Analyzing and Mitigating Phishing Outbreaks:

A Case Study on the 2016 DNC Email Network

Github: https://github.com/ppsekhar/CS224W

Andrew Zhao(andrewzh@), Priyanka Sekhar (psekhar@), Sameer Merchant (smerchan@)

1. Introduction

1.1 Motivation

In this project, we take a perspective of a system administrator seeking to secure an email network. Given a budget constraint in the form of number of detectors (cybersecurity resources) available, how do we secure our email network by maximizing likelihood of outbreak detection, minimizing detection time and reducing the population affected? This project aims to optimally detect phishing outbreaks in an empirical email network dataset that maps the communications of 2016 Democratic National Committee [4]. We use CELF as a benchmark for choosing “detector” nodes in our outbreak. While CELF is very effective in placing detector nodes, it can be computationally expensive and its runtime can be prohibitive when simulating large numbers of outbreaks in a dynamically changing email network. We investigate methods of speeding up CELF through use of a node metric called Social Score [3], to assist CELF with node selection. Our experimental results on the DNC email network show a 10x speed up in selecting detector nodes. Based on our results, we believe Social Score serves as an effective heuristic to speed up CELF despite expected decreases in quality of selection.

1.2 Dataset

We use a directed (unweighted multiple edge) network of emails in the 2016 Democratic National Committee email leak [4]. Nodes in the network correspond to persons in the dataset (e.g. Hillary Clinton, Debbie Wasserman Schultz, etc.). A directed edge in the dataset denotes that a person has sent an email to another person. The timestamp field is the Unix datetime at which the email was sent. Our dataset was aggregated and provided publicly by the Institute for Web Science and Technologies at the University of Koblenz-Landau. The data-set has already been engineered into a text file, with three columns for the source node, destination node, and timestamp.
Full set of statistics and graphs: http://konect.uni-koblenz.de/networks/dnc-temporalGraph
Total Vertices: 2,029 - Edges (with timestamps): 39,246

2. Related Work

2.1 CELF

Our project draws heavily on previous work in influence maximization and outbreak detection. “Maximizing the Spread of Influence through a Social Network” by Kempe et al. first specifies the greedy hill-climbing approximation algorithm to select a discrete set

of optimal detectors with provable bounds on its optimality [1]. “Cost-effective Outbreak Detection in Networks” by Leskovec et al. further improves on greedy-hill with the Cost-effective Lazy Forward (CELF) algorithm [2]. We take CELF to be the state-of-the-art benchmark for detector placement in our dataset, given it’s guarantee of 63% optimality, and compare its empirical runtime with the runtimes of our hybrid solutions. We implement this algorithm for our dataset and modify it using work highlighted in 2.2.

2.2 Social Score

We looked at the Social Score algorithm proposed in “Automated social hierarchy detection through email network analysis” by Rowe et al. [3]. The algorithm discovers social hierarchy through mining email network datasets. The social score algorithm combines a node’s structural attributes, e.g. degree centrality, hub and authority score, betweenness centrality, closeness centrality and dynamic behavioral attributes such as volume of emails exchanged with neighbors and average email response time. The algorithm exploits timestamp characteristics of email communications to judge strength of communication between nodes. It further uses weights to control the contribution of structural and behavioral attributes in computing the overall social score of a node. The inclusion of dynamic behavior attributes in computing social score makes it a suitable metric for selecting nodes as detectors. We use the social scores to optimize selection of detector nodes in DNC email network.

2.3 Outbreaks

Finally, Jin et al. highlight different methods of modeling online information spread through existing outbreak simulation models [5]. We draw inspiration from the SEIZ model described in this paper in modeling our own phishing spread. While Jin et. al find that marking individuals as “skeptics” realistically models the spread of information on Twitter, we modeled our outbreak using the paradigms of an SI model - in which nodes can be silently infected and neither recover nor become re-susceptible after recovery. Because we did not harden nodes and assumed every node was equally likely to be infected, we can simulate our outbreak accurately with less sophisticated models. Nevertheless, the work of Jin et al. served as a good example of how information spread can be modeled using traditional disease epidemic simulation techniques.

3. Model and Methods

3.1 Phishing Outbreak Simulations

3.1.1 Hyper-parameters

In our email network, we run phishing outbreaks with a susceptible-infected (SI) model where infected nodes do not recover. We use two hyper-parameters, which we pinpointed with a grid search in the milestone: 1) *proportion initially infected* to specify the number of initially compromised email accounts and 2) *probability of infection* to stochastically determine if an infected node would successfully compromise its neighbor through an email. Specifically, we ran simulations with two or ten “infected” nodes at t=0, selected arbitrarily from the graph, and we used 30% or 50% as probability of infection. The probability of infection parameters are based on the Verizon Data Breach Investigations Report, which reports that 30% of users would open a phishing email. We also include an even higher probability of infection because we believe that users would be even more likely to open a malicious email from another trusted internal email address. [6] Moreover, we ran our outbreaks in strict accordance with the timestamps scraped from the original emails. As our dataset included 32K emails, we follow the timestamps to run our outbreaks. Our phishing simulations thus exactly replicated the email transmission sequence of the 2016 DNC.

3.1.2 Metrics

We use the three metrics of outbreak detection discussed in class: probability of detecting an outbreak, population affected, and time to detection. We consider an outbreak to be detected and consequently stop a simulation when a single detector node has been infected. Otherwise, the outbreak continues until the 32K emails have been processed. Thus, in calculating the population affected, we take the list of currently infected nodes if we do detect an outbreak and otherwise sum all the infected nodes if we do not detect an outbreak. In calculating time to detection, we use simulation steps, which corresponds to how many emails have been sent since the first email in the dataset. When an outbreak has been detected, time to detection represents how many emails have been sent between the first email and infection of the detector. Otherwise, we use 32K to represent that all the emails have been transmitted without detection.

3.1.3 Live Edge Implementation

To implement our outbreaks, we invoke the principle of deferred decisions as outlined in Lecture 12 - Influence Maximization. For the hyper-parameters of initially infected nodes and probabilities of infection, we save 200,000 deterministic graphs with only live edges that fire successfully. In all the results shown below, we thus average our influence set sizes over this set or subsets of these saved graphs. This is important because we contrast our algorithms over the same set of outbreaks to directly compare their outbreak detection performance and runtimes.

3.2 Cost-Effective Lazy Forward Evaluation (CELF)

We implement the CELF algorithm discussed in Section 2. Our detectors have uniform cost, and we allocate a budget of 40-50 detectors for our simulations. For our three metrics, we seek to maximize the likelihood of detecting an outbreak, as well as minimize the population affected and time to detection. For the latter two metrics, previously framed as maximization objectives in Lecture 12, we reframe them as minimization functions by simply taking the most negative marginal gain. Submodularity is still preserved in these metrics because the marginals become less negative and converge to zero in the minimization framework. Thus, the results for population affected and time to detection display decreasing functions that are concave up.

3.3 Social Score

We implemented the Social Score algorithm defined by Rowe et al. [3]. We computed response time (t) for emails sent by a node using the difference in request and response email timestamps. We consider an email from a destination node to be a response email when a source node sends mail to the destination, and the subsequent email from destination node to source node is sent within 24hrs. Only emails with responses are considered in computing average response time. We compute a feature vector for each node that includes attributes like degree centrality, betweenness centrality, closeness centrality, authority and hub index, raw clique score $(R = \sum_i 2^{n_i - 1})$ where n_i is number nodes in i^{th} clique to which node belongs and weighted clique $(W = t * R,)$ where t is an average email response time for the node. Social Score for each node is computed as a normalized weighted combination of all feature metrics

$$w_x \cdot C_x = w_x \cdot 100 \cdot \left[\frac{x_i - Inf_x}{Sup_x - Inf_x} \right] for\ x^{th}\ feature\ of\ i^{th}\ node$$

$$S = \frac{\sum_{all\ x} w_x \cdot C_x}{\sum_{all\ x} w_x}$$

Increasing weights of structural attributes favors nodes with higher network centrality. Increasing weights of behavioral attributes favors nodes based on their communication strength. Our experiments suggest higher weights for behavior attributes works best in identifying detector nodes. We used weight 0.9 for the *weighted clique* metric that captures dynamic behavior, and weights 0.8 and 0.7 respectively for *raw cliques* and *number of cliques* metrics. We used weight 0.5 for betweenness centrality and closeness centrality metrics. This combination of weights selected nodes that are most “socially important” based on dynamic behavior and network centrality in DNC email network.

3.4 Modified Social Score

The dynamic behavior of a node is determined by its *weighted clique* metric. This metric relies on determining average response time for emails. The method proposed by Rowe et al. isn't robust. A delay in response due to time zone differences between nodes or different working hours can skew average response time computation. We defined a new measure to determine the importance of node in an email network. We consider a node to be important if a large fraction of emails sent by a node are responded to with high priority by the receivers. We consider a mail to be handled with high priority if a destination node, after receiving an email from a source node, responds to the node within next N emails. We used N=5, i.e node responds within next five emails. We compute the fraction of such high priority responses. A higher fraction indicates higher social rank. We used a modified weighted clique metric $(W' = f * R)$ where f is fraction of emails handled with priority. We only consider pairs of nodes that have exchanged a significant volume of emails (>200) for computing priority fractions. The top 50 nodes picked by modified social score algorithm were identical to the nodes picked by the Rowe et al. algorithm, but the order of a few nodes differed. This indicates that our modified social score algorithm performs just as well on this email network while our modified algorithm could provide a more robust method for determining the social hierarchy of nodes spread across different time zone or different working hours.

3.5 CELF with Social Score Speed-up

For our speed-up algorithm, we simply run CELF over a subset of nodes outputted from the modified social score algorithm described above. To pick the first detector, we compute the marginal gains from the top √n of social score-ranked nodes. We then greedily select the node with the largest marginal gain and store the rest of the nodes and their respective marginals in a priority queue. In each subsequent step, we add one more node from the social score-ranked nodes and add it to the priority queue. This heap of √n nodes constitutes candidate nodes from which we then run the CELF algorithm to select a detector. Our speed-up algorithm thus ensures that each round of selecting a detector at most requires running outbreaks for √n nodes, a substantial improvement upon the worst-case scenario of linear-time evaluations for CELF.
Our algorithm still exhibits characteristics of submodularity. We do see step-like changes in performance only when we peek at the next node in the social score list and add it to our priority queue of marginals. However, after immediately selecting a previously unseen node, lazy forward evaluation of the top nodes ensures that we are still greedily grabbing the largest marginal gain because the other marginals would only be lower with the addition of the unseen node to whatever set of nodes those marginals were previously computed over.

4 Results

4.1 Contrasting Social Score vs. CELF vs. Speed-up

Below are three pairs of graphs, which correspond to the three metrics of *outbreak detection probability*, *population affected*, and *time to detection*, labeled on the y-axis of the left plots. In each pair, the graph on the left shows the performance of the three algorithms as we add detectors, while the graph on the right contrasts the runtimes of the three algorithms. The hyper-parameters (*probability of initial infection, probability of infection, number of outbreaks*) are listed in the plot title.

Figure 1: Maximizing Probability of Detecting and Outbreak

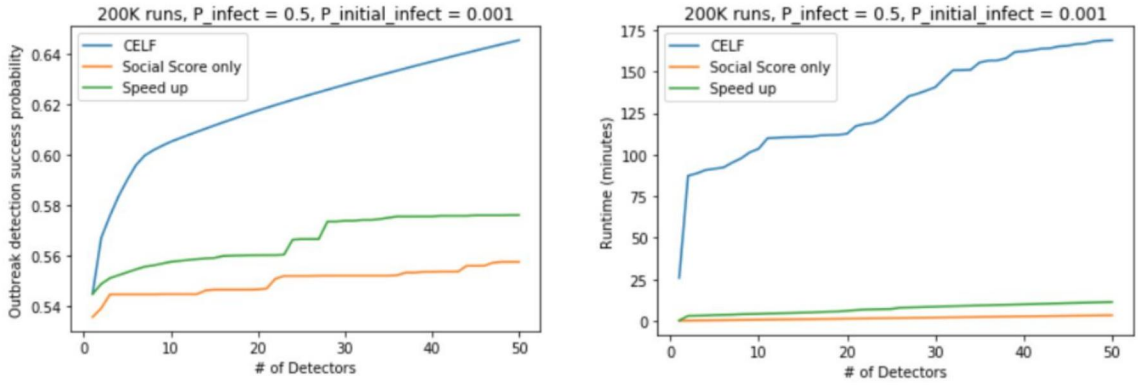
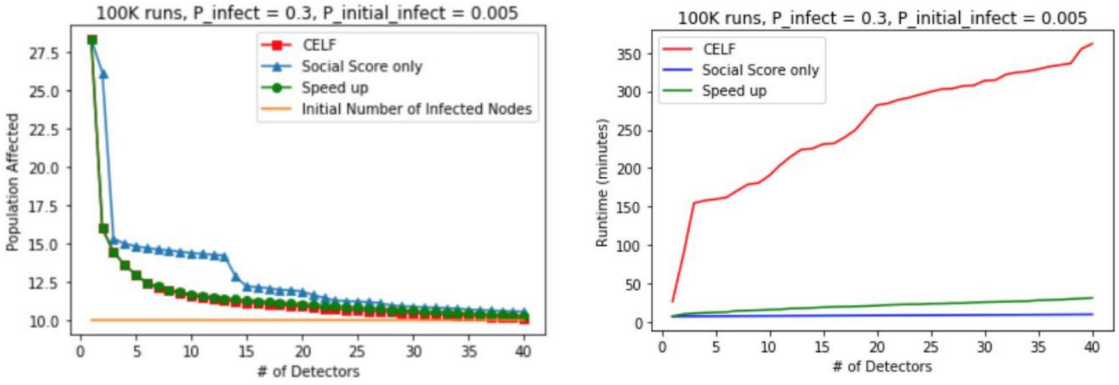


Figure 1 confirms the trade-offs in our implementation. Our speed-up contribution performs worse than CELF, but better than a purely static social score implementation. The right plot shows a drastic improvement in runtime for our speed-up algorithm. Additionally, our speed up implementation is also mostly concave down, which attests to the submodularity of the greedy selection method, in contrast to the step-like nature of social score.

Figure 2: Minimizing Population Infected



For Figure 2, we add a yellow asymptote to denote the initial number of infected nodes and convergence goal for our algorithms. For this metric, and across other hyper-parameters as well, we see that speed up performs just as well as CELF. Indeed, for the first few detector

nodes, speed-up would select the exact same detectors. Again, the runtime comparisons confirm the tremendous speed improvement of our algorithm.

Figure 3: Minimizing Time to Detection

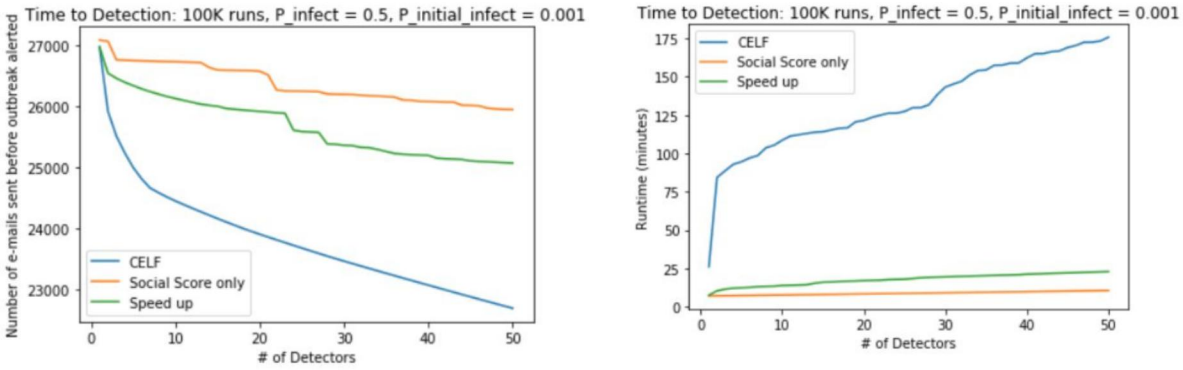


Figure 3 reinforces the trade-offs we make with our speed-up algorithm. Time to detection is measured in simulation steps, or number of emails transmitted in the exact same sequence as the DNC email network, as explained in Section 3.1. We trade time to detection performance for drastic increases in runtime again. We also see that the speed up algorithm is concave up for the first few detectors and thus generally exhibits submodular characteristics for initial detectors.

4.2 Analyzing Social Score Node Structure

Figure 4 (below, left) shows the cumulative egonet set size vs. number of nodes selected. It illustrates why statically selecting nodes ranked by social score can be suboptimal due to overlapping influence sets. The graph shows distinct step-like increases in cumulative egonet set size for social score, while the greedy algorithms are smooth curves, maximizing reach with every additional node. Figure 5 (a) and (b) (below, right) shows that the egonet size of the second detector nodes selected by CELF is much larger when compared to the second node selected based on Social Score. Figure 5 (c) and (d) show the egonets of nodes selected by CELF that have lower social scores but significant influence.

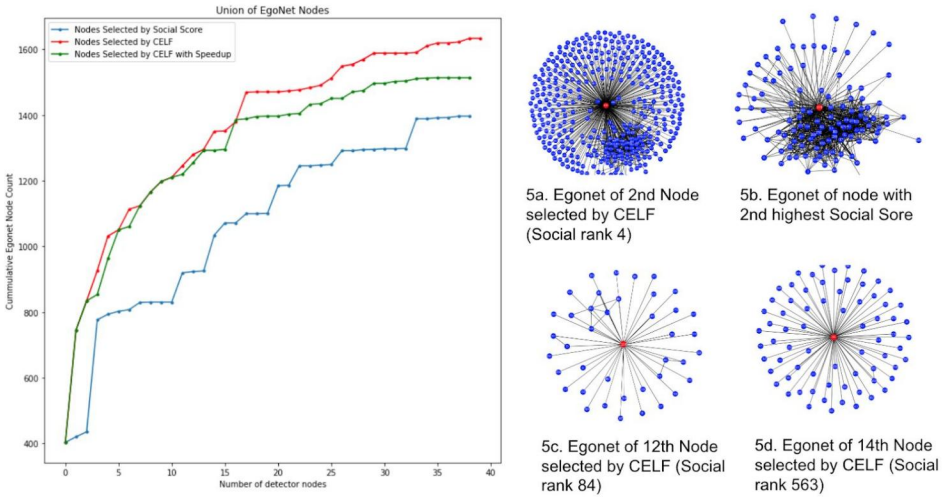
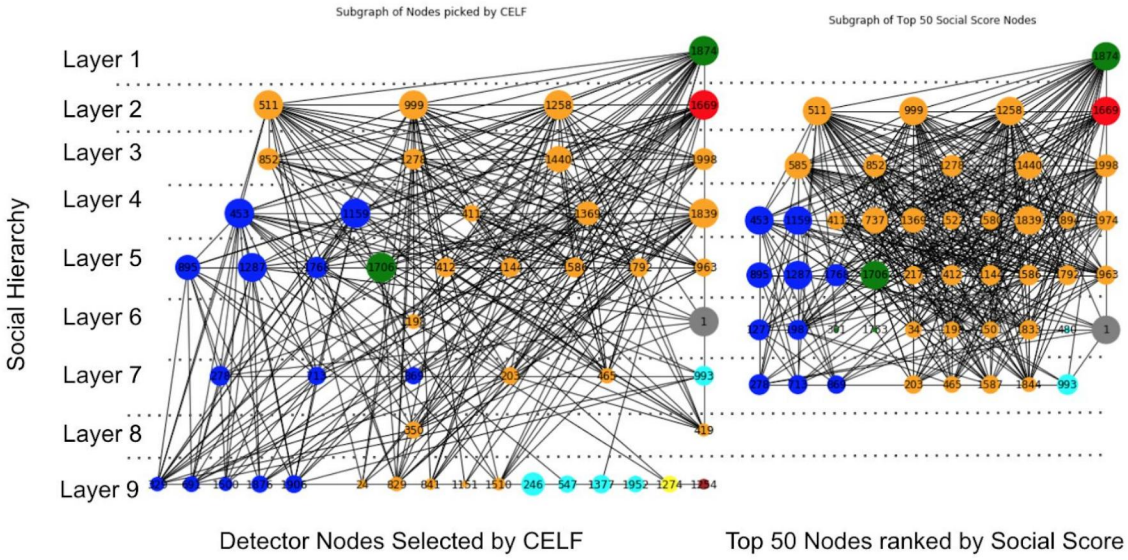
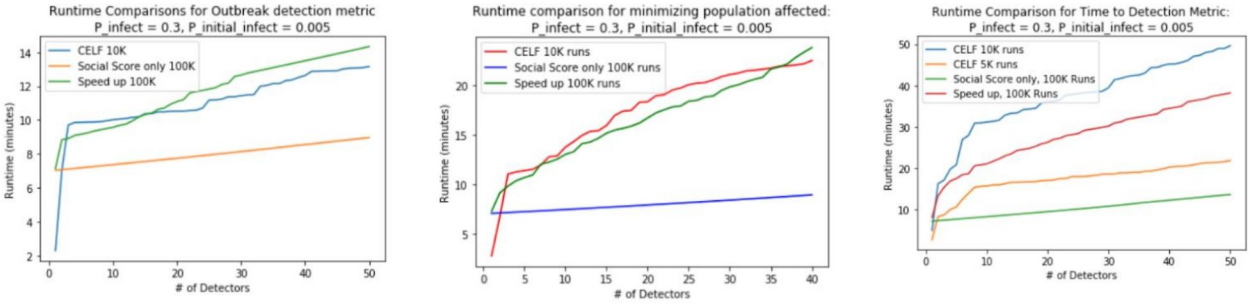


Figure 6 (a) and (b) (below) show the top 50 nodes picked by CELF and social score respectively. We linearly split the nodes into 10 bins (layers) using social score. The size of the node in the figure is proportional to the egonet size of the node. We detect communities within the email network using Clauset-Newman-Moore greedy modularity maximization algorithm. The nodes are colored based on their community. The social score algorithm effectively identifies social hierarchy and detects influential nodes within different communities at different levels within the organization. Figure 6 shows CELF is effective in selecting nodes with lower social score (layer 9) that have higher influence.



4.3 Further Runtime Analyses for Speed-up Algorithm

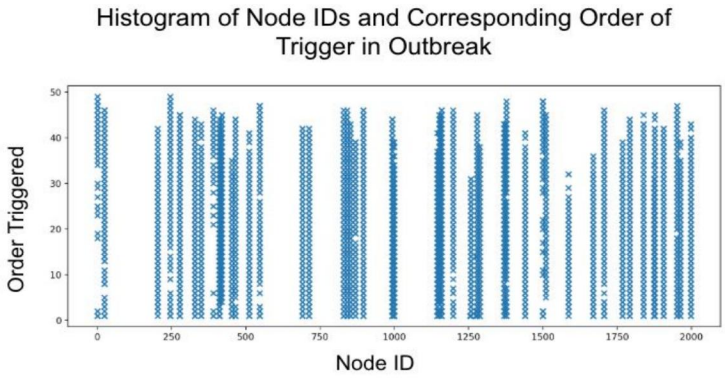
Figure 7



In Section 4.1, we showed the performance vs. speed trade-offs we observed in our speed-up implementation. Figure 7 compares the different algorithms, with the three plots corresponding to the three metrics, over the detectors on the x-axis and the runtime on the y-axis. The legends label how many outbreaks we use to evaluate each marginal gain for each algorithm. For the three metrics, we observe that the CELF algorithm, with 10K outbreaks per marginal gain, runs in similar time to our speed-up algorithm over 100K outbreaks, which is a 10x speed-up. Indeed, for the time to detection metric, our speed-up algorithm with 100K outbreaks (red line in third plot) runs quicker than CELF w/ 10K outbreaks (blue line), which is even more than a 10x improvement. Comparing runtimes of 100k outbreaks v/s 5k outbreaks further demonstrates that large number of outbreaks (to reduce variance in Monte Carlo simulation) present the most beneficial scenarios to see runtime improvements from our speed-up algorithm.

These analyses empirically verify the $O(v\sqrt{n})$ runtime bounds of speed-up in contrast to the O(v*n) runtime bounds on CELF. Additionally, the order of magnitude runtime improvement empirically observed above holds ramifications for variance reduction in outbreaks simulations as well.

4.4 Investigation of Order of Detector Triggering



We briefly investigated the possibility of predicting the order of detector triggering in our outbreak simulation using linear and logistic regression. However, the order of triggering had very limited discernable patterns across 1,000 outbreaks. For example, note in the in the histogram above that although node 1624 tends to be triggered early (positions 1-30 in each outbreak) most other nodes are equally likely to be triggered first, last, or somewhere in the middle. While the prediction of detector trigger order may be an interesting area of future work, our project reinforces the power of the CELF algorithm in remaining consistently effective despite the inherent stochasticity in these simulations, which can pose a challenge for machine learning and pattern-recognition based models.

5 Conclusion and Future Work

CELF is a powerful tool for optimal detector placement in outbreaks, but researchers and network scientists working on massive datasets could benefit from a faster algorithm. We have shown that we can achieve reasonable quality across detection probability, population infected, and time to detection with significant speedups in runtime. Indeed, we show that we can essentially match CELF performance on population infected in our empirical dataset while achieving a 10x runtime gain. These runtime improvements are additionally important for variance reduction in future Monte Carlo simulation work on outbreak modeling. Future work would focus on validating our findings on other email networks with different architectures to demonstrate the robustness of our results. From the perspective of a sysadmin, this speedier placement can lead to a wider array of investigations and faster modelling, consequently helping mitigate the impact of phishing attacks such as the DNC outbreak of 2016.

References

[1] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In KDD 2003.

[2] J. Leskovec et al. Cost-effective outbreak detection in networks. In KDD 2007.

[3] Ryan Rowe, German Creamer, Shlomo Hershkop, and Salvatore J Stolfo. 2007. Automated social hierarchy detection through email network analysis. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*

[4] Dnc emails network dataset. KONECT, April 2017.

[5] Epidemiological Modeling of News and Rumors on Twitter. Jin et al. SNAKDD 2013.

[6] Verizon Data Breach Investigations Report 2017. https://www.ictsecuritymagazine.com/wp-content/uploads/2017-Data-Breach-Investigations-Report.pdf