

## Activity Preserving Graph Simplification

Francesco Bonchi ·  
Gianmarco De Francisci Morales ·  
Aristides Gionis · Antti Ukkonen

Received: date / Accepted: date

**Abstract** We study the problem of simplifying a given directed graph by keeping a small subset of its arcs. Our goal is to maintain the connectivity required to explain a set of observed traces of information propagation across the graph. Unlike previous work, we do not make any assumption about an underlying model of information propagation. Instead, we approach the task as a combinatorial problem.

We prove that the resulting optimization problem is **NP**-hard. We show that a standard greedy algorithm performs very well in practice, even though it does not have theoretical guarantees. Additionally, if the activity traces have a tree structure, we show that the objective function is supermodular, and experimentally verify that the approach for size-constrained submodular minimization recently proposed by Nagano et al. (2011) produces very good results. Moreover, when applied to the task of reconstructing an unobserved graph, our methods perform comparably to a state-of-the-art algorithm devised specifically for this task.

**Keywords** Information propagation, graph sparsification, submodular minimization

### 1 Introduction

Simplifying graphs is a problem that has been studied extensively. The bulk of this literature considers only structural properties of the graph. For example, a spanner of a given graph  $G$  is a sparse subgraph of  $G$ , so that distances

---

F. Bonchi, G. De Francisci Morales  
Yahoo! Research Barcelona, Spain.  
E-mail: {bonchi,gdfm}@yahoo-inc.com

A. Gionis, A. Ukkonen  
HIIT / Aalto University, Finland.  
E-mail: {aristides.gionis,antti.ukkonen}@aalto.fi

between pairs of nodes in  $G$  are not distorted by much when measured on the spanner subgraph (Peleg and Schäffer 1989). Other problem formulations involve sparsifying a graph with the objective of preserving the flow properties (Misiolek and Chen 2006), the cuts (Fung et al 2011), the connectivity (Zhou et al 2010), or the community structure of the original graph (Arenas et al 2007).

In this paper we study the problem of simplifying a graph while maintaining the connectivity required to *explain a given set of observed activity traces over the graph*. Similar to the work by Mathioudakis et al (2011), our goal is to identify the most important pathways of the graph for understanding observed information propagation across its nodes. However, in contrast to previous work, we take a *model-free* approach. This gives the methods proposed in this paper wider applicability, because different types of information propagation, such as sharing a link on Facebook on the one hand, or adopting a product on the other, are unlikely to follow the same principles of contagion. Therefore, a single model may not fit all applications. We suggest to overcome this problem by not using a model at all. To summarize, sparsifying a graph on the basis of observed activation traces can be useful in many applications, among others:

**Information propagation in social networks.** In this case, the graph is the social network and entities represent information memes that spread in the network. Finding the social ties that are the most important information-diffusion channels can be a useful tool to answer questions such as “what distinguishes the way politics memes and sport memes propagate?” or “what is the structural difference between the backbones of actual news and false rumors?”

**Network reconstruction (Gomez-Rodriguez et al 2010).** In some cases we might only observe a set of traces, but not the arcs of the underlying graph. Our methods for finding the most important arcs can also be used to reconstruct an unobserved graph by sparsifying an imaginary complete graph on the nodes.

**Website usage analysis and re-organization (Srikant and Yang 2001).**

The graph represents webpages and links between these, and every trace corresponds to the activity of a certain user in the website. By sparsifying the graph we can find the most important hyperlinks, which might provide important information regarding user behavior inside the website and reachability of different parts of the website.

**Information filtering and personalization.** Given the amount of information received by the average user of on-line social networks such as **facebook** or **twitter**, simplifying the social graph on the basis of previously observed traces can be used to identify the important connections of a user, and e.g. give preference to messages arriving along these connections.

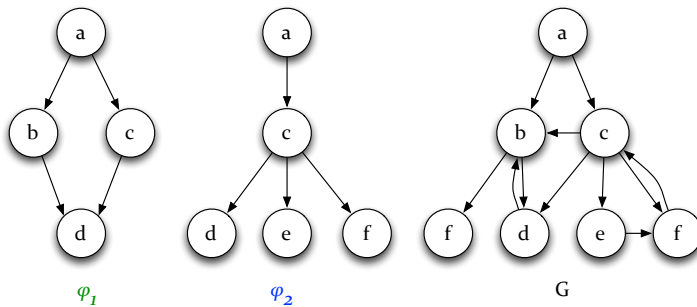
**Influence maximization.** Graph simplification can be used as a data reduction mechanism in the seed selection problem (Kempe et al 2003). In our previous work (Mathioudakis et al 2011) we show that sparsifying the

social graph on the basis of past traces yields significant improvements in terms of efficiency and scalability, while sacrificing little in terms of quality.

In this paper we consider the following problem, illustrated by the simple example in Fig. 1. We are given a *directed graph*  $G = (N, A)$ , and a database of activation *traces*. Our task is to select a small subset of arcs from  $A$  that maintain connectivity in the traces. Fig. 1 shows a directed graph  $G$  together with two traces, denoted  $\phi_1$  and  $\phi_2$ . (In real applications we can have tens of thousands of traces.)

The graph  $G$  represents relations between nodes, and the traces correspond to, e.g., information cascades in the graph  $G$ . Each trace is a directed acyclic graph (DAG) defined on a subset of  $N$ , and it captures the temporal order in which the nodes got activated. A trace  $\phi$  has an arc from node  $u$  to node  $v$  if the arc  $(u, v)$  exists in  $A$  and the node  $u$  got activated before node  $v$ . In trace  $\phi_1$  of Fig. 1 node  $b$  must have been activated before node  $c$ , because the arc  $(c, b)$  does not exist in  $\phi_1$  even though it is an arc of  $G$ . The nodes of a trace with zero in-degree are called *sources*. In Fig. 1 both traces  $\phi_1$  and  $\phi_2$  have only a single source: the node  $a$ .

Given a simplified graph, the *coverage* of a trace is the number of nodes that can be reached from at least one source along a path in the simplified graph. Note that the sources do not contribute to the coverage. Our task is to simplify the input graph  $G$  by keeping only a small subset of its arcs so that the coverage over all input traces is maximized. In the toy example of Fig. 1, the subgraph consisting only of the arcs  $(a, c)$ ,  $(c, f)$ , and  $(b, d)$  has a coverage of 1 in trace  $\phi_1$ , and a coverage of 2 in trace  $\phi_2$ , for a total coverage of 3. The subgraph with arcs  $(a, c)$  and  $(c, d)$  has a total coverage of 4, as both  $\phi_1$  and  $\phi_2$  have a coverage of 2. There are thus two objectives to optimize: minimize the number of arcs in the simplified graph and maximize the coverage. In this example we would prefer the latter subgraph, because it is smaller and yields a larger coverage. Observe that we do not assume any particular generative process for the traces when computing coverage.



**Fig. 1** A directed graph  $G$  and two traces,  $\phi_1$  and  $\phi_2$ .

To avoid a multi-objective optimization problem, we can constrain one quantity and optimize the other, and thus obtain two complementary problem formulations. We name the two problems MAXCOVER and MINARCSET, and we show they are **NP**-hard.

We show that when traces are general DAGs, coverage is neither submodular nor supermodular. As a consequence the standard greedy heuristic that adds arcs by maximizing marginal gain in coverage has no provable quality guarantees. Nonetheless, our empirical evaluation shows that Greedy gives very good solutions in practice.

In some applications the parent of a node in a trace is unique. This can happen for instance when information propagates by an explicit repost *via* a certain neighboring node. We show that if all input traces are directed *trees* rooted at the sources, then the coverage function is supermodular. In Fig. 1 the trace  $\phi_2$  is such a tree, but the trace  $\phi_1$  is not. Since maximizing a supermodular function is equivalent to minimizing a submodular function, we develop an algorithm based on recent advances in size-constrained submodular minimization by Nagano et al (2011). This algorithm, called MNB, gives *optimal solutions* for *some* sizes of the simplified graph. These sizes, however, can not be specified in advance, but are part of the output of the algorithm.

We implemented both methods and applied them on real datasets. The main conclusions drawn from our empirical evaluation are the following.

- The greedy algorithm is a reliable method that gives good results and scales gracefully. Over all our datasets, Greedy achieves a performance that is at least 85% of the optimal.
- With tree-shaped traces MNB is the most efficient algorithm and outperforms Greedy by up to two orders of magnitude in running time.
- We apply our algorithms to the task of reconstructing an unobserved graph based on observed propagations by simplifying a complete graph (clique). The empirical evaluation suggests that our methods slightly outperform NetInf (Gomez-Rodriguez et al 2010), an algorithm specifically designed for this network-reconstruction task.
- We use our methods to simplify a graph where the arcs contain social-influence information, as a preprocessing step before influence maximization, as done by Mathioudakis et al (2011). When compared to their algorithm, our methods perform reasonably well, although our methods are at a disadvantage as explained in detail in Section 5.

The rest of the paper is organized as follows. In Section 2 we review related work. In Section 3 we formally define our problem and study the properties of our objective function. Our algorithms are discussed in Section 4 and our empirical evaluation is presented in Section 5. Finally, Section 6 concludes by outlining open problems and future research directions.

## 2 Related work

Conceptually, our work contributes to the literature on network simplification, the goal of which is to identify subnetworks that preserve properties of a given network. Toivonen et al (2010) as well as Zhou et al (2010), for instance, prune arcs while keeping the quality of best paths between all pairs of nodes, where quality is defined on concepts such as shortest path or maximum flow. Misiolek and Chen (2006) 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 (Elkin and Peleg 2005). In *pathfinder networks* (Quirin et al 2008, Serrano et al 2010) the approach is to select weighted arcs that do not violate the triangular inequality. Fung et al (2011) study *cut-sparsifiers*, i.e., subsets of arcs that preserve cuts up to a multiplicative error. Serrano et al (2009) and Foti et al (2011) focus on weighted networks and select arcs that represent statistically significant deviations with respect to a null model. In a similar setting, Arenas et al (2007) select arcs that preserve modularity, a measure that quantifies quality of community structure.

The approach we take in this paper is substantially different from the work discussed above. The main difference is that our problem of simplification is defined in terms of *observed activity* in the network, and not only in terms of structural properties of the network. A similar approach is taken by Gomez-Rodriguez et al (2011; 2010) and Mathioudakis et al (2011). Gomez-Rodriguez et al (2011; 2010) assume that connections between nodes are unobserved, and use observed traces of activity to infer a sparse, “hidden” network of information diffusion. Mathioudakis et al (2011) instead focus on sparsifying an available network. Both lines of research build on a probabilistic *propagation model* that relies on estimated “influence probabilities” between nodes. In contrast, we define graph simplification as a combinatorial problem that *does not assume any underlying propagation model*. This is an important contribution, because the accuracy of propagation models to describe real-world phenomena has not been shown conclusively. Our empirical evaluation suggests that our model-free methods compare well with the model-based approaches by Gomez-Rodriguez et al (2010) and Mathioudakis et al (2011).

## 3 Problem definition

In this section we provide formal definitions of the concepts that we already introduced in Section 1. Our examples come from the context of social media, but we want to emphasize that the results and algorithms are agnostic of the application, and can easily be used with suitable data from any domain.

**Underlying graph.** The first input to our problem is a *directed graph*  $G = (N, A)$ . The direction of the arcs in  $A$  indicates the direction in which information propagates. As an example, in Twitter the arc  $(u, v)$  belongs to  $A$

whenever user  $v$  follows user  $u$ , and therefore information can potentially flow from  $u$  to  $v$ .

In some cases it is not possible to observe the graph. For example, when studying the blogosphere we may assume that any blog can influence any other blog even though there are no explicit links in between. When the graph is not observable, we simply assume that the set of arcs  $A$  is *complete*, i.e.,  $A = N \times N$ .

**Traces over the graph.** The second input to our problem is a set  $\Pi$  of *traces* of information propagation over  $G$ . Each trace  $\phi \in \Pi$  corresponds to a *different piece of information*. When a piece of information reaches a node, we say the node becomes *activated* in the corresponding trace. For example, in Twitter a trace could correspond to some particular URL, and a user is activated whenever she tweets that URL. By considering different URLs, we obtain a set of different traces. Note that a node can become activated only once in the same trace.

We represent traces as directed acyclic graphs (DAGs) that capture the temporal order in which the node activations occur in  $G$ . Formally, a trace  $\phi$  is represented by a DAG  $G_\phi = (N_\phi, A_\phi)$ , where  $N_\phi$  is the set of activated nodes, and  $A_\phi$  is a set of arcs that indicate *potential information flow* in  $G$ . That is, we have  $(u, v) \in A_\phi$  whenever  $(u, v)$  is an arc of  $G$ , both  $u$  and  $v$  belong to  $N_\phi$ , and  $u$  got activated before  $v$  in  $\phi$ . Note that  $(u, v) \in A_\phi$  does not imply that  $v$  indeed was “influenced” by  $u$  in any way. The trace graph  $G_\phi$  merely indicates the *possible* information pathways of a trace  $\phi$  in  $G$ . Finally, nodes in  $N_\phi$  with no incoming arcs in  $A_\phi$  are called *trace sources*, and belong to the set  $S_\phi$ . A node is a source if it is among the first nodes of a trace to become active, meaning it has no neighbors that got activated earlier in the same trace.

As detailed in Section 3.1, when traces have a tree structure they possess interesting properties that we can leverage in our algorithms. Before formalizing the problem, we provide motivation why considering tree traces is an interesting special case. There are many cases where we can unambiguously observe the information propagation paths. For example, when considering re-tweeting activity in Twitter, either the string ‘RT@’ is present in the tweet text, followed by the username of the information source, or an explicit `retweet_source` field is available via the API. Likewise in Facebook, when users share, say, links, the source is always explicitly mentioned. As our objective is to provide methods to study information propagation in networks, using tree traces is a very meaningful abstraction for many real-world applications.

**Definition 1** (TREE TRACES) Let  $G_\phi = (N_\phi, A_\phi)$  represent a trace  $\phi$  with set of sources  $S_\phi$ . A trace  $\phi$  is a *tree trace* if it is the disjoint union of directed trees rooted at the source nodes  $S_\phi$ .

According to Definition 1, a trace is a tree trace when the DAG has one connected component for each source. Without loss of generality, if the trace has multiple sources we can consider each as a separate trace. Therefore, hereafter we refer to a tree trace as a single directed tree rooted at the source.

**Objective function.** Our objective function is the *coverage* attained by a set of arcs  $A'$ . A precise definition of coverage is given below. In less formal terms, coverage refers to the total number of nodes in every trace that are *reachable* from one of the sources of the trace by using only arcs in  $A'$ .

**Definition 2** (REACHABILITY) Let  $G_\phi = (N_\phi, A_\phi)$  represent a trace  $\phi$  with set of sources  $S_\phi$ , and let  $u \in N_\phi \setminus S_\phi$ . For a set of arcs  $A' \subseteq A$  we define the variable  $r(A', u \mid \phi)$  to be 1 if there is a path in  $G_\phi$  from any node in  $S_\phi$  to  $u$  using only arcs in  $A'$ . If no such path exists we define  $r(A', u \mid \phi)$  to be 0.

**Definition 3** (COVERAGE) Consider a graph  $G = (N, A)$  and a set of traces  $\Pi$ . We define the *coverage* of a set of arcs  $A' \subseteq A$  as

$$C_\Pi(A') = \sum_{\phi \in \Pi} \sum_{u \in N_\phi \setminus S_\phi} r(A', u \mid \phi). \quad (1)$$

For simplicity of notation, and when the set of traces is implied by the context, we also write  $C(A')$  for  $C_\Pi(A')$ .

Our goal is to find a small set of arcs that has large coverage. Thus, there are two quantities to optimize: the number of arcs and the coverage achieved. We can constrain one quantity and optimize the other, thus obtaining two complementary problem formulations.

**Problem 1** (MAXCOVER) Consider a graph  $G = (N, A)$ , a set of traces  $\Pi$ , and an integer  $k$ . Find a set of arcs  $A' \subseteq A$  of cardinality at most  $k$  that maximizes the coverage  $C_\Pi(A')$ .

**Problem 2** (MINARCSET) Consider a graph  $G = (N, A)$ , a set of traces  $\Pi$ , and a coverage ratio  $\eta \in [0, 1]$ . Find a set of arcs  $A' \subseteq A$  of minimum cardinality such that  $C_\Pi(A') \geq \eta \sum_{\phi \in \Pi} |N_\phi \setminus S_\phi|$ .

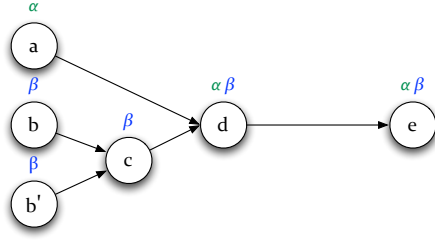
### 3.1 Problem characterization

Next we discuss the complexity of MAXCOVER and MINARCSET, and study the properties of the coverage function.

We first note that MAXCOVER and MINARCSET are equivalent: for instance, if there is an algorithm  $\mathcal{A}$  that solves the MAXCOVER problem, then we can solve the MINARCSET problem with calls to  $\mathcal{A}$  while performing a binary search on the parameter  $k$ . Indeed both problems, are optimization versions of the same decision problem,  $(k, \eta)$ -COVER.

**Problem 3** ( $(k, \eta)$ -COVER) Consider a graph  $G = (N, A)$ , a set of traces  $\Pi$ , a number  $k$ , and a coverage ratio  $\eta \in [0, 1]$ . Does there exist a set of arcs  $A' \subseteq A$  such that  $|A'| \leq k$  and  $C_\Pi(A') \geq \eta \sum_{\phi \in \Pi} |N_\phi \setminus S_\phi|$ .

**Theorem 1** *The  $(k, \eta)$ -COVER problem is NP-hard.*



**Fig. 2** A counterexample that shows that the coverage measure is neither submodular nor supermodular.

*Proof.* We obtain a reduction from the SETCOVER problem, which is an **NP**-complete problem defined as follows. A problem instance for SETCOVER is specified by a ground set  $U = \{1, \dots, n\}$  of  $n$  elements, a collection  $\mathcal{X} = \{X_1, \dots, X_m\}$  of subsets of  $U$ , and a number  $k$ . A solution to SETCOVER is provided by a sub-collection  $\mathcal{X}' \subset \mathcal{X}$  of at most  $k$  sets that *covers* the ground set  $U$ , i.e.,  $\bigcup_{X_i \in \mathcal{X}'} X_i = U$ . The SETCOVER problem is to decide whether there exists a solution, given a problem instance.

We now proceed to build an instance of the  $(k, \eta)$ -COVER problem starting from an instance of SETCOVER. Given the ground set  $U$  and the set collection  $\mathcal{X} = \{X_1, \dots, X_m\}$ , we build a graph  $G = (N, A)$  with  $m + 1$  nodes where  $N = \{v_1, \dots, v_m, w\}$ , i.e., there is a node  $v_i$  for each set  $X_i$  and one additional node  $w$ . The set of arcs of  $G$  is  $A = \{(v_i, w)\}_{i=1}^m$ , i.e., there is an arc from each node  $v_i$  to  $w$ . The set of traces  $\Pi$  will contain the trace  $\phi_u$  for every element  $u \in U$ , meaning  $|\Pi| = |U|$ . The DAG of  $\phi_u$  has an arc from  $v_i$  to  $w$  for every  $X_i$  that contains  $u$ , that is,  $A_{\phi_u} = \{(v_i, w) \mid u \in X_i\}$ . Note that in every  $\phi_u$  every vertex except  $w$  is a source node, and is thus always reachable, i.e.  $|N_{\phi} \setminus S_{\phi}| = 1$ . Hence, the maximum coverage is equal to the size of  $U$ . Finally, we set the arc budget  $k$  equal to the parameter  $k$  of the SETCOVER instance, and  $\eta = 1$ , i.e., we seek complete cover.

Clearly any solution  $A'$  to the  $(k, \eta)$ -COVER problem can be directly mapped to a solution of the original SETCOVER problem instance we were given: include the set  $X_i$  to the SETCOVER solution if the arc  $(v_i, w)$  belongs to the  $(k, \eta)$ -COVER solution. Any solution  $A'$  to  $(k, \eta)$ -COVER that is at most of size  $k$  and has coverage ratio  $\eta = 1$  (and thus coverage  $|U|$ ) must map to a SETCOVER solution that is at most of size  $k$  and covers the entire universe  $U$ .  $\square$

A direct corollary of Theorem 1 is that both problems, MAXCOVER and MINARCSET, are **NP**-hard. We now examine whether the coverage measure is *submodular* or *supermodular*. The reason is that there is extensive literature on optimizing functions with these properties: for example, greedy strategies provide approximation guarantees when maximizing submodular functions (Nemhauser et al 1978). Recall that, given a ground set  $U$ , a function



$f : 2^U \rightarrow \mathbb{R}$  is submodular if it satisfies the *diminishing-returns* property

$$f(X \cup \{z\}) - f(X) \geq f(Y \cup \{z\}) - f(Y), \quad (2)$$

for all sets  $X \subseteq Y \subseteq U$  and  $z \in U \setminus Y$ . The function  $f$  is supermodular if  $-f$  is submodular. We first show that the coverage objective function is neither submodular nor supermodular.

**Lemma 1** *When  $\Pi$  may contain general DAGs, the coverage function  $C_\Pi$  is neither submodular nor supermodular.*

*Proof.* We prove the theorem by providing a counterexample for each case. Consider Figure 2 illustrating a graph  $G = (N, A)$ , with node set  $N = \{a, b, b', c, d, e\}$ , and arc set  $A = \{(a, d), (b, c), (b', c), (c, d), (d, e)\}$ . Consider a set of two traces  $\Pi = \{\alpha, \beta\}$ , with DAGs  $G_\alpha = (N_\alpha, A_\alpha)$  and  $G_\beta = (N_\beta, A_\beta)$ , where  $N_\alpha = \{a, d, e\}$ ,  $A_\alpha = \{(a, d), (d, e)\}$ ,  $N_\beta = \{b, b', c, d, e\}$ , and  $A_\beta = \{(b, c), (b', c), (c, d), (d, e)\}$ . The source sets are  $S_\alpha = \{a\}$ , and  $S_\beta = \{b, b'\}$ .

Consider the following sets of arcs:  $X = \{(a, d)\} \subset Y = \{(a, d), (c, d), (d, e)\}$  and  $z = (b, c)$ . We have

$$C_\Pi(X \cup \{z\}) - C_\Pi(X) = 2 - 1 < 5 - 2 = C_\Pi(Y \cup \{z\}) - C_\Pi(Y),$$

which contradicts Inequality (2) and proves that the coverage function is not submodular.

Next, let  $X = \{(a, d), (c, d), (d, e)\} \subset Y = \{(a, d), (b, c), (c, d), (d, e)\}$ , and  $z = (b', c)$ . In this case we have

$$C_\Pi(X \cup \{z\}) - C_\Pi(X) = 5 - 2 > 5 - 5 = C_\Pi(Y \cup \{z\}) - C_\Pi(Y),$$

which conforms to Inequality (2) and proves that the coverage function is not supermodular.  $\square$

As a result of Lemma 1, the greedy heuristic does not have any performance guarantee. On the other hand, the coverage function is supermodular when we can attribute the activation of a node to precisely one of its neighbors in every trace in  $\Pi$ . This is true if and only if  $\Pi$  only contains tree traces. For example, the trace  $\phi_1$  in Figure 1 is a tree trace, while  $\phi_2$  of the same figure is not.

**Theorem 2** *The coverage function  $C_\Pi$  is supermodular when every trace  $\phi \in \Pi$  is a tree trace.*

*Proof.* We show that the coverage function is supermodular for one trace  $\phi$ , by letting  $\Pi = \{\phi\}$ . Since  $C_\Pi$  is additive with respect to traces, and the sum of supermodular functions is supermodular, the property holds for sets of traces.

Let  $Z$  be a set of arcs, and let  $(u, v)$  be an arc not in  $Z$ . Consider the tree  $G_\phi$  of trace  $\phi$  and denote by  $R(u, v \mid Z)$  the set of nodes in  $G_\phi$  that become reachable from any node in  $S_\phi$  when the arc  $(u, v)$  is added to  $Z$ . Note that if  $u$  is not reachable from  $S_\phi$  by using arcs in  $Z$ , then the set  $R(u, v \mid Z)$  is empty. However, if  $u$  is reachable from  $S_\phi$  given  $Z$ ,  $R(u, v \mid Z)$  will consist

of the node  $v$ , as well as every node that is reachable from  $v$  given  $Z$ . If any vertex that is reachable from  $v$  given  $Z$  were also reachable from  $S_\phi$  without the arc  $(u, v)$ , the DAG  $G_\phi$  would not be a tree. We have thus

$$|R(u, v \mid Z)| = C_\Pi(Z \cup \{(u, v)\}) - C_\Pi(Z). \quad (3)$$

To show supermodularity, consider two sets of arcs  $X$  and  $Y$ , with  $X \subseteq Y$ , and let  $(u, v)$  be an arc not in  $X$  or  $Y$ . Since  $X \subseteq Y$  it is easy to show that

$$|R(u, v \mid X)| \leq |R(u, v \mid Y)|. \quad (4)$$

The result follows by combining Equations (3) and (4).  $\square$

Lemma 1 indicates that optimizing the coverage function in the general case is a difficult problem. However, for the special case of the problem when all traces are trees, we use the result of Theorem 2 to develop an efficient and nontrivial algorithm, which is described in the next section. Furthermore, our techniques can be useful even when the observed traces are not trees, but general DAGs. This can be done by first converting the observed DAGs into trees via an application-dependent heuristic. Although we have not experimented with such ideas for this paper, we think that it is a fruitful research direction.

## 4 Algorithms

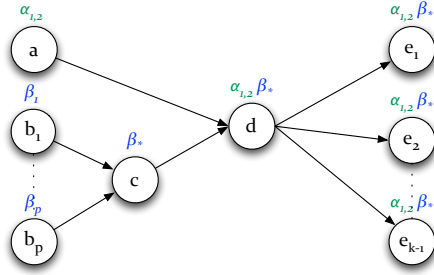
We now describe two algorithms for graph simplification. Both algorithms are developed for the MAXCOVER problem. As already noted, any algorithm for MAXCOVER can be used to solve MINARCSET by applying binary search on the coverage score.

In addition to the algorithms discussed below, we have also formulated MAXCOVER as an integer program. In Section 5 we apply a standard linear programming solver to find optimal solutions for small problem instances. We also experiment with a simple rounding technique to find good solutions given a fractional solution to the linear program. Whether more sophisticated algorithms based on the linear programming formulation can be devised is an interesting open question.

### 4.1 Greedy algorithm

The first algorithm, *Greedy*, is the standard greedy heuristic for covering problems. It starts with the empty solution set  $A' = \emptyset$ . At each step it adds to  $A'$  the arc  $(u, v)$  that yields the largest marginal gain to the current coverage. The marginal gain for an arc  $(u, v)$  is defined as  $\rho_{A'}(u, v) = C_\Pi(A' \cup \{(u, v)\}) - C_\Pi(A')$ . The algorithm terminates when it reaches a solution of size  $k$ , i.e.,  $|A'| = k$ .

Lemma 1 suggests that there is no immediate proof that *Greedy* is an approximation algorithm for our problem. In fact, we can construct problem



**Fig. 3** An example where Greedy reaches only a  $O(\frac{1}{k})$ -factor approximation.

instances for which Greedy gives bad solutions. In the lemma below, an algorithm with *approximation factor* equal to  $c$  is an algorithm that, for any problem instance, guarantees to find a solution whose value is at least  $c$  times the value of the optimal solution in that instance.

**Lemma 2** *When every trace  $\phi \in \Pi$  is a tree trace, the approximation factor of Greedy is no better than  $O(\frac{1}{k})$ .*

*Proof.* An example that satisfies the result of the lemma is shown in Figure 3. In this example, we assume that the arc budget  $k$  is an even number, and we set  $p = k/2$ . The graph contains the nodes  $a, c$  and  $d$ , as well as the nodes  $b_1, \dots, b_p$  all of which have an arc to node  $c$ , and the nodes  $e_1, \dots, e_{k-1}$  all of which have an arc from node  $d$ . We assume *two* different traces (named  $\alpha_1$  and  $\alpha_2$ ) observed on the nodes  $\{a, d, e_1, \dots, e_{k-1}\}$ , and  $p$  different traces (named  $\beta_i$ ) each of which is observed on the nodes  $\{b_i, c, d, e_1, \dots, e_{k-1}\}$ , for  $i = 1, \dots, p$ .

It is easy to see that Greedy selects as solution the  $k$  arcs  $\{(a, d), (d, e_1), \dots, (d, e_{k-1})\}$  and achieves a coverage score of  $2k$ . On the other hand, the optimal solution consists of the  $k$  arcs  $\{(b_1, c), \dots, (b_p, c), (c, d), (d, e_1), \dots, (d, e_{p-1})\}$  and achieves a coverage score of  $p(p+1)$ . By our selection of  $p$ , it follows that the relative performance of Greedy with respect to the optimal is no better than  $O(\frac{1}{k})$ .  $\square$

This result only implies that there are adversarial examples for which Greedy has poor performance. Furthermore, note that the traces in the example of Figure 3 are trees and therefore the coverage function is supermodular. *The worst-case performance of Greedy is thus unaffected by supermodularity.* However, our empirical evaluation shows that in practice Greedy gives solutions of good quality in all the datasets we experimented.

## 4.2 Minimum-norm base algorithm

Our second algorithm is based on mapping our cover-maximization problem to a problem of minimizing a submodular function under size constraints. Minimization of submodular functions has a rich theory (Fujishige 2005) much of

which utilizes the *basic polyhedron* associated with the submodular function. A family of algorithms for minimizing submodular functions is based on finding a *minimum-norm base* vector on the basic polyhedron. Our method, called MNB, is an instantiation of such an algorithm. We start our discussion on MNB by reviewing the related theory and then we present the details of the algorithm for the specific problem we consider in this paper.

First recall from Section 3 that a function  $f : 2^U \rightarrow \mathbb{R}$ , defined over subsets of a ground set  $U$ , is submodular if it satisfies Equation (2). Given a submodular function  $f$  and an integer  $k$ , the size-constrained minimization problem is to find a set  $X \subseteq U$  that

$$\text{minimizes } f(X) \text{ subject to } |X| \leq k. \quad (5)$$

We note that minimizing a submodular function without size constraints is a polynomially-time solvable problem; alas, the faster strongly-polynomial algorithm, by Iwata and Orlin (2009), has complexity  $O(n^5 F + n^6)$ , where  $F$  is the time required to evaluate the function  $f$ . On the other hand, the size-constrained version of the problem, as defined in problem (5), is also an **NP**-hard problem, with an  $o(\sqrt{\frac{n}{\ln n}})$  in-approximability lower bound (Svitkina and Fleischer 2011). In this paper we follow the recently-developed approach of Nagano et al (2011). Even though Nagano et al. do not provide an approximation algorithm, they are able to show how to obtain *optimal* solutions for *some* values of the size constraint  $k$ . These values can not be specified in advance, however, but are a part of the output of the algorithm. Turns out that this property is not a limitation in practice.

#### 4.2.1 Basic definitions

Consider the space  $\mathbb{R}^m$ , where  $m = |U|$ , that is, each dimension of  $\mathbb{R}^m$  is associated with one and only one element  $i \in U$ . Let  $\mathbf{x}$  denote a vector in  $\mathbb{R}^m$ . For any  $X \subseteq U$ ,  $x(X)$  is the total *weight* those elements of  $\mathbf{x}$  that are at coordinates specified by  $X$ , that is,  $x(X) = \sum_{i \in X} x_i$ . Given a submodular function  $f : 2^U \rightarrow \mathbb{R}$ , we can consider the *submodular polyhedron*  $P(f)$  and the *base polyhedron*  $B(f)$  that are both subsets of  $\mathbb{R}^m$ . In particular, these polyhedra are defined as

$$\begin{aligned} P(f) &= \{\mathbf{x} \in \mathbb{R}^m \mid x(X) \leq f(X), \text{ for all } X \subseteq U\}, \\ B(f) &= \{\mathbf{x} \in \mathbb{R}^m \mid \mathbf{x} \in P(f) \text{ and } x(U) = f(U)\}. \end{aligned}$$

Consider now the *minimum-norm base* vector  $\mathbf{x}^*$  on  $B(f)$ , that is,

$$\mathbf{x}^* = \arg \min \left\{ \sum_{i=1}^m x_i^2 \mid \mathbf{x} \in B(f) \right\}.$$

From the general theory of submodular functions it is known that the minimum-norm base  $\mathbf{x}^*$  is closely related to the problem of unconstrained submodular

minimization (Fujishige 2005). In particular, the negative coordinates of  $\mathbf{x}^*$  specify the minimizer set  $X$  of  $f$ . If we define

$$X_- = \{i \in U \mid x_i^* < 0\}, \text{ and } X_0 = \{i \in U \mid x_i^* \leq 0\},$$

then  $X_-$  is the unique minimal minimizer of  $f$ , and  $X_0$  is the unique maximal minimizer of  $f$ .

#### 4.2.2 The SSM algorithm

Recently, Nagano et al (2011) show how the minimum-norm base vector  $\mathbf{x}^*$  can also be used to give optimal solutions to the problem of size-constrained submodular minimization, for some possible values of the budget  $k$ . Their algorithm, named SSM, consists of the following two steps:

1. Compute the minimum-norm base  $\mathbf{x}^* \in B(f)$ .
2. Let  $\xi_1 < \dots < \xi_L$  denote all distinct values of the coordinates of  $\mathbf{x}^*$ . Return the sets  $T_0 = \emptyset$  and  $T_j = \{i \in U \mid x_i^* \leq \xi_j\}$ , for all  $j = 1, \dots, L$ .

Nagano et al. show the following surprising result.

**Theorem 3 (Nagano et al (2011))** *Let  $T_0, T_1, \dots, T_L \subseteq U$  be the sets returned by the SSM algorithm. Then, for all  $j = 0, 1, \dots, L$ , the set  $T_j$  is the optimal solution for the size-constrained submodular minimization problem defined in (5) for  $k = |T_j|$ .*

Given the result of Theorem 3 one can find optimal solutions to the size-constrained minimization problem for a number of values of  $k$ , which, however, are prescribed in the structure of the minimum-norm base  $\mathbf{x}^*$  and not specified in the input. Observe that the algorithm will find the optimal solution for exactly as many different  $k$  as there are distinct values in the vector  $\mathbf{x}^*$ . In the worst case all elements of  $\mathbf{x}^*$  have the same value, which means that the algorithm has only found the solution that corresponds to choosing every item of the universe  $U$ . However, in practice we observe the vector  $\mathbf{x}^*$  to contain several distinct elements that allow us to construct optimal solutions for different values of the size constraint  $k$ .

From the computational point of view, the heart of the SSM algorithm is finding the minimum-norm base  $\mathbf{x}^*$ . Recall that  $\mathbf{x}^*$  is a point of the  $B(f)$  polyhedron, which implies that  $\mathbf{x}^*$  will be in one of the extreme points of the polyhedron. The problem of finding a minimum-norm vector in a polyhedron defined by a set of linear constraints can be solved by the standard Wolfe algorithm (Fujishige 2005, Wolfe 1976), which is not known to be polynomial, but performs very well in practice. In Section 4.3 we provide a brief outline of the algorithm, mostly in reference to implementing the algorithm efficiently in our setting. More details on the algorithm can be found elsewhere (Fujishige 2005).

### 4.2.3 The MNB algorithm: applying SSM for MAXCOVER

Next we discuss how to apply the theory reviewed above on our problem definition. The main observation stems from Theorem 2, namely from the fact that the cover measure  $C_{\Pi}$  is supermodular when all trace DAGs are trees. Thus, in the case that  $\Pi$  contains only tree traces, the MAXCOVER problem asks to *maximize a supermodular function*  $C_{\Pi}$  over sets of size  $k$ . This is equivalent to *minimizing the submodular function*  $-C_{\Pi}$  over sets of size  $k$ . We can solve the latter problem by direct application of the SSM algorithm, which only requires evaluations of the submodular function  $-C_{\Pi}$ . Note that each dimension of the minimum-norm base  $\mathbf{x}^*$  corresponds to one and only one arc in  $A$ . By Theorem 3 we can read the optimal arc subsets  $T_j$  from  $\mathbf{x}^*$ .

Additionally, Theorem 3 implies that for the values of  $k$  not falling on any of the values  $|T_j|$ , for  $j = 1, \dots, L$ , the algorithm does not yield an optimal solution. We address this shortcoming by introducing a greedy strategy to *extend* one of the sets returned by the minimum-norm base approach to a set with size exactly  $k$ . In particular, let  $j^* \leq L$  be the largest index among all  $j = 1, \dots, L$  such that  $|T_{j^*}| \leq k$ . The set  $T_{j^*}$  is a feasible solution to our problem since it has cardinality less than  $k$ . We can then use the set  $T_{j^*}$  as an initial solution, and extend it by iteratively adding arcs until we reach at a solution set with cardinality exactly  $k$ . Adding arcs to the initial set  $T_{j^*}$  is done by the greedy strategy used in Greedy. The coverage obtained by the final solution is at least as good as the coverage obtained by  $T_{j^*}$ . The MNB algorithm can be summarized as follows:

1. Given the graph  $G = (N, A)$ , the integer  $k$  and  $\Pi$ , run the SSM algorithm with  $-C_{\Pi}$ . Of the solution sets given by SSM, let  $A'$  denote the largest that has size at most  $k$ .
2. If  $|A'| < k$ , run Greedy starting from the set  $A'$ , until  $A'$  contains exactly  $k$  arcs.
3. Return  $A'$ .

### 4.3 Comparing Greedy and MNB

The Greedy algorithm requires computing the marginal gain

$$\rho_{A'}(u, v) = C_{\Pi}(A' \cup \{(u, v)\}) - C_{\Pi}(A')$$

of an arc  $(u, v)$  with respect to a current solution  $A'$ . In particular, one iteration of Greedy corresponds to a loop that computes the marginal gain  $\rho_{A'}(u, v)$  for all arcs  $(u, v) \in A$ , and selects the arc that yields the largest gain.

It turns out that at the heart of the MNB algorithm there is a similar computation, namely, a loop that computes marginal gains  $\rho(u, v)$  over all arcs  $(u, v) \in A$ . To see how computation of marginal gains enters in the MNB algorithm, we provide a high-level description of the Wolfe algorithm (Fujishige 2005, Wolfe 1976), which computes the minimum-norm base point  $\mathbf{x}^*$  of the

polyhedron  $B(f)$  for the submodular function  $f$ . The Wolfe algorithm traverses *extreme* points of the polyhedron  $B(f)$  until finding a minimum-norm point. It starts with an arbitrary, but feasible, extreme point  $\hat{\mathbf{x}} \in B(f)$  and iteratively moves to other extreme points with smaller norm. In each iteration, the algorithm needs to find a new extreme point  $\hat{\mathbf{x}}$  of  $B(f)$  that minimizes a linear function  $\mathbf{w} \cdot \hat{\mathbf{x}}$ , where  $\mathbf{w} \in \mathbb{R}^m$  is a weight vector appropriately defined. (Since  $\mathbf{w} \in \mathbb{R}^m$  with  $m = |A|$ , we can index the elements of  $\mathbf{w}$  with arcs in  $A$ .)

Now, given the submodular function  $f = -C_{II}$  and a vector  $\mathbf{w}$ , finding the extreme point  $\hat{\mathbf{x}}$  in  $B(f)$  that minimizes  $\mathbf{w} \cdot \hat{\mathbf{x}}$  can be solved efficiently by the following greedy algorithm, due to Edmonds (2003):

1. Sort the coordinates of  $\mathbf{w}$  in order of increasing value, i.e., find an ordering  $\pi : \{1, \dots, m\} \rightarrow A$  such that  $w_{\pi(1)} \leq \dots \leq w_{\pi(m)}$ .
2. For every  $i$ , compute

$$\hat{x}_{\pi(i)} = f(A_{\pi,i}) - f(A_{\pi,i-1}) = -(C_{II}(A_{\pi,i}) - C_{II}(A_{\pi,i-1})),$$

where  $A_{\pi,i} = \{\pi(1), \dots, \pi(i)\} \subseteq A$ , that is, the subset of  $A$  containing the first  $i$  arcs according to the ordering  $\pi$ .

Observe that  $A_{\pi,i} = A_{\pi,i-1} \cup \pi(i)$ , and thus  $\hat{x}_{\pi(i)}$  is simply the negative marginal gain of the arc  $\pi(i) \in A$  when it is added to the set of arcs that precede it in the ordering  $\pi$ . This means that one iteration of the Wolfe algorithm involves almost exactly the same computation that Greedy must carry out on each iteration: a loop that computes marginal gains over all arcs. The only difference is that Greedy computes the marginal gains with respect to a fixed  $A'$ , while the Wolfe algorithm computes them w.r.t. the sets induced by the ordering  $\pi$ , and these sets are slightly different for each  $\pi(i) \in A$ .

From a computational point of view this difference between Greedy and the Wolfe algorithm is in practice negligible. What matters is that in both cases we must compute the marginal gain for every arc in  $A$  given some subset of  $A$ . The complexity of this is linear in the size of  $II$ . We also want to point out that other computations carried out by the Wolfe algorithm only involve solving small systems of linear equations, which is extremely fast in comparison to evaluating  $C_{II}$  even for a relatively small  $II$ . This means that the computationally intensive part is exactly the same for both Greedy and MNB.

To speed up the above computation we employ the following optimization: we iterate over all traces in  $II$ , and we incrementally compute the contribution of each trace  $\phi \in II$  to the marginal gains for every arc in  $A$ . Since traces are in general small compared to the number of arcs in the graph, each iteration can be done very fast. In practice *this optimization leads to two orders of magnitude improvement* over a naïve computation of the cover measure. The same optimization applies both to Greedy and to MNB algorithms.

Finally, note that Greedy computes the marginal gain of every arc  $k$  times, while MNB computes the vector  $\hat{\mathbf{x}}$  as many times as the number of iterations required by the Wolfe algorithm to find a solution that is accurate enough. As the Wolfe algorithm converges fast, the number of iterations required is

usually *significantly smaller than typical values of  $k$* . As a result, in practice the MNB algorithm is significantly faster than Greedy.

## 5 Empirical evaluation

In this section we report the experiments we conducted in order to assess of our methods. We present four kinds of experiments:

1. *Effectiveness* with respect to the basic task of selecting a subset of arcs that maximize the coverage, i.e., our objective function;
2. *Efficiency* of the algorithms;
3. *Network reconstruction*: comparing with the NetInf algorithm (Gomez-Rodriguez et al 2010) for reconstructing an unobserved network;
4. *Influence maximization*: comparing with the Spine algorithm (Mathioudakis et al 2011) in the task of sparsifying the social graph while trying to maximize the spread of influence (Kempe et al 2003).

We start by presenting the experimental settings: implementation of the algorithms and datasets.

**Algorithms.** We implemented the Greedy algorithm in Java, while our MNB implementation is based on the Matlab Toolbox for Submodular Optimization by Krause (2010). In the experiments we stop the iterations in Wolfe’s algorithm after 50 rounds. As discussed above, both Greedy and MNB compute marginal gains in almost exactly the same way. Our implementations of the algorithms can thus share this part of the code. We implemented the marginal gain computation in Java, and use the Java interface of Matlab to call this from within the Wolfe algorithm.

As mentioned before, we also formulated our problem as a linear program. This could possibly be used to devise efficient approximation algorithms by applying existing techniques. These are left as future work, however. In the experiments discussed here we simply solve the linear programs directly.

We consider both an integer program (IP), as well as its relaxed form (LP) that we combine with a rounding scheme that selects the  $k$  edges having the largest weight in the optimal fractional solution. The IP finds optimal solutions (for small problem instances), and it is included in this study mainly to show that the Greedy algorithm tends to find solutions that are close to optimal. For small problem instances solving the IP can be a reasonable approach even in practice. The LP will not produce an optimal solution in general. However, we think it is an alternative worth studying, because it can be solved much more efficiently than the IP, and even simple rounding schemes can produce solutions that are reasonably good in practice.

Efficiency of both linear programming approaches heavily depend on the solver being used. These experiments were carried out using Gurobi<sup>1</sup>, a highly optimized linear programming solver.

---

<sup>1</sup> [www.gurobi.com](http://www.gurobi.com)



**Table 1** Dataset characteristics: numbers are rounded.

| Dataset  | Traces | Nodes | Arcs   | Arcs in DAGs | Max. cover |
|----------|--------|-------|--------|--------------|------------|
| kron-cp  | 2000   | 720   | 2040   | 1900         | 8970       |
| YMeme-M  | 1800   | 1000  | 163000 | 11100        | 6100       |
| YMeme-L  | 4500   | 2570  | 464000 | 86400        | 49800      |
| MTrack-M | 1800   | 35000 | 111000 | 1400         | 4500       |
| MTrack-L | 9000   | 44000 | 196000 | 4580         | 6800       |
| Flixter  | 16400  | 12900 | 176000 | 32800        | 188400     |

**Datasets.** We extract samples from three different datasets, referred to as YMeme, MTrack and Flixter. YMeme is a set of microblog postings in Yahoo! Meme.<sup>2</sup> Nodes correspond to users, actions to postings, and arcs  $(u, v)$  indicate that  $v$  follows  $u$ .

The second dataset, MTrack,<sup>3</sup> is a set of phrases propagated over prominent online news sites in March 2009, obtained by the MemeTracker system (Leskovec et al 2009). Nodes are news portals or news blogs and actions correspond to phrases found to be repeated across several sites. Arcs  $(u, v)$  indicate that the website  $v$  linked to the website  $u$  during March 2009.

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.

Finally, our third data set comes from *Flixster*,<sup>4</sup> a social movie site. The data was originally collected by Jamali and Ester (2010). Here, an action represents a user rating a movie. If user  $u$  rates “The King’s Speech,” and later on  $u$ ’s friend  $v$  does the same, we consider the action of rating “The King’s Speech” as having propagated from  $u$  to  $v$ . We use a subset of the data that corresponds to taking one unique “community,” obtained by means of graph clustering performed using **Graclus**.<sup>5</sup>

We also experimented with random graphs and traces generated by a tool supplied with the **NetInf** algorithm (Gomez-Rodriguez et al 2010). This tool creates random graphs using the Kronecker model (Leskovec and Faloutsos 2007). In our experiments we used a “core-periphery” graph generated with the seed matrix 0.962, 0.535; 0.535, 0.107 as was done by Gomez-Rodriguez et al (2010).

A summary of the datasets is shown in Table 1. “Arcs in DAGs” refers to the size of the union of the DAGs, and “Max. cover” is the maximum coverage that a solution can reach.

<sup>2</sup> [meme.yahoo.com](http://meme.yahoo.com)

<sup>3</sup> [snap.stanford.edu/data/memetracker9.html](http://snap.stanford.edu/data/memetracker9.html)

<sup>4</sup> [www.cs.sfu.ca/~sja25/personal/datasets](http://www.cs.sfu.ca/~sja25/personal/datasets)

<sup>5</sup> [www.cs.utexas.edu/users/dml/Software/gracclus.html](http://www.cs.utexas.edu/users/dml/Software/gracclus.html)

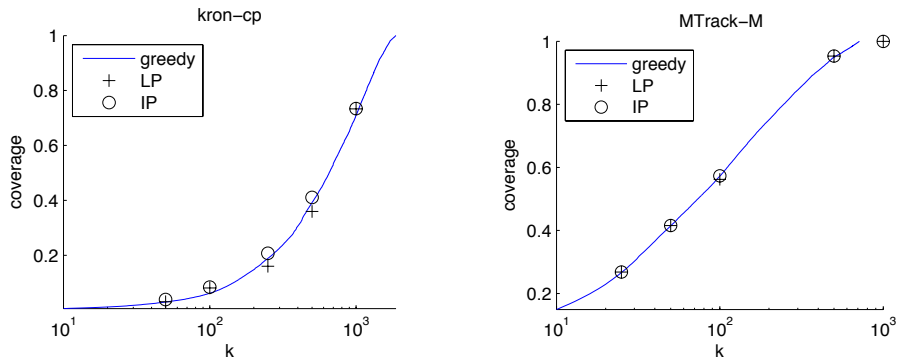
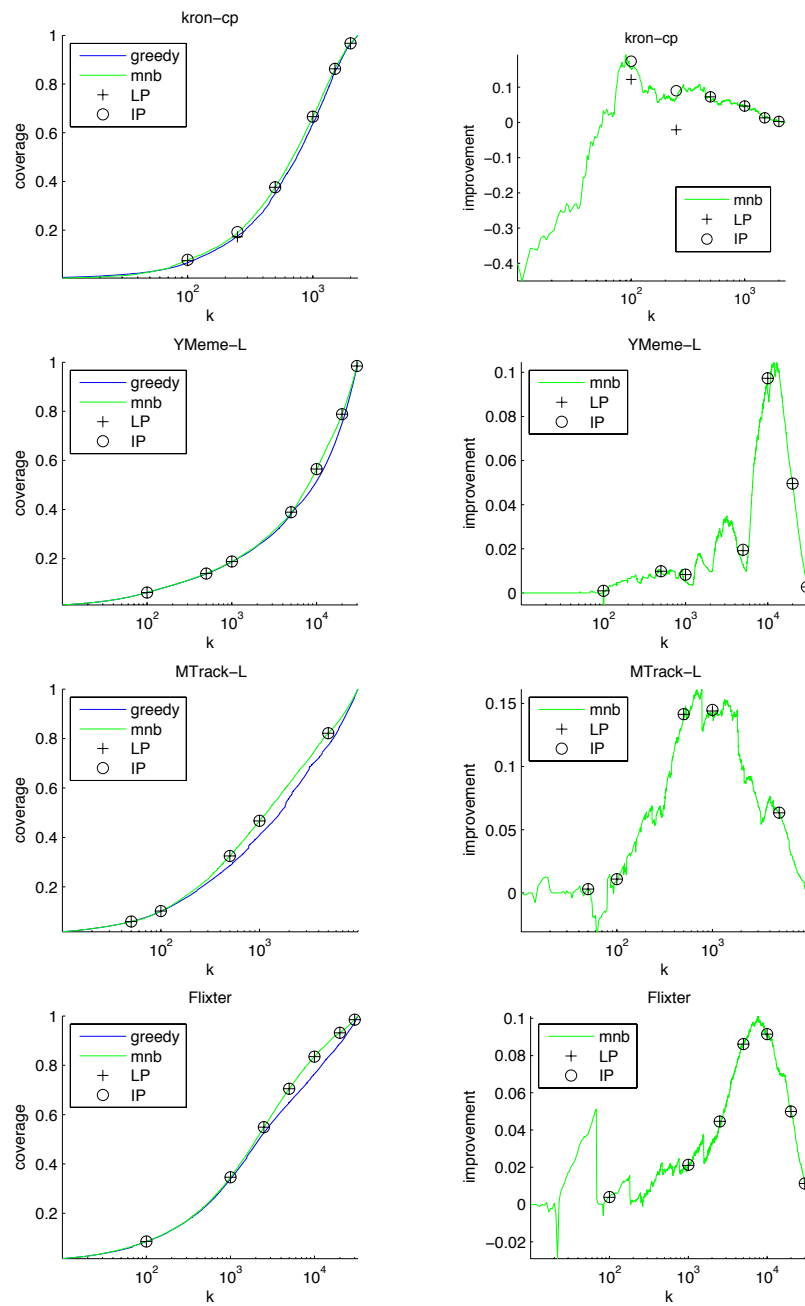


Fig. 4 Coverage with DAG traces.

**Effectiveness.** First we compare our algorithms in the basic task of selecting a subset of arcs to maximize coverage. Overall we observe only a very small difference in solution quality between the methods, and the Greedy algorithm produces very good solutions despite its bad worst-case performance. Two examples are shown in Fig. 4, with coverage plotted as a function of the budget  $k$ . We want to emphasize that the Greedy algorithm finds solutions that are surprisingly close to the optimal ones found by IP. With tree traces we can also use the MNB algorithm. Fig. 5 shows again coverage vs.  $k$  for various datasets, but also the improvement obtained over the Greedy algorithm. We note that MNB finds optimal solutions as verified by solutions to the integer program (IP), especially when  $k > 100$ . The MNB algorithm can improve solution quality up to 10-15 percent when traces are trees.

**Efficiency.** The MNB algorithm is very fast, much faster than Greedy. Table 2 shows the running times of MNB and Greedy on various datasets to produce the curves shown in Figure 5. (We do not report times for IP and LP because they are more than one order of magnitude larger.) For example, with Flixter we observe a speedup of two orders of magnitude together with a higher solution quality. Note that this comparison is fair, because both Greedy and MNB use the same implementation to compute marginal gains, and other operations carried out by the algorithms are negligible. Table 2 also shows the number of marginal gain computations carried out by both algorithms. These help to explain the better performance of MNB. Note that the runtime needed for a single marginal gain computation depends on the number of traces in the input. For example, Flixter contains a larger number of traces than YMeme-L (see Table 1), and hence Greedy runs much faster on YMeme-L than on Flixter, even if the number of marginal gain computations is not that different.

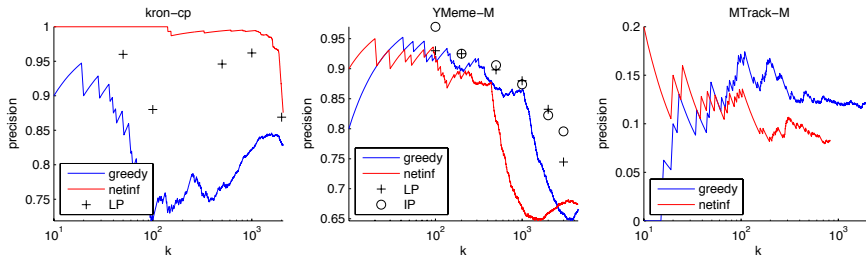
**Network reconstruction.** In this experiment we use our approach to reconstruct an unobserved network given a set of traces. Recall that we can convert any sequence of node activations to DAGs that correspond to cliques where a node has every previously activated node as a parent. Clearly the resulting DAGs are not trees, and hence the MNB algorithm can not be used. The exper-



**Fig. 5** Covering tree traces with kron-cp, YMeme-M and MTrack-M. Left side shows fraction of traces covered as a function of  $k$  (notice logarithmic scale) and right side shows relative improvement over the Greedy algorithm.

**Table 2** Running time and number of marginal gain computations of the algorithms Greedy and MNB with tree traces as input. (The numbers are rounded.)

| Dataset  | time   |        | marginal gain computations |                    |
|----------|--------|--------|----------------------------|--------------------|
|          | Greedy | MNB    | Greedy                     | MNB                |
| kron-cp  | 10 sec | 2 sec  | $2.6 \times 10^6$          | $0.11 \times 10^6$ |
| YMeme-M  | 20 sec | 2 sec  | $12.9 \times 10^6$         | $0.25 \times 10^6$ |
| YMeme-L  | 5 min  | 16 sec | $474 \times 10^6$          | $1.54 \times 10^6$ |
| MTrack-M | 17 sec | 3 sec  | $3.4 \times 10^6$          | $0.13 \times 10^6$ |
| MTrack-L | 3 min  | 7 sec  | $49.8 \times 10^6$         | $0.49 \times 10^6$ |
| Flixter  | 1 hour | 30 sec | $539 \times 10^6$          | $1.64 \times 10^6$ |

**Fig. 6** Network reconstruction with kron-cp, YMeme-M and MTrack-M. On both real-world datasets our methods are at least as good as the NetInf algorithm for  $k > 10$  in terms of precision. See also Fig. 7. (MTrack-M results in too many constraints for the linear programming approaches to be feasible.)

iment is run using Greedy and the linear-programming algorithms. We compare these against the original implementation of the NetInf algorithm by Gomez-Rodriguez et al (2010) that is specifically designed for the network-inference problem. Since all of our datasets have an underlying graph associated with them, we can use this graph as the ground truth.

Fig. 6 shows precision (defined as the fraction of true positives in a set of  $k$  arcs chosen by the algorithm) as a function of  $k$ . As shown by Gomez Rodriguez et al. the NetInf algorithm has a very good performance on the synthetic Kronecker graph, which we confirm. On the real-world datasets our approach gives results that are at least as good as those obtained by NetInf, or slightly better. (Results on the remaining datasets are qualitatively similar.) Notice that with YMeme-M the rounded LP has in some cases a higher precision than IP, even if IP is guaranteed to have the highest possible coverage. Alas, due to the very large size of the problem when we assume the network to be complete, we were not able to run IP and LP on all the possible datasets. Finally, we note that even if the algorithms perform similarly in terms of precision, our methods reach a higher coverage, as shown in Fig. 7.

**Influence maximization.** The problem of *influence maximization* (Kempe et al 2003) has received a lot of attention in the data-mining community. The problem is defined as follows. We are given a directed graph  $G = (N, A, p)$ , where  $p(u, v)$  represents the probability that an action performed by user  $u$  will cause node  $v$  to perform the action, too. We then ask to select a set of

nodes  $S \subseteq N$ ,  $|S| = k$ , such that the expected size of a propagation trace that has  $S$  as the set of source nodes, is maximized. The problem is **NP**-hard but due to the submodularity property of the objective function, the simple greedy algorithm produces a solution with provable approximation guarantee.

Following Mathioudakis et al (2011) we experiment with graph simplification as a pre-processing step before the influence maximization algorithm. The idea is to show that computations on simplified graphs give up little in terms of effectiveness (measured as the expected size of the trace generated by  $S$ ), but yield significant improvements in terms of efficiency.

The experiment is conducted following Mathioudakis et al (2011). First of all we assume the *independent cascade model* as the underlying propagation model. Given our graph  $G = (N, A)$  and the log of sequences  $\mathcal{D}$ , we learn for each arc  $(u, v)$  the probability  $p(u, v)$  with the same expectation-maximization method used by Mathioudakis et al (2011), and using their own original implementation. Then given the directed probabilistic graph  $G = (N, A, p)$ , we sparsify it with the **Spine** algorithm and with our **Greedy** method, and we compare the effectiveness and the run time of the influence maximization algorithm of Kempe et al (2003) when run on the whole of  $G = (N, A, p)$  and on the two sparsified graphs.

It is worth noting that the comparison is “unfair” for our method for at least two reasons: (i) both the evaluation process and the **Spine** algorithm assume the same underlying propagation model (the independent cascade model), while our method does not assume any propagation model; (ii) both the evaluation process and **Spine** use exactly the same influence probabilities associated to the arcs, while our methods only use the graph structure and the set of traces  $\mathcal{D}$ .

We sparsify YMeme-L to 25% of its original size by using both **Spine** and **Greedy**, and compare the expected size of the cascades. We also measure the run time of the influence maximization algorithm for different sizes of the seed set when running on the full network and on the two sparsified ones.

Fig. 8 shows the results of the experiments. The running time of the influence maximization algorithm is comparable when run on the two sparsified networks. **Spine** achieves larger cascades in expectations. This results is not surprising given that **Spine** is designed to take advantage of the underlying propagation model. Nevertheless, **Greedy** achieves good results with only a constant difference from the results obtained on the full network.

## 6 Conclusions

We studied the problem of simplifying a graph while maintaining the maximum connectivity required to explain observed activity in the graph. Our problem can be expressed as follows: given a directed graph and a set of DAGs (or trees) with specified roots, select a subset of arcs in the graph so as to maximize the number of nodes reachable in all DAGs by the corresponding DAG roots. We

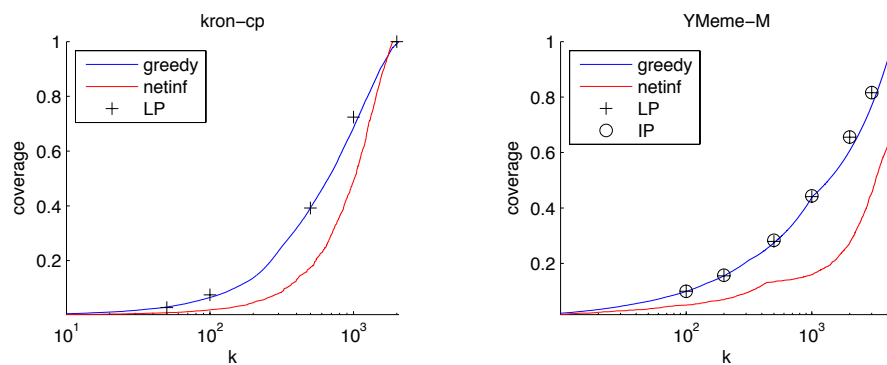


Fig. 7 Coverage in the network reconstruction experiment. See also Fig. 6.

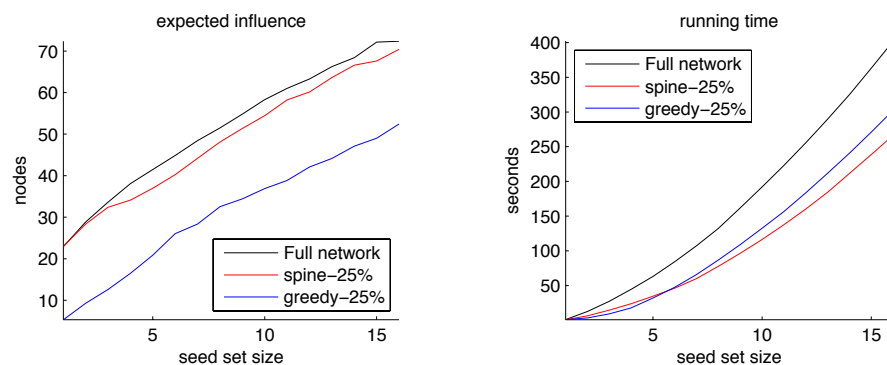


Fig. 8 Size of the cascades and run time for influence maximization.

studied the properties of this problem and we developed different algorithms, which we evaluated on real datasets.

Several future research directions and open problems remain. Our **NP**-hardness proof (Theorem 1) relies on traces being DAGs. What is the complexity of the problem when all traces are trees? Also gaining a deeper understanding of the MNB algorithm is of interest. Under what conditions does it produce solutions that cover a useful range of  $k$ ? Finally, can we find other efficient algorithms for the problem on the basis of the linear programming formulation?

## References

- Arenas A, Duch J, Fernández A, Gómez S (2007) Size reduction of complex networks preserving modularity. *New Journal of Physics* 9(6):176
- Edmonds J (2003) Submodular functions, matroids, and certain polyhedra. In: *Combinatorial Optimization—Eureka, You Shrink!*, Springer, pp 11–26

- 
- Elkin M, Peleg D (2005) Approximating  $k$ -spanner problems for  $k > 2$ . *Theoretical Computer Science* 337(1):249–277
- Foti NJ, Hughes JM, Rockmore DN (2011) Nonparametric sparsification of complex multiscale networks. *PloS one* 6(2):e16,431
- Fujishige S (2005) *Submodular functions and optimization*, vol 58. Elsevier Science
- Fung WS, Hariharan R, Harvey NJ, Panigrahi D (2011) A general framework for graph sparsification. In: *Proceedings of the 43rd annual ACM symposium on Theory of computing*, ACM, pp 71–80
- Gomez-Rodriguez M, Leskovec J, Krause A (2010) Inferring networks of diffusion and influence. In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp 1019–1028
- Gomez-Rodriguez M, Balduzzi D, Schölkopf B (2011) Uncovering the temporal dynamics of diffusion networks. In: *Proceedings of the 28th International Conference on Machine Learning*, pp 561–568
- Iwata S, Orlin JB (2009) A simple combinatorial algorithm for submodular function minimization. In: *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, Society for Industrial and Applied Mathematics, pp 1230–1237
- Jamali M, Ester M (2010) Modeling and comparing the influence of neighbors on the behavior of users in social and similarity networks. In: *2010 IEEE International Conference on Data Mining Workshops (ICDMW)*, IEEE, pp 336–343
- Kempe D, Kleinberg J, Tardos É (2003) Maximizing the spread of influence through a social network. In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp 137–146
- Krause A (2010) Sfo: A toolbox for submodular function optimization. *The Journal of Machine Learning Research* 11:1141–1144
- Leskovec J, Faloutsos C (2007) Scalable modeling of real graphs using kronecker multiplication. In: *Proceedings of the 24th international conference on Machine learning*, ACM, pp 497–504
- Leskovec J, Backstrom L, Kleinberg J (2009) Meme-tracking and the dynamics of the news cycle. In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp 497–506
- Mathioudakis M, Bonchi F, Castillo C, Gionis A, Ukkonen A (2011) Sparsification of influence networks. In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp 529–537
- Misiulek E, Chen DZ (2006) Two flow network simplification algorithms. *Information processing letters* 97(5):197–202
- Nagano K, Kawahara Y, Aihara K (2011) Size-constrained submodular minimization through minimum norm base. In: *Proceedings of the 28th International Conference on Machine Learning*, pp 977–984
- Nemhauser GL, Wolsey LA, Fisher ML (1978) An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*

- 14(1):265–294
- Peleg D, Schäffer AA (1989) Graph spanners. *Journal of graph theory* 13(1):99–116
- Quirin A, Cordon O, Santamaria J, Vargas-Quesada B, Moya-Anegón F (2008) A new variant of the pathfinder algorithm to generate large visual science maps in cubic time. *Information processing & management* 44(4):1611–1623
- Serrano E, Quirin A, Botia J, Cerdón O (2010) Debugging complex software systems by means of pathfinder networks. *Information Sciences* 180(5):561–583
- Serrano MÁ, Boguñá M, Vespignani A (2009) Extracting the multiscale backbone of complex weighted networks. *Proceedings of the national academy of sciences* 106(16):6483–6488
- Srikant R, Yang Y (2001) Mining web logs to improve website organization. In: *Proceedings of the 10th international conference on World Wide Web*, ACM, pp 430–437
- Svitkina Z, Fleischer L (2011) Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM Journal on Computing* 40(6):1715–1737
- Toivonen H, Malher S, Zhou F (2010) A framework for path-oriented network simplification. In: *Advances in Intelligent Data Analysis IX*, Springer, pp 220–231
- Wolfe P (1976) Finding the nearest point in a polytope. *Mathematical Programming* 11(1):128–149
- Zhou F, Malher S, Toivonen H (2010) Network simplification with minimal loss of connectivity. In: *Data Mining (ICDM), 2010 IEEE 10th International Conference on, IEEE*, pp 659–668