Graph Structure Learning from Unlabeled Data for Event Detection

Sriram Somanchi [*]
Event and Pattern Detection Laboratory, Carnegie Mellon University
Committee: Daniel B. Neill, Christos Faloutsos and Roy Maxion

Abstract

Processes such as disease propagation and information diffusion often spread over some latent network structure which must be learned from observation. Given a set of unlabeled training examples representing occurrences of an event type of interest (e.g., a disease outbreak), our goal is to learn a graph structure that can be used to accurately detect future events of that type. Motivated by new theoretical results on the consistency of constrained and unconstrained subset scans, we propose a novel framework for learning graph structure from unlabeled data by comparing the most anomalous subsets detected with and without the graph constraints. Our framework uses the mean normalized log-likelihood ratio score to measure the quality of a graph structure, and efficiently searches for the highest-scoring graph structure. Using simulated disease outbreaks injected into real-world Emergency Department data from Allegheny County, we show that our method learns a structure similar to the true underlying graph, but enables faster and more accurate detection.

Keywords: graph learning, event detection, disease surveillance, spatial scan statistic

1 Introduction

Event detection in massive data sets has applications to multiple domains, such as information diffusion or detecting disease outbreaks. In many of these domains, the data has an underlying graph or network structure: for example, an outbreak might spread via person-to-person contact, or the latest trends might propagate through a social network. In the typical, graph-based event detection problem, we are given a graph structure $G = (V, E)$ and a time series of observed counts for each graph node v_i, and must detect connected subgraphs where the recently observed counts

[*] somanchi@cmu.edu

are significantly higher than expected. For example, public health officials wish to achieve early and accurate detection of emerging outbreaks by identifying connected regions (e.g., subsets of spatially adjacent zip codes v_i) with anomalously high counts of disease cases.

Assuming that the graph structure is known, various graph-based event detection methods (Patil and Taillie, 2004) can be used to detect anomalous subgraphs. We review these methods in §1.1 below. Typically, however, the network structure is *unknown*. For example, the spread of disease may be influenced not only by spatial adjacency but also by commuting patterns (e.g., individuals work downtown but live in a suburb), contamination of food or water sources, animal migrations, or other factors. Assuming an incorrect graph structure can result in less timely and less accurate event detection, since the affected areas may be disconnected and hence may not be identified as an anomalous subgraph. In such cases, *learning* the correct graph structure (e.g., from historical data) has the potential to dramatically improve detection performance.

Thus we consider the graph-based event detection problem in the case where the true graph structure G_T is unknown and must be inferred from data. To learn the graph, we are given a set of training examples $\{D_1 \ldots D_J\}$, where each example D_j represents a different "snapshot" of the data when an event is assumed to be occurring in some subset of nodes that is connected given the (unknown) graph structure. We assume that training examples are generated from some underlying distribution on the true latent graph structure, and wish to accurately detect future events drawn from that same distribution. Thus our goal is to learn a graph structure that minimizes detection time and maximizes accuracy when used as an input for event detection.

Several recent methods (Gomez-Rodriguez *et al.*, 2010; Myers and Leskovec, 2010; Gomez-Rodriguez and Schölkopf, 2012) learn an underlying graph structure using *labeled* training data, given the true affected subset of nodes S_j^T for each training example D_j. However, in many cases labeled data is unavailable: for example, public health officials might be aware that an outbreak has occurred, but may not know which areas were affected and when. Hence we focus on learning graph structure from *unlabeled* data, where the affected subset of nodes S_j^T for each training example is not given, and we observe only the observed and expected counts at each node. In the remainder of this paper, we present a novel framework for graph structure learning from unlabeled data, and show that the graphs learned by our approach enable more timely and more accurate event detection. We support these empirical evaluations with new theoretical results on the consistency of constrained and unconstrained subset scans, as described in §3 and §4.4 below.

1.1 Graph-Based Event Detection

Given a graph $G = (V, E)$ and the observed and expected counts at each graph node, existing methods for *graph-based event detection* can be used to identify the most anomalous connected subgraph. Here we focus on the *spatial scan* framework for event detection, which was first developed by Kulldorff (1997), building on work by Naus (1965) and others, and extended to graph data by Patil and Taillie (2004). These methods maximize the *log-likelihood ratio statistic* $F(S) = \log \frac{\Pr(\text{Data} \mid H_1(S))}{\Pr(\text{Data} \mid H_0)}$ over connected subgraphs S. Searching over connected subgraphs, rather than clusters of fixed shape such as circles (Kulldorff, 1997) or rectangles (Neill and Moore, 2004), can increase detection power and accuracy for irregularly shaped spatial clusters.

In this paper, we assume that the score function $F(S)$ is an *expectation-based scan statistic* (Neill *et al.*, 2005). The null hypothesis H_0 assumes that no events are occurring, and thus each observed count x_i is assumed to be drawn from some distribution with mean equal to the expected count μ_i: $x_i \sim \text{Dist}(\mu_i)$. The alternative hypothesis $H_1(S)$ assumes that counts in subgraph S are increased by some constant multiplicative factor $q > 1$: $x_i \sim \text{Dist}(q\mu_i)$ for $v_i \in S$, and $x_i \sim \text{Dist}(\mu_i)$ for $v_i \notin S$, where q is chosen by maximum likelihood estimation. We further assume that Dist is some distribution in the *separable exponential family* (Neill, 2012), such as the Poisson, Gaussian, or exponential. This assumption enables efficient identification of the highest-scoring connected subgraph and highest-scoring unconstrained subset, which will be important components of our graph structure learning framework described below. Our evaluation results below assume the expectation-based Poisson statistic (Neill *et al.*, 2005). In this case, the log-likelihood ratio score can be computed as $F(S) = C \log(C/B) + B - C$, if $C > B$, and 0 otherwise, where $C = \sum_{v_i \in S} x_i$ and $B = \sum_{v_i \in S} \mu_i$.

Maximizing the log-likelihood ratio statistic $F(S)$ over connected subgraphs is a challenging computational problem for which multiple algorithmic approaches exist. The two main methods we consider in this paper are GraphScan (Speakman *et al.*, 2015b) and Upper Level Sets (ULS) (Patil and Taillie, 2004). GraphScan is guaranteed to find the highest-scoring connected subgraph for the expectation-based scan statistics considered here, but can take exponential time in the worst case. ULS scales quadratically with graph size, but is a heuristic that is not guaranteed to find the optimal subgraph. GraphScan requires less than a minute of computation time for a ∼100 node graph, and improves detection power as compared to ULS, but is computationally infeasible for graphs larger than 200 to 300 nodes (Speakman *et al.*, 2015b). We also note that the previously proposed FlexScan method (Tango and Takahashi, 2005) identifies subgraphs nearly identical to those detected by GraphScan, but is computationally infeasible for graphs larger than ∼30 nodes.

As shown by Speakman *et al.* (2015b), the detection performance of GraphScan and ULS is often improved by incorporating proximity as well as connectivity constraints, thus preventing these methods from identifying highly irregular tree-like structures. To do so, rather than performing a single search over the entire graph, we perform separate searches over the "local neighborhood" of each of the N graph nodes, consisting of that node and its $k-1$ nearest neighbors for some constant k. We then report the highest-scoring connected subgraph over all local neighborhoods.

2 Problem Formulation

Our framework for graph learning takes as input a set of training examples $\{D_1 \ldots D_J\}$, assumed to be independently drawn from some distribution $\mathbf{D}$. For each example D_j, we are given the observed count x_i and expected count μ_i for each graph node v_i, $i = 1 \ldots N$. We assume that each training example D_j has an set of affected nodes S_j^T that is a connected subgraph of the true underlying graph structure G_T; note that both the true graph G_T and the subgraphs S_j^T are unobserved. Unaffected nodes $v_i \notin S_j^T$ are assumed to have counts x_i that are drawn from some distribution with mean μ_i, while affected nodes $v_i \in S_j^T$ are assumed to have higher counts. Given these training examples, we have three main goals:

1) Accurately estimate the true underlying graph structure G_T. Accuracy of graph learning is measured by the precision and recall of the learned set of graph edges G^* as compared to the true graph G_T.

2) Given a separate set of test examples $\{D_1 \ldots D_J\}$ drawn from $\mathbf{D}$, identify the affected subgraphs S_j^T. Accuracy of detection is measured by the average overlap coefficient between the true and identified subgraphs.

3) Distinguish test examples drawn from $\mathbf{D}$ from examples with no affected subgraph ($S_j^T = \emptyset$). Detection power is measured by the true positive rate (proportion of correctly identified test ex-

amples) for a fixed false positive rate (proportion of incorrectly identified null examples).

The second and third performance measures assume that the learned graph G^* is used as an input for a graph-based event detection method such as GraphScan, and that method is used to identify the highest scoring connected subgraph of G^* for each test example.

A key insight of our graph learning framework is to evaluate the quality of each graph structure G_m (m denotes number of edges in the graph) by comparing the most anomalous subsets detected with and without the graph constraints. For a given training example D_j, we can use the fast subset scan (Neill, 2012) to identify the highest-scoring unconstrained subset $S_j^* = \arg\max_{S \subseteq V} F(S)$, with score $F_j = F(S_j^*)$. This can be done very efficiently, evaluating a number of subsets that is linear rather than exponential in the number of graph nodes, for any function satisfying the linear-time subset scanning property (Neill, 2012), including the expectation-based scan statistics considered here. We can use either GraphScan (Speakman *et al.*, 2015b) or ULS (Patil and Taillie, 2004) to estimate the highest-scoring connected subgraph $S_{mj}^* = \arg\max_{S \subseteq V:\, S \text{ connected in } G_m} F(S)$, with score $F_{mj} = F(S_{mj}^*)$. We then compute the *mean normalized score* $\bar{F}_{norm}(G_m) = \frac{1}{J} \sum_{j=1 \ldots J} \frac{F_{mj}}{F_j}$, averaged over all J training examples, as a measure of graph quality.

As noted above, we assume that the affected subset of nodes for each training example is a connected subgraph of the true (unknown) graph structure G_T. Intuitively, if a given graph G_m is similar to G_T, then the maximum connected subgraph score F_{mj} will be close to the maximum unconstrained subset score F_j for many training examples, and $\bar{F}_{norm}(G_m)$ will be close to 1. On the other hand, if graph G_m is missing essential connections, then we expect the values of F_{mj} to be much lower than the corresponding F_j, and $\bar{F}_{norm}(G_m)$ will be much lower than 1. Additionally, we would expect a graph G_m with high scores F_{mj} on the training examples to have high power to detect future events drawn from the same underlying distribution. However, any graph with a large number of edges will also score close to the maximum unconstrained score. For example, if graph G_m is the complete graph on N nodes, all subsets are connected, and $F_{mj} = F_j$ for all training examples D_j, giving $\bar{F}_{norm}(G_m) = 1$. Such under-constrained graphs will produce high scores F_{mj} even when data is generated under the null hypothesis, resulting in reduced detection power. Thus we wish to optimize the tradeoff between higher mean normalized score and lower number of edges m. Our solution is to compare the mean normalized score of each graph structure

G_m to the distribution of mean normalized scores for random graphs with the same number of edges m, and choose the graph with the most significant score given this distribution.

3 Theoretical Development

In this section, we provide a theoretical justification for using the mean normalized score, $\bar{F}_{norm}(G_m) = \frac{1}{J}\sum_{j=1...J}\frac{F_{mj}}{F_j}$, as a measure of the quality of graph G_m. Our key result is a proof that the expected value $E\left[\frac{F_{mj}}{F_j}\right] = 1$ if and only if graph G_m contains the true graph G_T, assuming a sufficiently strong and homogeneous signal. More precisely, let us assume the following:

(A1) Each training example D_j has an affected subset S_j^T that is a connected subgraph of G_T. Each D_j is an independent random draw from some distribution $\mathbf{D}$, where each connected subgraph S_j^T is assumed to have some non-zero probability P_j of being affected.
(A2) The score function $F(S)$ is an expectation-based scan statistic in the separable exponential family. Many distributions, such as the Poisson, Gaussian, and exponential, satisfy this property.

Now, for a given training example D_j, we define the *observed excess risk* $g_{ij} = \frac{x_i}{\mu_i} - 1$ for each node v_i. Let $r_{\max}^{\text{aff},j} = \max_{v_i \in S_j^T} g_{ij}$ and $r_{\min}^{\text{aff},j} = \min_{v_i \in S_j^T} g_{ij}$ denote the maximum and minimum of the observed excess risk over affected nodes, and $r_{\max}^{\text{unaff},j} = \max_{v_i \notin S_j^T} g_{ij}$ denote the maximum of the observed excess risk over unaffected nodes, respectively. We say that the signal for training example D_j is α-strong if and only if $r_{\min}^{\text{aff},j} > \alpha r_{\max}^{\text{unaff},j}$, and we say that the signal for training example D_j is α-homogeneous if and only if $r_{\max}^{\text{aff},j} < \alpha r_{\min}^{\text{aff},j}$. We also define the *signal size* for training example D_j, $\eta_j = \frac{\sum_{v_i \in S_j^T}\mu_i}{\sum_{v_i}\mu_i} \leq 1$. Given assumptions (A1)-(A2) above, we can show:

Lemma 1. *For each training example D_j, there exists a constant $\alpha_j > 1$ such that, if the signal is α_j-homogeneous and 1-strong, then the highest scoring unconstrained subset $S_j^* \supseteq S_j^T$. We note that α_j is a function of $r_{\max}^{aff,j}$, and $\alpha_j \geq 2$ for the Poisson, Gaussian, and exponential distributions.*

Lemma 2. *For each training example D_j, there exists a constant $\beta_j > 1$ such that, if the signal is $\frac{\beta_j}{\eta_j}$-strong, then the highest scoring unconstrained subset $S_j^* \subseteq S_j^T$. We note that β_j is a function of $r_{\max}^{unaff,j}$, and $\beta_j \leq 2$ for the Gaussian distribution.*

Proofs of Lemma 1 and Lemma 2 are provided in the Appendix.

Theorem 1. *If the signal is α_j-homogeneous and $\frac{\beta_j}{\eta_j}$-strong for all training examples $D_j \sim \mathbf{D}$, then the following properties hold for the assumed graph G_m and true graph G_T:*

a) If $G_T \setminus G_m = \emptyset$ then $E\left[\frac{F_{mj}}{F_j}\right] = 1$.

b) If $G_T \setminus G_m \neq \emptyset$ then $E\left[\frac{F_{mj}}{F_j}\right] < 1$.

Proof. Lemmas 1 and 2 imply that $S_j^* = S_j^T$ for all $D_j \sim \mathbf{D}$. For part a), $G_T \setminus G_m = \emptyset$ implies that the affected subgraph S_j^T (which is assumed to be connected in G_T) is connected in G_m as well. Thus $S_{mj}^* = S_j^T$, and $\frac{F_{mj}}{F_j} = 1$ for all $D_j \sim \mathbf{D}$. For part b), $G_T \setminus G_m \neq \emptyset$ implies that there exists some pair of nodes (v_1, v_2) such that v_1 and v_2 are connected in G_T but not in G_m. By assumption (A1), the subset $S_j^T = \{v_1, v_2\}$ has non-zero probability P_j of being generated, and we know $S_j^* = \{v_1, v_2\}$, but $S_{mj}^* \neq \{v_1, v_2\}$ since the subset is not connected in G_m. Since the signal is α_j-homogeneous and $\frac{\beta_j}{\eta_j}$-strong, we observe that S_j^* is the unique optimum. Thus we have $F_{mj} < F_j$ for that training example, and $E\left[\frac{F_{mj}}{F_j}\right] \leq 1 - P_j\left(1 - \frac{F_{mj}}{F_j}\right) < 1$. □

4 Learning Graph Structure

We can now consider the mean normalized score $\bar{F}_{norm}(G_m) = \frac{1}{J}\sum_{j=1\ldots J}\frac{F_{mj}}{F_j}$ as a measure of graph quality, and for each number of edges m, we can search for the graph G_m with highest mean normalized score. However, it is computationally infeasible to search exhaustively over all $2^{\frac{|V|(|V|-1)}{2}}$ graphs. Even computing the mean normalized score of a single graph G_m may require a substantial amount of computation time, since it requires calling a graph-based event detection method such as Upper Level Sets (ULS) or GraphScan to find the highest-scoring connected subgraph for each training example D_j. In our general framework for graph structure learning, we refer to this call as BestSubgraph(G_m, D_j), for a given graph structure G_m and training example D_j. Either ULS or GraphScan can be used to implement BestSubgraph, where ULS is faster but approximate, and GraphScan is slower but guaranteed to find the highest-scoring connected subgraph. In either case, to make graph learning computationally tractable, we must *minimize* the number of calls to BestSubgraph, both by limiting the number of graph structures under consideration, and by reducing the average number of calls needed to evaluate a given graph.

Thus we propose a *greedy* framework for efficient graph structure learning that starts with the complete graph on N nodes and sequentially removes edges until no edges remain (Algorithm 1).

This procedure produces a sequence of graphs G_m, for each m from $M = \frac{N(N-1)}{2}$ down to 0. For each graph G_m, we produce graph G_{m-1} by considering all m possible edge removals and choosing the one that maximizes the mean normalized score. We refer to this as BestEdge(G_m, D), and consider three possible implementations of BestEdge in §4.1 below. Once we have obtained the sequence of graphs $G_0 \ldots G_M$, we can then use randomization testing to choose the most significant graph G_m, as described in §4.2. The idea of this approach is to remove unnecessary edges, while preserving essential connections which keep the maximum connected subgraph score close to the maximum unconstrained subset score for many training examples.

However, a naive implementation of greedy search would require $O(N^4)$ calls to BestSubgraph, since $O(N^2)$ graph structures G_{m-1} would be evaluated for each graph G_m to choose the next edge for removal. Even a sequence of random edge removals would require $O(N^2)$ calls to BestSubgraph, to evaluate each graph $G_0 \ldots G_M$. Our efficient graph learning framework improves on both of these bounds, performing exact or approximate greedy search with $O(N^3)$ or $O(N \log N)$ calls to BestSubgraph respectively. The key insight is that removal of an edge only requires us to call BestSubgraph for those examples D_j where removing that edge disconnects the highest scoring connected subgraph. See §4.3 for further analysis and discussion.

4.1 Edge Selection Methods

Given a graph G_m with m edges, we consider three methods BestEdge(G_m, D) for choosing the next edge e_{ik} to remove, resulting in the next graph G_{m-1}. First, we consider an exact greedy search. We compute the mean normalized score $\bar{F}_{norm}(G_{m-1})$ resulting from each possible edge removal e_{ik}, and choose the edge which maximizes $\bar{F}_{norm}(G_{m-1})$. As noted above, computation of the mean normalized score for each edge removal is made efficient by evaluating the score $F_{m-1,j}$ only for training examples D_j where removing edge e_{ik} disconnects the highest scoring subgraph. The resulting graph G_{m-1} will have $\bar{F}_{norm}(G_{m-1})$ as close as possible to $\bar{F}_{norm}(G_m)$. We show in §4.3 that only $O(N)$ of the $O(N^2)$ candidate edge removals will disconnect the highest scoring subgraphs, reducing the number of calls to BestSubgraph from quartic to cubic in N. However, this still may result in overly long run times, necessitating the development of the alternative approaches below.

In the early stages of the greedy edge removal process, when the number of remaining edges m is large, many different edge removals e_{ik} might not disconnect any of the subgraphs S^*_{mj},

Algorithm 1 Graph structure learning framework

```
1: Compute correlation $\rho_{ik}$ between each pair of nodes $v_i$ and $v_k$, $i \neq k$. These will be used in step 5.
2: Compute highest-scoring unconstrained subset $S_j^*$ and its score $F_j$ for each example $D_j$ using the fast subset scan (Neill, 2012).
3: For $m = \frac{N(N-1)}{2}$, let $G_m$ be the complete graph on $N$ nodes. Set $S_{mj}^* = S_j^*$ and $F_{mj} = F_j$ for all training examples $D_j$, and set $\bar{F}_{norm}(G_m) = 1$.
4: while number of remaining edges $m > 0$ do
5:     Choose edge $e_{ik} = \text{BestEdge}(G_m, D)$, and set $G_{m-1} = G_m$ with $e_{ik}$ removed.
6:     for each training example $D_j$ do
7:         If removing edge $e_{ik}$ disconnects subgraph $S_{mj}^*$, then set $S_{m-1,j}^* = \text{BestSubgraph}(G_{m-1}, D_j)$ and $F_{m-1,j} = F(S_{m-1,j}^*)$. Otherwise set $S_{m-1,j}^* = S_{mj}^*$ and $F_{m-1,j} = F_{mj}$.
8:     end for
9:     Compute $\bar{F}_{norm}(G_{m-1}) = \frac{1}{J}\sum_{j=1\ldots J} \frac{F_{m-1,j}}{F_j}$.
10:    $m \leftarrow m - 1$
11: end while
12: Repeat steps 3-11 for $R$ randomly generated sequences of edge removals to find the most significant graph $G_m$.
```

and all such graphs would have the same mean normalized score $\bar{F}_{norm}(G_{m-1}) = \bar{F}_{norm}(G_m)$. To avoid removing potentially important edges, we must carefully consider how to break ties in mean normalized score. In this case, we choose the edge e_{ik} with lowest *correlation* between the counts at nodes v_i and v_k. If two nodes are connected to each other in the latent graph structure over which an event spreads, we expect both nodes to often be either simultaneously affected by an event in that part of the network, or simultaneously unaffected by an event in some other part of the network, and hence we expect the observed counts in these nodes to be correlated. Hence, if the Pearson correlation ρ_{ik} between two nodes v_i and v_k is very low, the probability that the two nodes are connected is small, and thus edge e_{ik} can be removed. We refer to the resulting algorithm, removing the edge e_{ik} which reduces the mean normalized score the least, and using correlation to break ties, as the Greedy Correlation (GrCorr) method.

Our second approach is based on the observation that GrCorr would require $O(m)$ calls to BestSubgraph for each graph G_m, $m = 1 \ldots M$, which may be computationally infeasible depending on the graph size and the implementation of BestSubgraph. Instead, we use the fact that $F_{m-1,j} = F_{mj}$ if removing edge e_{ik} does not disconnect subgraph S^*_{mj}, and $F_{m-1,j} < F_{mj}$ otherwise. To do so, we *count* the number of subgraphs S^*_{mj}, for $j = 1 \ldots J$, which would be disconnected by removing each possible edge e_{ik} from graph G_m, and choose the e_{ik} which disconnects the *fewest* subgraphs. The resulting graph G_{m-1} is expected to have a mean normalized score $\bar{F}_{norm}(G_{m-1})$ which is close to $\bar{F}_{norm}(G_m)$, since $F_{m-1,j} = F_{mj}$ for many subgraphs, but this approach does not guarantee that the graph G_{m-1} with highest mean normalized score will be found. However, because we choose the edge e_{ik} for which the fewest subgraphs S^*_{mj} are disconnected, and only need to call BestSubgraph for those examples D_j where removing e_{ik} disconnects S^*_{mj}, we are choosing the edge e_{ik} which requires the *fewest* calls to BestSubgraph for each graph G_m. Again, correlation is used to break ties: if two edge removals e_{ik} disconnect the same number of subgraphs, the edge with lower correlation is removed. We refer to this as Pseudo-Greedy Correlation (PsCorr), and we show in §4.3 that this approach reduces the number of calls to BestSubgraph from $O(N^3)$ to $O(N \log N)$ as compared to exact greedy search.

In our empirical results below, we compare GrCorr and PsCorr to a simple implementation of BestEdge(G_m, D), which we refer to as Correlation (Corr). Corr chooses the next edge removal e_{ik} to be the edge with the lowest value of ρ_{ik}, and hence the greedy edge removal approach corresponds to keeping all edges with correlation above some threshold ρ. Our empirical results, presented below, demonstrate that GrCorr and PsCorr significantly improve timeliness and accuracy of event detection as compared to Corr.

4.2 Finding the Most Significant Graph

Our proposed graph structure learning approach considers a set of nested graphs $\{G_1 \ldots G_M\}$, $M = \frac{N(N-1)}{2}$, where graph G_m has m edges and is formed by removing an edge from graph G_{m+1}. We note that, for this set of graphs, $\bar{F}_{norm}(G_m)$ is monotonically increasing with m, since the highest scoring connected subgraph S^*_{mj} for graph G_m will also be connected for graph G_{m+1}, and thus $F_{m+1,j} \geq F_{mj}$ for each training example D_j. Our goal is to identify the graph G_m with the best tradeoff between a high mean normalized score $\bar{F}_{norm}(G_m)$ and a small number of edges m,

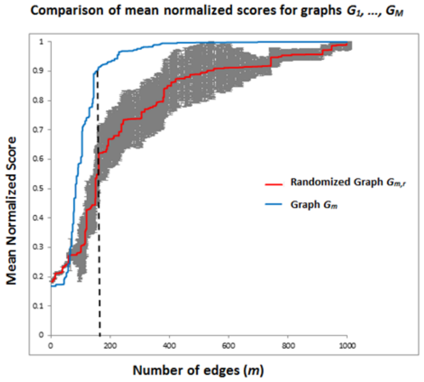

Figure 1: Example of finding the most significant graph. Blue line: mean normalized score $\bar{F}_{norm}(G_m)$ for each graph $G_1 \ldots G_M$. Red line and grey shadow: mean and standard deviation of $\bar{F}_{norm}(G_{m,r})$ for randomized graphs with m edges. Dashed line: most significant graph G^*_m.

as shown in Figure 1. Our solution is to generate a large number R of *random* permutations of the $M = \frac{N(N-1)}{2}$ edges of the complete graph on N nodes. For each permutation $r = 1 \ldots R$, we form the sequence of graphs $G_{1,r} \ldots G_{M,r}$ by removing edges in the given random order, and compute the mean normalized score of each graph. For a given number of edges m, we compute the mean μ_m and standard deviation σ_m of the mean normalized scores of the R random graphs with m edges. Finally we choose the graph $G^*_m = \arg\max_m \frac{\bar{F}_{norm}(G_m)-\mu_m}{\sigma_m}$. This "most significant graph" has the most anomalously high value of $\bar{F}_{norm}(G_m)$ given its number of edges m. Ideally, in order to compute the most significant graph structure, we want to compare our mean normalized score to the mean normalized score of any random graph with the same number of edges. However, due to the computational infeasibility of scoring all the random graph structures with varying number of edges, we instead choose random permutations of edges to be removed.

4.3 Computational Complexity Analysis

We now consider the computational complexity of each step of our graph structure learning framework (Alg. 1), in terms of the number of nodes N, number of training examples J, and number of randomly generated sequences R. Step 1 (computing correlations) requires $O(J)$ time for each of the $O(N^2)$ pairs of nodes. Step 2 (computing the highest-scoring unconstrained subsets) requires $O(N \log N)$ time for each of the J training examples, using the linear-time subset scanning method (Neill, 2012) for efficient computation. Steps 5-10 are repeated $O(N^2)$ times for the original sequence of edges and $O(N^2)$ times for each of the R randomly generated sequences of edges. Within the loop, the computation time is dominated by steps 5 and 7, and depends on our choice of BestSubgraph(G, D) and BestEdge(G, D).

For each call to BestSubgraph, GraphScan requires worst-case exponential time, approximately $O(1.2^N)$ based on empirical results by Speakman *et al.* (2015b), while the faster, heuristic ULS method requires only $O(N^2)$ time. In step 7, BestSubgraph could be called up to J times for each graph structure, for each of the R randomly generated sequences of edge removals, resulting in a total of $O(JRN^2)$ calls. However, BestSubgraph is only called when the removal of an edge e_{ik} disconnects the highest scoring connected subgraph S^*_{mj} for that graph G_m and training example D_j. We now consider the sequence of edge removals for graphs $G_1 \ldots G_M$, where $M = \frac{N(N-1)}{2}$, and compute the expected number of calls to BestSubgraph for these $O(N^2)$ edge removals. We focus on the case of random edge removals, since these dominate the overall runtime for large R.

For a given training example D_j, let x_m denote the number of nodes in the highest-scoring connected subgraph S^*_{mj} for graph G_m, and let T_m denote any spanning tree of S^*_{mj}. We note that the number of edges in T_m is $x_m - 1$, which is $O(\min(N, m))$. Moreover, any edge that is not in T_m will not disconnect S^*_{mj}, and thus the probability of disconnecting S^*_{mj} for a random edge removal is upper bounded by the ratio of the number of disconnecting edges $O(\min(N, m))$ to the total number of edges m. Thus the expected number of calls to BestSubgraph for graphs $G_1 \ldots G_M$ for the given training example is $\sum_{m=1 \ldots M} \frac{O(\min(N,m))}{m} = O(N) + \sum_{m=N \ldots M} \frac{O(N)}{m} = O(N) + O(N) \sum_{m=N \ldots M} \frac{1}{m} = O(N \log N)$. Hence the expected number of calls to BestSubgraph needed for all J training examples is $O(JN \log N)$ for the given sequence of graphs $G_1 \ldots G_M$, and $O(JRN \log N)$ for the R random sequences of edge removals.

Finally, we consider the complexity of choosing the next edge to remove (step 5 of our graph structure learning framework). The BestEdge function is called $O(N^2)$ times for the given sequence of graphs $G_1 \ldots G_M$, but is not called for the R random sequences of edge removals. For the GrCorr and PsCorr methods, for each graph G_m and each training example D_j, we must evaluate all $O(m)$ candidate edge removals. This requires a total of $O(JN^4)$ checks to determine whether removal of each edge e_{ik} disconnects the highest scoring connected subgraph S^*_{mj} for each graph G_m and training example D_j. The GrCorr method must also call BestSubgraph whenever the highest scoring subgraph is disconnected. However, for a given graph G_m and training example D_j, we show that only $O(N)$ of the $O(m)$ candidate edge removals can disconnect the highest scoring subset, thus requiring only $O(JN^3)$ calls to BestSubgraph rather than $O(JN^4)$. To see this, let x_m be the number of nodes in the highest-scoring connected subgraph S^*_{mj}, and let T_m be any spanning tree of S^*_{mj}. Then any edge that is not in T_m will not disconnect S^*_{mj}, and T_m only has $x_m - 1 = O(N)$ edges.

4.4 Consistency of Greedy Search

The greedy algorithm described above is not guaranteed to recover the true graph structure G_T. However, we can show that, given a sufficiently strong and homogeneous signal, and sufficiently many training examples, the true graph will be part of the sequence of graphs $G_0 \ldots G_M$ identified by the greedy search procedure. More precisely, let us make assumptions (A1) and (A2) given in §3 above. We also assume that GraphScan (GS) or Upper Level Sets (ULS) is used for BestSubgraph, and that Greedy Correlation (GrCorr) or Pseudo-Greedy Correlation (PsCorr) is used for selecting the next edge to remove (BestEdge). Given these assumptions, we can show:

Theorem 2. *If the signal is α_j-homogeneous and $\frac{\beta_j}{\eta_j}$-strong for all training examples $D_j \sim \mathbf{D}$, and if the set of training examples $D_1 \ldots D_J$ is sufficiently large, then the true graph G_T will be part of the sequence of graphs $G_0 \ldots G_M$ identified by Algorithm 1.*

Proof. Given an α_j-homogeneous and $\frac{\beta_j}{\eta_j}$-strong signal, both GS and ULS will correctly identify the highest-scoring connected subgraph S^*_{mj}. This is true for GS in general, since an exact search is performed, and also true for ULS since S^*_{mj} will be one of the upper level sets considered. Now let m_T denote the number of edges in the true graph G_T, and consider the sequence of graphs G_M, G_{M-1}, $\ldots$, G_{m_T+1} identified by the greedy search procedure. For each of these graphs G_m, the next edge to be removed (producing graph G_{m-1}) will be either an edge in G_T or an edge in $G_M \setminus G_T$. We will show that an edge in $G_M \setminus G_T$ is chosen for removal at each step. Given assumptions (A1)-(A2) and an α_j-homogeneous and $\frac{\beta_j}{\eta_j}$-strong signal, Theorem 1 implies:

a) For any graph that contains all edges of the true graph ($G_T \setminus G_m = \emptyset$), we will have $S^*_{mj} = S^*_j = S^T_j$ for all $D_j \sim \mathbf{D}$, and thus $\bar{F}_{norm}(G_m) = 1$.

b) For any graph that does not contain all edges of the true graph, and for any training example D_j drawn from $\mathbf{D}$, there is a non-zero probability that we will have $S^*_{mj} \neq S^*_j$, $F_{mj} < F_j$, and thus $\bar{F}_{norm}(G_m) < 1$.

We further assume that the set of training examples is sufficiently large so that every pair of nodes $\{v_1, v_2\}$ in G_T is the affected subgraph for at least one training example D_j; note that assumption (A1) ensures that each such pair will be drawn from $\mathbf{D}$ with non-zero probability. This means that removal of any edge in G_T will disconnect S^*_{mj} for at least one training example D_j, leading to $S^*_{(m-1)j} \neq S^*_{mj}$ and $\bar{F}_{norm}(G_{m-1}) < \bar{F}_{norm}(G_m)$, while removal of any edge in $G_M \setminus G_T$ will not disconnect S^*_{mj} for any training examples, maintaining $\bar{F}_{norm}(G_{m-1}) = \bar{F}_{norm}(G_m)$. Hence for both GrCorr, which removes the edge that maximizes $\bar{F}_{norm}(G_{m-1})$, and PsCorr, which removes the edge that disconnects S^*_{mj} for the fewest training examples, the greedy search procedure will remove all edges in $G_M \setminus G_T$ before removing any edges in G_T, leading to $G_{m_T} = G_T$. □

5 Related Work

We now briefly discuss several streams of related work. As noted above, various spatial scan methods have been proposed for detecting the most anomalous subset in data with an underlying, known graph structure, including Upper Level Sets (Patil and Taillie, 2004), FlexScan (Tango and Takahashi, 2005), and GraphScan (Speakman *et al.*, 2015b), but none of these methods attempt to learn an unknown graph structure from data. Link prediction algorithms such as (Taskar *et al.*, 2004; Vert and Yamanishi, 2005) start with an existing network of edges and attempt to infer additional edges which might also be present, unlike our scenario which requires inferring the complete edge structure. Much work has been done on learning the edge structure of graphical models such as Bayesian networks and probabilistic relational models (Getoor *et al.*, 2003), but these methods focus on understanding the dependencies between multiple attributes rather than learning a graph structure for event detection. Finally, the recently proposed NetInf (Gomez-Rodriguez *et al.*, 2010), ConNIe (Myers and Leskovec, 2010), and MultiTree (Gomez-Rodriguez and Schölkopf, 2012) methods share our goal of efficiently learning graph structure. NetInf is a submodular approximation algorithm for predicting the latent network structure and assumes that all connected nodes influence their neighbors with equal probability. ConNIe relaxes this assumption and uses convex programming to rapidly infer the optimal latent network, and MultiTree is an extension of NetInf which considers all possible tree structures instead of only the most probable ones. The primary difference of the present work from NetInf, ConNIe, and MultiTree is that we learn the underlying graph structure from *unlabeled* data: while these methods are given the affected subset of nodes for each time step of an event, thus allowing them to learn the network edges along which the event spreads, we consider the more difficult case where we are given only the observed and expected counts at each node, and the affected subset of nodes is not labeled. Further, these methods are not targeted towards learning a graph structure for event detection, and we demonstrate below that our approach achieves more timely and accurate event detection than MultiTree, even when MultiTree has access to the labels.

6 Experimental Setup

In our general framework, we implemented two methods for BestSubgraph(G, D): GraphScan (GS) and Upper Level Sets (ULS). We also implemented three methods for BestEdge(G, D): GrCorr, PsCorr, and Corr. However, using GraphScan with the true greedy method (GS-GrCorr) was computationally infeasible for our data, requiring 3 hours of run time for a single 50-node graph, and failing to complete for larger graphs. Hence our evaluation compares five combinations of BestSubgraph and BestEdge: GS-PsCorr, GS-Corr, ULS-GrCorr, ULS-PsCorr, and ULS-Corr.

We compare the performance of our learned graphs with the learned graphs from MultiTree, which was shown to outperform previously proposed graph structure learning algorithms such as NetInf and ConNIe (Gomez-Rodriguez and Schölkopf, 2012). We used the publicly available implementation of the algorithm, and considered both the case in which MultiTree is given the true labels of the affected subset of nodes for each training example (MultiTree-Labels), and the case in which these labels are not provided (MultiTree-NoLabels). In the latter case, we perform a subset scan for each training example D_j, and use the highest-scoring unconstrained subset S_j^* as an approximation of the true affected subset.

6.1 Description of Data

Our experiments focus on detection of simulated disease outbreaks injected into real-world Emergency Department (ED) data from ten hospitals in Allegheny County, Pennsylvania. The dataset consists of the number of ED admissions with respiratory symptoms for each of the $N = 97$ zip codes for each day from January 1, 2004 to December 31, 2005. The data were cleaned by removing all records where the admission date was missing or the home zip code was outside the county.

The resulting dataset had a daily mean of 44.0 cases, with a standard deviation of 12.1.

6.2 Graph-Based Outbreak Simulations

Our first set of simulations assume that the disease outbreak starts at a randomly chosen location and spreads over some underlying graph structure, increasing in size and severity over time. We assume that an affected node remains affected through the outbreak duration, as in the Susceptible-Infected contagion model (Bailey, 1975). For each simulated outbreak, we first choose a center zip code uniformly at random, then order the other zip codes by graph distance (number of hops away from the center for the given graph structure), with ties broken at random. Each outbreak was assumed to be 14 days in duration. On each day d of the outbreak ($d = 1 \ldots 14$), we inject counts into the k nearest zip codes, where $k = \mathit{SpreadRate} \times d$, and $\mathit{SpreadRate}$ is a parameter which determines how quickly the inject spreads. For each affected node v_i, we increment the observed count c_i^t by Poisson(λ_i^t), where $\lambda_i^t = \frac{SpreadFactor \times d}{SpreadFactor + \log(dist_i + 1)}$, and $SpreadFactor$ is a parameter which determines how quickly the inject severity decreases with distance. The assumption of Poisson counts is common in epidemiological models of disease spread; the expected number of injected cases λ_i^t is an increasing function of the inject day d, and a decreasing function of the graph distance between the affected node and the center of the outbreak. We considered 4 different inject types, as described below; for each type, we generated $J = 200$ training injects (for learning graph structure) and an additional 200 test injects to evaluate the timeliness and accuracy of event detection given the learned graph.

6.2.1 Zip code adjacency graph based injects

We first considered simulated outbreaks which spread from a given zip code to spatially adjacent zip codes, as is commonly assumed in the literature. Thus we formed the *adjacency graph* for the 97 Allegheny County zip codes, where two nodes are connected by an edge if the corresponding zip codes share a boundary. We performed two sets of experiments: for the first set, we generated simulated injects using the adjacency graph, while for the second set, we added additional edges between randomly chosen nodes to simulate travel patterns. As noted above, a contagious disease outbreak might be likely to propagate from one location to another location which is not spatially adjacent, based on individuals' daily travel, such as commuting to work or school. We hypothesize that inferring these additional edges will lead to improved detection performance.

6.2.2 Random graph based injects

Further, in order to show that we can learn a diverse set of graph structures over which an event spreads, we performed experiments assuming two types of random graphs, Erdos-Renyi and preferential attachment. For each experiment, we used the same set of nodes V consisting of the 97 Allegheny County zip codes, but created a random set of edges E connecting these nodes; the graph $G = (V, E)$ was then used to simulate 200 training and 200 test outbreaks, with results averaged over multiple such randomly chosen graphs.

First, we considered *Erdos-Renyi graphs* (assuming that each pair of nodes is connected with a constant probability p), with edge probabilities p ranging from 0.08 to 0.20. The relative performance of methods was very similar across different p values, and thus only the averaged results are reported. Second, we considered *preferential attachment graphs*, scale-free network graphs which are constructed by adding nodes sequentially, assuming that each new node forms an edge to each existing node with probability proportional to that node's degree. We generated the preferential attachment graph by first connecting three randomly chosen nodes, then adding the remaining nodes in a random order. Each new node that arrives attaches itself to each existing node v_j with probability $\frac{deg(v_j)}{\sum_i deg(v_i)}$, where each node's maximum degree was restricted to $0.2 \times |V|$.

6.3 Simulated Anthrax Bio-Attacks

We present additional evaluation results for one potentially realistic outbreak scenario, an increase in respiratory Emergency Department cases resulting from an airborne release of anthrax spores (e.g. from a bio-terrorist attack). The anthrax attacks are based on a state-of-the-art, highly realistic simulation of an aerosolized anthrax release, the Bayesian Aerosol Release Detector (BARD) simulator (Hogan *et al.*, 2007). BARD uses a combination of a dispersion model (to determine which areas will be affected and how many spores people in these areas will be exposed to), an infection model (to determine who will become ill with anthrax and visit their local Emergency Department),and a visit delay model to calculate the probability of the observed Emergency Department visit counts over a spatial region. These complex simulations take into account weather data when creating the affected zip codes and demographic information when calculating the

Table 1: Average run time in minutes for each learned graph structure, for $N = 97$ nodes.

Experiment	GraphScan (GS)		ULS			MultiTree	
	PsCorr	Corr	GrCorr	PsCorr	Corr	Labels	NoLabels
Adjacency	41	38	13	2	1	<1	<1
Adjacency+Travel	53	47	15	3	1	<1	<1
Erdos-Renyi (avg)	93	89	22	6	3	<1	<1
Pref. Attachment	49	44	17	3	1	<1	<1

Table 2: Average run time in minutes for each learned graph structure, for Erdos-Renyi graphs with varying numbers of nodes N.

Size	GraphScan (GS)		ULS			MultiTree	
	PsCorr	Corr	GrCorr	PsCorr	Corr	Labels	NoLabels
N=50	2	2	1	<1	<1	<1	<1
N=75	37	32	3	1	<1	<1	<1
N=100	58	53	13	3	<1	<1	<1
N=200	-	-	91	33	1	1	1
N=500	-	-	2958	871	27	2	2

number of additional Emergency Department cases within each affected zip code. The weather patterns are modeled with Gaussian plumes resulting in elongated, non-circular regions of affected zip codes. Wind direction, wind speed, and atmospheric stability all influence the shape and size of the affected area. A total of 82 simulated anthrax attacks were generated and injected into the Allegheny County Emergency Department data, using the BARD model. Each simulation generated between 33 and 1324 cases in total (mean = 429.2, median = 430) over a ten-day outbreak period; half of the attacks were used for training and half for testing.

7 Experimental Results

7.1 Computation Time

For each of the experiments described above (adjacency, adjacency plus travel patterns, Erdos-Renyi random graphs, and preferential attachment graphs), we report the average computation time required for each of our methods (Table 1). Randomization testing is not included in these results, since it is not dependent on the choice of BestEdge. Each sequence of randomized edge removals $G_{1,r}, \ldots, G_{M,r}$ required 1 to 2 hours for the GraphScan-based methods and 1 to 3 minutes for the ULS-based methods.

For each of the $J = 200$ training examples, all methods except for ULS-GrCorr required fewer than 80 calls to BestSubgraph on average to search over the space of $M = 4,656$ graph structures, a reduction of nearly two orders of magnitude as compared to the naive approach of calling BestSubgraph for each combination of graph structure and training example. Similarly, a naive implementation of the true greedy search would require approximately 11 million calls to BestSubgraph for each training example, while our ULS-GrCorr approach required only ∼5000 calls per training example, a three order of magnitude speedup. As expected, ULS-Corr and ULS-PsCorr had substantially faster run times than GS-Corr and GS-PsCorr, though the GraphScan-based approaches were still able to learn each graph structure in less than two hours.

Next, in order to evaluate how each method scales with the number of nodes N, we generated Erdos-Renyi random graphs with edge probability $p = 0.1$ and N ranging from 50 to 500. For each graph, we generated simulated counts and baselines, as well as simulating injects to produce $J = 200$ training examples for learning the graph structure. Table 2 shows the average time in minutes required by each method to learn the graph structure. We observe that the ULS-based methods were substantially faster than the GraphScan-based methods, and were able to scale to graphs with $N = 500$ nodes, while GS-Corr and GS-PsCorr were not computationally feasible for $N \geq 200$. We note that MultiTree has much lower computation time as compared to our graph learning methods, since it is not dependent on calls to a graph-based event detection method (BestSubgraph); however, its detection performance is lower, as shown below in our experiments.

7.2 Comparison of True and Learned Graphs

For each of the four graph-based injects (adjacency, adjacency plus travel patterns, Erdos-Renyi, and preferential attachment), we compare the learned graphs to the true underlying graph over which the simulated injects spread. Table 3 compares the number of edges in the true underlying graph to the number of edges in the learned graph structure for each of the methods, and Table 4 shows the precision and recall of the learned graph as compared to the true graph. Given the true set of edges E^T and the learned set of edges E^*, the edge precision and recall are defined to be $\frac{|E^* \cap E^T|}{|E^*|}$ and $\frac{|E^* \cap E^T|}{|E^T|}$ respectively. High recall means that the learned graph structure identifies a high proportion of the true edges, while high precision means that the learned graph does not contain too many irrelevant edges. We observe that GS-PsCorr had the highest recall, with nearly identical precision to GS-Corr and ULS-GrCorr. MultiTree had higher precision and comparable recall to GS-PsCorr when it was given the true labels, but 3-5% lower precision and recall when

Table 3: Comparison of true and learned number of edges m.

Experiment	Edges	Learned Edges						
	(true)	GraphScan (GS)		ULS			MultiTree	
		PsCorr	Corr	GrCorr	PsCorr	Corr	Labels	NoLabels
Adjacency	216	319	297	305	332	351	280	308
Adjacency+Travel	280	342	324	329	362	381	316	342
Erdos-Renyi ($p = 0.08$)	316	388	369	359	398	412	356	382
Pref. Attachment	374	394	415	401	428	461	399	416

the labels were not provided.

Table 4: Comparison of edge precision for learned graphs.

Experiment	Precision						
	GraphScan (GS)		ULS			MultiTree	
	PsCorr	Corr	GrCorr	PsCorr	Corr	Labels	NoLabels
Adjacency	0.60	0.62	0.62	0.53	0.50	0.66	0.58
Adjacency+Travel	0.70	0.71	0.69	0.60	0.52	0.75	0.65
Erdos-Renyi (avg)	0.56	0.59	0.61	0.59	0.54	0.62	0.56
Pref. Attachment	0.83	0.79	0.80	0.69	0.59	0.86	0.80

Table 5: Comparison of edge recall for learned graphs.

Experiment	Recall						
	GraphScan (GS)		ULS			MultiTree	
	PsCorr	Corr	GrCorr	PsCorr	Corr	Labels	NoLabels
Adjacency	0.89	0.86	0.88	0.81	0.77	0.86	0.83
Adjacency+Travel	0.86	0.83	0.81	0.77	0.71	0.85	0.79
Erdos-Renyi (avg)	0.87	0.81	0.83	0.79	0.70	0.84	0.79
Pref. Attachment	0.88	0.81	0.86	0.79	0.73	0.91	0.89

7.3 Comparison of Detection Performance

We now compare the detection performance of the learned graphs on the test data: a separate set of 200 simulated injects (or 41 injects for the BARD anthrax simulations), generated from the same distribution as the training injects which were used to learn that graph. To evaluate a graph, we use the GraphScan algorithm (assuming the given graph structure) to identify the highest-scoring connected subgraph S and its likelihood ratio score $F(S)$ for each day of each simulated inject, and for each day of the original Emergency Department data with no cases injected. We note that performance was substantially improved by using GraphScan for detection as compared to ULS, regardless of whether GraphScan or ULS was used to learn the graph, and GraphScan required less than a few seconds of run time for detection per day of the ED data.

We then evaluate detection performance using two metrics: average time to detection (assuming a false positive rate of 1 fp/month, typically considered acceptable by public health), and spatial accuracy (overlap between true and detected clusters). To compute detection time, we first compute the score threshold F_{thresh} for detection at 1 fp/month. This corresponds to the 96.7th percentile of the daily scores from the original ED data. Then for each simulated inject, we compute the first outbreak day d with $F(S) > F_{thresh}$; for this computation, undetected outbreaks are counted as 14 days (maximum number of inject days) to detect. We then average the time to detection over all 200 test injects. To evaluate spatial accuracy, we compute the average overlap coefficient between the detected subset of nodes S^* and the true affected subset S^T at the midpoint (day 7) of the outbreak, where overlap is defined as $\frac{|S^* \cap S^T|}{|S^* \cup S^T|}$.

As noted above, detection performance is often improved by including a proximity constraint, where we perform separate searches over the "local neighborhood" of each of the N graph nodes, consisting of that node and its $k-1$ nearest neighbors, and report the highest-scoring connected subgraph over all neighborhoods. We compare the detection performance of each graph structure by running GraphScan with varying neighborhood sizes $k = 5, 10, \ldots, 45$ for each outbreak type.

7.3.1 Results on zip code adjacency graphs

We first evaluate the detection time and spatial accuracy of GraphScan, using the learned graphs, for simulated injects which spread based on the adjacency graph formed from the 97 Allegheny County zip codes, as shown in Figure 2. This figure also shows the performance of GraphScan given the true zip code adjacency graph. We observe that the graphs learned by GS-PsCorr and ULS-GrCorr have similar spatial accuracy to the true zip code adjacency graph, as measured by the overlap coefficient between the true and detected subsets of nodes, while the graphs learned by GS-Corr and MultiTree have lower spatial accuracy. Surprisingly, all of the learned graphs achieve more timely detection than the true graph: for the optimal neighborhood size of $k = 30$, ULS-GrCorr and GS-PsCorr detected an average of 1.4 days faster than the true graph. This may be because the learned graphs, in addition to recovering most of the edges of the adjacency graph, also include additional edges to nearby but not spatially adjacent nodes (e.g. neighbors of neighbors). These extra edges provide added flexibility to consider subgraphs which would be almost but not quite connected given the true graph structure. This can improve detection time when some nodes are more strongly affected than others, enabling the strongly affected nodes to be detected earlier in the outbreak before the entire affected subgraph is identified. Finally,

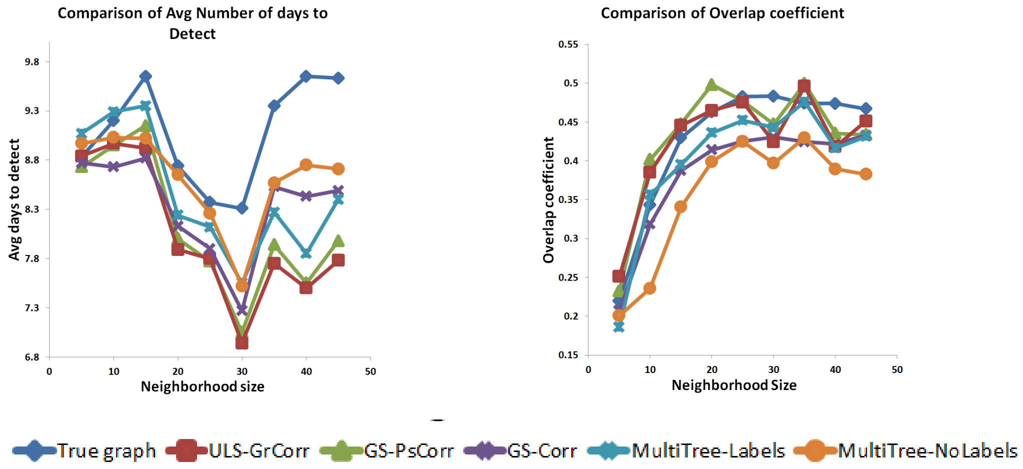

Figure 2: Comparison of detection performance of the true and learned graphs for injects based on zip code adjacency.

ULS-GrCorr and GS-PsCorr detected 0.6 days faster than MultiTree for $k = 30$.

7.3.2 Results on adjacency graphs with simulated travel patterns

Next we compared detection time and spatial accuracy, using the graphs learned by each of the methods, for simulated injects which spread based on the zip code adjacency graph with additional random edges added to simulate travel patterns, as shown in Figure 3. This figure also shows the detection performance given the true (adjacency plus travel) graph and the adjacency graph without travel patterns. We observe again that GS-PsCorr and ULS-GrCorr achieve similar spatial accuracy to the true graph, while the original adjacency graph, GS-Corr, and MultiTree have lower spatial accuracy. Our learned graphs are able to detect outbreaks 0.8 days earlier than MultiTree, 1.2 days earlier than the true graph, and 1.7 days earlier than the adjacency graph without travel patterns. This demonstrates that our methods can successfully learn the additional edges due to travel patterns, substantially improving detection performance.

7.3.3 Results on random graphs

Next we compared detection time and spatial accuracy using the learned graphs for simulated injects which spread based on Erdos-Renyi and preferential attachment graphs, as shown in Figures 4 and 5 respectively. Each figure also shows the performance of the true randomly generated graph.

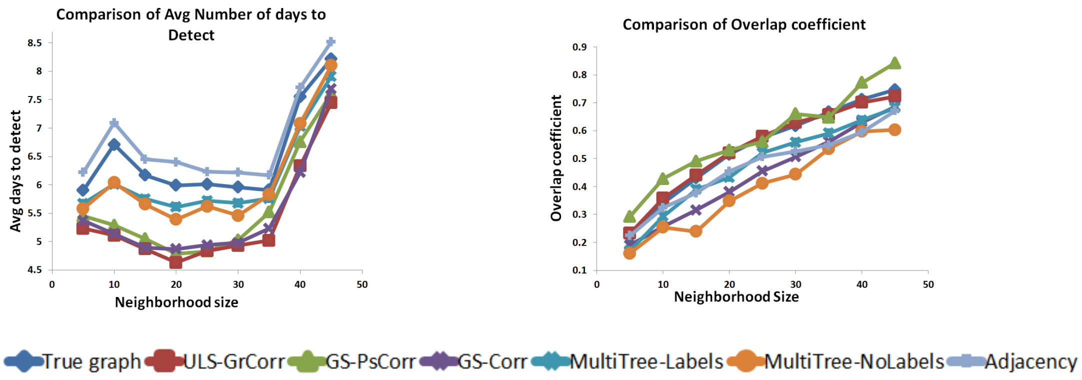

Figure 3: Comparison of detection performance of the true, learned, and adjacency graphs for injects based on adjacency with simulated travel patterns.

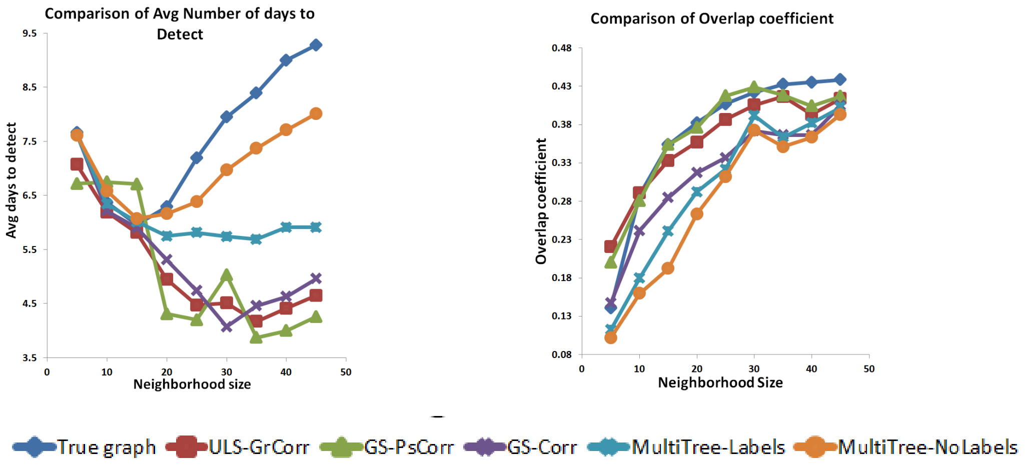

Figure 4: Comparison of detection performance of the true and learned graphs averaged over seven inject types ($p = 0.08, \ldots, 0.20$) based on Erdos-Renyi random graphs.

As in the previous experiments, we observe that our learned graphs achieve substantially faster detection than the true graph and MultiTree. For preferential attachment, the learned graphs also achieve higher spatial accuracy than the true graph, with GS-PsCorr and ULS-GrCorr again outperforming GS-Corr and MultiTree. For Erdos-Renyi, GS-PsCorr and ULS-GrCorr achieve similar spatial accuracy to the true graph, while GS-Corr and MultiTree have lower accuracy.

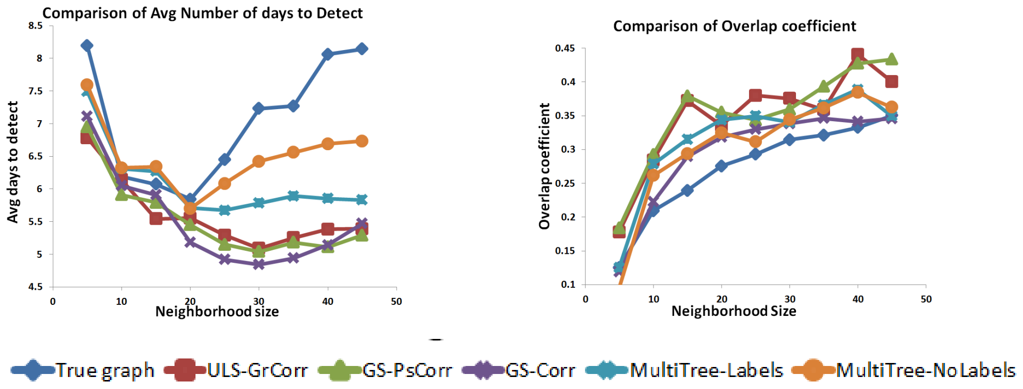

Figure 5: Comparison of detection performance of the true and learned graphs for injects based on a preferential attachment graph.

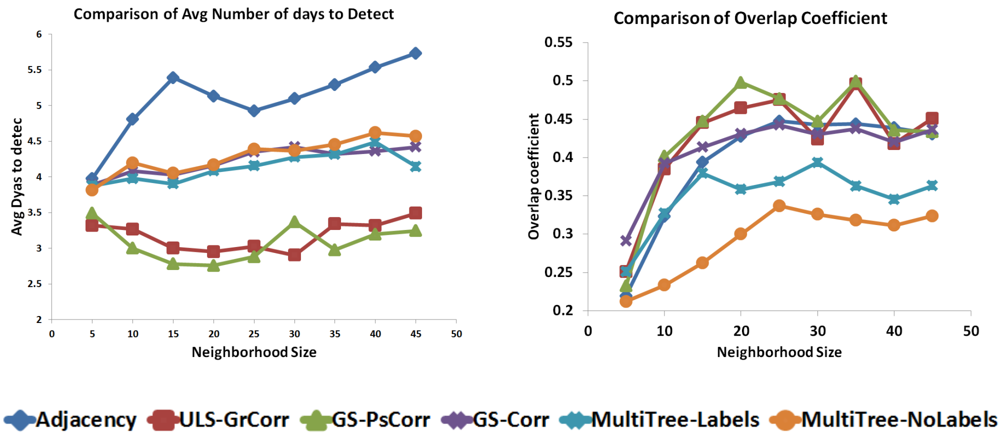

Figure 6: Comparison of detection performance of the true and learned graphs for injects based on simulated anthrax bio-attacks.

7.3.4 Results on BARD simulations

We further compared the detection time and spatial accuracy using learned graphs based on realistic simulations of anthrax bio-attacks, as shown in Figure 6. In these simulations there is no “true” graph structure as these were generated using spatial information based on environmental characteristics (wind direction, etc.). Hence, we compare the performance of various graphs learned or assumed. It can be seen that the learned graphs using GS-PsCorr and ULS-GrCorr achieve substantially faster detection and higher spatial accuracy, as compared to assuming the adjacency graph and the graphs learned using GS-Corr and MultiTree.

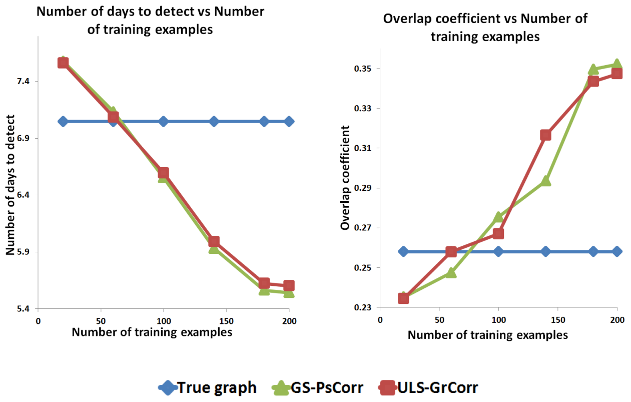

Figure 7: Effect of number of training examples on performance of GS-PsCorr and ULS-GrCorr.

7.4 Effect of number of training examples on performance

All of the experiments discussed above (except for the BARD simulations) assume $J = 200$ unlabeled training examples for learning the graph structure. We now evaluate the graphs learned by two of our best performing methods, GS-PsCorr and ULS-GrCorr, using smaller numbers of training examples ranging from $J = 20$ to $J = 200$. Simulated outbreaks were generated based on the preferential attachment graph described in §6.2.2. As shown in Figure 7, GS-PsCorr and ULS-GrCorr perform very similarly both in terms of average number of days to detect and spatial accuracy. Performance of both methods improves with increasing training set size, outperforming the true graph structure for $J > 60$.

7.5 Effect of percentage of injects in training data on performance

All of the experiments discussed above (except for the BARD simulations) assume that the J unlabeled training examples are each a "snapshot" of the observed count data c_i^t at each node v_i during a time when an event is assumed to be occurring. However, in practice the training data may be *noisy*, in the sense that some fraction of the training examples may be from time periods where no events are present. Thus we evaluate performance of the graphs learned by GS-PsCorr and ULS-GrCorr (for simulated outbreaks based on the preferential attachment graph described in §6.2.2) using a set of $J = 200$ training examples, where proportion p of the examples are based on simulated inject data, and proportion $1-p$ are drawn from the original Emergency Department data with no outbreaks injected. As shown in Figure 8, the performance of both GS-PsCorr and

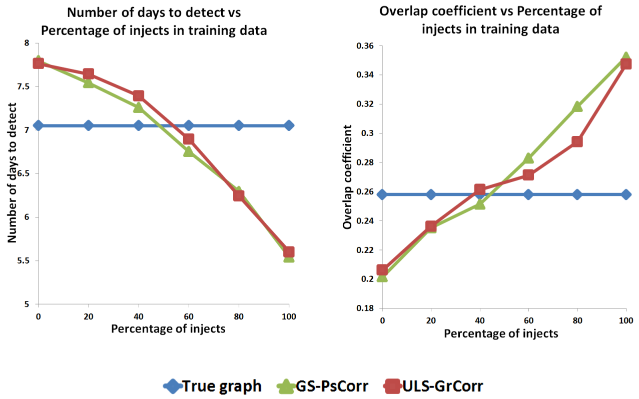

Figure 8: Effect of percentage of injects in training data on performance of GS-PsCorr and ULS-GrCorr learned graphs.

ULS-GrCorr improves as the proportion of injects p in the training data increases. For $p \geq 0.6$, both methods achieve more timely detection than the true underlying graph, with higher spatial accuracy. These results demonstrate that our graph structure learning methods, while assuming that all training examples contain true events, are robust to violations of this assumption.

8 Conclusions and Future Work

In this work, we proposed a novel framework to learn graph structure from unlabeled data, based on comparing the most anomalous subsets detected with and without the graph constraints. This approach can accurately and efficiently learn a graph structure which can then be used by graph-based event detection methods such as GraphScan, enabling more timely and more accurate detection of events (such as disease outbreaks) which spread based on that latent structure. Within our general framework for graph structure learning, we compared five approaches which differed both in the underlying detection method (BestSubgraph) and the method used to choose the next edge for removal (BestEdge), incorporated into a provably efficient greedy search procedure. We demonstrated both theoretically and empirically that our framework requires fewer calls to BestSubgraph than a naive greedy approach, $O(N^3)$ as compared to $O(N^4)$ for exact greedy search, and $O(N \log N)$ as compared to $O(N^2)$ for approximate greedy search, resulting in 2 to 3 orders of magnitude speedup in practice.

We tested these approaches on various types of simulated disease outbreaks, including outbreaks which spread according to spatial adjacency, adjacency plus simulated travel patterns, random graphs (Erdos-Renyi and preferential attachment), and realistic simulations of an anthrax bio-attack. Our results demonstrated that two of our approaches, GS-PsCorr and ULS-GrCorr, consistently outperformed the other three approaches in terms of spatial accuracy, timeliness of detection, and accuracy of the learned graph structure. Both GS-PsCorr and ULS-GrCorr consistently achieved more timely and more accurate event detection than the recently proposed MultiTree algorithm (Gomez-Rodriguez and Schölkopf, 2012), even when MultiTree was provided with labeled data not available to our algorithms. We observed a tradeoff between scalability and detection: GS-PsCorr had slightly better detection performance than ULS-GrCorr, while ULS-GrCorr was able to scale to larger graphs (500 nodes vs. 100 nodes). None of our approaches are designed to scale to massive graphs with millions of nodes (e.g. online social networks); they are most appropriate for moderate-sized graphs where labeled data is not available and timely, accurate event detection is paramount.

In general, our results demonstrate that the graph structures learned by our framework are similar to the true underlying graph structure, capturing nearly all of the true edges but also adding some additional edges. The resulting graph achieves similar spatial accuracy to the true graph, as measured by the overlap coefficient between true and detected clusters. Interestingly, the learned graph often has *better* detection power than the true underlying graph, enabling more timely detection of outbreaks or other emerging events. This result can be better understood when we realize that the learning procedure is designed to capture not only the underlying graph structure, but the characteristics of the events which spread over that graph. Unlike previously proposed methods, our framework learns these characteristics from unlabeled training examples, for which we assume that an event is occurring but are not given the affected subset of nodes. By finding graphs where the highest connected subgraph score is consistently close to the highest unconstrained subset score when an event is occurring, we identify a graph structure which is optimized for event detection. Our ongoing work focuses on extending the graph structure learning framework in several directions, including learning graph structures with directed rather than undirected edges, learning graphs with weighted edges, and learning dynamic graphs where the edge structure can change over time.

Acknowledgments

This work was partially supported by NSF grants IIS-0916345, IIS-0911032, and IIS-0953330. Preliminary work was presented at the 2011 International Society for Disease Surveillance Annual Conference, with a 1-page abstract published in the *Emerging Health Threats Journal.* This preliminary work did not include the theoretical developments and results, the computational algorithmic advances, and the large set of comparison methods and evaluations considered here.

A Proofs of Lemma 1 and Lemma 2

We begin with some preliminaries which will be used in both proofs. Following the notation in Neill (2012), we write the distributions from the exponential family as $\log P(x \mid \mu) = T(x)\theta(\mu) - \psi(\theta(\mu)) = T(x)\theta(\mu) - \mu\theta(\mu) + \phi(\mu)$, where $T(x)$ is the sufficient statistic, $\theta(\mu)$ is a function mapping the mean μ to the natural parameter θ, ψ is the log-partition function, and ϕ is the convex conjugate of ψ. By assumption (A2), $F(S)$ is an expectation-based scan statistic in the *separable exponential family*, defined by Neill (2012) as follows:

Definition 1. *The separable exponential family is a subfamily of the exponential family such that $\theta(q\mu_i) = z_i\theta_0(q) + v_i$, where the function θ_0 depends only on q, while z_i and v_i can depend on μ_i and σ_i but are independent of q.*

Such functions can be written in the form $F(S) = \max_{q>1} \sum_{s_i \in S} \lambda_i(q)$, where:

$$\lambda_i(q) = T(x_i)z_i(\theta_0(q) - \theta_0(1)) + \mu_i z_i \left(\theta_0(1) - q\theta_0(q) + \int_1^q \theta_0(x)\, dx\right).$$

Speakman *et al.* (2015a) have shown that $\lambda_i(q)$ is a concave function with global maximum at $q = q_i^{mle}$ and zeros at $q = 1$ and $q = q_i^{max}$, where $q_i^{mle} = \frac{T(x_i)}{\mu_i}$ and q_i^{max} is an increasing function of q_i^{mle}. Considering the corresponding excess risks $r_i^{mle} = q_i^{mle} - 1$ and $r_i^{max} = q_i^{max} - 1$, we know:

$$r_i^{mle} = r_i^{max} \left(\frac{\theta_0(r_i^{max} + 1) - \bar{\theta}_0}{\theta_0(r_i^{max} + 1) - \theta_0(1)}\right), \tag{1}$$

where $\bar{\theta}_0 = \frac{1}{r_i^{max}} \int_1^{r_i^{max}+1} \theta_0(x)dx$ is the average value of θ_0 between 1 and $r_i^{max} + 1$.

From this equation, it is easy to see that $r_i^{mle} \leq \frac{r_i^{max}}{2}$ when θ_0 is concave, as is the case for the Poisson, Gaussian, and exponential distributions, with $\theta_0(q) = \log(q)$, q, and $-\frac{1}{q}$ respectively. For the Gaussian, $r_i^{mle} = \frac{r_i^{max}}{2}$ since θ_0 is linear, while $r_i^{mle} < \frac{r_i^{max}}{2}$ for the Poisson and exponential.

Further, the assumption of an expectation-based scan statistic in the separable exponential family (A2) implies that the score function $F(S)$ satisfies the linear-time subset scanning property (Neill, 2012) with priority function $G(v_i) = \frac{T(x_i)}{\mu_i}$. This means that the highest-scoring unconstrained subset $S_j^* = \arg\max_S F(S)$ can be found by evaluating the score of only $|V|$ of the $2^{|V|}$ subsets of nodes, that is, $S_j^* = \{v_{(1)}, v_{(2)}, \ldots, v_{(k)}\}$ for some k between 1 and $|V|$, where $v_{(i)}$ represents the ith highest-priority node.

Given the set of all nodes $\{v_{(1)}, v_{(2)}, \ldots, v_{(|V|)}\}$ sorted by priority, we note that the assumption of a 1-strong signal implies that the true affected subset $S_j^T = \{v_{(1)}, v_{(2)}, \ldots, v_{(t)}\}$, where t is the cardinality of S_j^T. Thus, for Lemma 1 we need only to show that $|S_j^*| \geq t$, while for Lemma 2 we must show $|S_j^*| \leq t$. We can now prove:

Lemma 1. *For each training example D_j, there exists a constant $\alpha_j > 1$ such that, if the signal is α_j-homogeneous and 1-strong, then the highest scoring unconstrained subset $S_j^* \supseteq S_j^T$. We note that α_j is a function of $r_{\max}^{\mathit{aff},j}$, and $\alpha_j \geq 2$ for the Poisson, Gaussian, and exponential distributions.*

Proof. Let $\alpha_j = \frac{r_{\max}^{\text{aff},j}}{f(r_{\max}^{\text{aff},j})}$, where $f(r_i^{max}) = r_i^{mle}$ is the function defined in Equation (1) above. For distributions with concave $\theta_0(q)$, such as the Poisson, Gaussian, and exponential, we know that $f(r) \leq \frac{r}{2}$, and thus $\alpha_j \geq 2$. Now, the assumption of α_j-homogeneity implies $\frac{r_{\max}^{\text{aff},j}}{r_{\min}^{\text{aff},j}} < \frac{r_{\max}^{\text{aff},j}}{f(r_{\max}^{\text{aff},j})}$, $r_{\min}^{\text{aff},j} > f(r_{\max}^{\text{aff},j})$, and since $f(r)$ is an increasing and therefore invertible function, $f^{-1}(r_{\min}^{\text{aff},j}) > r_{\max}^{\text{aff},j}$.

Now we note that $r_{\min}^{\text{aff},j}$ is the observed excess risk $\frac{T(x_i)}{\mu_i} - 1$ for the lowest-priority affected node $v_{(t)}$, where t is the cardinality of S_j^T, while $r_{\max}^{\text{aff},j}$ is the observed excess risk for the highest-priority affected node $v_{(1)}$. Moreover, the contribution of node $v_{(t)}$ to the log-likelihood ratio statistic, $\lambda_t(q)$, will be positive for all $q < 1 + f^{-1}(r_{\min}^{\text{aff},j})$, and we know that the maximum likelihood estimate of q for any subset of nodes $\{v_{(1)}, v_{(2)}, \ldots, v_{(k)}\}$ will be at most $q = 1 + r_{\max}^{\text{aff},j} < 1 + f^{-1}(r_{\min}^{\text{aff},j})$. Thus node $v_{(t)}$ will make a positive contribution to the log-likelihood ratio and will be included in S_j^*, as will nodes $v_{(1)} \ldots v_{(t-1)}$. Hence $|S_j^*| \geq t$, and $S_j^* \supseteq S_j^T$. □

Lemma 2. *For each training example D_j, there exists a constant $\beta_j > 1$ such that, if the signal is $\frac{\beta_j}{\eta_j}$-strong, then the highest scoring unconstrained subset $S_j^* \subseteq S_j^T$. We note that β_j is a function of $r_{\max}^{\mathit{unaff},j}$, and $\beta_j \leq 2$ for the Gaussian distribution.*

Proof. Let $\beta_j = \frac{f^{-1}(r_{\max}^{\text{unaff},j})}{r_{\max}^{\text{unaff},j}}$, where $f^{-1}(r_i^{mle}) = r_i^{max}$ is the inverse of the function defined in Equation (1) above. For distributions with convex $\theta_0(q)$, such as the Gaussian, we know that

$f^{-1}(r) \leq 2r$, and thus $\beta_j \leq 2$. Now, the assumption that the signal is $\frac{\beta_j}{\eta_j}$-strong, where $\eta_j = \frac{\sum_{v_i \in S_j^T} \mu_i}{\sum_{v_i} \mu_i}$, implies $\frac{r_{\text{min}}^{\text{aff},j}}{r_{\text{max}}^{\text{unaff},j}} > \frac{f^{-1}(r_{\text{max}}^{\text{unaff},j})}{\eta_j r_{\text{max}}^{\text{unaff},j}}$ and thus $\left(\frac{\sum_{v_i \in S_j^T} \mu_i}{\sum_{v_i} \mu_i}\right) r_{\text{min}}^{\text{aff},j} > f^{-1}(r_{\text{max}}^{\text{unaff},j})$.

Now we note that $r_{\text{min}}^{\text{aff},j}$ is the observed excess risk $g_{ij} = \frac{T(x_i)}{\mu_i} - 1$ for the lowest-priority affected node $v_{(t)}$, and $r_{\text{max}}^{\text{unaff},j}$ is the observed excess risk for the highest-priority unaffected node $v_{(t+1)}$, where t is the cardinality of S_j^T. Moreover, the contribution of node $v_{(t+1)}$ to the log-likelihood ratio statistic, $\lambda_{t+1}(q)$, will be negative for all $q > 1 + f^{-1}(r_{\text{max}}^{\text{unaff},j})$. Finally, we know that the maximum likelihood estimate of q for any $\{v_{(1)}, v_{(2)}, \ldots, v_{(k)}\}$ will be at least $q = \frac{\sum_{v_i} T(x_i)}{\sum_{v_i} \mu_i} = 1 + r$, where $r = \frac{\sum_{v_i} g_{ij} \mu_i}{\sum_{v_i} \mu_i} = \frac{\sum_{v_i \in S_j^T} g_{ij} \mu_i + \sum_{v_i \notin S_j^T} g_{ij} \mu_i}{\sum_{v_i} \mu_i} > \frac{\sum_{v_i \in S_j^T} r_{\text{min}}^{\text{aff},j} \mu_i}{\sum_{v_i} \mu_i} > f^{-1}(r_{\text{max}}^{\text{unaff},j})$, where the key step is to lower bound each g_{ij} by $r_{\text{min}}^{\text{aff},j}$ for $v_i \in S_j^T$ and by 0 for $v_i \notin S_j^T$ respectively. Thus node $v_{(t+1)}$ will make a negative contribution to the log-likelihood ratio and will be excluded from S_j^*, as will nodes $v_{(t+2)} \ldots v_{(|V|)}$. Hence $|S_j^*| \leq t$, and $S_j^* \subseteq S_j^T$. □

References

Bailey, N. T. J. (1975). The mathematical theory of infectious diseases and its applications. *Hafner Press*.

Getoor, L., Friedman, N., Koller, D., and Taskar, B. (2003). Learning probabilistic models of link structure. *J. Mach. Learn. Res.*, **3**, 679–707.

Gomez-Rodriguez, M., Leskovec, J., and Krause, A. (2010). Inferring networks of diffusion and influence. In *Proc. 16th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 1019–1028.

Gomez-Rodriguez, M. G. and Schölkopf, B. (2012). Submodular inference of diffusion networks from multiple trees. In *ICML '12: Proceedings of the 29th International Conference on Machine Learning*, pages 489–496.

Hogan, W. R., Cooper, G. F., Wallstrom, G. L., Wagner, M. M., and Depinay, J. M. (2007). The Bayesian aerosol release detector: an algorithm for detecting and characterizing outbreaks caused by atmospheric release of Bacillus anthracis. *Stat. Med.*, **26**, 5225–52.

Kulldorff, M. (1997). A spatial scan statistic. *Communications in Statistics: Theory and Methods*, **26**(6), 1481–1496.

Myers, S. and Leskovec, J. (2010). On the convexity of latent social network inference. In *Advances in Neural Information Processing Systems 23*, pages 1741–1749.

Naus, J. I. (1965). The distribution of the size of the maximum cluster of points on the line. *Journal of the American Statistical Association*, **60**, 532–538.

Neill, D. B. (2012). Fast subset scan for spatial pattern detection. *Journal of the Royal Statistical Society (Series B: Statistical Methodology)*, **74**(2), 337–360.

Neill, D. B. and Moore, A. W. (2004). Rapid detection of significant spatial clusters. In *Proc. 10th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 256–265.

Neill, D. B., Moore, A. W., Sabhnani, M. R., and Daniel, K. (2005). Detection of emerging space-time clusters. In *Proc. 11th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 218–227.

Patil, G. P. and Taillie, C. (2004). Upper level set scan statistic for detecting arbitrarily shaped hotspots. *Envir. Ecol. Stat.*, **11**, 183–197.

Speakman, S., Somanchi, S., McFowland-III, E., and Neill, D. B. (2015a). Penalized fast subset scanning. *Journal of Computational and Graphical Statistics*, **(in press)**.

Speakman, S., McFowland-III, E., and Neill, D. B. (2015b). Scalable detection of anomalous patterns with connectivity constraints. *Journal of Computational and Graphical Statistics*, **(in press)**.

Tango, T. and Takahashi, K. (2005). A flexibly shaped spatial scan statistic for detecting clusters. *International Journal of Health Geographics*, **4**, 11.

Taskar, B., Wong, M.-F., Abbeel, P., and Koller, D. (2004). Link prediction in relational data. In *Advances in Neural Information Processing Systems 16*, pages 659–666.

Vert, J.-P. and Yamanishi, Y. (2005). Supervised graph inference. In *Advances in Neural Information Processing Systems 17*, pages 1433–1440.