CutTheTail: an Accurate and Space-Efficient Heuristic Algorithm for Influence Maximization

Diana Popova[1], Ken-ichi Kawarabayashi[2] and Alex Thomo[1]

[1]*Computer Science Department, University of Victoria, British Columbia, Canada*
[2]*National Institute of Informatics, Tokyo, Japan*
Email: dpopova@uvic.ca, k_keniti@nii.ac.jp, thomo@uvic.ca

The algorithmic problem of finding the most influential nodes in an arbitrary directed graph (influence maximization) is an important theoretical and practical problem and has been extensively studied for decades. For massive graphs (*e.g.*, modelling huge social networks), randomized algorithms are the answer as the exact computation is prohibitively complex, both for the runtime and space. This paper concentrates on developing new accurate and efficient randomized algorithms that drastically cut the memory footprint and scale up the influence maximization. Using the Reverse Influence Sampling method proposed by Borgs, Brautbar, Chayes, and Lucier in 2014, we engineered a novel algorithm, CutTheTail, that finds a good approximation to the optimal influence maximization while using up to five orders of magnitude smaller space than the existing renown algorithms. CutTheTail is a heuristic algorithm. We tested the accuracy of CutTheTail on large real-world graphs using Monte Carlo simulation as the benchmark. Experiments show that CutTheTail provides solutions with quality comparable to the quality of the algorithms with theoretically proven approximation to the optimal influence maximization. Savings in required space allow to successfully run CutTheTail on a consumer-grade laptop for a graph with 640 million edges. To the best of our knowledge, no other influence maximization algorithm can compute a solution for a graph of this size on a 16 GB RAM laptop.

Keywords: Algorithms, Social computing, Approximate computing, Probabilistic computing, Performance analysis

1. INTRODUCTION

An actively researched problem in graph structure discovery is the problem of *influence maximization* (IM): in an arbitrary graph, given an integer k, find a subset S of k nodes that maximizes some *influence function*. A commonly used influence function is *reachability* [9, 26, 13]: the network is modelled as a directed graph where entities correspond to nodes; a node influence is calculated as the number of other nodes reachable from it. For a probabilistic graph, where edge existence is not determined, but probable, it is *probabilistic* reachability: to reach the neighbouring nodes, IM algorithms select each edge with a given probability. Then a node influence is defined as the expected number of reachable nodes.

Kempe *et al.* [13] formalized several IM models, including the *Independent Cascade* (IC) model [9]: IM starts with a sampling of a probabilistic graph performing Breadth-First-Search or Depth-First-Search from a randomly selected node, proceeding at each step with the given probabilities of graph edges, to get a list of reached nodes. Kempe *et al.* showed that IM on the IC model is *monotone* and *submodular*, and the use of a Greedy algorithm for finding the most influential nodes produces good quality solutions. They also proved that the IM problem is NP-hard, but can be approximated to within a factor of $(1 - 1/e - \epsilon)$ for any $\epsilon > 0$, in polynomial time, via a Greedy hill-climbing method ([13, 16, 15]).

In 2014, a different method for IM on probabilistic graphs was proposed by Borgs *et al.* [3]: the Reverse Influence Sampling (RIS) method[3]. The idea is to select a node v uniformly at random, with replacement, and determine the set of nodes that *would have influenced* v. This can be done by simulating the influence process using the IC model on the graph with the directions of edges reversed (*transpose* graph). If a certain node u appears often in samples from different randomly selected nodes, then u is a good candidate for a most influential node. Comparing to Greedy on Monte Carlo

[3]The latest version 5 of the paper, issued on the 22nd of June 2016, can be found at `https://arxiv.org/pdf/1212.0884.pdf`

simulations, RIS can be much faster, obtaining an approximation factor of $(1 - 1/e - \epsilon)$, for any $\epsilon > 0$, in time $O((m+n)k\epsilon^{-2} \log n)$, where n is the number of nodes, m is the number of edges, and k is the number of seeds [3].

Substantial research has been done on developing approximation algorithms for IM, both heuristic and with a theoretical guarantee: [13, 5, 4, 10, 12, 8, 6, 22, 32, 31, 23, 14, 18, 28], to name a few. However, in spite of ingenious design and successful implementation of the above algorithms, the achieved scalability is not enough for the modern massive networks ([1]).

All of the approximation methods, including RIS, require a large number of graph samples. Keeping the sampling results in main memory for the consequent calculation of the most influential nodes consumes vast amounts of memory resources. There are several algorithms that propose tighter bounds than RIS on the number of samples to be taken (for example, [31, 20, 11, 19, 17] and references therein). In this paper, we focus on the Borgs *et al.* Reverse Influence Sampling (RIS) family of algorithms. We approach the IM complexity from a different angle: we concentrate on minimizing the memory footprint of RIS-based IM algorithms instead of trying to cut down on the number of samples. That is, we are managing the *space complexity*.

We note that we are not claiming that sampling algorithms not in the RIS family are better or worse that those in the RIS family. This is orthogonal to our focus in this paper. Our approach and engineering techniques may as well be used for space savings while using different methods of sampling than RIS. For our future research, we plan to apply our space-saving approach to other algorithms to further increase their scalability.

In 2017 - 2018, Tang *et al.* researched a hop-based approach to approximate the IM solution [29, 30]. They modelled the one-hop and two-hop spreads, and compared the resulting sets of seeds with other algorithms. Tang *et al.* main focus was the time complexity, but a by-product of the design was a significant savings in required space. However, this work does not belong to the RIS-based family of algorithms and as such does not provide the same set of guarantees as those based on RIS. Again, the focus of our paper is not to advocate for one or the other approach, but to provide space-savings techniques for the RIS-based family of algorithms.

To the best of our knowledge, our research is the first one focusing on data structures for space-efficient IM algorithms. We designed several data structures for storing the sampling results [24, 25], tested them on real-world graphs, and selected the most space-efficient data structure, Webgraph framework [2], for our IM algorithms. Both the input graph and the intermediate results (samples) of the IM computation are compressed. Compression helps to process large graphs on a consumer-grade machine: it gives us up to one order of magnitude saving of space.

This paper proposes a CutTheTail (CTT) concept that works by separating the samples into important ones that must be saved and others that can be ignored. It is a further development of the same idea we researched in our previous papers [24, 25]: to minimize the memory footprint in order to increase the scalability without sacrificing the quality. Our algorithms allow space savings up to five orders of magnitude compared with other IM algorithms. For example, tests in [25] performed on DIM [23] and D-SSA [20] show that the memory required by these algorithms for processing a large graph is on the order of terabytes, while our IM algorithms process the same graph on a laptop with 16 GB of RAM. We design two variants of CTT:

1. CTT1 which saves samples containing the nodes with a larger number of outgoing connections; and
2. CTT2 which saves the longer samples.

The main contributions of this paper are as follows:

1. We develop a new approach to IM complexity: minimizing the memory footprint of IM algorithms, we significantly increase the IM scalability.
2. We design and implement new accurate and space-efficient IM algorithms: CTT1 and CTT2.
3. We present experiments on eleven different large real-world graphs, conducted on a consumer-grade laptop, with an analysis of the results.

Details of CTT1 and CTT2 design can be found in Sections 4 and 5. The source code is published on GitHub: https://github.com/dianapopova/InfluenceMax.

2. PRELIMINARIES

Let $G = (V, E, p)$ be a directed probabilistic graph, where the number of nodes $|V| = n$, the size of the edge set $|E| = m$, and $p : E \to [0, 1]$ is a probability function on edge existence. In this paper, we consider the case where p is constant, i.e., for some constant number $c < 1$, it holds that $p(e) = c$ for all $e \in E$. IM researchers have often used a constant probability p. For example, Jung *et al.* [12] use the "trivalency" model where p is randomly assigned a value from the set (0.1, 0.01, 0.001). We are using a larger set of p values: (0.1, 0.05, 0.01, 0.005, 0.001). Using p as a constant allows to research information diffusion under different conditions by varying p. This is especially advantageous while experimenting with social networks where degrees of influence of one node over the others is often unknown or highly volatile.

2.1. Independent Cascade

The Independent Cascade (IC) model [9]: Starting from a set $S \subseteq V$ of *seeds* (initial nodes), IC selects incident

edges with *independent* probabilities. That is, each edge is selected or not selected independently of the previous trials. It is implemented by generating a random number in the closed interval $[0, 1]$ and comparing it with the given probability of the edge. If an edge is selected (considered existing), the node at the other end of the edge is influenced. Influenced nodes, in their turn, have a possibility to influence their neighbours forming a *cascade* of influence propagation. Hence, the name – Independent Cascade. The *influence spread* of a seed set S, denoted by $\sigma(S)$, is defined as the expected total number of influenced nodes for S.

2.2. IM Problem

PROBLEM 1 (Influence Maximization Problem (IM)). Given a graph $G = (V, E, p)$ and an integer k, find a node set $S \subseteq V$ of size k that maximizes the influence spread $\sigma(S)$.

2.3. Reversed Influence Sampling Method

In the Reverse Influence Sampling (RIS) method, Borgs *et al.* [3] suggested the following sampling process: select (uniformly at random, with replacement) a node and find a set of nodes that *would have influenced* it. If a node appears in samples often, then this node is a good candidate for a most influential node in the graph.

To find the "influencers", Borgs *et al.* propose to repeatedly run a graph search, starting from a randomly selected node, on the *transpose* (with the directions of edges reversed) graph. The resulting list of nodes reached by a search is called a *sketch*. The algorithm creates a structure that Borgs *et al.* called a *hypergraph* that stores the information derived from the sketches. We are using the term "hypergraph" throughout this paper. In order to determine the time at which to stop sampling, Borgs *et al.* calculate the target *weight* of the hypergraph, R. Borgs *et al.* proved that when the hypergraph weight is calculated by their formulae, the resulting IM solution is a guaranteed approximation to optimal: [3, Theorem 3.1] and [3, Theorem 4.1]. To find the most influential nodes, a Greedy algorithm is run on the hypergraph. The most influential node is the one that participated in the most sketches.

Theoretically, RIS achieves a near-linear time and space complexity. Despite these strong theoretical results, RIS and its implementations suffer from a practical inefficiency: the hypergraph has to be kept in main memory during the build and for the calculation of seeds. The hypergraph is large and the space needed for keeping the RIS hypergraph makes it impossible to run RIS on a consumer-grade machine for large graphs.

3. DATA STRUCTURES FOR IM

One of the recent algorithms implementing RIS is DIM [23]. Essentially (though with interesting shortcuts and improvements), DIM implements the hypergraph as a list of lists following a suggestion by Borgs *et al.* [3]. Figure 1 left, shows the DIM data structure. We designed a more space-efficient

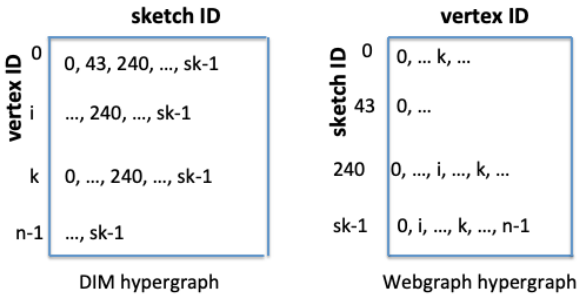


FIGURE 1: Comparing hypergraph structures

data structure, a Webgraph hypergraph [24, 25]. The Webgraph framework [2] is based on a compression technique that decreases the graph size down to 10% of its edge list size. It also includes multiple algorithms implemented in Java for a quick and easy manipulation of compressed graphs.

Let us compare how hypergraphs are built and stored using DIM data structure vs. Webgraph data structure. DIM hypergraph is built as a list of graph nodes, from 0 through $n - 1$. Each node has a corresponding list of sketch IDs. After each sample, the resulting sketch ID is added to the lists of all nodes participating in the sketch. At the end of the sampling process, the most influential nodes will have the longest lists of sketch IDs. Note that the whole DIM hypergraph must be available for a random access, in order to update the hypergraph with each new sketch.

Webgraph hypergraph building process saves the sketches appending them, one by one, to a stack of sketches, in no particular order: as long as each sketch ID is unique, the order is irrelevant. Note that the building process does not require a random access to the hypergraph. This makes the building process simpler, quicker, and easily parallelized: each machine core takes samples and stacks them; when the sampling is finished, the core stacks are merged together.

After its build is completed, DIM hypergraph resides in main memory for the seed calculation. The hypergraph is wiped out when the seed calculation is completed. This read-once hypergraph is computationally expensive; can we make the cost-per-usage lower? Our solution is to store the hypergraph on a secondary memory medium, with a possibility to load the hypergraph into main memory for seed calculation (the load takes only few seconds for billion-size graphs).The hypergraph in main memory will be wiped out after the seed calculation, but it persists on the disk. We can re-use the hypergraph for, for example, computing the influence spread of different sets of seeds.

Algorithm 1 BuildHypergraph

Input: directed graph G, weight R, int *node_tail*
Output: hypergraph H

```
1:  sk_num ← 0
2:  while H_weight < R do
3:      v ← random node of G^T
4:      sk ← BFS in G^T starting from v
5:      sk_num = sk_num + 1
6:      sk_deg ← 0
7:      sk_outdeg ← 0
8:      for each u ∈ sk do
9:          node_cover[u] ← node_cover[u] + 1
10:         sk_deg ← sk_deg + G^T.outdeg(u)
11:         sk_outdeg ← sk_outdeg + G.outdeg(u)
12:     if sk_cardinality > 1 & sk_outdeg > node_tail
    then
13:         append sk to hypergraph H
14:     H_weight ← H_weight + sk_deg
    return H
```

4. CUT_THE_TAIL1 ALGORITHM

The logic of CutTheTail1 (CTT1) algorithm is based on the following assumption:

ASSUMPTION 1. 80% of the nodes that have lower out-degrees will never become seeds.

In other words, we assume that top 20% of nodes is the pool where we will find all the seeds. This is an application of the Pareto principle to the out-degree of nodes in the **original graph** (with the original edge directions). The Pareto principle works well for many phenomena, e.g., PageRank calculation, and experimental results in Section 6 show that it works for IM as well.

4.1. CTT1 Hypergraph

An out-degree of a node is the number of edges going out to other nodes. For an undirected graph, the out-degree is equal to the in-degree, as each undirected edge is replaced by two edges pointing in the opposite directions. Calculation of the 80th percentile for the node out-degrees in the original graph is done before the hypergraph build starts. The 80th percentile is stored into an integer *node_tail*. The *BuildHypergraph* procedure (Algorithm 1) gets *node_tail* as an input parameter.

For each sketch, the *BuildHypergraph* sums up the out-degrees of all the nodes participating in the sketch, calculating a *combined out-degree* of the sketch. The sketches with the combined out-degree at less than the 80th percentile are not saved. Imagine a list of sketches sorted by the combined out-degrees. CTT1 is cutting off a long "tail" of sketches with low combined out-degrees; this is why we named the algorithm "CutTheTail". Additionally, CTT1 does not save sketches containing only one node, regardless of the node out-degree.

While creating CTT1 hypergraph, the *BuildHypergraph* procedure also stores the number of sketches for each node in an integer array *node_cover* (line 9 in Algorithm 1). The *node_cover* index corresponds to a node ID, and the stored integer is equal to the number of sketches the node participated in; we call this number "count". After the build is completed, *node_cover* is saved on the secondary memory along with the CTT1 hypergraph.

4.2. CTT1 Seeds Computing

The seed calculation starts with loading the hypergraph and the *node_cover* array into main memory and finding a largest count. The corresponding node is the first seed. Its influence is calculated (line 6) by the formula provided by Borgs *et al.* [3]. After finding the first seed, the *node_cover* array and the hypergraph must be updated, to avoid the overlapping of influenced nodes.

Algorithm 2 shows how to do it. The hypergraph (line 7) is scanned, and after the first seed in a sketch is found (line 8), the counts in *node_cover* are decreased for all the nodes in the sketch (lines 9 - 10). Then the sketch is removed from the hypergraph (line 11) and the count for the first seed is set to zero (line 12). To find the next seed, the algorithm finds a node with a largest count in the updated *node_cover* array (line 4), adds it to the set of seeds (line 5), and calculates its influence (line 6). The process is repeated until k seeds are found.

Algorithm 2 GetSeeds

Input: hypergraph H, array *node_cover*, number of seeds k
Output: seeds S, set influence $\sigma(S)$

```
1:  S ← ∅
2:  σ(S) ← 0
3:  for i = 1, ..., k do
4:      v_i ← argmax_v{node_cover[v]}
5:      S.insert(v_i)
6:      σ(S) ← σ(S) + node_cover[v_i] * n/sk_num
7:      scan H
8:      if v_i ∈ sk_j then
9:          for each u ∈ sk_j do
10:             node_cover[u] ← node_cover[u] − 1
11:         remove sk_j from H
12:     node_cover[v_i] ← 0
    return S, σ(S)
```

4.3. Analysis of CTT1

Quality. An important question to answer is "How good is the quality of CTT1 solution for IM problem?". How cutting off a potentially large number of sketches affects it? First of all, dropping all the sketches

containing only one node regardless of this node's out-degree, does not affect the solution. [25, Theorem 5.1] proves that.

Furthermore, the first seed is always calculated correctly, holding the RIS theoretical guarantee of its approximation to optimal, because CTT1 finds the first seed and calculates its spread following the RIS method exactly: the *node_cover* array contains the number of sketches each node participated in. This includes all the sketches, saved or dropped.

To find the second seed, CTT1 needs to

1. remove from the hypergraph all the sketches the first seed participated in, and
2. lower the counts for all the other nodes participating in the same sketches.

To satisfy (1), all the sketches the first seed participated in are removed and if there are some sketches with the first seed that were not saved, they can be considered "removed" as well.

It is more difficult with (2): if there happened to be a sketch with the first seed that was not saved, and if the sketch included other nodes, the counts for all these nodes cannot be updated. Consequently, the counts for all the nodes in dropped sketches with the first seed will become inflated. This makes it possible to incorrectly pick up the next seed(s): the count of a node was inflated, and it might, erroneously, become the largest in the *node_cover* array.

What is the probability of CTT1 not saving a sketch containing the first seed? It could happen only if Assumption 1 is wrong, and the first seed is at a lower than the 80th percentile of out-degrees.

The same reasoning can be applied to all found seeds: CTT1 might get an inaccurate result only if

1. the previously found seed is at a lower than 80th out-degree percentile, and consequently,
2. the *node_cover* update became inaccurate: the counts for the other nodes in dropped sketches were not decreased.

Is it possible that Assumption 1 is wrong? Well, we can imagine a node influencing just one other node, but enjoying an enormous influence overall. This can happen if that other node has huge influence over many. We all heard about "a shadow behind the throne": a person influencing just a king (or just a queen, or, in modern times, just a president) and making history. How often does it happen in an arbitrary graph? We conducted hundreds of tests on over a dozen graphs of different types, modelling different phenomena, and all the results show a high IM solution quality of CTT1 algorithm. Details are provided in subsection 6.1.3.

Space. CTT1 requires loading the original graph into main memory and keeping it there, along with the transpose graph, for the duration of the hypergraph build. It increases the pressure on the main memory compared to DIM [23] or NoSingles [25] algorithms where we need only the transpose graph. But CTT1 hypergraph is smaller as it contains only samples with high out-degree nodes. Subsection 6.5 show that this saving is more than enough to cover the additional memory for the original graph.

Time. Time complexity was not a primary concern for this research, as we propose to build the Webgraph hypergraph once and use it multiple times. Still, we analyzed the CTT1 runtime. CTT1 requires additional computations:

1. calculate the 80th out-degree percentile , and
2. compute a combined out-degree for each sketch.

Calculation time for (1) is negligible compared to the sampling time, but (2) might noticeably increase the runtime of CTT1. On the other hand, compared to other IM algorithms, CTT1 takes less time saving sketches (because it saves a smaller number of sketches), and CTT1 creates a smaller hypergraph, so that computing seeds takes less time. Testing shows that CTT1 is faster than NoSingles algorithm and much faster than DIM (subsection 6.6).

5. CUT_THE_TAIL2 ALGORITHM

CutTheTail2 (CTT2) algorithm logic is based on the following assumption:

ASSUMPTION 2. Sketches with relatively few nodes can be ignored when calculating the seeds.

5.1. CTT2 definition of "tail"

CTT2 saves all sketches with cardinality (the number of participating nodes) exceeding the *sk_tail* value. The value of *sk_tail* is calculated by an additional sampling of the graph before the hypergraph build starts. After multiple tests on many different graphs, we came up with an empirical formula for the number of samples for the *sk_tail* calculation:

$$sk\_limit = max(n/100, 10000) \quad (1)$$

where n is the number of nodes in the graph.
From each sample, we save the cardinality of the resulting sketch. After taking *sk_limit* samples, we find the longest sketch; its cardinality is saved in an integer, *max_card*. The value of *sk_tail* is calculated by the following formula:

$$sk\_tail = max(min(0.1 * max\_card/\log k, 100), 2) \quad (2)$$

where k is the number of seeds to compute.
The formula establishes the range of *sk_tail* values. Algorithm CTT2
(1) drops all the sketches containing 1 or 2 nodes, and
(2) saves all the sketches containing more than 100 nodes
for any graph, regardless of the initial sampling results.

For a particular graph, *sk_tail* can get a value between 2 and 100. The value depends on k: the error (in the spread estimate) might get larger the more seeds CTT2 computes.

5.2. CTT2 hypergraph

Algorithm 3 BuildHypergraph2

Input: directed graph G, weight R, int *sk_tail*
Output: hypergraph H

```
1:  sk_num ← 0
2:  while H_weight < R do
3:      v ← random node of G^T
4:      sk ← BFS in G^T starting from v
5:      sk_num = sk_num + 1
6:      sk_deg ← 0
7:      for each u ∈ sk do
8:          node_cover[u] ← node_cover[u] + 1
9:          sk_deg ← sk_deg + G^T.outdeg(u)
10:     if sk_cardinality > sk_tail then
11:         append sk to hypergraph H
12:     H_weight ← H_weight + sk_deg
    return H
```

Algorithm 3 shows that CTT2 hypergraph build is similar to CTT1. However, CTT2 does not have to calculate the combined out-degree in the original graph for each sketch, saving time and space.

5.3. CTT2 Seeds Computing

CTT2 computes the seeds exactly as CTT1 does (Algorithm 2):

1. finds a largest value in the *node_cover* array,
2. adds the corresponding node to the seed set S,
3. scans the hypergraph, finds all the sketches the newly found seed participated in, and
4. updates *node_cover* and the hypergraph accordingly.

(3) and (4) are done only if S does not contain k seeds yet.

5.4. Analysis of CTT2

Quality. CTT2 drops the sketches that contain fewer than *sk_tail* nodes. The dropped sketches might include influential nodes. The first seed is always selected correctly, as the *node_cover* array keeps counts of all the sketches including those that were not saved. Using the *node_cover* array with the counts of **all** sketches is crucial: in spite of the fact that CTT1 and CTT2 drop a large number of sketches, node counts reflect the sampling in its entirety. The guaranteed approximation to optimal by Borgs *et al.* [3] holds for the first seed.

To find the next seed, CTT2 updates the *node_cover* array decreasing the counts for all the nodes that participated in the sketches with the selected seed. It is possible that some of the sketches with the selected seed were not saved; then, after the update, all the nodes participating in them will have their counts inflated. It might lead to selecting as the next seed a node with lower real count than another node that did not participate in the dropped sketches. It is clear that a possible error is accumulating as the number of seeds increases. The intuition underlining our formula for calculating *sk_tail* is that highly influential nodes, as a rule, participate in longer sketches. The small number of short sketches with the highly influential nodes will not lead to a big error. Subsection 6.1 shows test results that support this intuition.

Space. CTT2 requires additional initial sampling before the hypergraph creation. Results (the number of nodes in each sketch) are saved in an integer array. The memory taken by this array is very small, and it is released immediately after calculating *max_card*. CTT2 takes less memory than DIM [23], NoSingles [25], or CTT1, for all tested graphs (subsection 6.5).

Time. CTT2 requires additional computations:

1. calculate the *sk_tail* , and
2. compare the number of nodes in each sketch with *sk_tail*.

Calculation time for (2) is negligible compared with the sampling time, but calculating *sk_tail* requires taking additional 10000 or more $(n/100)$ samples before the hypergraph build. Subsection 6.6 shows that the time increase could be noticeable only for small graphs, where 10000 additional samples take the time comparable with the hypergraph build. On the other hand, CTT2 takes less time saving the sketches (because it saves a smaller number of sketches), and CTT2 creates a smaller hypergraph, where computing seeds takes less time. Tests (subsection 6.6) show that CTT2 is faster than other IM algorithms.

However, there are cases when we recommend using CTT1: (a) CTT2 gives a larger error than CTT1 for a large number of seeds. The inflation of spread estimation is potentially increasing with each selected seed. We recommend using CTT1 when more than 25 seeds are needed.
(b) CTT2 can be slower than other algorithms for $p >= 0.1$ and dense graphs, because CTT2 has to take at least 10,000 additional samples for calculating the *sk_tail*. For the dense graphs with high probability on edges (for example, protein-to-protein interaction, where p can be 0.5 – 0.9) we recommend using CTT1.

6. EXPERIMENTAL RESULTS

While any formula for the number of samples can be used by our algorithms, we have been using Borgs *et al.* formula from [3, Theorem 3.1]:

$$R = (c * m * k * \epsilon^{-2} * \log(n)) \quad (3)$$

Dataset	n	m	type
WordAsn	10.6K	72K	association, directed
Caida	65.5K	106.7K	social, directed
FB	4K	176K	social, undirected
EnronD	69K	275K	e-mails, directed
Enron	36.7K	368K	e-mails, undirected
Deezer	54.6K	996K	social, undirected
DBLP2010	326 K	1.6 M	collaboration, undirected
UK100K	100 K	3 M	web, directed
CNR2000	326 K	3.2 M	web, directed
DBLP2011	986K	6.7M	collaboration, undirected
Arabic2005	23M	640M	web, directed

TABLE 1: Test datasets ordered by m.

where

$$c = 4.0 * (1 + \epsilon) * (1 + 1/k) \quad (4)$$

m is the number of graph edges, k is the number of seeds, and n is the number of graph nodes.

This is done for two reasons: (1) the formula guarantees an approximation to the optimal solution that can be explicitly calculated[4], and (2) for consistency, when evaluating the scalability of our algorithms.

We implemented the algorithms in Java 8 and used Webgraph [2] as a graph compression framework.

Datasets. The graphs were downloaded from [27], and [7]. We purposefully picked up graphs of different types (Table 1), to thoroughly test our algorithms' performance.

Equipment. All the experiments were conducted on a laptop with processor 2.2 GHz Intel Core i7 (4-core), RAM 16GB 1600 MHz DDR3, running OS X Yosemite.

The experiments could be divided into two categories: evaluating **the quality** of the calculated IM solutions by comparing the results with other IM algorithms; and **the scalability** of the proposed algorithms.

6.1. Quality of IM Solution

To evaluate the quality of IM solutions calculated by CTT1 and CTT2, the following assessments were made:
(1) confidence in the approximation;
(2) accuracy of spread estimation;
(3) influence spread comparison.

6.1.1. Confidence in the approximation

The first question we would like to answer is: "How much confidence can we have in the IM solution when using Borgs *et al.* formula 3?" In [3], Borgs *et al.* explain that, theoretically, the confidence in IM solution computed by the RIS method, is "at least 3/5". It is just 60% confidence, because there is a danger that, by chance, RIS will pick up a dense part of the graph and reach the required weight of the hypergraph with too small number of samples. Then the sampling will not adequately represent the graph structure, and IM solution will be a poor approximation to optimal. Borgs *et al.* advised to run RIS $\log(n)$ times and pick up the largest number of samples taken, for the confidence to rise from 3/5 to $(1 - 1/n)$. We did a number of tests on different graphs, varying the probability of edge existence, to see how much the number of samples is changing from run to run.

p	samples taken	mean	St. Dev	St.Dev/mean
0.1	504,244 506,257 503,812 505,355 500,904 504,016 504,385 502,700 504,063 498,966 501,107 504,427 506,203 502,782 504,890 503,593 504,357	503,651	1,899	0.38%
0.001	2,319,933,031 2,319,182,149 2,318,487,182 2,320,343,923 2,319,352,759 2,320,169,958 2,319,320,032 2,318,046,484 2,319,662,355 2,318,291,684 2,319,869,627 2,318,774,277 2,320,434,972 2,319,981,483 2,318,148,705 2,318,818,136 2,318,574,922	2,319,258,334	792611	0.03%

TABLE 2: UK100K: Samples taken by $\log(n) = 17$ runs.

The tests show that the number of samples taken by our algorithms varies negligibly. After hundreds and hundreds of tests, we never encountered a situation where the difference in the number of samples appeared significant; it is always well below 1%. Table 2 presents the results for UK100K graph. We got similar results for other graphs and probabilities. From Table 2, we can also see that with probability of edge existence p decreasing, the difference in the sample numbers is decreasing too. We came to the conclusion that, for the real-world graphs, especially social networks, the

[4]Borgs *et al.* proved that this formula guarantees the approximation factor of $(1 - 1/e - \epsilon)$, for any $\epsilon > 0$, with the confidence of at least 3/5.

Graph	k	p	spread CTT	spread MC	spread CTT2	spread MC
UK100K	5	0.1	15,078.54	14,988.20	14,920.17	14,984.00
		0.05	4,158.83	4,138.54	4,131.67	4,133.76
		0.01	166.35	166.04	166.57	166.04
		0.005	61.76	61.56	61.67	61.56
		0.001	15.17	15.19	15.16	15.19
UK100K	10	0.1	16,617.89	16,617.00	16,639.59	16,619.00
		0.05	5,130.09	5,136.73	5,145.54	5,141.64
		0.01	250.30	250.34	250.75	250.00
		0.005	90.00	90.11	90.08	90.11
		0.001	24.32	24.34	24.33	24.34
DBLP	5	0.1	25,998.84	25,989.80	26,126.97	25,859.80
		0.05	3,970.59	3,970.13	3,964.74	3,909.11
		0.01	27.80	27.78	27.85	27.78
		0.005	12.61	12.63	12.61	12.62
		0.001	6.22	6.21	6.21	6.21
DBLP	10	0.1	26,160.32	26,126.80	26,231.23	25,862.90
		0.05	4,194.50	4,179.77	4,282.92	3,969.08
		0.01	46.51	46.43	46.51	46.43
		0.005	22.64	22.65	22.64	22.65
		0.001	12.06	12.08	12.09	12.07

TABLE 3: Comparison of Spread Estimations.

confidence in IM solution is higher than theoretically proven 60% and approaches 100%.

6.1.2. Accuracy of Spread Estimation

The accurate estimation of influence spread (the estimated number of reached vertices by a set of seeds) is an important factor in ensuring the high quality of IM solution.

Benchmark As a benchmark, we use Monte Carlo simulation of influence spread in the original graph. The benchmark takes as an input two text files: the graph edge list, and the list of seeds. The output of the benchmark is the estimation of information spread. We tuned the benchmark for 20,000 simulations for each estimation.

Spread Estimation by CTT1, CTT2 *vs.* Benhmark Table 3 shows the comparison of spread estimations calculated by our algorithms with the ones calculated by the benchmark, for the same seed sets computed by CTT1 or CTT2. We tested the estimations for different edge probabilities and different graphs. We present the results for two graphs of different types and two probabilities p, to save the paper space. The table displays the spread calculated by CTT1 and CTT2 side by side with the benchmark spread ("spread MC" in the heading). The presented results are very similar to all other tested graphs: spread estimation by CTT1 is practically identical to the ones by the benchmark; spread estimation by CTT2, in some cases, gets slightly inflated. In our testing, the largest difference of spread estimation by CTT2 and the benchmark was under 7%.

6.1.3. Influence Spread Comparison

DIM code was downloaded from GitHub [21] and tuned to compute IM solutions using Formula 3. The tuning was reviewed and approved by the DIM author, Dr. Naoto Ohsaka. The large amount of memory required for a successful run of DIM is the reason for presenting only two smaller graphs in that subsection (the larger one, FB, has $m = 176$ K). These were the only graphs DIM computed IM solution for. We tried to test other graphs, but 16 GB memory of our laptop was not enough: DIM ran out of memory.

To compare the quality of the selected seeds, we assessed the value of the resulting influence spread using Monte Carlo simulation. Figure 2 presents the results of testing CTT1 *vs.* CTT2 *vs.* DIM [23] *vs.* NoSingles [25] algorithms. The quality of IM solutions by DIM, NoSingles, CTT1, and CTT2 is so close, that we present, instead of the charts, the actual values of spread (Figure 2).

NoSingles, TopDegree. To test the quality of CTT algorithm solutions on larger graphs, we compared the spreads of CTT1 and CTT2 seeds with the spread of NoSingles seeds. NoSingles is proven ([25, Theorem 5.1]) to hold Borgs *et al.* theoretical guarantee for the approximation to optimal, so it can serve as a yardstick. For a fair comparison, we calculated all the spreads using our benchmark, Monte Carlo simulation. Additionally, we tested the quality of a simple IM solution that is often used by marketing teams: pick up the top out-degree nodes in the network as the seeds. We found the top-5 and top-10 nodes in each of the tested graphs and used the benchmark to calculate their spread.

The charts in Figure 3 show the spreads achieved by NoSingles (NS), CutTheTail1 (CTT1), CutTheTail2 (CTT2) algorithms, and Top Degree (TopDeg) nodes. Analysis of the test results shows that the quality of CTT1 and CTT2 solutions is very close to the quality of NoSingles solution for all tested graphs. The quality of Top Degree solutions varies. Top Degree spread is never larger than the spread achieved by our algorithms, and often smaller. The largest difference was detected while testing Caida graph: Top Degree spread reached only 33% of the number of nodes reached by the seeds computed by NoSingles, CTT1, or CTT2 (Figure 3a, $p = 0.005$). This result shows that using Top Degree solution can lead to a poor quality result. Moreover, the marketing team would not realize that the result was only a fraction of what it might have been, had they used an IM algorithm for calculating the influential nodes.

6.2. Scalability

The quality of an IM solution is the most important characteristic of IM algorithms, but the scalability is what defines the utilitarian possibilities in practical scenarios. Imagine a campaign manager acquiring a piece of positive information about her candidate (or a piece of negative information about a rival). With a model of the constituency in graph form, she can calculate the most influential individuals to share the information. If it is possible to do this on a consumer-grade laptop, it makes the usage of a scalable IM algorithm both practical and desirable.

A fair evaluation of the scalability of IM algorithms is complex:

(1) The algorithms might use different sets of input

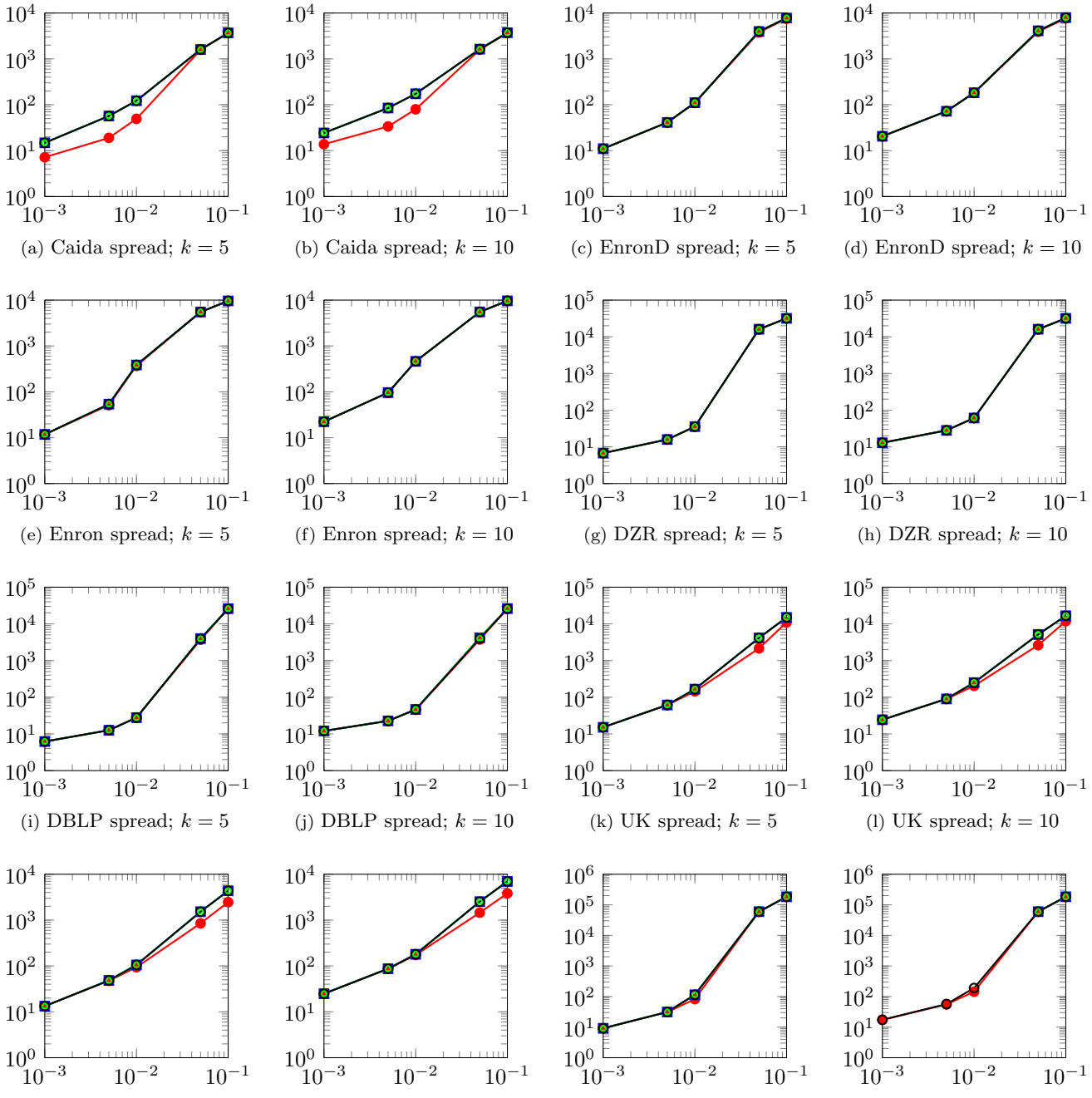
p	spread NS	spread CTT	spread CTT2	spread DIM
0.1	689.256	689.713	687.236	690.172
0.05	28.7223	28.6297	28.5145	28.7223
0.01	6.8527	6.83805	6.85405	6.8534
0.005	5.85685	5.8395	5.84275	5.8584
0.001	5.13855	5.1189	5.13525	5.14325

(a) WordAsn spreads; $k = 5$.

p	spread NS	spread CTT	spread CTT2	spread DIM
0.1	3055.5	3055.15	3055.36	3055.44
0.05	2202.17	2202.17	2202.55	2201.59
0.01	269.85	269.85	269.85	269.85
0.005	35.2356	35.2356	35.2356	35.2356
0.001	8.66075	8.6597	8.6473	8.66075

(b) FB spreads; $k = 5$.

FIGURE 2: Quality: DIM, CTT1, CTT2, and NoSingles.



(a) Caida spread; $k = 5$ (b) Caida spread; $k = 10$ (c) EnronD spread; $k = 5$ (d) EnronD spread; $k = 10$

(e) Enron spread; $k = 5$ (f) Enron spread; $k = 10$ (g) DZR spread; $k = 5$ (h) DZR spread; $k = 10$

(i) DBLP spread; $k = 5$ (j) DBLP spread; $k = 10$ (k) UK spread; $k = 5$ (l) UK spread; $k = 10$

(m) CNR spread; $k = 5$ (n) CNR spread; $k = 10$ (o) DBLP2011 spread; $k = 5$ (p) DBLP2011 spread; $k = 10$

FIGURE 3: Information Spread of IM solution, varying p; TopDegree, NS, CTT1, and CTT2.

parameters, sometimes overlapping, sometimes disjoint. (2) The scalability achieved by the research teams that

they describe in their papers can vary significantly from the results of the replicating tests.
Tests of eleven different IM algorithms by Arora *et al.* [1] proved both points. According to [1, Table 3], under IC model the only IM algorithm that successfully completed on a 1.5-billion-size graph, was PMC [22]. It took almost 300 GB of memory.

In [25], our team tested DIM ([23]) and D-SSA ([20]) *vs.* our NoSingles algorithm. For the number of samples calculation, we used the formula from [3, Theorem 3.1]. That formula allows to take a smaller number of samples (compared with Formula 3 used in this paper), but guarantees a rather low approximation to the optimal solution (at the most, 25 %). To the best of our knowledge, the tests conducted by other IM researchers using RIS are all using that formula, with a reduced number of samples. Even for the reduced number of samples, D-SSA and DIM required so much memory, that we had to use a machine with 1TB RAM, to successfully complete the testing for medium-size graphs.

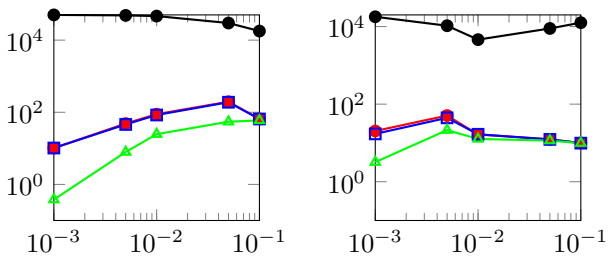
In this paper, we did all the testing on a laptop with 16 GB RAM. Furthermore, we used Formula 3 for the number of samples calculation. This formula requires a much higher number of samples, and guarantees a much higher approximation to optimal: with the parameters we used, it is 59%.

The following experiments were conducted to evaluate the scalability of CTT algorithms:
(1) comparison with DIM;
(2) comparison with TESA;
(3) required space;
(3) runtime;
(5) large graph IM solution computed on a laptop.

6.3. Comparison with DIM

6.3.1. Space Comparison

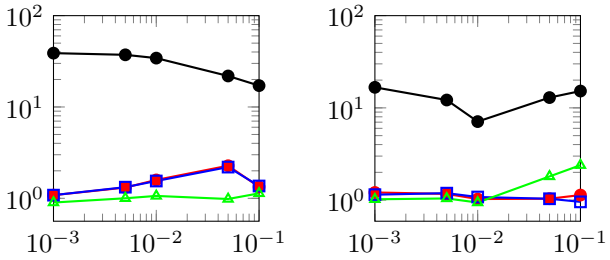
Figure 4 shows the memory required for a successful run of DIM and our algorithms. DIM needs up to five orders of magnitude more memory for keeping the intermediate results of IM computing.



(a) WordAsn; $k = 5$ (b) FB; $k = 5$

FIGURE 4: Space, MB, varying p; NS, CTT1, CTT2, and DIM.

6.3.2. Runtime Comparison



(a) WordAsn; $k = 5$ (b) FB; $k = 5$

FIGURE 5: Time, min, varying p; NS, CTT1, CTT2, and DIM.

Figure 5 shows the difference in the runtime between DIM, CTT1, CTT2, and NoSingles. Our algorithms take few minutes to output a solution, while DIM takes several times more. It must be noted that these charts are shown for information only; the comparison of DIM runtime to our algorithms' can be seen as unfair, because DIM builds hypergraph sequentially, while CTT1, CTT2, and NoSingles utilize the parallel processing: eight logical cores sampled the graphs in parallel.

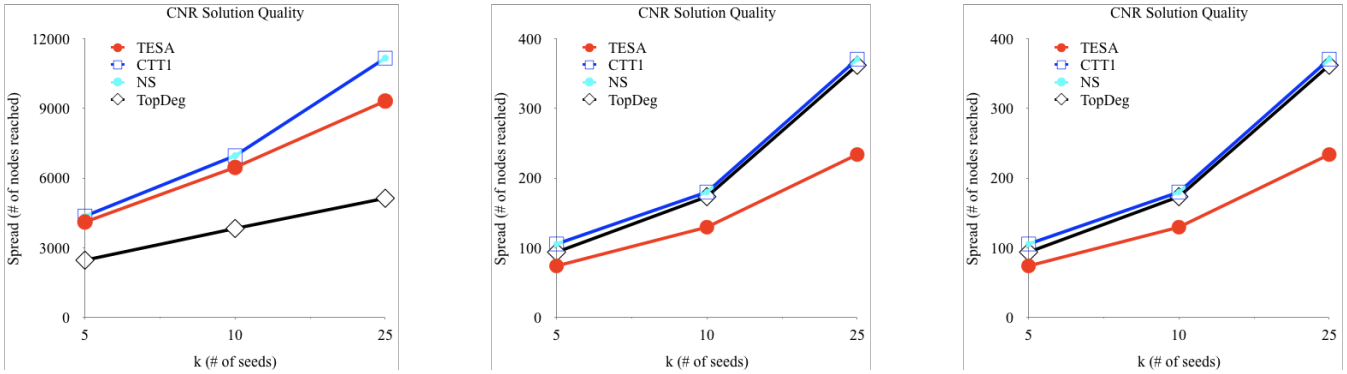
6.4. Comparison with TESA

A new IM algorithm, TESA, was published in 2019 [33]. TESA code was not made available to us. We implemented TESA as described in the paper.

1. TESA conducts an initial sorting of graph nodes by their degrees; one of the top 20% of nodes is randomly selected as the root for each sampling. This invalidates the quality guarantee proved by Borgs *et al.*
2. The samples are saved only if they include three or more nodes, regardless of the graph structure. This design further erodes the quality of IM solution.
3. TESA is sampling only and exclusively the most dense parts of a graph, and saves only the longer samples. The resulting hypergraph reflects not the orginal, but a distorted graph structure.
4. TESA paper provides no analysis of possible errors. Moreover, no experiments are evaluating the quality of TESA solutions for IM.

We conducted dozens of experiments on several graphs. Our testing shows that TESA is providing solutions that are always worse than our algorithms' solutions and, in many cases, worse than the top-degree solutions.

Fig. 6 demonstrates the solution quality of TESA compared to our algorithms NoSingles and CTT1. The spreads achieved by the seeds were calculated by Monte Carlo simulation, for a fair comparison. Note, that



(a) CNR spreads; $p = 0.1$. (b) CNR spreads; $p = 0.01$. (c) CNR spreads; $p = 0.001$.

FIGURE 6: Quality: TESA, CTT1, TopDegree, and NoSingles.

NoSingles and CTT1 spreads are so close, that they appear the same on the charts.

The seeds computed by TESA, produce the spreads that are always smaller than our algorithms'. TESA disadvantage appears to increase when the number of seeds grows: for example, Fig. 6(b) shows that TESA spread is about 80% for $k = 5$; but only 60% for $k = 25$.

The poor performance of TESA is caused by a major design flaw: as TESA sampling starts not at random, but always at one of the top 20% degree nodes, the seeds are calculated not for the actual graph, but for a subgraph of most connected nodes. The key point of RIS is a random, with replacement, selection of any node in the graph. TESA sampling is conducted by a different method than RIS. Consequently, the theoretical proof of the solution quality by Borgs *at al.* is not applicable.

Additionally, we compared TESA with the TopDegree method, when top out-degree nodes are taken as seeds. TopDegree has no quality guarantee, the results depend on the graph structure. More TopDegree results, for diverse graphs, can be found in Subsection 6.1.3 and Fig 3. Our algorithms always produce a better solution than TopDegree. The black line on the Fig. 6 charts shows the TopDegree spreads. In many cases TESA (the red line) computed a worse IM solution than TopDegree.

6.5. Required Space

To thoroughly test the scalability of CTT1 and CTT2, we conducted experiments on large, different types graphs (Table 1).

Figure 7 shows the space required by CTT1, CTT2, and NoSingles. For all tested graphs, NoSingles takes the most space to store IM intermediate results, CTT1 follows closely, and CTT2 takes less space. Dynamics of space requirement vary from graph to graph. For some, with probability of edge existence p decreasing, less and less space is required for storing the hypergraph. For other graphs, the dynamics is more complex: the most space is needed when the probability $p = 0.01$ or $p = 0.05$. It depends on the graph structure and the distribution of degrees over the nodes.

6.5.1. *Sketches Saved*

Table 4 provides statistics on the number of sketches taken by our algorithms and the number of sketches saved in the hypergraph. We present this statistics for two graphs, to save the paper space, though we collected it for all the graphs we tested. It is rather surprising that, for example, CTT2 can drop 99.99% of sketches (for DBLP graph, when $p = 0.001$), and compute a good set of seeds from just 0.01% longest sketches. Note, that other RIS method implementations save all of the sketches; this explains the enormous savings in memory consumption by our algorithms.

6.6. Runtime

Figure 8 shows the time taken by CTT1, CTT2, and NoSingles for computing a seed set. As a rule, NoSingles takes the most time to complete the task, CTT1 – a little less time, and CTT2 - the least time. However, as can be seen in Figure 8g, CTT2 can be slower than other algorithms for $p = 0.1$ and dense graphs, because CTT2 has to take at least 10,000 additional samples for calculating the sk_tail, as explained in subsection 5.4.

6.7. CTT2 on Arabic-2005

We successfully ran CTT2 on a large graph, Arabic-2005 ($n = 23$ M, $m = 640$ M). We set $k = 5$, $\epsilon = 0.2$, and $p = 0.001$. Results are presented in Table 5, where R is the hypergraph weight, sk means sketch, and H means the hypergraph. Table 5 shows that CTT2 took 2.5 billion sketches, and saved 1.4 million of them, which took only 437 MB of space. This small hypergraph allowed CTT2 to calculate the seeds with an estimated spread of 8,256 nodes. Monte Carlo simulation for the seeds shows that the spread is 8,235 nodes. CTT2 estimation is inflated by 21 nodes, which

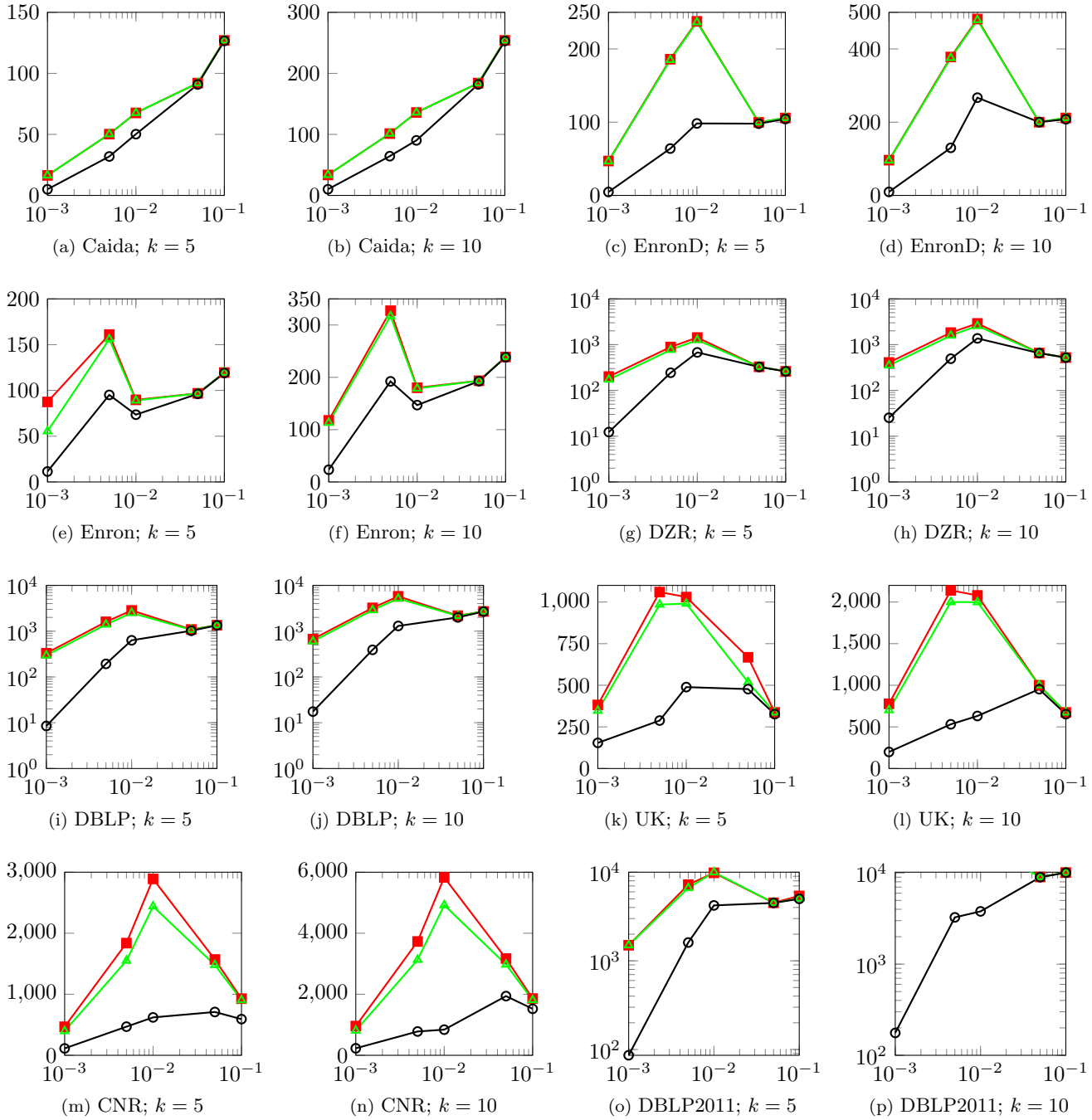


FIGURE 7: Space, MB, varying p; NS, CTT1, and CTT2.

is just 0.25% of the spread calculated by Monte Carlo simulation. This demonstrates high accuracy of CTT2 algorithm. Note that our algorithms can be modified for a tighter bound on the number of samples, with the corresponding increase in scalability.

7. CONCLUSIONS AND FURTHER RESEARCH

We present CutTheTail, a new heuristic algorithm for computing influence maximization on large graphs. CutTheTail implements a novel approach to the influence maximization problem: minimizing the memory footprint for storing influence maximization intermediate results. With two variants of this algorithm, CutTheTail1 and CutTheTail2, we were able to scale up influence maximization computation by strategically saving only a small fraction of graph sampling results (sketches). For some graphs, storing only 0.01% of sketches is enough to get a high quality influence maximization solution. The CTT memory footprint is drastically cut compared to other IM

Graph	k	p	NStaken	NSsaved	%	CTTtaken	CTTsaved	%	CTT2taken	CTT2saved	%
		0.1	504,104	172,671	34.25	500,612	161,354	32.23	500,612	161,354	32.23
		0.05	3,781,711	904,552	23.92	3,777,766	839,588	22.22	3,800,353	278,757	7.34
UK	5	0.01	616,675,480	55,864,238	9.06	616,689,630	50,704,781	8.22	615,797,837	4,102,052	0.67
		0.005	1,526,286,139	86,516,196	5.67	1,526,807,116	78,136,116	5.12	1,526,802,460	5,288,614	0.35
		0.001	2,318,758,443	41,562,254	1.79	2,318,647,899	37,168,980	1.60	2,319,550,427	12,143,373	0.52
		0.1	1,005,292	343,319	34.15	1,008,624	324,821	32.20	1,006,479	188,455	18.72
		0.05	7,575,294	1,809,956	23.89	7,622,432	1,691,236	22.19	7,542,918	553,720	7.34
UK	10	0.01	1,232,812,184	111,676,143	9.06	1,232,442,194	101,325,197	8.22	1,233,408,606	7,433,148	0.60
		0.005	3,054,992,501	173,175,531	5.67	3,055,627,402	156,372,643	5.12	3,054,695,748	8,805,332	0.29
		0.001	4,638,051,350	83,155,221	1.79	4,638,029,103	74,375,955	1.60	4,637,274,400	12,479,372	0.27
		0.1	1,284,575	402,773	31.35	1,285,569	347,623	27.04	1,280,650	101,726	7.94
		0.05	35,719,493	6,697,295	18.75	35,750,958	5,828,306	16.30	35,797,155	421,462	1.18
DBLP	5	0.01	7,345,860,286	338,513,636	4.61	7,345,906,804	299,282,101	4.07	7,345,585,104	60,801,600	0.83
		0.005	8,303,430,952	198,025,876	2.38	8,303,805,759	175,565,190	2.11	8,303,626,741	20,200,381	0.24
		0.001	8,955,580,165	44,014,278	0.49	8,955,521,972	39,116,784	0.44	8,955,785,780	1,011,315	0.01
		0.1	2,566,013	805,633	31.40	2,567,324	694,218	27.04	2,572,919	203,434	7.91
		0.05	71,593,557	13,419,844	18.74	71,328,170	11,630,416	16.31	71,470,266	846,732	1.18
DBLP	10	0.01	14,691,953,097	677,084,716	4.61	14,690,970,229	598,512,951	4.07	14,691,195,814	121,611,500	0.83
		0.005	16,607,865,682	396,039,223	2.38	16,607,673,063	351,134,963	2.11	16,607,436,867	40,398,152	0.24
		0.001	17,911,485,062	88,015,564	0.49	17,911,789,392	78,225,120	0.44	17,911,349,219	2,021,773	0.01

TABLE 4: Statistics on Sketches Saved.

R	sk_tail	sk taken	sk saved	H space	CTT2 spread	MC
6.4 T	100	2.5 B	1.4 M	437 MB	8,256	8,235

TABLE 5: CTT2 on Arabic-2005.

designs. Experiments show that our algorithms use up to five orders of magnitude smaller space than DIM [23]. The achieved scalability allows to use consumer-grade machines for processing massive graphs, which, in turn, raises the possibility of the algorithms' practical applications.

Furthermore, an algorithm can specify different conditions for saving or dropping samples as was demonstrated by CTT1 and CTT2. The resulting compressed representation of the graph contains a wealth of information that can be processed accordingly to the problem at hand. Multiple processing using different algorithms is possible, as the samples are kept on a secondary medium.

One of the possible ways of re-using the hypergraph: run CTT algorithm for as big a k as possible for the available machine memory. The stored hypergraph will provide the spreads (the number of nodes reachable by the seeds) for any k in the range [1,k]. Another possibility: in the k-centre clustering problem for the probabilistic graphs, an IM algorithm can be used for selecting the clusters' centres. Our team is working on applying the CTT concept for this problem.

High scalability of our data structure and flexibility of its utilization is a significant contribution to the massive graph discovery.

ACKNOWLEDGEMENTS

We are grateful to Dr. Naoto Ohsaka for providing the code for Monte Carlo simulation and reviewing our tuning of his algorithm, DIM [23].

REFERENCES

[1] A. Arora, S. Galhotra, and S. Ranu. Debunking the myths of influence maximization: An in-depth benchmarking study. In *Proceedings of the 43rd ACM SIGMOD International Conference on Management of Data*, pages 651–666, 2017.

[2] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *Proceedings of the 13th International Conference on World Wide Web*, pages 595–602, 2004.

[3] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 946–957, 2014.

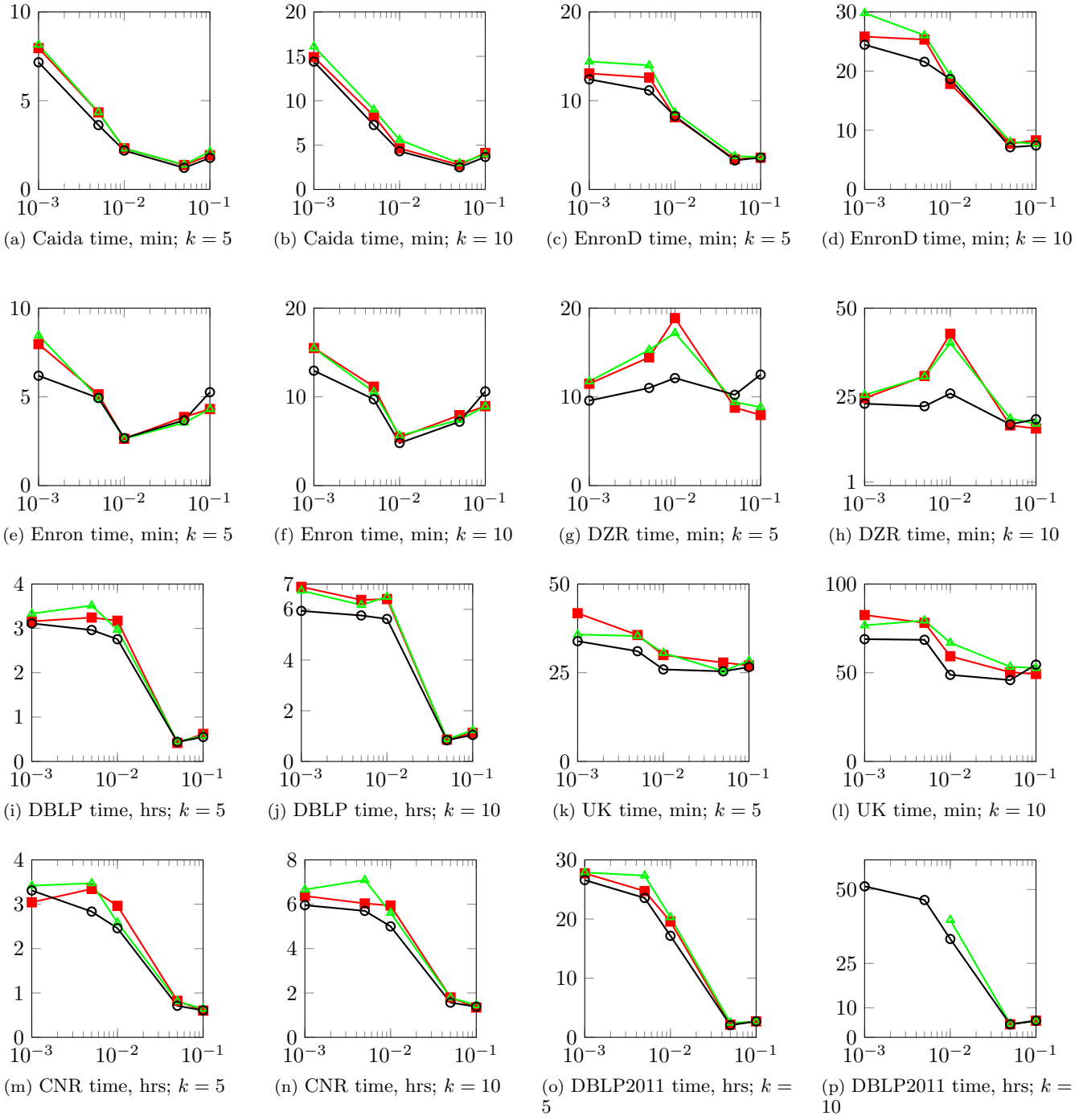
[4] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1029–1038, 2010.

[5] 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*, pages 199–208, 2009.

[6] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 629–638, 2014.

[7] Datasets. http://law.di.unimi.it/datasets.php. In *Laboratory for Web Algorithmics*, 2018.

[8] N. Du, L. Song, M. G. Rodriguez, and H. Zha. Scalable influence estimation in continuous-time diffusion



(a) Caida time, min; $k = 5$ (b) Caida time, min; $k = 10$ (c) EnronD time, min; $k = 5$ (d) EnronD time, min; $k = 10$

(e) Enron time, min; $k = 5$ (f) Enron time, min; $k = 10$ (g) DZR time, min; $k = 5$ (h) DZR time, min; $k = 10$

(i) DBLP time, hrs; $k = 5$ (j) DBLP time, hrs; $k = 10$ (k) UK time, min; $k = 5$ (l) UK time, min; $k = 10$

(m) CNR time, hrs; $k = 5$ (n) CNR time, hrs; $k = 10$ (o) DBLP2011 time, hrs; $k = 5$ (p) DBLP2011 time, hrs; $k = 10$

FIGURE 8: Runtime, varying p; NS, CTT, and CTT2.

networks. In *Proceedings of the Advances in Neural Information Processing Systems 26*, pages 3147–3155, 2013.

[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*, 12(3):211–223, 2001.

[10] A. Goyal, W. Lu, and L. Lakshmanan. CELF++: Optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th International Conference on World Wide Web*, pages 47–48, 2011.

[11] K. Huang, S. Wang, G. S. Bevilacqua, X. Xiao, and L. V. S. Lakshmanan. Revisiting the stop-and-stare algorithms for influence maximization. *PVLDB*, 10:913–924, 2017.

[12] K. Jung, W. Heo, and W. Chen. IRIE: Scalable and robust influence maximization in social networks. In *Proceedings of the 12th IEEE International Conference on Data Mining*, pages 918–923, 2012.

[13] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing

the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137–146, 2003.

[14] B. Lucier, J. Oren, and Y. Singer. Influence at scale: Distributed computation of complex contagion in networks. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 735–744, 2015.

[15] G. L. Nemhauser and L. A. Wolsey. Maximizing submodular set functions: formulations and analysis of algorithms. *Studies on Graphs and Discrete Programming*, 11:279–301, 1981.

[16] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978.

[17] H. T. Nguyen, T. N. Dinh, and M. T. Thai. Cost-aware targeted viral marketing in billion-scale networks. In *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2016.

[18] H. T. Nguyen, T. P. Nguyen, T. N. Vu, and T. N. Dinh. Importance sketching of influence dynamics in billion-scale networks. In *Proceedings of the 17th IEEE International Conference on Data Mining*, pages 337–346, 2017.

[19] H. T. Nguyen, T. P. Nguyen, T. N. Vu, and T. N. Dinh. Outward influence and cascade size estimation in billion-scale networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(1):20:1–20:30, 2017.

[20] H. T. Nguyen, M. T. Thai, and T. N. Dinh. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *Proceedings of the 42nd ACM SIGMOD International Conference on Management of Data*, pages 695–710, 2016.

[21] N. Ohsaka. https://github.com/todo314/dynamic-influence-analysis. In *GitHub*, 2017.

[22] N. Ohsaka, T. Akiba, Y. Yoshida, and K. Kawarabayashi. Fast and accurate influence maximization on large networks with pruned monte-carlo simulations. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 138–144, 2014.

[23] N. Ohsaka, T. Akiba, Y. Yoshida, and K. Kawarabayashi. Dynamic influence analysis in evolving networks. *Proceedings of the VLDB Endowment*, 9(12):1077–1088, 2016.

[24] D. Popova, A. Khot, and A. Thomo. Data structures for efficient computation of influence maximization and influence estimation. In *Proceedings of the 21st International Conference on Extending Database Technology*, ISSN: 2367-2005, pages 505–508. EDBT, OpenProceedings.org, 2018.

[25] D. Popova, N. Ohsaka, K.-i. Kawarabayashi, and A. Thomo. Nosingles: A space-efficient algorithm for influence maximization. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, SSDBM '18, pages 18:1–18:12, New York, NY, USA, 2018. ACM.

[26] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 61–70. ACM, 2002.

[27] SNAP. http://snap.stanford.edu. In *Stanford Network Analysis Project*, 2018.

[28] J. Tang, X. Tang, X. Xiao, and J. Yuan. Online processing algorithms for influence maximization. In *Proceedings of the 2018 International Conference on Management of Data*, pages 991–1005, 2018.

[29] J. Tang, X. Tang, and J. Yuan. Influence maximization meets efficiency and effectiveness: A hop-based approach. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, pages 64–71, 2017.

[30] J. Tang, X. Tang, and J. Yuan. An efficient and effective hop-based approach for influence maximization in social networks. *Social Network Analysis and Mining*, 8:1–19, 2018.

[31] Y. Tang, Y. Shi, and X. Xiao. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 41st ACM SIGMOD International Conference on Management of Data*, pages 1539–1554, 2015.

[32] Y. Tang, X. Xiao, and Y. Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proceedings of the 40th ACM SIGMOD International Conference on Management of Data*, pages 75–86, 2014.

[33] G. Xia. Influence maximization: A time-space efficient algorithm. *IOP Conference Series: Materials Science and Engineering*, 533:012048, 2019.