Finding Effectors in Social Networks

Theodoros Lappas
UC Riverside
tlappas@cs.ucr.edu

Evimaria Terzi
Boston University
evimaria@cs.bu.edu

Dimitrios Gunopulos
University of Athens
dg@di.uoa.gr

Heikki Mannila
University of Helsinki
mannila@cs.helsinki.fi

ABSTRACT

Assume a network (V, E) where a subset of the nodes in V are *active*. We consider the problem of selecting a set of k active nodes that best explain the observed activation state, under a given information-propagation model. We call these nodes *effectors*. We formally define the k-EFFECTORS problem and study its complexity for different types of graphs. We show that for arbitrary graphs the problem is not only NP-hard to solve optimally, but also NP-hard to approximate. We also show that, for some special cases, the problem can be solved optimally in polynomial time using a dynamic-programming algorithm. To the best of our knowledge, this is the first work to consider the k-EFFECTORS problem in networks. We experimentally evaluate our algorithms using the **DBLP** co-authorship graph, where we search for effectors of topics that appear in research papers.

Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph Theory— *oryŮ-Graph algorithms*; H.2.8 [**Database Management**]: Database applications—*Data Mining*

General Terms

Algorithms, Experimentation, Theory

Keywords

social networks, information propagation, graph algorithms

1. INTRODUCTION

Consider the directed network shown in Figure 1, where the black nodes are *active* and the white nodes are *inactive*. The activation state of the network is described by an *activation vector*, $\mathbf{a}$. In the example of Figure 1, $\mathbf{a}(x) = \mathbf{a}(y_i) = 1$ for $0 \leq i \leq 2$ and $\mathbf{a}(x_i) = 0$ for $1 \leq i \leq \ell$. Assume a simple probabilistic information-propagation model such that every node v that becomes active activates a neighbor x via a directed link $(v \rightarrow x)$; this activation succeeds with probability equal to the weight of the directed link $(v \rightarrow x)$. Given a budget k, our goal is to find a set of k active nodes, such that, had the propagation started from them, it would have caused an activation state similar to the one described by $\mathbf{a}$. We call these nodes *effectors*[1] and the corresponding optimization problem the k-EFFECTORS problem. Effectors need not be the nodes that first became active during the information-propagation process; therefore, complete knowledge of the timestamps associated with the activation of every node would not necessarily help in identifying the effectors. Further, effectors need not be centrally-located in the network. They are simply the nodes that best explain the observed activation vector.

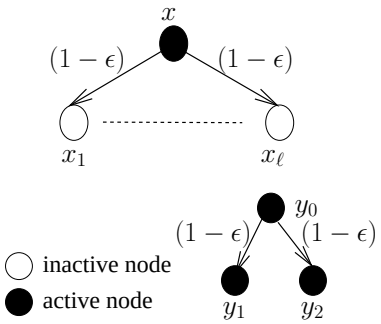


Figure 1: A network with active (black) and inactive (white) nodes. Edge weights represent the probability of an active node activating its neighbors; $\epsilon \in (0, 1)$.

In our example, assume that $k = 2$, $0 < \epsilon < 1$ and let the set of effectors be $X = \{x, y_1\}$. For this set X and the given propagation model, the *expected* final state of the propagation process assigns to every node v a probability of being active $\alpha(X, v)$. In this example, $\alpha(X, x) = \alpha(X, y_1) = 1$, $\alpha(X, x_i) = (1 - \epsilon)$ for $1 \leq i \leq \ell$, $\alpha(X, y_0) = 0$ and $\alpha(X, y_2) = 0$. We define the cost of solution X to be $C(X) = \sum_{v \in V} |\mathbf{a}(v) - \alpha(X, v)| = (1 - \epsilon)\ell + 2$. On the other hand, solution $X' = \{y_0, y_1\}$ would have cost $C(X') = 1 + \epsilon$ and it would be the optimal solution for every $\epsilon \in (0, 1)$.

In *social networks*, the identification of effectors can improve our understanding of the dynamics of information propagation. Effectors can be interpreted as key nodes that determine whether a novel concept dies out quickly or prop-

[1]In biochemistry, an effector is a substance that increases or decreases the activity of an enzyme.

KDD'10, July 25–28, 2010, Washington, DC, USA.

agates to cover a significant portion of the network. In *epidemiological studies*, the effectors are the key individuals (or countries) that cause a particular diffusion pattern. The discovery of effectors can be leveraged in the design of vaccination strategies and quarantine policies. In *computer networks*, the effectors are computers in the network that affect the spread pattern of a computer virus. Again, effector discovery can facilitate inoculation strategies: rather than blindly investing on security software for protecting large parts of the network, system administrators can only focus on securing the effector nodes.

Our contribution: In this paper, we first introduce the k-EFFECTORS problem and explore its connections to other existing problems in the literature. We prove that, in a general setting, the k-EFFECTORS problem is not only NP-hard to solve optimally, but also NP-hard to approximate. We also show that, in trees, the k-EFFECTORS problem can be solved optimally in polynomial time by using an efficient dynamic-programming algorithm. We also explore the performance of other computationally-efficient heuristics. Although our worst-case analysis shows that these heuristics are clearly suboptimal, our experimental evaluation reveals that, in certain settings, they can perform reasonably well. Finally, we experimentally validate our methods on the co-authorship graph defined by the **DBLP** dataset. More specifically, we use the **DBLP** co-authorship graph to find effectors of topics that appear in computer-science papers. We present qualitative evidence to show that the effectors identified by our methods convey meaningful information about the data.

We believe that the notion of effectors can improve our understanding of diffusion processes in networks. Although we focus our attention on the role of effectors in social networks, our framework can be applied to a variety of network data – including computer and biological networks – and give useful insights to the data analysts.

Our approach: Our approach for solving the k-EFFECTORS problem on tree networks consists of an optimal dynamic-programming algorithm. For general graphs, we proceed in two steps: first, for a given network and activation vector, we construct the *most probable tree* $\mathcal{T}$, that spans all the active nodes in the network. Then, we use the optimal dynamic-programming algorithm to identify the optimal effectors on $\mathcal{T}$. We believe that the extraction of the most probable tree from the input graph is interesting in its own right, since this tree models the backbone of information propagation in the network.

Roadmap: The rest of the paper is organized as follows: in Section 2 we survey the related work and in Section 3 we give the necessary notation and describe the information-propagation model. The problem definition and the complexity results are presented in Section 4. In Section 5 we describe the optimal polynomial-time algorithm for trees and in Section 6 we present our algorithm for extracting the most probable tree from any given input graph. In Section 7 we provide a thorough set of experiments on on a co-authorship graph and we conclude the paper in Section 8.

2. RELATED WORK

To the best of our knowledge, we are the first to formally define and study the k-EFFECTORS problem. Although the exact combinatorial definition of effectors does not exist in the literature, there has been a lot of work on problems related to the identification of *influential* nodes or trends in social or other networks. As expected, different definitions of influential nodes lead to different computational challenges. We summarize some of this work here.

In the blogosphere, there is significant research in the identification of influential blogs [12] and bloggers [1, 18]. Similarly, for marketing surveys, the problem of identifying the set of early buyers has been addressed [19]. However, the algorithmic settings are very different from ours. For example, Gruhl et. al. [12] study information diffusion of various topics in the blogoshere. The focus is on studying how the topics propagate or how "sticky" the topics are. In our setting, we do not touch upon the issue of durability of the trends; once a node becomes active, it remains active. In other studies, the focus is on the identification of influential bloggers [1, 18]. In these cases, the authors define a metric that determines the influence potential of a blogger. The focus is on developing efficient algorithms for computing the top-k influential nodes. In contrast, we evaluate groups of effectors and how they collectively affect the network.

Further, the aforementioned papers do not explicitly take into account the information-propagation model. Information-propagation models have been considered in the context of influence maximization [3, 7, 13]. The focus of those works is on identifying the set of nodes in the network that need to be targeted (e.g., for targeted advertisement), so that the propagation of a product or an idea spreads as much as possible. In influence maximization, the goal is to identify the nodes that will cause the most propagation effect in the network. In our case, the goal is to identify the nodes that better explain a particular - *observed*- activation pattern in the network. In fact, our problem definition contains influence maximization as a special case.

Bharathi et al. [3] consider the influence-maximization problem on bidirectional trees and develop an FPTAS. In addition to the fact that their objective function is different from ours, they also focus on *undirected* trees. Our own focus is on directed graphs and directed trees. For undirected trees, their problem is NP-complete. On the other hand, ours is solvable in polynomial time for the case of directed trees.

The problem of identifying early adopters from transaction data has been addressed by Rusmevichientong et. al. [19]. In that work, the set of early buyers is identified by taking as input the detailed purchase information of each consumer. Then, a weighted directed graph is constructed: the nodes correspond to consumers and the edges to purchases these consumers have in common. Identifying early buyers corresponds to the problem of finding a subset of nodes in the graph with maximum difference between the weights of the outgoing and incoming edges. Contrary to our setting, the framework proposed by Rusmevichientong et. al. does not consider any information-propagation dynamics or any underlying social network.

The problem of finding links and initiators was also studied by Mannila and Terzi [17]. Their problem-setting is the following: given a set of individuals and the set of items each of them has purchased, the goal is twofold: a) For each item, identify the individuals that acted as its initiators. b) Infer the social relationships between individuals. The main difference between that work and our current work, is that here we assume that the social graph is given as part of the input. Further, we identify the set of effectors while ignor-

ing the temporal information associated with the purchased items. Finally, the method of Mannila and Terzi is based on an MCMC sampling of the space of all possible graphs and initiators. Here, we solve the optimization problem of finding the *best* set of effectors rather than assigning probabilities to nodes being effectors.

Other definitions of "important" nodes in a network focus on the development of network inoculation strategies [2] and early epidemic detection [4, 16]. Since our goal is to find a set of effectors that best explain the network's final state, both our problem definition as well as our algorithmic approaches are significantly different.

3. PRELIMINARIES

We assume a social network represented by graph a $G = (V, E, p)$. The nodes in V correspond to individuals. There is an edge between two individuals $u, v \in V$ if u and v are associated with each other. The edges in the network are *directed*; edge $(u \to v) \in E$ is associated with an *influence weight* $p(u \to v) \in [0, 1]$. This weight quantifies the effect that node u has on the decisions of node v. We give a probability interpretation to this weight. Note that we use the terms "graph", "social network" and "influence graph" interchangeably.

We assume that the influence weights are part of the input. For example, one can ask the users themselves to assign their own estimates of how much they are influenced by their own friends. Alternatively, one can employ a machine-learning algorithm to infer such probabilities [14]. For our experiments, we use a simple and intuitive method for computing the influence probabilities. The details of this computation are given in Section 7. The exploration of alternative methods for such computation, though interesting, is beyond the scope of this paper.

Further, we assume that the influence of one node to another is the same for all items that propagate in the network. Exploring the performance of more specialized techniques that cluster the items and compute different influence probabilities per cluster is beyond the scope of this paper.

Apart from the network and the influence probabilities, we also assume a particular (information) item I. For every node $v \in V$, an item I either appears or does not appear in v. We represent this information using a 0–1 $n \times 1$ vector $\mathbf{a}$; $\mathbf{a}(i) = 1$ if item I is observed at node i. Otherwise, $\mathbf{a}(i) = 0$. If $\mathbf{a}(i) = 1$ (resp. $\mathbf{a}(i) = 0$) we say that node i is *active* (resp. *inactive*). We call this vector the *activation vector* of I. Note that we assume that the entries of the activation vector are either 0 or 1. However, all our results carry over to the case where the observed activation vector takes real values in the interval $[0, 1]$.

3.1 The information-propagation model

We consider the following information-propagation model in a social network: when node u becomes active for the first time at step t, it gets a single chance to activate node v through the edge $(u \to v)$; u succeeds in this activation attempt with probability $p(u \to v)$ – as defined in the influence graph. If u succeeds, then v will become active at step $(t + 1)$. Otherwise, u cannot make any more attempts to activate v in any subsequent rounds. This model is called the *Independent Cascade* (IC) model [10, 11, 13]. IC is a *probabilistic* propagation model, since the activation process is influenced by probabilistic choices. Given a seed of nodes that are originally active, each node in the network is active with some probability. upon the termination of the process.

In the special case where all the the influence weights are equal to one, the IC model becomes equivalent to the *deterministic propagation* (DM) model. In the DM model every node that becomes active at step t activates all its neighbors with probability 1. Therefore, the activation of a single node in a strongly-connected component [2] is sufficient to activate all the nodes in the component.

Although we focus our attention on the IC model, our framework can be combined with any information-propagation model, including the *Linear Threshold* (LT) model [13] or the *Susceptible - Infected - Susceptible* (SIS) model [4].

Given a set $X \subseteq V$ of originally active nodes, the propagation of information with IC will terminate in at most n discrete timestamps. Since the information-propagation model is non-deterministic, one needs to compute the probability that a node $v \in V$ is active at the end of the process. Computing this probability, denoted by $\alpha(v, X)$, requires exponential time in arbitrary graphs. On the other hand, one can estimate it by using the following simple heuristic: For graph $G = (V, E, p)$ keep every edge $(u \to v)$ with probability $p(u \to v)$. The edges of the resulting graph G' have influence probabilities equal to 1. That is, one can run the DM model on G'. After repeating this process N times, one can estimate $\alpha(v, X)$ by simply counting the fraction of the times v was active in the sampled graphs.

Further, if $G = (V, E, p)$ is a *directed* tree, then for $X \subseteq V$, we can find a closed-form expression of $\alpha(v, X)$. That is, for every node v we have that:

$$\alpha(v, X) = 1 - \prod_{x \in X} \left(1 - \prod_{(y \to z) \in \text{path}(x,v)} p(y \to z) \right). \quad (1)$$

The term inside the parenthesis corresponds to the probability that node v does not get influenced by node x. Therefore, the outer product computes the probability that node v does not get active. The probability that v gets active is, naturally, one minus this product.

4. THE PROBLEM

Assuming a particular information-propagation model, our goal is to solve the following problem.

PROBLEM 1 (k-EFFECTORS PROBLEM). *Given a social network graph $G = (V, E, p)$ and an activation vector $\mathbf{a}$, find a set X of active nodes (effectors), of cardinality at most k such that*

$$C(X) = \sum_{v \in V} |\mathbf{a}(v) - \alpha(v, X)| \quad (2)$$

is minimized.

The k-EFFECTORS problem asks for the set of individuals that, once activated, cause an activation pattern which is as similar as possible to the activation observed in vector $\mathbf{a}$. We also use $C(v, X)$ to refer to the contribution of node v in the cost function. In other words, we define $C(v, X) = |\mathbf{a}(v) - \alpha(v, X)|$ and thus $C(X) = \sum_{v \in V} C(v, X)$.

[2]In a strongly-connected component of a directed graph there is a directed path from every node to every other node of the component.

The definition of the k-EFFECTORS problem is independent of the information-propagation model. Although some of our results generalize to many information-propagation models, we focus here on the IC model. Also, we restrict the effectors to be selected from the set of active nodes. Although allowing any node (active or inactive) to be an effector would not change our theoretical results, we put this constraint mostly because picking inactive nodes as effectors contradicts our intuition.

Next, we study the complexity of the k-EFFECTORS problem under the IC propagation model. For the complexity results we use the decision version of the k-EFFECTORS problem, which we parameterized by cost c. That is, k-EFFECTORS(c) is formulated as the following decision problem: Given a social network $G = (V, E, p)$ and an activation vector $\mathbf{a}$ does there exist a set $X \subseteq V$, $|X| \leq k$ with $C(X) \leq c$? We begin by proving the following lemma.

LEMMA 1. *Assuming the IC propagation model, the k-EFFECTORS(0) problem is NP-complete.*

PROOF. Consider an instance of the NP-complete SET COVER problem, defined by a collection of subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of a ground set $U = \{u_1, u_2, \ldots, u_n\}$. The question is whether there exist k subsets from $\mathcal{S}$ whose union is equal to U. Given an arbitrary instance of the SET COVER PROBLEM, we define the corresponding graph G to be a directed graph with $n + m + 1$ nodes. There is a node i corresponding to each set S_i, a node j corresponding to each element u_j, and a directed edge $(i \to j)$ with influence probability $p(i \to j) = 1$ whenever $u_j \in S_i$. The $(n + m + 1)$-th node of G is node ℓ. Every node j (corresponding to element u_j) is connected to node ℓ via a directed edge with weight $p(j \to \ell) = 1/n$. Finally, node ℓ is connected every node i (that correspond to set S_i) via a directed edge with probability $p(\ell \to i) = 1$. Finally, we set the activation vector so that all nodes in the graph G are active, i.e., $\mathbf{a} = \vec{1}$. There exists a solution consisting of k sets to the SET COVER problem if and only if there exists a set X of k effectors in this graph with cost $C(X) = 0$. The problem is trivially in NP. □

Lemma 1 allows us to prove the following inapproximability result.

LEMMA 2. *Assuming the IC propagation model, there does not exist a β-approximation algorithm for the k-EFFECTORS problem, with $\beta > 1$, unless $P = NP$.*

PROOF. The proof is by contradiction. Assume that there is a polynomial time β-approximation algorithm for the k-EFFECTORS problem; call this algorithm `Approx`. For any instance $G = (V, E, p)$ and activation vector $\mathbf{a}$, `Approx` will produce a solution $X \subseteq V$ such that $C(X) \leq \beta C(X^*)$, where X^* is the optimal solution. Assume now an instance of the k-EFFECTORS(0) problem (see Lemma 1). If we give this instance as input to the `Approx` algorithm then, `Approx` should be able to *decide* whether there is a 0-cost solution to the instance or not. However, from Lemma 1, we know that k-EFFECTORS(0) is NP-complete and thus we reach a contradiction. □

In fact, the k-EFFECTORS problem is a generalization of the INFLUENCE MAXIMIZATION problem [13]. In our context, the INFLUENCE MAXIMIZATION problem asks for the set $Y \subseteq V$ with $|Y| \leq k$, such that $\sum_{v \in V} \alpha(v, Y)$ is maximized. Maximizing $\sum_{v \in V} \alpha(v, Y)$ is equivalent to minimizing $\sum_{v \in V}(1 - \alpha(v, Y))$. Thus, when the activation vector $\mathbf{a}$ contains all 1's, i.e., $\mathbf{a} = \mathbf{1}$, the two problems are equivalent.

This observation allows us to infer that the k-EFFECTORS problem is NP-complete for all the information propagation models used by Kempe et al. [13]. In fact, by the results of Kempe et. al. [13] (due the construction used in the proof of Theorem 2.4), we also know that INFLUENCE MAXIMIZATION, for the IC propagation model, is NP-complete even for Directed Acyclic Graphs (DAGs). As a result the k-EFFECTORS problem is also NP-complete for DAGs.

COROLLARY 1. *Assuming the IC propagation model, the k-EFFECTORS problem is NP-complete even when the input graph $G = (V, E, p)$ is a DAG.*

However, for the DM propagation model, the k-EFFECTORS problem can be solved optimally in polynomial time. The polynomial-time algorithm first finds all the strongly connected components of the input graph G. Let there be ℓ such components that partition the nodes in V into parts $V_1, \ldots, V_\ell$. Let $N_l = |\{v \mid v \in V_l \text{ and } \mathbf{a}(v) = 1\}|$. Then, the optimal solution can be constructed by picking one (arbitrarily chosen) node from each of the connected components with the k highest N_l scores. Within these k components, all the nodes have an equal probability of being picked as effectors. This observation makes the DM model inappropriate for realistic settings.

5. FINDING EFFECTORS ON TREES

Here we show that the k-EFFECTORS problem can be solved optimally in polynomial time when the graph $G = (V, E, p)$ is a tree. For clarity, we denote such a graph by $\mathcal{T} = (V, E, p)$.

5.1 The optimal DP algorithm

Our polynomial-time algorithm uses dynamic programming. The main idea is the following: given a subtree whose root has δ children, the optimal way of specifying at most k effectors from this subtree must follow one of two patterns: in the first pattern, we include the root of the subtree to the set of effectors, and then recurse on the children with budget $(k - 1)$. In the second, we do not include the root of the subtree, and instead recurse on the children with a budget k.

A naive way of implementing the recursion would result in partitioning the δ children into k (or $k - 1$) parts and taking the minimum-cost partition. However, when $\delta >> 2$, computing the cost of all possible partitions is expensive. To circumvent this, we make a simple transformation that converts any tree to a binary tree.

We construct the new tree $\mathcal{T}_b$ from the original tree $\mathcal{T}$ as follows: we start from the root of $\mathcal{T}$, *root*($\mathcal{T}$). Suppose that v is an internal node of $\mathcal{T}$ with children $v_1, \ldots, v_\delta$, with $\delta > 2$. We replace v with a binary tree of depth at most $\log \delta$ and leaves $v_1 \ldots, v_\delta$. Picking each one of the leaves $v_1, \ldots, v_\delta$ introduces a cost calculated the way we described above. Recall that we have a budget of k effectors. Every node v_i that corresponds to an actual node in the original tree $\mathcal{T}$ uses one unit of the budget, if picked as an effector. Further, the newly-created internal nodes in $\mathcal{T}_b$ that do not correspond to any actual nodes in $\mathcal{T}$ can never be picked

as initiators. Directed edges are added between node v and these new internal nodes, as well as between the internal nodes themselves. The direction is always from the root to the leaves and the weight of these edges is set to 1. In this way, the directed edges that are associated with the newly added internal nodes in $\mathcal{T}_b$ do not influence the propagation from v to its children. This transformation is repeated recursively for each child $v_1, \ldots, v_\delta$. We denote the set of newly added (dump) nodes by D.

The following two observations are a direct consequence of the above process. Moreover, they guarantee that the newly-created binary tree causes bounded increase in the number of nodes and the depth of the original tree.

OBSERVATION 1. *The number of nodes in the binary tree $\mathcal{T}_b$ is at most twice the number of nodes of tree $\mathcal{T}$.*

OBSERVATION 2. *If Δ is the maximum out-degree of a node in tree $\mathcal{T}$, then the depth of the binary tree $\mathcal{T}_b$ is at most a factor of $\log \Delta$ larger than the depth of $\mathcal{T}$.*

Following the proofs appearing in similar constructions for different problems [9, 15, 20] we can also prove the following observation.

OBSERVATION 3. *the optimal solution to the k-EFFECTORS problem on $\mathcal{T}_b$ is the same as the optimal solution of the k-EFFECTORS problem on tree $\mathcal{T}$.*

Intuitively, this is because the newly-added nodes in $\mathcal{T}_b$ can neither be picked as effectors nor influence the information-propagation process. This is because all their outgoing edges have weight 1.

Given the above transformation, we can always assume that our influence tree is binary and we use $\mathcal{T}$ to refer to such binary tree. For a node v of the tree, we use $\textsc{Opt}(v, X, k)$ to denote the cost of the best solution in the subtree rooted at node v, using at most k effectors; X simply keeps the effectors in the current solution. Finally, for a node v we use $r(v)$ ($\ell(v)$) to refer to the right (the left) child of node v. Then, we evaluate the following dynamic-programming recursion on the nodes of the tree $\mathcal{T}$:

$$\begin{aligned}\textsc{Opt}(v, S, k) = \min \quad & (3)\\ & \Big\{ \min_{k'=0}^{k} \big\{\textsc{Opt}(r(v), S, k') + \textsc{Opt}(\ell(v), S, k - k') + C(v, S)\big\},\\ & C(v, S \cup \{v\}) + \min_{k'=0}^{k-1} \big\{\textsc{Opt}(r(v), S \cup \{v\}, k') +\\ & +\textsc{Opt}(\ell(v), S \cup \{v\}, k - k' - 1\big\}\Big\}.\end{aligned}$$

The first term of the dynamic-programming recursion corresponds to not choosing v to be in S and the bottom term corresponds to choosing v to be in S. In order to guarantee that no newly-added node in the set D is picked as an effector, we set $C(v, S) = \infty$ for every $v \in D$ and any $S \subseteq V$. In addition, since the effectors are always selected from active nodes we also add a similar check to guarantee that no inactive nodes are picked. We call this dynamic-programming algorithm the `DP` algorithm.

The first term of the dynamic-programming recursion consists of $2k$ lookups on precomputed values in the table OPT and it thus takes $\mathcal{O}(k)$ time. The bottom term, however, needs to go through all the nodes in the subtrees rooted at $\ell(v)$ and $r(v)$ and compute the additional cost incurred by the addition of node v as an effector. In the worst case, there are $\mathcal{O}(n)$ such terms and there are $O(k)$ evaluations that need to be done. Therefore, the computation of a single entry in table OPT requires $\mathcal{O}(nk)$ time. This is an overestimate of the actual time since, on average, the expected size (number of nodes) of a subtree in a binary tree is $\mathcal{O}(\log n)$. Therefore, the expected time required for the evaluation of a single entry is $\mathcal{O}(k \log n)$. Given that there are kn different entries, the worst-case time complexity of the `DP` algorithm is $\mathcal{O}(n^2k^2)$, while the expected running time is $\mathcal{O}(k^2 n \log n)$. In the above analysis we have assumed that given $C(v, S)$ we can compute $C(v, S \cup x)$ in constant time. In fact, this can be done by keeping at every node v the value of the product $\prod_{s \in S} \left(1 - \prod_{(y \to z) \in \text{path}(x,v)} p(y \to z)\right)$. The addition of a new node x in S would then simply require the update of this product and the use of Equation (1) for computing $\alpha(v, S \cup \{x\})$

Such bookkeeping comes with increasing space requirements: apart from storing the $n \times k$ values of table OPT, we also need to store k values of the product per node. Therefore, the total space required by `DP` is $O(2nk)$.

Although `DP` is optimal, its running time and space requirements may make it inappropriate for very large datasets. Therefore, we propose two alternative algorithms: `Sort` and `OutDegree`. Both algorithms have sub-quadratic running times and require much less memory than `DP`. Experiments on real data show that the two algorithms perform almost as well as the optimal. However, one can construct examples in which the quality of the results degrades.

5.2 The `Sort` algorithm

For a given tree $\mathcal{T} = (V, E, p)$, the `Sort` algorithm evaluates, for every node $v \in V$, the cost incurred when v is the only effector in $\mathcal{T}$. That is, for every node v the cost $C(\{v\}) = \sum_{x \in V} |\alpha(x, \{v\}) - \mathbf{a}(x)|$ is computed. The set of k effectors is then formed by picking the k active nodes with the smallest cost.

Computing the cost $C(\{v\})$ for every node $v \in V$ has worst-case running time $\mathcal{O}(n^2)$. However, the expected running time on binary trees is $\mathcal{O}(n \log n)$. Finally, the nodes are sorted based on their $C(\{v\})$ scores in $\mathcal{O}(n \log n)$ time.

Although `Sort` performs well on real datasets, one can construct cases where the algorithm's performance is far from optimal. Consider for example the directed influence tree in Figure 2. The tree consists of $2n$ active (denoted by black) nodes. Nodes $w_1, \ldots, w_n$ are activated by the root u with probability ϵ. Root has influence probability 1 to v_1 and every node v_i activates node v_{i+1} also with probability 1 ($1 \le i \le (n-1)$). The cost of the root u is $C(\{u\}) = (1 - \epsilon)n$. The cost of node any node v_i (for $1 \le i \le n-1$) is $C(\{v_i\}) = i + n$. Similarly, the cost of activating one of the w_j nodes (for $1 \le j \le n$) is $C(\{w_j\}) = 2n - 1 > C(\{v_i\})$ for $1 \le i \le n-2$; for $i = n-1$ we have a tie in which case the algorithm resolves it by setting $C(\{v_{n-1}\}) < C(\{w_j\})$. Solving the k-EFFECTORS problem for $k = n$, the `Sort` algorithm would report as effectors $S = \{u, v_1, \ldots, v_{n-1}\}$, with cost $C(S) = n$. However, the optimal set of effectors is $S^* = \{u, w_1, \ldots, w_{n-1}\}$, with cost $C(S^*) = (1 - \epsilon)$. Therefore, the performance ratio of `Sort` is $\frac{n}{(1-\epsilon)}$. This is a ratio of order $\mathcal{O}(n)$. Thus, `Sort` can produce solutions that are at least $\mathcal{O}(n)$ times worse than the optimal.

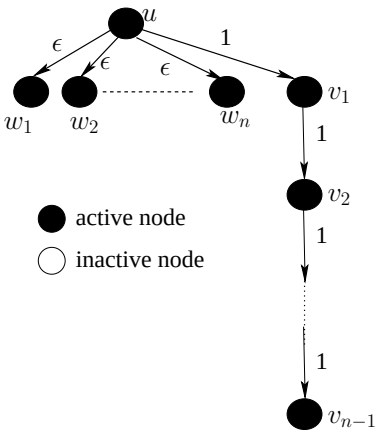


Figure 2: Influence tree with $2n$ active nodes. Edge weights are in $[0, 1]$ with $\epsilon \in (0, 1)$. The `Sort` algorithm for $k = n$ reports a solution that is $\mathcal{O}(n)$-times worse than the optimal.

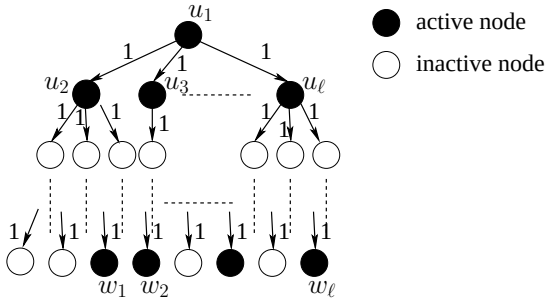


Figure 3: Influence tree with 2ℓ active and $n - \ell$ inactive nodes. All edge weights are equal to 1. The `OutDegree` algorithm for $k = \ell$ reports a solution that is $\mathcal{O}(n)$ times worse than the optimal.

5.3 The `OutDegree` algorithm

For tree $\mathcal{T} = (V, E, p)$, the `OutDegree` algorithm picks the k active nodes with the highest weighted out-degree in the influence tree $\mathcal{T}$. The complexity is defined by the computation of these degrees and the time required for sorting them. Therefore, the total running time is $\mathcal{O}(n + n \log n)$.

Our experiments with real data show that `OutDegree` often performs well in practice. However, there are also cases where the solutions reported by `OutDegree` are far from optimal. For example, consider the influence tree in Figure 3. The tree has n nodes, 2ℓ of which are active (black nodes) and $(n - 2\ell)$ are inactive (white nodes). That is, apart from nodes $u_1, \ldots, u_\ell$ and $w_1, \ldots, w_\ell$ all other nodes are inactive. We assume that all edges go from nodes closer to the root to nodes closer to the leaves of the tree and all weights are equal to 1. If we use `OutDegree` to solve the k-Effectors problem with $k = \ell$, the algorithm will report solution $S = \{u_1, \ldots, u_\ell\}$, with cost $C(S) = (n - 2\ell)$. However, the optimal solution is $S^* = \{w_1 \ldots, w_\ell\}$, with cost $C(S^*) = \ell$. This is because none of the $(n - 2\ell)$ inactive nodes will get activated. Therefore, `OutDegree` can report solutions that are $\frac{n-2k}{k} = \mathcal{O}(n)$ times worse than the optimal.

5.4 Finding effectors in forests

So far, we have assumed that the influence tree is connected. Here, we show how to allocate the budget of k effectors among the trees of an influence forest. We show that this can be achieved by another dynamic-programming recursion.

If we use $\mathcal{F}$ to denote this forest consisting of L trees, $\mathcal{T}_1, \ldots, \mathcal{T}_L$, then we need to find the optimal way of distributing the k effectors to these L trees. Recall that, if for a tree $\mathcal{T}_i$ we assign budget $k_i \leq k$ effectors, then we can compute the optimal set of k_i effectors on this tree using the dynamic-programming recursion given by Equation (3). Let $\textsc{Opt}(\mathcal{T}_i, k_i)$ be the solution obtained using the `DP` algorithm on tree $\mathcal{T}_i$. Then, the optimal solution on forest $\mathcal{F}$ is calculated again using dynamic programming: let $\mathcal{T}_1, \ldots, \mathcal{T}_L$ a random but fixed ordering of the trees in the forest $\mathcal{F}$ and $\textsc{Gl}(\ell, c)$ be the cost of the optimal assignment of $c \leq k$ effectors on the first ℓ trees $\mathcal{T}_1, \ldots, \mathcal{T}_\ell$. Then, $\textsc{Gl}(L, k)$ will give the optimal solution to our problem. The values of the Gl table are given using the following dynamic-programming recursion:

$$\textsc{Gl}(\ell, c) \quad = \quad \min_{0 \leq c' \leq c} \textsc{Gl}(\ell - 1, c - c') + \textsc{Opt}(\mathcal{T}_\ell, c').$$

This dynamic programming recursion is a generic method of allocating the budget of k effectors to the connected components of the input graph. We presented it here for the case of forests because for trees we can compute $\textsc{Opt}(\mathcal{T}_i, c)$. However, this computation cannot be done (or approximated) in polynomial time within each component of an arbitrary graph (see Lemma 1 and Lemma 2).

6. EXTRACTING THE INFLUENCE TREE

While the input influence graphs may not be trees, we show here how one can extract an influence tree from an arbitrary graph. Given $G = (V, E, p)$, our goal is to extract the influence tree $\mathcal{T}$ that captures most of the information in G. We quantify the optimization problem using a maximum-likelihood approach.

For a tree $\mathcal{T} = (V_T, E_T, p)$ with $E_T \subseteq E$, we compute the *likelihood* of $\mathcal{T}$ as follows:

$$L(\mathcal{T}) \quad = \quad \prod_{(u \to v) \in E_T} p(u \to v).$$

Therefore, our goal is to extract the influence tree $\mathcal{T}$ that maximizes $L(\mathcal{T})$. In fact, instead of maximizing the likelihood we minimize the negative log-likelihood. That is,

$$\textsc{mll}(\mathcal{T}) \quad = \quad - \sum_{(u \to v) \in E_T} \log p(u \to v). \qquad (4)$$

Our approach for constructing the influence tree is query-dependent. That is, given the set of active nodes in G, we extract the influence tree $\mathcal{T}$ that *spans all the active* nodes in G and minimizes Equation 4. We call this subproblem the Active Tree problem and the extracted influence tree the *active tree* of G.

Unfortunately, solving the Active Tree problem is NP-hard. In fact, the problem is identical to the Directed Steiner Tree problem. In the Directed Steiner Tree problem the input consists of a directed weighted graph $G' = (V', E')$ a specified root $r \in V'$ and a set of terminals $X' \subseteq V'$. The objective is to find the minimum-cost tree rooted at r and spanning all the vertices in X' (i.e., r should have

a path to every vertex in X'). Our setting is identical; the required nodes are the set of active nodes in G.

Here, we use the following efficient heuristic for constructing the *active* tree for a given influence graph: first, we construct the set of nodes R that consist of all the nodes in V that have no incoming edges. For each such root node $r \in R$ and for each node $s \in S$ we compute the shortest path from r to s in $\mathcal{T}$. Let $\mathcal{T}(r)$ be the tree consisting of the union of the edges in such shortest paths for the root node r. We then report as a solution the tree $\mathcal{T} = \arg\min_{r \in R} w'(\mathcal{T}(r))$. We call this simple algorithm the `dSteiner` algorithm.

So far we have assumed that the influence graph G is connected. If this is not the case we can follow one of the following two alternatives: (a) Find the strongly-connected components of the graph and then apply `dSteiner` independently in every component. This would output a forest of influence trees and, therefore, we can use the method described in Section 5.4. (b) Introduce an artificial node to the input graph is connect it via very low probability edges to all the nodes. This guarantees connectivity and allows us to use the `dSteiner` algorithm directly on this enhanced graph.

`dSteiner` can be replaced by any other approximation algorithm proposed for the directed Steiner tree problem [21, 5]. However, since the focus of the paper is not on the study of methods for the directed Steiner tree problem, we only use the `dSteiner` algorithm for our experimental evaluation. `dSteiner` requires a simple all-pairs shortest path computation and it is much less computationally demanding than the majority of other existing methods for the same task.

Discussion: Our approach for extracting the influence tree from the influence graph finds the most probable tree that spans all active nodes. Therefore, different activation vectors lead to different trees. We believe that for large graphs, where only some of the nodes are active, it makes sense to extract influence trees that are item-dependent. Alternatively, one could construct the influence tree to be item-independent. That is, one could try to extract from G the tree that spans *all* the nodes in G and minimizes Equation (4). This problem is equivalent to solving the directed minimum-cost spanning tree (D-MST) problem on a directed graph G. Such a tree can be extracted in polynomial time [6, 8]. It can then be used for all activation vectors. The performance of this approach depends on the portion of active nodes in the input activation vector. For a small number of active nodes, it would create influence forests with a small number of active nodes per component. In fact, further experimental analysis verified this intuition. Due to space constraints we do not report these results here.

7. EXPERIMENTS

In this section we evaluate the proposed algorithms for the k-EFFECTORS problem using the co-authorship graph extracted from the DBLP data. Our evaluation focuses on (a) showing indicative results from our methods and (b) evaluating the quality of the results with respect to the objective function.

7.1 The DBLP dataset

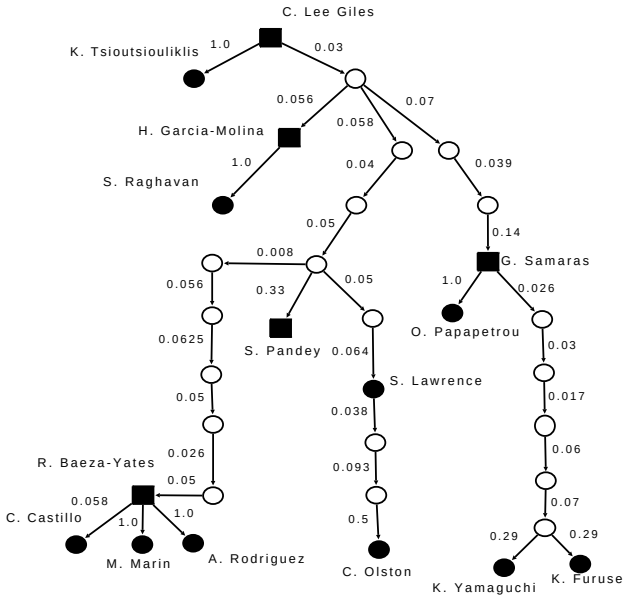
Using a snapshot of the DBLP data taken on April 12, 2006 we create a benchmark dataset for our experiments. We only keep entries of the snapshot that correspond to papers published in the areas of *Database* (DB), *Data mining* (DM), *Artificial intelligence* (AI) and *Theory* (T) conferences. Given this snapshot we create the network $G_{\text{dblp}} = (V, E, p)$ as follows: nodes in V correspond to authors; an author is included in V if she has at least three papers in the data. Each author i is associated with a set of terms S_i; these are the terms that appear in at least two titles of papers that i has co-authored. This process creates $|V| = 5508$ individuals and a set of 1792 distinct terms. Each term $t \in S_i$ is also associated with a timestamp $T_i(t)$, i.e., the year first used by author i. Two authors i, i' are connected by an edge in G_{dblp} if they co-authored at least two papers. The weight of the directed edge $(i \to i')$ is computed using the following simple rule:

$$p(i \to i') = \frac{|\{t \mid t \in S_i \wedge t \in S_{i'} \wedge T_i(t) < T_{i'}(t)\}|}{|S_{i'}|}$$

That is, we compute the probability that an item appearing in i' is a result of the influence of node i on i'.

We focus our experiments on activation vectors for 15 terms that correspond to research themes in computer science. The list of these 15 terms is shown in the first column of Table 1. For each term q, we extract the corresponding activation $\mathbf{a}_q$ so that $\mathbf{a}_q(i) = 1$ if $q \in S_i$. Given graph G_{dblp} and activation vector $\mathbf{a}_q$, we compute the active tree associated with q, denoted by $\mathcal{T}_q$, using the `dSteiner` algorithm (see Section 6). We always use those trees to identify the set of effectors using one of the three effector-finding algorithms for trees: `DP`, `OutDegree` and `Sort`.

Figure 4: Influence Tree $\mathcal{T}_q$ for q ="crawling". The tree is extracted from the original G_{dblp} influence graph.



7.2 An illustrative example

We start by showing an indicative output of our approach on the tree $\mathcal{T}_q$ for q ="crawling".[3] The active tree $\mathcal{T}_q$ is shown in Figure 4. The black nodes are active with respect

[3]The choice of the term was guided by the size of its active tree, which proved small enough to visualize.

Table 1: Cost of the solutions reported by DP, OutDegree, Sort and Random algorithms on the active trees for 15 distinct terms.

Term	DP	OutDegree	Sort	Random
collaborative filtering	31.40	34.19	34.19	40.13
graphs	558.75	560.41	558.75	582.92
wavelets	16.39	16.73	17.40	19.95
pagerank	2.33	4.20	4.20	4.20
privacy	47.09	47.56	50.22	59.59
clustering	514.94	520.74	519.10	560.99
classification	343.54	344.44	343.54	361.86
xml	382.59	385.29	382.59	418.01
svm	20.29	21.15	21.15	27.92
crawling	0.49	3.07	4.03	4.07
semisupervised	25.25	25.45	25.31	30.66
boosting	86.02	89.08	86.02	98.82
microarrays	24.35	28.93	29.07	42.46
streams	275.72	279.16	279.68	300.84
active learning	11.62	12.49	12.49	18.55

to the term, while the white nodes are inactive. Next to every active node, we show the name of the author it represents. For every edge we also display its weight in $\mathcal{T}_q$. We extract the effectors on this tree using DP and $k = 5$. The black square nodes are the 5 effectors chosen by the algorithm. The choices made by the algorithm are intuitive. C. Lee Giles covers K. Tsioutsiouklikis whom he influences with high probability. Similarly, H. Garcia-Molina covers S. Raghavan and G. Samaras covers O. Papapetrou. R. Baeza-Yates is also picked as an effector, due to the high influence probabilities to his co-authors. S. Pandey is the only leaf node chosen as an effector; this is simply because there are no high-probability paths to him from other active nodes. On the other hand, there are some active nodes (e.g. C. Olston) that are not chosen as effectors. This is because there was not enough budget and the algorithm determined that selecting the other nodes benefited the objective function. In fact, when we increased the budget to $k = 6$, C. Olston was the only new addition to the set of effectors. For $k = 7$, K. Furuse was also included, even though choosing K. Yamaguchi would clearly result in the same overall cost.

7.3 Comparison of effector-finding algorithms

This section evaluates the different algorithms for the k-Effectors problem with respect to the cost function $C()$. We use the activation vectors for the 15 terms shown Table 1 and construct the 15 different active trees (one tree per term). Then, we run the DP, Sort and OutDegree algorithms on each of the 15 trees. In addition to these three algorithms, we also evaluate Random; an algorithm that randomly picks the effectors on a given input tree. The performance of all the algorithms with respect to the objective function and $k = 10$, is shown in Table 1. Recall that, since our problem is one of cost minimization, the lower the value, the better the performance of the algorithm. Also, since DP is optimal, its cost serves as the baseline for the other algorithms.

As we can see from the table, the Random algorithm is clearly worse than the others for all 15 terms. In contrast, the Sort and OutDegree algorithms report solutions with costs consistently close to the optimal (achieved by the DP algorithm), for most of the terms in the table.

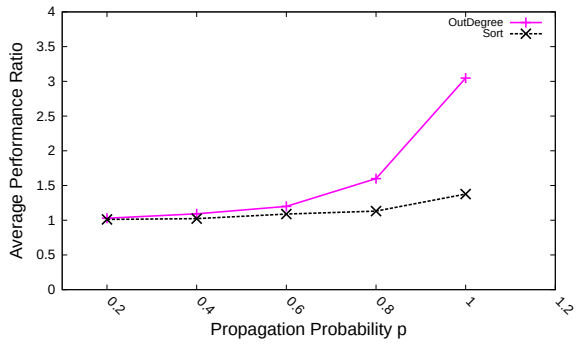


Figure 5: Average performance ratio of Sort and OutDegree for 15 trees $\mathcal{T}_q$ with modified influence probabilities p set uniformly across all the edges; $p \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$.

The near-optimal performance of the two algorithms is way beyond the expectations set by the worst-case analysis of Sections 5.2 and 5.3. This can be explained by the structure of the G_{dblp} graph: many prolific and highly influential authors are also good effectors, particularly for terms with a large number of active nodes. This clearly helps Sort and OutDegree, since they favor such nodes.

In order to evaluate the algorithms under different scenarios we proceed as follows: first, we generate the active tree $\mathcal{T}_q$ for each of the terms in Table 1, as in the previous experiment. Then, we replace the actual influence probabilities with some constant probability $p \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. Therefore, for every term q, we construct 5 different instances of the $\mathcal{T}_q$ tree and apply the DP, Sort and OutDegree algorithms on each of them. Our motivation is to moderate the effects of highly influential nodes, thus making it more challenging to identify the set of effectors. In Figure 5, we plot the average *performance ratio* of the algorithms (ratio of the cost of the solution reported by an algorithm divided by the cost of the optimal solution reported by DP), for the different values of p. The closest the ratio is to 1, the better. The average is taken over all the 15 trees.

The results show that the performance of OutDegree and Sort deteriorates as the value of p approaches 1. In particular, for higher values of p, the performance ratio of OutDegree is significantly higher than 1. This can be explained by the fact that, as the edge weights approach 1, an increasing number of nodes gain high-probability paths to many other nodes. As a result, the weighted criterion used by OutDegree to pick effectors loses its advantage and the performance of the algorithm deteriorates. A similar argument can be made for Sort: the algorithm favors nodes with high-probability paths to the active parts of the network. When such paths exist for most of the nodes, it becomes harder for the algorithm to identify the optimal set. The variance values of the ratios reported in Figure 5 for $p = \{0.2, 0.4, 0.6, 0.8, 1\}$ are $\{0.02, 0.04, 0.18, 0.23, 0.51\}$ for Sort and $\{0.03, 0.14, 0.28, 0.89, 2.55\}$ for OutDegree, respectively. Note that OutDegree is more susceptible than Sort in making incorrect choices as the value of of p increases. As a result, we observe larger values of variance in the ratios observed by OutDegree.

Table 2: The $k = 10$ effectors reported by the DP algorithm for terms {graphs, XML, Collaborative Filtering}.

Graphs	XML	Collaborative Filtering
A. Brandstadt	A.Zhou	A. Nakamura
A. Z. Broder	D. Srivastava	B. Mobasher
C. Faloutsos	E. A. Rundensteiner	D. Heckerman
D. Peleg	F. Bry	D. Poole
F. Hurtado	H. V. Jagadish	F. Yang
F. T. Leighton	J. Srinivasan	H.-P. Kriegel
N. Linial	M. Krishnaprasad	J. M. Kleinberg
N. Alon	O. Diaz	M. Li
S. Leonardi	S. Pal	R. S. Zemel
W. Wang	T. Milo	W. Du

7.4 Qualitative evidence

Next, we present qualitative evidence of the results obtained by optimally solving the k-EFFECTORS problem on G_{dblp}. Our motivation is to show that, in a realistic setting, the results we obtain are reasonable and intuitive. Table 2 shows the results obtained using the optimal DP algorithm to solve the k-EFFECTORS problem on three different $\mathcal{T}_q$ trees for $k = 10$. We report the results for three terms q = {graphs, XML, and Collaborative Filtering}. We purposefully select terms from three popular -albeit different- areas of computer science, in order to capture results coming from diverse parts of the G_{dblp} graph.

A quick observation indicates that the reported sets of effectors include some very prolific authors. This can be verified by checking the overall number of papers per author, as recorded in the **DBLP** dataset. In fact some of the authors have over 150 papers (e.g,. D. Peleg (213), N. Alon (250), H. V. Jagadish (156), E. A. Rundensteiner (191), H.-P. Kriegel (214)). The number of papers serves as an indicator of the author's influence in the graph. Authors with more papers typically have more distinct coauthors and are active with respect to more terms. Also, recall that we operate on the active tree $\mathcal{T}_q$, extracted so that it mostly consists of active nodes associated with a term. As a result, prolific authors are likely to be chosen as effectors, since they have high-probability paths to many of these active nodes. However, authors with relatively small number of papers are also included as effectors. An intuitive explanation for this is the following: even though well-connected nodes can be reasonable effectors that explain a large part of the observed activation vector, they are also more likely to be connected to inactive nodes. As a result, selecting only highly-connected nodes as effectors increases the overall cost of the solution. Overall, the set of effectors can include nodes of variable connectivity and influence, as long as they can best describe the given activation state of the network.

Although the reported effectors per term are all from the general area of computer science indicated by the term itself, each one of them covers a different sub-community. For example, for the term "Collaborative Filtering", we can see D. Heckerman and D. Poole – both effectors for the machine-learning community – J. Kleinberg – an effector for the theory community – and H-P. Kriegel – an effector for the database community. Further, many of the effectors come from different geographical regions, and, thus, act as effectors for different sets of authors. In fact, further analysis showed that the reported effectors have small overlap in their sets of co-authors.

Similar observations can be made for the other two terms. For example, for the term "graphs", C. Faloutsos is an effector for the data-mining community, while the majority of the other authors are effectors that cover different parts of the theory community. Again, the number of common co-authors between every pair of the reported effectors is very small.

8. CONCLUSIONS

Given a network where a subset of the nodes are active, and a probabilistic propagation model, we defined the problem of finding the subset of active nodes that best explain the observed activation state. We called these nodes *effectors*. We studied the complexity of the k-EFFECTORS problem in directed graphs and trees. For general directed graphs, we showed that the k-EFFECTORS problem is NP-hard to solve or even to approximate. However, we showed that for directed trees the problem can be solved optimally in polynomial time via dynamic programming. We also presented a general framework, where, given a directed influence graph and an activation vector, we first extract the most probable active tree that spans all the active nodes in the network. We then use the dynamic-programming algorithm to identify the optimal set of effectors in this tree. In our experimental evaluation, we demonstrated that our algorithms perform well with respect to our objective function. The reported sets of effectors provide useful insight about the network and the interactions between the nodes. In the future, we plan to further explore the utility of effectors in other types of networks, including computer and biological graphs.

9. REFERENCES

[1] N. Agrawal, H. Liu, L. Tang, and P. S. Yu. Identifying the influential bloggers in a community. In *WSDM*, pages 207–218, 2008.

[2] J. Aspnes, K. L. Chang, and A. Yampolskiy. Inoculation strategies for victims of viruses and the sum-of-squares partition problem. *J. Comput. Syst. Sci.*, 72(6), 2006.

[3] S. Bharathi, D. Kempe, and M. Salek. Competitive influence maximization in social networks. In *WINE*, pages 306–311, 2007.

[4] D. Chakrabarti, Y. W. 0008, C. Wang, J. Leskovec, and C. Faloutsos. Epidemic thresholds in real networks. *ACM Trans. Inf. Syst. Secur.*, 10(4), 2008.

[5] M. Charikar, C. Chekuri, T.-Y. Cheung, Z. Dai, A. Goel, S. Guha, and ming Li. Approximation algorithms for directed steiner problems. *J. Algorithms*, 33:73–91, 1999.

[6] Y. Chu and T. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

[7] P. Domingos and M. Richardson. Mining the network value of customers. In *KDD*, pages 57–66, 2001.

[8] J. Edmonds. Optimum branchings. *J. Research of the National Bureau of Standards*, 71B:233–240, 1967.

[9] R. Fagin, R. V. Guha, R. Kumar, J. Novak, D. Sivakumar, and A. Tomkins. Multi-structural databases. In *PODS*, pages 184–195, 2005.

[10] 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.

[11] J. Goldenberg, B. Libai, and E. Muller. Using complex systems analysis to advance marketing theory development. *Academy of Marketing Science Review*, 2001.

[12] D. Gruhl, D. Liben-Nowell, R. V. Guha, and A. Tomkins. Information diffusion through blogspace. *SIGKDD Explorations*, 6(2):43–52, 2004.

[13] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.

[14] M. Koivisto. Advances in exact bayesian structure discovery in bayesian networks. In *UAI*, 2006.

[15] R. Kumar, K. Punera, and A. Tomkins. Hierarchical topic segmentation of websites. In *KDD*, pages 257–266, 2006.

[16] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429, 2007.

[17] H. Mannila and E. Terzi. Finding links and initiators: a graph-reconstruction problem. In *SDM*, pages 1207–1217, 2009.

[18] M. Mathioudakis and N. Koudas. Efficient identification of starters and followers in social media. In *EDBT*, pages 708–719, 2009.

[19] P. Rusmevichientong, S. Zhu, and D. Selinger. Identifying early buyers from purchase data. In *KDD*, pages 671–677, 2004.

[20] A. Tamir. An $o(pn^2)$ algorithm for the p-medial and related problems on tree graphs. *Operations Research Letters*, 19:59–64, 1996.

[21] A. Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18(1):99–110, 1997.