SANDIA REPORT

SAND2018-12068
Unlimited Release
Printed September, 2018

An Example of Counter-Adversarial Community Detection Analysis

W. Philip Kegelmeyer, Jeremy D. Wendt, Ali Pinar

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology and Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2018-12068
Unlimited Release
Printed September, 2018

An Example of Counter-Adversarial Community Detection Analysis

W. Philip Kegelmeyer, Jeremy D. Wendt, Ali Pinar

Abstract

Community detection is often used to understand the nature of a network. However, there may exist an adversarial member of the network who wishes to evade that understanding. We analyze one such specific situation, quantifying the efficacy of certain attacks against a particular analytic use of community detection and providing a preliminary assessment of a possible defense.

Acknowledgment

We are grateful for insightful comments from Kristen Altenburger of Stanford University, Jon Berry and Cindy Phillips of Sandia Laboratories, and Cliff Anderson-Bergman of Lawrence Livermore National Laboratory.

Contents

1 Motivation and Data **9**

2 A Model for Adversarial Tampering with Community Detection **11**

Developed Attacks 12

Quantifying Attack Performance 13

Results 19

3 Conclusion **25**

References **26**

Appendix

A Appendix: Edge Features **29**

List of Figures

2.1 **Stable Structure Matrix**: This matrix illustrates varying levels of stability for our sample graph of 208 nodes. For visual coherence, the nodes are ordered such that fully stable structures (dark purple squares) are together, and less stable structures are similarly grouped together. The color at (i, j) indicates the percentage of the Louvain runs where nodes i and j were in the same community. Our Stable Structure Attack orders these fully stable structures in increasing temperature order and adds edges to the coldest stable communities first. 14

2.2 **Quantifying One Attack's Effectiveness**: This shows how we quantify a single attack's effectiveness; Stratified Random used here as an example. (a) For one probe node, define only one attack, and run Louvain only one time. (b) The same, but running Louvain 20 times. (c) The same as (a) but showing only the average from 2,000 Louvain runs. (d) The same as (c) but averaging 20 Stratified Random attack definitions. (e) The same as (d) but showing 10 different probe nodes. (f) The same as (e) but showing all possible probe nodes averaged. 15

2.3 **All Attacks' Effectiveness**: All attacks' averaged results show that Greedy Pessimal is the best attack found so far. Stable Structure drops the probe's temperature nearly as quickly for the first several edges added, but then begins increasing the temperature. 16

2.4 **Attack and Defense Example**: For a specific pairing of train probe node and test probe node, we generate plots like the above. The top row shows the amount the attack changed the probe's community temperature (the darker the blue, the more effective the attack), while the bottom five rows show how much the training attack (row) and test attack (column) pairing changed the probe's community temperature (the darker the red, the more effective the remediation). 19

2.5 **Attack and Defense: Isolated Defense Models**: This shows average defense results when we train against the features of a single canacSBM graph. Each internal square is the averaged result across 100 different canacSBM graph instances. The external rows are the training probe nodes; external columns are test probe nodes. In general, this defense performed rather poorly, particularly in the seventh column, where the dark blue indicates effective attacks, but the lack of dark red indicates few effective defenses. 20

2.6 **Attack and Defense Results: Combined Defense Models**: This shows results when we train against the combined features from all 100 canacSBM graphs (merged on same train probe node and train attack). On the whole, this improved over the previous result, but still was not uniformly good. 21

2.7 **Attack and Defense Results: Combined Defense Models with Skew Correction**: This shows results when we train against the combined features from all 100 canacSBM graphs when resampled using the SMOTE algorithm set to 30. Thus far, this level of defense provided our best result. 23

Chapter 1

Motivation and Data

We are motivated by prior work [4] which investigated the use of community detection as a means of helping human analysts assess supply chain risk. The supply chain analysis made use of "vendor graphs", where every node was a web page related to the business domain of interest, and every edge is a standard web link. In that work it was noted that there were properties of the nodes that were not statistically interesting at the node level, but which possessed a group dynamic; the likelihood that *all* members of a community should be assessed for supply chain risk correlated with the number of nodes with that property.

To support an analysis of this situation, we created a vendor graph constructed from web crawling. We started with four "seed" home pages for businesses related to ham radios, created a 2.5 hop ego network from each seed, took the union of the graph that resulted, and eliminated the degree-one nodes that resulted from the termination of the web crawl. To simulate the "node vs group" dynamic just discussed, each node was assigned a binary "temperature". That is, it was labeled as "hot" or "cold"; Chapter 2 explains the use of those labels.

The resulting graph was deliberately relatively small (208 nodes, 594 edges), because the vendor graphs examined by human analysts indeed tend to be sized by human attentional capacity, and also to permit the exhaustive computational investigation needed to fuel the discovery of more feasible heuristics that might scale to larger graphs.

Chapter 2

A Model for Adversarial Tampering with Community Detection

As mentioned above, we are motivated to assess the Louvain community detection method for vulnerabilities. Louvain, like many alternative methods [3, 6], focuses on "modularity maximization"; that is, it attempts to find partitions where the internal structure of the partition is maximally denser than would be found in a random partition. Further, Louvain generates exclusive communities, where all nodes are assigned to exactly one community each. Formally, exclusive community detection accepts a graph ($G = (V, E)$; $V = \{v_0, v_1, \ldots, v_n\}$; $E = \{(v_i, v_j)\}$, $|E| = m$) and produces a partitioning of the graph.

Community detection is generally part of some larger graph analysis. As discussed in the introduction, here we consider a case where an analyst has a weak indicator, based on node metadata, that we refer to as "per-node temperature". A hot node may be of greater interest to the analyst than a cold node. However, as per-node temperature is a very weak indicator, and is significant only in aggregate, the analyst prioritizes which nodes to investigate based on "community temperature". Nodes in a hotter community are more interesting than those in a colder community — independent of any individual node's temperature. In order to support a quantitative interpretation of community temperature, without loss of generality we use a simple assignment where a node with a per-node temperature of "hot" is assigned the value 1, "cold" nodes map to -1, and any unknown nodes map to 0.

With those idioms in place, computing the *community* temperature from the nodes in the community is straightforward. After identifying community via community detection, sum the per-node temperatures of all nodes in that community. The community temperature is this sum divided by the number of nodes in the community.

We now model how an adversary might attack this community-detection-aided temperature analytic to reduce the community temperature of a particular node — thus making that node less likely to be highlighted for further analysis. We make the worst case assumption of a *fully informed, empowered* adversary. Specifically, we assume the adversary has the following capabilities:

- precise knowledge of which dataset we will collect,
- full understanding of our temperature analysis (including the weak indicator function),

and

- the ability to add b edges from their node to any other nodes in the graph. Note that in vender graphs, this is a cheap, easy, and fairly unobtrusive intervention, as linking to other sites is one of the most common things web sites do.

We refer to b as the adversary's budget. Thus, a quantitative measure of the efficacy of the adversary's attack is the amount they can reduce their node's temperature given the addition of b false edges.

An adversary's attack is defined, then, by the following three elements:

- **Data**: The graph as defined with per-node temperatures and edges before the adversary attacks.
- **Probe node**: The node whose temperature is to be lowered.
- **Edge attack order**: A listing of all non-probe-node nodes giving the order in which the adversary would add edges from the probe node to these nodes. A budget of b would mean that the adversary would add edges to the first b entries in this list.

Developed Attacks

We developed five main attack families. Each attack-defining method takes as input a graph and a probe node and generates an edge attack order.

- **Random** (rr): This serves as a baseline for all other attacks. It simply generates a randomly ordered list of all remaining nodes in the graph.
- **Stratified Random** (sr): This attack groups the nodes by per-node temperature (in the order of cold, unknown, hot) and then randomly orders the nodes within each group. This is the simplest sensible attack.
- **Cold and Lonely** (cal): This attack sorts the nodes by per-node temperature (in the order cold, unknown, hot), then orders the nodes within each group by increasing node degree. When both per-node temperature and node degree are the same, nodes are randomly ordered. The motivation here was to attempt to exploit the nature of the modularity metric, which weights cutting a node's edge more highly if the node has few edges to begin with.
- **Greedy Pessimal** (gp): This attack exhaustively tests how the probe node's community temperature changes when an edge is added to each other node individually. It adds the edge which best decreases resulting temperature. This exhaustive check is then repeated for each remaining node against the resulting graph after adding the

first attack edge. This is repeated until an edge is added to each node. Note that for graphs of any considerable scale, this attack becomes infeasible; it was included to provide a sense of the worst case (for the defender) outcome.

- **Stable Structure** (ss): This attack leverages Louvain's stochastic nature. That is, Louvain will give slightly different results each time it is run with a different random seed. If we run the same graph through Louvain multiple times, tracking how often each pair of nodes are in the same community, we will identify those structures in the graph that appear more "stable" — nodes that Louvain places in the same community across many runs (see all dark purple squares in Figure 2.1). The stable structure attack orders those stable structures by temperature and connects to them in ascending order (random order for nodes within each stable structure). Some nodes are in no stable structures. These nodes are ordered after all stable structures using the stratified random algorithm.

Quantifying Attack Performance

We consider the best attack to be the one which most quickly decreases a probe node's temperature. However, given that all attacks are affected by Louvain's stochastic nature, we must define how to quantify attack effectiveness.

Figure 2.2 illustrates how we must average along three different dimensions to obtain a single quantifiable plot for an attack type. Each component plot in the figure plots the community temperature of a probe node (on the y-axis) as a function of the number of added inserted edges (on the x-axis). So, for Figure 2.2(a), we see that the probe node `www.kenwoodusa.com` starts with a relatively warm community temperature of around 0.4. After adding about twenty edges its community temperature had dropped dramatically to around -0.43 (the mean temperature of all nodes in the graph), and hits a minimum of around -0.7 after adding 140 edges, at which point it has connected to all of the cold nodes in the graph.

However, Figure 2.2(a) is the result of a single Louvain run. To obtain a sense of expected behavior, we must average across many Louvain runs (Figure 2.2(b)). Experimentally, we found that 2,000 runs provided good stability and was relatively quickly obtained (Figure 2.2(c)). Further, as all attacks have some stochastic component to the edge ordering, we also must average across multiple results from the same attack with different seeds (20 provided good stability; Figure 2.2(d)). Finally, an attack may be more effective from one probe node than another (Figure 2.2(e)). In order to get an overall sense of the efficacy of an attack heuristic, we average its probe-specific curve over all probe nodes (Figure 2.2(f)).

Figure 2.3 shows the relative performance of each of our five attack heuristics against each other. Each of these curves has gone through the same averaging process described for Figure 2.2(f). Greedy Pessimal is clearly the best attack for dropping the probe node's community temperature and keeping the temperature low. However, Stable Structure performs

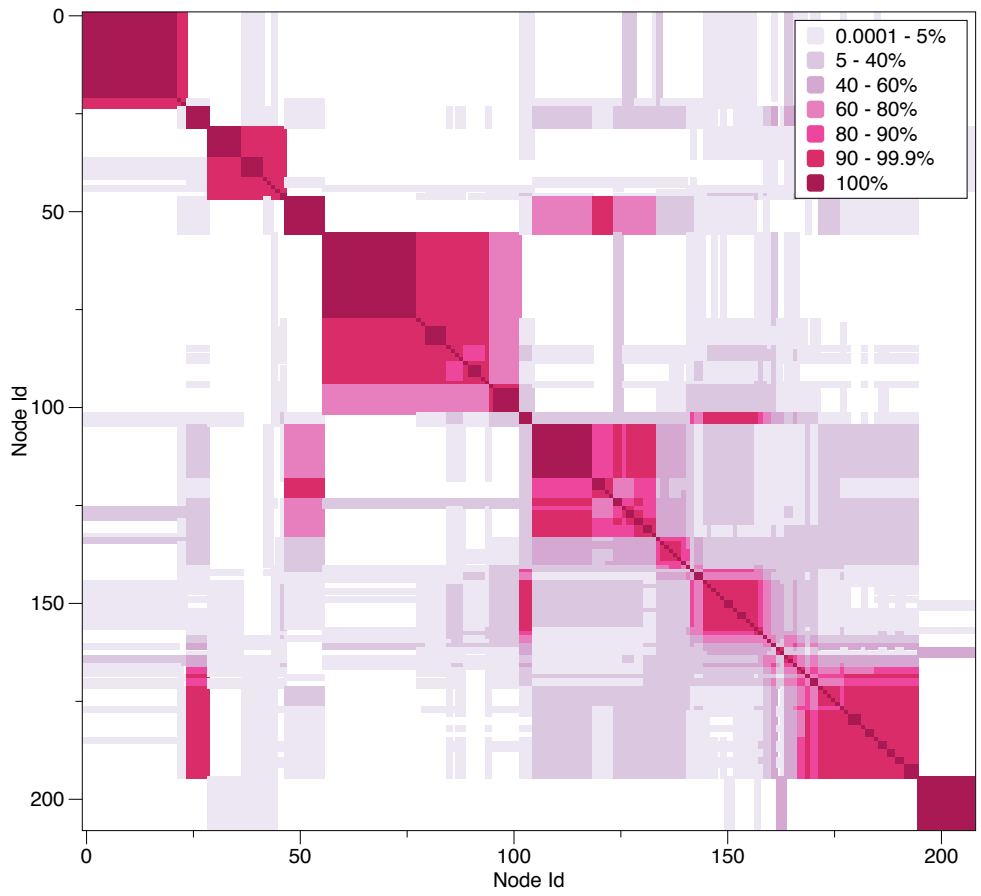
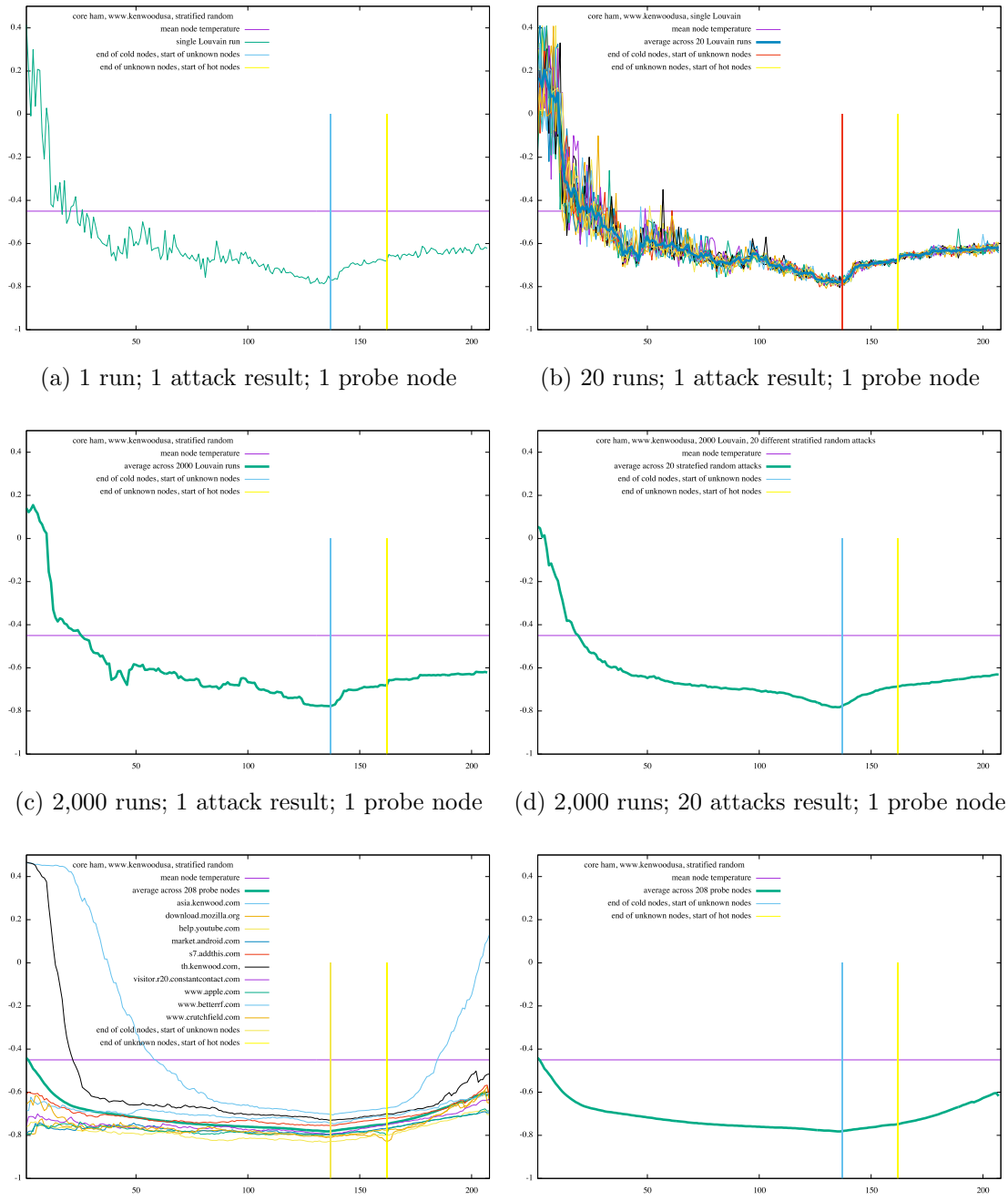


Figure 2.1: **Stable Structure Matrix**: This matrix illustrates varying levels of stability for our sample graph of 208 nodes. For visual coherence, the nodes are ordered such that fully stable structures (dark purple squares) are together, and less stable structures are similarly grouped together. The color at (i, j) indicates the percentage of the Louvain runs where nodes i and j were in the same community. Our Stable Structure Attack orders these fully stable structures in increasing temperature order and adds edges to the coldest stable communities first.



(a) 1 run; 1 attack result; 1 probe node

(b) 20 runs; 1 attack result; 1 probe node

(c) 2,000 runs; 1 attack result; 1 probe node

(d) 2,000 runs; 20 attacks result; 1 probe node

(e) 2,000 runs; 20 attacks result; 10 probe nodes

(f) 2,000 runs; 20 attacks result; All probe nodes

Figure 2.2: **Quantifying One Attack's Effectiveness**: This shows how we quantify a single attack's effectiveness; Stratified Random used here as an example. (a) For one probe node, define only one attack, and run Louvain only one time. (b) The same, but running Louvain 20 times. (c) The same as (a) but showing only the average from 2,000 Louvain runs. (d) The same as (c) but averaging 20 Stratified Random attack definitions. (e) The same as (d) but showing 10 different probe nodes. (f) The same as (e) but showing all possible probe nodes averaged.

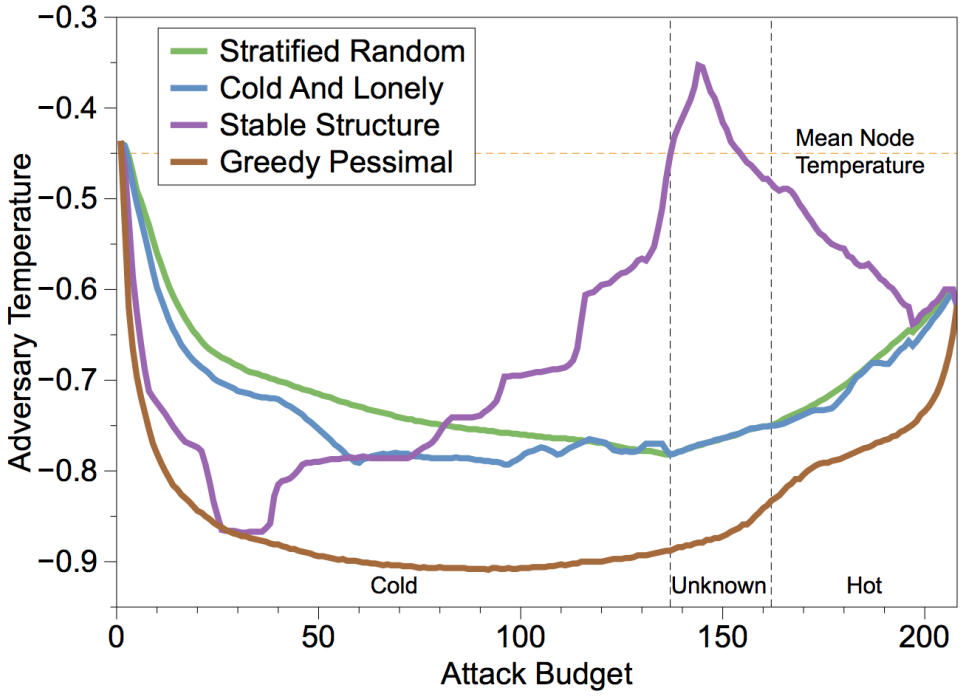


Figure 2.3: **All Attacks' Effectiveness**: All attacks' averaged results show that Greedy Pessimal is the best attack found so far. Stable Structure drops the probe's temperature nearly as quickly for the first several edges added, but then begins increasing the temperature.

nearly as well at dropping the temperature quickly. This is significant, as Stable Structure is feasible for large graphs, where Greedy Pessimal is not. Admittedly, shortly after its early success Stable Structure begins raising the probe's community temperature and reaches the highest temperature of any of the attacks. However, no attacker would execute an attack beyond the tens of edges — both because the attack decreases in effectiveness beyond that point and because the more edges inserted, the greater the risk of detection. While the Cold and Lonely Attack performs slightly better than the Stratified Random Attack, they both follow a similar structure: a slow decrease through the cold nodes is followed by slight warming when adding unknown nodes — ending with a rapid increase through the hot nodes. The Random Attack is not shown because when averaged across all probe nodes, it results in approximately mean node temperature throughout.

Defending Against Community Attack

An earlier "Counter-Adversarial Data Analytics" (CADA) research effort at Sandia investigated attacks on supervised machine learning executed through tampering with the groundtruth labels used to train machine learning models [5]. The CADA project demonstrated some ability to defend against such attacks by building machine learning models to detect, and remediate, tampered labels. Inspired by that approach, we similarly used ensembles of decision trees to try to identify inserted edges and remove them from an attacked graph. This requires several important components: training data, training features, and hyperparameter tuning.

In classical machine learning problems, splitting a dataset into training and testing datasets can be done in a straightforward manner — often randomly splitting data points between the two datasets. This is possible because the datapoints' features are based solely on the features of each datapoint. However, in the case of a graph (and in this case, the edges of a graph), any edge features derived based on either of the connected nodes will bleed across to any other edge that shares the same node. Thus, either the features must be carefully limited to eliminate across-datapoint contamination, or identifying a proper training set must be done differently.

In this work, we felt we needed the necessary power that many node-based features would convey, therefore we needed to find a principled way to have separable training and testing datasets. We decided that leveraging the statistically similar graphs generated by canacSBM [1] would serve well. canacSBM ("community and node attribute corrected stochastic block model") is a newly developed generative graph model where all nodes preserve statistically similar overall degree distributions, temperature linking distributions, and both within- and across-community linking patterns. These graphs preserve similar overall community detection results as well. Importantly, however, they do not preserve specific node-to-node linking behavior — thus resulting in different-but-similar graphs on the same set of nodes.

Therefore, we generated 100 statistically similar canacSBM graphs from our one input

graph. We then performed a series of different experiments which all followed the same overall pattern:

Train defense model:

1. Select a training probe node and a training attack.
2. Attack a canacSBM graph from the selected probe node using the selected attack for a budget of 20 inserted edges.
3. Extract features for each edge in the attacked graph, preserving the label of "original" or "inserted" edge.
4. Train an ensemble of decision trees to identify "original" or "inserted" edges using these features.

Use the defense model to find inserted edges:

1. Select a testing probe node and a testing attack.
2. Attack the original graph from the selected probe node using the selected attack for a budget of 20 inserted edges.
3. Extract features for each edge in the attacked graph, preserving the label of "original" or "inserted" edge (without providing that label information to the defense model).
4. Apply the ensemble created in Training Step 4 against these just-generated features.

Measure the effectiveness of the defense:

1. Remove all edges from the attacked graph that the ensemble identified as "inserted" (whether correctly or not) — we call this remediation.
2. Determine probe's community temperature at three times — before attack (b_a), after attack (a_a), and after remediation (a_r).

We report two measures for each defense: attack effect is the amount the temperature changed due to attack ($b_a - a_a$) and remediation effect is the amount the temperature changed due to remediation ($a_a - a_r$).

Figure 2.4 illustrates how we visualize attack and remediation effects. Each plot like this one is for some specified train probe node and test probe node pairing. The top row of the plot shows each column attack's effectiveness against the test graph. In each square, darker blue indicates making the temperature colder, white indicates no change, and red indicates making the temperature hotter. Using this same color schema, the five lower rows show how effectively training on the row's attack type performed against the column's testing attack. For this plot, the random attack (rr) was not effective at changing the probe's community temperature and none of the remediations altered it much either. The other four attacks were quite effective, and were always effectively remediated when trained and tested on the

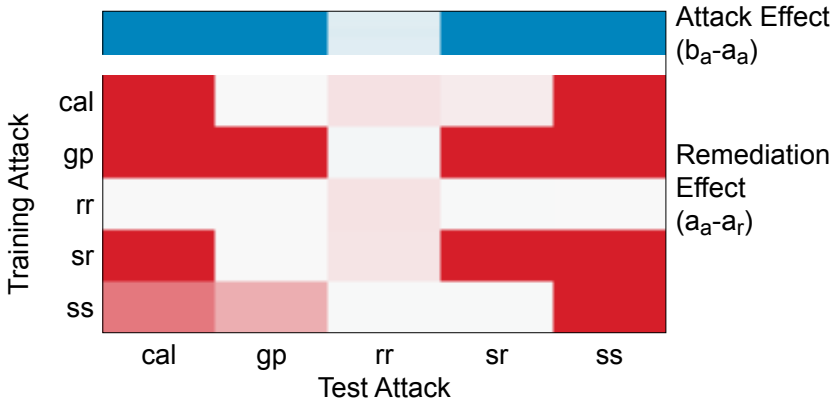


Figure 2.4: **Attack and Defense Example**: For a specific pairing of train probe node and test probe node, we generate plots like the above. The top row shows the amount the attack changed the probe's community temperature (the darker the blue, the more effective the attack), while the bottom five rows show how much the training attack (row) and test attack (column) pairing changed the probe's community temperature (the darker the red, the more effective the remediation).

same (non-rr) attack, and sometimes remediated well when trained and tested on different attacks.

As generating the underlying data for these attacks was expensive, and visualizing all possible pairings of the 208 nodes in the test graph would be infeasible, we limited our experiments to eight example probe nodes. Our original test graph was generated as the unified ego nets of four seed nodes; we used three of these as probes. For the remaining five nodes, we selected a set of five nodes selected at random from the remaining 205 nodes.

The ensemble of decision trees requires descriptive features for each edge. We extracted a number of such; see the Appendix. One feature which we considered using but decided against was an edge's source node degree and destination node degree. While we believe this feature will be truly useful in general, in this case, we felt it unfairly aided the defender: many nodes in the graph are of very low degree, and the attacker was guaranteed to be at least degree 20. In early tests, this feature was the most important and made all edges from the probe node quite suspicious.

Results

One-to-one train-and-test. For our first analysis, we trained against the features from a single attacked canacSBM graph and tested against our attacked test graph — following the previously described steps directly. To generate the final visualization (Figure 2.5), we averaged the results from the 100 different canacSBM graphs.

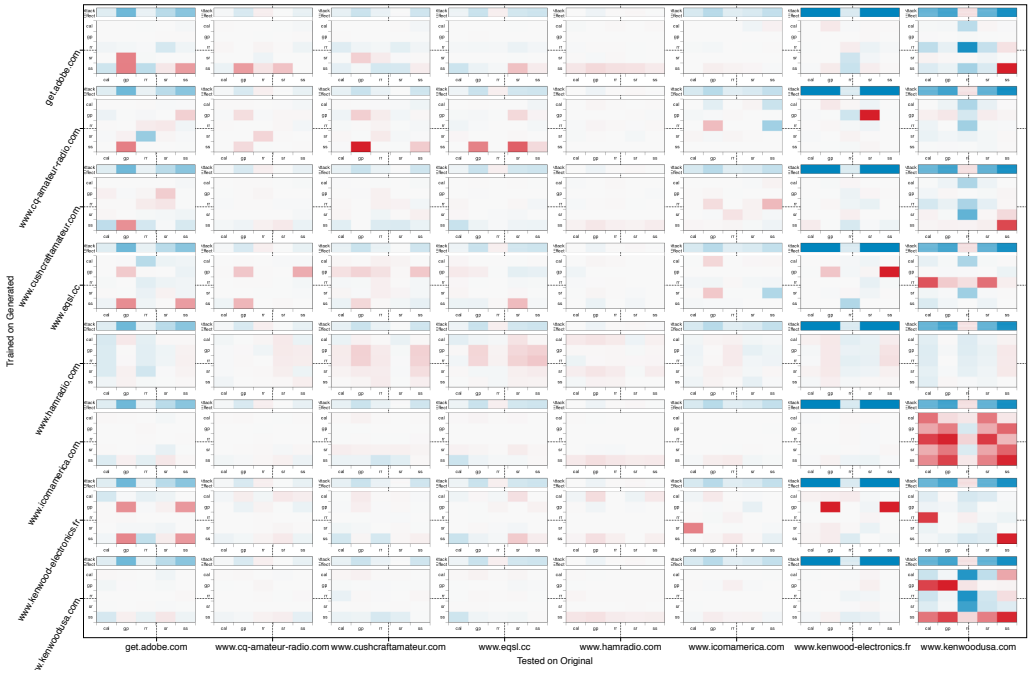


Figure 2.5: **Attack and Defense: Isolated Defense Models**: This shows average defense results when we train against the features of a single canacSBM graph. Each internal square is the averaged result across 100 different canacSBM graph instances. The external rows are the training probe nodes; external columns are test probe nodes. In general, this defense performed rather poorly, particularly in the seventh column, where the dark blue indicates effective attacks, but the lack of dark red indicates few effective defenses.

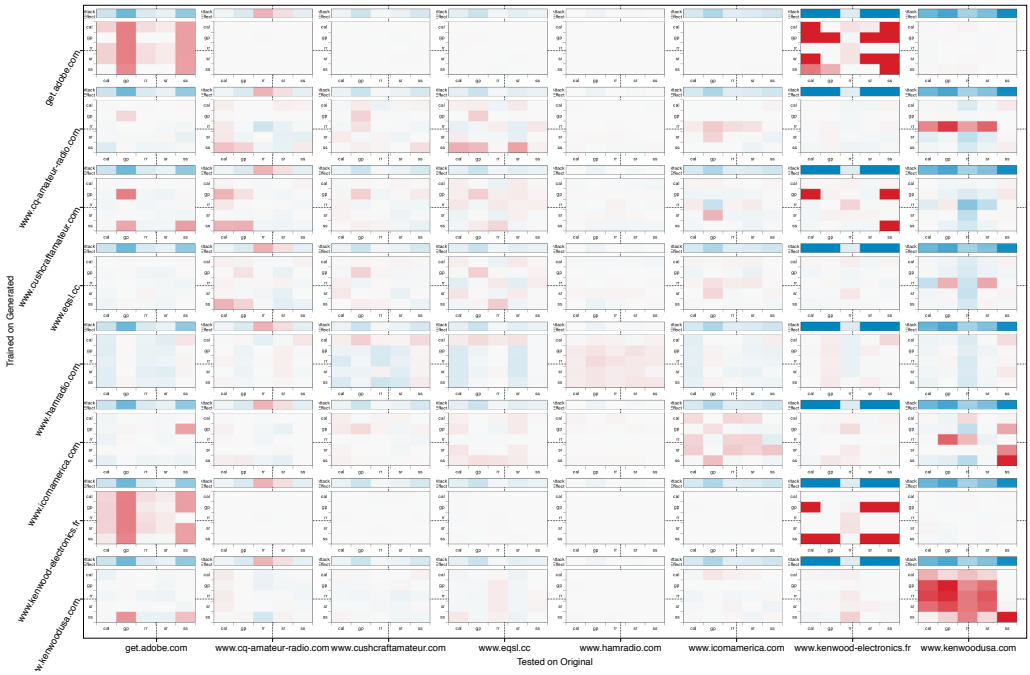


Figure 2.6: **Attack and Defense Results: Combined Defense Models**: This shows results when we train against the combined features from all 100 canacSBM graphs (merged on same train probe node and train attack). On the whole, this improved over the previous result, but still was not uniformly good.

While it is discouraging that, in general, these defenses are ineffective, as we studied Figure 2.5, we did notice one key thing. The attacks from four of the tested probes were essentially ineffective in altering the probe's community temperatures (www.cq-amateur-radio.com, www.cushcraftamateur.com, www.eqsl.cc, www.hamradio.com). As these attacks didn't make it much colder, we should hope that our remediations do not make it colder or hotter. In this remediation setup, there are a few cases where this is not true (four of the squares are pretty dark red under very light blue attacks), but in later setups, this serves as a good way to ensure that remediations do not make hotter what was not made considerably colder.

Merged canacSBMs-train. We realized that with 100 different canacSBM graphs generating features, we could merge all graphs' features into one massive feature file for each probe node/train attack configuration. This much larger file results in only one attack remediation per square (instead of averaging 100 as previously). This led to somewhat improved results (Figure 2.6).

Merged canacSBMs-train with improved sampling. The unattacked graph has almost 600 edges. In attacking, we add only 20 edges. Thus, even when merging the features across all of the files, the ensemble of decision trees still has a nearly 30-to-1 imbalance between original and inserted edges. In general, learners improve when trained on more balanced datasets. We therefore applied the SMOTE resampling algorithm [2], which uses a rough non-parametric estimate of the distribution of the minority class to temporarily generate synthetic samples for use in training, to obtain a 70/30 original/inserted balance in the training data. As Figure 2.7 indicates, we indeed saw additional improvement in the number of cases where remediation was effective.

In general, it appears that if we guess the probe node correctly, remediations do reasonably well. (Though, of course, if we can indeed correctly guess the probe node, much of the subsequent analysis may be unnecessary.) If the assumed probe node does not match the probe node of the actual attack, it generally does not work — with one exception.

In this data, get.adobe.com and kenwood-electronics.fr seem to serve as proxies for each other. Training on one provides adequate defense against both itself and the other. Moreover, this result began to be visible even when we trained without SMOTE (Figure 2.6). We have not had a chance to dig into why this might be, but our hypothesis is that these two nodes fulfill the same role (e.g., hub, bridge, etc.) in the graph.

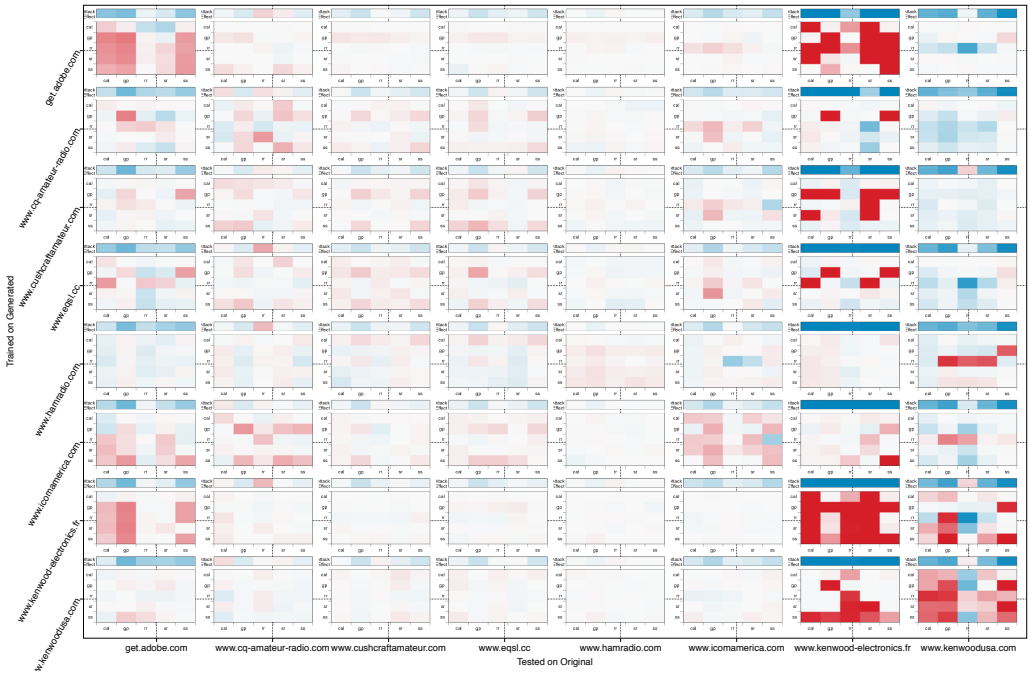


Figure 2.7: **Attack and Defense Results: Combined Defense Models with Skew Correction**: This shows results when we train against the combined features from all 100 canacSBM graphs when resampled using the SMOTE algorithm set to 30. Thus far, this level of defense provided our best result.

Chapter 3

Conclusion

This paper presents attacks and defenses for community detection-based analytics. Our most effective, feasible attack was based on the novel discovery of "stable structures", derived from tracking co-assignment of nodes to the same community across multiple runs of the Louvain algorithm. We believe this stable structure matrix (Figure 2.1) is a useful concept beyond counter adversarial analytics, in that it points the way to less forced community detection assessments that permit nodes to have fuzzy membership in communities, or to be part of no community at all. Moreover, while we leveraged Louvain's randomness to generate our stable structure matrix herein, a stable structure matrix could be derived for any randomized community detection algorithm or even for ensembles of many community detection algorithms.

Overall, our analysis indicates that community detection-based analytics can be easily subverted by a determined and empowered adversary with relatively little effort on their part, through adding a small number of intelligently chosen edges to nodes outside their "natural" community. However, we also presented defenses that can sometimes remediate the effects of these attacks – a prepared and informed defender is not powerless against all attacks.

References

[1] Kristen M. Altenburger, W. Philip Kegelmeyer, Ali Pinar, and Jeremy D. Wendt. A community and node attribute-corrected stochastic blockmodel. In preparation.

[2] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[3] Aaron Clauset, Mark E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70(06611), 2004.

[4] Mark C. Foehse, Terry A. Bacon, Zachary O. Benz, Richard D. Colbaugh, Kristin L. Glass, Steven N. Kempka, Cynthia A. Phillips, Ali Pinar, David G. Robinson, Smitha S. Gianoulakis, Jason F. Shepherd, Michael W. Trahan, and David J. Zage. Analytic methodology for assessing supply chains. Sandia Report SAND2012-8255, Sandia National Laboratories, October 2012.

[5] Philip Kegelmeyer, Timothy M. Shead, Jonathan Crussell, Katie Rodhouse, Dave Robinson, Curtis Johnson, Dave Zage, Warren Davis, Jeremy Wendt, Justin "J.D." Doak, Tiawna Cayton, Richard Colbaugh, Kristin Glass, Brian Jones, and Jeff Shelburg. Counter adversarial data analytics. SAND Report SAND2015-3711, Sandia National Laboratories, May 2015.

[6] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. *Lecture Notes in Computer Science*, 3733:284–293, 2005.

Appendix A

Appendix: Edge Features

The ensemble of decision trees required descriptive features for each edge. We extracted the following features:

- *NumEdgeTriangles*: The number of triangles this edge participates in. For edges well embedded in a true community, this should be relatively high.
- *EdgeTriangleDensity*: (2 * NumEdgeTriangles) / (degree(source) + degree(destination) - 2) (with adjustments to ensure no division by zero). This measures how many triangles are on the edge vs. how many are expected to be on the edge.
- *EdgeJaccardSimilarity*: The Jaccard similarity of the neighbors of source node and destination node. A similar measure to EdgeTriangleDensity.
- *PercentInSameCommunity*: The percentage of the time source node and destination node are in the same community. Measures how likely this edge is to cross communities.
- *SrcAndDstLabelsMatch*: True if the label on source node and label on destination node are the same.
- *SrcLabel*: The label for the source node (HOT, COLD, UNKNOWN).
- *DstLabel*: The label for the destination node (HOT, COLD, UNKNOWN).
- *SrcBetweennessCentrality*: The betweenness centrality for the source node: the percentage of shortest paths for all pairs of nodes that goes through source.
- *DstBetweennessCentrality*: The betweenness centrality for the destination node: the percentage of shortest paths for all pairs of nodes that goes through destination.
- *EdgeBetweennessCentrality*: The betweenness centrality for the edge: the percentage of shortest paths for all pairs of nodes that goes across the edge.
- *SrcEccentricityWithEdge*: The eccentricity of the source node when the edge is left in the graph.
- *DstEccentricityWithEdge*: The eccentricity of the destination node when the edge is left in the graph.

- *SrcEccentricityWithoutEdge*: The eccentricity of the source node when the edge is removed from the graph.
- *DstEccentricityWithoutEdge*: The eccentricity of the destination node when the edge is removed from the graph.
- *DistanceBetweenEndsWithoutEdge*: The distance between the two nodes when this edge is removed. Note that if the edge's NumTriangles > 0, this will be 2.
- *EdgeTemperatureMeanDelta*: The temperature difference between source's and destination's communities (0 if in the same community); the average of all runs of Louvain.
- *EdgeTemperatureStdevDelta*: The temperature difference between source's and destination's communities (0 if in the same community); the standard deviation of all runs of Louvain.
- *EdgeTemperatureMaxDelta*: The temperature difference between source's and destination's communities (0 if in the same community); the maximum of all runs of Louvain.
- *SrcNodeTemperatureMean*: The average temperature for source node's community across all runs of Louvain.
- *SrcNodeTemperatureStdev*: The standard deviation of the temperature for source node's community across all runs of Louvain.
- *SrcNodeTemperatureMax*: The maximum temperature for source node's community across all runs of Louvain.
- *DstNodeTemperatureMean*: The average temperature for destination node's community across all runs of Louvain.
- *DstNodeTemperatureStdev*: The standard deviation of the temperature for destination node's community across all runs of Louvain.
- *DstNodeTemperatureMax*: The maximum temperature for destination node's community across all runs of Louvain.
- *SrcAndDstTemperatureMean*: The average temperature for both source and destination's temperatures across all runs of Louvain.
- *SrcAndDstTemperatureStdev*: The standard deviation of temperature for both source and destination's temperatures across all runs of Louvain.
- *SrcAndDstTemperatureMax*: The maximum temperature for both source and destination's temperatures across all runs of Louvain.
- *RandomNumber*: A truly random number. Should never be indicative of anything.

The RandomNumber Feature is included not because we expect it to be predictive, but because in post-hoc analysis of feature importance, we can be certain that any feature less predictive than RandomNumber is generally non-predictive.

DISTRIBUTION:

1	MS 0359	D. Chavez, LDRD Office, 1911
1	MS 0899	Technical Library, 8944 (electronic copy)


Sandia National Laboratories