Hidden Hazards: Finding Missing Nodes in Large Graph Epidemics

Shashidhar Sundareisan[°] Jilles Vreeken[•] B. Aditya Prakash[°]

Abstract

Given a noisy or sampled snapshot of an infection in a large graph, can we automatically and reliably recover the truly infected yet somehow missed nodes? And, what about the seeds, the nodes from which the infection started to spread? These are important questions in diverse contexts, ranging from epidemiology to social media.

In this paper, we address the problem of simultaneously recovering the missing infections and the source nodes of the epidemic given noisy data. We formulate the problem by the Minimum Description Length principle, and propose NetFill, an efficient algorithm that automatically and highly accurately identifies the number *and* identities of both missing nodes and the infection seed nodes.

Experimental evaluation on synthetic and real datasets, including using data from information cascades over 96 million blog posts and news articles, shows that our method outperforms other baselines, scales near-linearly, and is highly effective in recovering missing nodes and sources.

1 Introduction

Epidemics in graphs are common. That is, many graph databases store in one way or another how information virally propagates through a graph. Natural examples include real-world viral infections (such as the flu) that propagate on population contact networks. In these epidemics, institutions like the Centers for Disease Control (CDC) try to find those who are truly infected as well as to discover the sources of the infection.

Another example is the spread of 'memes' in social media; popular phrases or links that are posted on Facebook, or re-tweeted on Twitter; with 'infected' followers doing the same. It is important to understand, from both the social sciences and marketing points of view, how such epidemics behave, what their starting points were, which nodes helped to spread it and so on.

In reality, snapshots and graphs are very noisy. There are many reasons why data may be missing in a cascade. In epidemiology for example, surveillance data on who is infected is limited and noisy [21] – the well-known 'surveillance-pyramid' demonstrates that detected infections are often only a fraction of the actual infections [17]. In Facebook, most users keep their activity and profiles private, while in Twitter only a percentage sample of Tweets are accessible by the public API. In general, as externals we seldom have access to complete cascades and even when we do, we typically analyze only small samples because of the extreme velocity of social media data. In practice we hence have to make do with noisy and incomplete snapshots.

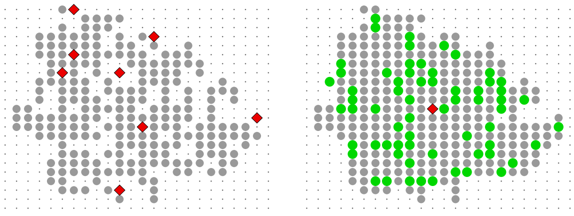

(a) NetSleuth [18] (b) NetFill

Figure 1: **Recovering missing nodes:** (a) the state of the art does *not* recover missing nodes, correct number of sources (red diamonds), nor their locations. (b) Our method finds the correct number of sources *and* recovers the missing nodes with high precision (green).

In this paper, we study the problem of recovering the missing infections as well as the source nodes (so-called 'culprits') of an epidemic. Finding culprits given noise-free snapshots of noise-free graphs is already highly complex problem [18, 22]. At the same time, in spite of its importance, missing data in context of cascades has virtually received no attention. In this paper we show that *both* these problems can be efficiently solved *simultaneously*. Figure 1 demonstrates how our method, NetFill, recovers missing data, as well as identifies culprits with high precision.

More in particular, we consider finding culprits under the Susceptible-Infected (SI) model, and we allow the given snapshot to include false positives and false negatives; nodes erroneously reported as respectively infected and healthy. Our goal is to efficiently identify the input errors – who were *truly infected* by the virus, but not reported as such in the snapshot – as well as reliably find the starting points of the epidemic.

The paper is structured as usual. After related work and preliminaries, we give the problem formulation in § 4, formally propose and empirically evaluate NetFill in § 5 and § 6, and finally we round up with discussion and conclusions in § 7 and § 8.

[°] Department of Computer Science, Virginia Tech. Email: {shashi,badityap}@cs.vt.edu.

[•] Max-Planck Institute for Informatics and Saarland University. Email: jilles@mpi-inf.mpg.de

2 Related Work

Although diffusion processes have been widely studied, the problem of 'reverse engineering' an epidemic so far only received little attention. Shah and Zaman [22] formalized the notion of *rumor-centrality* for identifying the single source node of an epidemic under the SI model, and showed an optimal algorithm for *d-regular trees.* Prakash et al. [18] studied recovering multiple seed nodes under the SI model by MDL, while Lappas et al. [12] study the problem of identifying k seed nodes, or *effectors* of a partially activated network, which is assumed to be in steady-state under the IC (Independent-Cascade) model. All three assume *complete* graphs and *noise-free* snapshots.

Missing data in networks is an important yet relatively poorly understood problem. A related line of work studies the effect of sampling on measured structural properties [3, 10, 2] or network construction [11, 15]. Another related line of work is learning graphs from sets of observed cascades [7, 6], though they assume there are no missing nodes in a cascade. Correcting for the effects of missing data in cascades in general has not seen much attention – the exception is Sadikov et al. [20], who attempt to correct for the sampling, yet only in broad statistical terms (like recovering the average size and depth of cascades) assuming a modified new cascade model (k-trees). In contrast, we address the much more general problem of *automatically* directly correcting at a *per-node* level, under a fundamental epidemic model (the SI model).

In short, to the best of our knowledge, this paper is the first to deal with *simultaneously* finding missing nodes and concealed culprits in sampled epidemics.

3 Preliminaries

We first introduce the SI model and the MDL principle.

3.1 The Susceptible-Infected Model The most basic epidemic model is the so-called 'Susceptible-Infected' (SI) model [1]. Each object/node in the underlying graph is in one of two states: Susceptible (S), or Infected (I). Once infected, a node stays infected forever. The model can be formalized for both continuous time and discrete time, the latter being intuitively most simple. At every discrete time-step, each infected node attempts to infect each of its uninfected neighbors independently with probability β, which reflects the strength of the virus. Note that $1/\beta$ hence defines a natural time-scale; intuitively it is the expected number of time-steps for a successful attack over an edge. As an example, if we assume that the underlying network is a clique of N nodes, the model can be written as: $I(t+1) = I(t) + \beta(N - I(t))I(t)$, where $I(t)$ is the number of infected nodes at time t.

3.2 Minimum Description Length Principle We employ the Minimum Description Length (MDL) principle [9] to define our objective function. Loosely speaking, MDL is a practical version of Kolmogorov Complexity [14], with both embracing the slogan *Induction by Compression.* Given a set of models $\mathcal{M}$, MDL identifies the best model M^* as the model $M \in \mathcal{M}$ that minimizes $\mathcal{L}(M) + \mathcal{L}(\mathcal{D} \mid M)$, in which $\mathcal{L}(M)$ is the length in bits of the description of model M, and $\mathcal{L}(\mathcal{D} \mid M)$ is the length of the data encoded with M.

4 Problem Formulation

In this section we discuss the problem setting, and then formalize our objective in terms of MDL.

4.1 The Problem: General Terms We consider undirected graphs $G(V, E)$, with V the nodes, and E the edges. In addition we are given a snapshot $D \subset V$ of nodes observed to be infected at time t. We allow snapshots to be incomplete with regard to the true set of infected nodes I^* (e.g., sampling errors) as well as allow nodes to be infected independent of the graph structure. We assume the contagion spread following the SI model with parameter β.

Loosely speaking, our goal is to find that 'correction' of the snapshot D that allows us to most easily describe it in terms of a SI infection cascade. The key idea is that describing D will be easier when we 'allow' the cascade to infect true missing nodes than when we force it to 'go around' these nodes. To formalize this in terms of MDL we have to define a model class $\mathcal{M}$, and how to encode model and data in bits. In recent work [18] we considered *noise-free* snapshots. Here, we do allow noise, and hence do need to formalize the cost of reaching the given data given a model, $\mathcal{L}(D \mid M)$.

4.2 Our MDL Model Class We refer to the starting points of an infection as its *seed* nodes. In the SI model, nodes neighboring an infected node are under constant attack. That is, per iteration each infected node has a probability β to successfully attack an uninfected neighbor. We refer to the set of nodes under attack at iteration i as the frontier set F^i. We write F_d^i for the subset of F^i of nodes under attack by d infected neighbors. Note that all nodes in F_d^i have the same probability of getting infected. That is, the probability of a node n getting infected depends only on β and d_{n_i}, the number of infected neighbors in iteration i.

We refer to a cascade of node infections as an infection *ripple.* Basically, a ripple R is a list that, starting from the seed nodes S, per iteration identifies the sets of nodes that were successfully attacked. We do not put any restrictions on the nodes that R may or needs to infect. That is, R may infect any node in V, also those missing from D, and R does not have to infect all of D, allowing for externally infected nodes.

Combined, a tuple (S, R) is a model $M \in \mathcal{M}$ for a given graph G and snapshot D. Together, they identify the infected footprint $I \subset V$ as the nodes infected having run the ripple from the seed nodes. Ideally, I will be equal to the true set of SI infected nodes I^*.

4.3 The Cost of the Data Given a tuple (S, R), describing D means to *correct* I wrt. D – identifying the nodes in I not in D, and vice-versa. For the nodes in I missing from D we write $C^- = I \setminus D$, and for externally infected nodes we have $C^+ = D \setminus I$. Importantly, $(I \setminus C^-) \cup C^+ = D$ describes D without loss.

In terms of MDL we have $\mathcal{L}(D \mid S, R) = \mathcal{L}(C^-) + \mathcal{L}(C^+)$. Intuitively, nodes observed as infected but which prove hard (impossible) to reach from the seeds are likely candidates for C^+, whereas those nodes not observed as infected but which strongly simplify reaching the infected footprint are likely candidates for C^-. We will use this intuition in our algorithm.

There exist use-cases without formal expectation on the number of missing nodes. We then use the intuition that larger C^- resp. C^+, should cost more bits. We have $\mathcal{L}(C^-) = \mathcal{L}_{\mathbb{N}}(|C^-| + 1) + \log \binom{|I|}{|C^-|}$, and $\mathcal{L}(C^+) = \mathcal{L}_{\mathbb{N}}(|C^+| + 1) + \log \binom{|V \setminus I|}{|C^+|}$.

In both we transmit the number of nodes using the MDL optimal code for integers ≥ 1, $\mathcal{L}_{\mathbb{N}}(z) = \log^*(z) + \log c_0$, which requires more bits for larger numbers – with $\log^*(z) = \log z + \log \log z + ...$ including only the positive terms and c_0 such that all probabilities sum to 1 [19]. The node ids we encode by an index over a canonical enumeration.

More commonly, e.g., when sampling D ourselves, we do have an expectation on the number of missing nodes and know the probability p of keeping an infected node – in other words, we expect $(100 \times p)\%$ of the truly infected nodes I^* to be in D. Here we assume a uniform sampling rate – a common strategy, e.g., used in the Twitter API. We can also interpret this as a probability $\gamma = (1 - p)$ on each node in I^* to not be in D, i.e., to be truly missing. In practice, we do not have access to I^*, yet assuming I is a good approximation we can use γ as a probability on I to be in C^-. We then have

$$\begin{aligned} \mathcal{L}(C^- \mid \gamma) &= -\log \Pr(|C^-| \mid \gamma) + \log \binom{|I|}{|C^-|} \\ \text{with } \Pr(|C^-| \mid \gamma) &= \binom{|I|}{|C^-|} \gamma^{|C^-|} (1 - \gamma)^{|I| - |C^-|} \quad , \end{aligned}$$

where we encode the size of C^- using an optimal prefix code, and then identify the node ids analogue to above. A particular strength of this encoding is that it is general for any other sampling strategy, as long as $\Pr(n \mid \theta)$ is defined accordingly.

4.4 The Cost of a Model To encode the seed nodes and the ripple, (S, R), we can use the encoding for noise-free snapshots [18]. For self-containedness we here repeat its main aspects. For seeds, we have $\mathcal{L}(S) = \mathcal{L}_{\mathbb{N}}(|S| + 1) + \log \binom{|V|}{|S|}$. For the encoded length of an SI-model infection ripple R starting from seed nodes S we have $\mathcal{L}(R \mid S) = \mathcal{L}_{\mathbb{N}}(T) + \sum_i^T \mathcal{L}(\mathcal{F}^i)$, where T is the length of the ripple in number of iterations. Per iteration we require $\mathcal{L}(\mathcal{F}^i)$ bits to identify the nodes in $\mathcal{F}^i$ that are successfully attacked, $\mathcal{L}(\mathcal{F}^i) = -\sum_{\mathcal{F}_i^d \in \mathcal{F}^i} \left(\log \Pr(m_d \mid f_d, d) + m_d \log \frac{m_d}{f_d} + (f_d - m_d) \log \left(1 - \frac{m_d}{f_d}\right)\right)$ where $f_d = |\mathcal{F}_d^i|$, and m_d is the number of nodes of attack degree d that get infected. As the SI model considers every attack an independent event with probability of success β, we can calculate the probability of seeing m_d nodes out of f_d infected by a Binomial with parameter p_d, with $\Pr(m_d \mid f_d, d) = \binom{f_d}{m_d} p_d^{m_d} (1 - p_d)^{f_d - m_d}$, where p_d expresses the independent probability of a node in $\mathcal{F}_d$ being infected, $p_d = 1 - (1 - \beta)^d$. Knowing p_d we can encode m_d using an optimal prefix code, the length of which can be calculated by Shannon entropy [4]. Knowing m_d, we use optimal prefix codes to encode whether each node in $\mathcal{F}_d$ was successfully infected or not.

4.5 The Problem: Formally With the above encodings we can now state the problem formally.

Minimal Noisy Infection Snapshot Problem *Given a graph $G(V, E)$, the SI model with infection parameter β, and nodes $D \subseteq V$ observed as infected, find the missing nodes $C^- \subseteq V \setminus D$, the observation errors $C^+ \subseteq D$, and the propagation ripple R that starts from $S \subseteq V$ and infects $I = (D \cup C^-) \setminus C^+$, minimizing $\mathcal{L}(D, S, R) = \mathcal{L}(S) + \mathcal{L}(R \mid S) + \mathcal{L}(D \mid S, R)$.*

This definition explicitly identifies the noise in D: the set of nodes $C^- = I \setminus D$ are the false negatives of D, and $C^+ = D \setminus I$ are the false positives of D. To find the optimal solution, we, however, face an immense search space: any node in $V \setminus D$ may be missing, and any node in D may be erroneous. To identify the best I we have to consider all ripples R for all $I \subseteq V$, starting from any $S \subseteq V$. As there exists no trivial structure (e.g., monotonicity) that we can exploit for fast search, and knowing that finding a *single* MLE seed node in a general graph without missing nodes or false-positives errors is #P-Complete [22], we resort to heuristics.

5 Solution and Algorithms

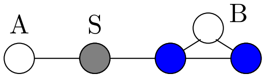
We now describe our proposed approach. Suppose an oracle gives us the true seeds from which the infection started, our goal then is to find a footprint I from which it is easy to reconstruct the observed snapshot D. To make reaching that I simpler, we can (at a cost) 'flip' uninfected nodes and consider them infected – and vice versa. For C^-, these should be nodes that make reaching I simpler; nodes for which the optimal ripple would otherwise spend bits on many unsuccessful attacks, nodes that make it easier to reach D. In contrast, C^+ should consist of nodes observed as infected but which are extremely costly to reach from the observed infections; far-apart disconnected components that can only be reached by adding overly many nodes to C^-.

5.1 Overall Strategy The above observation suggests a simple strategy: keep track of how many bits the ripple needs to encode the final state of each node, and flip the nodes with the highest cost. That is, keep track of *both* the total cost to keep it uninfected and the effort to reach the node. Intuitively, the first part corresponds to aggregated local costs: nodes with high such cost are likely good candidates for C^-. Our algorithm quickly identifies all such candidates *without* having to calculate these individual costs.

Minimizing the second part of the cost requires calculating the MLE path to *every node* in D. This is infeasible for large graphs. We note that in most applications, including epidemiology and social media, false positive rates are very low (i.e., the probability that an *observed* 'infection' is wrong). This allows the leap of faith that all connected components in D of at least 2 nodes are not purely due to chance. That is, they either require their own seed, or should be connected to another component – e.g. by nodes in C^-. Under the same assumption, we axiomatically identify C^+ as the (rare) disconnected singleton nodes of D.

To summarize, with C^+ defined as above, our task is to find a set of missing nodes C^-, a seed set S and a ripple R such that under the SI model if we start from the seed set S the infection ripple R spreads to all nodes in D and C^-, and it is the cheapest setting according to our MDL score. Next, to solve this problem we propose NetFill (§ 5.2 and 5.3).

5.2 NetFill – Main Idea Assume we are given a budget of at maximum k missing nodes $|C^-| \leq k$: how can we find the best nodes? Following from the discussion above, good candidates would be nodes which have a high cost of attempted infections. Intuitively, the larger the number of infected neighbors – i.e., the infected degree d_{n_i} – of a healthy node n, the larger the number of infection attempts, and hence the higher the cost we will have to pay to keep the node *un-infected*. Hence it seems sensible to choose the k nodes with highest d_{n_i} from the set $V \setminus D$. There are, however, two clear disadvantages to this approach: (a) we *do not* know k, and (b) we ignore both the seeds and the ripple of the infection. For example, consider the following:



Here if node S was the seed then intuitively node A should have been infected, whereas by using infected degrees, B would be the top-most candidate. Preliminary experiments showed that in practice this strategy indeed consistently outputs the wrong number and identity of seeds – even when given the true k as input.

To solve these issues, we follow a different approach. It is easy to see that the choice of C^- will affect the identity of the seed set S. Additionally, the above example demonstrates that the choice of S also affects the choice of the missing node set – this is because the seeds determine what is the best possible ripple. Consequently, cheaper ripples will require fewer missing nodes to be 'filled-in'. Following this observation, we take an EM-style alternating-minimization approach:

Task (a) find best seeds given a set of missing nodes,

Task (b) find best missing nodes given seed nodes.

Given an initialization, we alternate these steps until convergence. Task (a) is similar to finding seeds under perfect data, while Task (b) requires us to efficiently find missing node sets given a set of seeds.

5.3 NetFill – Details Next, we discuss how we solve each of the two tasks above, and then how to combine them into the NetFill algorithm.

5.3.1 Task (a): Find seeds given missing nodes In this task, we are given a set of missing nodes, and need to find the best seeds under this perfect, noise-free, information. In principle we can use any seed-finding algorithm for this task. Under perfect information, however, our MDL score reduces to that of NetSleuth [18], which is a solution towards finding a good set of seed nodes S given an accurate D: exactly Task (a)'s assumption. Hence, we simply instantiate findSeeds$(D, G(V, E), C^-)$ using NetSleuth$(G, D \cup C^-)$. Note that this allows the seeds to be 'concealed' w.r.t. D, as they may be selected from the nodes in C^-.

5.3.2 Task (b): Find missing nodes given seeds

In this step, we assume that the seed set S along with the missing nodes from the previous iteration, C^-_{prev}, are given, and our task is to find the best set of missing nodes C^-. With S given and assumed accurate, the naive approach is to list all possible C^- and let MDL decide which is the best solution. Sadly, this approach is computationally infeasible. Instead we propose to find C^- incrementally and greedily; at every step we find the next best 'hidden hazard' node n^* to add to C^-.

How should we select this 'next best node' n? From our MDL score we know that adding a node will change both the cost of the missing nodes, $\mathcal{L}(C^-)$, as well as the cost $\mathcal{L}(R)$ of the ripple from S for reaching the infected set. By the connection between encodings and distributions [9], we can interpret $\mathcal{L}(R)$ as the negative log-likelihood of the ripple. Our strategy is hence to choose that node n which maximally increases the likelihood $\mathfrak{L}(\cdot)$ that S is indeed the seed set for the resulting infections $(D \cup n)$. We remark that considering only the number of infected neighbors only takes the cost d_{n_i} of *not* infecting a node into account. Instead here the likelihood measures the *total effect* of flipping a node; that is, we implicitly consider the cost of infecting n, its neighbors, neighbors-of-neighbors and so on till we reach D. However, computing the exact total likelihood is computationally very expensive, and hence we use the total expected error $\mathfrak{R}$ instead, i.e., the empirical risk between the actual state and expected state of the snapshot if the seed set was S. Formally,

$$\mathfrak{R}(D \mid S) = \sum_{i \in V} (\mathbb{1}_{i \in D} - \mathbb{E}[\text{state of } i \mid S]) \quad , \tag{5.1}$$

where $\mathbb{1}_{i \in D} = 1$ if $i \in D$ (0 otherwise), and state of a node can be infected/uninfected. We can then use the MDL score of $\mathcal{L}(C^-) + \mathcal{L}(R)$ to see if adding n reduced the total bits – note that as S is constant in this Task, so is $\mathcal{L}(S)$.

Single seed: Assume for now we have one seed $S = \{s\}$; later we discuss how to extend this to multiple seeds. Start with $C^- = \emptyset$. From above, the best single node to add to C^- minimizes total expected error $\mathfrak{R}(D \cup n \mid s)$. Equivalently,

$$n^* = \arg\max_n \left[\mathfrak{R}(D \mid s) - \mathfrak{R}(D \cup n \mid s)\right] \quad . \tag{5.2}$$

As s was computed in Task (a) via NetSleuth, we use Lemma 3 from [18] for $\mathbb{E}[\cdot]$ into Equation 5.1 and obtain

$$\mathfrak{R}(D \mid s) \approx \sum_{i \in D} (1 - u_1(i)u_1(s)) \quad ,$$

where u_1 is the smallest eigenvector of $L_{D \cup C^-_{\text{prev}}}$ – the *submatrix* of the graph laplacian $L = \text{Deg} - G$ corresponding to the 'infected set' on which s was computed, i.e. $D \cup C^-_{\text{prev}}$. In other words, we take the subset of rows and columns from L corresponding to the nodes in $D \cup C^-_{\text{prev}}$. Note that the sum is only over the *observed* infections D while the expected states are calculated using $u_1(\cdot)$ based on s (which, in turn, was based on C^-_{prev}). So the best single node to add to C^- can be written as:

$$n^* \quad = \quad \arg\max_n \left[\sum_{i \in D \cup n} \tilde{u_1}(i)\tilde{u_1}(s) - \sum_{i \in D} u_1(i)u_1(s) \right]$$

where $\tilde{u_1}$ is the smallest eigenvector of the new laplacian submatrix we obtain after adding the node n. Thus we need to compute the *change* of the smallest eigenvector from u_1 to $\tilde{u_1}$ when the laplacian submatrix $L_{(\cdot)}$ changes after a node n is added to the infected set. Again, directly computing this change for each node is expensive as this involves $O(N)$ eigenvalue computations.

How to do this faster? We propose to use matrix perturbation theory [23] to compute this change approximately. Our Lemma 5.1 below together with the fact that many real graphs have large eigen-gap gives us a way of quickly approximating and finding the node n^*.

Lemma 5.1. *Given a seed node s, and $\lambda_1 - \lambda_2 > 3$ for $L_{D \cup C^-_{\text{prev}}}$, with $nb(n)$ the neighbors of a node n, under spectral perturbation we have $n^* \approx \arg\max_n \sum_{i \in nb(n)} u_1(i)$.*

Proof. Omitted for brevity.

Loosely speaking, u_1 measures the closeness of nodes in the infected graph to seed s. In particular, s has the largest value of $u_1(\cdot)$ and so minimizes $\mathfrak{R}(\cdot)$. Hence using above, $\mathcal{Z}_n = \sum_{i \in nb(n)} u_1(i)$ measures how *connected* a node n is to *centrally located infected nodes w.r.t. s* in D. This immediately captures our intuition that the missing nodes should depend on the seed *as well as* the structure. It is important to note that while choosing the new C^- we have to ignore the effect of the old C^-_{prev}; we need to find nodes C^- based directly on the seed s; *not* the missing-node set based on which s was itself computed. So before computing $\mathcal{Z}_n$, we set $\forall\ i \in C^-_{\text{prev}}\ u_1(i) = 0$, which ensures that z-scores are computed based only on the observed infected set and seed s. Though, nodes in C^-_{prev} can re-appear in C^- as they can still have non-zero $\mathcal{Z}$s.

Multiple seeds: The extension of the above to multiple seeds is not straightforward. Consider the case in which we have two seeds s_1 and s_2. Naively applied, our $\mathcal{Z}$-score will choose only those nodes that are 'close' to seed s_1 – e.g., in the grid network of Fig. 4(e) we would choose only nodes close to the seed in the left-blob. How

Algorithm 1: findMissing: Finds the set of missing nodes given a set of seed nodes

```
input  : Data $D$, graph $G(V,E)$, seed set $S$ and the
         old missing set $C^-_{\text{prev}}$
output : Missing nodes $C^-$
Let $S = \{\emptyset, s_1, s_2, \ldots, s_{l-1}\}$;
$\mathcal{Z}^S_n$ = findNodeScores($G,\ D,\ C^-_{\text{prev}},\ S$);
$C^- = \emptyset$ and $i = 0$;
while $\mathcal{L}(S, D, R, C^-)$ decreases do
    $C^- = C^- \cup \underset{n \in V \setminus D \cup C^-}{\arg\max}\ \mathcal{Z}^S_n$;
    $i = i + 1$;
return $C^-$;
Function findNodeScores ($G, D, C^-_{\text{prev}}, S$) :
for $i = 1$ to $l$ do
    $G_i = D \cup C^-_{\text{prev}} \setminus \bigcup_{j=0}^{i-1}\{s_j\}$ = infected subgraph;
    $u_i$ = smallest eigenvector of $G_i$;
    $u_i(l) = 0\ \ \forall\ l \in C^-_{\text{prev}}$;
    for $n \in V \setminus D$ do
        $\mathcal{Z}^i_n = \sum_{j \in nb(n)} u_i(j)$;
for node $n \in V \setminus D$ do
    $\mathcal{Z}^S_n = \max \mathcal{Z}^1_n, \mathcal{Z}^2_n, \ldots \mathcal{Z}^k_n$;
return $\mathcal{Z}^S_n$;
```

Algorithm 2: NetFill

```
input  : Data $D$, graph $G$, and infectivity $\beta$
output : Missing nodes $C^-$, ripple $R$ and seed set $S$
$C^-$ = frontier-set of $D$ in $G$;
$\{S, R\}$ = findSeeds($D, C^-, G$);
while $\mathcal{L}(S, D, R, C^-)$ decreases do
    $\{S, R\}$ = findSeeds($D, C^-, G$);
    $C^-_{\text{prev}} = C^-$;
    $C^-$ = findMissing($G, S, D, C^-_{\text{prev}}$);
return $S, R, C^-$;
```

to instead choose nodes that are close to either left and right blobs? To boost diversity, we adopt 'exoneration': we set the first seed s_1 as *un-infected* (and hence 'exonerate' the nodes close to it) and recompute the smallest eigenvector of the laplacian submatrix defined by $D \cup C^-_{\text{prev}} \setminus s_1$ (call this vector u_2). Then the $\mathcal{Z}_n$-score based on u_2 (call it $\mathcal{Z}^2_n$) will measure the appropriateness of adding a node based on its centrality w.r.t. to seed s_2. In general, for a l seed problem we will have u_1, u_2,...u_l. For these l eigenvectors we will have l $\mathcal{Z}_n$'s for every node in $V \setminus D$. For node n, we define the consolidated $\mathcal{Z}^S_n = \max\{\mathcal{Z}^1_n, \mathcal{Z}^2_n, \ldots \mathcal{Z}^l_n\}$ (as before, we also set $\forall\ i \in C^-_{\text{prev}},\ \forall\ l\ u_l(i) = 0$). Thus the best node to be added in the case of multiple seeds is a node which is very central and close to at least one of the seeds i.e. which has the maximum value of $\mathcal{Z}^S_n$.

How many nodes to add? Finally, we just add the top scoring nodes according to the $\mathcal{Z}^S_n$ scores to C^- until MDL tells us to stop. Note that we don't need to re-compute $\mathcal{Z}^S_n$ after every addition, as S is assumed correct in this Task. Algorithm 1 gives the pseudo-code.

5.3.3 The complete algorithm Given the two procedures above for Task (a) and Task (b), we can now combine them into the NetFill algorithm. We give the pseudo-code as Algorithm 2. First we need to initialize C^- for the procedure. In principle any heuristic can be used, here we choose to use the frontier-set; we set the initial C^- as the set of all those uninfected nodes which are connected to at least one infected node (the frontier set). After initialization, we calculate the seeds S for this C^- and D. We then use the alternating approach by iteratively optimizing C^- and S, until the stopping condition. As discussed at the start of this section, C^+ is defined as the disconnected *singleton* nodes in $D \cup C^-$. We stop when the MDL cost ceases to decrease. One detail is that we need to calculate the ripple R when calculating the MDL score. When doing so we know I and can hence simply follow [18] and greedily maximize the likelihood of the ripple by iteratively infecting the most likely number of nodes, which is easily computed based on the mode of Bernoulli trials.

Due to its complex nature, $\mathcal{L}(S, D, R, C^-)$ is not a pure convex function – though in practice it does show a convex-like structure. Hence, if we add one node at a time in Algorithm 1 we might get stuck at a local minima. To be more robust, instead we add batches of k nodes. Experiments show a value of ~10 works well.

Complexity: The complexity of NetFill is O($j \times (l \times (E + V) + k \times \log(V - D))$) with j is the number of iterations to converge. In our experiments we found $j \approx 3$, while l and k are $\ll D$. Hence in practice, NetFill is sub-quadratic (and near-linear in many cases).

6 Experiments

We evaluate NetFill[1] on both synthetic and real-world data. For ease of visualization we use the synthetic *Grid* network, where each node has 4 neighbors. *AS-Oregon* has a power-law degree distribution [5] and is hence a natural exemplar for biological and social networks. *MemeTracker* is based on memes (short phrases) cascading on blogs. In the *Flixster* dataset, cascades of movie ratings happen over a social network. There exist different ways of learning the historical graph from these datasets [13, 8], and they have been used in multiple information diffusion studies [7]. For *Grid* and *AS-Oregon* we simulate a SI process by

[1] Our code is available for research purposes at: `http://www.cs.vt.edu/~badityap/CODE/netfill-code.tgz`

randomly choosing seeds and validate by randomly sampling the infected nodes. For *MemeTracker* and *Flixster* we use *real* cascades from the data itself.

There exist no direct competitors to NetFill and hence we compare against three baselines (a) NetSleuth, (b) Frontier, which returns the uninfected frontier of D (i.e. $\mathcal{F}$) as C^- and seeds chosen by NetSleuth on $D \cup \mathcal{F}$ as S; and (c) Simulation, which simulates SI up till reaching D from the same seeds as Frontier. As mentioned in the survey, we do not compare against [20], as they correct only in broad statistical terms, not at the same finer granularity level as NetFill.

6.1 Evaluation Functions – Subtle Issues

There are several subtle issues for using evaluation functions. In short, as for evaluating missing nodes C^- the true negative rate is important we use both precision and the well-known Matthews Correlation Coefficient (*MCC*) [16] that takes the complete confusion matrix into account. For evaluating the seed nodes S, as the SI process is stochastic and computing exact likelihoods is intractable in practice, hence following [18] we use $Q = \frac{\mathcal{L}_{\text{alg}}(\cdot)}{\mathcal{L}_{\text{TRUE}}(\cdot)}$. For all three metrics, the closer to 1 the better.

6.2 Performance on Synthetic data

We simulate two scenarios on *Grid* using the SI process with $\beta = 0.1$: (a) n seeds to produce n distinct yet connected blobs (*Grid-Con*); and (b) n seeds to produce n disconnected blobs (*Grid-Disc*). We let the process infect between 450 and 650 nodes, we sample to get the input using $p = 0.9$. We here report results for $n = 2$, noting these are representative for larger seed sets.

First we consider NetSleuth and find, as mentioned in the introduction, *it does not work* for this problem: it does not recover missing nodes (precision zero), hence it can not find low-cost ripples (Q scores $\gg 1$), and thus too many and wrongly located seeds. See, e.g., Figs. 1 and 4a. For the remainder we hence do not compare against NetSleuth any further.

With regards to C^-, Fig. 4 shows that all other methods show good performance in returning true missing nodes (green). Whereas NetFill matches human intuition, Frontier and Simulation, however, return overly many false positives (orange). We plot the precision and *MCC* in Fig. 3 (left). This figure shows that NetFill performs best for recovering missing nodes. The baselines Simulation and Frontier choose overly many nodes. Hence, their predictions are no better than random, and we see *MCC* scores closer to zero.

Next, we investigate the near-convexity of $\mathcal{L}$, which decides how many missing nodes NetFill chooses.

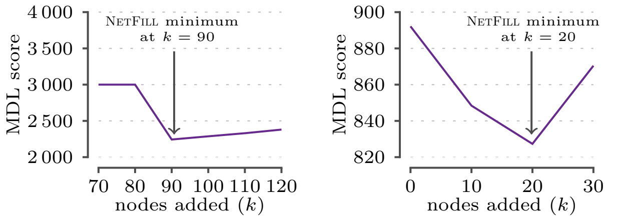


Figure 2: **Our score is near-convex, and identifies the correct size of $|C^-|$:** MDL scores for *Grid-Con* (left) and the left component of *Grid-Disc* (right).

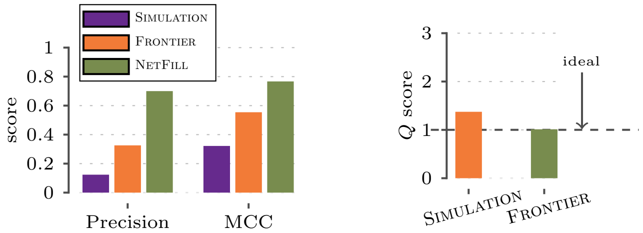


Figure 3: **NetFill performs well:** Precision and *MCC* scores (left) and Q score (right) for *Grid-Con*.

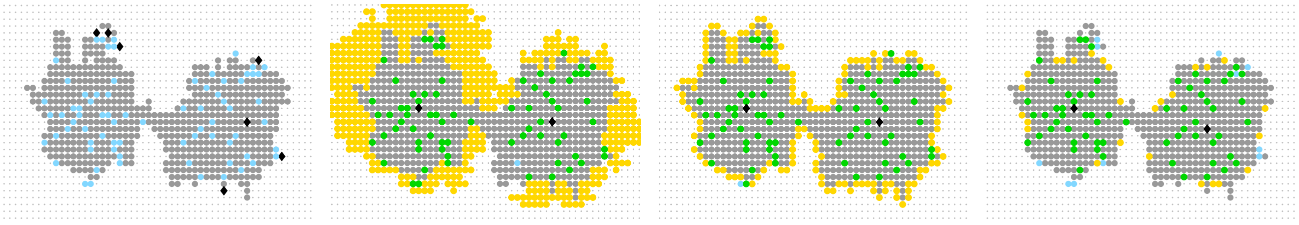
Fig. 2 shows that in practice the score is close to convex for k, with minima close to the ground truth.

With regards to seed nodes, in Fig. 4 we see that both NetFill and Frontier discover good candidates (black) – note that Simulation requires the true seed set as input parameter. To evaluate the quality of the ripple R and the seed set S, we consider the Q scores in Fig. 3. NetFill closely approximates the ideal – its solution requires as few bits as are needed for the ground truth – while many more bits are needed to describe the solutions of Simulation and Frontier. For *Grid-Disc* NetFill even finds a solution that is more simple to describe (has higher likelihood) than the ground truth.

6.3 Real Graphs with Simulated Cascades

Next, we evaluate performance of NetFill on a graph with power-law degree distribution, but keep control over the infection model. More in particular, we run SI multiple times on the *AS-Oregon* graph using a single random seed of medium to high degree. We vary the number of infected nodes from 700–1500, choose $\beta = 0.1$ and sampling using $p = 0.9$.

For the missing nodes, the results in Fig. 5 (a) we see that NetFill performs well in terms of both precision and *MCC* scores, while the baselines return so many false positives that their precision and *MCC* scores are low. Fig. 5 (b) shows that NetFill scores well too identifying seed nodes. In fact, averaged over all simulations NetFill has a Q of 1.03 – close to the ideal, implying that its seed sets S and ripple R are of high quality. Simulation and Frontier perform rather poorly with average Q scores of 7.45 and 4.57.



(a) NetSleuth (b) Simulation *given* true seeds (c) Frontier (d) NetFill

Figure 4: **Seeds and Missing Nodes:** Performance of NetFill and competitors on *Grid-Con*. NetSleuth finds *no* missing nodes and returns wrong seeds (black). Simulation (given the true seeds) and Frontier (finding good seeds) return overly many false positives (orange). NetFill performs well in both identifying missing nodes (green) as well as recovers the seeds. Grey nodes are infected, false positives are orange, and false negatives are cyan. Best viewed in color.

6.4 Real Graphs with Real Cascades

Next, we use data that defines the infected set D for us. That is, here we *do not* simulate the SI model to construct D. There exist multiple ways to extract the cascades and the graph. For ***MemeTracker*** we consider two sets of cascades: the Memetracker (*MT*) methodology of phrase matching, and cascades based on explicit hyperlinks (*HL*) [13]. For the graph, we use the *HL* and *MT* networks as learnt by NetInf [7]. We thus consider four combinations of network and infected set: *HL-HL*, *HL-MT*, *MT-HL* and *MT-HL*.

We select missing nodes using a sampling rate of $p = 0.7$, and choose $\beta = 0.1$ – noting that different values lead to similar results. We use two high-volume memes that were popular in 2008: "Lipstick on a pig" and "The state of the economy". It is important to emphasize here that SI is an abstraction; although the network here was learnt using a SI-inspired model [7], the actual spread of information in the data may not precisely match our assumptions. Moreover the network extracted here using machine-learning algorithms is itself noisy. In spite of all this, NetFill discovers interesting results.

For conciseness we only report results for *HL-MT*, noting these are representative for the other combinations. With respect to missing nodes, NetFill outperforms the baselines as seen in Fig. 5 (c). Just as for *AS-Oregon*, Frontier and Simulation perform poorly as they select almost the entire graph as C^-: NetFill's precision is more than 5× better than these baselines. With regards to the culprits, Fig. 5 (d) shows the solution discovered by NetFill has a Q score only a fraction higher than the ground truth.

Truly Missing Nodes: As *MemeTracker* is itself a sample from true cascades in the web, we can apply NetFill on the *complete* data set with the goal of discovering nodes that were missed when collecting the data. We used *HL-MT* using an expected sampling rate of $p = 0.9$. At this sampling rate NetFill identifies 22 nodes (websites) as missing, including 'nbcbayarea.com' and 'chicagotribune.com'. By checking their archives we verified that 6 sites indeed contain the meme "The state of the economy", whereas 2 others discuss politics and economic situation in USA in general. For the others verification was not possible – some are not online anymore, others do not offer searchable archives.

***Flixster*:** Last, we consider the *Flixster* dataset to evaluate how well NetFill does when the data does *not* follow our assumptions. That is, unlike for *MemeTracker*, for *Flixster* is unclear whether SI is a meaningful model – it is interesting to see whether NetFill still outperforms the baselines. In *Flixster*, the infected set D is a group of people who rated a certain movie, the undirected graph constructed from the friend relationship. We consider movies of medium volume infection, $|D| \approx 3000$. We used a sampling rate of $p = 0.9$. The edge weights were computed following [8], and we set the infection probability β as the mean edge weight.

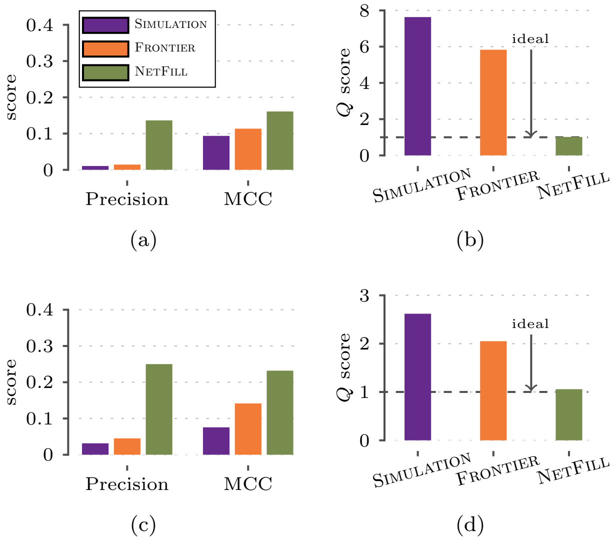
We find that NetFill here also outperforms the baselines in precision, *MCC*, as well as Q-score with a wide margin (figure omitted for brevity) – the difference being dramatic for the latter. These scores also show that NetFill is conservative when the data does not follow the model, which prevents overfitting and allows it to find relatively accurate descriptions.

Scalability and Robustness Last, but not least, we considered the scalability and robustness of NetFill. In short, NetFill is *near-linear* and robust against varying sampling rates. We omit details for brevity.

7 Discussions

The experiments demonstrate that NetFill performs very well – it obtains high precision and *MCC* scores for both simulated and real-world graphs and cascades – outperforming the baselines by a clear margin. Interestingly, NetFill works well even for the *MemeTracker* and *Flixster* datasets – which do not necessarily follow the (idealized) SI model, and for which the sampling rates are unknown – showcasing the power of our method and formulation.

The small-scale case study for *MemeTracker* also shows that our overall approach works in practice: NetFill recovered 8 *truly* missing infections from a



(a) (b) (c) (d)

Figure 5: **Good performance on real data:** *AS-Oregon* (top) and *MemeTracker HL-MT* (bottom). NetFill has best precision, *MCC*, and Q scores.

real dataset. It is valid to argue that the SI model is somewhat simplistic for the type of information diffusion in this data. Investigating how NetFill can be extended towards the richer infection models like SIR and SEIR, as well as how to incorporate infection timestamps will make for engaging future research.

8 Conclusions

In summary, we studied the problem of finding missing nodes and concealed culprits in noisy infected graphs. We approach the problem using compression, and give an efficient method, NetFill, that approximates the ideal and automatically recovers both number and identities of missing nodes and seed nodes effectively. Our main contributions include:

(a) *Problem Formulation:* We defined the missing node problem in terms of MDL: the best solution describes the data most succinctly.

(b) *Fast Algorithm:* We provide a conceptually simple and fast algorithm, NetFill, which principally optimizes our score with an EM-like approach.

(c) *Extensive Experiments:* NetFill performs very well on synthetic and real data, even giving meaningful results when our assumptions may not hold.

Acknowledgements: This material is based on work supported by the National Science Foundation under Grant no. IIS-1353346 and by the Maryland Procurement Office under contract H98230-14-C0127. JV is supported by the Cluster of Excellence "Multimodal Computing and Interaction" within the Excellence Initiative of the German Federal Government. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the respective funding agencies.

References

[1] R. M. Anderson and R. M. May. *Infectious Diseases of Humans.* Oxford University Press, 1991.

[2] S. P. Borgatti, K. M. Carley, and D. Krackhardt. On the robustness of centrality measures under conditions of imperfect data. *Soc. Netw.*, 28(2):124 – 136, 2006.

[3] E. Costenbader and T. W. Valente. The stability of centrality measures when networks are sampled. *Soc. Netw.*, 25(4):283–307, 2003.

[4] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley-Interscience, 2006.

[5] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, 1999.

[6] M. Gomez-Rodriguez, D. Balduzzi, and B. Schölkopf. Uncovering the temporal dynamics of diffusion networks. In *ICML*, 2011.

[7] M. Gomez-Rodriguez, J. Leskovec, and A. Krause. Inferring networks of diffusion and influence. In *KDD*, 2010.

[8] A. Goyal, F. Bonchi, and L. V. Lakshmanan. Learning influence probabilities in social networks. In *WSDM*, 2010.

[9] P. Grünwald. *The Minimum Description Length Principle.* MIT Press, 2007.

[10] G. Kossinets. Effects of missing data in social networks. *Soc. Netw.*, 28(3):247–268, 2006.

[11] A. Lakhina, J. W. Byers, M. Crovella, and P. Xie. Sampling biases in ip topology measurements. In *In IEEE INFOCOM*, pages 332–341, 2003.

[12] T. Lappas, E. Terzi, D. Gunopulos, and H. Mannila. Finding effectors in social networks. In *KDD*, 2010.

[13] J. Leskovec, L. Backstrom, and J. Kleinberg. Meme tracking and the dynamics of news cycle. In *KDD*, 2009.

[14] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer, 1993.

[15] A. Maiya and T. Berger-Wolf. Benefits of bias: Towards better characterization of network sampling. In *KDD*, 2011.

[16] B. Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *BBA Prot. Struct.*, 405(2):442 – 451, 1975.

[17] H. Nishiura, G. Chowell, and C. Castillo-Chavez. Did modeling overestimate the transmission potential of pandemic H1N1-2009? Sample size estimation for post-epidemic seroepidemiological studies. *PLoS ONE*, 6(3), 03 2011.

[18] B. A. Prakash, J. Vreeken, and C. Faloutsos. Spotting culprits in epidemics: How many and which ones? In *ICDM*. IEEE, 2012.

[19] J. Rissanen. Modeling by shortest data description. *Annals Stat.*, 11(2):416–431, 1983.

[20] E. Sadikov, M. Medina, J. Leskovec, and H. Garcia-Molina. Correcting for missing data in information cascades. In *WSDM*. ACM, 2011.

[21] M. Salathé, L. Bengtsson, T. J. Bodnar, D. D. Brewer, J. S. Brownstein, C. Buckee, E. M. Campbell, C. Cattuto, S. Khandelwal, P. L. Mabry, and A. Vespignani. Digital epidemiology. *PLoS Comput Biol*, 8(7), 2012.

[22] D. Shah and T. Zaman. Rumors in a network: Who's the culprit? *IEEE TIT*, 57(8):5163–5181, 2011.

[23] L. Wu, X. Ying, X. Wu, and Z.-H. Zhou. Line orthogonality in adjacency eigenspace with application to community partition. In *IJCAI*, 2011.