13] is a recent sketch-based algorithm that *provides accuracy same as that of Greedy but runs faster. We use this algorithm as the benchmark on measuring accuracy.*

3.2 Influence Maximization on graph streams

Streaming technologies are becoming popular with the increase in data availability. Temporal graphs can be streamed in the form of edge streams. In the sequence of interactions representation, the number of edges in the stream maybe relatively large. The problem of solving IM for dynamic graphs has received significant attention in the research literature. Many previous approaches ([14] [15] [16] [17] [18]) in IM for dynamic graphs try to adapt Greedy algorithm to the dynamic domain. Others like ([19][11][20]) use heuristics. The latter works share similar approaches of dealing with the problem as the proposed work. Using heuristics like centrality measures to estimate the ranking of nodes for influence spread has the advantage that every node can be considered independently. Thus, the computational effort required for traversing repeatedly over the collection of nodes is saved. It is important for any streaming algorithm to answer real-time queries quickly while processing the stream. In addition to that, the memory usage should be limited to satisfy the data stream model constraints. Essentially, the focus while designing the algorithm should be to keep both time and space complexities lower. There is another stream of research that solves the generalized problem of submodular optimization over the graph streams. In these works, they consider the function $f(S)$ to be submodular i.e. the marginal increase by adding a new node to a smaller set S is at least as high as that of adding a new node to a larger set S. According to Theorem 2.2 from [2], the function $f(S)$ used to measure influence spread is a submodular function. Some popular works, each building upon the previous algorithms, are the Sieve Streaming [21], Sieve Streaming++ [22], Three Sieves [23], etc. They try to mathematically solve the approximation problem and try to reach the approximation attained by the Greedy algorithm in the static domain. These algorithms develop effective summaries of the stream and answer real-time queries using those summaries. However, these generalized methods have to be customized for graph streaming settings.

4 Methodology

4.1 Estimation of Diffusion degree on graph streams

We are now in the position to get started with the proposed work. To find the Diffusion degree of a node u, DD_u [2], we require the degree of node u as well the sum of the degrees of all of its neighbors to be available. As discussed earlier, a streaming algorithm will try to summarize the given stream of large data in a limited amount of memory. A graph stored as an adjacency list requires $O(n + m)$ storage space. In practical cases while considering large graphs, where m is in the order of 10^6 or more, the space required to store the entire graph (about 229 GB for 10^7 edges) quickly exceeds typical memory sizes in most of the available RAMs today. In the proposed approach, we build a sketch that occupies $O(n * q)$ memory. The idea is simple: *to store q out of d_u neighbors of a node u instead of all the neighbors.* Here, q is an input integer value such that $q << d_{max}$ (maximum degree in the graph).
For this paper, we have focused only on directed graphs. For undirected graphs, this approach can be easily extended as discussed ahead. So every directed edge has a head node h_e and a tail node t_e. As a convention, for every edge encountered on the stream, we consider h_e influences t_e. So, for directed graphs, we store at most q out of d_u *incoming neighbors* of a node in the adjacency list.

Which q neighbors? We select q uniformly randomly sampled with replacement neighbors of u. Therefore, every one of these q neighbors will have the probability of $\frac{1}{d_u}$ to be selected to any one of the q places. Consequently, there can be multiple instances of a single node among these q neighbors. To achieve this, we use the random sampling with replacement technique proposed in [24] that performs the sampling as discussed above on a graph stream. The procedure followed is that initially an array of size q, $S_q^{(u)}$ having null entries is stored. For every edge in the stream that has u as its h_e, we run q independent Bernoulli trials having success probability as $\frac{1}{d_u}$. Among these, consider k number of trials are successes. We replace k nodes from $S_q^{(u)}$ indexed by their trial numbers with t_e. We need not know d_u apriori to perform this kind of sampling.
There are two cases when the number of neighbors in $S_q^{(u)}$ is less than q:

1. The total number of neighbors of u is less than q.
2. All q places in $S_q^{(u)}$ are not filled by random sampling.

In both cases, we consider those many number of neighbors instead of q. Now that we have our set of neighbors ready, we can use their degrees for estimation of diffusion degree. We take the *normalized empirical sum* over the selected neighbors to cover up for the remaining $d_u - q$ neighbors. Thus, we approximate the diffusion degree value by doing so.

Equation: Likewise, modification to the latter term in [2] give us the following equation for the estimates:

$$\widehat{DDS}_u = \lambda * (d_u + \frac{d_u}{q} * \sum_{j \in S_q^{(u)}} d_j)$$

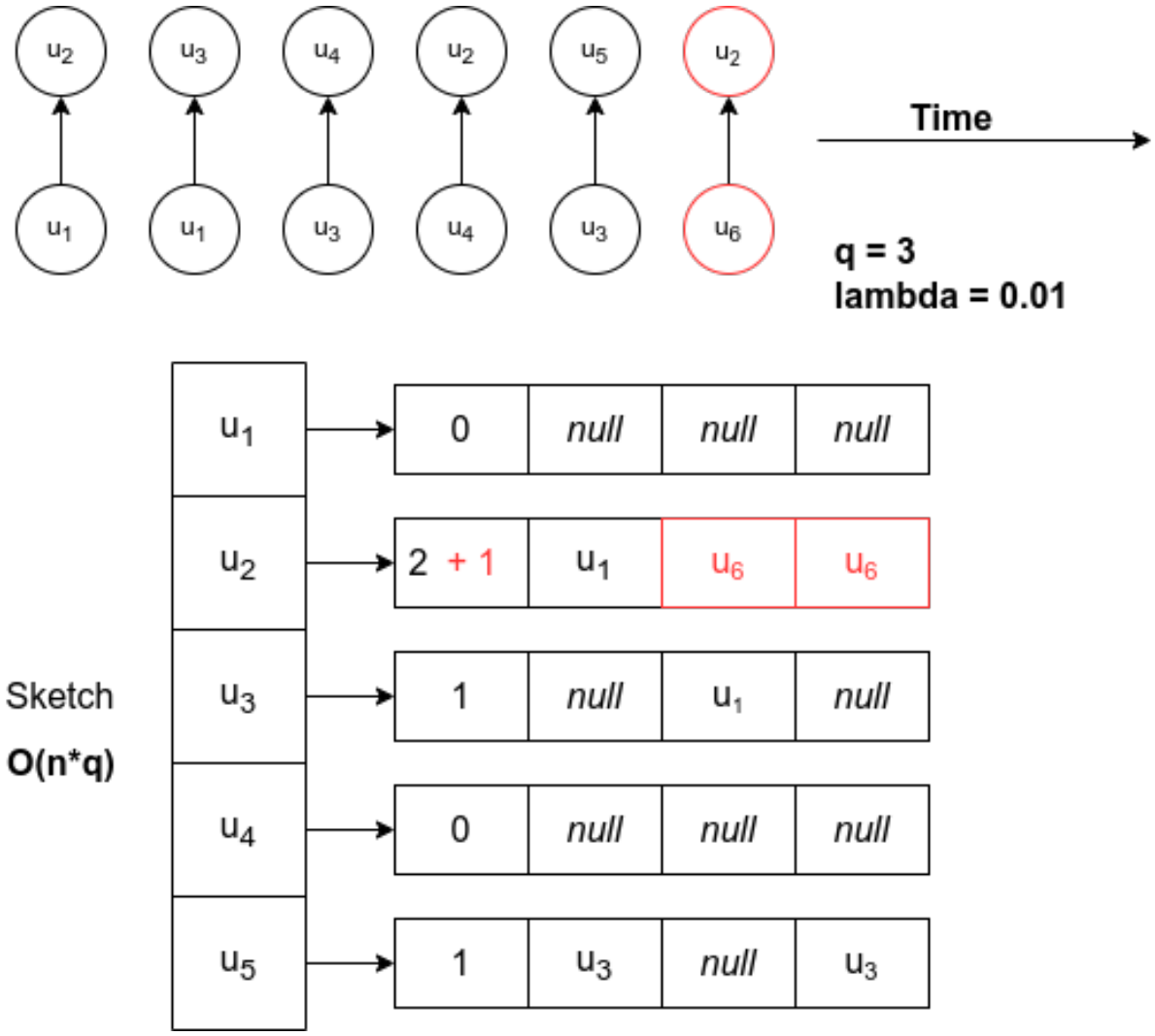


Figure 4.1: Data structure `ADJ` used to estimate diffusion degrees on edge stream of graph. For every node, we store a list of $q+1$ cells. First cell stores the degree count for the node, followed by q neighbors. q and λ are inputs. Every new neighbor is inserted according by random sampling with replacement.

Specifically, for every new edge (t_e, h_e), the h_e node is queried in the data structure. Its degree is incremented by 1 and t_e is added to its list at k positions using uniform random sampling with replacement as discussed before. Whenever a query for $\widehat{DDS}_u$ is made, 1 query is performed to get the list of u, then $O(q+1)$ queries are made to get its degree and traverse through its neighbors, and further $O(q)$ queries are made to get the degrees of its neighbors. Therefore, a total of $O(2q+1)$ queries are made to get the DDS value of a single node. This is how we estimate the Diffusion degree for a single node. Let us now proceed towards the larger picture. We will now understand how to store DDS for all nodes in a memory-efficient way. We shall construct an adjacency list-like data structure using a HashMap of lists. For every node u, the first element of its list will store the counter of the degree of u (d_u). The following elements will store at most q neighbors. 4.1 shows the visualization of the data structure. Initially all the q neighbors are *null*. The data structure will keep updating with the stream.

4.1.1 Correctness Claim

In this section, we prove that the estimates given by the proposed approach are correct. In probabilistic analysis, if the mean of the random variable is equal to the actual value, then it is said that the random variable provides correct estimates.

Theorem: The expected value of the Diffusion degree estimate is the same as the Diffusion degree value, i.e.

$$E[\widehat{DDS}_u] = DD_u$$

proving the correctness of Diffusion degree estimate.

Proof:

$$E[\widehat{DDS}_u] = E[\lambda * (d_u + \frac{d_u}{q} * \sum_{j \in S_q^{(u)}} d_j)]$$

$$E[\widehat{DDS}_u] = \lambda * (d_u + d_u.E[\frac{1}{q} * \sum_{j \in S_q^{(u)}} d_j]) \qquad \{\text{linearity of expectations}\}$$

$$E[\widehat{DDS}_u] = \lambda * (d_u + E[d_u].E[(\frac{1}{q} * \sum_{j \in S_q^{(u)}} d_j)]) \qquad \{\text{degrees are independent}\}$$

Here, $\frac{1}{q} * \sum_{j \in S_q^{(u)}} d_j$ itself is a random variable. Let X_1, X_2, ..., X_q be independent random variables such that $X_l = d_j$ for $l \in [q]$ and $j \in S_q^{(u)}$ and degree d_j is in the range $[a, b]$. The nodes j_1, j_2, ..., j_q are sampled from Γ_u uniformly at random.

Therefore, $X_l, l \in [q]$ are i.i.d. random variables. Therefore:

$$E[X_l] = E[X] = \frac{1}{d_u} \sum_{i \in \Gamma_u} d_i \{\text{expectation of a uniform random variable}\} \tag{4}$$

$$\text{Define: } \bar{X} = \frac{1}{q} \sum_{l \in [q]} X_l \tag{5}$$

$$\text{Therefore, } E[\bar{X}] = \frac{1}{q} \sum_{l \in [q]} E[X_l] \qquad \{\text{linearity of expectations}\}$$

From [4], we have:

$$= \frac{1}{q} \sum_{l \in [q]} (\frac{1}{d_u} \sum_{i \in \Gamma_u} d_i)$$

$$= \frac{1}{d_u} \sum_{i \in \Gamma_u} d_i . \frac{1}{q} \sum_{l \in [q]} (1) \qquad \{\text{sum does not have any } l \text{ term}\}$$

$$= \frac{1}{d_u} \sum_{i \in \Gamma_u} d_i . \frac{1}{q} . q$$

$$E[\bar{X}] = \frac{1}{d_u} \sum_{i \in \Gamma_u} d_i \tag{6}$$

Therefore, after substituting the value of $\bar{X}$ from [5] in the previous equation, we get:

$$\begin{aligned}
E[\widehat{DDS}_u] &= \lambda * (d_u + d_u.E[\bar{X}]) \\
&= \lambda * (d_u + d_u.\frac{1}{d_u} \sum_{i \in \Gamma_u} d_i) && \{\text{from [6]}\} \\
&= \lambda * (d_u + \sum_{i \in \Gamma_u} d_i) \\
E[\widehat{DDS}_u] &= DD_u && \{\text{Hence proved}\}
\end{aligned}$$

Upper Bound on error of estimate: In this section, we provide a bound on how large the error can go for a given set of input parameters in the proposed approach:

Corollary: For any $\epsilon, \delta \in (0,1)$, with q equals $O(\epsilon^{-2} log\frac{1}{\delta})$, the error between estimate and actual value is given by:

$$|\widehat{DDS}_u - DD_u| \leq \epsilon(b_u - a_u)d_u\lambda$$

with probability $1 - \delta$. Here, ϵ is the approximation of error and δ is the error probability. We define b_u, a_u as max degree and min degree amongst the neighbors of u respectively.

Proof: Consequently, in Hoeffding's inequality, we can use $\bar{X}$ from equation [5] as the estimated random variable. Putting it in the Hoeffding's inequality, we have:

$$\begin{aligned}
Pr[|\bar{X} - E[\bar{X}]| \geq t] &\leq 2\exp(\frac{-2qt^2}{(b_u - a_u)^2}) \\
Pr[|\frac{1}{q} \sum_{j \in S_q^{(u)}} d_j - \frac{1}{d_u} \sum_{i \in \Gamma_u} d_i| \geq t] &\leq 2\exp(\frac{-2qt^2}{(b_u - a_u)^2}) && \text{From [5] and [6]} \\
Pr[|\frac{d_u}{q} \sum_{j \in S_q^{(u)}} d_j - \sum_{i \in \Gamma_u} d_i| \geq td_u] &\leq 2\exp(\frac{-2qt^2}{(b_u - a_u)^2}) \\
Pr[|\lambda * (d_u + \frac{d_u}{q} \sum_{j \in S_q^{(u)}} d_j) - \lambda(d_u + \sum_{i \in \Gamma_u} d_i)| \geq td_u\lambda] &\leq 2\exp(\frac{-2qt^2}{(b_u - a_u)^2}) \\
Pr[|\widehat{DDS}_u - DD_u| \geq td_u\lambda] &\leq 2\exp(\frac{-2qt^2}{(b_u - a_u)^2}) && \text{From [4.1] and [2]}
\end{aligned}$$

For $\epsilon \in (0,1)$, we put $t = \epsilon(b_u - a_u)$:

$$Pr[|\widehat{DDS}_u - DD_u| \geq \epsilon(b_u - a_u)d_u\lambda] \leq 2\exp(-2q\epsilon^2)$$

Let $\delta \in (0,1)$ such that $\delta \geq 2\exp(-2q\epsilon^2)$

$$Pr[|\widehat{DDS}_u - DD_u| \geq \epsilon(b_u - a_u)d_u\lambda] \leq 2\exp(-2q\epsilon^2) \leq \delta \tag{7}$$

Therefore,

$$|\widehat{DDS}_u - DD_u| \leq \epsilon(b_u - a_u)d_u\lambda \text{ with probability } 1 - \delta$$

4.1.2 Algorithm

The algorithm is given in 1. The algorithm maintains an adjacency matrix `ADJ` of size $n * q$ to store the summary of the stream at every time step t. Every row r of `ADJ` stores the summary for the node u_r. The first cell of every row counts the degree of u_r over the stream. The remaining q cells store the q neighbors of u_r with replacement. Initially these q cells are *null*.

Algorithm 1 Estimate Diffusion degree on streams: DDS(object)

Require: q: max number of neighbors, $\texttt{ADJ}_{t-1}$: adjacency matrix of size $n * q$ from previous time step, $\texttt{ADJ}_{t'}$: when queried

— $next(t_t, h_t)$ —
$\texttt{ADJ}_t \leftarrow \texttt{ADJ}_{t-1}$
$\texttt{ADJ}_t[h_t][0] \leftarrow \texttt{ADJ}_{t-1}[h_t][0] + 1$
$rswr(t_t, h_t, q)$
— $query_{t'}(u)$ —
$d_u^{(t')} \leftarrow \texttt{ADJ}_{t'}[u][0]$; $sum \leftarrow 0$; $nCount \leftarrow 0$
for $i \in [q]$ **do**
  if $\texttt{ADJ}_{t'}[u][i]$ is not *null* **then**
    $nCount \leftarrow nCount + 1$; $sum \leftarrow sum + \texttt{ADJ}_{t'}[\texttt{ADJ}_{t'}[u][i]][0]$
  end if
end for
if $nCount > 0$ **then**
  return $\lambda\ (\frac{d_u^{(t')}}{nCount} * sum + d_u^{(t')})$
else
  return 0
end if

When an edge (t_t, h_t) from the stream appears at time step t, it get processed in the following way. As discussed in 4.1, we would be considering the in-degree for the nodes. Therefore, we consider the head node h_t for processing and the only row in `ADJ` updated at time step t is that of node h_t. The degree of h_t is incremented by one. After, the $rswr$ function is called. As discussed in 4.1, it performs q Bernoulli trials with probability equivalent to $1/d_{h_t}^{(t)}$ and stores the result of each one of them in an array of size q. Say k of these trials succeed, then we replace the cells at these k indexes with t_t. At this point, the ingredients to find the estimate of Diffusion degree for h_t are available to us. The $query_{t'}(u)$ function finds the estimate of Diffusion degree from the stored summary in $\texttt{ADJ}_{t'}[u]$. Notice that t' may be different from t. i.e. query can occur anytime irrespective of the time steps in the stream. To query, we find the count of neighbors with duplicates stored in the row array $\texttt{ADJ}_{t'}[u]$. There may be some cells that were never filled with any neighbor. So only not null cells are considered. We add up the degrees of these neighbors and store them in sum. When $nCount$ is 0, no neighbors were added so the estimate becomes 0. Otherwise, we are returning the normalized empirical sum value as given in [4.1].

4.1.3 Analysis

With this algorithm, our goal is to process the edge stream in a limited memory. The above algorithm requires O($n*q$) memory. If the size of the node datatype is s, the total memory required is nqs. Coming to the time complexity, atleast O(q) time is required for every call of $rswr(t_t, h_t, q)$ and that of $query(u)$. In total, the algorithm accesses the data structure `ADJ` $2q + 1$ times.

4.2 Influence maximization using Diffusion degree estimates

In the previous section, we saw how to find diffusion degree estimates for a single node. Then we saw how much time is required to query for the estimate for a single node. In 1, we saw the procedure of estimation of diffusion degree of a single node. In this section, we will use the diffusion degree estimates to design a simple algorithm that uses the Diffusion degree estimates to provide an approximate solution for the Influence maximization problem. We will then analyze the time and space complexities of the complete approach.

4.2.1 Algorithm

2 ranks nodes using the Diffusion degree estimates as the heuristic and outputs a seed set of top-K nodes. It maintains a min-heap `HEAP` to maintain the top-K influential nodes at every time step t. `HEAP` stores the tuple (estimate, node) for every node added to it. The tuples are arranged in the heap according to their estimate of Diffusion degree values. Smaller estimates are evicted over time.

Algorithm 2 Influence Maximization using Diffusion degree estimates

Require: K: Seed set size, q: max number of neighbors, dds: object of DDS, `HEAP`: min-heap of size K

— $next(e_t)$ —
$(t_t, h_t) \leftarrow e_t$; $\texttt{HEAP}_t \leftarrow \texttt{HEAP}_{t-1}$
$dds.next(t_t, h_t)$
$estimate \leftarrow dds.query_t(h_t)$
if h_t is in $\texttt{HEAP}_{t-1}$ **then**
  update estimate of h_t with $estimate$ in $\texttt{HEAP}_t$
else
  $minEstimate, minNode \leftarrow \texttt{HEAP}_{t-1}.peek()$
  if $estimate > minEstimate$ **then**
    add $(estimate, h_t)$ to $\texttt{HEAP}_t$; $\texttt{HEAP}_t.pop()$
  end if
end if
— $Query_{t'}()$ —
return `HEAP`

We first process the edge using $dds.next(t_t, h_t)$. Then, we find the diffusion degree estimate of h_t using $dds.query(h_t)$. While adding to `HEAP`, we first check if the node h_t was previously added to heap. We perform a linear search over the heap to search for h_t. If found, we update its estimate with the new $estimate$. Else, if $estimate$ is greater than the smallest estimate, we add h_t to the heap and evict the node with the smallest estimate.

4.2.2 Analysis

The space occupied by the min-heap `HEAP` is O(K). Since, in the `HEAP` we are storing the output seed set, we do not consider the memory occupied by it in space complexity analysis. Therefore, the space complexity of the algorithm is the same as that of DDS - O(nq). The time complexity of the algorithm is O($m * K$). We run over a large edge stream of size m, and we process each edge in O(K) time where K time is required to search through the heap ($K << m$) (refer 2 line 5). However, we search through the heap only if the $estimate > minEstimate$. As the algorithm proceeds over the stream, the $minEstimate$ goes on increasing. Hence, the likelihood of the above comparison being $True$ goes on decreasing. So, the amortized analysis of the time complexity would give value less than O($m * K$). However, atleast O(q) time is required for every call of *dds* methods. Therefore, time complexity can also be given as $\Omega(m * q)$.

What is the range of q? Typically, for an approximation algorithm, user inputs are approximation term (ϵ) and error probability (δ). With these two parameters, the user can tune the accuracy of the result while keeping the probability of success $(1 - \delta)$ above certain threshold. However, for the proposed algorithm, the user inputs are number of neighbors to be stored in the summary (q) and δ. q indirectly controls the error term ϵ, as the value of ϵ can be derived from q as we will see later. Moreover, q is a more relatable parameter than ϵ in the context of our algorithm.

From 7, we have $2exp(-2q\epsilon^2) \leq \delta$. Therefore, $\epsilon \geq \sqrt{\frac{1}{2q} log \frac{2}{\delta}}$. Thus, ϵ is inversely proportional to $\sqrt{q}$. Therefore, larger values of q will give smaller error values. However, keeping q arbitrarily large will make the `ADJ` data structure occupy more space.

Claim: To gain the space advantage of the algorithm, q should be less than $d_{in} - 1$, where d_{in} is the average in-degree of the nodes in the graph stream.

Proof: Considering the size of the datatype of nodes as implicit, the space occupied by a graph stored as an adjacency list is $n + m$. On the other hand, the space occupied by `ADJ` is given by $n + n(q + 1)$. To get space advantage for the proposed algorithm, the space occupied by `ADJ` should be less than space required to store the graph without approximations.

$$
\begin{aligned}
n + n(q + 1) &< n + m_{in} \\
\text{Here, } d_{in} &= \frac{m_{in}}{n} \\
n + n(q + 1) &< n + n d_{in} \\
q + 2 &< d_{in} + 1 \\
q &< d_{in} - 1
\end{aligned}
$$

Hence, q could be safely chosen in $[1, d_{in} - 2]$.

5 Experimental findings

In order to validate our theoretical claims, we performed experiments to compare the IM using Diffusion degree and IM using Diffusion degree estimates with each other.

5.1 Experimental Setup

For the experiments, we used two datasets as given in 5.1. The NetHept dataset [25] and the Mathoverflow-c2q [26] dataset are directed social networks. The NetHept dataset represents structures and properties of a real-world social network. It is used in many social network influence maximization studies. The Mathoverflow-c2q dataset is a real-world large social network. For diffusion degree, we load the graph as a Networkx [27] object. On the other hand, to simulate the streaming setting, we iterate over a file of directed edges and process each edge at a time. For IMM, we reuse the code available online[1]. Diffusion degree and IMM are deterministic while Diffusion degree on streams in stochastic.

Name	n	m	d_{in}
NetHept	15229	62752	4.121
MathOverflow-c2q	16826	101329	6.02

Table 5.1: Datasets used

For each of the three algorithms and for every seed set size K, we first find the optimal seed set using the algorithm. Then we use NDLib [28] [29] library for information diffusion to simulate Independent cascade model by keeping the seed set as initial active nodes. Every experiment is run for a certain number of simulations and then the mean over the result of all simulations is considered as the influence spread for that experiment. The value of q is taken to be $d_{in} - 1$ for both networks. Further, since the diffusion degree stream is stochastic, we run 5 rounds of the algorithm - every time we find different seed sets and then perform diffusion. Finally we take the mean over the results of all the runs.

5.2 Results

The [5.1] show the experimental results. The left plots show the comparison of the influence spread performed by Diffusion degree (DD) and Diffusion degree on streams (DDS). We observe that the influence spread values attained by both the algorithms are close to each other. Thereby, we successfully verify the correctness claim made earlier. The closeness of the influence spread values of DD or DDS with the benchmark IMM algorithm asserts that approximation performed by the Diffusion degree heuristic is closer to the best approximation $(1 - \frac{1}{e})$. The right plots show the mean error of the K DDS estimates in its seed set to their actual DD values.

The results in 5.2 show the trend in time duration measured in seconds. However, the time required to load the graph object would be $O(m)$ and to run Diffusion degree over it is $O(nlogn)$. So, time taken by Diffusion degree would be $O(m)$. In case of Diffusion degree on streams, time required is $O(mK)$. Here, we see the slightly increasing trend in the time duration. However, since DDS is stochastic the trend is

[1]IMM source code: https://sourceforge.net/projects/im-imm/

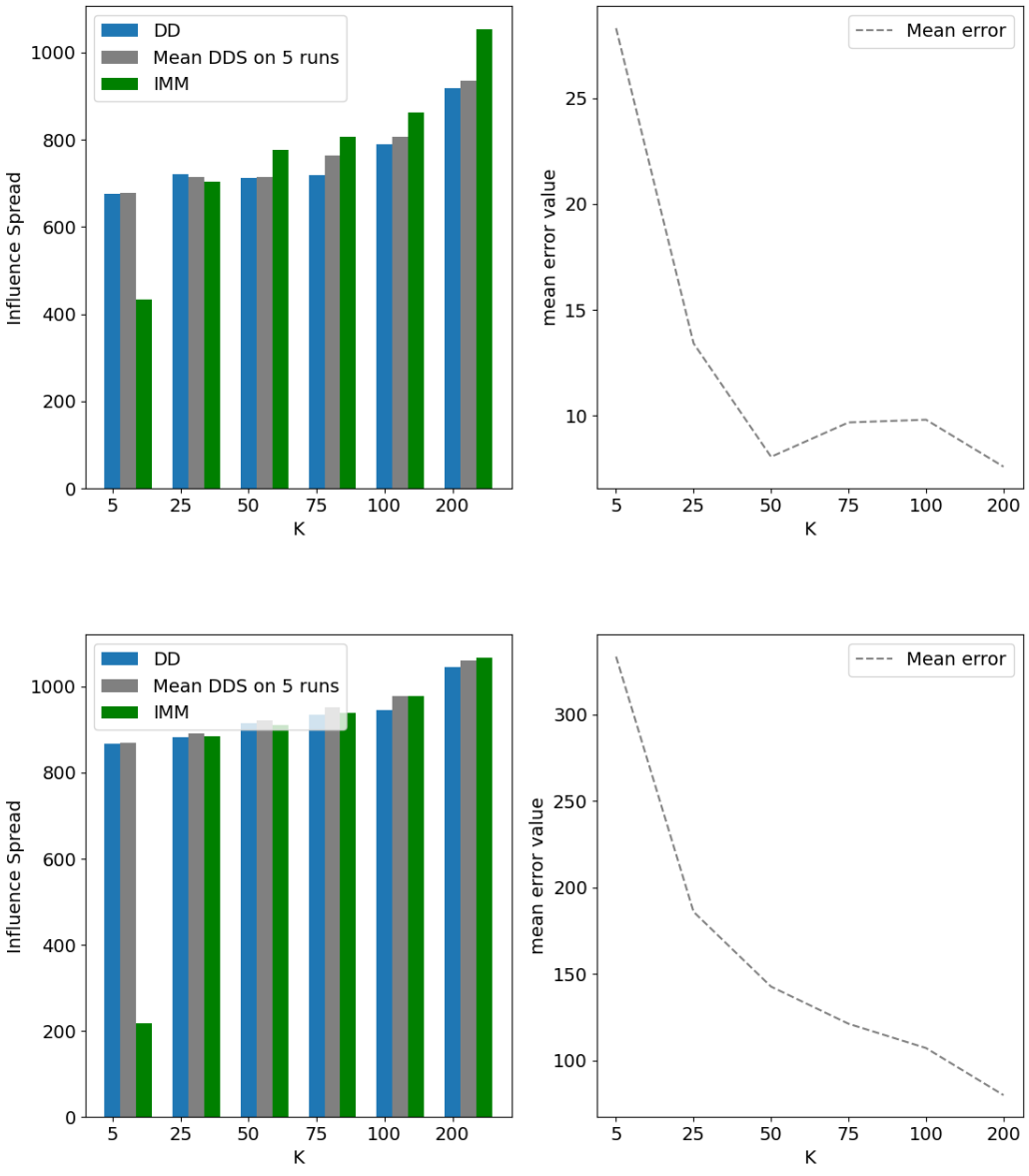


Figure 5.1: (Left) Influence spread w.r.t K on graphs NetHept (Top) and MathOverflow-c2q (Bottom). (Right) Mean error between DD and DDS estimates measured on single runs on the same datasets.

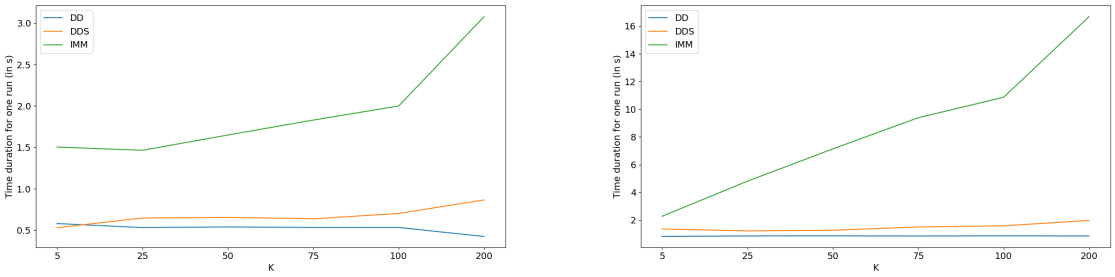


Figure 5.2: Time required on NetHept (Left) and MathOverflow-c2q (Right)

not stable. IMM requires $O(\frac{K(m+n)logn}{\epsilon^2})$ time i.e. more time than Diffusion degree and its streaming counterpart.

6 Summary and Future plan of work

6.1 Discussion

The problem of influence maximization on graph streams has received less attention in the research literature. Very few attempts are made to solve this problem. In this paper, we have successfully attempted to solve the influence maximization - an NP-Hard problem in the streaming setting. Our algorithm is applicable in memory-constrained environments where the summary of the graphs of size O(nq) is stored instead of the entire graph. Here q can be any positive value less than average degree (d) - 1 to get memory advantage. Furthermore, it can be used to query the stream in real-time with the processing time of O(K). It provides better time complexity over the popular influence maximization algorithms like Greedy. We validate our theoretical claims by relevant experiments over real-world graphs. In order to extend the proposed approach to undirected graphs, we have to consider both directions for an incoming edge. In that case, for every edge, there will be an update in the row of both the vertices in the `ADJ` matrix. While querying the estimate, degree should be considered instead of in-degree. By this paper, we introduce the possibility of bringing algorithms for influence maximization on static graphs into the streaming setting. In that regard, the works in the literature involving building effective sketch or summary of a stream of a dataset [21], [22], [23] could be leveraged. We assert that the algorithms that provide node ranking based on some heuristics like centrality measures could be similarly converted to the streaming setting. On the other hand, is there a possibility of converting those algorithms to the streaming setting that run the diffusion process to select the next best node? In that case, the key question would be: how to model information diffusion in the streaming setting?
We had also made an attempt to further reduce the space requirements of the sketch used by the algorithm from O(n) to O($logm$). For this, we considered using the Count Sketch algorithm to store the degree of every node with some approximation. Alongside, we considered a heap of size cK, where $c > 0$. The idea was to store top-cK nodes in the heap such that the high-degree neighbors of the top-K also appear in the top-cK nodes. Essentially, we considered penalizing the low degree neighbors of the top-K nodes. However, we could not find a data structure to store the references of these neighbors efficiently due to time constraints. Having said that, there is a possibility of considering this approach as a future work.

6.2 Conclusion

In this study, we have successfully tried to address the problem of computing diffusion degree centrality measure over temporal networks modelled as graph streams. Our contribution shows that it is possible to compute centrality measures over temporal networks in a streaming setting. We, thereby, proposed an algorithm that maintains a simple sketch to estimate the diffusion degree of a node in a single pass over the edge stream of the graph. We provide theoretical proofs and experiments to support our claims. Our method finds application in finding top-K influential nodes on a graph stream. At the end, we mention the possibilities and open problems in the discussions.

References

[1] A. Saxena and S. Iyengar, "Centrality measures in complex networks: A survey," *arXiv preprint arXiv:2011.07190*, 2020.

[2] D. Kempe, J. Kleinberg, and É. 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*, 2003, pp. 137–146.

[3] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, T. Mishima, and M. Onizuka, "Fast and exact top-k algorithm for pagerank," in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[4] Y. Zhang, J. Zhou, and J. Cheng, "Preference-based top-k influential nodes mining in social networks," in *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2011, pp. 1512–1518.

[5] S. Kundu, C. Murthy, and S. K. Pal, "A new centrality measure for influence maximization in social networks," in *International conference on pattern recognition and machine intelligence*. Springer, 2011, pp. 242–247.

[6] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 199–208.

[7] P. Rozenshtein and A. Gionis, "Mining temporal networks," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 3225–3226.

[8] A. McGregor, "Graph stream algorithms: a survey," *ACM SIGMOD Record*, vol. 43, no. 1, pp. 9–20, 2014.

[9] J. Goldenberg, B. Libai, and E. Muller, "Talk of the network: A complex systems look at the underlying process of word-of-mouth," *Marketing letters*, vol. 12, no. 3, pp. 211–223, 2001.

[10] ——, "Using complex systems analysis to advance marketing theory development: Modeling heterogeneity effects on new product growth through stochastic cellular automata," *Academy of Marketing Science Review*, vol. 9, no. 3, pp. 1–18, 2001.

[11] W. Chen, C. Wang, and Y. Wang, "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*, 2010, pp. 1029–1038.

[12] Y. Li, J. Fan, Y. Wang, and K.-L. Tan, "Influence maximization on social graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 10, pp. 1852–1872, 2018.

[13] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A martingale approach," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, 2015, pp. 1539–1554.

[14] S. Lattanzi, S. Mitrovic, A. Norouzi-Fard, J. Tarnawski, and M. Zadimoghaddam, "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*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/9715d04413f296eaf3c30c47cec3daa6-Abstract.html

[15] M. Monemizadeh, "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*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/6fbd841e2e4b2938351a4f9b68f12e6b-Abstract.html

[16] Y. Wang, Q. Fan, Y. Li, and K.-L. Tan, "Real-time influence maximization on dynamic social streams," *arXiv preprint arXiv:1702.01586*, 2017.

[17] N. Ohsaka, T. Akiba, Y. Yoshida, and K. Kawarabayashi, "Dynamic influence analysis in evolving networks," *Proc. VLDB Endow.*, vol. 9, no. 12, pp. 1077–1088, 2016. [Online]. Available: http://www.vldb.org/pvldb/vol9/p1077-ohsaka.pdf

[18] X. Chen, G. Song, X. He, and K. Xie, "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*, S. Venkatasubramanian and J. Ye, Eds. SIAM, 2015, pp. 613–621. [Online]. Available: https://doi.org/10.1137/1.9781611974010.69

[19] H. Zhuang, Y. Sun, J. Tang, J. Zhang, and X. Sun, "Influence maximization in dynamic social networks," in *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*, H. Xiong, G. Karypis, B. M. Thuraisingham, D. J. Cook, and X. Wu, Eds. IEEE Computer Society, 2013, pp. 1313–1318. [Online]. Available: https://doi.org/10.1109/ICDM.2013.145

[20] Y. Yang, Z. Wang, J. Pei, and E. Chen, "Tracking influential individuals in dynamic networks," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 11, pp. 2615–2628, 2017. [Online]. Available: https://doi.org/10.1109/TKDE.2017.2734667

[21] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause, "Streaming submodular maximization: Massive data summarization on the fly," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 671–680.

[22] E. Kazemi, M. Mitrovic, M. Zadimoghaddam, S. Lattanzi, and A. Karbasi, "Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3311–3320.

[23] S. Buschjäger, P.-J. Honysz, L. Pfahler, and K. Morik, "Very fast streaming submodular function maximization," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2021, pp. 151–166.

[24] B.-H. Park, G. Ostrouchov, and N. F. Samatova, "Sampling streaming data with replacement," *Computational statistics & data analysis*, vol. 52, no. 2, pp. 750–762, 2007.

[25] N. Chen, "On the approximability of influence in social networks," *SIAM Journal on Discrete Mathematics*, vol. 23, no. 3, pp. 1400–1415, 2009.

[26] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks," in *Proceedings of the tenth ACM international conference on web search and data mining*, 2017, pp. 601–610.

[27] "Networkx," https://networkx.org/, open source package to work with networks on Python.

[28] G. Rossetti, L. Milli, S. Rinzivillo, A. Sîrbu, D. Pedreschi, and F. Giannotti, "Ndlib: a python library to model and analyze diffusion processes over complex networks," *International Journal of Data Science and Analytics*, vol. 5, no. 1, pp. 61–79, 2018.

[29] G. Rossetti, L. Milli, S. Rinzivillo, A. Sirbu, D. Pedreschi, and F. Giannotti, "Ndlib: Studying network diffusion dynamics," in *2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2017, pp. 155–164.