Data-Aware Vaccine Allocation over Large Networks

Yao Zhang, Department of Computer Science, Virginia Tech. Email: yaozhang@cs.vt.edu.
B. Aditya Prakash, Department of Computer Science, Virginia Tech. Email: badityap@cs.vt.edu.

Given a graph, like a social/computer network or the blogosphere, in which an infection (or meme or virus) has been spreading for some time, how to select the k best nodes for immunization/quarantining immediately? Most previous works for controlling propagation (say via immunization) have concentrated on developing strategies for vaccination *pre-emptively* before the start of the epidemic. While very useful to provide insights in to which baseline policies can best control an infection, they may not be ideal to make *real-time* decisions as the infection is progressing.

In this paper, we study how to immunize healthy nodes, in presence of already infected nodes. Efficient algorithms for such a problem can help public-health experts make more informed choices, tailoring their decisions to the actual distribution of the epidemic on the ground. First we formulate the *Data-Aware Vaccination* problem, and prove it is NP-hard and also that it is hard to approximate. Secondly, we propose three effective polynomial-time heuristics `DAVA`, `DAVA`-prune and `DAVA`-fast, of varying degrees of efficiency and performance. Finally, we also demonstrate the scalability and effectiveness of our algorithms through extensive experiments on multiple real networks including large epidemiology datasets (containing millions of interactions). Our algorithms show substantial gains of up to *10 times* more healthy nodes at the end against many other intuitive and non-trivial competitors.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications–*Data mining*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Graph Mining, Social Networks, Immunization, Diffusion

1. INTRODUCTION

Given a contact network, and some already infected people, which healthy persons should be immediately given vaccines to best control the epidemic? Similarly, given the follower-followee network and some rumors spreading on it, which accounts should Twitter decide to suspend/delete/warn to stop the misinformation as fast as possible? Propagation-style processes on graphs/networks are important tools to model situations of interest in real-life like in epidemiology, in cyber-security and in social-systems. For instance, infectious diseases spreading over contact networks, malware propagating over computer networks, spam/rumors spreading on Twitter, all are propagation-style processes. Hence, controlling and stopping such malicious propagation is a natural and significant problem with a large number of applications.

In this paper, we focus on the problem of how to select the best nodes to distribute vaccines on a network, when the disease has already spread over some parts of the network. Intuitively speaking, we want to study how to best build a "wall" in the network against an already spreading contagion. We assume the vaccines completely "immunize" nodes, i.e. they are removed from the network. Most previous work in immunization is only interested in designing algorithms and policies for so-called *pre-emptive* immunization (see Section 2 for a detailed survey), in which they try to find the best baseline strategies for controlling an epidemic before the epidemic has started spreading. As we discuss later, these papers assume that the epidemic can start anywhere on the network at any random point. However,

DOI:http://dx.doi.org/10.1145/2803176

in practice this assumption is almost never realistic—e.g., children and elderly are more susceptible to the flu, and peoples working with animals are more likely to get infected (say in case of avian flu), and hence a flu-epidemic tends to start from them. Therefore, though such pre-emptive policies give us good baseline strategies, they may not be ideal for *responsive* decision-making if an epidemic has already infected some people. In this paper we study a novel "data-aware" immunization setting, which takes into account the current infections at the time of vaccine allocation. Efficient solutions for such a *Data-Aware Vaccination* problem can help policy makers make better decisions about how to best distribute vaccines. For example, one way our algorithms can eventually enable "real-time' vaccine allocation is by re-running our algorithms every hour or every day. The *Data-Aware Vaccination* problem can naturally be applied to cyber security and social media as well: which computers should install the anti-virus software first when malware attacks have already spread; which user accounts should be blocked in Twitter to best stop spam/rumor spreading?

Our main contributions include:

(1) *Problem Formulation and Hardness Results*: We formulate the *Data-Aware Vaccination* (*DAV*) problem on arbitrary graphs as an combinatorial optimization problem, and prove it is NP-hard and also hard to approximate within an absolute error. To the best of our knowledge, we are the first to address the *Data-Aware Vaccination* problem on arbitrary graphs (please see Related Work for more details).
(2) *Effective Algorithms*: We first propose DAVA-tree, an optimal algorithm for merged trees, then extend it to general graphs (DAVA algorithm). Furthermore, using careful pruning techniques, we present a faster algorithm, DAVA-prune, which gives the same result as DAVA. In addition, we provide a third heuristic, DAVA-fast, which is the fastest algorithm with some loss of performance.
(3) *Experimental Evaluation*: We present extensive experiments against several popular immunization algorithms on multiple real networks (including large epidemiological social-contact graphs), and demonstrate the efficacy and scalability of our algorithms. Our algorithms outperform other competitors by up to 10 times in both magnitude and running-time. In addition to that, we also compare DAVA-prune and DAVA-fast: DAVA-prune saves up to $16k$ more nodes than DAVA-fast in large networks while DAVA-fast is the fastest algorithm.

Outline of the paper: The rest of the paper is organized as follows. We first discuss the related work in Section 2, and then introduce some preliminaries in Section 3. Section 4 formulates the Data-Aware Vaccination problem, and Section 5 discusses the computational complexity of our problem. Section 6 presents our algorithms and Section 7 extends our algorithms to the Susceptible-Infected-Recovered (SIR) model. Experimental results on several datasets are in Section 8. We finally discuss future work in Section 9 and conclude in Section 10.

2. RELATED WORK

This paper is an extended version of our previous paper on the same topic [Zhang and Prakash 2014]. In this journal version, we add more related works including Epidemiology, Other Optimization Problems and Information Dissemination. In addition, we give all non-trivial proofs of our theorems including proofs for the hardness result and correctness of our algorithms. Furthermore, we propose a new prune algorithm, which is as accurate as the best algorithm we had proposed in the conference version of the paper but clearly much faster. And we give a more accurate way to extend our algorithms to the Susceptible-Infected-Recovered (SIR) model. Finally, in addition to giving more experimental results, we also show the effectiveness of our new algorithm.

To the best of our knowledge, except for [Zhang and Prakash 2014], the problem of immunization under the more realistic setting of *prior knowledge* on general *arbitrary graphs*, has not been studied before. However, our work is related to the following previous work: Epidemiology, Immunization Algorithms, Other Optimization Problems and Information Dissemination.

Epidemiology. The canonical textbooks and surveys discussing fundamental epidemiological models like Susceptible-Infected-Susceptible (SIS) and Susceptible-Infected-Recovered (SIR) include [Hethcote 2000; Anderson and May 1991]. Much work has gone into in finding epidemic thresholds (minimum virulence of a virus which results in an epidemic) for a variety of networks [Bailey 1975; McKendrick 1925; Anderson and May 1991; Kephart and White 1993; Pastor-Satorras and Vespignani 2002; Wang et al. 2003; Ganesh et al. 2005; Prakash et al. 2011; Meyers et al. 2006]. All the epidemic threshold work assume that the infection will start from a *random* node—in contrast, in the DAV problem we are given a *specific* set I_0 of initially infected nodes.

Immunization Algorithms. Most existing work deals with identifying optimal strategies for vaccine allocation *before* the start of the epidemic. The *acquaintance* immunization policy for both the SIS and SIR model has been proposed (pick a random person, and immunize one of its neighbors at random) [Cohen et al. 2003]. In addition, immunization problems on power law graphs have been studied as well [Madar et al. 2004] [Briesemeister et al. 2003]. E-mail virus attack on power-law computer networks under the so-called SHIR model (Susceptible, Hidden, Infectious, Recovered) has been described [Hayashi et al. 2003]. The problem of blocking links in a network has been studied in [Kimura et al. 2008]. And two formulations of the problem of blocking a contagion through edge removals under the model of discrete dynamical systems [Barrett et al. 2008] are given [Kuhlman et al. 2013]. A game of inoculation given a cost and loss-risk, under *random* starting points has been studied [Aspnes et al. 2005]. In a similar vein, a $O(\log n)$ approximation for minimizing 'societal' cost under a deterministic propagation model was given [Chen et al. 2010a]. Based on minimizing the largest eigenvalue of the graph, *pre-emptive* node-based and edge-based immunization algorithms for arbitrary graphs have been propose [Tong et al. 2010; Tong et al. 2012].

Many works also compare the performance of a limited number of pre-determined sequences of interventions (like school closure, antiviral for treatment) within simulation models [Ferguson et al. 2006; Halloran et al. 2008; Dushoff et al. 2007].

Finally, the most related work is [Yaesoubi and Cohen 2011]'s paper, which considered the prior information of the epidemic. However, their problem is a special case of ours: they only focused on the dynamic policies under a simplified flu-like model, and a *homogenous* population (i.e. every node is connected to every other node).

To summarize, none of the above works look into the problem of distributing vaccines under *prior information* and on *arbitrary networks.*

Other Optimization Problems. There are several node importance measurements in previous literature, including betweeness centrality [Freeman 1977; Newman 2005], PageRank [Page et al. 1998], HITS [Kleinberg 1998], and coreness score [Moody and White 2003]. The Independent Cascade (IC) model and the influence maximization problem has been introduced [Kempe et al. 2003], which aims to select nodes to infect so as to *maximize* the number of nodes infected at the end. In contrast, we want to pick nodes to be *removed*, under given information, to *minimize* the expected number of infections. Further, their goal was submodular and hence approximable, but as we show later, our problem does not have that structure. Other remotely related work is outbreak detection [Leskovec et al. 2007a] and finding most-likely 'culprits' of an infection [Lappas et al. 2010; Prakash et al. 2012] in the sense that we aim to select a subset of '*important*' nodes on graphs. All the above problems are also known to be NP-hard.

Information Dissemination. Virus propagation is closely related to many important dynamic processes on graphs. The recent paper [Matsubara et al. 2012] gave a Susceptible-Infected (SI) based model (related to the IC model used in this paper) unifying patterns in the popularity of online content exhibits on Twitter, blog posts and news media articles. There is also a lot of research interest in studying other types of information dissemination processes on large graphs, including (a) information cascades [Bikhchandani et al. 1992; Goldenberg et al. 2001], (b) blog propagations [Leskovec et al. 2007b; Gruhl et al. 2004; Kumar et al. 2003; Richardson and Domingos 2002], and (c) viral marketing and product penetration [Leskovec et al. 2006]. For example, we can choose which Facebook or Twitter accounts should be blocked to stop an existing spam and/or a rumor.

3. PRELIMINARIES

We give preliminaries in this section. Table I lists the main symbols used in the paper.

Table I. Terms and Symbols

Symbol	Definition and Description
DAV	Data-Aware Vaccination problem
SIR	Susceptible-Infected-Recovered Model
IC	Independent Cascade Model
$G(V,E)$	graph G with nodes set V and edges set E
I_0	infected nodes set
$p_{u,v}$	propagation probability from node u to v
δ	curing probability for the SIR model
k	the budget (i.e., #nodes to give vaccines to)
S	set of nodes selected for vaccination
$\sigma_{G,I_0}(S)$	the expected number of infectious nodes at the end (footprint)
$\sigma'_{G,I_0}(S)$	the expected number of healthy nodes at the end
$\gamma_S(j)$	the expected benefit of $\sigma'_{G,I_0}(*)$ when adding j into S

3.1. Contact Network

There exists an underlying contact network $G(V,E)$ (between people/computers/blogs etc.) on which the contagion (disease/virus/meme etc.) can spread, and we assume our graph is weighted and undirected. Each vertex $v \in V$ represents an individual of the network and edges represent the interaction between these individuals. In this paper, edge weight $p_{u,v}$ represents the propagation probability of the contagion from u to v.

3.2. Virus Propagation Models

We use two widely used discrete-time virus propagation models to describe how the virus spreads on the network: the so-called Susceptible-Infected-Recovered (SIR) model, and the Independent Cascade (IC) model (a special case of SIR model).

Susceptible-Infected-Recovered Model. SIR, a fundamental model which has been extensively used in epidemiology [Hethcote 2000; Anderson and May 1991; Shah and Zaman 2011], is used to model mumps (or chicken-pox) like infections. In the SIR model, there are three states for each node in the network: susceptible (healthy), infected and recovered. In each time-step, each infected node u tries to infect each of its healthy neighbors v *independently* with probability $p_{u,v}$ (the weight on each edge). The healthy neighbors who are successfully infected will be infected from the next time-step. In addition, at each time-step each infected node has a curing probability δ to become recovered (from the next step). Once recovered, a node can not participate in the epidemic further. The process begins when some initial nodes are infected and ends when no infected nodes remain.

Independent Cascade Model. IC is a well-known model [Kempe et al. 2003; Chen et al. 2010b; Goyal et al. 2010; Lin et al. 2013] which is used to describe viral marketing and related meme processes. In contrast to SIR, in the IC model, each infected node has exactly one chance to infect ('activate') its neighbors independently with the propagation probability (in effect the curing probability $\delta = 1$ here). The process of the IC model proceeds as follows. It is a discrete-time model, in which a node in the graph can be either infected or healthy. When a node u first becomes an infectious node at timestep t, it is given a single chance to infect each of its currently healthy neighbors w, which it succeeds with probability $p_{u,w}$ (the weight of the edge $\{u, w\}$ in the contact graph G). If u succeeds, then w will become infectious at time-step $t + 1$. If multiple neighbors of w first become infectious at time-step t, then their activation attempts are sequenced in an arbitrary order, but all performed at time-step t. Whether or not u succeeds, it cannot make any further attempts to infect w in subsequent rounds. The process terminates when no additional node becomes infectious.

We will first describe our algorithms in the IC model. Then, we will see that extending our algorithms to the general SIR model is reasonably straightforward (which we explain in Section 7).

4. PROBLEM FORMULATION

4.1. The "Data" in Data-Aware Vaccine Allocation

In this paper, we are interested in effectively allocating vaccines in the network given a set of initially infected nodes (I_0) and we call it *Data-Aware Vaccine Allocation.* Note that the "initially" infected node set I_0 does not *necessarily* refer to the infected nodes at the beginning of the epidemic. Instead, it refers to the set of infected nodes as observed whenever we decide to allocate the vaccines. Loosely speaking, we call I_0 the "starting point" or "initially" infected node set for ease of our description, but it does not mean the starting point of the epidemic. The size of I_0 is determined by how severe an epidemic is when we observe it and decide to perform an intervention. For instance, if it is a deadly disease which was undetected till a long time, then the I_0 will be relatively large. Generally speaking, our goal is to design a robust allocation strategy which can handle any size of I_0.

We assume I_0 is known, and can be drawn from any distribution. In practice, there are several ways to obtain I_0. In particular, in social media such as Twitter, the accounts that post a certain rumor can be sampled from Twitter API [Morstatter et al. 2013]. In epidemiology, we can get infectious cases from hospital reports or CDC documents [Medlock and Galvani 2009]. In addition, there are large-scale micro-scale realistic simulations on how to generate infection spreads [Eubank et al. 2004].

Note that in our paper, for the observability of infected nodes set I_0, we assume a simple scenario: as long as individuals have become symptomatic, and been diagnosed as infected (hospitals will report the diagnosed cases), we assume they are infected. In social media, this is easier and straightforward. We leave more complicated conditions of the observability of I_0 to future work.

4.2. Problem Definition

Now we are ready to formulate our problem formally. We are given a fixed-set of nodes I_0, which contains all infected nodes at certain time of the epidemic process as we discussed above. We assume that if a vaccine is given to a *healthy* node v, v cannot be infected by its neighbors at any time, which means v is effectively *removed* from the graph. We are given a graph $G(V, E)$, the set I_0 and a budget of k vaccines. We want to find the 'best' set S of healthy nodes that should be given vaccines at the beginning. As the propagation model is stochastic, let us denote $\sigma_{G,I_0}(S)$ to be the expected total number of infected nodes at the end of the process (the 'footprint'), given that I_0 nodes were infected at the start, and S was the vaccinated set. The best set S is the one which minimizes $\sigma_{G,I_0}(S)$. Formally, PROBLEM 1: *Data-Aware Vaccination* problem $DAV(G, I_0, P, \delta, k)$.

GIVEN: *A graph $G(V, E)$ with node set V and edge set E, the infected node set I_0, SIR model with propagation probability on each edge $\{i, j\}$ $p_{i,j} \in P$ and curing probability δ, and an integer (budget) k.*

FIND: *A set S of k nodes from $V - I_0$ to distribute vaccine to minimize $\sigma_{G,I_0}(S)$, i.e.*

$$\begin{aligned} S^* &= \operatorname*{argmin}_S \sigma_{G,I_0}(S) \\ s.t.\ |S| &= k \end{aligned} \tag{1}$$

Comment 1. Define $\sigma'(\cdot)$ to be the expected number *healthy* nodes at the end i.e. $\sigma'_{G,I_0}(S) = |V| - \sigma_{G,I_0}(S)$. Given the same set S and an integer k, clearly minimizing $\sigma_{G,I_0}(S)$ is equivalent to maximizing $\sigma'_{G,I_0}(S)$ (as nodes can either be infected or healthy). In this paper, for ease of description we adopt this alternate form (of maximizing $\sigma'_{G,I_0}(S)$).

Comment 2. Clearly the problem is trivial when $k \geq |N(I_0) - I_0|$, where $N(I_0)$ is the set of immediate neighbors of I_0 in the graph G (as we can just vaccinate all of these nodes, and the disease will stop). In reality, this is never the case, as vaccines are expensive and networks are huge. For example, for $k = 10$, in our experiments, we found that $|N(I_0) - I_0|$ ranged from 10 to 250 times our budget. Hence, we will implicitly assume that $k < |N(I_0) - I_0|$.

5. COMPLEXITY OF THE DAV PROBLEM

We discuss the complexity of the Data-Aware Vaccination problem in this section. In summary, we first prove that *on general graphs*, the DAV problem is NP-hard, then show that DAV problem is also hard to approximate.

5.1. Hardness result

Unfortunately, our problem is NP-hard. We will reduce it from the MINIMUM K-UNION (MINKU) set problem (where we want to minimize the union of k subsets), which was proven to be hard [Vinterbo 2004].

Consider the corresponding decision version of DAV:

PROBLEM 2: Data-Aware Vaccination (Decision Version) $DAV(G, I_0, P, \delta, k, \tau)$:

GIVEN: *A graph $G(V, E)$, the node set I_0, the SIR model with propagation probability $p_{i,j} \in P$ and curing probability δ, and an integer (budget) $k \geq 0$, and $\tau \geq 0$.*

FIND: *Is there a set S of k nodes of G to distribute the vaccine such that $\sigma'_{G,I_0}(S) \geq \tau$?*

THEOREM 5.1. *DAV (Decision Version) is NP-hard.*

PROOF. We prove that DAV is NP-hard via a reduction from the NP-Complete Minimum k-union Problem (MINKU) problem [Vinterbo 2004].

MINIMUM K-UNION PROBLEM (DECISION VERSION). *given a collection of n subsets $C = \{S_1, S_2, ..., S_n\}$ over a finite set of elements $U = \{u_1, u_2, ..., u_m\}$ and an integer $0 \leq k < n$, and $\tau \geq 0$, is there exactly k subsets $S_{j1}, ..., S_{jk}$ in S such that $|S_{j1} \cup ... \cup S_{jk}| \leq \tau$*

Note that [Vinterbo 2004] only proves that MINKU is NP-hard. But given an instance of MINKU (Decision Version), it is easy to check whether $|S_{j1} \cup ... \cup S_{jk}| \leq \tau$ in polynomial time, so MINKU $\in NP$ as well and hence MINKU (Decision Version) is NP-complete.

Given an instance of MINKU, we construct a graph G with uniform propagation probability $p = 1$, and $\delta = 1$ (essentially IC model, a special case of SIR). Let each subset S_i be a node in the graph. Also let there be a node j in the graph for each element u_j. Then a sole infected node I connects to the all nodes in the set $\{S_1, S_2, ...S_n\}$ and subsequently each S_i connects to its elements j. So G has $1 + m + n$ nodes. Clearly this construction takes polynomial time. Figure 1 shows an example where $S_1 = \{1\}$, $S_2 = \{1, 2\}$, $S_3 = \{1, 2\}$, $S_4 = \{m\}$, $S_5 = \{m\}$, and $S_n = \emptyset$.

Based on the graph G, we can create an instance $IN \equiv (G, I, P, \delta, n-k, n-k+m-\tau)$ of DAV. Now we need to show that (1) if the instance of MINKU problem has an solution, then the instance IN also has the solution; (2) if the MINKU Problem does not have a solution, then the instance IN also does not have a solution.

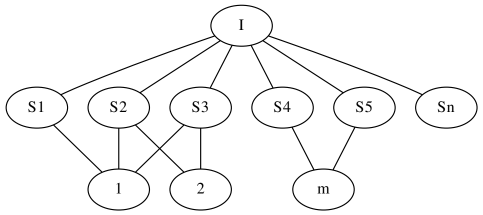


Fig. 1. Graph used in the reduction from Minimum k-union Problem.

(1). This is true. If we can select k subsets $S_{j1}, ..., S_{jk}$ for the MINKU problem, then we can choose other $n-k$ nodes from the collection C for our DAV problem. In this case, $\sigma_{G,I}(S) = k + |S_{j1} \cup ... \cup S_{jk}| + 1 \leq \tau + k + 1$, so $\sigma'_{G,I}(S) = |V| - \sigma_{G,I_0}(S) = m + n + 1 - \sigma_{G,I_0}(S) \geq m + n - k - \tau$.

(2). This is also true. Suppose the instance I for DAV has a solution, we will show that MINKU Problem also has a solution which is a contradiction. We can assume that the solution set is S_0, so we have $\sigma'_{G,I}(S_0) \geq m + n - k - \tau$.

Firstly note that because of the construction of the graph G, if an assignment contains nodes from $1, 2, ...m$, we can always swap them with nodes named as $S_1, S_2, ...S_n$ to get an alternate assignment which is at least as good as the original. Note that if we choose a node t from $1, 2, ...m$, then the value of $\sigma'_{G,I}(S)$ (the number of nodes we block) only increases by one. This is so as it is impossible to block nodes S_i without choosing them (all the nodes $S_1, S_2, ...S_n$ are directly connected to I). Moreover t has connections only with nodes S_i. So t can not block any other nodes. Finally, if we choose a substitute node from $S_1, S_2, ...S_n$, it is possible that some additional nodes in $1, 2, ...m$ may be blocked. Hence, swapping t with a node in $S_1, S_2, ...S_n$ will result in a solution which is at least as good.

Hence we can assume without loss of generality that the optimal solution S^* contains only nodes from S_i. Then we have $\sigma'_{G,I}(S^*) \geq \sigma'_{G,I_0}(S_0) \geq m + n - k - \tau$, so $\sigma_{G,I}(S^*) = n + m + 1 - \sigma'_{G,I}(S^*) \leq \tau + k + 1$. In this case S^* contains $n-k$ nodes, if we choose other k nodes $S_{j1}, ..., S_{jk}$ from $C - S^*$ as the solution for MINKU problem, then we have $|S_{j1} \cup ... \cup S_{jk}| = \sigma_{G,I}(S^*) - 1 - k \leq \tau$. This means the MINKU problem has a solution which is a contradiction.

Combining (1) and (2) proves the hardness. □

5.2. Approximability

Typically related optimization problems arising in graphs have a submodular structure lending themselves to the near-optimal greedy solution. But unfortunately, our function is *not* submodular.

Remark 5.2. $\sigma'_{G,I_0}(S)$ in DAV is not a submodular function.

PROOF. See Fig. 2. A submodular function has the property that if $A \subseteq B$, then adding an element j into both sets, we should have $f(A \cup \{j\}) - f(A) \geq f(B \cup \{j\}) - f(B)$. Suppose I is infected and $A = \emptyset$ and $B = \{X\}$, we have $\sigma'_{G,I_0}(A \cup \{Y\}) - \sigma'_{G,I_0}(A) = 5$ and $\sigma'_{G,I_0}(B \cup \{Y\}) - \sigma'_{G,I_0}(B) = 8 - 2 = 6$. So $\sigma'_{G,I_0}(S)$ is not a submodular function. □

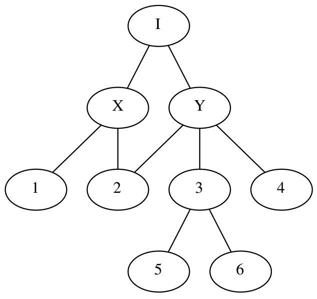


Fig. 2. Counter-Example.

Vinterbo [Vinterbo 2004] gave a greedy algorithm which can approximate MinKU and its equivalent problem Maximum k-Intersection problem (MaxKI) (where we want to *maximize* the *intersection* of k given subsets) within a constant factor if the cardinality of all the subsets are bounded by a constant. However, in our case the 'subsets' can be very large, and thus the approximation result is not useful. The algorithm we develop is related to Vinterbo's setting in the sense that it is also greedy, though we use different efficient techniques for our particular setting. For the general case, Xavier [Xavier 2012] proved that the MaxKI problem cannot be approximated within a tighter constant $\frac{1}{N^\epsilon}$ where N is the number of subsets and $\epsilon > 0$, under $\mathbf{NP} \not\subset \mathbf{BPTIME}(2^{N^\epsilon})$. Recently Sheih et al [Shieh et al. 2012] proved MaxKI is inapproximable within an absolute error, with a smaller inapproximable gap and under the weaker $\mathbf{P}\neq\mathbf{NP}$ assumption. Using the results in [Shieh et al. 2012], unfortunately our problem is also inapproximable within an absolute error $\frac{1}{2}m^{1-2\epsilon} + O(m^{1-3\epsilon})$ if $m = |V| - |N(I_0) - I_0| - |I_0|$. Here, m means the number of nodes except for the infected nodes and their neighbors.

Theorem 5.3. *Given any constant $0 < \epsilon < 1/3$, there exists a m_ϵ such that the Data-Aware Vaccination problem with $m > m_\epsilon$, cannot be approximated in polynomial time within an absolute error of $\frac{1}{2}m^{1-2\epsilon} + \frac{3}{8}m^{1-3\epsilon} - 1$ unless **P=NP**.*

Proof. The Minimum k-union Problem (MinKU) is equivalent to the Maximum k-intersection Problem (MaxKI) (where we want to *maximize* the *intersection* of k given subsets) [Vinterbo 2004] . Shieh [Shieh et al. 2012] proved that MaxKI problem[1] of universe size $m \geq m_\epsilon$ cannot be approximated in polynomial time within an absolute error $\frac{1}{2}m^{1-2\epsilon} + \frac{3}{8}m^{1-3\epsilon} - 1$ unless $\mathbf{P}=\mathbf{NP}$. Using the reduction in the NP-hard proof, we can prove that the DAV problem is hard to approximate if MaxKI cannot be approximated in polynomial time within an absolute error. □

6. OUR PROPOSED METHODS

Due to the results in the previous section, we present effective heuristics next. We describe our methods assuming the IC model in the DAV problem in this section, and then extend them to handle the general SIR case (next section).

***Roadmap of this section:*.** We first simplify the DAV problem by merging infected nodes. Then we propose an optimal solution called `DAVA-TREE` on trees. Based on `DAVA-TREE`, we give an effective algorithm, `DAVA`, on any arbitrary graph. However, `DAVA` is not scalable to large networks, hence we provide a faster algorithm, `DAVA`-prune, which returns the same result as `DAVA`. In addition to that, we propose a much faster heuristic, `DAVA`-fast in the end.

[1]Interestingly, the related *inverse* problem of *minimizing* the intersection, equivalent to the max-coverage problem, has a $1 - 1/e$ approximation.

6.1. Simplification—Merging infected nodes

To simplify our problem, as the hardness reduction from the previous section suggests, we merge all the infected nodes into a *single* 'super infected' node and get an equivalent problem with only a single infected node. Intuitively this is because it does not matter how the infected nodes are connected among themselves—all it matters for our problem is how they are connected to *healthy* nodes. If a healthy node has multiple infected neighbors, it will have a new edge probability which would be the logical-OR of the individual probabilities. For example, if a healthy node c has two infected neighbors a and b with edge probabilities p_a and p_b, the new edge probability between I' and c would be $1-(1-p_a)(1-p_b) = p_a + p_b - p_a p_b$.

Remark 6.1. Given an instance of the DAV problem (G, I_0, P, k) under IC model, Algorithm 1 outputs an equivalent problem instance (G', I', P', k) where I' is the sole infected node in the new graph G'.

PROOF. We can prove it by induction. If a healthy node c connects to two infected nodes a and b, according to IC model, the propagation probability will be $1-(1-p_a)(1-p_b) = p_a + (1-p_a)p_b$ (equal to the probability of getting infected from at least one of a or b). Hence Algorithm 1 (Line 9) outputs the same propagation probability for c. When c has $l+1$ infected nodes, suppose $p_{I'c}$ is the propagation probability after merging l nodes, when merging a new infected node d, the propagation probability is $1-(1-p_{I'c})(1-p_d) = p_{I'c} + (1-p_{I'c})p_d$, which is the same as the output of Algorithm 1. □

Algorithm 1 MERGE

```
Require: Input graph $G$, infected node set $I_0$, probability set $P$
 1: $G' = G$
 2: Add node $I'$ to $G'$
 3: for each node $i$ in $I_0$ do
 4:   if there exists an edge $e_{ij}$ between $i$ and $j$ then
 5:     if there is no edge $e_{I'j}$ then
 6:       Add edge $e_{I'j}$ into $G'$
 7:       $p_{I'j} \leftarrow p_{ij}$
 8:     else
 9:       $p_{I'j} \leftarrow p_{I'j} + (1 - p_{I'j})p_{ij}$
10:     end if
11:     Remove $e_{ij}$ from $G'$
12:   end if
13: end for
14: Remove all nodes in $I_0$ from $G'$
15: return graph $G'$ and the infected node $I'$
```

In Algorithm 1, line 3-13 is used to copy edges from previous infected nodes set I_0 to the new infected node I'. Line 5-10 shows how to assign new propagation probability $p_{I'j}$. Suppose the previous infected nodes set I_0 has E_{I_0} edges in total, Algorithm 1 will take $O(|I_0| + E_{I_0})$ time.

6.2. DAVA-TREE—Optimal solution when the merged graph is a tree

Let us call the graph we get after merging (i.e. after Algorithm 1) the *m-graph*. The second important observation is that as we show next, if the *m-graph* of our instance is a tree, then we can get an optimal polynomial time algorithm under IC model, for any edge propagation probability $p_{i,j} \in [0, 1]$. We call the algorithm DAVA-TREE (Data Aware Vaccine Allocation on a tree).

Before we describe our algorithm, define the quantity $\gamma_S(j)$—which is the '*benefit*' of node j to the optimization goal when nodes in S have *already* been removed. It is essentially the expected number of nodes we save after removing j, given that we have already removed nodes from set S. We have:

$$\gamma_S(j) = \sigma'_{G,I_0}(S \cup \{j\}) - \sigma'_{G,I_0}(S) \tag{2}$$

Let $\gamma(j) = \gamma_{\emptyset}(j)$. Algorithm 2 proceeds by computing γ_j efficiently for every neighbor node of I' (in a simple tree traversal) and then taking those neighbors of the infected node I' with top-k $\gamma(\cdot)$ values. Lemma 6.2 proves that this gives us the optimal solution. In short, as there is only one path from any node to any other node in the tree: the optimal solution must be a subset of the immediate neighbors of I' with top k value of $\gamma(j)$.

Algorithm 2 DAVA-TREE

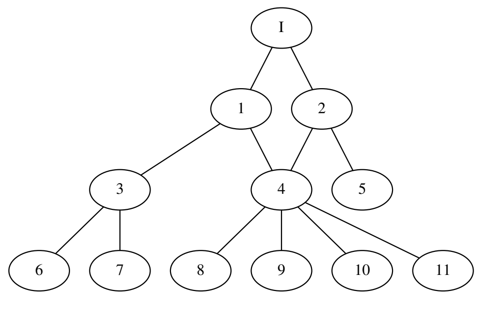
```
Require: Tree $T$, infected node $I'$, $k$ and $p_{ij}$
 1: Set $I'$ as the root
 2: for each neighbor $j$ of $I'$ do
 3:     $\gamma(j) \leftarrow p_{I'j} \times$ calPartial($j$)
 4: end for
 5: $S$ = nodes with top-$k$ values of $\gamma(j)$
 6: return $S$

   Function calPartial(node $n$)
   if $n$ is not a leaf then
       benefit $\leftarrow 1$
       for each child $i$ of $n$ do
           benefit $\leftarrow$ benefit + calPartial($i$) $\times p_{ni}$
       end for
   else
       benefit $\leftarrow 1$
   end if
   return benefit
   EndFunction
```

LEMMA 6.2 (CORRECTNESS OF DAVA-TREE). *If the m-graph $G(V, E)$ is a tree, then we can get optimal solution of the DAV problem under IC model by Algorithm 2.*

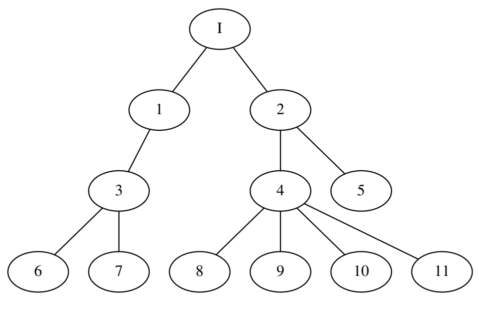
PROOF. We show the following things: in the optimal set, the chosen nodes must be neighbors of the infected node I'; the benefit of each such node is independent of the rest of the set S; and finally that we correctly calculate $\gamma(j)$.

First the nodes we select must be neighbors of the infected node I'. Suppose a solution set $M = \{S \cup v\}$ (for some S) selects a node v that is not a neighbor of I', and since $G(V, E)$ is a tree there is only one path p from I' to v. If in S there is no other node in the path p, then instead of v, we can choose the neighbor of I' in the path p, say node u. be smaller if we select u instead of v. Note that v can get infected only if u gets infected. Hence, $\sigma'_{G,I'}(S \cup \{u\}) - \sigma'_{G,I'}(S \cup \{v\}) > 0$ as u can save at least *one more* node from being infected than v. On the other hand, if except v there is a node in p in S, then we know that $\sigma'_{G,I'}(S \cup \{v\}) = \sigma'_{G,I'}(S)$, so we can choose some another neighbor of I' instead of v, and get a better value. Hence in either case, we can swap v with a neighbor of I' and get a better solution than M.

Second we must select k neighbors of I' with top values of $\gamma(j)$. For a S that contains only I''s neighbors, note that $\sigma'_{G,I'}(S) = |V| - \sigma_{G,I'}(S) = |V| - (\sigma_{G,I'}(\emptyset) - \Sigma_{i \in S}\gamma(i)) =$



(a) Original Graph.



(b) Minimum Spanning Tree.

Fig. 3. An example of minimum spanning tree. For p=1 and k=1, the optimal solution is node 2 in MST. However, in the original graph, the optimal solution is node 4.

$\sigma'_{G,I'}(\emptyset) + \Sigma_{i \in S}\gamma(i)$. In other words, the benefit of each node is independent of S. This is because the subtree of each node $j \in S$ is disconnected from the subtree of any other node $i \in S$: hence node $i \in S$ can not impact any other node $j \in S$. Thus we can just choose the top-k values of γ to maximize $\sigma'_{G,I'}(S)$.

Finally, note that Line 3 in the `DAVA-TREE` calculates $\gamma(j)$ correctly. See that $\gamma(j) = \sigma'_{G,I_0}(\{j\}) - \sigma'_{G,I'}(\emptyset) = \sigma_{G,I'}(\emptyset) - \sigma_{G,I'}(\{j\})$. As each subtree is independent, this quantity is equal to the probability that j is infected times the expected number of *sick* nodes in node j's subtree, when j is infected. It is easy to see that the function *calPartial* exactly computes the latter value, and so we get the correct value of $\gamma(j)$. $\square$

LEMMA 6.3 (RUNNING TIME OF `DAVA-TREE`). *Algorithm 2* `DAVA-TREE` *costs* $O(|V|+|E|+k \log |V|)$ *time in the worst case.*

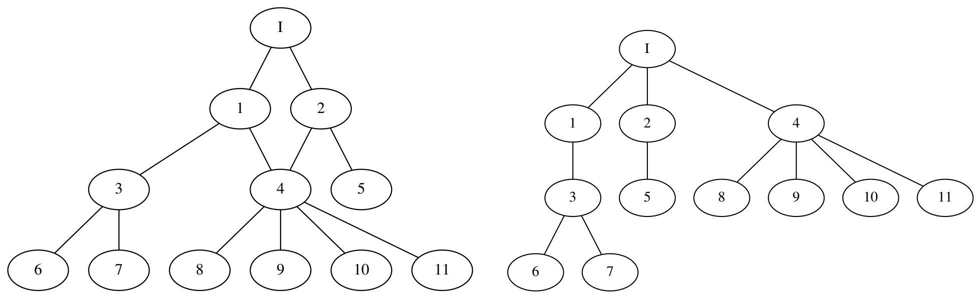
PROOF. Calculating the expected benefit needs a complete tree traversal, so it takes $O(|V|+|E|)$ time. Suppose I' has m neighbors, selecting top k nodes needs $O(k \log m)$ using a heap. $\square$

6.3. `DAVA`—An effective algorithm on arbitrary graphs under IC model

What if the m-graph is not a tree? We give an effective heuristic next when m-graphs are arbitrary networks. After the merge algorithm, we can guarantee that a connected graph has only one infected node I'. Intuitively, we need to capture (a) the '*closeness*' of nodes to the infection (represented by I') and at the same time, (b) the importance of the nodes in '*saving*' other nodes. Thus good solutions are composed of nodes which are close to I' and also prevent the infection of many others.

We can still use `DAVA-TREE` algorithm by generating a tree from the m-graph, rooted at I'. Which tree should we use? People have typically used spanning trees like the Minimum Spanning Tree (MST) in related problems. The problem with MST is that potential solution nodes (for the original graph) can reside at higher depths in the MST—but as we saw in the previous section, the `DAVA-TREE` algorithm only chooses nodes which are neighbors of I'. See Figure 3 for an example. Let $p = 1$ on all edges, and budget $k = 1$. Then the MST rooted at I' will not have the node 4 as a neighbor of node I', but the optimal solution in this case will exactly be node 4. On the other hand, if $p = 0.5$ and $k = 1$, then it is easy to verify that node 1 will become the optimal solution.

Dominator tree. We propose to use the *dominator tree* of the graph and then run `DAVA-TREE` algorithm on the dominator tree. As we explain later, the dominator tree avoids this problem, by precisely capturing the 'closeness' property required from the solutions.



(a) Original Graph. (b) Dominator Tree.

Fig. 4. An example of dominator tree. For p=1 and k=1, the optimal solution is node 4. For p=0.5 and k=1, the optimal solution is node 1.

In graph theory, given a source node I', a node v dominates another node u if every path from I' to u contains v. Node v is the immediate dominator of u, denoted by $v = idom(u)$, if v dominates u and every other dominator of u dominates v. We can build a dominator tree rooted at I' by adding an edge between the nodes u and v if $v = idom(u)$. Dominator trees have been used extensively in studying control-flow graphs. Building dominator trees is a very well-studied topic, with near-linear time algorithms available [Lengauer and Tarjan 1979; Buchsbaum et al. 1998]. Figure 4(b) shows the dominator tree for graph in Figure 4(a). Note that the edges in the dominator tree may not in fact exist in the original graph (compared to say the MST).

Note that we have not specified how to weight the edges of this dominator tree yet. Even the simple unweighted version of the dominator tree has structural properties which are very useful for the DAV problem. As we show in the next lemma, the *optimal* solution for the original graph can only be a subset of the neighbors of I' in the *unweighted dominator* tree.

LEMMA 6.4. *For the DAV problem, the optimal solution should be the children of root I' in the unweighted dominator tree of the m-graph G.*

PROOF. Suppose the optimal solution set S^* contains a node v which is not the neighbor of I' in the dominator tree. So there must exist a node u that is the child of I', and it dominates v in the original graph (as the dominating relationship is transitive). This means that all paths from I' to v have to pass through u. In other words, node v does not get infected if u is not infected. The rest of the proof is similar to the argument used in Lemma 5.1. In sum, we can swap v with a neighbor of I' and get solution which is at least as good. $\square$

Note that by building the dominator tree we can reduce the search space substantially without losing any information—we demonstrate this in experiments as well, the number of neighbors of I' in the dominator tree is typically a fraction of the total number of nodes in the original graph. Further, we can prove that if $p = 1$, running `DAVA-TREE` on dominator tree of m-graph G returns the best *first* node.

LEMMA 6.5. *For the special case when the budget $k = 1$ and propagation probability $p = 1$, running algorithm `DAVA-TREE` on dominator tree T of m-graph G weighted with $p_{u,v}$ as above, gives the optimal solution.*

PROOF. According to Lemma 6.4, the node v we select should be a child of root in the dominator tree. Also, when $k = 1$, by the definition the best node to pick is the one with the max. value of $\gamma(\cdot)$. The only nodes v can prevent from infection are those nodes u for which v lies on every path from I' to u (as removing v will cut off all possible paths of infection)—or the nodes exactly in the subtree of v in T. And for those nodes that are not in the subtree of v in T, when v is removed, they are still infected since they have at least one path from I' to them in the original graph. So the value of $\gamma(\cdot)$ for v in the original graph is the number of nodes for which v lies on every path from I' to them. This means the value of $\gamma(\cdot)$ in the original graph is equal to the value of $\gamma(\cdot)$ in T. So the best solution for the dominator tree is also the best solution for the original graph. $\square$

Weighting the dominator tree. DAVA-TREE assumes that the edges in the network denote propagation probabilities. We want to preserve such information (coming from the original graph) in the dominator tree, to make the 'benefit' computation accurate. Hence we weight each edge $\{u, v\}$ in the dominator tree by $p_{u,v}$, the total probability that node u can infect v in the *original graph* (note that u and v may not be neighbors in the original graph).

Lemma 6.5 and the preceding discussion suggest a natural greedy heuristic: find the best single node i using DAVA-TREE in the dominator tree (weighted with $p_{u,v}$ as defined before); then remove node i from the graph; recompute the dominator tree; and repeat till budget k is exhausted. We call this algorithm BASIC.

***Speeding up* BASIC.** Unfortunately, computing $p_{u,v}$ for a given pair of nodes u and v in the IC model is #P-complete [Chen et al. 2010b]. It is essentially the canonical $s-t$ connectivity problem in random graphs. We can use Monte-Carlo sampling to get an estimate through simulations, but even that is too slow. Hence we propose to approximate it by using the *maximum propagation path probability* between nodes u and v, which is intuitively the most-likely path through which an infection can spread from node u to v in the original graph.

In the original graph, suppose $p_{pathi}(u, v)$ means the propagation probability from u to v through path i. We define *maximum propagation path probability* $\tilde{p}_{i,j}$ as the maximum value of $p_{pathi}(u, v)$. Here we can use $\tilde{p}_{u,v}$ as the approximate propagation probability for the edge $\{u, v\}$ in the dominator tree. Max. path probability has been used before in context of the influence maximization problem [Chen et al. 2010b], but they need to compute it between all pairs of nodes. On the other hand, in our problem, we need to compute it only for edges in the dominator tree of the graph. In fact, further, it is easy to see that in a dominator tree rooted at I', if $v = idom(u)$, then $\tilde{p}_{v,u} = \frac{\tilde{p}_{I',u}}{\tilde{p}_{I',v}}$. This means we only need to calculate the maximum propagation path probability from root I' to all other nodes, which is similar to find shortest-paths in graph theory.

Hence a faster algorithm than BASIC would be to assign probabilities $\tilde{p}_{v,u}$ in the dominator tree. We call the complete algorithm DAVA for arbitrary graphs. Pseudocode is given in Algorithm 3.

Algorithm 3 DAVA Algorithm for Arbitrary Networks

```
Require: Graph $G$, $P$, budget $k$, infected set $I_0$
1: $S = \emptyset$
2: $G'$ = Run MERGE on $G$ and $I_0$
3: repeat
4:     $T$ = Build the dominator tree from $G'$ and assign probabilities $\tilde{p}_{v,u}$
5:     $v$ = Run DAVA-TREE on $T$ with budget = 1
6:     $S = S \cup v$
7:     Remove node $v$ from $G'$
8: until $|S| = k$
9: return $S$
```

LEMMA 6.6. *(Running time of* `DAVA`*) Algorithm 3 takes $O(k(|E|+|V|\log|V|))$ worst-case time.*

PROOF. Building a dominator tree costs $O(|V|+|E|)$ time [Buchsbaum et al. 1998]. If the original graph has the uniform propagation probability, we are able to obtain $\tilde{p}_{I',v}$ by through Breadth-First Search (BFS) which takes $O(|V|+|E|)$ time. Otherwise we can use Dijkstra's Algorithm to get $\tilde{p}_{I',v}$ which takes $O(|E|+|V|\log|V|)$ time. Thus, assigning probabilities costs $O(|E|+|V|\log|V|)$ worst-case time. Removing node v also takes linear time. □

6.4. `DAVA`-prune—A faster algorithm with the same result as `DAVA`

`DAVA` works fine on small graphs, but can be slow on large graphs, as it re-builds the dominator tree multiple times. Hence we propose an faster algorithm `DAVA`-prune, which speeds up `DAVA` using prune techniques while returning the same result as `DAVA`. (Note: we also present *another* costlier pruning algorithm, which is still faster than `DAVA` but slower than `DAVA`-prune. We describe it in the appendix as it may be of independent interest).

Building a dominator tree costs linear time, while weighting a dominator tree using maximum propagation probability takes $O(|E|+|V|\log|V|)$ time. Hence, the bottleneck for `DAVA` is the reweighting process. If we are able to reduce the reweighting time, we can speed up `DAVA`. The following lemmas (Lemma 6.7, Lemma 6.8 and Lemma 6.9) will give us some guidance to reduce the time complexity of reweighting. Basically, the idea is that we do not need to recalculate weights for some nodes, and we can reduce the complexity of recalculating necessary weights as well.

To recap, we have the original graph G, *m-graph* G', and the infected node I' in G'. Let us denote v to be the node selected from G' using `DAVA`, G_{new} to be the new graph after removing v from G', T_{old} to be the old dominator tree before removing v, and T_{new} to be new (re-built) dominator tree.

First, the following two lemmas will show that we do not need to reweight nodes that are not in the first layer of the old dominator tree T_{old}.

LEMMA 6.7. *After removing the selected node v from the m-graph G' using* `DAVA`*, nodes that are not in the first layer of the old dominator tree T_{old} will have the same parent nodes in the new (re-built) dominator tree T_{new}.*

PROOF. First, we prove that for any node i that is neither in the first layer of the old dominator tree T_{old} nor under the subtree of v in T_{old}, after removing v, i's neighbors in G_{new} will not change, which means i's neighbors in G' cannot be under the subtree of v in T_{old}. Here we use a non-trivial property of the dominator tree from [Lengauer and Tarjan 1979] that in the graph the dominators of a node a is the intersection of its neighbors' dominators. Since i's dominators is the intersection of its neighbors' dominators, after removing v, i's dominators in T_{new} cannot increase. Furthermore, i must have the same dominators in T_{new}: if i has a smaller number of dominators in T_{new}, at least one of i's neighbor j in G' must be under the subtree of v in T_{old}. Since i is not in the subtree of v in T_{old}, there exists at least one path from I' to i in G' which does not include v. So there exists at least one path from I' to j as well such that v is not in that path. Hence, j is not under the subtree of v in T_{old}, which contradicts our assumption. Therefore, none of i's neighbor is under the subtree of v in T_{old}. Similarly, for any node i that is not in the first layer of T_{old}, i's neighbors must be under the same subtree as i.

Now we prove the lemma by induction. First, consider node u in the second layer of T_{old}, as we proved above, u's neighbors cannot under the subtree of v in T_{old}, and they are at the same subtree of T_{old} as u, hence the intersection of dominators of u's neighbors must contain u's parent in T_{old}. Therefore, u's parent remains the same in the new dominator tree T_{new}. Second, suppose for nodes under the lower layer L of T_{old} where $2 < L \leq N$, they have the same parents in T_{new}. Then, consider node u under the $N+1$ layer of T_{old}. If u's neighbors are under the higher layer, their parents will be the same in T_{new}, which means they have the same immediate dominators in T_{new}. And for nodes that are under the lower layer, their dominators is also the dominators of u, hence u's parent will be the same node in T_{new}.

Therefore, nodes that are not in the first layer of T_{old} will have the same parent nodes in the new dominator tree T_{new}. □

LEMMA 6.8. *After removing the selected node v from G' using* DAVA*, nodes that are not in the first layer of the old dominator tree T_{old} will have the same edge-weights to their parents in the new (re-built) dominator tree T_{new}.*

PROOF. Suppose i is the node not in the first layer of T_{old}, and j is its parent. According to Lemma 6.7, j is still i's parent in the new dominator tree T_{new}. $p_{j,i}$ is the probability from the maximum propagation path from j to i in G'. If the maximum propagation path from j to i does not contain nodes under the subtree of v in T_{old}, clearly $p_{j,i}$ remains the same in T_{new}.

Let us assume there exists a node n that is under the subtree of v in T_{old}, and also in the maximum propagation path from j to i. Since j is not under the subtree of v in T_{old}, there must exist at least one path from I' to j in G' that does not contain v. Since n is in the maximum propagation path from j to i, there must have a path from I' to n which goes through j but does not contain v. Hence, n cannot be dominated by v, which contradicts our assumption that n is under the subtree of v in T_{old}. Therefore, the maximum propagation path from j to i does not contain nodes under the subtree of v in T_{old} which means $p_{j,i}$ will stay the same in T_{new}. □

Though all the edge-weights of nodes that are not in the first layer of T_{old} remain the same in T_{new}, we cannot ignore these nodes because they may be in the maximum propagation path from I' to nodes in the first layer of T_{new}. However, luckily the next lemma will show that we do not need to consider nodes that that not in the first layer of T_{old} when calculating weights of nodes in the first layer of T_{new} (the maximum propagation path from I' to nodes that are in the first layer of T_{new}).

LEMMA 6.9. *After removing the selected node v from G' using* DAVA*, in the new (re-built) dominator tree T_{new}, the maximum propagation path from I' to nodes that are in the first layer of the old dominator tree T_{old} only contains nodes in the first layer of T_{old}.*

PROOF. First, for node j in the first layer of T_{old}, either j only has one neighbor, I', in G', or there exist at least two distinct paths from I' to j. Here, distinct paths mean there is no common nodes in these paths. If there exist common nodes in different paths from I' to j, j cannot be in the first layer of T_{old}.

If j only has one neighbor I' in G', then clearly in the new dominator tree T_{new}, $p_{I'j}$ remains the same. Now let us consider the case that there exist at least two distinct paths from I' to j. Suppose there exists a node n that is not in the first layer of T_{old}, but belongs to a path from I' to j. Since j is not the direct neighbor of I' in G', there must exist at least two distinct path from I' to j in G'. Hence for node n, there also exist at least two distinct paths from I' to n in G', which means n only has one dominator, that is I'. Hence n is in the first layer of T_{old}, which contradicts the assumption that n is not in the first layer of T_{old}. Hence, every path from I' to node j in the first layer of T_{old} can only contain nodes that are in the first layer of T_{old}.

Therefore, after removing v, in T_{new} the maximum propagation path from I' to j only contains nodes in the first layer of T_{old}. □

Lemma 6.7 shows that if a node u is not in the first layer of T_{old}, u's parent in the new dominator tree T_{new} remains the same, and furthermore according to Lemma 6.8, the maximum propagation probability from u's parent to u in T_{new} will not change as well. In addition, Lemma 6.9 demonstrates that to calculate new edge-weights for nodes in the first layer of T_{new}, we do not need to run Dijkstra's algorithm on the whole graph G'. Instead, using the fact that maximum propagation paths from I' to node j in the first layer of T_{old} only contain neighbors of I' in T_{old}, we can get the new edge-weight $p_{I'j}$ by running Dijkstra's algorithm on the subgraph with only I' and nodes in the first layer of T_{old}.

Hence we propose a faster heuristic than `DAVA`. We call this algorithm `DAVA`-prune. Pseudocode is given in Algorithm 4. `DAVA`-prune rebuilds the dominator tree after selecting a node but only reweights the edge-weights of those nodes in a subgraph of G' (see Lines 8-10). The next lemma shows that `DAVA`-prune returns the same result as `DAVA`.

Algorithm 4 `DAVA`-prune Algorithm

```
Require: Graph $G$, $P$, budget $k$, infected set $I_0$
 1: $S = \emptyset$
 2: $G'$ = Run MERGE on $G$ and $I_0$
 3: $T'$ = Build the dominator tree from $G'$ and assign probabilities $\tilde{p}_{v,u}$
 4: repeat
 5:    $v$ = Run DAVA-TREE on $T_{old}$ with budget = 1
 6:    $S = S \cup v$
 7:    Remove node $v$ from $G'$
 8:    $T'$ = Build the dominator tree from $G'$
 9:    $G''$ =Removing nodes that are not in the first layer of $T$ on $G'$
10:    Run Dijkstra's algorithm on $G''$ to get new weights
11: until $|S| = k$
12: return $S$
```

LEMMA 6.10. *(Correctness of* `DAVA`*-prune)* `DAVA`*-prune returns the same result as* `DAVA`.

PROOF. Lemma 6.8 shows that for nodes that are not in the first layer of the old dominator tree T_{old}, all of their edge-weights remain the same, while Lemma 6.9 indicates that for node u in the first layer of T_{old}, the maximum propagation path from I' to u only contains nodes in the first layer of T_{old}. Hence, if we run Dijkstra's algorithm on G'' (a subgraph only containing nodes in the first layer of T), we can get the same the maximum propagation probability from I' to u in the new (re-built) dominator tree T_{new}. Therefore, `DAVA`-prune gives the same results as `DAVA`. □

LEMMA 6.11. *(Running time of* `DAVA`*-prune) Algorithm 4 takes* $O(|V|\log|V| + k(|E| + F\log F)$ *worst-case time where* F *is the number of nodes in the first layer of the dominator tree* T.

PROOF. Building a dominator tree costs $O(|V| + |E|)$ time [Buchsbaum et al. 1998], and weighting the dominator tree for the first take costs $O(|E| + |V|\log|V|)$. After that, reweighting dominator tree only takes $O(F\log F)$ time. Hence in general, the worst-case time complexity for `DAVA`-prune is $O(|V|\log|V| + k(|E| + F\log F)$. □

6.5. `DAVA`-fast—An even faster heuristic

Even with pruning techniques, `DAVA`-prune may still be slow on large graph, as it depends on the number of nodes in the first layer. Hence we propose an even faster heuristic `DAVA`-fast, which runs `DAVA-TREE` on the dominator tree with the full budget k, instead of running it with $k = 1$ after each step. Essentially we are picking neighbor nodes (in the dominator tree) of I' with the top-k $\gamma(\cdot)$ values. Pseudo-code is in Algorithm 5. `DAVA`-fast performs well in our experiments, with some loss of quality but at a fraction of the running time of `DAVA`.

Algorithm 5 DAVA-fast Algorithm

```
Require: Graph $G$, $P$, budget $k$, infected set $I_0$
1: $S = \emptyset$
2: $G'$ = Run MERGE on $G$ and $I_0$
3: $T$ = Build the dominator tree from $G'$ and assign probabilities $\tilde{p}_{v,u}$
4: $S$ = Run DAVA-TREE on $T$ with budget $=k$
5: return $S$
```

LEMMA 6.12. *(Running time of* DAVA*-fast) Algorithm 5 takes* $O(|E| + |V| \log |V|)$ *worst-case time.*

PROOF. Building a dominator tree costs $O(|V| + |E|)$ time [Buchsbaum et al. 1998], and assigning probabilities costs $O(|E| + |V| \log |V|)$ worst-case time. Running DAVA-TREE costs $O(|V| + |E| + k \log |V|)$ time. □

6.6. Discussion of proposed methods

We will discuss the performance of DAVA, DAVA-prune and DAVA-fast in terms of the structure of graph next.

DAVA. As we showed in Lemma 6.2, if the graph is a tree, DAVA gives the optimal solution. In addition to that, the total benefit of the set S (the number of nodes we can save after removing S) as computed from the dominator tree, denoting as $\sigma'_{Dom}(S)$, is related to the true benefit $\sigma'_{G,I'}(S)$ from the original graph.

LEMMA 6.13. $\sigma'_{Dom}(S) \leq \sigma'_{G,I'}(S)$.

PROOF. First, $\sigma'_{Dom}(S)$ equals to footprint of the diffusion from I' to nodes in the subtree of $u \in S_i$ after running IC model. For any node u in the set S, after removing u from the graph G, every node v under the subtree of u in the dominator tree will be isolated from the super node I' in G since every path from I' to v must go through u. So the benefit we can at least get is the footprint of the diffusion from I' to all nodes under the subtree of $u \in S$, which is exactly $\sigma'_{Dom}(S)$. Hence, $\sigma'_{Dom}(S) \leq \sigma'_{G,I'}(S)$. □

Lemma 6.13 shows the connection between the dominator tree and the original graph in terms of the nodes we save. Furthermore, it also demonstrates how our DAVA algorithm approximates the DAV problem: it maximizes the lower bound of the number of nodes we can actual save from the original graph.

However, if there are too many nodes in the first layer of the dominator tree, the bound in Lemma 6.13 will become loose, and DAVA will not perform well because under this scenario, we lose too much information from the original graph. Nevertheless we found in the experiments (Section 8.2), in practice, the number of nodes in the first layer of the dominator trees is not large enough (only a fraction of total number of healthy nodes in the graph). Hence, DAVA performs very well on real networks.

DAVA-***prune.*** We showed that DAVA-prune gives the same result as DAVA (Lemma 6.10). In addition to that, it is faster than DAVA because we can reweight the dominator tree in a smaller graph. However, if the number of nodes in the first layer of the dominator tree is large enough, DAVA-prune can be as slow as DAVA. As we discuss later, in practice the first layer of the dominator tree is a fraction of the size of the graph, so DAVA-prune is much faster than DAVA. Hence, since we cannot run DAVA on large networks, DAVA-prune can serve as a baseline for measuring the performance of faster heuristics. Additionally, as we discuss below, the difference in performance between DAVA-prune and DAVA-fast depends on the structure of the network. If a dominator tree changes dramatically, DAVA-prune will provide a much better performance compared to DAVA-fast, no matter how network size changes. Our experiments show that DAVA-prune can save $16,000$ more nodes (for the budget $k = 2000$) for large networks like PORTLAND. Furthermore, for small graphs like STANFORD, it can save almost 10% more nodes than DAVA-fast. Hence, even though DAVA-prune is not as fast as DAVA-fast, it is still a viable option for applications where effectiveness is very critical: to achieve high performance, one can afford to sacrifice some running time.

DAVA-***fast***. As is clear from the pseudo-code, if the dominator tree does not change after we select certain nodes, DAVA-fast will give the same result as DAVA and DAVA-prune. Suppose we remove node u in G' (the merged graph). The dominator tree will change only if there exists a node v such that after removing u, every path from I' to v in G' must go through a certain node. In practice, the number of such nodes v is not that large; hence, the performance of DAVA-fast is competitive. Experimental results also demonstrate that DAVA-fast performs well: though it is not as good as DAVA and DAVA-prune, it outperforms other baseline algorithms.

7. EXTENDING TO THE SIR MODEL

In this section, we will explain how to extend our solution to the SIR case. Recall that in the SIR model, as opposed to the IC model, a node u tries to infect its neighbor v multiple times. Suppose Z_u is the random variable denoting the number of time-steps u stays infected until recovery (this is also the number of time-steps that u tries to infect v). The probability that v gets infected by u is $B_{uv} = 1 - (1 - p_{u,v})^{Z_u}$. Note that Z_u has a geometric distribution $Pr(Z_u = z) = (1-\delta)^{z-1}\delta$, with the expectation $\mathbb{E}[Z_u] = \frac{1}{\delta}$ (δ is the curing probability). If we force u to be infected for only one time-step before recovering (as in the IC model), then $\beta_{u,v}$, the equivalent probability that u infects v successfully, can be approximated by the expectation of B_{uv}, $\mathbb{E}[B_{uv}]$.

LEMMA 7.1. *The expectation of B_{uv} (the probability that node v gets infected by a neighbor node u) in the SIR model is given by,* $\mathbb{E}[B_{uv}] = \frac{p_{u,v}}{1-(1-\delta)(1-p_{u,v})}$.

PROOF. First, $\mathbb{E}[B_{uv}] = 1 - \mathbb{E}[(1-p_{u,v})^{Z_u}] = 1 - \mathbb{E}[e^{Z_u \ln(1-p_{u,v})}]$. Let us denote $t = \ln(1-p_{u,v})$, then $\mathbb{E}[B_{uv}] = 1 - \mathbb{E}[e^{tZ_u}]$.

Since Z_u is a geometric distribution where $Pr(Z_u = z) = (1-\delta)^{z-1}\delta$, and $t \le 0 < -\ln(1-\delta)$, using its moment-generating function $\mathbb{E}[e^{tZ_u}] = \frac{\delta e^t}{1-(1-\delta)e^t}$ [Goldberg 1986], we can get:

$$\mathbb{E}[B_{uv}] = 1 - \frac{\delta e^t}{1-(1-\delta)e^t} = 1 - \frac{\delta(1-p_{u,v})}{1-(1-\delta)(1-p_{u,v})} = \frac{p_{u,v}}{1-(1-\delta)(1-p_{u,v})}$$

$\square$

According to Lemma 7.1 we can directly apply our algorithms to the SIR case, by just using an equivalent IC model with $\beta_{u,v}$ as the propagation probability where $\beta_{u,v}$ can be approximated by $\frac{p_{u,v}}{1-(1-\delta)(1-p_{u,v})}$.

Recall that the IC model is a special case of the SIR model where each infected node u has only one chance to infect its neighbor v with the probability $p_{u,v}$, which essentially means $\delta = 1$. So as a sanity check, if we apply Lemma 7.1 to the IC model, $\beta_{u,v}$ is exactly $p_{u,v}$, as expected (i.e. we recover the original weights).

8. EXPERIMENTS

In this section, we give an experimental evaluation of our algorithms. We describe our setup next. We conducted the experiments using a 4 Xeon E7-4850 CPU with 512GB of 1066Mhz main memory, and implemented the algorithms in Python[2].

[2]Code can be downloaded from http://people.cs.vt.edu/~yaozhang/code/tkdd-dava.

We seek to answer the following questions via our experiments:

(1) How do DAVA, DAVA-prune and DAVA-fast perform compared with the baselines?
(2) How effective DAVA-prune is compared to DAVA?
(3) What is the scalability of our algorithms?
(4) Does I_0 affect the performance of our algorithms?

8.1. Experimental Setup

Datasets. We run our experiments on multiple real datasets. In addition to trying to pick datasets of various sizes, we also chose them from different domains where the DAV problem is especially applicable. Table II shows datasets we used.

Table II. Datasets

Dataset	Model	#Vertices	#Edges
OREGON	IC	633	2172
STANFORD	IC	8929	53829
GNUTELLA	IC	10876	39994
BRIGHTKITE	IC	58228	214078
PORTLAND	SIR	0.5 million	1.6 million
MIAMI	SIR	0.6 million	2.1 million

OREGON: The Oregon AS router graph is a network graph collected from the Oregon router views. It contains 633 links among 2172 AS peers.[3] The contagion here can be thought of malware and computer-network viruses, which we want to control by shutting-off or patching relevant routers.

STANFORD: It is the Stanford CS hyperlink network from 2001, in which a web page links to another page.[4] We made the graph undirected, and chose the largest connected component—it contains 8929 nodes and 53829 links. Contagions here can be false information spreading through the webspace, and we want to prevent their spread by posting true information at strategic web pages.

GNUTELLA: It is a peer-to-peer network showing the snapshot of the Gnutella P2P file sharing network from August 2012. It contains 39994 links among 10876 peers. Similar to OREGON, we can control the spread of malware and harmful files by patching some important peers.

BRIGHTKITE: It is a friendship network from the SNAP dataset collection,[5] from a former location-based social networking service provider Brightkite[6], which consists of 58228 nodes and 214078 edges. As friends regularly frequent the same places, such location-based networks can be useful for the public-health.

PORTLAND: We experimented on this social-contact graph based on detailed microscopic simulations [NDSSL 2007], versions of which have been used in national smallpox modeling studies [Eubank et al. 2004]. We used the dataset we get by tracking all the different types of activities of about $0.5 million$ people (nodes) in the city of Portland, Oregon. The resulting graph had about $1.6 million$ edges (interactions). The edges were weighted with

[3]http://topology.eecs.umich.edu/data.html.
[4]http://www.cise.ufl.edu/research/sparse/matrices/Gleich/.
[5]http://snap.stanford.edu/data/index.html.
[6]http://www.brightkite.com.

the contact-times (in secs) between people.

`MIAMI`: It is another social-contact graph based on detailed microscopic simulations [NDSSL 2007] with about 0.6*million* people and 2.1*million* edges.

Settings. For the IC model, we use three illustrative settings: (a) Uniform probability $p = 0.6$; (b) Uniform probability $p = 1$; and (c) $p_{u,v}$ uniformly randomly chosen from $\{0.1, 0.5, 0.9\}$ (following literature [Chen et al. 2010b]). For the SIR model, since our socio-contact graphs have contact time between people [NDSSL 2007], we will use normalized contact time as the attack probability $p_{u,v}$, and set a uniform recovery rate $\delta = 0.6$. Note that the performance of our algorithms does not change when δ varies since when we change δ, edge weights in the network are changed (see Section 7).

For most of experiments, we randomly choose 100 nodes to be infected as the set I_0 for the IC model, and 300 nodes for the SIR model (as the graphs are larger). Using such setting, we aim to model the infection at the time early intervention happens. However, we also empirically study the effect of the distribution and size of I_0 on our algorithms (see Section 8.2.3 and 8.2.4 for more details).

We choose roughly 1% of the whole population as the budget k for large networks in our experiments. To get the expected number of healthy nodes after immunization, we run the processes (either IC or SIR) 1000 times and take the average.

Baseline Algorithms. We compare our algorithms `DAVA`, `DAVA`-prune and `DAVA`-fast against various other competitors to better judge their performance. Except for `RANDOM`, all baselines use weighted edges.

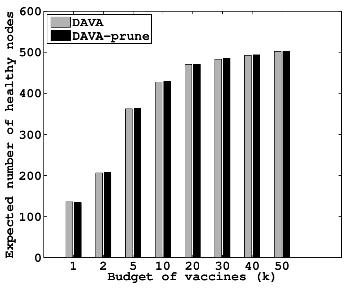
(1) `RANDOM`: In this method, we choose to give the vaccines to k uniformly randomly chosen *healthy* nodes.
(2) `DEGREE`: In this method, we choose to give the vaccines to the top-k healthy nodes according to their weighted degree. This is similar to the popular acquaintance immunization method [Cohen et al. 2003].
(3) `PAGERANK`: Here, we choose to give the vaccines to the top-k healthy nodes with the highest pagerank. `PAGERANK` is a popular node importance measurement which has been widely used in social media for many tasks [Page et al. 1998]. We use the restart probability of 0.15.
(4) `PER-PRANK`: In this method, we choose the top-k healthy nodes with the highest personalized pagerank with respect to the given infected nodes [Jeh and Widom 2003]. Intuitively, `PER-PRANK` takes into consideration how close the nodes are to the infected nodes set (I_0). We use the restart probability of 0.15.
(5) `NETSHIELD`: This is a state-of-the-art pre-emptive immunization algorithm [Tong et al. 2010], which aims to minimize the epidemic threshold of the graph. We take the top-k healthy nodes according to the algorithm (i.e. we get the ranking from this algorithm, and ignore nodes which are already infected in our problem).

Implementation Note. Lenguaer and Tarjan [Lengauer and Tarjan 1979] proposed two algorithms for constructing a dominator tree: a complicated near-linear time algorithm which takes $O(m\alpha(m, n))$ time where m is the number of edges, n is the number of nodes and $\alpha(m, n)$ is a functional inverse of Ackermann's function; and a simpler one which takes $O(m \log n)$ time. Buchsbaum et al [Buchsbaum et al. 1998] presented an exact linear-time dominator algorithm, but their algorithm ran slower on real flowgraphs than the algorithms in [Lengauer and Tarjan 1979]. We hence use the simpler $O(m \log n)$ algorithm in our implementation (common libraries also include this version typically).

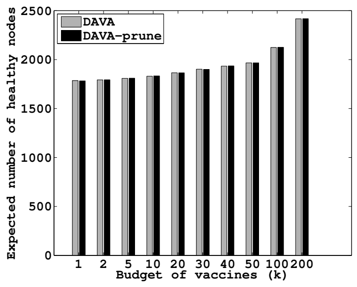
8.2. Experimental Results

We describe the results of our experiments next. First, we show that DAVA and DAVA-prune give the same results. Second, we demonstrate that DAVA, DAVA-prune and DAVA-fast get upto 10 *times* better solutions compared to the baselines (resulting in thousands of more healthy nodes), and DAVA-prune saves upto $16k$ (roughly 3% of the graph) more nodes than DAVA-fast for large networks. In addition, we demonstrate that the size of I_0 will not change the accuracy of our algorithms. Finally, we also show the scalability of our algorithms on large datasets.

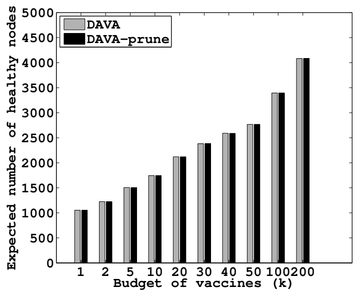
We would like to note here that the number of the nodes in the first layer of the dominator trees of the graphs were a fraction (ranging from 0.3 to 0.7) of the total number of healthy nodes in the graph. As discussed before, because of Lemma 6.4, this reduces the solution space substantially.



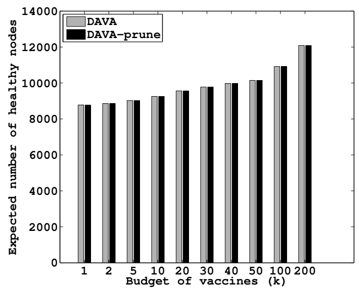
(a) OREGON



(b) GNUTELLA



(c) STANFORD

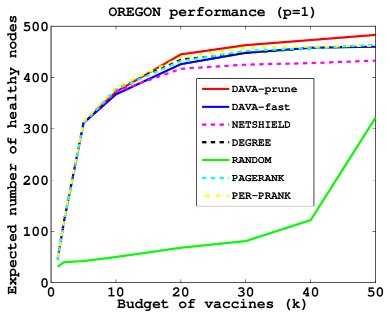


(d) BRIGHTKITE

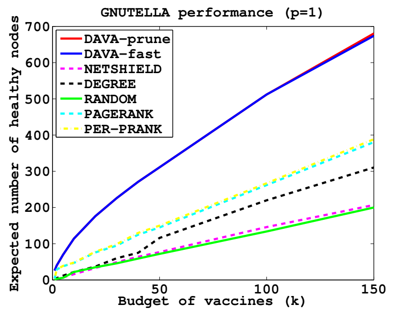
Fig. 5. Effectiveness of DAVA-prune: IC model with $p = \{0.1, 0.5, 0.9\}$ on various Real Datasets. Expected number of healthy nodes after distributing vaccines according to different algorithms (i.e. $\sigma'_{G,I'}(S)$) VS budget k. DAVA-prune outputs the same results as DAVA.

8.2.1. **Effectiveness of DAVA-prune**. We compare DAVA-prune with DAVA to demonstrate that DAVA-prune outputs *exactly* the same result as DAVA. Since DAVA is not scalable to PORTLAND and MIAMI, we do not show plots for them. also show how well the performance of DAVA-prune when comparing it with DAVA-fast.

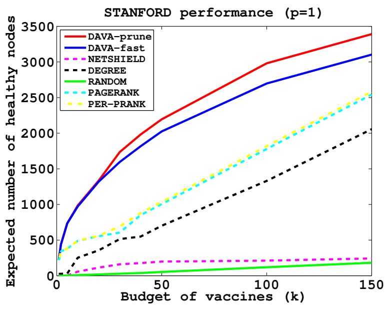
Figure 5 shows the results of DAVA-prune and DAVA when $p = \{0.1, 0.5, 0.9\}$ for OREGON, GNUTELLA, STANFORD and BRIGHTKITE: DAVA-prune always gives the same results as DAVA for all networks when we vary k, which is consistent with our theoretical result (see Lemma 6.10).



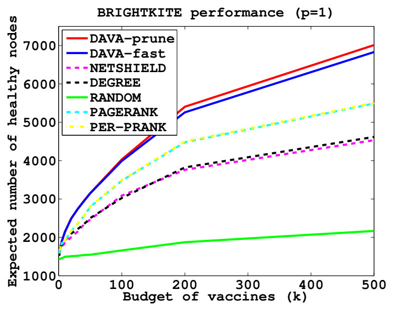
(a) OREGON



(b) GNUTELLA



(c) STANFORD



(d) BRIGHTKITE

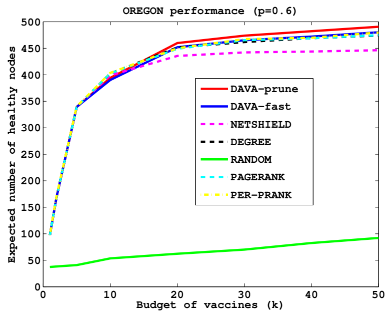
Fig. 6. Effectiveness for DAV (Comparison with baselines): IC model with $p = 1$ on various Real Datasets. Expected number of healthy nodes after distributing vaccines according to different algorithms (i.e. $\sigma'_{G,I'}(S)$) VS budget k. Higher is better. Note that DAVA-prune and DAVA-fast consistently outperform the other algorithms by up to 10 times in magnitude, and DAVA-prune saves more nodes than DAVA-fast. Best seen in color.

8.2.2. **Comparison with baselines**. Since DAVA and DAVA-prune output the same result as we discussed before, in this section, we only show the performance of DAVA-prune for ease of exposition.

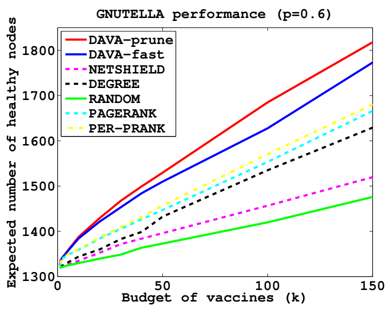
Summary of results. DAVA-prune and DAVA-fast consistently outperform other baseline algorithms for all networks with different settings. Comparing DAVA-prune and DAVA-fast, we found that, as expected DAVA-prune has better performance: it saves upto **16k** (roughly 3% of the graph) more nodes than DAVA-fast in large networks. Hence, if the performance is the top priority, DAVA-prune is the best choice. If the running time is equally crucial, DAVA-fast with a much faster running time (which does not depend on k) would be a better choice. Having said that, the running time for DAVA-prune is also competitive (see Section 8.2.3): it itself is much faster than DAVA.

IC model. Figures 6, 7 and 8 demonstrate our experimental results when $p = 1$, $p = 0.6$ and $p = \{0.1, 0.5, 0.9\}$. In all networks, DAVA-prune and DAVA-fast consistently outperform baseline algorithms, and DAVA-prune gives the best results for all networks.

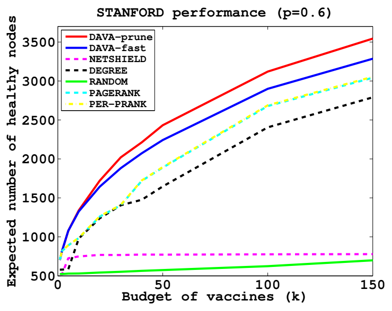
As OREGON had only ~ 600 nodes, we varied k till 50 (roughly 9% of the graph). OREGON has a jelly-fish-type structure, hence for lower k, most algorithms work well by targeting the nodes in the core. But for larger k, the periphery needs to be targeted, and here our algorithms provide the best solution. For bigger networks like GNUTELLA and STANFORD with tens of thousands nodes, the difference in performance of DAVA-prune and DAVA-fast from the other algorithms is clearer. Also, the gains from our algorithms reduced as the



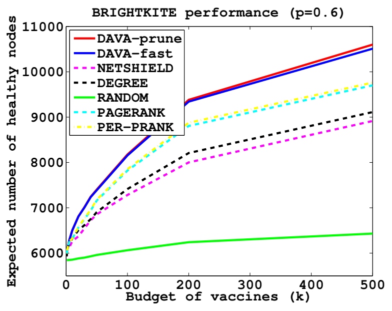
(a) OREGON



(b) GNUTELLA



(c) STANFORD



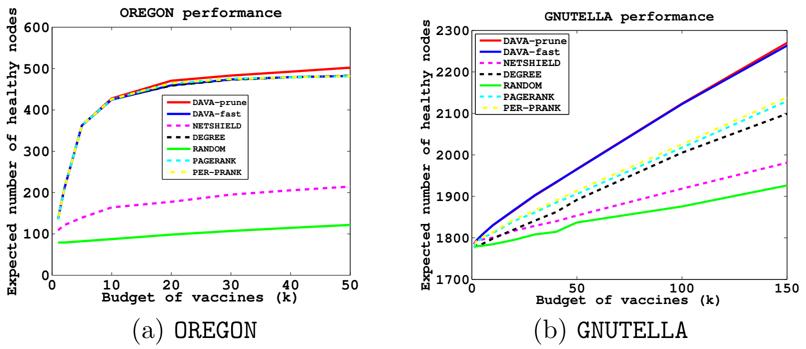
(d) BRIGHTKITE

Fig. 7. Effectiveness for DAV (Comparison with baselines): IC model with $p = 0.6$ on various Real Datasets. Expected number of healthy nodes after distributing vaccines according to different algorithms (i.e. $\sigma'_{G,I'}(S)$) VS budget k. Higher is better. Note that DAVA-prune and DAVA-fast consistently outperform the other algorithms by up to 10 times in magnitude, and DAVA-prune saves more nodes than DAVA-fast. Best seen in color.

p value decreased. This is expected as weaker the disease (in spreading), lower is benefit of vaccinating i.e. lower is the savings gain from carefully selecting important nodes (as removing a node will not further change the expected infections much). PER-PRANK and PAGERANK perform well in STANFORD (a web-graph, with many hubs which these baselines exploit). Additionally NETSHIELD performed well only in BRIGHTKITE, demonstrating that the type of solutions change drastically, once we take the data of who is infected into account. In all these cases, DAVA-prune and DAVA-fast performed the best: DAVA-prune and DAVA-fast both saved about upto 10 times more nodes than baseline algorithms on STANFORD.

SIR model. We got similar results under the SIR model on PORTLAND and MIAMI too (Figure 9): DAVA-prune and DAVA-fast outperform other baselines. We notice that the larger k becomes, the better DAVA-prune and DAVA-fast perform than other algorithms: they both save more than $75k$ (roughly 15% of the graph) nodes than DEGREE when $k = 2000$ (which is similar to the well-known acquaintance immunization method used in practice [Cohen et al. 2003])!

Finally, under both the IC and SIR experiments, though our faster heuristic DAVA-fast performed very well, getting competitive solutions, it is not as good as DAVA-prune: DAVA-prune gives the best results for all networks, e.g., it saves about 400 more nodes than DAVA-fast on STANFORD when $k = 150$. The difference is clearer on PORTLAND and MIAMI: DAVA-prune saves **16k** (roughly 3% of the graph) more nodes than DAVA-fast when $k = 2000$. Hence, if we care mainly about the performance, DAVA-prune is the best choice, and the



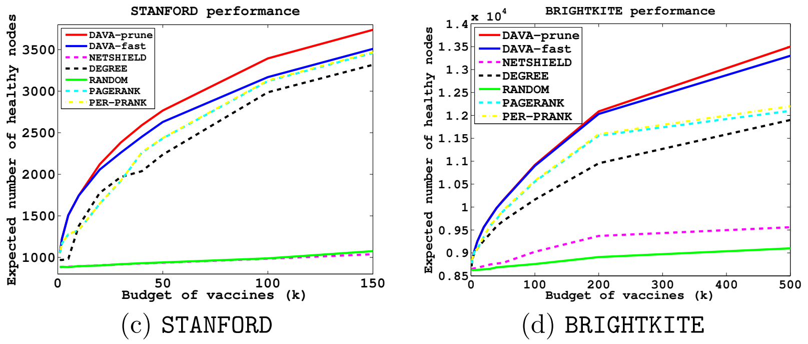


Fig. 8. Effectiveness for DAV (Comparison with baselines): IC model with $p = \{0.1, 0.5, 0.9\}$ on various Real Datasets. Expected number of healthy nodes after distributing vaccines according to different algorithms (i.e. $\sigma'_{G,I'}(S)$) VS budget k. Higher is better. Note that DAVA-prune and DAVA-fast consistently outperform the other algorithms, and DAVA-prune saves more nodes than DAVA-fast. Best seen in color.

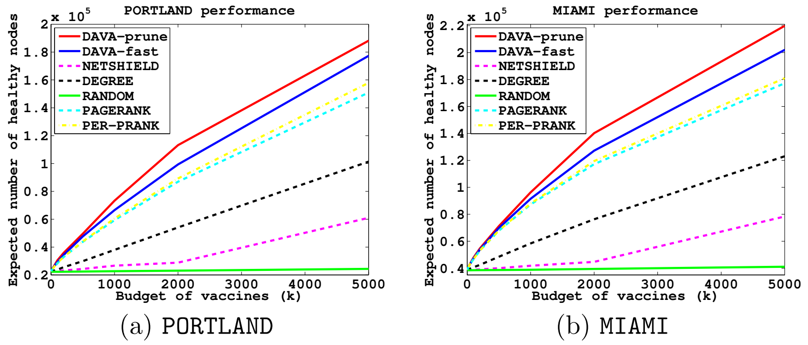
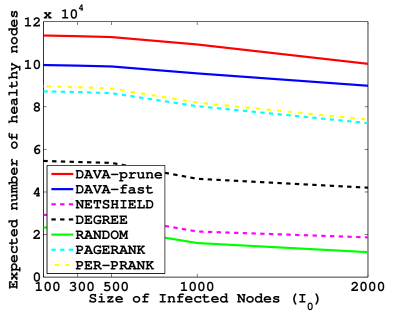


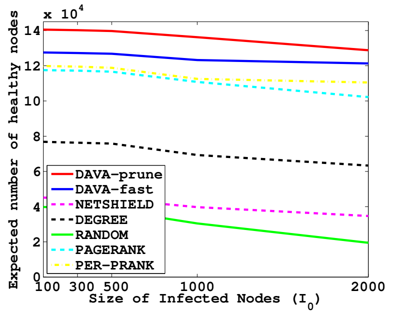
Fig. 9. Effectiveness for DAV (Comparison with baselines): SIR model on Real Datasets. Expected number of healthy nodes after distributing vaccines according to different algorithms (i.e. $\sigma'_{G,I'}(S)$) VS budget k. Higher is better. Note that DAVA-prune and DAVA-fast consistently outperform the other algorithms, and DAVA-prune saves upto $16k$ more nodes than our second best algorithm DAVA-fast. Best seen in color.

running time of DAVA-prune is also competitive (see Section 8.2.3). Though DAVA-fast is not as good as DAVA-prune, it has its advantage: it is a much faster algorithm.

8.2.3. **Quality w.r.t. size of** I_0. We also show the performance of our algorithms w.r.t. the size of I_0. We demonstrate the results of PORTLAND and MIAMI in Figure 10. For other networks with IC model, we got similar results. First, we have an obvious observation that as the size of I_0 increases, the expected number of healthy node at the end decreases. Second, on both `PORTLAND` and `MIAMI`, `DAVA`-prune and `DAVA`-fast consistently outperform other baselines as the size of I_0 changes. Hence, our algorithms are robust w.r.t. the size of I_0, i.e., the gains of our algorithms are consistently higher than other baselines.



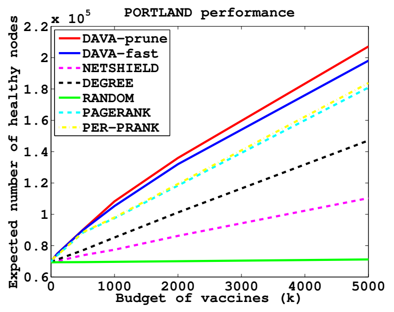
(a) `PORTLAND`



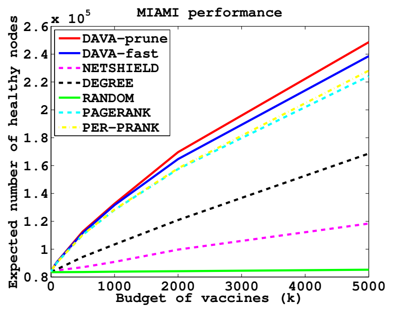
(b) `MIAMI`

Fig. 10. Effectiveness for DAV (Comparison with baselines over the size of I_0): SIR model on Real Datasets. Expected number of healthy nodes after distributing vaccines according to different algorithms (i.e. $\sigma'_{G,I'}(S)$) VS Size of I_0. Higher is better. Note that `DAVA`-prune and `DAVA`-fast consistently outperform the other algorithms. Best seen in color.

8.2.4. **Quality w.r.t. distribution of** I_0. We now show the performance of our algorithms w.r.t. a specific distribution of I_0 as well. Figure 11 shows the results when I_0 is uniformly at random chosen from people with the age over 60 (commonly treated as the vulnerable population) on `PORTLAND` and `MIAMI`. As shown in Figure 11, `DAVA`-prune and `DAVA`-fast consistently outperform other baselines as well. Notice that plots in Figure 11 is almost the same as the result in Figure 9 when I_0 is chosen from the whole population. Therefore, our algorithms are robust w.r.t. the distribution of I_0 too, i.e., the gains of our algorithms are consistent with different distributions of I_0.



(a) `PORTLAND`



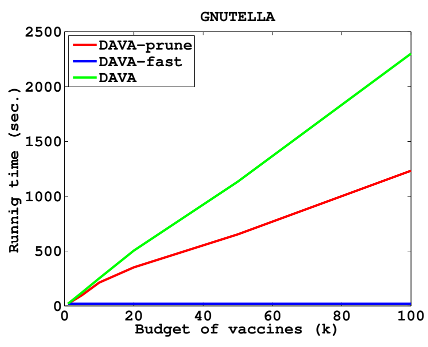
(b) `MIAMI`

Fig. 11. Effectiveness for DAV w.r.t. distribution of I_0: SIR model on Real Datasets. I_0 is uniformly at random chosen from the population with the age 60 or above. Expected number of healthy nodes after distributing vaccines according to different algorithms (i.e. $\sigma'_{G,I'}(S)$) VS Size of I_0. Higher is better. Note that `DAVA`-prune and `DAVA`-fast consistently outperform the other algorithms. Best seen in color.

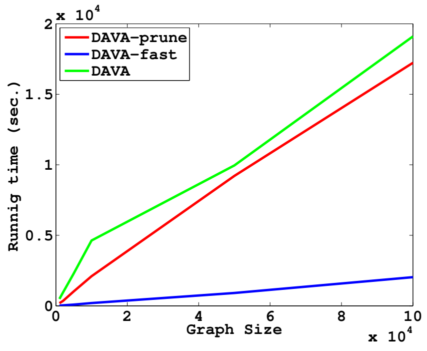
Table III. Running time (sec.) of DAVA, DAVA-prune DAVA-fast and NETSHIELD when $k = 200$. Runs terminated when running time $t > 24$ hours (shown by '-'). We did not show the running time of RANDOM, DEGREE, and PAGERANK because they are fast heuristics.

	DAVA-fast	DAVA-prune	DAVA	NETSHIELD
OREGON	**0.89**	17.5	23.3	4.9
STANFORD	**14.2**	1365.2	2920.4	74.1
GNUTELLA	**20.1**	2245.4	4700.5	79.4
BRIGHTKITE	**109.3**	8921.3	19444.1	246.8
PORTLAND	**778.4**	66424.4	-	8211.6
MIAMI	**1034.2**	81638.5	-	11233.9

8.2.5. **Scalability**. Although our algorithms are polynomial-time, we show some running time results to demonstrate scalability. Table III shows the running time for DAVA, DAVA-prune, DAVA-fast and NETSHIELD for different datasets when the budget $k = 200$. We did not show the running time of RANDOM, DEGREE, PAGERANK and PER-PRANK because they are fast heuristics that finish in the order of seconds, e.g., all four heuristics finish within 60 seconds on the largest network, MIAMI. DAVA-fast is much faster than both DAVA and DAVA-prune—it took only 20 seconds to select 200 nodes in GNUTELLA while DAVA takes more than an hour to finish it. For the large-scale datasets like PORTLAND and MIAMI, DAVA-fast took less than 15 minutes to select 200 nodes, while DAVA could not finish in the allotted time. Further, DAVA-fast is up to 10 times faster than NETSHIELD—this is because NETSHIELD has a $O(nk^2)$ complexity (while our algorithms are linear in the budget). DAVA-prune is faster than DAVA as well—it only took about half the time DAVA takes. Further, for PORTLAND and MIAMI, DAVA-prune can finish in the allotted time. Despite being slower than DAVA-fast, for small graphs, the performance improvement of DAVA-prune is worth its increased running time: e.g., on STANFORD, it saves 350 more nodes than DAVA-fast (i.e., more than 10% as many nodes as DAVA-fast saves, and 4% of nodes w.r.t. the graph size), while having a competitive running time (within 25 minutes) especially when we consider applications where performance is the top priority. On the other hand, for large graphs such as PORTLAND and MIAMI, DAVA-fast has its advantage: it is about 80 times faster than DAVA-prune while the difference in nodes saved is marginal (about 250 nodes for $k = 200$). However, for a larger budget k, as we see above in Section 8.2.2, the difference in performance between DAVA-prune and DAVA-fast may increase a bit. In that case, DAVA-prune might be a good option. Nevertheless, the choice of DAVA-prune and DAVA-fast really depends on the application: if we do not need a fast algorithm, DAVA-prune is clearly a better choice; while for some time-sensitive applications, DAVA-fast will be more suitable, e.g. in case we need to quickly update our allocations.



(a) GNUTELLA



(b) BTER

Fig. 12. Running time (sec.). (a) Running time VS budget k; (b) Running time VS graph size ($k = 200$).

We also shows the running times of our algorithms w.r.t. k (see Figure 12(a)). We demonstrate the result of GNUTELLA, but for other networks we got similar results. We observe that as k increases, the running time increases linearly for both DAVA-prune and DAVA which are consistent with their theoretical complexities, while the running time of DAVA-fast is constant as its running time does not depend on k (see Lemma 6.12). Finally, we also experimented with different graphs to show the running times of our algorithms w.r.t. the size of the graph (see Figure 12(a)) when $k = 200$. We use the BTER Model [Seshadhri et al. 2012] to generate graphs of various sizes. BTER Model is a well-known graph model which accurately captures the properties of real-world networks. Here the graphs we generated preserve the community structure and degree distribution of GNUTELLA. We observe that as the size of graph increases, the running time increases almost linearly for DAVA, DAVA-prune and DAVA-fast, again as expected from their running time complexities.

9. DISCUSSION AND FUTURE WORK

Infected Node Set I_0. As mentioned in Section 4.1, I_0 does not refer to the starting point of the epidemic. Instead, it denotes the set of infected nodes observed at any time of the epidemic when we want to intervene. Nevertheless, in this paper we call I_0 "initially" infected node for ease of our description. In addition, I_0 can be drawn from any distribution: so our problem is a general problem in a sense that we do not assume any distribution. However, it is an interesting question if we can get more accurate or efficient solution if we assume specific distributions such as favoring elderly people and children. In our opinion, it is possible that assuming a specific distribution can lead to a better algorithm. For example, considering the trivial case that if we know there is only one infected node, then we just need to vaccinate its neighbors if it is within the budget.

Nevertheless, in this paper, we have empirically investigated the effect of the distribution and size of I_0 on our algorithms in our experiments and found that our algorithms constantly outperform others baselines no matter how I_0 changes.

Edge weight $p_{u,v}$. In this paper, we assume that the edge weights $p_{u,v}$ (the infectivity of the contagion) are known. For the IC model, there has been some work on learning propagation probabilities from historical data [Saito et al. 2008; Goyal et al. 2010; Lin et al. 2013]. For the SIR model, we use the normalized contact time as the weight of each edge. The higher contact time is, higher the probability that a person will get infected from his/her neighbor. There are also general guidelines for estimating the empirical infectivity of certain diseases such as [Nelson 2005]. It is definitely an interesting problem to relax the assumption that $p_{u,v}$ is known.

Generalizations. Future work can also consist of applying our algorithms to directed graphs (DAVA and DAVA-fast are easily generalizable, while DAVA-prune will require more work) and generalizing our methods to more complicated versions of the Data-Aware Vaccination problem. These can include considering other epidemiological spreading models (like the so-called SIS and SIRS models with temporary immunity and the SEIR model with an incubation period), investigating the effects of uncertain data (e.g. dealing with noisy infection data and uncertain propagation probabilities on edges), and designing accurate real-time immunization strategies.

10. CONCLUSION

This paper addresses the problem of immunizing healthy nodes in presence of already infected nodes given a graph like a social/computer network or the blogsphere. The potential applications are broad: from distributing vaccines to control the epidemic, to stopping already present rumors in social media.

Our main contributions are: we formulated the problem called *Data-Aware Vaccination*, proved it is NP-hard and also hard to approximate within an absolute error. After that we gave an optimal algorithm `DAVA`-tree for m-trees, and presented three polynomial-time heuristics `DAVA`, `DAVA`-prune and `DAVA`-fast for general graphs of varying degrees of performance. We demonstrated the effectiveness and efficiency of our algorithms through extensive experiments on multiple datasets, including large epidemiological social contact networks, computer networks and social media networks, on both the well-known IC and SIR models. Our algorithms outperform other competitors by up to *10 times* in magnitude. Among our methods, `DAVA`-prune saves up to $16k$ more nodes than `DAVA`-fast, but `DAVA`-fast is the fastest. In addition we also presented the scalability of `DAVA`-prune and `DAVA`-fast on large-scale networks.

Acknowledgment

The authors would like to thank the editors and anonymous reviewers for their comments, and the Virginia Bioinformatics Institute for sharing the PORTLAND and MIAMI datasets with us. This material is based upon work partially supported by the NSF under Grant No. IIS-1353346, by the Maryland Procurement Office under contract H98230-14-C-0127, and by a Facebook faculty gift. Any opinions, findings and conclusions or recommendations express in this material are those of the author(s) and do not necessarily reflect the views of the respective funding agencies.

References

Roy M. Anderson and Robert M. May. 1991. *Infectious Diseases of Humans.* Oxford University Press.

James Aspnes, Kevin Chang, and Aleksandr Yampolskiy. 2005. Inoculation strategies for victims of viruses and the sum-of-squares partition problem. *In SODA* (2005), 43–52.

Norman Bailey. 1975. *The Mathematical Theory of Infectious Diseases and its Applications.* Griffin, London.

Christopher L. Barrett, Harry B. Hunt III, Madhav V. Marathe, S. S. Ravi, Daniel J. Rosenkrantz, Richard Edwin Stearns, and Mayur Thakur. 2008. Errata for the paper "Predecessor existence problems for finite discrete dynamical systems" [TCS 386 (1-2) (2007) 3-37]. *Theor. Comput. Sci.* 395, 1 (2008), 132–133.

Sushil Bikhchandani, David Hirshleifer, and Ivo Welch. 1992. A Theory of Fads, Fashion, Custom, and Cultural Change in Informational Cascades. *Journal of Political Economy* 100, 5 (October 1992), 992–1026.

Linda Briesemeister, Patric Lincoln, and Philip Porras. 2003. Epidemic Profiles and Defense of Scale-Free Networks. *WORM 2003* (Oct. 27 2003).

Adam L. Buchsbaum, Haim Kaplan, Anne Rogers, and Jeffery R. Westbrook. 1998. A new, simpler linear-time dominators algorithm. *ACM Trans. Program. Lang. Syst.* 20, 6 (Nov. 1998), 1265–1296.

Po-An Chen, Mary David, and David Kempe. 2010a. Better vaccination strategies for better people. In *Proceedings of the 11th ACM conference on Electronic commerce (EC '10).* ACM, New York, NY, USA, 179–188.

W. Chen, C. Wang, and Y. Wang. 2010b. Scalable influence maximization for prevalent viral marketing in large-scale social networks. *KDD* (2010).

Reuven Cohen, Shlomo Havlin, and Daniel ben Avraham. 2003. Efficient Immunization Strategies for Computer Networks and Populations. *Physical Review Letters* 91, 24 (Dec. 2003).

Ron Cytron, Jeanne Ferrante, Barry K Rosen, Mark N Wegman, and F Kenneth Zadeck. 1989. An efficient method of computing static single assignment form. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages.* ACM, 25–35.

J Dushoff, JB Plotkin, C Viboud, L Simonsen, M Miller, M Loeb, and DJ Earn. 2007. Vaccinating to protect a vulnerable subpopulation. *PLoS Med* 4, 5 (2007), e174.

Stephen Eubank, Hasan Guclu, V. S. Anil Kumar, Madhav V. Marathe, Aravind Srinivasan, Zoltan Toroczkai, and Nan Wang. 2004. Modelling disease outbreaks in realistic urban social networks. *Nature* 429, 6988 (May 2004), 180–184.

NM Ferguson, DA Cummings, C Fraser, JC Cajka, PC Cooley, and DS Burke. 2006. Strategies for mitigating an influenza pandemic. *Nature* 442, 7101 (2006), 448–452.

L. C. Freeman. 1977. A set of measures of centrality based on betweenness. *Sociometry* (1977), 35–41.

Ayalvadi Ganesh, Laurent Massoulié, and Don Towsley. 2005. The effect of network topology on the spread of epidemics. In *IEEE INFOCOM.* IEEE Computer Society Press, Los Alamitos, CA.

Samuel Goldberg. 1986. *Probability: an introduction.* Courier Dover Publications.

Jacob Goldenberg, Barak Libai, and Eitan Muller. 2001. Talk of the Network: A Complex Systems Look at the Underlying Process of Word-of-Mouth. *Marketing Letters* (2001).

Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. 2010. Learning influence probabilities in social networks. In *Proceedings of the third ACM international conference on Web search and data mining*. ACM, 241–250.

D. Gruhl, R. Guha, D. Liben-Nowell, and A. Tomkins. 2004. Information Diffusion Through Blogspace. In *WWW '04*. www.www2004.org/proceedings/docs/1p491.pdf

M. Elizabeth Halloran, Neil M. Ferguson, Stephen Eubank, Ira M. Longini, Derek A. T. Cummings, Bryan Lewis, Shufu Xu, Christophe Fraser, Anil Vullikanti, Timothy C. Germann, Diane Wagener, Richard Beckman, Kai Kadau, Chris Barrett, Catherine A. Macken, Donald S. Burke, and Philip Cooley. 2008. Strategies for mitigating an influenza pandemic. *Proceedings of the National Academy of Sciences* 105, 12 (2008), 4639–4644.

Yukio Hayashi, Masato Minoura, and Jun Matsukubo. 2003. Recoverable prevalence in growing scale-free networks and the effective immunization. *arXiv:cond-mat/0305549 v2* (Aug. 6 2003).

H. W. Hethcote. 2000. The mathematics of infectious diseases. *SIAM Rev.* 42 (2000).

Glen Jeh and Jennifer Widom. 2003. Scaling Personalized Web Search. In *Proceedings of the 12th International Conference on World Wide Web (WWW '03)*. ACM, New York, NY, USA, 271–279. DOI:http://dx.doi.org/10.1145/775152.775191

David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *Conference of the ACM Special Interest Group on Knowledge Discovery and Data Mining*. ACM Press, New York, NY.

J. O. Kephart and S. R. White. 1993. Measuring and modeling computer virus prevalence. *IEEE Computer Society Symposium on Research in Security and Privacy* (1993).

Masahiro Kimura, Kazumi Saito, and Hiroshi Motoda. 2008. Minimizing the spread of contamination by blocking links in a network. In *Proceedings of the 23rd national conference on Artificial intelligence (AAAI'08)*. AAAI Press, 1175–1180.

Jon M. Kleinberg. 1998. Authoritative Sources in a Hyperlinked Environment. In *ACM-SIAM Symposium on Discrete Algorithms.*

Chris J. Kuhlman, Gaurav Tuli, Samarth Swarup, Madhav V. Marathe, and S. S. Ravi. 2013. Blocking Simple and Complex Contagion by Edge Removal. In *ICDM*. 399–408.

Ravi Kumar, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. 2003. On the bursty evolution of blogspace. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*. ACM Press, New York, NY, USA, 568–576. DOI:http://dx.doi.org/10.1145/775152.775233

T. Lappas, E. Terzi, D. Gunopoulos, and H. Mannila. 2010. Finding Effectors in Social Networks. *SIGKDD* (2010).

Thomas Lengauer and Robert Endre Tarjan. 1979. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.* 1, 1 (Jan. 1979), 121–141.

Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. 2006. The dynamics of viral marketing. In *EC '06: Proceedings of the 7th ACM conference on Electronic commerce*. ACM Press, New York, NY, USA, 228–237. DOI:http://dx.doi.org/10.1145/1134707.1134732

Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie S. Glance. 2007a. Cost-effective outbreak detection in networks. In *KDD*. 420–429.

Jure Leskovec, Mary McGlohon, Christos Faloutsos, Natalie Glance, and Matthew Hurst. 2007b. Cascading Behavior in Large Blog Graphs: Patterns and a Model. In *Society of Applied and Industrial Mathematics: Data Mining.*

Shuyang Lin, Fengjiao Wang, Qingbo Hu, and Philip S Yu. 2013. Extracting social events for learning better information diffusion models. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 365–373.

Nilly Madar, Tomer Kalisky, Reuven Cohen, Daniel ben Avraham, and Shlomo Havlin. 2004. Immunization and Epidemic Dynamics in Complex Networks. *Eur. Phys. J. B* 38, 2 (2004), 269–276.

Yasuko Matsubara, Yasushi Sakurai, B. Aditya Prakash, Lei Li, and Christos Faloutsos. 2012. Rise and fall patterns of information diffusion: model and implications. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '12)*. 6–14.

A G McKendrick. 1925. Applications of Mathematics to Medical Problems. In *Proceedings of Edin. Math. Society*, Vol. 44. 98–130.

J. Medlock and A. P. Galvani. 2009. Optimizing Influenza Vaccine Distribution. *Science* 325 (2009). Issue 5948.

Lauren Ancel Meyers, M.E.J. Newman, and Babak Pourbohloul. 2006. Predicting epidemics on directed contact networks. *Journal of Theoretical Biology* 240, 3 (2006), 400 – 418. DOI:http://dx.doi.org/DOI:10.1016/j.jtbi.2005.10.004

James Moody and Douglas R. White. 2003. Social Cohesion and Embeddedness: A Hierarchical Conception of Social Groups. *American Sociological Review* (2003), 1–25.

Fred Morstatter, Jürgen Pfeffer, Huan Liu, and Kathleen M Carley. 2013. Is the Sample Good Enough? Comparing Data from Twitter's Streaming API with Twitter's Firehose.. In *ICWSM*.

NDSSL. 2007. Synthetic Data Products for Societal Infrastructures and Protopopulations: Data Set 2.0. *NDSSL-TR-07-003* (2007). http://ndssl.vbi.vt.edu/Publications/ndssl-tr-07-003.pdf

Kenrad E Nelson. 2005. Epidemiology of infectious disease: general principles. *Infectious Disease Epidemiology Theory and Practice. Gaithersburg, MD: Aspen Publishers* (2005), 17–48.

M.E.J. Newman. 2005. A Measure of Betweenness Centrality Based on Random Walks. *Social Networks* 27 (2005), 39–54.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1998. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report. Stanford Digital Library Technologies Project. http://dbpubs.stanford.edu/pub/1999-66 Paper SIDL-WP-1999-0120 (version of 11/11/1999).

Romualdo Pastor-Satorras and Alessandro Vespignani. 2002. Epidemic dynamics in finite size scale-free networks. *Physical Review E* 65 (2002), 035108.

B. Aditya Prakash, Deepayan Chakrabarti, Michalis Faloutsos, Nicholas Valler, and Christos Faloutsos. 2011. Threshold Conditions for Arbitrary Cascade Models on Arbitrary Networks. In *ICDM*.

B. Aditya Prakash, Jilles Vreeken, and Christos Faloutsos. 2012. Spotting Culprits in Epidemics: How many and Which ones?. In *ICDM*.

M. Richardson and P. Domingos. 2002. Mining Knowledge-Sharing Sites for Viral Marketing. In *SIGKDD Conference*.

Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. 2008. Prediction of information diffusion probabilities for independent cascade model. In *Knowledge-Based Intelligent Information and Engineering Systems*. Springer, 67–75.

C Seshadhri, Tamara G Kolda, and Ali Pinar. 2012. Community structure and scale-free collections of Erdős-Rényi graphs. *Physical Review E* 85, 5 (2012), 056109.

Devavrat Shah and Tauhid Zaman. 2011. Rumors in a Network: Who's the Culprit? *IEEE Transactions on Information Theory* 57, 8 (2011), 5163–5181.

Min-Zheng Shieh, Shi-Chun Tsai, and Ming-Chuan Yang. 2012. On the inapproximability of maximum intersection problems. *Inf. Process. Lett.* 112, 19 (Oct. 2012), 723–727.

Hanghang Tong, B. Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. 2012. Gelling, and melting, large graphs by edge manipulation. In *Proceedings of the 21st ACM international conference on Information and knowledge management (CIKM '12)*. New York, NY, USA, 245–254.

Hanghang Tong, B. Aditya Prakash, Charalampos E. Tsourakakis, Tina Eliassi-Rad, Christos Faloutsos, and Duen Horng Chau. 2010. On the Vulnerability of Large Graphs. In *ICDM*.

Staal A. Vinterbo. 2004. Privacy: A Machine Learning View. *IEEE Trans. on Knowl. and Data Eng.* 16, 8 (Aug. 2004), 939–948.

Yang Wang, Deepayan Chakrabarti, Chenxi Wang, and Christos Faloutsos. 2003. Epidemic spreading in real networks: An eigenvalue viewpoint. In *Symposium on Reliable Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, 25–34.

Eduardo C. Xavier. 2012. A note on a Maximum k-Subset Intersection problem. *Inf. Process. Lett.* 112, 12 (June 2012), 471–472.

Reza Yaesoubi and Ted Cohen. 2011. Dynamic health policies for controlling the spread of emerging infections: Influenza as an example. *PLoS ONE* 6, 9 (2011), e24043.

Yao Zhang and B. Aditya Prakash. 2014. DAVA: Distributing Vaccines over Large Networks under Prior Information. In *Proceedings of the SIAM International Conference on Data Mining (SDM'14)*.

Appendix—Another prune algorithm

The appendix provides a method for maintaining an updated dominator tree in face of node deletions without rebuilding from scratch each time, which might be useful for other problems that utilize dominator trees. For example, dominator trees are widely used to analyze the dependency of variables/expressions after building a control flow graph in Software Engineering [Cytron et al. 1989]. However, when conducting dynamic analysis for a program, it might be necessary to maintain the dominator tree in the phase of variable deletion. The technique in the appendix may help such application to speed up the analysis since rebuilding the dominator tree will not be required.

Here we show another prune technique for `DAVA`. The idea is that, since nodes that are not in the first layer of the dominator tree will not change their parent nodes (Lemma 6.7), we may not need to re-build the dominator tree multiple times. Hence we need to find out the new immediate dominators of nodes in the previous dominator tree, and update the previous dominator tree without re-building it. However, this prune algorithm may run slow in practice because updating the dominator tree in this algorithm may not be completed in linear time. Hence, we do not use this pruning technique in our algorithm `DAVA`-prune, and present it here for independent interest.

First, according to Lemma 6.7 and 6.8, we do not need to consider nodes that are not in the first layer of the old dominator tree T_{old}. Now let us only consider nodes in the first layer of T_{old}. Suppose v is the node we select after running `DAVA-TREE`. For a node u in the first layer of T_{old}, according to the definition of the dominator [Lengauer and Tarjan 1979], the new set of dominators of u is $\bigcap_{i\in N_{G'}(u)\bigcap i\notin N_{G'}(v)} DOM(i)$ where $N_{G'}(u)$ is the set of neighbors of u in the merged graph G' and $DOM(i)$ is the set of dominators of i in G' (the intersection of u's neighbors dominators). Hence, we can build the dominator tree of G' only once, and after that update the new dominators of nodes in the first layer appropriately. Algorithm 6 shows the prune process, and Algorithm 7 shows the whole new prune algorithm. Lines 3-9 in Algorithm 6 show how to update the new immediate dominators of nodes in the first layer. For the reweighting process, we use the same technique as Algorithm 4.

Algorithm 6 `Prune` Algorithm

```
Require: $G$,$P$, $I_0$, $T$, $v$
 1: {//Step 1 ReLocating: Change locations of nodes in the first layer}
 2: NodeCandidates=Neighbors$_T(I_0)$
 3: for node $u$ in NodeCandidates do
 4:     DomCandidates=$\bigcap_{i\in Neighbors_G(u)\bigcap i\notin Neighbors_G(v)} DOM(i)$
 5:     if DomCandidates$\neq\emptyset$ then
 6:         $parent_u$ =a node in DomCandidates with the lowest layer
 7:         Move $u$ to its new parent $parent_u$ in $T$
 8:     end if
 9: end for
10: {//Step 2 ReWeighting: Change weights of nodes in the first layer}
11: $G''$ =Removing nodes that are not in the first layer of $T$ in $G'$
12: Run Dijkstra's algorithm on $G''$ to get new weights
13: Remove node $v$ from $G'$ and $T$
14: return $T$
```

Running Time: Suppose D means the maximum degree in the graph G, H is the height of the dominator tree, and recall that F is the number of nodes in the first layer of dominator tree.

LEMMA 10.1. *(Running time of* `DAVA`*-prune-2) Algorithm 7 takes* $O(|V|\log|V|+|E|+k(FHD+F\log F))$ *worst-case time.*

PROOF. Building a dominator tree costs $O(|V|+|E|)$ time [Buchsbaum et al. 1998], and weighting the dominator tree for the first take costs $O(|E|+|V|\log|V|)$. After that, pruning dominator tree costs $O(FHD+F\log F)$ time ($O(FHD)$ is the worst time for relocating nodes in the first layer of the dominator tree while $O(F\log F)$ is the running time for

Algorithm 7 DAVA-prune-2 Algorithm

```
Require: Graph $G$, $P$, budget $k$, infected set $I_0$
1: $S = \emptyset$
2: $G' =$ Run MERGE on $G$ and $I_0$
3: $T =$ Build the dominator tree from $G'$ and assign probabilities $\tilde{p}_{v,u}$
4: repeat
5:     $v =$ Run DAVA-TREE on $T$ with budget $= 1$
6:     $S = S \cup v$
7:     Run Prune on $G'$, $T$, $P$ and $v$
8: until $|S| = k$
9: return $S$
```

reweighting nodes in the first layer). Hence in general, the worst-case time complexity for DAVA-pruning is $O(|V| \log |V| + |E| + k(FD + F \log F))$. $\square$

Unlike DAVA-prune, Algorithm 7 does not need to rebuild the dominator tree multiple times. However, the running time to change locations of nodes in the first layer in Algorithm 7 can be quadratic for the worst case. But in DAVA-prune, rebuilding the dominator tree can be done in the linear time. Hence for the worst case, DAVA-prune is faster than Algorithm 7.