Fast Graph Structure Learning from Unlabeled Data for Event Detection

Sriram Somanchi
Daniel Neill
Event and Pattern Detection Lab
Carnegie Mellon University

This work was partially supported by NSF grants:
IIS-0916345, IIS-0911032, and IIS-0953330

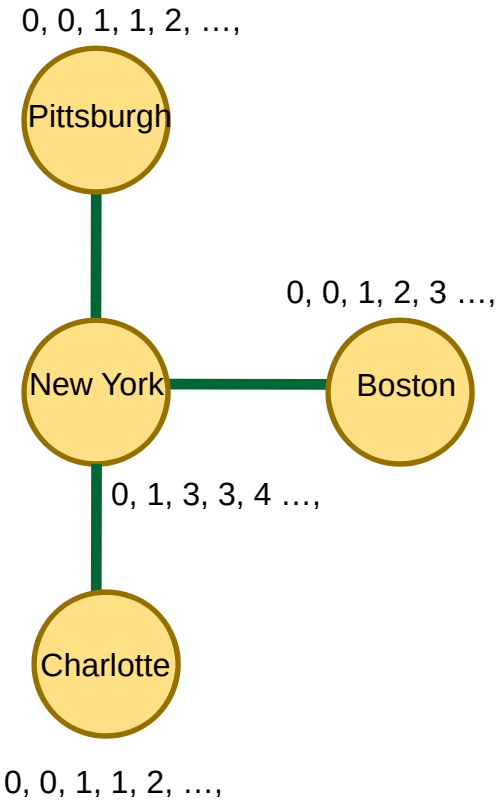
Agenda

- Introduction
- Problem Statement
  - Why is it important?
- Event Detection with known graph structure
- Learning graph structure for event detection
- Related Research
- Experimental Setup
- Experimental Results
- Conclusions and Future work

Introduction

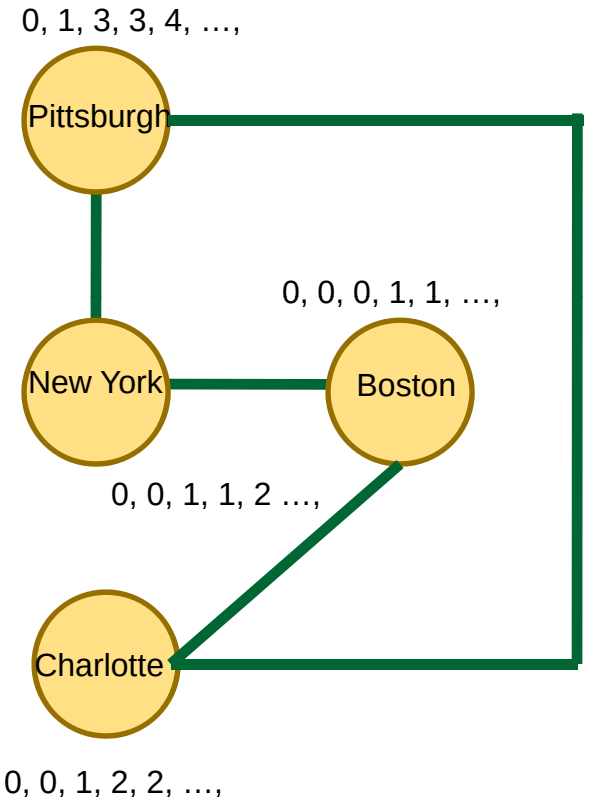
- Often times it is hard to directly observe the network over which an outbreak is spreading
  - Spread of disease outbreak due to person-to-person contact
  - Information spread on an implicit social media network
- However, we can observe the individual nodes getting effected, example
  - Increased counts of ED visits in a zip code
  - Information appearing in blogs for social media networks
- Applicable domains
  - Information diffusion
  - Disease surveillance

Problem Statement



- Given time series of data about each node can we infer the underlying graph network.

Problem Statement



- Given time series of data about each node can we infer the underlying graph network.
- Data is unlabeled: We only observe, counts at each node and we want to learn the graph.
- Primary goal is to learn a graph that will improve the detection power.

Why is it important?

- In case of disease surveillance, it helps authorities
  - Identify the network on which disease is spreading
    - Restrict further spreading
  - Find the anomalous subset given the network that is learnt, by running GraphScan
    - Assuming an incorrect graph can result in less timely and less accurate detection

Event Detection with Known Graph Structure

- If the graph is known, event detection problem is to detect connected subgraphs where recently observed counts are significantly higher than expected.
- This is achieved by maximizing the log-likelihood ratio statistic F(S) over all connected subgraphs S.
- Here we assume "Expectation-based Poisson" statistic, where counts are Poisson-distributed.

Event Detection with Known Graph Structure – Algorithms

- There are multiple algorithms for event detection on a known graph
  - FlexScan – exhaustive search
  - Upper level sets – heuristic based search
  - GraphScan – reduces the search space using Linear Time Subset Scan (LTSS) property
- If a scoring function F(S) satisfies LTSS, then unconstrained optimization over all subsets, can be efficiently computed in linear time.
- In order to score a graph structure we use GraphScan, as it finds optimal score for the graph.

Event Detection with Known Graph Structure – Algorithms

- FlexScan, There are multiple algorithms earches exhaustively over all connected subgraphs with in a fixed neighborhood size ‘n’ of each graph node.
- Upper level sets, is a heuristic based search which provides scalability, however, may fail to find the optimal subgraph.
- GraphScan also searches over all connected subgraphs, however prunes many subgraphs based on Linear Time Subset Scan property to find optimal subgraph and corresponding optimal score.

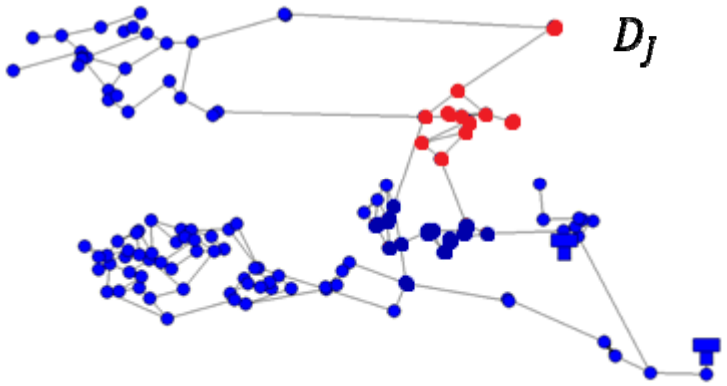
Event Detection with Known Graph Structure – GraphScan

- If a scoring function F(S) satisfies LTSS, then unconstrained optimization over all subsets, can be efficiently computed in linear time.
- GraphScan, builds on LTSS to speed up the search ruling out large number of subgraphs which have provably suboptimal scores.
- GraphScan, substantially improved the timeliness and accuracy of event detection as compared to Upper Level Sets.

Related Research

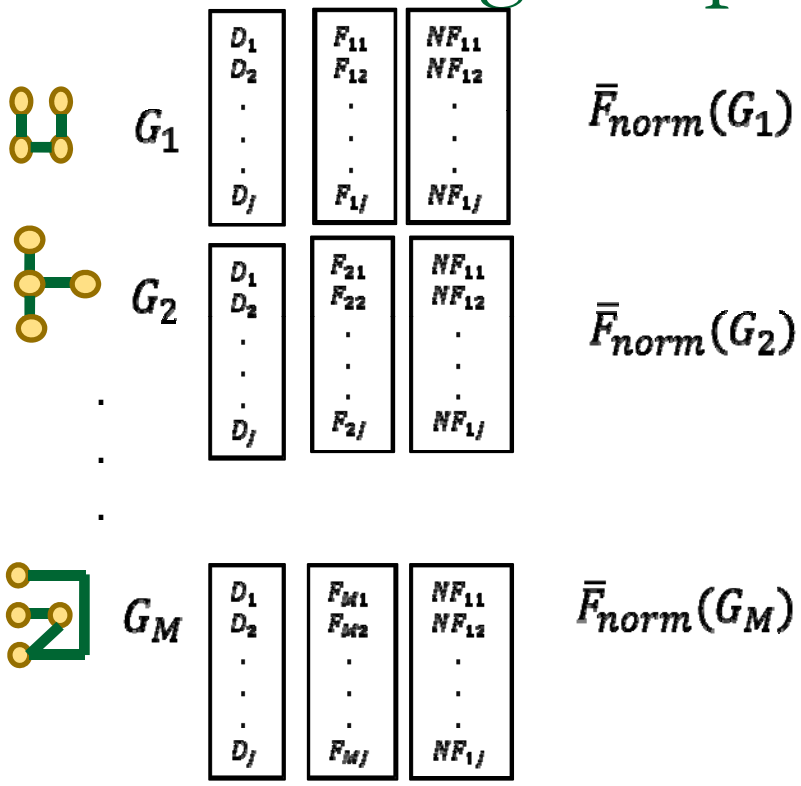
- There were two algorithms proposed to learn latent graphs in the context of social networks
- NetInf[1], forms an optimization problem to find the best graph, and shows it to be NP-hard. It further gives an approximation algorithm
- ConNIe[2], takes a maximum likelihood approach based on convex programming and further uses L_1 penalty to favor sparse graphs
- However, both of the algorithms assume labeled data, where affected subset of nodes at each time step is given

Learning Graph Structure for Detection – Data



- The graph structure is not always known and must be inferred from data.
- We have unlabeled training examples $D_1, D_2, \ldots, D_J$, where each D_j represents a different 'snapshot' of the data at a time when an event is assumed to be occurring in some connected subset of nodes.

Learning Graph Structure for Detection – Evaluating Graphs

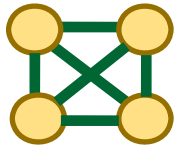


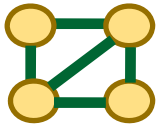
- Let us say we have set of graphs $G_1, \ldots, G_M$ and want to find the best graph G_m that represents the given training data.
- For each graph find the mean normalized score across training data
  - Score each graph for each training example
  - Normalize the score by dividing by unconstrained score computed using LTSS.
  - Take the mean across all training examples
- If a graph has low mean normalized score then it might be missing essential connections
- Though a graph with lot of edges has high score, it is less informative.

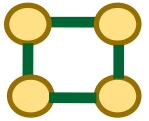
In order to score each graph we chose GraphScan algorithm, as it finds the optimal score for a given graph.

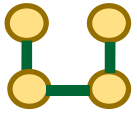
Algorithm – Fast Graph Structure Learning

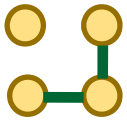
- All the possible graphs for a given set of N nodes is exponentially large, $M = 2^{\frac{N(N-1)}{2}}$, and makes it computationally intractable to search.
- In order to make it tractable, we propose a greedy approach.
- We start with a complete graph and sequentially remove edges until no edges remain, thus we have a tractable number $M = \frac{N(N-1)}{2}$.
- The idea is to remove unnecessary edges, while preserving essential connections.

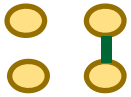
G_1 

G_2 

G_3 

G_4 

G_5 

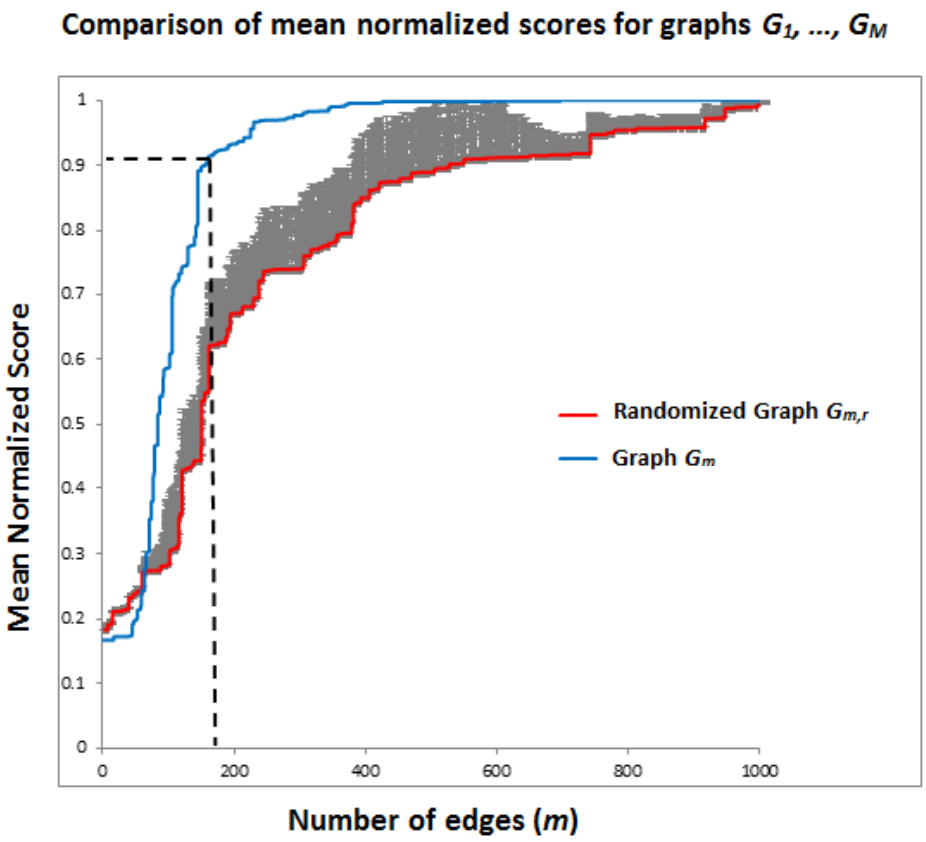
G_6 

We have to be careful in the order of our edge removal

Algorithm – Edge Selection Methods

- Ideally, we want to remove an edge from the graph G_m such that the resulting graph G_{m-1} preserved the highest mean normalized score. This is might be computationally infeasible.
- Hence we propose greedy method to choose an edge to remove, which disconnects the fewest subgraphs of G_m for each training example.
- In early stages there might be a lot of ties, and we resolve them based on correlation coefficient of the training data between two nodes. We prefer the edge with higher correlation. (Gr-Corr)

Algorithm – Finding Most Significant Graph



- We have normalized scoring distribution of $G_1, \ldots, G_M$.
- We are going to remove edges in a random order, multiple times and compute the mean normalized score of each graph.
- We propose the "most significant graph" that has the most anomalously high value of mean normalized score given its number of edges "m".

Algorithm – Computational Complexity Analysis

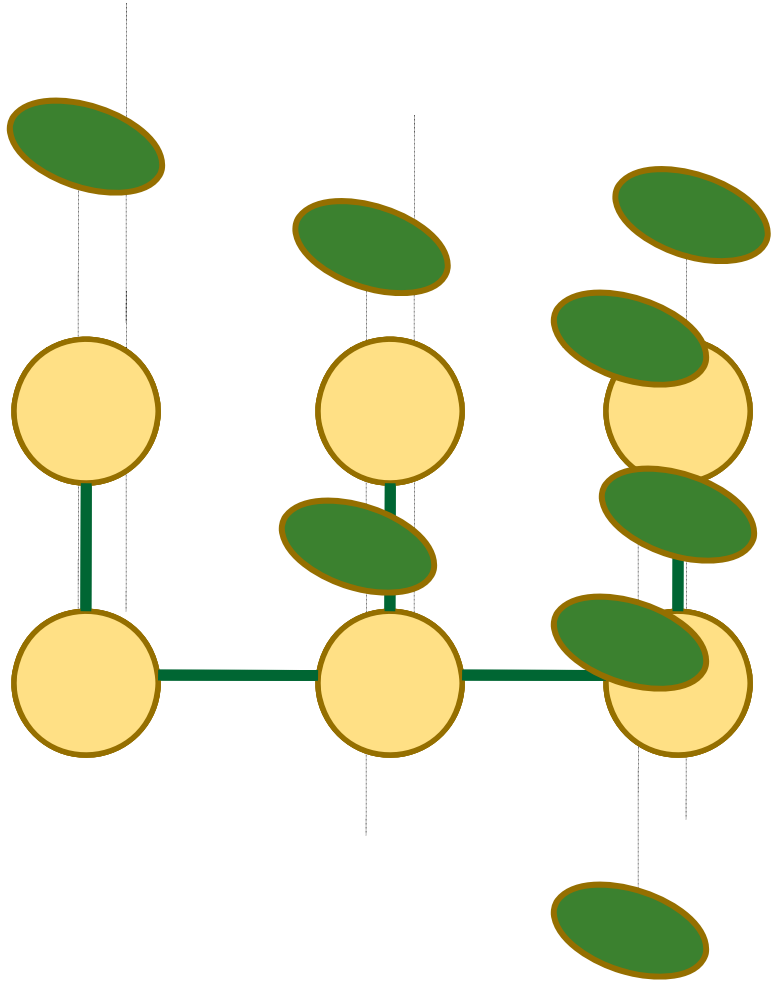
- The bottleneck in our computation analysis is number of times we need to score a graph.
- Here we use "GraphScan", which was shown empirically to take $O(1.2^N)$ time.
- We prove that expected number of runs of GraphScan for a given graphs $G_1, \ldots, G_M$, with J training examples is $O(JN \log N)$.
- Hence the total expected running time of our algorithm is $O(1.2^N JN \log N)$

Experimental Setup - Dataset

- Emergency Department Data
  - Visits with respiratory symptoms for each of 97 Alleghany zip codes.
  - We have data for each day from January 1, 2004 to December 31, 2005.
  - The resulting data set had a daily mean of 44.0 cases, and a standard deviation of 12.1 cases.
- We simulate disease outbreak into this real-world Emergency Department data.

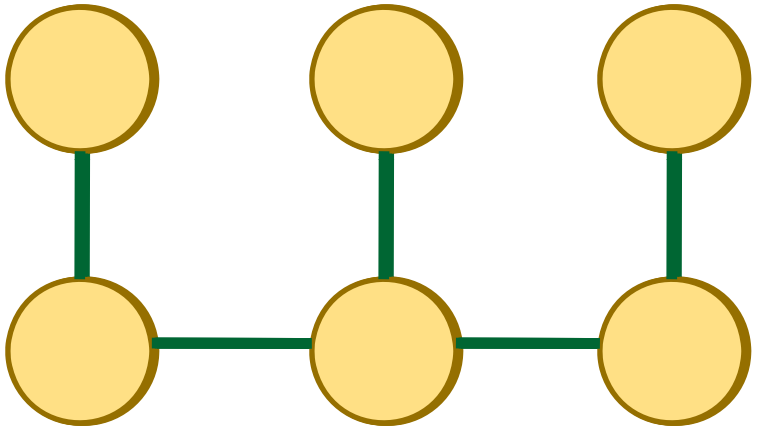
Experimental Setup – Outbreak Simulations

- We assume that our outbreaks follow Susceptible-Infected contagion model.
- Outbreaks spread over some underlying graph structure increasing in size and severity over time.
- We simulate multiple outbreaks each time starting at a center chosen uniformly at random.
- We generated $J = 200$ injects as each of training and test datasets.
- Each inject is about 14 days in duration.



Experimental Setup – Outbreak Simulations

- We assume that our outbreaks follow Susceptible-Infected contagion model.
- Outbreaks spread over some underlying graph structure increasing in size and severity over time.
- We simulate multiple outbreaks each time starting at a center chosen uniformly at random.
- We generated $J = 200$ injects as each of training and test datasets.
- Each inject is 14 days in duration.



Experimental Setup – Outbreak Simulations

- We simulate outbreaks spreading over the following kinds of graphs
  - Adjacency graph + Random Edges
    - Random edges can simulate travel patterns.
  - Preferential Attachment graph

Experimental Results – True vs. Learned graph

Experiment	Edges (true)	Precision		Recall	
		GrCorr	Corr	GrCorr	Corr
Adjacency	216	0.60	0.62	0.89	0.86
Adjacency+Travel	280	0.70	0.71	0.86	0.83
Erdos-Renyi (avg)	(varies)	0.56	0.59	0.87	0.81
Pref. Attachment	374	0.93	0.91	0.88	0.81

- Recall is high which shows that we could retrieve a lot of edges from the original graph.
- Precision is not too low which shows that we have do not have too many irrelevant edges.

Experimental Results – True vs. Learned graph

Graph	Edges (true)	Precision	Recall
Adjacency + Random	280	0.70	0.86
Pref. Attachment	374	0.93	0.81

- Recall is high which shows that we could retrieve a lot of edges from the original graph.
- Precision is not too low which shows that we have do not have too many irrelevant edges.

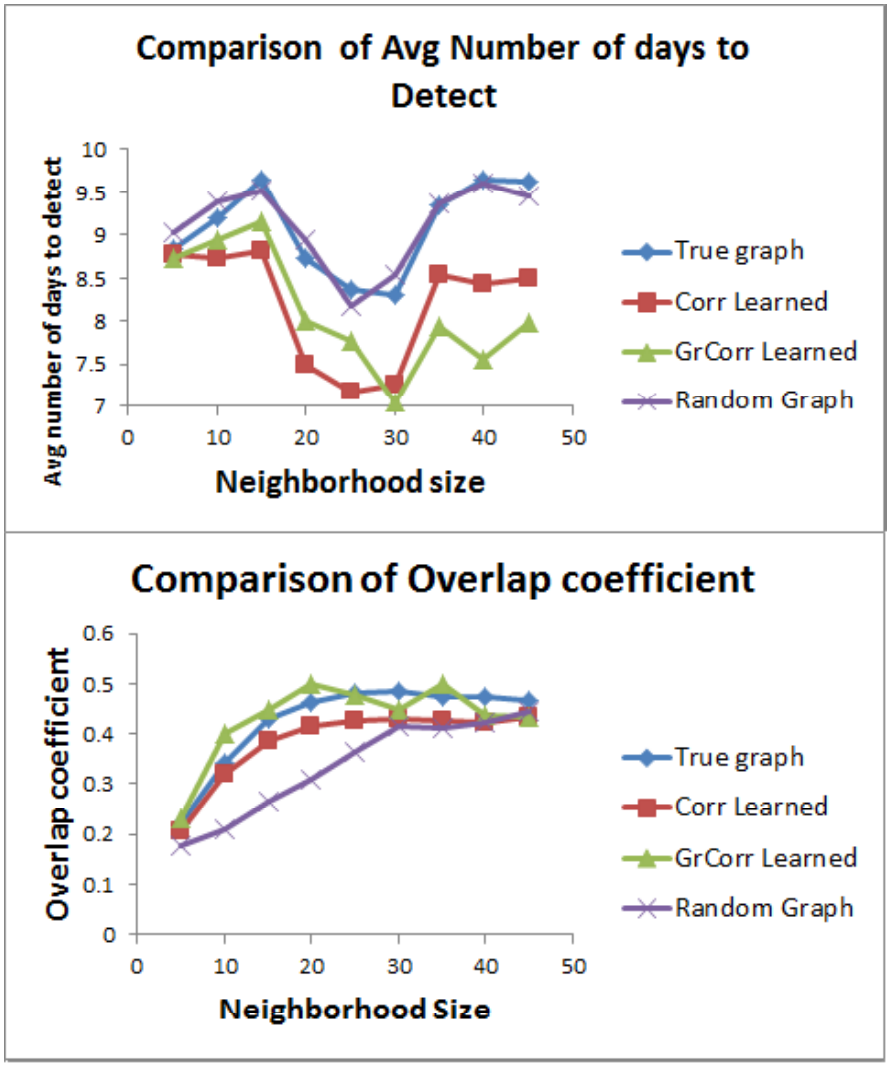
Experimental Results – Computation time

Experiment	GraphScan Runs		Run Time (minutes)	
	GrCorr	Corr	GrCorr	Corr
Adjacency	5527	5765	41	48
Adjacency+Travel	6985	7306	53	61
Spatial	5952	6266	39	45
Erdos-Renyi (avg)	6983	7467	93	104
Pref. Attachment	6982	7188	49	56

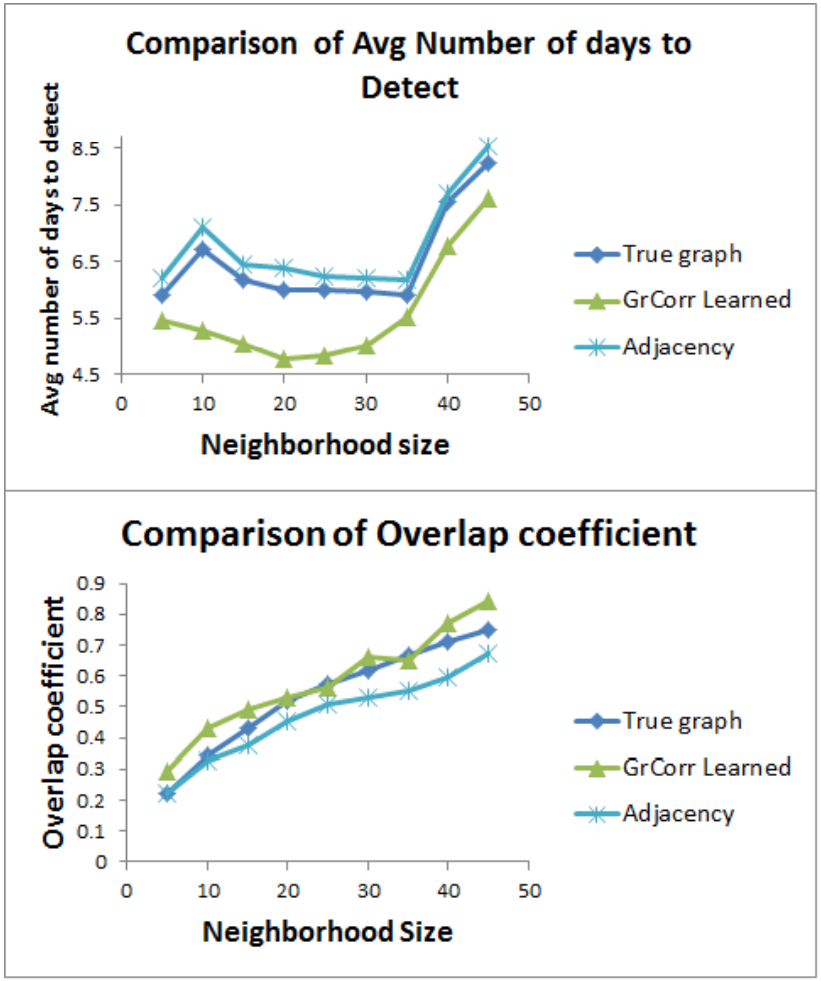
Experimental Results – Detection Performance

- We compare the graphs based on following parameters of event detection.
  - Number of days to detect (for a fixed fpr)
  - Overlap measure $\frac{|T \cap D|}{|T \cup D|}$
    - T- set of truly affected nodes, D- detected set of nodes returned by algorithm using a graph
- Also given a graph structure, we might be interested in searching only among "local neighborhoods" of size k.

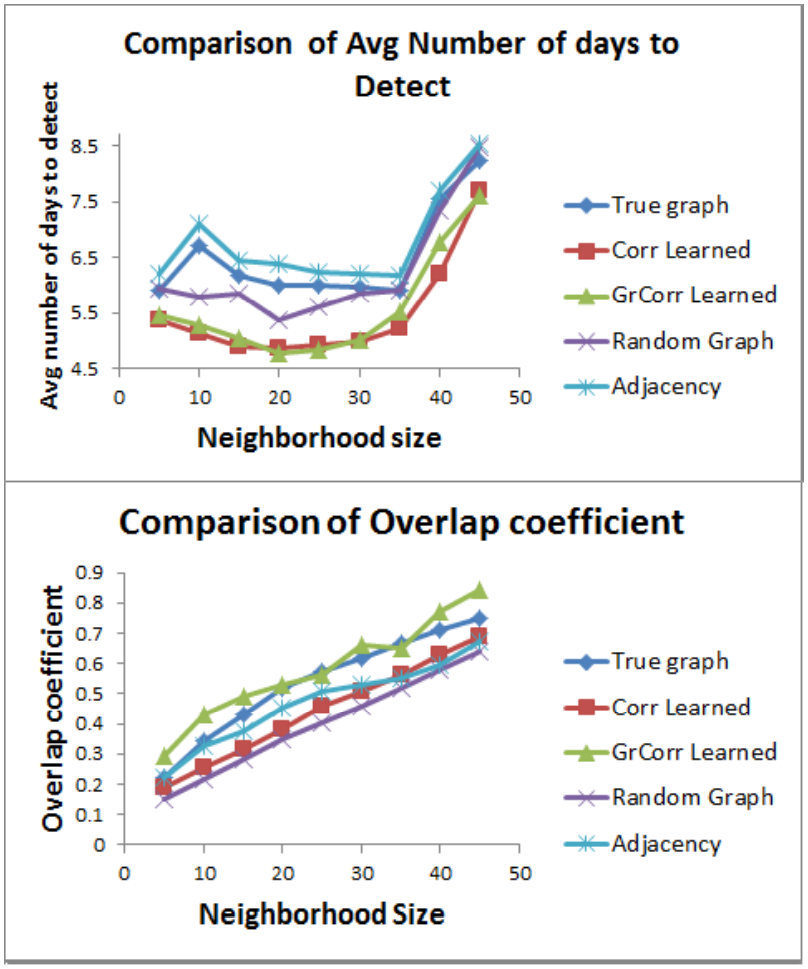
Performance – Adjacency Graph



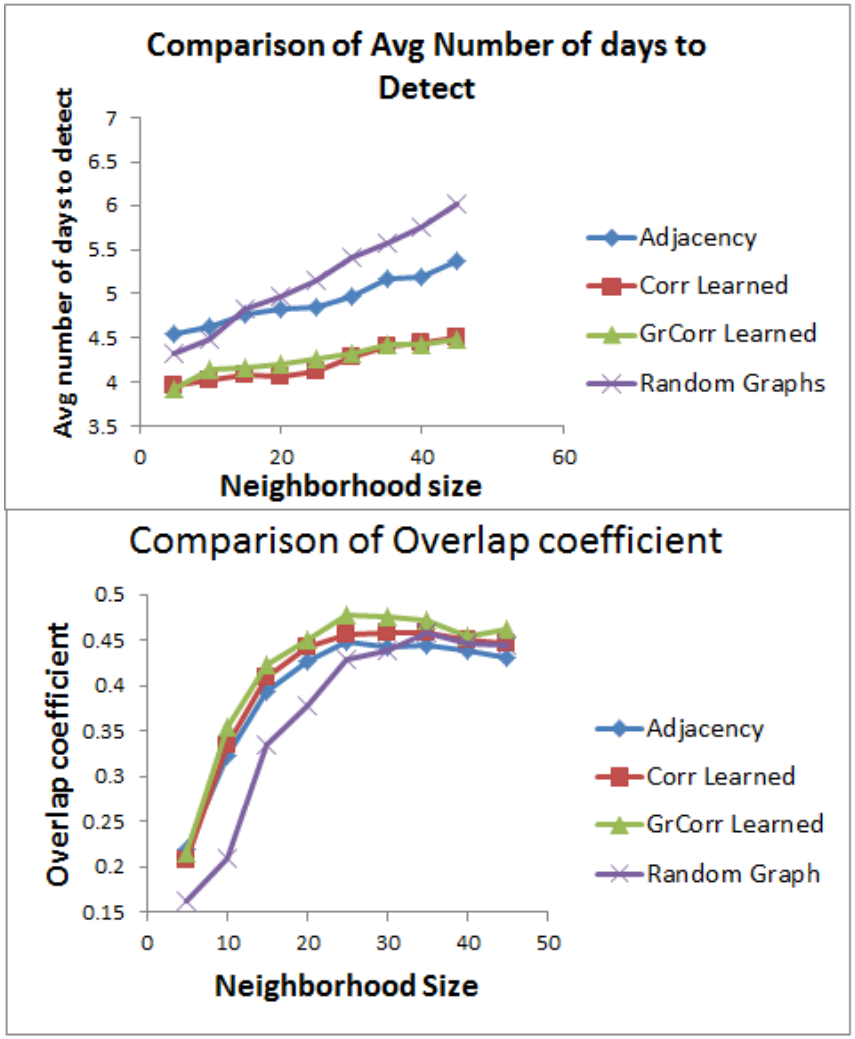
Performance – Adjacency + Random edges



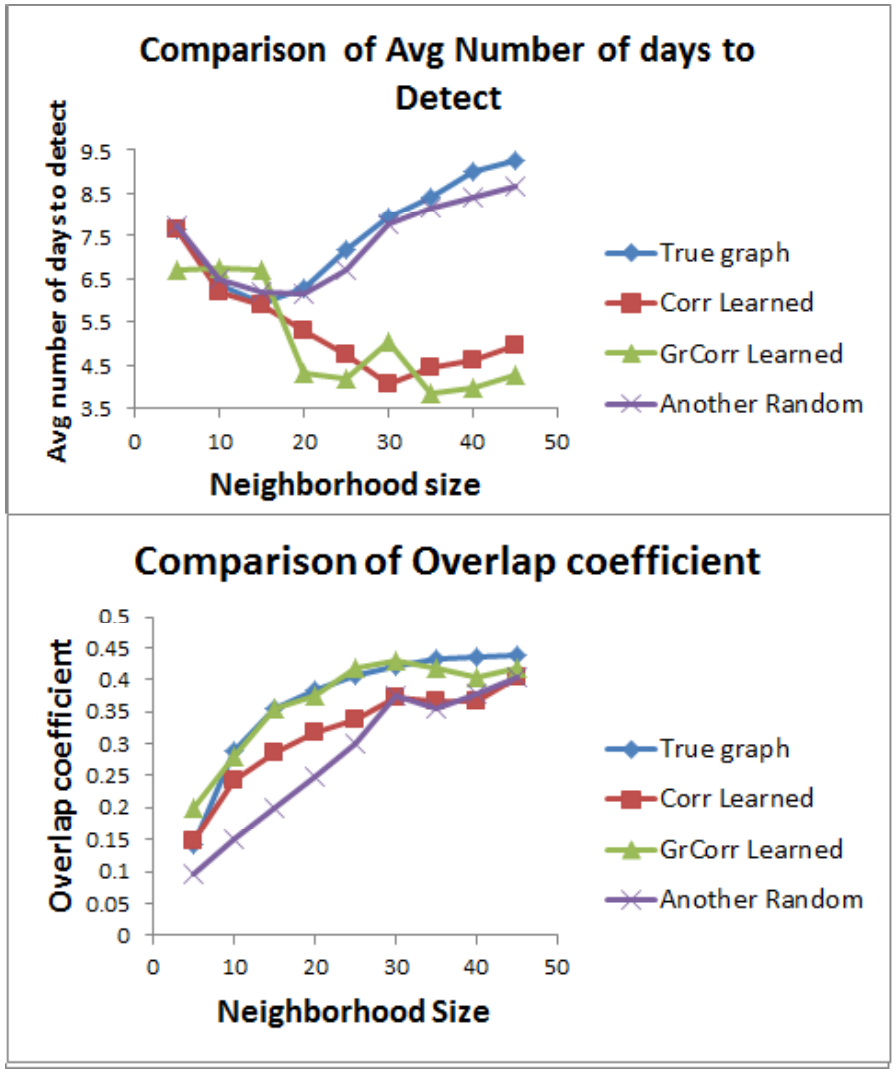
Performance – Adjacency + Random edges



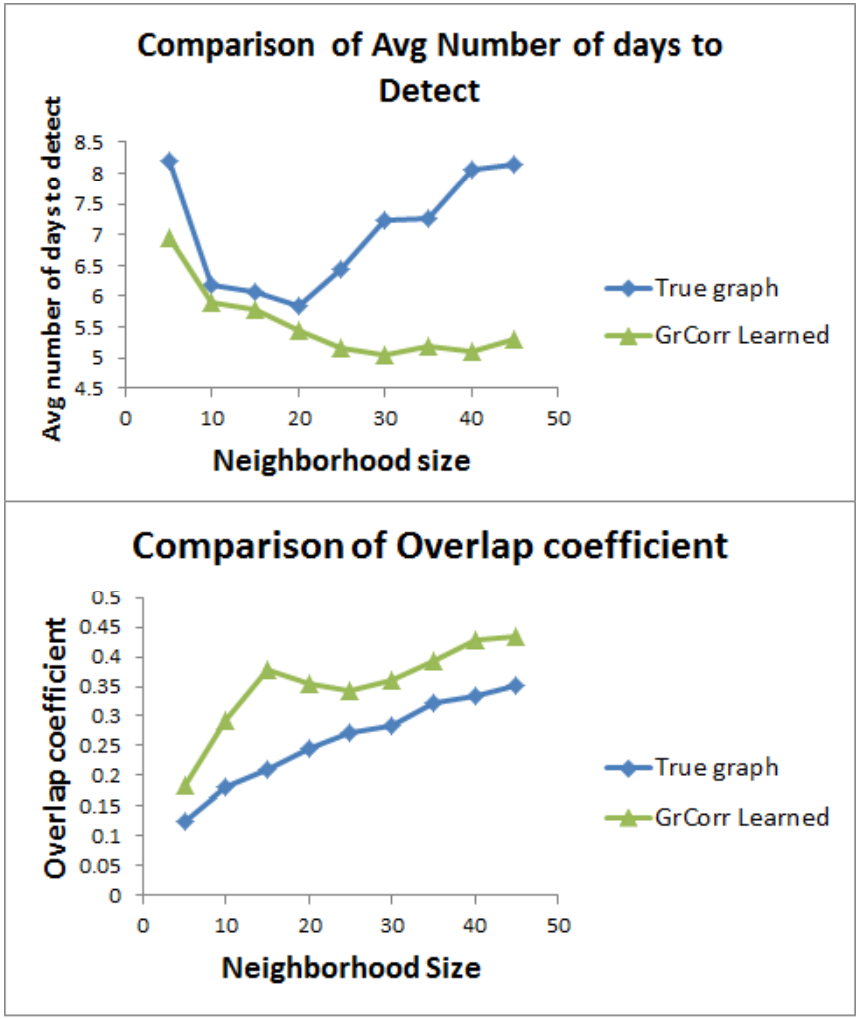
Performance – Spatial Spread



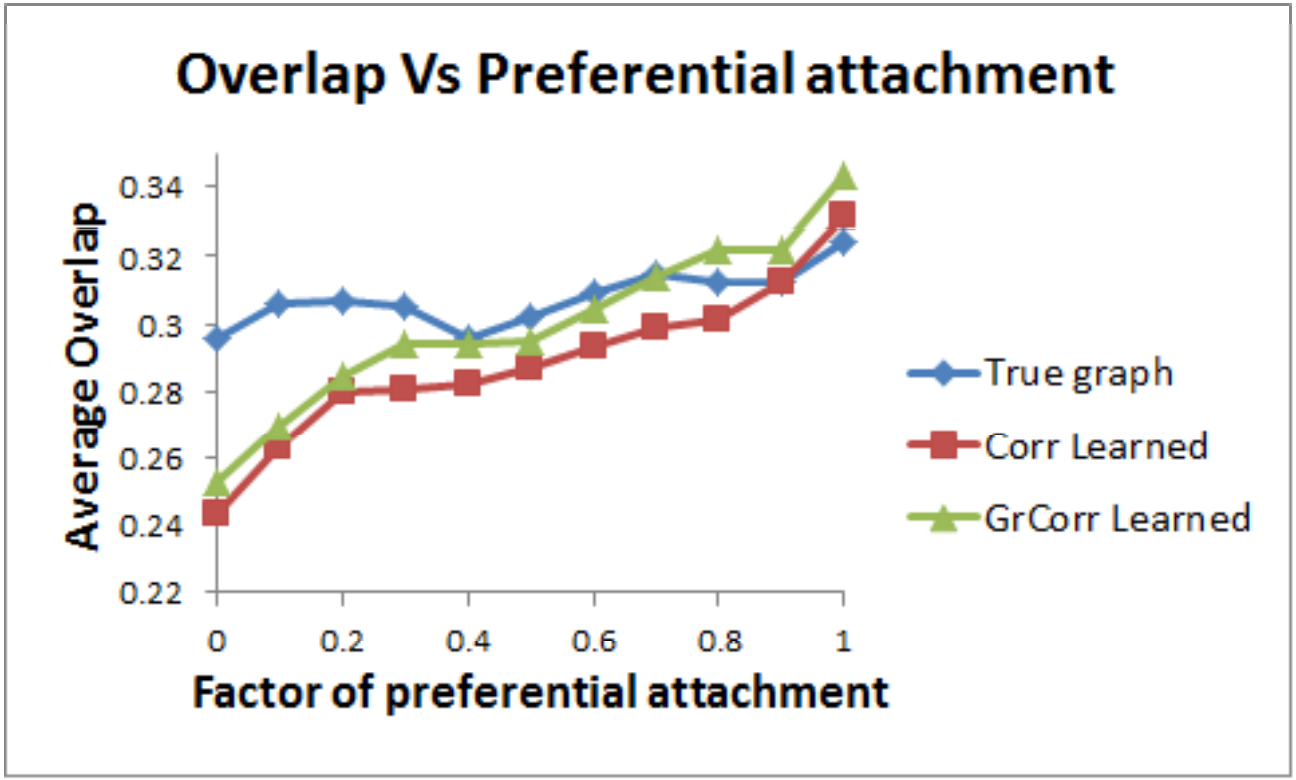
Performance – Erdos-Renyi random graph



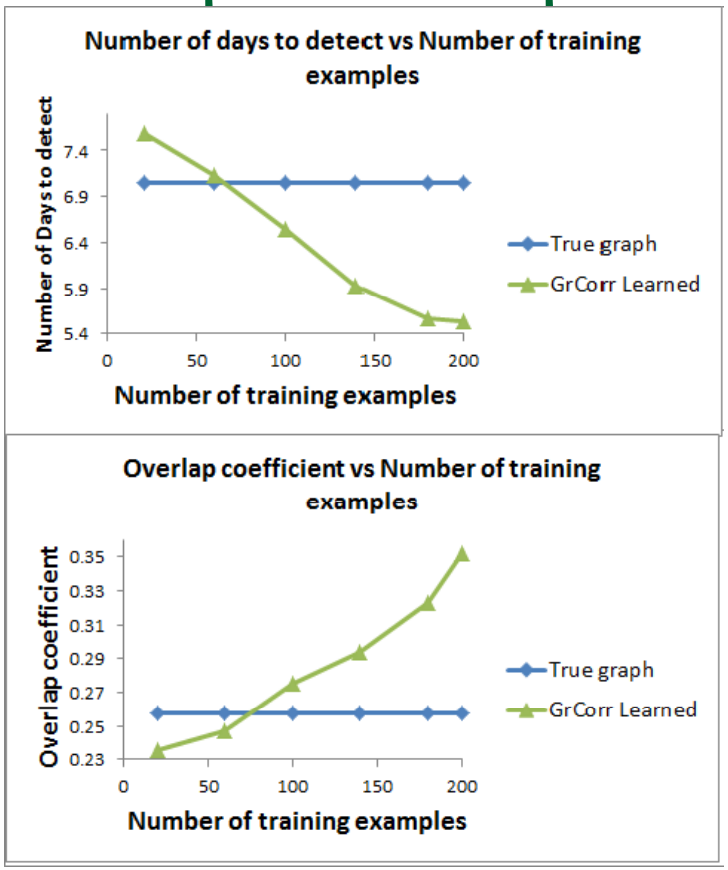
Performance – Preferential attachment graph



Performance – Continuum of Preferential attachment

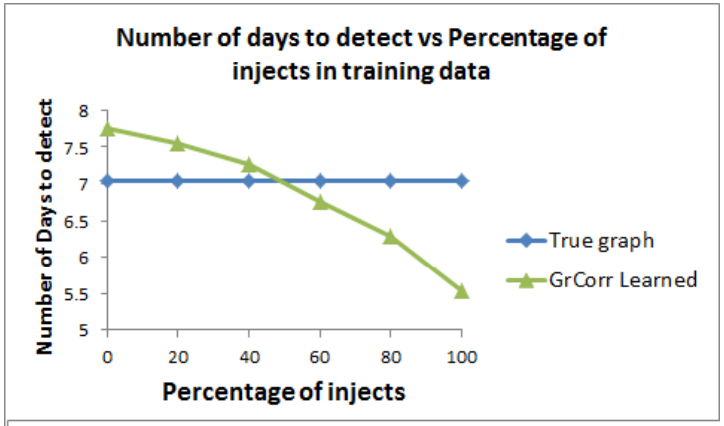


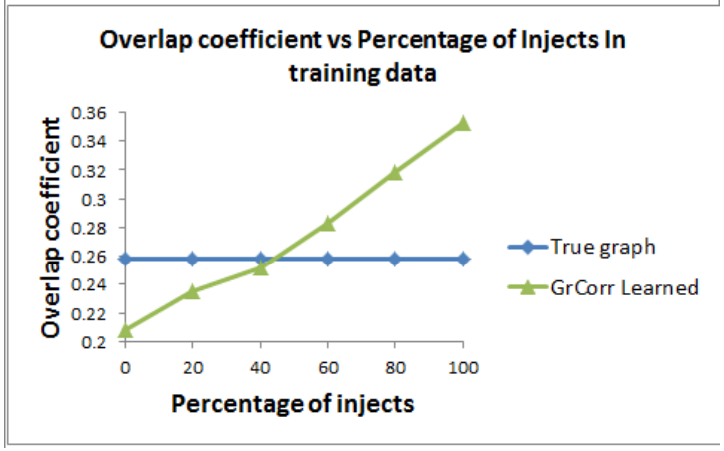
Robustness – Effect of number of training examples on performance



The results are for preferential attachment graphs

Robustness – Effect of percentage of injects in training data on performance





The results are for preferential attachment graphs

Conclusions

- We proposed a novel framework to learn graph structure from unlabeled data.
- Our method is based on comparing the most anomalous subsets with and without graph constraints.
- We showed from our experiments that we can accurately learn the graph structure and the learned graph structure has better performance for event detection.
- We showed that our learning algorithm is robust to noise in the training data.

Next steps

- In order to scale up our learning algorithms, we would like to experiment using fast and heuristic scoring functions for graphs like Upper Level Sets (ULS).
- Compare the performance of our methods with existing methods which require labelled data (NetInf, ConNIe).
- Theoretically or empirically bound the error due to greedy approach we have followed in our edge selection.
- Experiment with other datasets.

References

1. D. B. Neill. Fast subset scan for spatial pattern detection. Journal of the Royal Statistical Society (Series B: Statistical Methodology), 2011, to appear.
2. S. Speakman, E. McFowland III and D.B. Neill (2010) Scalable Detection of Anomalous Patterns with Connectivity Constraints. To Appear
3. L. Shi and V. P. Janeja (2009) Anomalous window discovery through scan statistics for linear intersecting paths (sslip). Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining
4. J. Leskovec, L. Backstrom, and J. Kleinberg (2009) Meme-tracking and the dynamics of the news cycle. Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining
5. M. Gomez-Rodriguez, J. Leskovec, and A. Krause (2010) Inferring net-works of diusion and inuence. KDD 10
6. S. A. Myers and J. Leskovec (2010) On the Convexity of Latent Social Network Inference. NIPS.

Thank Q?