Editor: **Daniel B. Neill,** H.J. Heinz III College, Carnegie Mellon University, neill@cs.cmu.edu

Graph Structure Learning from Unlabeled Data for Early Outbreak Detection

Sriram Somanchi, *University of Notre Dame*
Daniel B. Neill, *Carnegie Mellon University*

Event detection in massive datasets 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. 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 in which the recently observed counts are significantly higher than expected. Assuming that the graph structure is known, we can use various graph-based event detection methods to detect anomalous subgraphs.[1–3] A standard approach is to maximize a log-likelihood ratio statistic $F(S) = \log\left(\Pr\left(Data|H_1(S)\right)\right)/\Pr\left(Data|H_0\right)\right)$ over connected subgraphs S. For example, we can compute the expectation-based Poisson scan statistic,[4] which assumes Poisson-distributed case counts and a uniform multiplicative increase over the affected subgraph, as $F(S) = \left(C\log\left(C/B\right) + B - C\right)1\{C > B\}$, in which the observed and expected counts are aggregated over subgraph S and denoted as C and B, respectively. Maximizing $F(S)$ over connected subgraphs is computationally challenging, but the GraphScan algorithm[3] can optimize $F(S)$ efficiently and exactly, scaling to graphs an order of magnitude larger than the previously proposed FlexScan approach[2] while outperforming heuristic approaches such as upper-level sets.[1]

In many cases, however, the network structure is unknown. For example, the spread of disease may be influenced by latent commuting patterns. Assuming an incorrect graph structure can result in less timely and less accurate event detection, because the affected area may be disconnected and therefore may not be identified as an anomalous subgraph. In such cases, learning the correct graph structure has the potential to dramatically improve detection performance. 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 learn an underlying graph structure using labeled training data.[5–7] However, in many cases, labeled data is unavailable: for example, public health officials might be aware that an outbreak has occurred but might not know precisely which areas were affected and when. Hence, we focus on learning graph structure from unlabeled data, in which the affected subset of nodes for each training example is not given, and we observe only the observed and expected counts at each node.

Graph Learning Framework

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 $\boldsymbol{D}$. 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 are connected in the true (unknown) underlying graph structure G_T. 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 unobserved set of affected nodes S_j^T that is a connected subgraph of G_T. Unaffected nodes $v_i \notin S_j^T$ are assumed to have counts x_i drawn from some distribution with mean μ_i, whereas affected nodes $v_i \in S_j^T$ are assumed to have higher counts. Given these training examples, we have three main goals:

1541-1672/17/$33.00 © 2017 IEEE
Published by the IEEE Computer Society

- Accurately estimate the true underlying graph structure G_T.
- Given a separate set of test examples $\{D_1, \ldots, D_J\}$ drawn from $\boldsymbol{D}$, identify the affected subgraphs S_j^T. Accuracy of detection is measured by the average overlap coefficient between the true and identified subgraphs.
- Distinguish test examples drawn from $\boldsymbol{D}$ from examples with no affected subgraph ($S_j^T = \phi$). Detection power is measured by the true positive rate for a fixed false-positive rate.

A key insight of our graph learning framework is to evaluate the quality of each graph structure G_m, with m edges, 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[8] 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 efficiently (in linear rather than exponential time) because expectation-based scan statistics satisfy the linear-time subset scanning property.[8] We can use GraphScan to compute 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 $\overline{F}_{norm}(G_m) = (1/J) \Sigma_{j=1,\ldots,J(Fmj/Fj)}$ averaged over all J training examples as a measure of graph quality.

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 $\overline{F}_{norm}(G_m)$ will be close to 1. On the other hand, if graph G_m is missing essential connections, we expect the values of F_{mj} to be much lower than the corresponding F_j, and $\overline{F}_{norm}(G_m)$ 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, then $\overline{F}_{norm}(G_m) = 1$. Such underconstrained graphs result in reduced detection power. Thus, we wish to optimize the tradeoff between a higher mean normalized score and a 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.[9]

> To avoid removing potentially important edges, we use correlation to break ties.

Learning Graph Structure Algorithm

Considering the mean normalized score $\overline{F}_{norm}(G_m) = (1/J) \Sigma_{j=1,\ldots,J} (F_{mj}/F_j)$ as a measure of graph quality, we can search for the graph G_m with the highest mean normalized score. However, it is computationally infeasible to search exhaustively over all $2^{|V|(|V|-1)/2}$ graphs. Even computing the mean normalized score of a single graph G_m could require a substantial amount of computation time, because it requires calling a graph-based event detection method such as GraphScan to find the highest-scoring connected subgraph for each training example D_j. We refer to this call as BestSubgraph(G_m, D_j) for a given graph structure G_m and training example D_j. Here, we instantiate BestSubgraph using the GraphScan algorithm[3] (for a comparison of other alternatives, see our previous work[9]).

Thus, we propose Learning Graph Structure (LGS), a greedy framework for efficiently learning graph structure. LGS starts with the complete graph on N nodes and sequentially removes edges until no edges remain (see Figure 1). 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, which we refer to as BestEdge(G_m, D). 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 earlier. The idea is to remove unnecessary edges while preserving essential connections that 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, because $O(N^2)$ 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 $G_0, \ldots, G_M$. As described in our previous work,[9] 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, for a given graph G_m and example D_j, only $O(N)$ of the $O(N^2)$ candidate edge removals disconnects the highest-scoring subgraph S_j^*. For the remaining edges, we know $S_{m-1,j}^* = S_{mj}^*$ and do not need to call BestSubgraph.

To implement BestEdge(G_m, D), we note that greedily choosing the edge that

1. Compute correlation ρ_{ik} between each pair of nodes v_i and v_k, $i \neq k$, to be used in step 5.

2. Compute the highest-scoring unconstrained subset S_j^* and its score F_j for each example D_j using the fast subset scan.[8]

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 $\overline{F}_{norm}(G_m) = 1$.

4. **while** number of remaining edges $m > 0$ **do**

5. Choose edge e_{ik} = 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 $S_{m-1,j}^*$ = 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 $\overline{F}_{norm}(G_m) = \frac{1}{J}\sum_{j=1,\ldots,J}\frac{F_{m-1,j}}{F_j}$.

10. **end while**

11. Repeat steps 3 through 10 for R randomly generated sequence of edge removals to find the most significant graph G_m.

Figure 1. The Learning Graph Structure (LGS) framework.

maximizes the mean normalized score for each graph G_m could still be prohibitively expensive. Thus, we consider a faster (but approximate) "pseudo-greedy" approach that uses 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. Thus, 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 we choose the e_{ik} that disconnects the fewest subgraphs. The resulting graph G_{m-1} is expected to have a mean normalized score $\overline{F}_{norm}(G_{m-1})$, which is close to $\overline{F}_{norm}(G_m)$, 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} that requires the fewest calls to BestSubgraph for each graph G_m. This results in only $O(N \log N)$ calls to BestSubgraph instead of $O(N^3)$ for the exact greedy method.[9] To avoid removing potentially important edges, we use correlation to break ties: if two edge removals e_{ik} disconnect the same number of subgraphs, the edge with the lower correlation is removed. The intuition is that if two nodes are connected by an edge in the latent graph, then we expect both nodes to be simultaneously affected or unaffected by an event.

Experimental Setup

Our experiments focus on detection of simulated disease outbreaks injected into real-world Emergency Department (ED) data from 10 hospitals in Allegheny County, Pennsylvania.[9] The dataset consists of the number of ED admissions with respiratory symptoms for each of the $N = 97$ ZIP codes for each day from 1 January 2004 to 31 December 2005. Our simulations assume that the disease outbreak starts at a center location (chosen uniformly at random) and spreads over some underlying graph structure, increasing in size and severity over time. Outbreaks were assumed to be 14 days in length, and we assume that an affected node remains affected through the outbreak duration. Our previous work provides a detailed description of the outbreak simulation.[9]

We considered simulated outbreaks that 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, in which 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, whereas for the second set, we added additional edges between randomly chosen nodes to simulate travel patterns. As noted earlier, a contagious disease outbreak might be likely to propagate from one location to another that is not spatially adjacent, based on individuals' daily travel. We hypothesize that inferring these additional edges will lead to improved detection performance. For each set of experiments, we produced $J = 200$ training injects and an additional 200 test injects drawn from the same distribution.

We compared the performance of our learned graphs with that of the learned graphs from the MultiTree algorithm,[7] which was shown to outperform previously proposed graph structure learning algorithms such as NetInf[5] and ConNIe.[6] We used the publicly available implementation of the algorithm, and we assumed that MultiTree is given

the true labels of the affected subset of nodes for each training example. For each competing method, once a graph structure was learned, we used 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 ED data with no cases injected.

We evaluated detection performance using two metrics: average time to detection (assuming a false-positive rate of 1 per 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 false positive per 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 $F(S) > F_{\text{thresh}}$ and 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 $\left(|S^* \cap S^T| / |S^* \cup S^T|\right)$.

Detection performance is often improved by including a proximity constraint,[3] in which we perform separate searches over the local neighborhood of each of the N graph nodes, which comprises that node and its $k - 1$ nearest neighbors, and report the highest-scoring connected subgraph over all neighborhoods. We evaluate how performance varies as a function of neighborhood size, considering all $k = 5, 10, \ldots, 45$.

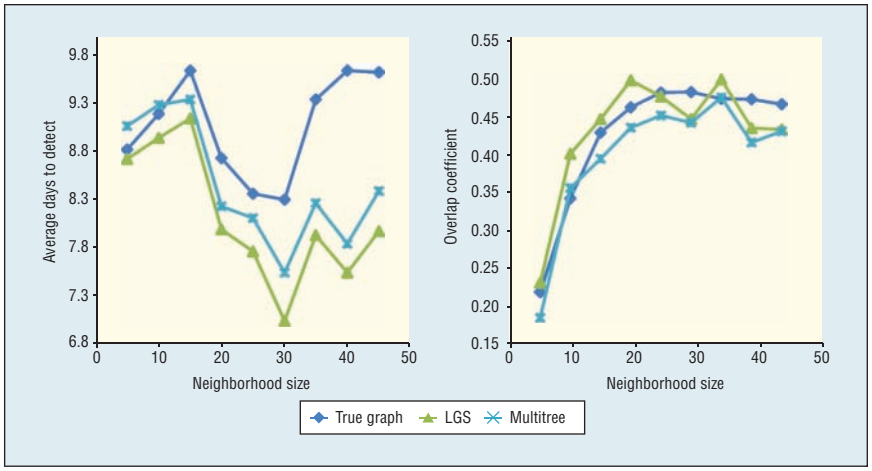

Figure 2. Comparison of detection performance of the true and learned graphs for injects based on ZIP code adjacency. The graphs learned by LGS achieved more timely detection than the true graph while maintaining a comparable spatial overlap coefficient.

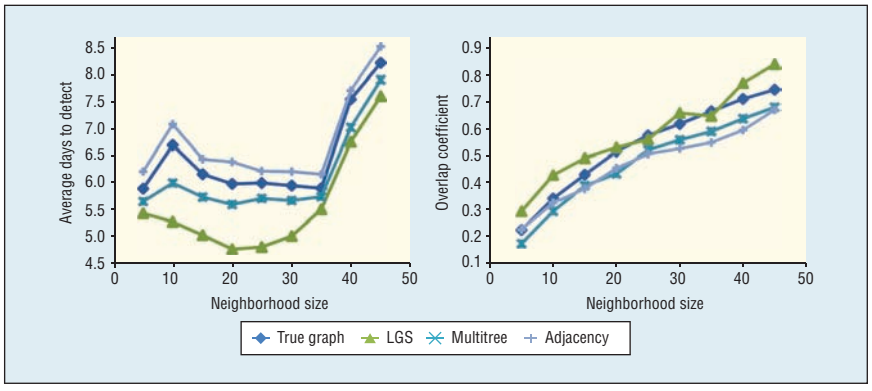

Figure 3. Comparison of detection performance of the true, learned, and adjacency graphs for injects based on adjacency with simulated travel patterns. The graphs learned by LGS achieved more timely detection than the true graph or assuming an incorrect graph (the adjacency graph in this case) while maintaining a comparable spatial overlap coefficient.

Experimental Results

We first evaluated the detection time and spatial accuracy of GraphScan, using the graphs learned by LGS and MultiTree, for simulated injects that spread based on the adjacency graph, as shown in Figure 2. The figure also shows GraphScan's performance given the true ZIP code adjacency graph. The graphs learned by LGS had a better spatial overlap coefficient and more timely detection as compared to graphs learned by MultiTree. Surprisingly, all of the learned graphs achieved more timely detection than the true graph: for the optimal neighborhood size of $k = 30$, LGS detected an average of 1.4 days faster than the true graph. This could be because the learned graphs, in addition to recovering most of the edges of the adjacency graph, also included additional edges to nearby but not spatially adjacent nodes (for example, neighbors of neighbors). These extra edges provided added flexibility and improved detection time when some nodes were more strongly affected than others, enabling the strongly affected nodes to be detected earlier

in the outbreak, before the entire affected subgraph was identified.

Next, we compared detection time and spatial accuracy using the graphs learned by LGS and MultiTree for simulated injects that spread based on the ZIP code adjacency graph, with additional random edges added to simulate travel patterns (see Figure 3). This figure also shows the detection performance given the true (adjacency plus travel) graph and the adjacency graph without travel patterns. Again, LGS has a better spatial overlap coefficient as compared to the original adjacency graph and MultiTree. Our learned graphs can detect outbreaks 0.8, 1.2, and 1.7 days earlier than MultiTree, the true graph, and the adjacency graph without travel patterns, respectively. This demonstrates that our methods can successfully learn the additional edges due to travel patterns, substantially improving detection performance.

As our associated technical report shows,[9] our results demonstrate that the graph structures learned by LGS 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 a similar spatial 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. We believe this is because the learning procedure is designed to capture not only the underlying graph structure, but the characteristics of the events that spread over that graph.

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. Our current approach does not rely on temporal information, using only the observed and expected counts at each node to compute correlations and to identify the highest scoring connected subgraph for each combination of graph structure and training example. To learn directed edges within our general structure learning framework, we plan to incorporate this temporal information by considering cross-correlations between each pair of nodes and by incorporating a new variant of GraphScan[10] that can detect dynamic patterns on graphs while enforcing constraints on temporal consistency. ■

Acknowledgments

This work was partially supported by NSF grant IIS-0953330.

References

1. G.P. Patil and C. Taillie, "Upper Level Set Scan Statistic for Detecting Arbitrarily Shaped Hotspots," *Environmental and Ecological Statistics*, vol. 11, no. 2, 2004, pp. 183–197.
2. T. Tango and K. Takahashi, "A Flexibly Shaped Spatial Scan Statistic for Detecting Clusters," *Int'l J. Health Geographics*, vol. 4, no. 11, 2005; doi:10.1186/1476-072X-4-11.
3. S. Speakman, E. McFowland III, and D.B. Neill, "Scalable Detection of Anomalous Patterns with Connectivity Constraints," *J. Computational and Graphical Statistics*, vol. 24, no. 4, 2015, pp. 1014–1033.
4. D.B. Neill et al., "Detection of Emerging Space-Time Clusters," *Proc. 11th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, 2005, pp. 218–227.
5. M. Gomez-Rodriguez, J. Leskovec, and A. Krause, "Inferring Networks of Diffusion and Influence," *Proc. 16th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, 2010, pp. 1019–1028.
6. S. Myers and J. Leskovec, "On the Convexity of Latent Social Network Inference," *Advances in Neural Information Processing Systems*, vol. 23, 2010, pp. 1741–1749.
7. M. Gomez-Rodriguez and B. Schölkopf, "Submodular Inference of Diffusion Networks from Multiple Trees," *Proc. 29th Int'l Conf. Machine Learning*, 2012, pp. 489–496.
8. D.B. Neill, "Fast Subset Scan for Spatial Pattern Detection," *J. Royal Statistical Soc. Series B: Statistical Methodology*, vol. 74, no. 2, 2012, pp. 337–360.
9. S. Somanchi and D.B. Neill, "Graph Structure Learning from Unlabeled Data for Event Detection," tech. report, Carnegie Mellon Univ., 2017; http://arxiv.org/abs/1701.01470.
10. S. Speakman, Y. Zhang, and D.B. Neill, "Dynamic Pattern Detection with Temporal Consistency and Connectivity Constraints," *Proc. 13th IEEE Int'l Conf. Data Mining*, 2013, pp. 697–706.

Sriram Somanchi is an assistant professor of business analytics in the Mendoza College of Business at the University of Notre Dame. Contact him at somanchi.1@nd.edu.

Daniel B. Neill is an associate professor of information systems and the director of the Event and Pattern Detection Laboratory at Carnegie Mellon University's Heinz College. Contact him at neill@cs.cmu.edu.