Cost-effective Outbreak Detection in Networks

Jure Leskovec
Carnegie Mellon University

Andreas Krause
Carnegie Mellon University

Carlos Guestrin
Carnegie Mellon University

Christos Faloutsos
Carnegie Mellon University

Jeanne VanBriesen
Carnegie Mellon University

Natalie Glance
Nielsen BuzzMetrics

ABSTRACT

Given a water distribution network, where should we place sensors to quickly detect contaminants? Or, which blogs should we read to avoid missing important stories?

These seemingly different problems share common structure: Outbreak detection can be modeled as selecting nodes (sensor locations, blogs) in a network, in order to detect the spreading of a virus or information as quickly as possible.

We present a general methodology for near optimal sensor placement in these and related problems. We demonstrate that many realistic outbreak detection objectives (*e.g.*, detection likelihood, population affected) exhibit the property of "submodularity". We exploit submodularity to develop an efficient algorithm that scales to large problems, achieving near optimal placements, while being 700 times faster than a simple greedy algorithm. We also derive online bounds on the quality of the placements obtained by *any* algorithm. Our algorithms and bounds also handle cases where nodes (sensor locations, blogs) have different costs.

We evaluate our approach on several large real-world problems, including a model of a water distribution network from the EPA, and real blog data. The obtained sensor placements are provably near optimal, providing a constant fraction of the optimal solution. We show that the approach scales, achieving speedups and savings in storage of several orders of magnitude. We also show how the approach leads to deeper insights in both applications, answering multicriteria trade-off, cost-sensitivity and generalization questions.

Categories and Subject Descriptors: F.2.2 Analysis of Algorithms and Problem Complexity: Nonnumerical Algorithms and Problems

General Terms: Algorithms; Experimentation.

Keywords: Graphs; Information cascades; Virus propagation; Sensor Placement; Submodular functions.

1. INTRODUCTION

We explore the general problem of detecting outbreaks in networks, where we are given a network and a dynamic process spreading over this network, and we want to select a set of nodes to detect the process as effectively as possible.

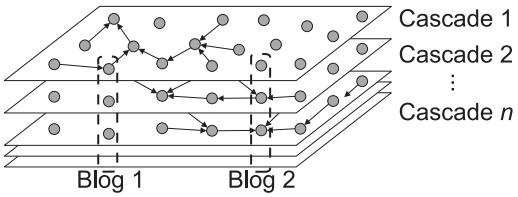


Figure 1: Spread of information between blogs. Each layer shows an information cascade. We want to pick few blogs that quickly capture most cascades.

Many real-world problems can be modeled under this setting. Consider a city water distribution network, delivering water to households via pipes and junctions. Accidental or malicious intrusions can cause contaminants to spread over the network, and we want to select a few locations (pipe junctions) to install sensors, in order to detect these contaminations as quickly as possible. In August 2006, the Battle of Water Sensor Networks (BWSN) [19] was organized as an international challenge to find the best sensor placements for a real (but anonymized) metropolitan area water distribution network. As part of this paper, we present the approach we used in this competition. Typical epidemics scenarios also fit into this outbreak detection setting: We have a social network of interactions between people, and we want to select a small set of people to monitor, so that any disease outbreak can be detected early, when very few people are infected.

In the domain of weblogs (blogs), bloggers publish posts and use hyper-links to refer to other bloggers' posts and content on the web. Each post is time stamped, so we can observe the spread of information on the "blogosphere". In this setting, we want to select a set of blogs to read (or retrieve) which are most up to date, *i.e.*, catch (link to) most of the stories that propagate over the blogosphere. Fig. 1 illustrates this setting. Each layer plots the propagation graph (also called *information cascade* [3]) of the information. Circles correspond to blog posts, and all posts at the same vertical column belong to the same blog. Edges indicate the temporal flow of information: the cascade starts at some post (*e.g.*, top-left circle of the top layer of Fig. 1) and then the information propagates recursively by other posts linking to it. Our goal is to select a small set of blogs (two in case of Fig. 1) which "catch" as many cascades (stories) as possible[1]. A naive, intuitive solution would be to select the

[1]In real-life multiple cascades can be on the same or similar story, but we still aim to detect as many as possible.

KDD'07, August 12–15, 2007, San Jose, California, USA.

big, well-known blogs. However, these usually have a large number of posts, and are time-consuming to read. We show, that, perhaps counterintuitively, a more cost-effective solution can be obtained, by reading smaller, but higher quality, blogs, which our algorithm can find.

There are several possible criteria one may want to optimize in outbreak detection. For example, one criterion seeks to minimize *detection time* (*i.e.*, to know about a cascade as soon as possible, or avoid spreading of contaminated water). Similarly, another criterion seeks to minimize the *population affected* by an undetected outbreak (*i.e.*, the number of blogs referring to the story we just missed, or the population consuming the contamination we cannot detect). Optimizing these objective functions is NP-hard, so for large, real-world problems, we cannot expect to find the optimal solution.

In this paper, we show, that these and many other realistic outbreak detection objectives are *submodular*, *i.e.*, they exhibit a diminishing returns property: Reading a blog (or placing a sensor) when we have only read a few blogs provides more new information than reading it after we have read many blogs (placed many sensors).

We show how we can exploit this submodularity property to *efficiently obtain* solutions which are *provably close* to the optimal solution. These guarantees are important in practice, since selecting nodes is expensive (reading blogs is time-consuming, sensors have high cost), and we desire solutions which are not too far from the optimal solution.

The main contributions of this paper are:

- We show that many objective functions for detecting outbreaks in networks are submodular, including detection time and population affected in the blogosphere and water distribution monitoring problems. We show that our approach also generalizes work by [10] on selecting nodes maximizing influence in a social network.
- We exploit the submodularity of the objective (*e.g.*, detection time) to develop an efficient approximation algorithm, CELF, which achieves near-optimal placements (guaranteeing at least a constant fraction of the optimal solution), providing a novel theoretical result for non-constant node cost functions. CELF is up to 700 times faster than simple greedy algorithm. We also derive novel online bounds on the quality of the placements obtained by *any* algorithm.
- We extensively evaluate our methodology on the applications introduced above – water quality and blogosphere monitoring. These are large real-world problems, involving a model of a water distribution network from the EPA with millions of contamination scenarios, and real blog data with millions of posts.
- We show how the proposed methodology leads to deeper insights in both applications, including multicriterion, cost-sensitivity analyses and generalization questions.

2. OUTBREAK DETECTION

2.1 Problem statement

The water distribution and blogosphere monitoring problems, despite being very different domains, share essential structure. In both problems, we want to select a subset $\mathcal{A}$ of nodes (sensor locations, blogs) in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, which detect outbreaks (spreading of a virus/information) quickly.

Fig. 2 presents an example of such a graph for blog network. Each of the six blogs consists of a set of posts. Connections between posts represent hyper-links, and labels show the time difference between the source and destination post, *e.g.*, post p_{41} linked p_{12} one day after p_{12} was published).

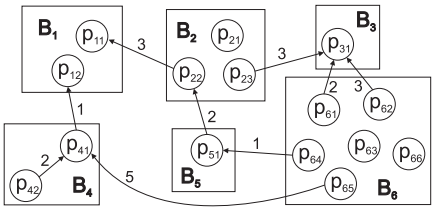


Figure 2: Blogs have posts, and there are time stamped links between the posts. The links point to the sources of information and the cascades grow (information spreads) in the reverse direction of the edges. Reading only blog B_6 captures all cascades, but late. B_6 also has many posts, so by reading B_1 and B_2 we detect cascades sooner.

These outbreaks (*e.g.*, information cascades) initiate from a single node of the network (*e.g.*, p_{11}, p_{12} and p_{31}), and spread over the graph, such that the traversal of every edge $(s,t) \in \mathcal{E}$ takes a certain amount of time (indicated by the edge labels). As soon as the event reaches a selected node, an alarm is triggered, e.g., selecting blog B_6, would detect the cascades originating from post p_{11}, p_{12} and p_{31}, after 6, 6 and 2 timesteps after the start of the respective cascades.

Depending on which nodes we select, we achieve a certain placement score. Fig. 2 illustrates several criteria one may want to optimize. If we only want to detect as many stories as possible, then reading just blog B_6 is best. However, reading B_1 would only miss one cascade (p_{31}), but would detect the other cascades immediately. In general, this placement score (representing, *e.g.*, the fraction of detected cascades, or the population saved by placing a water quality sensor) is a set function R, mapping every placement $\mathcal{A}$ to a real number $R(\mathcal{A})$ (our reward), which we intend to maximize.

Since sensors are expensive, we also associate a *cost* $c(\mathcal{A})$ with every placement $\mathcal{A}$, and require, that this cost does not exceed a specified budget B which we can spend. For example, the cost of selecting a blog could be the number of posts in it (*i.e.*, B_1 has cost 2, while B_6 has cost 6). In the water distribution setting, accessing certain locations in the network might be more difficult (expensive) than other locations. Also, we could have several types of sensors to choose from, which vary in their quality (detection accuracy) and cost. We associate a nonnegative cost $c(s)$ with every sensor s, and define the cost of placement $\mathcal{A}$: $c(\mathcal{A}) = \sum_{s \in \mathcal{A}} c(s)$.

Using this notion of reward and cost, our goal is to solve the optimization problem

$$\max_{\mathcal{A} \subseteq \mathcal{V}} R(\mathcal{A}) \text{ subject to } c(\mathcal{A}) \leq B, \quad (1)$$

where B is a budget we can spend for selecting the nodes.

2.2 Placement objectives

An event $i \in \mathcal{I}$ from set $\mathcal{I}$ of scenarios (*e.g.*, cascades, contaminant introduction) originates from a node $s' \in \mathcal{V}$ of a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and spreads through the network, affecting other nodes (*e.g.*, through citations, or flow through pipes). Eventually, it reaches a monitored node $s \in \mathcal{A} \subseteq \mathcal{V}$ (*i.e.*, blogs we read, pipe junction we instrument with a sensor), and gets detected. Depending on the time of detection $t = T(i, s)$, and the impact on the network before the detection (*e.g.*, the size of the cascades missed, or the population affected by a contaminant), we incur *penalty* $\pi_i(t)$. The

penalty function $\pi_i(t)$ depends on the scenario. We discuss concrete examples of penalty functions below. Our goal is to minimize the expected penalty over all possible scenarios i:

$$\pi(\mathcal{A}) \equiv \sum_i P(i)\pi_i(T(i,\mathcal{A})),$$

where, for a placement $\mathcal{A} \subseteq \mathcal{V}$, $T(i,\mathcal{A}) = \min_{s\in\mathcal{A}} T(i,s)$ is the time until event i is detected by one of the sensors in $\mathcal{A}$, and P is a (given) probability distribution over the events.

We assume $\pi_i(t)$ to be monotonically nondecreasing in t, *i.e.*, we never prefer late detection if we can avoid it. We also set $T(i,\emptyset) = \infty$, and set $\pi_i(\infty)$ to some maximum penalty incurred for not detecting event i.

Proposed alternative formulation. Instead of minimizing the penalty $\pi(\mathcal{A})$, we can consider the scenario specific *penalty reduction* $R_i(\mathcal{A}) = \pi_i(\infty) - \pi_i(T(i,\mathcal{A}))$, and the expected penalty reduction

$$R(\mathcal{A}) = \sum_i P(i)R_i(\mathcal{A}) = \pi(\emptyset) - \pi(\mathcal{A}),$$

describes the expected benefit (reward) we get from placing the sensors. This alternative formulation has crucial properties which our method exploits, as described below.

Examples used in our experiments. Even though the water distribution and blogosphere monitoring problems are very different, similar placement objective scores make sense for both applications. The detection time $T(i,s)$ in the blog setting is the time difference in days, until blog s participates in the cascade i, which we extract from the data. In the water network, $T(i,s)$ is the time it takes for contaminated water to reach node s in scenario i (depending on outbreak location and time). In both applications we consider the following objective functions (penalty reductions):

(a) *Detection likelihood (DL)*: fraction of information cascades and contamination events detected by the selected nodes. Here, the penalty is $\pi_i(t) = 0$, and $\pi_i(\infty) = 1$, *i.e.*, we do not incur any penalty if we detect the outbreak in finite time, otherwise we incur penalty 1.

(b) *Detection time (DT)* measures the time passed from outbreak till detection by one of the selected nodes. Hence, $\pi_i(t) = \min\{t, T_{\max}\}$, where $T_{\max}$ is the time horizon we consider (end of simulation / data set).

(c) *Population affected (PA)* by scenario (cascade, outbreak). This criterion has different interpretations for both applications. In the blog setting, the affected population measures the number of blogs involved in a cascade before the detection. Here, $\pi_i(t)$ is the size of (number of blogs participating in) cascade i at time t, and $\pi_i(\infty)$ is the size of the cascade at the end of the data set. In the water distribution application, the affected population is the expected number of people affected by not (or late) detecting a contamination.

Note, that optimizing each of the objectives can lead to very different solutions, hence we may want to simultaneously optimize all objectives at once. We deal with this multicriterion optimization problem in Section 2.4.

2.3 Properties of the placement objectives

The penalty reduction function[2] $R(\mathcal{A})$ has several important and intuitive properties: Firstly, $R(\emptyset) = 0$, *i.e.*, we do not reduce the penalty if we do not place any sensors. Secondly, R is nondecreasing, *i.e.*, $R(\mathcal{A}) \leq R(\mathcal{B})$ for all $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$. Hence, adding sensors can only decrease the incurred penalty. Thirdly, and most importantly, it satisfies the following intuitive diminishing returns property: If we add a sensor to a small placement $\mathcal{A}$, we improve our score at least as much, as if we add it to a larger placement $\mathcal{B} \supseteq \mathcal{A}$. More formally, we can prove that

THEOREM 1. *For all placements $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$ and sensors $s \in \mathcal{V} \setminus \mathcal{B}$, it holds that*

$$R(\mathcal{A} \cup \{s\}) - R(\mathcal{A}) \geq R(\mathcal{B} \cup \{s\}) - R(\mathcal{B}).$$

A set function R with this property is called *submodular*. We give the proof of Thm. 1 and all other theorems in [15].

Hence, both the blogosphere and water distribution monitoring problems can be reduced to the problem of maximizing a nondecreasing submodular function, subject to a constraint on the budget we can spend for selecting nodes. More generally, any objective function that can be viewed as an expected penalty reduction is submodular. Submodularity of R will be the key property exploited by our algorithms.

2.4 Multicriterion optimization

In practical applications, such as the blogosphere and water distribution monitoring, we may want to *simultaneously* optimize multiple objectives. Then, each placement has a vector of scores, $R(\mathcal{A}) = (R_1(\mathcal{A}), \ldots, R_m(\mathcal{A}))$. Here, the situation can arise that two placements $\mathcal{A}_1$ and $\mathcal{A}_2$ are *incomparable*, *e.g.*, $R_1(\mathcal{A}_1) > R_1(\mathcal{A}_2)$, but $R_2(\mathcal{A}_1) < R_2(\mathcal{A}_2)$. So all we can hope for are *Pareto-optimal solutions* [4]. A placement $\mathcal{A}$ is called Pareto-optimal, if there does not exist another placement $\mathcal{A}'$ such that $R_i(\mathcal{A}') \geq R_i(\mathcal{A})$ for all i, and $R_j(\mathcal{A}') > R_j(\mathcal{A})$ for some j (*i.e.*, there is no placement $\mathcal{A}'$ which is at least as good as $\mathcal{A}$ in all objectives R_i, and strictly better in at least one objective R_j).

One common approach for finding such Pareto-optimal solutions is *scalarization* (*c.f.,* [4]). Here, one picks positive weights $\lambda_1 > 0, \ldots, \lambda_m > 0$, and optimizes the objective $R(\mathcal{A}) = \sum_i \lambda_i R_i(\mathcal{A})$. Any solution maximizing $R(\mathcal{A})$ is *guaranteed* to be Pareto-optimal [4], and by varying the weights λ_i, different Pareto-optimal solutions can be obtained. One might be concerned that, even if optimizing the individual objectives R_i is easy (*i.e.*, can be approximated well), optimizing the sum $R = \sum_i \lambda_i R_i$ might be hard. However, submodularity is closed under nonnegative linear combinations and thus the new scalarized objective is submodular as well, and we can apply the algorithms we develop in the following section.

3. PROPOSED ALGORITHM

Maximizing submodular functions in general is NP-hard [11]. A commonly used heuristic in the *simpler* case, where every node has *equal* cost (*i.e.*, unit cost, $c(s) = 1$ for all locations s) is the *greedy algorithm*, which starts with the empty placement $\mathcal{A}_0 = \emptyset$, and iteratively, in step k, adds the location s_k which maximizes the *marginal gain*

$$s_k = \operatorname*{argmax}_{s\in\mathcal{V}\setminus\mathcal{A}_{k-1}} R(\mathcal{A}_{k-1} \cup \{s\}) - R(\mathcal{A}_{k-1}). \quad (2)$$

The algorithm stops, once it has selected B elements. Considering the hardness of the problem, we might expect the greedy algorithm to perform arbitrarily badly. However, in the following we show that this is not the case.

3.1 Bounds for the algorithm

Unit cost case. Perhaps surprisingly – in the unit cost case – the simple greedy algorithm is near-optimal:

[2]The objective R is similar to one of the examples of submodular functions described by [17]. Our objective, however, preserves additional problem structure (sparsity) which we exploit in our implementation, and which we crucially depend on to solve large problem instances.

THEOREM 2 ([17]). *If R is a submodular, nondecreasing set function and $R(\emptyset) = 0$, then the greedy algorithm finds a set $\mathcal{A}_G$, such that $R(\mathcal{A}_G) \geq (1-1/e) \max_{|\mathcal{A}|=B} R(\mathcal{A})$.*

Hence, the greedy algorithm is guaranteed to find a solution which achieves at least a constant fraction $(1-1/e)$ ($\approx 63\%$) of the optimal score. The penalty reduction R satisfies all requirements of Theorem 2, and hence the greedy algorithm approximately solves the maximization problem Eq. (1).

Non-constant costs. What if the costs of the nodes are not constant? It is easy to see that the simple greedy algorithm, which iteratively adds sensors using rule from Eq. (2) until the budget is exhausted, can fail badly, since it is indifferent to the costs (*i.e.*, a very expensive sensor providing reward r is preferred over a cheaper sensor providing reward $r - \varepsilon$. To avoid this issue, the greedy rule Eq. (2) can be modified to take costs into account:

$$s_k = \operatorname*{argmax}_{s \in \mathcal{V} \setminus \mathcal{A}_{k-1}} \frac{R(\mathcal{A}_{k-1} \cup \{s\}) - R(\mathcal{A}_{k-1})}{c(s)}, \qquad (3)$$

i.e., the greedy algorithm picks the element maximizing the benefit/cost ratio. The algorithm stops once no element can be added to the current set $\mathcal{A}$ without exceeding the budget. Unfortunately, this intuitive generalization of the greedy algorithm can perform arbitrarily worse than the optimal solution. Consider the case where we have two locations, s_1 and s_2, $c(s_1) = \varepsilon$ and $c(s_2) = B$. Also assume we have only one scenario i, and $R(\{s_1\}) = 2\varepsilon$, and $R(\{s_2\}) = B$. Now, $R((\{s_1\}) - R(\emptyset))/c(s_1) = 2$, and $R((\{s_2\}) - R(\emptyset))/c(s_2) = 1$. Hence the greedy algorithm would pick s_1. After selecting s_1, we cannot afford s_2 anymore, and our total reward would be ε. However, the optimal solution would pick s_2, achieving total penalty reduction of B. As ε goes to 0, the performance of the greedy algorithm becomes arbitrarily bad.

However, the greedy algorithm can be improved to achieve a constant factor approximation. This new algorithm, CEF (Cost-Effective Forward selection), computes the solution $\mathcal{A}_{GCB}$ using the benefit-cost greedy algorithm, using rule (3), and also computes the solution $\mathcal{A}_{GUC}$ using the unit-cost greedy algorithm (ignoring the costs), using rule (2). For both rules, CEF only considers elements which do not exceed the budget B. CEF then returns the solution with higher score. Even though both solutions can be arbitrarily bad, the following result shows that there is at least one of them which is not too far away from optimum, and hence CEF provides a constant factor approximation.

THEOREM 3. *Let R be the a nondecreasing submodular function with $R(\emptyset) = 0$. Then*

$$\max\{R(\mathcal{A}_{GCB}), R(\mathcal{A}_{GUC})\} \geq \frac{1}{2}(1 - 1/e) \max_{\mathcal{A}, c(\mathcal{A}) \leq B} R(\mathcal{A}).$$

Theorem 3 was proved by [11] for the special case of the Budgeted MAX-COVER problem[3], and here we prove this result for *arbitrary* nondecreasing submodular functions. Theorem 3 states that the better solution of $\mathcal{A}_{GBC}$ and $\mathcal{A}_{GUC}$ (which is returned by CEF) is at most a constant factor $\frac{1}{2}(1 - 1/e)$ of the optimal solution.

Note that the running time of CEF is $\mathcal{O}(B|\mathcal{V}|)$ in the number of possible locations $|\mathcal{V}|$ (if we consider a function evaluation $R(\mathcal{A})$ as atomic operation, and the lowest cost of a node is constant). In [25], it was shown that even in the non-constant cost case, the approximation guarantee of $(1-1/e)$ can be achieved. However, their algorithm is $\Omega(B|\mathcal{V}|^4)$ in the size of possible locations $|\mathcal{V}|$ we need to select from, which is prohibitive in the applications we consider. In addition, in our case studies, we show that the solutions of CEF are provably very close to the optimal score.

[3]In MAX-COVER, we pick from a collection of sets, such that the union of the picked sets is as large as possible.

3.2 Online bounds for any algorithm

The approximation guarantees of $(1-1/e)$ and $\frac{1}{2}(1-1/e)$ in the unit- and non-constant cost cases are *offline*, *i.e.*, we can state them in advance before running the actual algorithm. We can also use submodularity to acquire tight *online* bounds on the performance of an *arbitrary* placement (not just the one obtained by the CEF algorithm).

THEOREM 4. *For a placement $\widehat{\mathcal{A}} \subseteq \mathcal{V}$, and each $s \in \mathcal{V} \setminus \widehat{\mathcal{A}}$, let $\delta_s = R(\widehat{\mathcal{A}} \cup \{s\}) - R(\widehat{\mathcal{A}})$. Let $r_s = \delta_s / c(s)$, and let $s_1, \ldots, s_m$ be the sequence of locations with r_s in decreasing order. Let k be such that $C = \sum_{i=1}^{k-1} c(s_i) \leq B$ and $\sum_{i=1}^{k} c(s_i) > B$. Let $\lambda = (B - C)/c(s_k)$. Then*

$$\max_{\mathcal{A}, c(\mathcal{A}) \leq B} R(\mathcal{A}) \leq R(\widehat{\mathcal{A}}) + \sum_{i=1}^{k-1} \delta_{s_i} + \lambda \delta_{s_k}. \qquad (4)$$

Theorem 4 presents a way of computing how far any given solution $\widehat{\mathcal{A}}$ (obtained using CEF or *any* other algorithm) is from the optimal solution. This theorem can be readily turned into an algorithm, as formalized in Algorithm 2.

We empirically show that this bound is much tighter than the bound $\frac{1}{2}(1 - 1/e)$, which is roughly 31%.

4. SCALING UP THE ALGORITHM

4.1 Speeding up function evaluations

Evaluating the penalty reductions R can be very expensive. E.g., in the water distribution application, we need to run physical simulations, in order to estimate the effect of a contamination at a certain node. In the blog networks, we need to consider several millions of posts, which make up the cascades. However, in both applications, most outbreaks are sparse, *i.e.*, affect only a small area of the network (*c.f.,* [12, 16]), and hence are only detected by a small number of nodes. Hence, most nodes s do not reduce the penalty incurred by an outbreak (*i.e.*, $R_i(\{s\}) = 0$). Note, that this sparsity is *only* present if we consider penalty *reductions*. If for each sensor $s \in \mathcal{V}$ and scenario $i \in \mathcal{I}$ we store the actual penalty $\pi_i(s)$, the resulting representation is not sparse. Our implementation exploits this sparsity by representing the penalty function R as an *inverted index*[4], which allows fast lookup of the penalty reductions *by sensor index* s. By looking up all scenarios detected by all sensors in our placement $\mathcal{A}$, we can quickly compute the penalty reduction

$$R(\mathcal{A}) = \sum_{i: i \text{ detected by } \mathcal{A}} P(i) \max_{s \in \mathcal{A}} R_i(\{s\}),$$

without having to scan the entire data set.

The inverted index is the main data structure we use in our optimization algorithms. After the problem (water distribution network simulations, blog cascades) has been compressed into this structure, we use the same implementation for optimizing sensor placements and computing bounds.

In the water distribution network application, exploiting this sparsity allows us to fit the set of all possible intrusions considered in the BWSN challenge in main memory (16 GB), which leads to several orders of magnitude improvements in the running time, since we can avoid hard-drive accesses.

[4]The index is inverted, since the data set facilitates the lookup *by scenario index* i (since we need to consider cascades, or contamination simulations for each scenario).

Function: LazyForward($\mathcal{G}=(\mathcal{V},\mathcal{E})$,$R$,$c$,$B$,*type*)
$\mathcal{A} \leftarrow \emptyset$; **foreach** $s \in \mathcal{V}$ **do** $\delta_s \leftarrow +\infty$;
while $\exists s \in \mathcal{V} \setminus \mathcal{A} : c(\mathcal{A} \cup \{s\}) \leq B$ **do**
  foreach $s \in \mathcal{V} \setminus \mathcal{A}$ **do** $cur_s \leftarrow$ **false**;
  while *true* **do**
    if *type*=*UC* **then** $s^* \leftarrow \operatorname{argmax}_{s \in \mathcal{V} \setminus \mathcal{A}, c(\mathcal{A} \cup \{s\}) \leq B} \delta_s$;
    if *type*=*CB* **then** $s^* \leftarrow \operatorname{argmax}_{s \in \mathcal{V} \setminus \mathcal{A}, c(\mathcal{A} \cup \{s\}) \leq B} \frac{\delta_s}{c(s)}$;
    if cur_s **then** $\mathcal{A} \leftarrow \mathcal{A} \cup s^*$; **break** ;
    else $\delta_s \leftarrow R(\mathcal{A} \cup \{s\}) - R(\mathcal{A})$; $cur_s \leftarrow$ **true**;
return $\mathcal{A}$;

Algorithm:CELF($\mathcal{G}=(\mathcal{V},\mathcal{E})$,$R$,$c$,$B$)
$\mathcal{A}_{UC} \leftarrow$LazyForward($\mathcal{G}, R, c, B, UC$);
$\mathcal{A}_{CB} \leftarrow$LazyForward($\mathcal{G}, R, c, B, CB$);
return argmax$\{R(\mathcal{A}_{UC}), R(\mathcal{A}_{CB})\}$

Algorithm 1: The CELF algorithm.

Algorithm: GetBound($\mathcal{G}=(\mathcal{V},\mathcal{E})$,$\mathcal{A}$,$R$,$c$,$B$)
$\mathcal{A} \leftarrow \emptyset$; $\mathcal{B} \leftarrow \emptyset$; $\hat{R} = R(\mathcal{A})$;
foreach $s \in \mathcal{V}$ **do** $\delta_s \leftarrow R(\mathcal{A} \cup \{s\}) - R(\mathcal{A})$; $r_s = \frac{\delta_s}{c(s)}$;
while $\exists s \in \mathcal{V} \setminus (\mathcal{A} \cup \mathcal{B}) : c(\mathcal{A} \cup \mathcal{B} \cup \{s\}) \leq B$ **do**
  $s^* \leftarrow \operatorname{argmax}_{s \in \mathcal{V} \setminus \{\mathcal{A} \cup \mathcal{B}\}, c(\mathcal{A} \cup \mathcal{B} \cup \{s\}) \leq B} r_s$;
  $\hat{R} \leftarrow \hat{R} + \delta_{s^*}$; $\mathcal{B} \leftarrow \mathcal{B} \cup \{s^*\}$;
$s^* \leftarrow \operatorname{argmax}_{s \in \mathcal{V} \setminus \{\mathcal{A} \cup \mathcal{B}\}, c(\mathcal{A} \cup \mathcal{B} \cup \{s\}) \leq B} r_s$; $\lambda \leftarrow \frac{B - c(\mathcal{A} \cup \mathcal{B})}{c(s^*)}$;
return $\hat{R} + \lambda \delta_{s^*}$;

Algorithm 2: Getting bound $\hat{R}$ on optimal solution.

4.2 Reducing function evaluations

Even if we can quickly evaluate the score $R(\mathcal{A})$ of any given placement, we still need to perform a large number of these evaluations in order to run the greedy algorithm. If we select k sensors among $|\mathcal{V}|$ locations, we roughly need $k|\mathcal{V}|$ function evaluations. We can exploit submodularity further to require far fewer function evaluations in practice. Assume we have computed the marginal increments $\delta_s(\mathcal{A}) = R(\mathcal{A} \cup \{s\}) - R(\mathcal{A})$ (or $\delta_s(\mathcal{A})/c(s)$) for all $s \in \mathcal{V} \setminus \mathcal{A}$. The key idea is to realize that, as our node selection $\mathcal{A}$ grows, the marginal increments $\delta_{s'}$ (and $\delta_{s'}/c(s)$) (*i.e.*, the benefits for adding sensor s') can never increase: For $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$, it holds that $\delta_s(\mathcal{A}) \geq \delta_s(\mathcal{B})$. So instead of recomputing $\delta_s \equiv \delta_s(\mathcal{A})$ for every sensor after adding s' (and hence requiring $|\mathcal{V}| - |\mathcal{A}|$ evaluations of R), we perform *lazy* evaluations: Initially, we mark all δ_s as *invalid*. When finding the next location to place a sensor, we go through the nodes in decreasing order of their δ_s. If the δ_s for the top node s is invalid, we recompute it, and insert it into the existing order of the δ_s (*e.g.*, by using a priority queue). In many cases, the recomputation of δ_s will lead to a new value which is not much smaller, and hence often, the top element will stay the top element even after recomputation. In this case, we found a new sensor to add, without having reevaluated δ_s for every location s. The correctness of this lazy procedure follows directly from submodularity, and leads to far fewer (expensive) evaluations of R. We call this lazy greedy algorithm[5] CELF (Cost-Effective Lazy Forward selection). In our experiments, CELF achieved up to a factor 700 improvement in speed compared to CEF when selecting 100 blogs. Algorithm 1 provides pseudo-code for CELF.

When computing the online bounds discussed in Section 3.2, we can use a similar lazy strategy. The only difference is that, instead of lazily ensuring that the best δ_s is correctly computed, we ensure that the top k (where k is as in Eq. (4)) δ_s improvements have been updated.

5. CASE STUDY 1: BLOG NETWORK

5.1 Experimental setup

In this work we are not explicitly modeling the spread of information over the network, but rather consider cascades as *input* to our algorithms.

Here we are interested in blogs that actively participate in discussions, we biased the dataset towards the active part of the blogosphere, and selected a subset from the larger set of 2.5 million blogs of [7]. We considered all blogs that received at least 3 in-links in the first 6 months of 2006, and then took all their posts for the full year 2006. So, the dataset that we use has 45,000 blogs, 10.5 million posts, and 16.2 million links (30 GB of data). However, only 1 million links point inside the set of 45,000 blogs.

Posts have rich metadata, including time stamps, which allows us to extract information cascades, *i.e.*, subgraphs induced by directed edges representing the temporal flow of information. We adopt the following definition of a cascade [16]: every cascade has a single starting post, and other posts recursively join by linking to posts within the cascade, whereby the links obey time order. We detect cascades by first identifying starting post and then following in-links. We discover 346,209 non-trivial cascades having at least 2 nodes. Since the cascade size distribution is heavy-tailed, we further limit our analysis to only cascades that had at least 10 nodes. The final dataset has 17,589 cascades, where each blog participates in 9.4 different cascades on average.

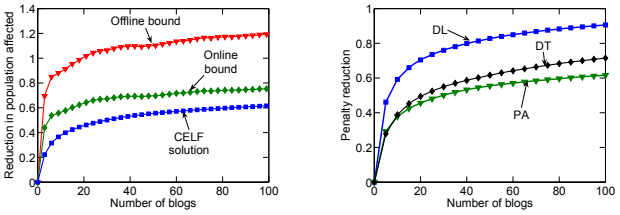
5.2 Objective functions

We use the penalty reduction objectives DL, DT and PA as introduced in Section 2.2. We normalize the scores of the solution to be between 0 and 1. For the DL (detection likelihood) criterion, the quality of the solution is the fraction of all detected cascades (regardless of when we detect it). The PA (population affected) criterion measures what fraction of the population included in the cascade after we detect it, *i.e.*, if we would be reading all the blogs initiating the cascades, then the quality of the solution is 1. In PA our reward depends on which fraction of the cascades we detect, and big cascades count more than small cascades.

5.3 Solution quality

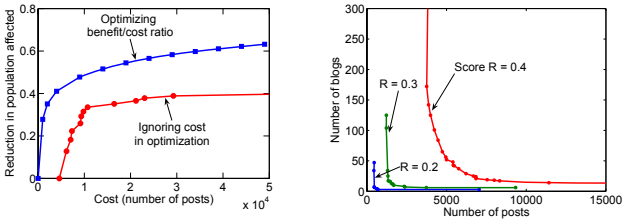
First, we evaluate the performance of CELF, and estimate how far from optimal the solution could be. Note, that obtaining the optimal solution would require enumeration of $2^{45,000}$ subsets. Since this is impractical, we compare our algorithm to the bounds we developed in Section 3. Fig. 3(a) shows scores for increasing budgets when optimized the PA (population affected) criterion. As we select more blogs to read, the proportion of cascades we catch increases (bottom line). We also plot the two bounds. The off-line bound

[5][22] suggested a similar algorithm for the *unit cost* case.



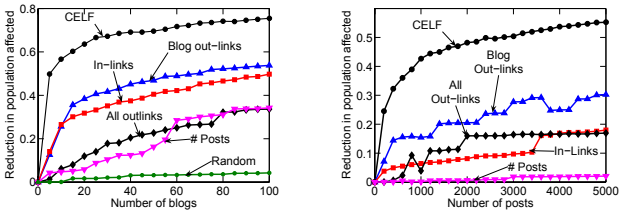
(a) Performance of CELF (b) Objective functions

Figure 3: (a) Performance of CELF algorithm and off-line and on-line bounds for PA objective function. (b) Compares objective functions.



(a) Cost of a blog (b) Cost tradeoff

Figure 4: (a) Comparison of the unit and the number of posts cost models. (b) For fixed value of PA R, we get multiple solutions varying in costs.



(a) Unit cost (b) Number of posts cost

Figure 5: Heuristic blog selection methods. (a) unit cost model, (b) number of posts cost model.

(Section 3.1) shows that the unknown optimal solution lies between our solution (bottom line) and the bound (top line). Notice the discrepancy between the lines is big, which means the bound is very loose. On the other hand, the middle line shows the online bound (Section 3.2), which again tells us that the optimal solution is somewhere between our current solution and the bound. Notice, the gap is much smaller. This means (a) that the our on-line bound is much tighter than the traditional off-line bound. And, (b) that our CELF algorithm performs very close to the optimum.

In contrast to the off-line bound, the on-line bound is *algorithm independent*, and thus can be computed regardless of the algorithm used to obtain the solution. Since it is tighter, it gives a much better worst case estimate of the solution quality. For this particular experiment, we see that CELF works very well: after selecting 100 blogs, we are at most **13.8**% away from the optimal solution.

Figure 3(b) shows the performance using various objective functions (from top to bottom: DL, DT, PA). DL increases the fastest, which means that one only needs to read a few blogs to detect most of the cascades, or equivalently that most cascades hit one of the big blogs. However, the population affected (PA) increases much slower, which means that one needs many more blogs to know about stories before the rest of population does. By using the on-line bound we also calculated that all objective functions are at most 5% to 15% from optimal.

5.4 Cost of a blog

The results presented so far assume that every blog has the same cost. Under this *unit cost* model, the algorithm tends to pick large, influential blogs, that have many posts. For example, `instapundit.com` is the best blog when optimizing PA, but it has 4,593 posts. Interestingly, most of the blogs among the top 10 are politics blogs: `instapundit.com`, `michellemalkin.com`, `blogometer.nationaljournal.com`, and `sciencepolitics.blogspot.com`. Some popular aggregators of interesting things and trends on the blogosphere are also selected: `boingboing.net`, `themodulator.org` and `bloggersblog.com`. The top 10 PA blogs had more than 21,000 thousand posts in 2006. They account for 0.2% of all posts, 3.5% of all in-links, 1.7% of out-links inside the dataset, and 0.37% of all out-links.

Under the unit cost model, large blogs are important, but reading a blog with many posts is time consuming. This motivates the *number of posts (NP)* cost model, where we set the cost of a blog to the number of posts it had in 2006.

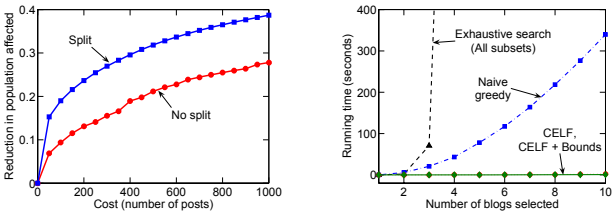
First, we compare the NP cost model with the unit cost in Fig. 4(a). The top curve shows the value of the PA criterion for budgets of B posts, *i.e.*, we optimize PA such that the selected blogs can have at most B posts total. Note, that under the unit cost model, CELF chooses expensive blogs with many posts. For example, to obtain the same PA objective value, one needs to read 10,710 posts under unit cost model. The NP cost model achieves the same score while reading just 1,500 posts. Thus, optimizing the benefit cost ratio (PA/cost) leads to drastically improved performance.

Interestingly, the solutions obtained under the NP cost model are very different from the unit cost model. Under NP, political blogs are not chosen anymore, but rather summarizers (*e.g.*, `themodulator.org`, `watcherofweasels.com`, `anglican.tk`) are important. Blogs selected under NP cost appear about 3 days later in the cascade as those selected under unit cost, which further suggests that that summarizer blogs tend to be chosen under NP model.

In practice, the cost of reading a blog is not simply proportional to the number of posts, since we also need to navigate to the blog (which takes constant effort per blog). Hence, a combination of unit and NP cost is more realistic. Fig. 4(b) interpolates between these two cost models. Each curve shows the solutions with the same value R of the PA objective, but using a different number of posts (x-axis) and blogs (y-axis) each. For a given R, the ideal spot is the one closest to origin, which means that we want to read the least number of posts from least blogs to obtain desired score R. Only at the end points does CELF tend to pick extreme solutions: few blogs with many posts, or many blogs with few posts. Note, there is a clear knee on plots of Fig. 4(b), which means that by only slightly increasing the number of blogs we allow ourselves to read, the number of posts needed decreases drastically, while still maintaining the same value R of the objective function.

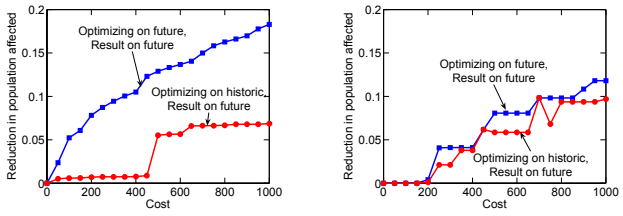
5.5 Comparison to heuristic blog selection

Next, we compare our method with several intuitive heuristic selection techniques. For example, instead of optimizing the DT, DL or PA objective function using CELF, we may



(a) Split vs. no split (b) Run time

Figure 6: (a) Improvement in performance by splitting big blogs into multiple nodes. (b) Run times of exhaustive search, greedy and CELF algorithm.



(a) All blogs (b) Only big blogs

Figure 7: Generalization to future data when CELF can select any blog (a), or only big blogs (b).

just want to select the most popular blogs and hope to detect many cascades. We considered several such heuristics, where we order blogs by some "goodness" criteria, and then pick top blogs (until the budget is exhausted). We consider the following criteria: the number posts on the blog, the cumulative number of out-links of blog's posts, the number of in-links the blog received from other blogs in the dataset, and the number of out-links to other blogs in the dataset.

As Fig. 5(a) shows, the CELF algorithm greatly outperforms all the heuristic selection techniques. More interestingly, the best heuristics (doing 45% worse than CELF) pick blogs by the number of in- or out-links from/to other blogs in the dataset. Number of posts, the total number of out-links and random blog selection do not perform well.

Number of in-links is the indicator of a blog's tendency to create cascades, while number of out-links (to other blogs) indicates blog's tendency to summarize the blogosphere. We also note, that the surprisingly good performance of the number of out-links to blogs in the dataset is an artefact of our "closed-world" dataset, and in real-life we can not estimate this. The results also agree well with our intuition that the number of in-links is a good heuristic, since it directly indicates the of propagation of information.

Fig. 5(b) explores the same setting under the NP cost model. Here, given a budget of B posts, we select a set of blogs to optimize PA objective. For the heuristics, we select a set of blogs to optimize chosen heuristic, *e.g.*, the total number of in-links of selected blogs while still fitting inside the budget of B posts. Again, CELF outperforms the next best heuristics by 41%, and again the number of in- and out-links are the best heuristics.

These results show that simple heuristics that one could use to identify blogs to read do not really work well. There are good summarizer blogs that may not be very popular, but which, by using few posts, catch most of the important stories propagating over the blogosphere.

5.6 Fractionally selecting blogs

Our framework also allows fractional selection of blogs, which means that instead of reading a large blog every day, we can read it, *e.g.*, only one day per week. This also allows us to ask: what is the best day of the week to read blogs?

In order to study whether fractional selection allows to achieve better benefit cost ratio, we split the blogs which had at least one post per day into 7 blogs, one for each day of the week. Fig. 6(a) shows, that by splitting big blogs, the population affected (PA) objective function increases for 12% over the setting where only whole blogs can be selected.

Returning to the original question, we performed the following experiment: given a budget of 1000 posts, what is the best day of the week to read posts (optimizing PA)? We found that Friday is the best day to read blogs. The value of PA for Friday is 0.20, while it is 0.13 for the rest of the week. We consider this surprising, since the activity of the blogosphere (number of posts and links created) drops towards the end of the week, and especially over the weekend [16].

5.7 Generalization to future data

Since the influence and popularity of the blogs also evolves over time we also want to know how well the selected blogs will detect cascades in the future. To evaluate the generalization to unknown future, we use the first 6 months of the dataset as historic data to select a set of blogs, and then use second 6 months of the dataset to evaluate the performance of selected blogs on unseen future cascades.

Fig. 7 compares the performance on the unknown future data. Top dashed curve in both plots shows the optimal performance on future data, *i.e.*, we select the blogs directly using the (unknown) future data. The bottom curve presents the realistic case where we select the blogs using historic data and evaluate using hidden future data.

As Fig. 7(a) shows, CELF overfits when evaluated on the future data, *i.e.*, it selects small blogs with very few posts that just by chance participate in cascades, and then these blogs do not generalize well for the second half of the year. One way to overcome this overfitting is to prevent CELF from picking very small blogs. To understand this restriction we show in Fig. 7(b) the performance when CELF can only select blogs with at least one post per day (365 posts per year).

Comparing Fig. 7(a) and Fig. 7(b) we see that the optimal performance (top curve) drops if CELF is limited on only picking big blogs. This is expected since CELF has less choice of which blogs to pick, and thus performs worse. However, when limiting the selection to only big blogs (Fig. 7(b)) the gap between the curves is very small (compared to the big gap of Fig. 7(a)). Moreover, the performance on the future data does not drop, and the method generalizes well.

5.8 Scalability

Figure 4(b) plots the running time of selecting k blogs. We see that exhaustively enumerating all possible subsets of k elements is infeasible (the line jumps out of the plot for $k = 3$). The simple greedy algorithm scales as $\Omega(k|\mathcal{V}|)$, since for every increment of k we need to consider selecting all remaining $|\mathcal{V}| - k$ blogs. The bottom line overlapping the x-axis of Fig. 4(b) shows the performance of our CELF algorithm. For example, for selecting 100 blogs, greedy algorithm runs 4.5h, while CELF takes 23 seconds (700 times faster). Calculation of the on-line bounds while running CELF takes 54s.

Exploiting the sparsity of the problem (*c.f.*, Section 4) allowed us to reduce the size of the inverted index from originally 3.5 GB to 50 MB, easily fitting it in main memory.

6. CASE STUDY 2: WATER NETWORKS

6.1 Experimental setup

In the water distribution system application, we used the data and rules introduced by the Battle of Water Sensor Networks (BWSN) challenge [19]. We considered both the small network on 129 nodes (BWSN1), and a large, realistic, 12,527 node distribution network (BWSN2) provided as part of the BWSN challenge. In addition we also consider a third water distribution network (NW3) of a large metropolitan area in the United States. The network (not including the household level) contains 21,000 nodes and 25,000 pipes (edges). To our knowledge, this is the largest water distribution network considered for sensor placement optimization so far. The networks consist of a static description (junctions and pipes) and dynamic parameters (time-varying water consumption demand patterns at different nodes, opening and closing of valves, pumps, tanks, etc.)

6.2 Objective functions

In the BWSN challenge, we want to select a set of 20 sensors, simultaneously optimizing the objective functions DT, PA and DL, as introduced in Section 2.2. To obtain cascades we use a realistic disease model defined by [19], which depends on the demands and the contaminant concentration at each node. In order to evaluate these objectives, we use the EPANET simulator [24], which is based on a physical model to provide realistic predictions on the detection time and concentration of contaminant for any possible contamination event. We consider simulations of 48 hours length, with 5 minute simulation timesteps. Contaminations can happen at any node and any time within the first 24 hours, and spread through the network according to the EPANET simulation. The time of the outbreak is important, since water consumption varies over the day and the contamination spreads at different rates depending on the time of the day. Altogether, we consider a set of 3.6 million possible contamination scenarios and each of these is associated with a "cascade" of contaminant spreading over the network.

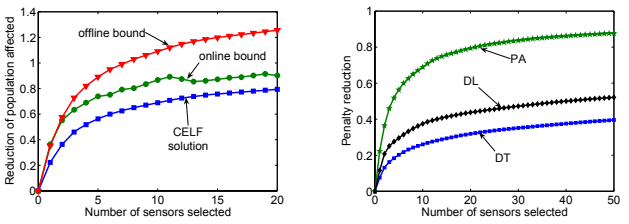
6.3 Solution quality

We first used CELF to optimize placements of increasing size, according to the three criteria DL, DT, PA. We again normalized the scores to be between 0 and 1, where 1 is the best achievable score when placing sensors at every node.

Fig. 8 (a) presents the CELF score, the off-line and on-line bounds for PA objective on the BWSN2 network. Consistently with the blog experiments, the on-line bound is much tighter than the off-line bound, and the solutions obtained by our CELF algorithm are very close to the optimum.

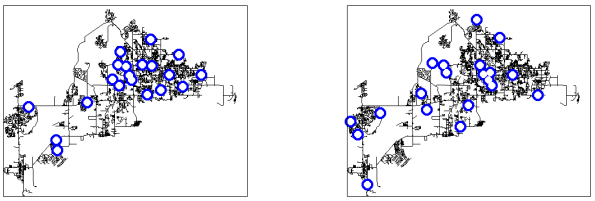
Fig. 8 (b) shows CELF's performance on all 3 objective functions. Similarly to the blog data, the population affected (PA) score increases very quickly. The reason is that most contamination events only impact a small fraction of the network. Using few sensors, it is relatively easy to detect most of the high impact outbreaks. However, if we want to detect all scenarios, we need to place a large number of sensors (2,263 in our experiment). Hence, the DL (and correspondingly DT) increase more slowly than PA.

Fig. 9 shows two 20 sensor placements after optimizing DL and PA respectively on BWSN2. When optimizing the population affected (PA), the placed sensors are concentrated in the dense high-population areas, since the goal is to detect outbreaks which affect the population the most. When optimizing the detection likelihood, the sensors are uniformly spread out over the network. Intuitively this makes sense, since according to BWSN challenge [19], outbreaks happen with same probability at every node. So, for DL, the placed sensors should be as close to all nodes as possible.



(a) Performance of CELF (b) Objective functions

Figure 8: (a) CELF with offline and online bounds for PA objective. (b) Different objective functions.



(a) PA (b) DL

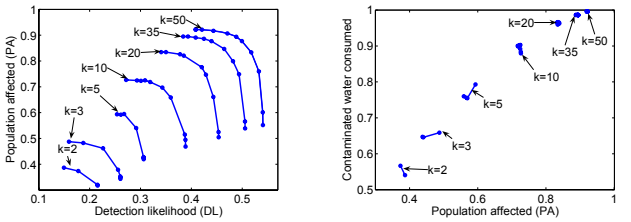
Figure 9: Water network sensor placements: (a) when optimizing PA, sensors are concentrated in high population areas. (b) when optimizing DL, sensors are uniformly spread out.

We also compared the scores achieved by CELF with several heuristic sensor placement techniques, where we order the nodes by some "goodness" criteria, and then pick the top nodes. We consider the following criteria: population at the node, water flow through the node, and the diameter and the number of pipes at the node. Fig. 11(a) shows the results for the PA objective function. CELF outperforms best heuristic by 45%. Best heuristics are placing nodes at random, by degree or their population. We see heuristics perform poorly, since nodes which are close in the graph tend to have similar flow, diameter and population, and hence the sensors will be spread out too little. Even the maximum over one hundred random trials performs far worse than CELF [15].

6.4 Multicriterion optimization

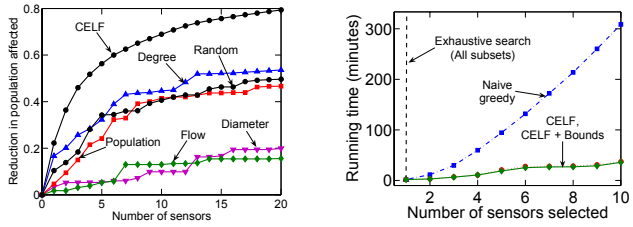
Using the theory developed in Section 2.4, we traded-off different objectives for the water distribution application. We selected pairs of objectives, *e.g.*, DL and PA, and varied the weights λ to produce (approximately) Pareto-optimal solutions. In Fig. 10 (a) we plot the tradeoff curves for different placement sizes k. By adding more sensors, both objectives DL and PA increase. The crves also show, that if we, *e.g.*, optimize for DL, the PA score can be very low. However, there are points which achieve near-optimal scores in both criteria (the *knee* in the curve). This sweet spot is what we aim for in multi-criteria optimization.

We also traded off the affected population PA and a fourth criterion defined by BWSN, the *expected consumption of contaminated water*. Fig. 10 (b) shows the trade-off curve for this experiment. Notice that the curves (almost) collapse to points, indicating that these criteria are highly correlated, which we expect for this pair of objective functions. Again,



(a) (b)

Figure 10: (a) Trading off PA and DL. (b) Trading off PA and consumed contaminated water.



(a) Comparison with random (b) Runtime

Figure 11: (a) Solutions of CELF outperform heuristic approaches. (b) Running time of exhaustive search, greedy and CELF.

the efficiency of our implementation allows to quickly generate and explore these trade-off curves, while maintaining strong guarantees about near-optimality of the results.

6.5 Scalability

In the water distribution setting, we need to simulate 3.6 million contamination scenarios, each of which takes approximately 7 seconds and produces 14KB of data. Since most of the computer cluster scheduling systems break if one would submit 3.6 million jobs into the queue, we developed a distributed architecture, where the clients obtain simulation parameters and then confirm the successful completion of the simulation. We run the simulation for a month on a cluster of around 40 machines. This produced 152GB of outbreak simulation data. By exploiting the properties of the problem described in Section 4, the size of the inverted index (which represents the relevant information for evaluating placement scores) is reduced to 16 GB which we were able to fit into main memory of a server. The fact that we could fit the data into main memory alone sped up the algorithms by at least a factor of 1000.

Fig. 11 (b) presents the running times of CELF, the naive greedy algorithm and exhaustive search (extrapolated). We can see that the CELF is 10 times faster than the greedy algorithm when placing 10 sensors. Again, a drastic speedup.

7. DISCUSSION AND RELATED WORK

7.1 Relationship to Influence Maximization

In [10], a *Triggering Model* was introduced for modeling the spread of influence in a social network. As the authors show, this model generalizes the Independent Cascade, Linear Threshold and Listen-once models commonly used for modeling the spread of influence. Essentially, this model describes a probability distribution over directed graphs, and the influence is defined as the expected number of nodes reachable from a set of nodes, with respect to this distribution. Kempe et al. showed that the problem of selecting a set of nodes with maximum influence is submodular, satisfying the conditions of Theorem 2, and hence the greedy algorithm provides a $(1-1/e)$ approximation. The problem addressed in this paper generalizes this Triggering model:

THEOREM 5. *The Triggering Model [10] is a special case of our network outbreak detection problem.*

In order to prove Theorem 5, we consider fixed directed graphs sampled from the Triggering distribution. If we revert the arcs in any such graph, then our PA objective corresponds exactly to the influence function of [10] applied to the original graph. Details of the proof can be found in [15].

Theorem 5 shows that spreading influence under the general Triggering Model is a special case of our outbreak detection formalism. The problems are fundamentally related since, when spreading influence, one tries to affect as many nodes as possible, while when detecting outbreak, one wants to minimize the effect of an outbreak in the network. Secondly, note that in the example of reading blogs, it is not necessarily a good strategy to affect nodes which are very influential, as these tend to have many posts, and hence are expensive to read. In contrast to influence maximization, the notion of cost-benefit analysis is crucial to our applications.

7.2 Related work

Optimizing submodular functions. The fundamental result about the greedy algorithm for maximizing submodular functions in the unit-cost case goes back to [17]. The first approximation results about maximizing submodular functions in the non-constant cost case were proved by [25]. They developed an algorithm with approximation guarantee of $(1-1/e)$, which however requires a number of function evaluations $\Omega(B|\mathcal{V}|^4)$ in the size of the ground set $\mathcal{V}$ (if the lowest cost is constant). In contrast, the number of evaluations required by CELF is $\mathcal{O}(B|\mathcal{V}|)$, while still providing a constant factor approximation guarantee.

Virus propagation and outbreak detection. Work on spread of diseases in networks and immunization mostly focuses on determining the value of the *epidemic threshold* [1], a critical value of the virus transmission probability above which the virus creates an epidemic. Several strategies for immunization have also been proposed: uniform node immunization, targeted immunization of high degree nodes [20] and acquaintance immunization, which focuses on highly connected nodes [5].In the context of our work, uniform immunization corresponds to randomly placing sensors in the network. Similarly, targeted immunization corresponds to selecting nodes based on their in- or out-degree. As we have seen in Figures 5 and 11, both strategies perform worse than direct optimization of the *population affected* criterion.

Information cascades and blog networks. Cascades have been studied for many years by sociologists concerned with the *diffusion of innovation* [23]; more recently, cascades we used for studying viral marketing [8, 14], selecting trendsetters in social networks [21], and explaining trends in blogspace [9, 13]. Studies of blogspace either spend effort mining topics from posts [9] or consider only the properties of blogspace as a graph of unlabeled URLs [13]. Recently, [16] studied the properties and models of information cascades in blogs. While previous work either focused on empirical analyses of information propagation and/or provided models for it, we develop a general methodology for node selection in networks while optimizing a given criterion.

Water distribution network monitoring. A number of approaches have been proposed for optimizing water sensor networks (*c.f.,* [2] for an overview of the literature). Most of these approaches are only applicable to small networks up to approximately 500 nodes. Many approaches are based on heuristics (such as genetic algorithms [18], cross-entropy selection [6], etc.) that cannot provide provable performance guarantees about the solutions. Closest to ours is an approach by [2], who equate the placement problem with a p-median problem, and make use of a large toolset of existing algorithms for this problem. The problem instances solved by [2] are a factor 72 smaller than the instances considered in this paper. In order to obtain bounds for the quality of the generated placements, the approach in [2] needs to solve a complex (NP-hard) mixed-integer program. Our approach is the first algorithm for the water network placement problem, which is guaranteed to provide solutions that achieve at least a constant fraction of the optimal solution within polynomial time. Additionally, it handles orders of magnitude larger problems than previously considered.

8. CONCLUSIONS

In this paper, we presented a novel methodology for selecting nodes to detect outbreaks of dynamic processes spreading over a graph. We showed that many important objective functions, such as detection time, likelihood and affected population are *submodular*. We then developed the CELF algorithm, which exploits submodularity to find *near-optimal* node selections – the obtained solutions are guaranteed to achieve at least a fraction of $\frac{1}{2}(1-1/e)$ of the optimal solution, even in the more complex case where every node can have an *arbitrary* cost. Our CELF algorithm is up to 700 times faster than standard greedy algorithm. We also developed novel online bounds on the quality of the solution obtained by *any* algorithm. We used these bounds to prove that the solutions we obtained in our experiments achieve 90% of the optimal score (which is intractable to compute).

We extensively evaluated our methodology on two large real-world problems: (a) detection of contaminations in the *largest water distribution* network considered so far, and (b) selection of *informative blogs* in a network of more than 10 million posts. We showed that the proposed CELF algorithm greatly outperforms intuitive heuristics. We also demonstrated that our methodology can be used to study complex application-specific questions such as multicriteria tradeoff, cost-sensitivity analyses and generalization behavior. In addition to demonstrating the effectiveness of our method, we obtained some counterintuitive results about the problem domains, such as the fact that the popular blogs might not be the most effective way to catch information.

We are convinced that the methodology introduced in this paper can apply to many other applications, such as computer network security, immunization and viral marketing.

Acknowledgements. This material is based upon work supported by the National Science Foundation under Grants No. CNS-0509383, SENSOR-0329549, IIS-0534205. This work is also supported in part by the Pennsylvania Infrastructure Technology Alliance (PITA), with additional funding from Intel, NTT, and by a generous gift from Hewlett-Packard. Jure Leskovec and Andreas Krause were supported in part by Microsoft Research Graduate Fellowship. Carlos Guestrin was supported in part by an IBM Faculty Fellowship, and an Alfred P. Sloan Fellowship.

9. REFERENCES

[1] N. Bailey. *The Mathematical Theory of Infectious Diseases and its Applications*. Griffin, London, 1975.
[2] J. Berry, W. E. Hart, C. E. Phillips, J. G. Uber, and J. Watson. Sensor placement in municipal water networks with temporal integer programming models. *J. Water Resources Planning and Management*, 2006.
[3] S. Bikhchandani, D. Hirshleifer, and I. Welch. A theory of fads, fashion, custom, and cultural change as informational cascades. *J. of Polit. Econ.*, (5), 1992.
[4] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge UP, March 2004.
[5] R. Cohen, S. Havlin, and D. ben Avraham. Efficient immunization strategies for computer networks and populations. *Physical Review Letters*, 91:247901, 2003.
[6] G. Dorini, P. Jonkergouw, and et.al. An efficient algorithm for sensor placement in water distribution systems. In *Wat. Dist. Syst. An. Conf.*, 2006.
[7] N. S. Glance, M. Hurst, K. Nigam, M. Siegler, R. Stockton, and T. Tomokiyo. Deriving marketing intelligence from online discussion. In *KDD*, 2005.
[8] J. Goldenberg, B. Libai, and E. Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, 12, 2001.
[9] D. Gruhl, R. Guha, D. Liben-Nowell, and A. Tomkins. Information diffusion through blogspace. *WWW* '04.
[10] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD*, 2003.
[11] S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Inf. Proc. Let.*, 1999.
[12] A. Krause, J. Leskovec, C. Guestrin, J. VanBriesen, and C. Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Submitted to the J. of Water Resources Planning an Management*, 2007.
[13] R. Kumar, J. Novak, P. Raghavan, and A. Tomkins. On the bursty evolution of blogspace. In *WWW*, 2003.
[14] J. Leskovec, L. A. Adamic, and B. A. Huberman. The dynamics of viral marketing. In *ACM EC*, 2006.
[15] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective Outbreak Detection in Networks. TR, CMU-ML-07-111, 2007.
[16] J. Leskovec, M. McGlohon, C. Faloutsos, N. S. Glance, and M. Hurst. Cascading behavior in large blog graphs. In *SDM*, 2007.
[17] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14, 1978.
[18] A. Ostfeld and E. Salomons. Optimal layout of early warning detection stations for water distribution systems security. *J. Water Resources Planning and Management*, 130(5):377–385, 2004.
[19] A. Ostfeld, J. G. Uber, and E. Salomons. Battle of water sensor networks: A design challenge for engineers and algorithms. In *WDSA*, 2006.
[20] R. Pastor-Satorras and A. Vespignani. Immunization of complex networks. *Physical Review E*, 65, 2002.
[21] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. *KDD* '02.
[22] T. G. Robertazzi and S. C. Schwartz. An accelerated sequential algorithm for producing D-optimal designs. *SIAM J. Sci. Stat. Comp.*, 10(2):341–358, 1989.
[23] E. Rogers. Diffusion of innovations. Free Press, 1995.
[24] L. A. Rossman. The epanet programmer's toolkit for analysis of water distribution systems. In *Ann. Wat. Res. Plan. Mgmt. Conference*, 1999.
[25] M. Sviridenko. A note on maximizing a submodular set function subject to knapsack constraint. *Operations Research Letters*, 32:41–43, 2004.