Detecting Malware Communities Using Socio-cultural Cognitive Mapping

Iain Cruickshank[1], Anthony Johnson[2], Timothy Davison[2], Matthew Elder[2], and Kathleen Carley[1]

[1] Institute for Software Research, Carnegie Mellon University, Pittsburgh, PA, USA
[2] Johns Hopkins University / Applied Physics Laboratory, Laurel, MD, USA
icruick@andrew.cmu.edu, anthony.johnson@jhuapl.edu

Abstract. We apply a variation of Socio-cultural Cognitive Mapping (SCM) to computer malware features explored previously by Saxe and Berlin that characterized malware binaries as benign or malicious based on 1,024 program features derived from a deep neural network-based detection system. In this work, we model the features as attributes within a latent spatial domain using a weighted consensus graph representation to visualize and analyze the malware binary communities. The data used in our analysis is extracted from a remote access trojan (RAT) family named Sakula that first appeared in 2012, and has been used to enable an adversary to run interactive commands and execute remote program functions. Our results show that by SCM we were able to identify distinct malware communities within the malware family, which revealed insights into the overall structure of the various binaries. Further work will include surveying other malware families using SCM and evaluating their efficacy in determining features that can be used to find core malware communities.

Keywords: malware analysis, social network analysis, cognitive mapping, graph learning.

1 Introduction

Cyber-attacks are a critical problem for society, and malware is used by attackers in a large class of cyber-attacks. The volume of malware produced is overwhelming, with hundreds of thousands of new samples discovered every day. The vast majority of malware samples are actually variants of existing malware, produced by transforming or obfuscating an existing sample in such a way that it can evade detection by security products and other defenses. Categorizing related malware into families and understanding communities of samples within a malware family can aid malware analysts and security operations in prioritizing responses and defenses for new malware samples.

While there are a number of existing techniques for analyzing malware [1], one approach that has not been explored significantly involves analyzing features from malware samples using social network analysis techniques.

Social network analysis (SNA) involves structural intuition based on ties linking social actors [2]. If we view malware binary samples as individual actors in a network, then we can harness many of the SNA methods to analyze commonalities within the binaries. SNA methods are useful because they are grounded in systematic empirical data, draw heavily from graph theory, and rely on the use of mathematical models. As people do not act in a manner independent of their social environment, these binary samples, which are created by people, exhibit similar traits and are subject to patterns, clusters, and structural mappings.

The remainder of this paper is organized as follows. In Section 2, we provide an overview of the malware family that is investigated and the malware analysis technique used to extract features from the malware data set. Section 3 presents the Socio-cultural Cognitive Mapping (SCM) method and Weighted Consensus Graph procedure that are applied to the malware feature data. In Section 4, we explore the SCM results in more detail, comparing the clustering results of Weighted Consensus Graph with those from the commonly used similarity analysis tool, *ssdeep* [3]. Finally, we provide some conclusions and future work.

2 The Data

The data used in our analysis comes from the Sakula family of malware which represents variations of remote access trojan (RAT) tools that seek to target victims in the aerospace, government, healthcare, and technology sectors. This RAT operates on Windows platforms and has been known by aliases Sakurel and VIPER. Many researchers have analyzed this RAT, and have noted its stealth and effectiveness [4]. Namely, Sakula allows an adversary to run interactive commands through a back-door, then download components, and execute additional components under the disguise of being legitimate software. The Sakula RAT first appeared in 2012. Since that time robust sets of labeled malware data have been available to the security community, and commercial threat intelligence has exponentially increased in light of the growing and persistent threat posed by malware [1]. Concurrent with the availability of the avalanche of threat data is the need to provide suitable means to organize, manipulate, and then analyze it.

We chose to use a feature extraction method proposed and outlined by Berlin and Saxe [5] to build our Sakula malware dataset for further analysis. This deep neural network-based extraction method uses four static feature domains to achieve high accuracy malware classification results. The first is the *byte-entropy* feature vector. It is produced by concatenating the row values of a two-dimensional histogram constructed from byte-entropy value pairs of all bytes in a given 1024-byte sliding window. New entropy values and byte-entropy pairs are computed for each window, which traverses the file using a step size of 256 bytes. This first extraction process provides byte values with an added entropy "context", which lends itself to preserve as much information as possible. Next is the *imports* feature vector. It is generated by hashing library and library function names to the range [0, 255), and measuring the frequency counts of each resulting hash value. The idea behind the use of the libraries is that it may capture the semantics of internal function calls that binary files rely on. This may allow files with

a combination of suspicious imports similar to known malware families to be detected and clustered. Third is the *printable strings* feature vector. It is produced by hashing all printable strings of length 6 or more (in the ASCII printable range) to the range [0, 16), pairing each string's hash with the log of its length, constructing a two-dimensional feature vector mapping those hash value / log-length pairs, and then concatenating the rows of the histogram. Finally, the *metadata* feature vector is produced by extracting numerical features from a file's Portable Executable (PE) packaging and aggregating those features into a 256-length array. This allows heuristically suspicious aspects of a given binary to be identified and matched against known malware families.

At the conclusion of the feature extraction process we have a comma separated values (csv) file with 1,024 feature columns and 289 rows. Each row (except the first row, which is the column header) represents a unique malware binary file. The first column identifies the malware binary number, followed by 1,024 columns of features extraction aggregates. Additional metadata columns are the binary MD5 hash, which is also the malware's file name, followed by a label assigned to the binary – in this experiment all files are labeled malware.

3 SCM Method

At a high level, the method to find the latent network of the malware was done as follows: 1) Extract features from the malware. 2) Use Socio-cultural Cognitive Mapping [6] to place each of the malware samples into a latent space by their features. 3) Use a Weighted Consensus Graph to extract the latent graph of the points in the latent space, and then analyze the latent network.

We begin with an overview of the SCM process. The first step in the SCM process is to convert the data matrix into a frequency matrix, where the entries are the counts of the shared levels of every attribute between each of the malware samples. It should be noted that all attributes are considered equal when contributing to the shared counts; there is no distinction between attributes. So, the frequency matrix is of size number-of-samples by number-of-samples, and has entries that are the counts of the number of same attribute values for all attributes of the data matrix between each sample and every other sample.

Next, we find the positions in a latent space for each of the data points such that the function determining the distances between points produces fitted frequencies that are as near as possible to the observed frequencies from the previous step. The fitted frequencies for each data point are calculated by:

$$F(i,j) = R_i \times C_j \times 2^{-d_{ij}^a} \tag{1}$$

where i is the row factor term, j is the column factor term, and d is the row and column correlation factor, where a is an attenuation term. The term a controls how much impact the interaction between two entities has on their co-occurrence frequency. A smaller a result in the distance affecting the frequency counts less as the distance between any given data points increases. When $a = 2$ the interaction between row and

column variables is modeled as a Gaussian Distribution. The distance metric, *d*, is defined as the *Minkowski* distance:

$$d_{ij} = \left(\sum_{k} |x_{ik} - y_{jk}|^{M} \right)^{\frac{1}{M}} \tag{2}$$

where *M* is the power setting of the metric that determines the space that the points are mapped into (i.e., *M*=2 produces a Euclidean distance and space). So, the SCM process then proceeds as an optimization problem where the points of each sample are inferred in a latent space with the target function as the χ^2-value between the fitted frequencies, *F(i,j)* and the observed frequencies, for user-supplied attenuation, power, and number of dimensions. The overall process of the SCM can be summarized as follows (Figure 1 further illustrates the SCM procedure):

1. Select a set of power and attenuation settings and a number of dimensions for the SCM procedure. Typical selections are $a = \{0.7,1,2,3\}$ and $M = \{0.7, 1, 2, 3\}$, and two dimensions for the inferred points of the agents.
2. For each combination of the parameters of *a* and *M*, do the following:
   a. Place each data point in the specified number of dimensions, dimensional space, and calculate the frequencies of this placement using equations aforementioned.
   b. Evaluate the current placement using the result of the last step and observed frequencies using a χ^2-value.
   c. Continue to adjust the placements of the points to achieve a better χ^2-value.
3. Return the coordinates of those placements of the points that have the best χ^2-value.

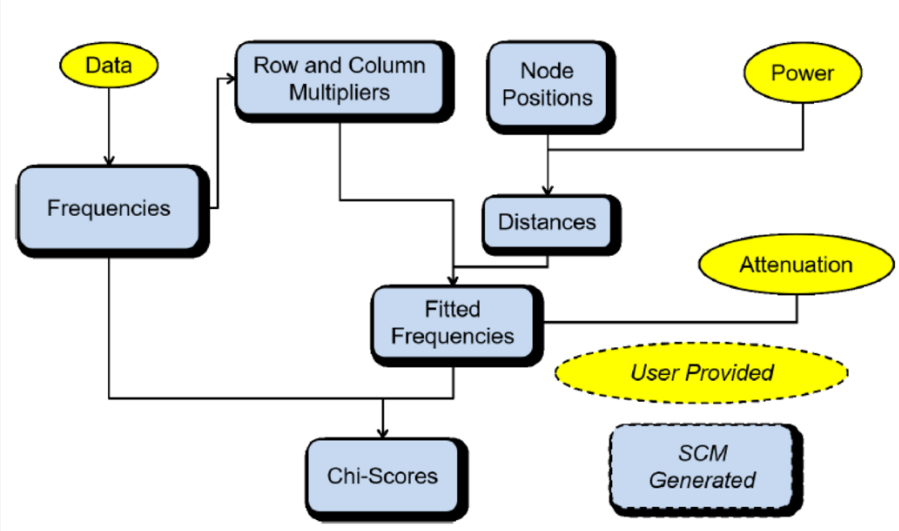

Fig. 1. Flowchart of the SCM process. User inputs are the number of dimensions, power, and attenuation. Outputs are the latent points of the nodes and the distances between the nodes in the latent space.

Having obtained the latent positions of each node, we can then obtain a pairwise distance matrix. Using this pairwise distance matrix, we extract the latent graph by the Weighted Consensus Graph, which is derived from the Consensus Graph [7]. At a high level, the Consensus Graph takes multiple k-Nearest Neighbor (k-NN) graphs and creates a graph, *C*, that is a count of all of the times that two nodes are in the same k-NN of any other node. The Consensus Graph procedure has seen a wide variety of uses and remains one of the prominent procedures for extracting a latent network using arbitrary similarity measurements [7]. One drawback of the Consensus Graph is that each time two nodes occur in the same k-NN of a third node, they always just receive a `+1' to their counts. Thus, in the case of the third node acting as a boundary spanner between two clusters of nodes, those clusters of nodes will end up being erroneously densely connected by the Consensus Graph procedure. In order to remedy this, and to have a method that has even greater sensitivity to local structures in the latent graph, we propose a Weighted Consensus Graph. Figure 2 is a simple illustration that demonstrates how the Weighted Consensus Graph can provide a more cluster sensitive procedure than the Consensus Graph. In the 3-NN graph, nodes *u* and *v* would receive a plus 1 to their consensus graph count, as they both share node *w* as a 3-NN neighbor. However, in the weighted consensus graph approach the fact that nodes *u* and *v* only share *w* as a neighbor in the 3-NN is considered, and so they only receive a plus one-third into their consensus graph count.

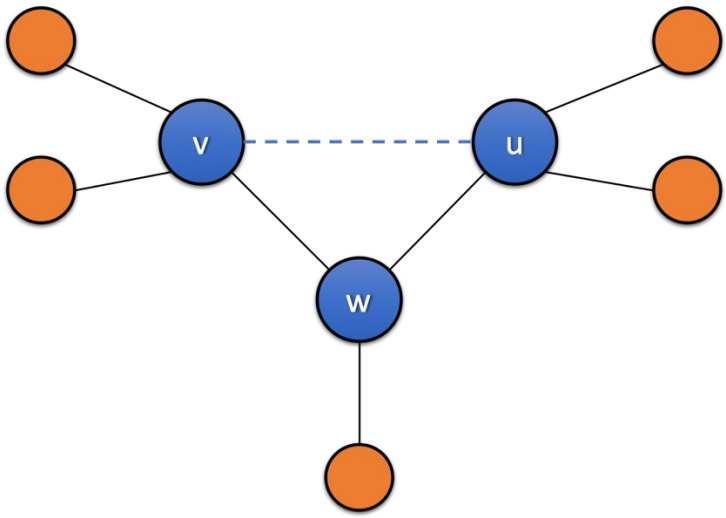

Fig. 2. Example 3-NN Weighted Consensus Graph.

In the Weighted Consensus Graph procedure, each co-occurrence between two nodes in the k-NN of a third node now gets a score addition based on the local neighborhoods of the two nodes at that particular k. In order to assess how much a score addition an edge between two nodes that share a neighbor in a k-NN gets, we turn to the Weighted Jaccard Index, which is formulated as:

$$J(u,v) = \frac{\sum \min\left(N(u), N(v)\right)}{\sum \max\left(N(u), N(v)\right)} \tag{3}$$

where *u* and *v* are two nodes in the latent graph, and *N(u)* and *N(v)* are their respective neighborhoods such that an entry in *N(u)* is the weight of the link from node *u* to all of the nodes to which it connects. The *min* and *max* operations are pairwise operations on the vectors of the neighborhoods of the two nodes. The use of a Jaccard index has demonstrated great utility in the network sampling literature as a means of assessing local community structure and as a basis for sampling to preserve the network structure [8], [9]. Thus, our proposed algorithm works analogously to the original Consensus Graph method, with the exception that each co-occurrence of two nodes in a k-NN of a third node gets an addition of the Weighted Jaccard Index between those two nodes. Since the Weighted Jaccard Index can be interpreted as the probability of an edge existing, given its endpoints, we also introduce a parameter, ε, which is a hyperparamter that can exclude non-probable edges from being included in the final graph. In this way, out method does not rely on any global thresholding after the graph's construction to remove erroneous edges, which is common with many graph learning techniques to include the consensus graph. Instead we dynamically remove erroneous edges as they are identified as being erroneous by a low probability of occurrence. The pseudo-code of our Weighted Consensus Graph procedure is detailed Algorithm 1, shown in Figure 3.

Algorithm 1 Weighted Consensus Graph

input: Distance or Affinity Matrix, $S \in R^{(N \times N)}$. Dynamic threshold parameter, $\epsilon \in [0, 1)$
output: Weighted Consensus Graph Adjacency Matrix, $C \in R^{(N \times N)}$
for $w = 1 : N$ **do**
  Obtain $kNN(w)$ by applying k-nearest neighbor rule to S
  for $u = 1 : N$ **do**
    for $v = u + 1 : N$ **do**
      if $u \in kNN(w) AND v \in kNN(w)$ **then**
        compute $J(u, v)$
        if $J(u, v) > \epsilon$ **then**
          $C(u, v) \leftarrow C(u, v) + J(u, v)$
          $C(v, u) \leftarrow C(v, u) + J(u, v)$
        end if
      end if
    end for
  end for
end for
$C \leftarrow \frac{C}{\sum_i^N C(i,:)}$
return C

Fig. 3. Algorithm 1: Weighted Consensus Graph.

In Algorithm 1, the final assignment to *C* is to row-normalize *C*. At the end of the Weighted Consensus Graph, we are left with a weighted, symmetric graph that represents the manifold of the data. From there, we employ the simple, yet effective, network

clustering technique of the Louvain Method [10]. It should be noted that any network clustering algorithm could be applied to the latent network, and that we choose to use Blondel et al.'s method do its ubiquity in various coding languages and proven performance.

For the malware, we choose to threshold on a per-edge basis, with a value of 0.3. This is to say, we only added *C* to the edge score if the edge score was greater than 0.3; an edge must have a probability of occurring, given its endpoints, greater than 0.3. We also selected the k-value as:

$$k = \lceil 2^{(log_2 N)} \rceil \tag{4}$$

where *N* is the number of malware samples.

4 SCM Results

Generally, the latent network of the malware samples displayed distinct, heterogenous sub-communities. First, the samples have a strong community structure, with many more inter-community than intra-community edges. Second, the malware samples have overlapping subgroups. Third, there is one possible outlier subgroup and only two malware samples that do not lie strongly within any group. Figure 4 is the latent network of the malware (coloring by *Louvain Modularity* subgrouping).

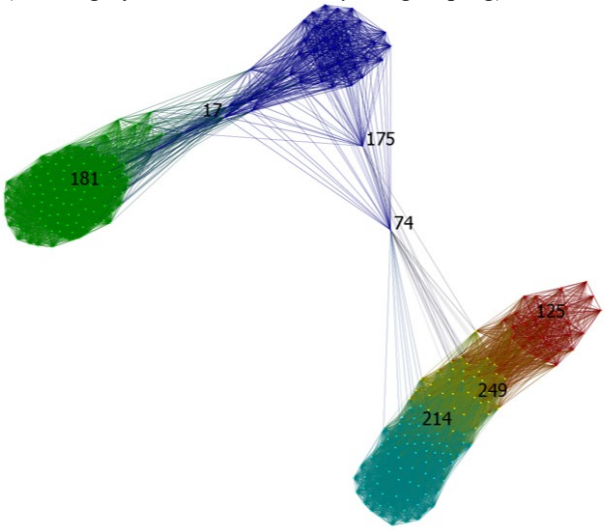

Fig. 4. Louvain Modularity of the latent network of malware.

Those nodes that are weak members of any group, 74 and 175, were also the highest in centrality betweenness. It is likely these were anomalous samples with respect to the other malware samples; they likely shared not only feature closeness to their assigned subgroup (dark blue), but also feature closeness to other subgroups. Of the 5 subgroups

found by Louvain, the most central node of the green was 181, of the light blue was 214, of the dark blue was 17, of the yellow was 249, and of the red was 125. These were likely prototypical nodes for each of their respective subgroups. It is interesting to note that 17, while the most central node of the dark blue subgroups, was also highly connected to the green subgroup. It is likely that if the network were further sparsified, 17 would end up in the green subgroup and there would be a small, disconnected outlier cluster made up of remnants of the dark blue subgroup.

As for the relationships between those subgroups found in the latent graph, there were two strongly modular subgroups, and three overlapping subgroups. The green and dark blue subgroups had 95.7% and 76.5% percent of their edges internal, respectively. So, while the subgrouping was done by Louvain, it was likely these subgroups would remain unchanged under any other subgrouping method. In contrast, the light blue, yellow, and red subgroups had 63.9%, 25.4%, and 40.7% of their edges internal, respectively. Thus, these subgroups were not as well defined as the other two and likely had a fair amount of overlap in the features between their respective constituents. Additionally, the yellow subgroup had ties between the light blue and the red subgroups, but the red and yellow subgroups had almost no ties between them. This in-between subgroup was likely a result of those samples belonging to either the light blue or red subgroups, but having some attributes from the other subgroup. This idea is further supported by the very few internal links for the yellow subgroup. Thus, the data in those three subgroups likely have some overlap in some feature space, and those samples which most overlap in the feature space were members of the yellow subgroup.

In order to gain better insight into the learned latent network, we then compared the network produced by SCM and Weighted Consensus Graph to the data found by a commonly used similarity analysis tool for malware analysis, *ssdeep* [3]. In order to create the latent network and find subgroups from the *ssdeep* similarity score values, we created a pairwise network of all the samples, where each edge was weighted by the *ssdeep* score comparing the two sample hashes. Figure 5 depicts the network created by the *ssdeep* scores.

Much like the network found by SCM and the Weighted Consensus Graph, there were some strongly clustered subgroups within the data, as well as an overlapping subgroup (colors red and light blue). In order to compare the subgroups found within the two networks, we evaluated both networks using the Louvain method where the nodes were entered into the algorithm in a non-random order. The subgroups had a reasonably high degree of overlap, with an Adjusted Mutual Information value of 0.7103. In particular, the green subgroup from the SCM and Weighted Consensus Graph and the brown subgroup from the *ssdeep* network very closely overlapped. One subgroup of major difference between the two methods, however, was the dark blue subgroup of the SCM and Weighted Consensus Graph, which was a cohesive subgroup in the latter network, but whose members were mostly a collection of isolates and very small components in the *ssdeep* network. Furthermore, the SCM and Weighted Consensus Graph and *ssdeep* networks themselves bear little relation. Using the Quadratic Assignment Procedure, we found the two networks had a correlation value of only 0.317 and weak ability to regress on each other with an $R^2 = 0.10$. Thus, while the two networks did produce very similar subgrouping results, they also had different topologies, with the

SCM and Weighted Consensus Graph having greater connectivity and heterogeneity in degree distribution.

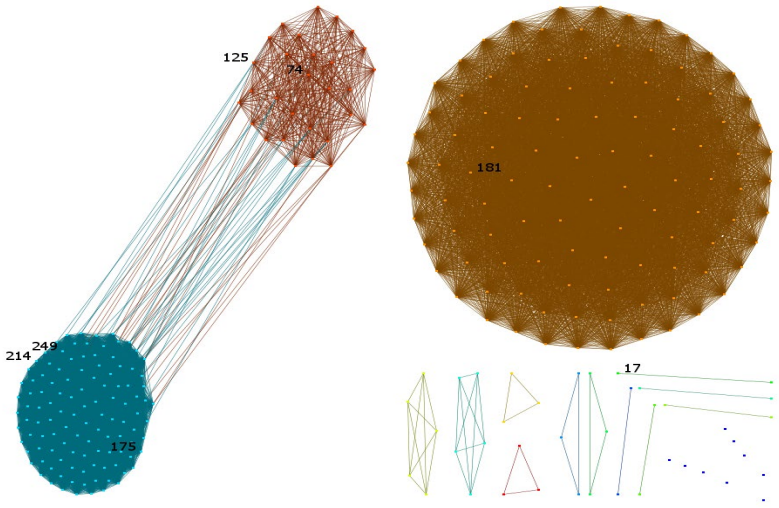

Fig. 5. Network structure created using *ssdeep* similarity scores.

Turning to look at the relationship among features, there was not as strong of a clustered network structure present in the latent graph. When SCM was applied to each of the malware features, wherein each of the malware samples now becomes the feature of evaluation, the SCM plus Weighted Consensus Graph procedure was not able to find easily distinguishable clusters or a distinct network topology beyond a uniformly, densely-connected network. This was likely a result of the features having similar manifestations across the malware samples. It is interesting to note, however, that if you cluster the latent network of the features, in the same way as was done with the samples, you produce 9 distinct subgroups. This is more than the default sets of features, which was 4 (256 features from each of 4 categories of: Byte-Entropy Histogram, PE Import, String 2D histogram, and PE Metadata). Furthermore, the found Louvain subgroups did not align with or within the default features and have a low Adjusted Mutual Information Score between them at 0.0375. Thus, it would seem that different parts of the malware feature sets were activated by different malware samples. So, while it was difficult to extract meaningful networks from the features by SCM and Weighted Consensus Graph, it did illustrate that the features do manifest differently for different malware samples.

5 Conclusions and Future Work

This initial exploration demonstrates the application of a social network analysis technique, Socio-cultural Cognitive Mapping, to analysis of a malware family (Sakula). Future work includes further analysis of the results to determine the underlying features

that drive clustering using the Socio-cultural Cognitive Mapping and Weighted Consensus Graph network structure. We also plan to compare the results to other similarity analysis tools used for malware analysis beyond *ssdeep*. We will investigate the application and analysis of additional malware families to explore whether these results generalize in additional future work.

Acknowledgments

This work was funded by the Minerva Research Initiative and is sponsored by the Department of the Navy, Office of Naval Research under ONR award number N00014-18-1-2111 and the National Science Foundation Graduate Research Fellowship (DGE 1745016). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research or the NSF.

References

1. Y. Ye, T. Li, D. Adjeroh and S. S. Iyengar, "A Survey on Malware Detection Using Data Mining Techniques," ACM Computing Surveys, vol. 50, no. 3, pp. 41:1--41:40, June 2017.
2. I. McCulloh and A. Johnson, Social Network Analysis with Applications, Hoboken, NJ: Wiley Publishing, 2013.
3. J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," The Digital Forensic Research Conference, no. 3, pp. 91-97, 2006.
4. Dell SecureWorks Counter Threat Unit, "Sakula Malware Family," SecureWorks, July 2015. [Online]. Available: https://www.secureworks.com/research/sakula-malware-family. [Accessed April 2019].
5. J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in 10th International Conference on Malicious and Unwanted Software (MALWARE), 2015.
6. G. P. Morgan, J. Levine and K. M. Carley, "Socio-Cultural Cognitive Mapping," in Social, Cultural, and Behavioral Modeling, Springer International Publishing, 2017, pp. 71--76.
7. V. Premachandran and R. Kakarala, "Consensus of k-NNs for Robust Neighborhood Selection on Graph-Based Manifolds," in 2013 IEEE Conference on Computer Vision and Pattern Recognition, 2013.
8. L. Qiao, L. Zhang, S. Chen and D. Shen, "Data-driven graph construction and graph learning: A review," Neurocomputing, vol. 312, pp. 336-351, 2018.
9. V. Satuluri, S. Parthasarathy and Y. Ruan, "Local Graph Sparsification for Scalable Clustering," in Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, 2011.
10. G. Lindner, C. L. Staudt, M. Hamann, H. Meyerhenke and D. Wagner, "Structure-Preserving Sparsification of Social Networks," CoRR, vol. abs/1505.00564, 2015.
11. V. D. Blondel, J.-L. Guillaume, R. Lambiotte and E. Lefebvre, "Fast unfolding of communities in large networks," Journal of Statistical Mechanics: Theory and Experiment, vol. 2008, p. 10008, 2008.