Sparsification of Influence Networks

Michael Mathioudakis[1]* mathiou@cs.toronto.edu
Francesco Bonchi[2] bonchi@yahoo-inc.com
Carlos Castillo[2] chato@yahoo-inc.com
Aristides Gionis[2] gionis@yahoo-inc.com
Antti Ukkonen[2] aukkonen@yahoo-inc.com

[1]Computer Science Department
University of Toronto, Canada

[2]Yahoo! Research
Barcelona, Spain

ABSTRACT

We present SPINE, an efficient algorithm for finding the "backbone" of an influence network. Given a social graph and a log of past propagations, we build an instance of the *independent-cascade model* that describes the propagations. We aim at reducing the complexity of that model, while preserving most of its accuracy in describing the data.

We show that the problem is inapproximable and we present an optimal, dynamic-programming algorithm, whose search space, albeit exponential, is typically much smaller than that of the brute force, exhaustive-search approach. Seeking a practical, scalable approach to sparsification, we devise SPINE, a greedy, efficient algorithm with practically little compromise in quality.

We claim that sparsification is a fundamental data-reduction operation with many applications, ranging from visualization to exploratory and descriptive data analysis. As a proof of concept, we use SPINE on real-world datasets, revealing the *backbone* of their influence-propagation networks. Moreover, we apply SPINE as a pre-processing step for the *influence-maximization* problem, showing that computations on sparsified models give up little accuracy, but yield significant improvements in terms of scalability.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications - *Data Mining*
General Terms: Algorithms
Keywords: Social Networks, Influence, Propagation

1. INTRODUCTION

For many scenarios of network analysis, sparsification is a fundamental data-reduction operation that equips the data analyst with the ability to visualize, explore, digest, and interpret more easily the available data. In addition, sparsification helps in reducing the noise in the data and in avoiding over-fitting, thus allowing to build more accurate models.

*Part of this work was done while the first author was visiting Yahoo! Research Barcelona, under the Yahoo! internship program.

In this paper we study a sparsification framework that is appropriate for analyzing information propagation in social networks. We aim at eliminating a large number of links in the network, and preserving only the links that play an important role on how information propagates. The importance of a link is measured by its ability to explain observed propagations. Sparsifying a network with respect to a log of "information actions" can be seen as revealing the *backbone* of information propagation in the network.

A high-level description of our approach is the following. We are given a social network, that is, a set of "friendship" links or "follower-followee" links. Additionally, we assume that we are given a log of actions performed by the nodes of the network. Such actions may include posting information "memes", joining an online community or an interest group, buying a paid subscription in an on-line service, and so on. We assume that those actions have propagated in the network via the *independent cascade* model [18]. The maximum likelihood parameters of this model can be found for instance by using the EM algorithm of Saito et al. [28]. Given the parameters we formulate the sparsification problem, which is the focus of this paper: we ask to preserve the k most important links in the model, i.e., the set of k links that maximize the likelihood of the observed data. Here k might be an input parameter specified by the data analyst, or alternatively k might be set automatically following common model-selection practice.

Our framework has a number of interesting applications.

Propagation characterization: Sparsifying separately different information topics can help us answer questions such as: "What distinguishes the way important news propagate, from the way funny "memes" do?" or "Is there any structural difference between the backbone of actual news and that of false rumors?"
Feed ranking: As users in social networks receive a continuous feed of information, ranking the most interesting feeds is becoming an important problem [15]. Sparsification provides a useful feature in this ranking problem by highlighting the most important links.
Viral marketing: Finding the set of users to target in order to maximize the spread of influence is a problem that has received a lot of attention, yet there are still serious computational challenges [3, 4]. As we show in Section 7, sparsification yields significant improvement for this problem, in terms of efficiency and scalability, while sacrificing little in terms of accuracy.

Even though there has been a lot of work recently on studying information propagation, mostly devoted either to empirical analysis of real-world propagations [1, 14, 19, 22], or to devise methods for influence maximization [6, 18, 27], not much effort has been devoted to develop techniques to mine large logs of propagation traces. On the other hand, there is extensive literature on the problem of network simplification, but the scenarios assumed are different than the problem we study in this paper. We review this literature in the next section.

Our contributions are summarized as follows.

- Given a social network and a log of actions, we study the problem of pruning the network to a prefixed extent while maximizing the likelihood of generating the propagation traces in the log (Sections 3 and 4).
- We show that our problem is **NP**-hard to approximate within any multiplicative factor (Section 4).
- We show that sparsification can be decomposed into a number of subproblems equal to the number of the nodes in the network. We then present an exponential, but optimal, dynamic programming algorithm, whose search space is typically much smaller than the brute force one, but still impracticable for graphs having nodes with a large in-degree (Section 5.1).
- We devise SPINE (*Sparsification of influence networks*), a greedy algorithm that achieves efficiency with practically little compromise in quality. SPINE is structured in two phases. During the first phase it selects a set of arcs D_0 that yields a finite log-likelihood; during the second phase, it greedily seeks a solution of maximum log-likelihood. The solution returned by SPINE is guaranteed to be "close" to the optimal among the subnetworks that contain arcs D_0 (Section 5.2).
- We show that SPINE identifies efficiently sparse networks of high or even optimal likelihood (Section 6).
- We apply SPINE as a pre-processing step for the problem of *influence maximization*, showing that computations on the sparse network give up little accuracy, but yield significant improvements in terms of efficiency and scalability (Section 7).

2. RELATED WORK

The study of information diffusion in social networks has a long history in social sciences. Early work studied the adoption of medical and agricultural innovations [5, 32]. Later, marketing researchers investigated the "word-of-mouth" diffusion process for *viral marketing* [2, 9, 17, 23].

From a computational perspective, a basic problem in viral marketing is that of *influence maximization*: given a social network, find k nodes to target in order to maximize the spread of influence. The first algorithmic treatment of the problem was provided by Domingos and Richardson [6, 27], who modeled the diffusion process in terms of Markov random fields, and proposed heuristic solutions to the problem. Subsequently, Kempe et al. [18] studied the influence-maximization problem for a different family of influence models. Part of their contribution was to provide approximation algorithms for the independent cascade model, which we also adopt in this paper. Recent work [3, 4, 21] improves the efficiency of influence maximization.

Conceptually, our work can be collocated with works on network simplification, the goal of which is to identify subnetworks that preserve properties of a given network. Toivonen et al. [31, 33], for instance, prune edges while keeping the original quality of best paths between all pairs of nodes. Here, quality is defined on path-based concepts, such as shortest path or maximum flow. Misiolek and Chen [24] prune arcs while maintaining the source-to-sink flow for each pair of nodes. In the theory community, the notion of k-spanner refers to a sparse subgraph that maintains the distances in the original graph up to a factor of k. The problem is to find the sparsest k-spanner [7]. In *pathfinder networks* [26, 29] the approach is to select weighted edges that do not violate the triangular inequality. Moreover, Fung et al. [35] study *cut-sparsifiers*, i.e. subsets of edges that preserve cuts up to a multiplicative error.

Serrano et al. [30], and Foti et al. [8], focus on weighted networks and select edges that represent statistically significant deviations with respect to a null model. In a similar setting, Arenas et al. [34] select edges that preserve modularity, a measure that quantifies quality of community detection.

The approach we take in this paper is substantially different than the approach considered in the above papers. One major difference is that the problem of sparsification is defined in terms of *observed activity* in the network, and not just in terms of the structural properties of the network.

A recent line of work, perhaps the closest to ours, assumes instead that the propagations are known, but the network is not [10, 11]. In their research, Gomez-Rodriguez et al. assume that connections between nodes cannot be observed, and they use observed traces of activity to infer a sparse, "hidden" network of information diffusion. Although similar to the sparsified networks we produce, in their setting the links are only inferred and might potentially not exist in the real, but unobserved, network. Instead, in our setting, connections between nodes are explicitly declared and *known* (as in current online social network services, for example), and our goal is to find the most important connections, and to obtain the network backbone.

Papers that share our approach of simultaneously mining the social graph and network activity include the work of Goyal et al. [12], on extracting frequent patterns of influence, and the works of Goyal et al. [13] and Saito et al. [28], on learning influence models from observed activity. However, they do not consider any notion of sparsification or other summarization operation.

3. INFLUENCE MODEL

Consider a social network $G = (V, D)$, where nodes V correspond to individuals and *directed* arcs D between nodes represent social connections. For two nodes u and v in V, arc (u, v) denotes that individual v knows, is socially connected to, or *follows* individual u. Individuals perform actions, and actions can *propagate* across the network. For example, social network users who read a news story on the New York Times may decide to share the story with their friends—who, in turn, may decide to spread the news to their own friends, and so on, contributing to the propagation of the story across the network. In this paper, we assume that the propagation of actions follows the independent cascade model [18]. According to this model, every arc (u, v) in D is associated with a probability $p(u, v)$ and propagations unfold in discrete time steps.

To be specific, let us consider the propagation of one action α across the network. According to the independent cascade model, if node u performs an action α in time step

t and node v follows u, then u makes a single attempt to influence v, and succeeds with probability $p(u, v)$. The probability of success is independent of other nodes that may attempt to influence v. If it is successful, and v has not previously performed α, then v performs α in time step $(t + 1)$. Note that, according to this model, several nodes may attempt to influence v, but v may perform α at most once.

Additionally, we assume that propagations are initiated by a special node $\Omega \in V$ (Figure 1(a)) that has the following two properties: (*i*) it performs α before any other node, and (*ii*) is followed by all other nodes in V. Node Ω models sources of influence that are external to the network, and thus $p(\Omega, v)$ is the probability that a propagation starts from node v.

The independent cascade model generates sequences of activations, or *traces*, as we call them. Traces of different actions are generated independently. Note that, given the trace of action α, it may not be possible to tell which node influenced a particular node v to perform α, since more than one node may have attempted to do so. In the example of Figure 1(b), for instance, one of nodes: u_1 or u_2, succeeded to influence v. We say that u_1 and u_2 *possibly influenced* v. On the other hand, neither u_3 nor u_4 could have influenced v, because then, according to the independent cascade model, v would have performed α at time t.

Every trace generated by the independent cascade model is associated with a likelihood value. Consider the trace of action α. Let $F^+_\alpha(v)$ be the set of nodes that *possibly influenced* v, and $F^-_\alpha(v)$ the set of nodes that *definitely failed to influence* v. Then, the likelihood $L_\alpha(G)$ of the trace can be written as

$$L_\alpha(G) = \prod_{v \in V} P^+_\alpha(v) \cdot P^-_\alpha(v), \qquad (1)$$

where

$$P^+_\alpha(v) = \begin{cases} 1 & \text{, if } F^+_\alpha(v) = \emptyset, \\ 1 - \prod_{u \in F^+_\alpha(v)} (1 - p(u, v)) & \text{, otherwise} \end{cases}$$

expresses the likelihood that at least one of the nodes in $F^+_\alpha(v)$ succeed to influence v, and

$$P^-_\alpha(v) = \prod_{u \in F^-_\alpha(v)} (1 - p(u, v)).$$

the likelihood that all nodes in $F^-_\alpha(v)$ fail. For the rest of the paper, following common practice, we opt to work with log-likelihood, due to its better numerical behavior

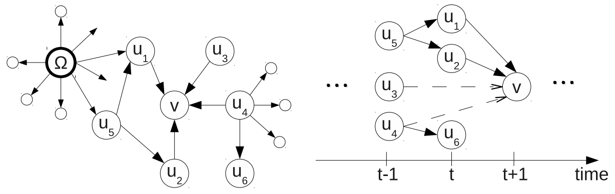
$$\log L_\alpha(G) = \sum_{v \in V} \left(\log P^+_\alpha(v) + \log P^-_\alpha(v) \right). \qquad (2)$$

We now describe how to estimate the probabilities $p(u, v)$ of the independent cascade model from a set of traces. Consider a set of actions $\mathcal{A}$. For each action $\alpha \in \mathcal{A}$ we observe its propagation trace, and assume that all traces are generated by the same model. Following [28], the probability values $p(u, v)$ that maximize the total log-likelihood

$$\log L(G) = \sum_{\alpha \in \mathcal{A}} \log L_\alpha(G)$$

can be computed using the following iterative formula

$$p^{(k+1)}(u, v) = \frac{p^{(k)}(u, v)}{|A^+_{v|u}| + |A^-_{v|u}|} \sum_{\alpha \in A^+_{v|u}} \frac{1}{P^+_\alpha(v)}, \qquad (3)$$



(a) Social Network (b) Trace

Figure 1: (a) Social network. An arc (u, v) indicates that v is a *follower* of u and repeats u's actions with probability $p(u, v)$. Node Ω initiates propagations of actions and influences all nodes. (b) Action trace. Solid-line arrows indicate possible influence, while dash-line arrows indicate failed influence attempts.

where actions in set

$$A^+_{v|u} = \{\alpha \in \mathcal{A} \mid F^+_\alpha(v) \ni u\}$$

have traces where u possibly influenced v, and actions in set

$$A^-_{v|u} = \{\alpha \in \mathcal{A} \mid F^-_\alpha(v) \ni u\}$$

have traces where u definitely failed to influence v.

According to the independent cascade model, actions propagate in discrete time steps. In real-world scenarios, however, users can freely choose *when* to propagate actions of the people they follow. Therefore, to use the independent cascade model in practice and estimate probabilities $p(u, v)$ in real-world datasets, we can not rely on discrete time steps.

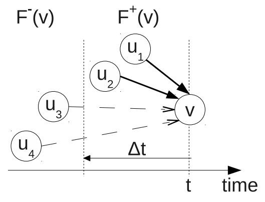


Figure 2: Delay Threshold

Instead, we may use a delay threshold Δt (Figure 2) for this purpose. Specifically, suppose that node v performs action α at time $t_\alpha(v)$. Then, the set $F^+_\alpha(v)$ of nodes that possibly influenced v are the nodes that performed action α before v and within Δt time, that is,

$$F^+_\alpha(v) = \{u : (u, v) \in D \mid t_\alpha(v) - \Delta t \le t_\alpha(u) < t_\alpha(v)\}.$$

In the scenario of Figure 2, for example, $F^+_\alpha(v) = \{u_1, u_2\}$. The set $F^-_\alpha(v)$ of nodes that definitely failed to influence v is defined similarly:

$$F^-_\alpha(v) = \{u : (u, v) \in D \mid t_\alpha(u) < t_\alpha(v) - \Delta t\}.$$

In the scenario of Figure 2, for example, $F^-_\alpha(v) = \{u_3, u_4\}$.

Note that if v does not perform α, then we write $t_\alpha(v) = \infty$ and have $F^+_\alpha(v) = \emptyset$. Note also that, in the special case of $\Delta t = \infty$, always one of the two sets $F^+_\alpha(v)$ and $F^-_\alpha(v)$ is empty. Specifically, if v does not perform α, then $F^+_\alpha(v) = \emptyset$, while if it does, then $F^-_\alpha(v) = \emptyset$.

4. SPARSIFICATION

Given an influence model learned (as in Section 3) from a social network $G = (V, D)$ and propagation traces for a set of actions $\mathcal{A}$, our goal is to identify a "backbone" of arcs that are most important for the propagation of actions $\mathcal{A}$. Specifically, we aim to identify a sparse subnetwork G_s of G, that consists of the k arcs that are most likely to have generated the observed traces of actions $\mathcal{A}$.

Formally, we define a network $G_s = (V, D_s)$ to be a *sparse* subnetwork of $G = (V, D)$ if its arcs D_s are a subset of D, $D_s \subseteq D$, and the probabilities $p_s(u, v)$ are equal with the corresponding probabilities $p(u, v)$ of the network G, that is, $p_s(u, v) = p(u, v)$, for all (u, v) in D_s. The problem of sparsifying network G is defined as follows.

Problem 1 (Sparsification). *Given a network $G = (V, D)$ with probabilities $p(u, v)$ on the arcs, a set $\mathcal{A}$ of action traces, and an integer k, find a sparse subnetwork $G_s = (V, D_s)$ of G of size $|D_s| = k$, so that the log-likelihood function $\log L(G_s)$ is maximized.*

Notice that Problem 1 is not solved by selecting the k arcs (u, v) in D with the largest probability values $p(u, v)$. As a counter-example, consider the network $G = (V, D)$ and traces of actions $\mathcal{A} = \{\alpha_1, \alpha_2, \alpha_3\}$, shown in Figure 3. Based on the three traces, we obtain the maximum likelihood estimates $p(u_1, v) = 1.0$ and $p(u_2, v) = 0.5$. Then, for $k = 3$, the best sparse model $G_s = (V, D_s)$ is the one with

$$D_s = \{(\Omega, u_1), (\Omega, u_2), (u_2, v)\} = D \setminus \{(u_1, v)\}, \qquad (4)$$

even though $p(u_2, v) < p(u_1, v)$. To see why, notice that, for example, the alternative option of

$$D_s = \{(\Omega, u_1), (\Omega, u_2), (u_1, v)\} = D \setminus \{(u_2, v)\} \qquad (5)$$

leads to zero likelihood ($\log L(G_s) = -\infty$). This is because the trace of action α_2 is impossible without arc (u_2, v). On the other hand, all three traces are possible in the absence of the arc (u_1, v).

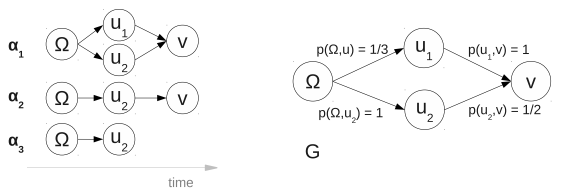


Figure 3: Sparsification is not solved by selecting the k arcs with largest probability value.

Complexity. By the definition of the independent cascade model, for a sparse network $G_s = (V, D_s)$ to have finite log-likelihood, $\log L(G_s) > -\infty$, it is necessary that the traces of all actions $\mathcal{A}$ are possible for its set of arcs D_s. This means that if node v performs an action α in $\mathcal{A}$, then D_s must include an arc from at least one of the nodes F^+_α that possibly influenced v. Our argument is formally stated below.

Lemma 1. *Deciding whether Problem 1 has finite solution is* **NP**-*hard.*

Proof. We obtain a reduction from the `Hitting Set` problem. We first remind the `Hitting Set` problem: Given a collection of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$ over a universe of n elements $U = \{1, \ldots, n\}$ (i.e., $S_j \subseteq U$), a hitting set for $\mathcal{S}$ is a set $H \subseteq U$ that intersects all sets in $\mathcal{S}$, that is, $H \cap S_j \neq \emptyset$ for all $j = 1, \ldots, m$. Given a set collection $\mathcal{S}$ and an integer k, it is **NP**-hard to decide whether there is a hitting set H for $\mathcal{S}$ that has size at most $|H| \leq k$.

Now, consider an instance of the `Hitting Set` problem, i.e. consider a collection $\mathcal{S} = \{S_1, \ldots, S_m\}$ and an integer k. We create an instance of our problem as follows. Our graph $G = (V, D)$ has $n + 1$ nodes, namely, $V = U \cup \{n + 1\} = \{1, \ldots, n, n + 1\}$. The nodes from 0 to n have no ancestors, while the node $n + 1$ has all nodes from 0 to n as ancestors. Thus $D = \{(i, n + 1) \mid i = 1, \ldots, n\}$. For all the edges in D we set $p(i, n + 1) = 1$. Next, for each set $S_j \in \mathcal{S}$ we consider an action α that was performed by nodes $S_j \cup \{n + 1\}$.

First consider a hitting set H of $\mathcal{S}$, with $|H| \leq k$. Take the sparsified graph $G_s = (V, D_s)$ with $D_s = \{(i, n+1) \mid i \in H\}$. It is $|D_s| \leq k$. Furthermore for each action $\alpha \in \mathcal{A}$ the set D_s contains at least one edge from a parent of node $n + 1$ that is also influenced by α. Given that all arcs have probability 1 it follows that the probability $P^+_\alpha(n + 1)$ is equal to 1, and consecutively the total log-likelihood is finite.

Conversely, for any sparsified graph with at most k edges and finite log-likelihood it should be the case that for each action $\alpha \in \mathcal{A}$ the set D_s contains at least one edge from a parent of node $n + 1$ that also performed α. Thus the set $H = \{i \mid (i, n + 1) \in D_s\}$ is a hitting set for $\mathcal{S}$. □

Lemma 1 leads to the following hardness results.

Theorem 1. *Problem 1 is* **NP**-*hard.*

Theorem 2. *Approximating Problem 1 up to any multiplicative factor is* **NP**-*hard.*

The latter Theorem follows from the simple observation that obtaining a multiplicative-factor approximation is at least as difficult as obtaining a finite solution. Theorem 1 is a special case of Theorem 2.

5. ALGORITHMS

5.1 An optimal algorithm

A brute-force approach to acquire an optimal solution to Problem 1 is to enumerate all possible subsets of arcs $D_s \subseteq D$ of size k, and select the network $G_s = (V, D_s)$ with maximum log-likelihood $\log L(G_s)$. This approach is exponential in the size of D and obviously does not scale.

Instead, we describe an optimal, dynamic programming algorithm, OptimalSparse, that makes relatively limited use of exhaustive enumeration. The algorithm is based on the observation that an optimal solution can be obtained by dividing Problem 1 into $|V|$ sub-problems, where V is the set of nodes of G, and combining their solutions. To see this, let us re-write log-likelihood as

$$\begin{aligned}\log L(G) &= \sum_{\alpha \in \mathcal{A}} \log L_\alpha(G) = \\ &= \sum_{\alpha \in \mathcal{A}} \sum_{v \in V} \left(\log P^+_\alpha(v) + \log P^-_\alpha(v)\right) \\ &= \sum_{v \in V} \sum_{\alpha \in \mathcal{A}} \left(\log P^+_\alpha(v) + \log P^-_\alpha(v)\right).\end{aligned} \qquad (6)$$

Observe that the inner sum of the above formula corresponds to a single node v. Also, recall that the probabilities $P^+_\alpha(v)$ and $P^-_\alpha(v)$ are "local" to node v, meaning that they depend only on influence probabilities on arcs from the parent nodes of v in G. Denote the set of these arcs by D_v, and observe that we can compute $P^+_\alpha(v)$ and $P^-_\alpha(v)$ for an arbitrary subset $X_v \subseteq D_v$ simply by omitting probabilities on edges not present in X_v. This way we can define

$$\lambda(X_v) = \sum_{\alpha \in \mathcal{A}} \left(\log P^+_\alpha(v|X_v) + \log P^-_\alpha(v|X_v)\right), \qquad (7)$$

for $X_v \subseteq D_v$. The log-likelihood of the full model can thus be written as $\log L(G) = \sum_v \lambda(D_v)$. Therefore we consider the following sub-problem for each node $v \in V$.

PROBLEM 2. *Given an integer b, identify a subset of arcs $D^b_v \subseteq D_v$ of size b, such that $\lambda(D^b_v)$ is maximized.*

Problem 2 is solved by exhaustive enumeration of subsets $D^b_v \subseteq D_v$ of size b. The solution space increases exponentially with the size of D_v, but is typically much smaller than that of the brute-force approach, which increases exponentially with the size of D.

Algorithm OPTIMALSPARSE uses dynamic programming to compute an optimal solution D_s to Problem 1 from optimal solutions of Problem 2. Specifically, let $V = \{v_1, v_2, \ldots\}$ be an enumeration of the nodes. For each node v_i, consider the solution $D^b_{v_i}$ to Problem 2 for all integers $b \in [1, k]$ (Algorithm 1, Lines 1-3). OPTIMALSPARSE then proceeds sequentially over nodes V (Algorithm 1, Line 5). At the end of the i-th step, it has processed nodes $v_1, v_2, \ldots, v_i$ and, for each integer $m \in [1, k]$ (Algorithm 1, Line 6) it identifies one number b for each node $v \in \{v_1, v_2, \ldots, v_i\}$ such that $b_1 + b_2 + \ldots + b_i = m$ and the sum

$$\Lambda(i, m) = \lambda(D^{b_1}_{v_1}) + \lambda(D^{b_2}_{v_2}) + \ldots + \lambda(D^{b_i}_{v_i})$$

is maximized (Algorithm 1, Lines 7-8). By construction, $\Lambda(|V|, k)$ contains the maximum log-likelihood value for Problem 1. The optimal solution D_s is computed with standard back-tracking (not described in Algorithm 1, in interest of brevity).

Algorithm 1 `OptimalSparse`

```
1: for $i = 1$ to $|V|$ do
2:     for $b = 1$ to $k$ do
3:         compute optimal $D^b_{v_i}$ (Problem 2)
4: array $\Lambda$, initialize $\Lambda(0, m) = 0$; $m = 1$ to $k$
5: for $i = 1$ to $|V|$ do
6:     for $m = 1$ to $k$ do
7:         $b_i := \arg\max\{\lambda(D^{b_i}_{v_i}) + \Lambda(i-1, m-b_i)\}$
8:         $\Lambda(i, m) := \lambda(D^{b_i}_{v_i}) + \Lambda(i-1, m-b_i)$
9: return $\Lambda(|V|, k)$
```

5.2 A greedy algorithm: SPINE

OPTIMALSPARSE follows a more sophisticated approach than brute-force, but is still prohibitively expensive for graphs that have nodes with large in-degree. In this section, we describe SPINE, a greedy algorithm for Problem 1. Although Problem 1 is inapproximable in the general case, we find experimentally that the quality of the results obtained by SPINE is comparable to the that of OPTIMALSPARSE.

SPINE produces a solution D_s to Problem 1 in k steps, adding to D_s one arc at each step. Those k steps are divided in two phases: during the first phase, SPINE aims to identify a solution D_0 of finite log-likelihood; during the second phase, it greedily seeks a solution of maximum log-likelihood. This two-phase approach is inspired by the observation that Problem 1 is at least as difficult as identifying a solution of finite log-likelihood.

First phase. Following the discussion on complexity of Problem 1 (Section 4), we look for a solution with finite $\log L$ by solving an instance of `Hitting Set` — that is, for each node $v \in V$ we seek for a hitting set of collection

$$C(v) = \{D^+_\alpha(v) \neq \emptyset \; ; \; \alpha \in \mathcal{A}\}.$$

As `Hitting Set` is **NP**-hard, we employ the greedy approximation algorithm described in [16] (Algorithm 2, Lines 1-8). According to that algorithm, arcs (u, v) are ordered by the number $n(u, v)$ of actions for which u possibly influenced v

$$n(u, v) = \left|\{D^+_\alpha(v) \in C(v) \; ; \; (u, v) \in D^+_\alpha(v)\}\right|. \qquad (8)$$

At each step, the arc (u, v) with the maximum number $n(u, v)$ is selected (Algorithm 2, Lines 6-7) and all sets $D^+_\alpha(v)$ that contain (u, v) are ignored for the rest of this process (Algorithm 2, Line 8). The first phase ends when either the limit of k arcs is reached, or $\lambda(D^b_v)$ for $b \leq k$ selected arcs becomes finite for the first time.

Second phase. Let D_0 be the set of arcs selected by the end of the first phase, and let $G_0 = (V, D_0)$ be the associated sparse network. If $|D_0| < k$, then we still need to select $k - |D_0|$ more arcs. One viable approach to select the remaining $k - |D_0|$ arcs is to continue in a fashion similar to the OPTIMALSPARSE algorithm: divide the problem into $|V|$ subproblems, one for each node in the graph, solve optimally each subproblem for all values of $b = 1, \ldots, k - |D_0|$, and then combine the solutions of the subproblems with dynamic programming. However, due to the exhaustive search and the dynamic programming, such an approach would not be scalable. To obtain a scalable solution we propose to speed up the computation by replacing both steps – exhaustive search and the dynamic programming – with a greedy process. We choose the remaining $k - |D_0|$ arcs by selecting greedily at each step the arc that offers the largest increase in log-likelihood (Algorithm 2, Lines 9-15).

For a detailed description, let D_v and λ be defined as above, and consider a table X, where $X_v(i)$ is the subset of D_v that the greedy algorithm would add the ith step when maximizing λ. More formally, we have

$$X_v(i) = \begin{cases} \emptyset & \text{if } i = 0, \\ X_v(i-1) \cup e_v(i) & \text{otherwise,} \end{cases}$$

where $e_v(i) = \arg\max_{e \in U_v}\{\lambda\big(X_v(i-1) \cup e\big)\}$, and $U_v = D_v \setminus X_v(i-1)$. Also, let $H_v(i)$ denote the marginal gain for λ when the ith edge is added, that is, let

$$H_v(i) = \begin{cases} 0 & \text{if } i = 0, \\ \lambda(X_v(i)) - \lambda(X_v(i-1)) & \text{otherwise.} \end{cases}$$

SPINE maintains the X_v and H_v structures for every v, and at every step chooses the v^* and i^* that maximize $H_{v^*}(i^*)$ (line 13). The edge $e_{v^*}(i^*)$ will be added to the solution, and we compute $X_{v^*}(i^*+1)$ and $H_{v^*}(i^*+1)$. Computing $X_v(i+1)$ given $X_v(i)$ is an $O(|D_v|)$ operation, while inserting and extracting from Q are $O(\log |V|)$. The worst-case complexity of the 2nd phase of SPINE is thus $O\big(k(\max_v(|D_v|)+\log |V|)\big)$.

We have the following approximation guarantee.

LEMMA 2. *Let $D_{\rm opt}$ be a superset of D_0 that contains k arcs and that induces a subgraph $G_{\rm opt} = (V, D_{\rm opt})$ of G with maximum log-likelihood. Also, let $D_{\rm sp}$ by the set of arcs returned by* SPINE *and let $G_{\rm sp} = (V, D_{\rm sp})$ be the induced subgraph. That is, $D_{\rm sp}$ is also a superset of D_0 and it has k arcs. Then, provided that $\log L(G_0) > -\infty$, we have*

$$\log L(G_{\rm sp}) \geq \frac{1}{e} \log L(G_0) + (1 - \frac{1}{e}) \log L(D_{\rm opt}). \tag{9}$$

Proof. We use a well-known theorem by Nemhauser et al. [25] on maximizing submodular set functions. A set function f is submodular, when for any $S \subseteq T$ we have $f(S \cup u) - f(S) \geq f(T \cup u) - f(T)$. The theorem in [25] states that for non-negative set functions f, with $f(\emptyset) = 0$, we have $f(X_k^A) \geq (1 - 1/e)f(X_k^*)$, where X_k^* is the optimal k-sized solution, and X_k^A a solution likewise of size k that is constructed by starting from $\emptyset$, and at each step adding the item that maximizes the increase in the value of f. Observe that this greedy strategy is also used by SPINE when adding edges to the sparse model.

A direct application of this result is impossible, as log-likelihood is negative, and not equal to zero for an empty solution. However, we consider the following modification: Denote by $LL(G)$ the log-likelihood $\log L(G)$, and let

$$g(S) = LL(G_0 \cup G_S) - LL(G_0), \tag{10}$$

where $S \subset D$ is a set of edges, G_0 is the graph consisting of the edges found in the 1st phase of SPINE, and G_S is the graph corresponding to the edges in the set S. Furthermore, observe that S can be seen as the set of edges added by SPINE in the 2nd phase. Clearly $g(S)$ is non-negative, and $g(\emptyset) = 0$. We can also show the following lemma, whose proof can be found in the extended version of this paper.

LEMMA 3. *Function $g(S)$ is submodular.*

Now we apply the result of [25] directly, and have

$$g(S^A) \geq (1 - 1/e)g(S^*). \tag{11}$$

Expanding $g(S)$, inequality (11) gives

$$\begin{aligned} LL(G_{\rm sp}) &\geq (1 - 1/e)\,(LL(G_{\rm opt}) - LL(G_0)) + LL(G_0) \\ &= \frac{1}{e} LL(G_0) + (1 - \frac{1}{e}) LL(G_{\rm opt}), \end{aligned}$$

where $G_{\rm sp} = G_0 \cup G^A$ is the solution returned by SPINE, and $G_{\rm opt} = G_0 \cup G^*$ is the optimal solution that contains G_0 as a subset, which concludes the proof of Lemma 2. □

Lemma 2 guarantees that the solution returned by SPINE is close to the optimal among the subnetworks that contain arcs D_0. Notice that this result is not a true approximation result for Problem 1. First, Lemma 2 does not associate $\log L(G_{\rm sp})$ with $\log L(G_s)$, the log-likelihood of the optimal solution, and secondly, it is true under the provision that $\log L(G_0) > -\infty$, i.e., that the first phase of SPINE "escapes" infinite log-likelihood before the k-th step.

A note on parallelization. In practice we have observed that the total execution time of SPINE is dominated by Phase 2 by at least an order of magnitude. Therefore we only concentrate on optimizing this part of the algorithm. Note that the computationally involved part in the 2nd phase is updating the table X on line 14. In Algorithm 2 we compute

Algorithm 2 SPINE

```
 1: {First Phase}
 2: for $i = 1$ to $|V|$ do
 3:    $D_0 := \emptyset$
 4:    $C(v) := \{D^+_\alpha(v) \neq \emptyset\ ;\ \alpha \in \mathcal{A}\}$
 5:    while $|D_0| < k$ and $\log L(G_0(V, D_0) = -\infty$ do
 6:       find $u : (u,v) \in D$ with max $n(u,v)$
 7:       $D_0 := D_0 \cup \{(u,v)\}$
 8:       $C(v) := C(v) -$ {all $D^+_\alpha(v)$ that contain $(u,v)$}
 9: {Second Phase}
10: $D_{\rm sp} := D_0$, $Q :=$ empty priority queue
11: $\forall v \in V$ : compute $X_v(1)$, insert $H_v(1)$ to $Q$
12: while $|D_{\rm sp}| \leq k$ do
13:    extract $H_{v^*}(i^*)$ from $Q$
14:    compute $X_{v^*}(i^*+1)$, insert $H_{v^*}(i^*+1)$ to $Q$
15:    $D_{\rm sp} := D_{\rm sp} \cup \{X_{v^*}(i^*) \setminus X_{v^*}(i^*-1)\}$
16: return $G_{\rm sp} = G_{\rm sp}(V, D_{\rm sp})$
```

entries of X on-demand, i.e., the entry $X_v(i)$ is only computed when we must find out if the ith parent of v should belong to the solution. Alternatively we could compute the entire table X in advance, and then run phase 2 of SPINE with this as the input. This approach has the advantage that the values $X_v(1), X_v(2), \ldots$ can be computed independently of other vertices for each $v \in V$, hence SPINE lends itself to easy parallelization by arbitrarily partitioning V.

Setting the value of k. The size k of sparse network G_s is given as input to problem 1. A natural question that arises, then, is how to specify k in a principled manner, so as to identify sparse networks that combine small size with high log-likelihood. Following common model-selection practice, one way to set k is via minimization of the *Bayesian Information Criterion* (BIC).

$$BIC(G_s) = -2 \log L(G_s) + k \log(|A|) \tag{12}$$

It is straightforward to modify our algorithms to use BIC instead of log-likelihood, and return a sparse network G_s that minimizes the BIC. Part of our experiments (Section 6) is devoted to assessing SPINE in terms of BIC.

6. EXPERIMENTS

In this section we present an evaluation of the performance of SPINE on real datasets.

6.1 Experimental framework

Datasets. We test our algorithms on samples extracted from two different datasets. The first data source, referred to as YMEME in the following, is a set of microblogging postings in Yahoo! Meme.[1] Nodes are users, actions correspond to postings (typically photos) that users share through the site, and arcs from a node u to a node v indicate that v follows u. The data source contains all posts in this platform from 2008 to early 2010.

The second data source, referred to as MTRACK in the following, is a set of phrases propagated over prominent online news sites in March 2009, obtained by Meme Tracker [20]. Nodes are mostly news portals or news blogs and actions correspond to phrases that are found by the Meme Tracker algorithm to be repeated across several sites. In absence of explicitly declared "follower-followee" relationships between

[1]`http://meme.yahoo.com/`

the sites, arcs from a node u to a node v indicate that the website v linked to the website u during March 2009, and thus v "follows" u. This dataset is publicly available.[2]

We used a snowball sampling procedure to obtain several subsets from these data sources. In the case of YMeme, we sampled a connected sub-graph of the social network containing the users that participated in the most reposted items. This yields very densely connected subgraphs. In the case of MTrack, we sampled a set of highly reposted items posted by the most active sites. This yields more loosely connected subgraphs. A summary of the subsets we created is shown in Table 1.

To estimate the probabilities associated with arcs, we use the EM algorithm of [28] (Section 3). In interest of simplicity, here we show results only for delay threshold $\Delta t = \infty$ and study effects of Δt in an extended version of the paper. As explained in Section 3, $\Delta t = \infty$ means that when node v performs action α, then $F^+_\alpha(v)$ contains all nodes followed by v that performed α before v, and $F^-_\alpha(v)$ is empty. On the other hand, if v does not perform action α, then $F^+_\alpha(v)$ is empty, and $F^-_\alpha(v)$ contains all nodes followed by v that performed α and thus failed to influence v.

As a product of the maximum-likelihood estimation, it may be that some arcs are associated with zero (0) influence probability. The last column of table 1 reports the number of arcs with positive probability after estimation terminates.

Table 1: Summary of datasets used.

Dataset	Actions	Nodes	Arcs input	Arcs prob.>0
YMeme-L	26 000	10776	1 247 711	432 613
YMeme-M	13 000	9 525	1 154 674	378 520
YMeme-S	5 000	2 573	466 284	73 396
MTrack-L	9 000	43 865	199 153	7 788
MTrack-M	120	35 304	110 759	1 417
MTrack-S	780	30 300	78 302	768

Alternative algorithms. While there is no established baseline or benchmark, there are algorithms we can compare with. On one extreme, we have `OptimalSparse` that should be the most effective, at the expense of speed. We experimented with `OptimalSparse` and discovered that its running time is prohibitive for the size of datasets we use; we thus omit its performance from our results. On the other extreme, there are fast heuristic methods that may yield good solutions. One such method is `SortByProbability` that sorts the arcs in D by decreasing probability (we considered a variant that sort arcs by decreasing number of actions traversing each arc, which yields worse results than `SortByProbability` and is thus omitted). Finally, we consider algorithm `Random`, that simply permutes all edges having a non-zero probability. Please note that both heuristics use Spine's first phase as an *initialization* procedure.

Implementation. All algorithms are implemented in Java using the COLT library implementation of sparse matrices and the Fastutils library for type-specific maps and collections.[3][4] Our experiments are performed on a single Dual-Core *2530MHz Intel* processor and using less than 10GBs of memory for the largest datasets.

[2]`http://snap.stanford.edu/data/memetracker9.html`
[3]`http://acs.lbl.gov/software/colt/`
[4]`http://fastutil.dsi.unimi.it/`

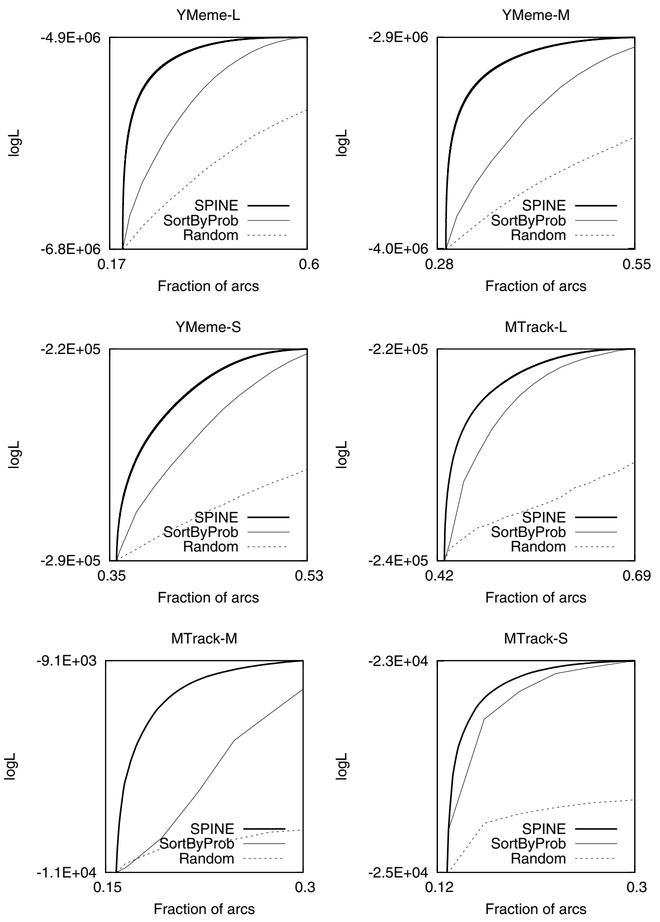


Figure 4: Log-likelihood vs fraction of non-zero-probability arcs.

For efficiency reasons we create $F^+_\alpha(v)$ and $F^-_\alpha(v)$ at initialization and keep them in memory to quickly compute the log-likelihood function. Most processing time during sparsification is spent accessing these data structures to compute the likelihood for a subset of the edges incident to a node.

6.2 Results

We execute the different sparsification algorithms and measure log-likelihood at different levels of k (Figure 4).

The plots in Figure 4 demonstrate that Spine obtains sparse graphs of high or even maximum likelihood with a fraction of the total network size. For instance, we observe that for the YMeme-L dataset, the initialization procedure produces a non-zero-likelihood solution at 17% of non-zero-probability arcs – that's about 90 000 arcs, i.e., 7% the total number of arcs. Then, Spine achieves maximum log-likelihood at 60% of non-zero arcs – that's about 250 000 arcs, i.e., 20% of the total number of arcs. (Note that we know Spine achieves maximum likelihood for that number of arcs, because the corresponding likelihood value remains at that level for larger values of k and is equal to the likelihood of the original network). Similar observations hold for all other datasets.

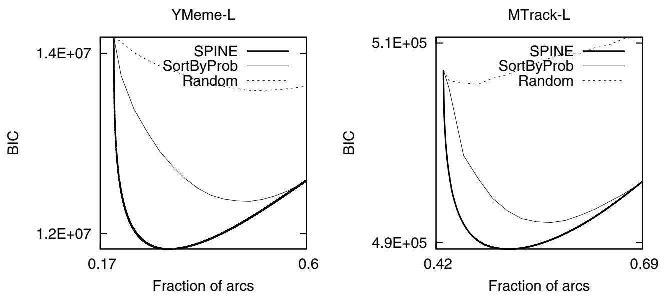


Figure 5: BIC vs fraction of non-zero-probability arcs.

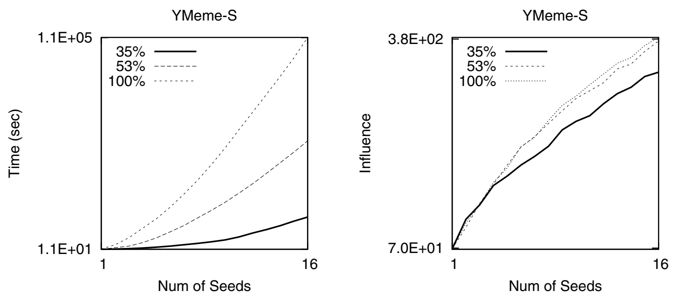


Figure 6: Seed selection running time and influence vs seed-set size, for sparse networks of different size (shown as percentage of non-zero probability arcs).

Notice also that Spine constantly outperforms the two heuristics, `SortByProbability` and `Random`, even though they begin from the same basis, i.e., the same set of arcs of non-zero likelihood. Among the two, `SortByProbability` performs better, and in some cases achieves log-likelihood close to that of Spine (e.g., MTrack-S, in Figure 4).

Moreover, Figure 5 displays the BIC scores for sparse networks obtained from YMeme-L and MTrack-L (similar plots were obtained for the rest of the datasets). We make two observations. First, there is a clear minimum, suggesting an optimal level of sparsification. Second, at that value of k, Spine outperforms the two heuristics by a wide margin in terms of likelihood. Specifically, for YMeme-L, Spine has its best BIC score at 39% of non-zero arcs (13% of all arcs), with log-likelihood that exceeds that of `SortByProbability` by 3×10^5. Similarly, for MTrack-L, Spine has its best score at 59% of non-zero arcs (2% of all arcs), with log-likelihood that exceeds that of `SortByProbability` by 8×10^2.

Running time. We experimented with Spine over many large datasets and observed that its performance offers a vast improvement over that of `OptimalSparse`, the running time of which increases exponentially with the network size and prohibits any use in practice. As a matter of fact, `Optimal-Sparse` takes several days to complete for smaller datasets than the ones we report here.

Under the current implementation, the two simpler heuristics typically performed faster than Şpine. In YMeme-L, the largest dataset of our experimental evaluation, the corresponding time to achieve maximum likelihood was 55 minutes for `Random` and `SortByProbability`. Spine achieved maximum likelihood (at $260,000$ arcs) in 3.5 hours and the execution time for the first phase of the algorithm was about 15 minutes. For MTrack-S, the smallest dataset, the corresponding running time was less than 1 second. The time needed by Spine to obtain a sparse network of maximum likelihood is reported in Table 2 for all datasets.

Table 2: Spine – Time to reach max-likelihood

YMeme-L	3.5 hrs	MTrack-L	10 sec
YMeme-M	35 min	MTrack-M	1 sec
YMeme-S	1.5 min	MTrack-S	< 1 sec

7. APPLICATION: INFLUENCE MAXIMIZATION

In Section 1 we claim that sparsification is a fundamental data-reduction operation with many applications. As a proof of concept, we apply Spine as a pre-processing step for the *influence-maximization problem* [18], showing that computations on sparsified models give up little accuracy, but yield significant improvements in terms of scalability.

Consider a social network G with arcs annotated with probabilities of influence and assume the independent cascade model of propagation. Consider also an arbitrary set S of nodes in G (the "seed set"), and assume they are the first to perform some action α at time $t = 0$. Then, the spread of influence of that set is defined as the (expected) total number of nodes that are eventually influenced to perform α as well. Given an integer s, the problem of influence maximization consists in identifying a set S of cardinality no more than s that has maximum spread of influence.

We apply Spine on the original network G of YMeme-S, one of our larger datasets, to identify two sparse networks G_1, G_2, of $k_1 = 25688$, $k_2 = 38899$ arcs, respectively. Network G_1 is the smallest network with non-zero likelihood identified by Spine. Network G_2 is the smallest network of maximum likelihood.

For each of the aforementioned networks (G, G_1, G_2) we run the influence maximization greedy algorithm of [18], denoted `MaxInfl`, to identify seed-sets of size s, for different values of s. `MaxInfl` greedily selects seed nodes, every time choosing the node that leads to the biggest increase in influence spread. To estimate influence spread, `MaxInfl` performs Monte Carlo simulations of the independent cascade model. In our experiments, we perform batches of 100 simulations for every candidate seed set, and consider our estimate stable when it changes less than 10%. For each network and seed-set size s, we measure (i) the running time of the algorithm, and (ii) the influence of the identified seed set.

As shown in Figure 6, running `MaxInfl` over G_2 returns seed sets of almost optimal influence, at a considerable gain in speed in comparison with running on G, even for small seed sets. In addition, we observe that performing seed selection over G_1, the sparsest network, leads to large gains in efficiency, while returning seed sets with high influence. For example, when run over G, `MaxInfl` identifies a seed set of size $|S| = 10$ and influence $I_0 = 290$, in $t_0 = 54582$ seconds (15 hours). On the other hand, when run over G_1, it identifies a seed set of the same size and influence $I_1 = 0.88 \cdot I_0 = 255$, in $t_1 = 0.09 \cdot t_0 = 5061$ seconds (1.4 hours). As we can see, the gain from efficiency outweighs the loss in quality, especially for large sizes of S where running `MaxInfl` on G becomes excessively expensive.

8. CONCLUSIONS AND FUTURE WORK

We study the problem of sparsifying influence networks. Given a social graph and a log of propagations, we select the k arcs that best describe the propagations in terms of likelihood. We show that the problem is inapproximable within any multiplicative factor, and introduce SPINE, a greedy algorithm, to solve it efficiently.

Through experimental evaluation over real datasets, we demonstrate that SPINE identifies sparse sub-networks with practically little compromise in quality. We demonstrate examples of such sparse subnetworks, and apply sparsification as a pre-processing step to the influence maximization problem, showing that it provides significant gains in efficiency at little loss of quality.

Given the decomposition of the problem and hence its suitability to parallelization (see Section 5.2, 'A note on parallelization'), in our on-going work we are developing a Hadoop implementation of SPINE, which will allow to scale to extremely large networks.

Reproducibility. Code implementing the SPINE algorithm, along with instructions to repeat the experiments over the public MTRACK data source, is available at `http://queens.db.toronto.edu/~mathiou//spine/`.

Acknowledgements. This work was partially supported by the Spanish Centre for the Development of Industrial Technology under the CENIT program, project CEN-20101037, "Social Media" (www.cenitsocialmedia.es).

9. REFERENCES

[1] E. Adar and L. A. Adamic. Tracking information epidemics in blogspace. In *Int. Conf. on Web Intelligence (WI'05)*.

[2] F. Bass. A new product growth model for consumer durables. *Management Science*, 15:215–227, 1969.

[3] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *16th Int. Conf. on Knowledge Discovery and Data Mining (KDD'10)*.

[4] W. Chen, Y. Yuan, and L. Zhang. Scalable influence maximization in social networks under the linear threshold model. In *10th IEEE Int. Conf. on Data Mining (ICDM'10)*.

[5] J. Coleman, H. Menzel, and E. Katz. *Medical Innovations: A Diffusion Study*. Bobbs Merrill, 1966.

[6] P. Domingos and M. Richardson. Mining the network value of customers. In *7th Int. Conf. on Knowledge Discovery and Data Mining (KDD'01)*.

[7] M. Elkin and D. Peleg. Approximating k-spanner problems for $k > 2$. In *8th Conf. on Integer Programming and Combinatorial Optimization*, 2001.

[8] N. J. Foti and J. M. Hughes and D. N. Rockmore. Nonparametric Sparsification of Complex Multiscale Networks. *PLoS ONE*, 6(2), 2011.

[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] M. Gomez-Rodriguez, D. Balduzzi, and B. Schölkopf. Uncovering the Temporal Dynamics of Diffusion Networks In *28th Int. Conf. on Machine Learning (ICML'11)*.

[11] M. Gomez-Rodriguez, J. Leskovec, and A. Krause. Inferring networks of diffusion and influence. In *16th Int. Conf. on Knowledge Discovery and Data Mining (KDD'10)*.

[12] A. Goyal, F. Bonchi, and L. Lakshmanan. Discovering leaders from community actions. In *17th Conf. on Inf. and Knowledge Management (CIKM'08)*.

[13] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan. Learning influence probabilities in social networks. In *Third ACM Int. Conf. on Web Search and Data Mining (WSDM'10)*.

[14] D. Gruhl, R. V. Guha, D. Liben-Nowell, and A. Tomkins. Information diffusion through blogspace. In *13th Int. Conf. on World Wide Web, (WWW'04)*.

[15] D. Ienco, F. Bonchi, and C. Castillo. The meme ranking problem: Maximizing microblogging virality. In *SIASP 2010 workshop at ICDM*.

[16] D. S. Johnson. Approximation algorithms for combinatorial problems. In *5th Symp. on Theory of Computing (STOC'73)*.

[17] S. Jurvetson. What exactly is viral marketing? *Red Herring*, 78:110–112, 2000.

[18] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *9th Int. Conf. on Knowledge Discovery and Data Mining (KDD'03)*.

[19] J. Leskovec, L. A. Adamic, and B. A. Huberman. The dynamics of viral marketing. *TWEB*, 1(1), 2007.

[20] J. Leskovec, L. Backstrom, and J. M. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *15th Int. Conf. on Knowledge Discovery and Data Mining (KDD'09)*.

[21] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In *13th Int. Conf. on Knowledge Discovery and Data Mining (KDD'07)*.

[22] J. Leskovec, A. Singh, and J. M. Kleinberg. Patterns of influence in a recommendation network. In *10th Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD'06)*.

[23] V. Mahajan, E. Muller, and F. Bass. New product diffusion models in marketing: A review and directions for research. *Journal of Marketing*, 54(1):1–26, 1990.

[24] E. Misiolek and D. Z. Chen. Two flow network simplification algorithms. *Inf. Process. Lett.*, 97(5):197–202, 2006.

[25] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1), 1978.

[26] A. Quirin et al. A new variant of the pathfinder algorithm to generate large visual science maps in cubic time. *Inf. Process. Manage.*, 44(4), 2008.

[27] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *8th Int. Conf. on Knowledge Discovery and Data Mining (KDD'02)*.

[28] K. Saito, R. Nakano, and M. Kimura. Prediction of information diffusion probabilities for independent cascade model. In *12th Int. Conf. on Knowledge-Based Int. Information and Engineering Systems (KES'08)*.

[29] E. Serrano, A. Quirin, J. A. Botía, and O. Cordón. Debugging complex software systems by means of pathfinder networks. *Inf. Sci.*, 180(5):561–583, 2010.

[30] M. A. Serrano, M. Boguñá, and A. Vespignani. Extracting the multiscale backbone of complex weighted networks. *Proc. of the National Academy of Sciences*, 106(16):6483–6488, April 2009.

[31] H. Toivonen, S. Mahler, and F. Zhou. A framework for path-oriented network simplification. In *9th Int. Symp. on Advances in Intelligent Data Analysis (IDA'10)*.

[32] T. Valente. *Network Models of the Diffusion of Innovations*. Hampton Press, 1955.

[33] F. Zhou, S. Mahler, and H. Toivonen. Network simplification with minimal loss of connectivity. In *9th Int. Symp. on Advances in Intelligent Data Analysis (IDA'10)*.

[34] A. Arenas, J. Duch, A.Fernandez, and S. Gomez. Size reduction of complex networks preserving modularity. In *New Journal of Physics 9 (2007) 176*.

[35] W.S. Fung, R. Hariharan, N.J.A. Harvey, D. Panigrahi. A General Framework for Graph Sparsification. In *33th Symp. on Theory of Computing (STOC 2011)*.