Graph-Based Clustering for Anomaly Detection in Network Data

Nicholas Yuen[1], Dr. Enyue Lu [2]

[1]Kean University NJCSTM,

[2]Salisbury University Department of Mathematics and Computer Science

Abstract

Currently, large amounts of data are transported over networks and require protection from breaches in security. Thus, Network Intrusion Detection Systems help to quickly alert the necessary personnel when the time comes. Because network data show the connections among multiple sources of interaction around the world, graphs are used to represent those connections. Since network data can be represented as graphs, clustering operations can be performed to detect anomalies. With multiple clustering algorithms to choose from, each one can be implemented, tested, and compared for efficiency and accuracy. Clustering algorithms are the most important component for anomaly detectionrk data, and as a result should be intensely examined. This study focused on an overview of the comparison between the Barycentric Clustering Algorithm and the Markov Clustering Algorithm (MCL). In addition, an investigation of possible improvements and approaches to the implementation of MCL was conducted.

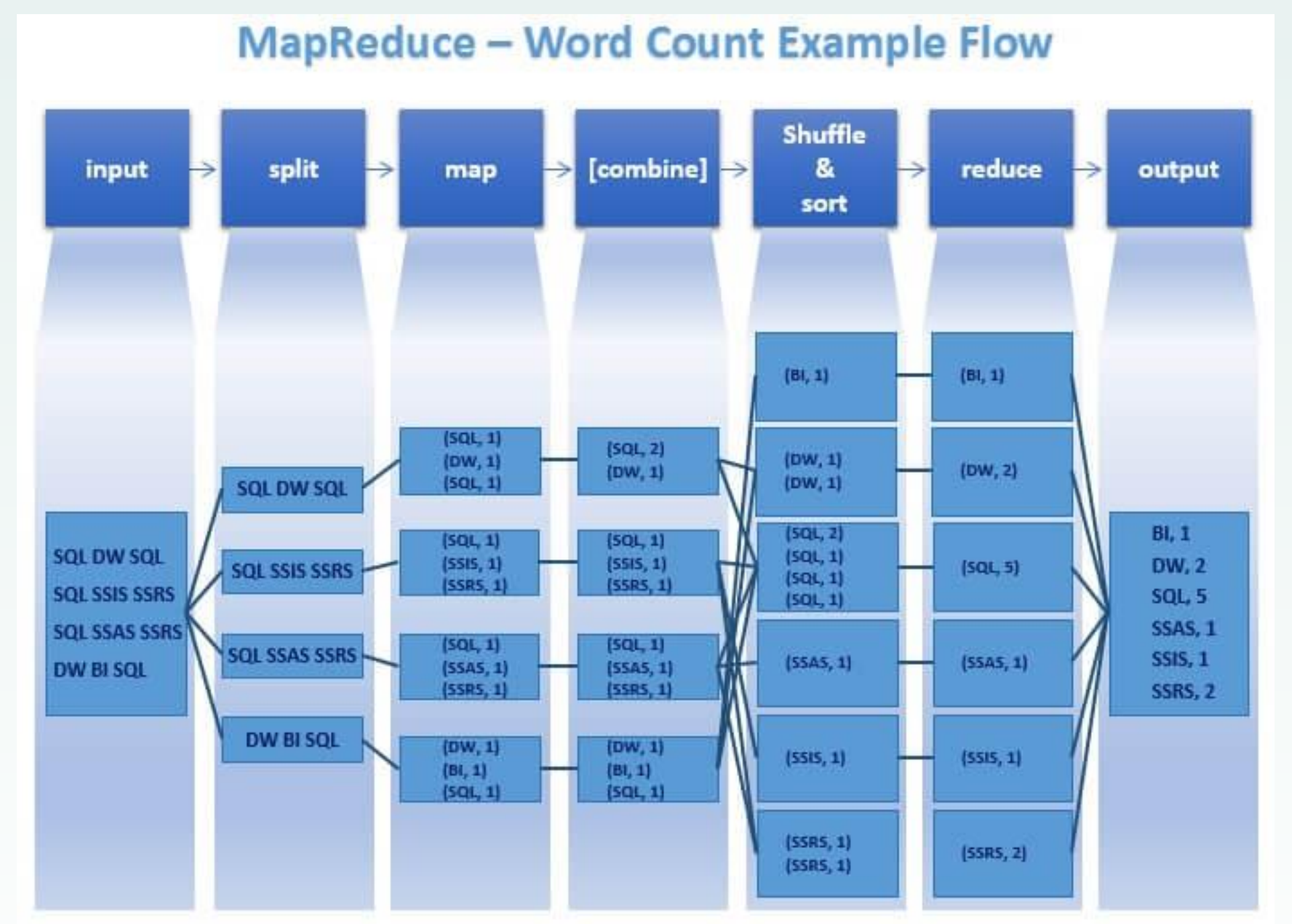
Introduction

The need for network security has become more indispensable than ever with the increasing amounts of transmitted data. To monitor the network data, Network Intrusion Detection Systems have been implemented to detect anomalous behavior during those transmissions. These unsupervised machine learning techniques can help accurately and quickly provide alerts when such behavior is detected. This approach is necessary with the constant evolution of anomalies attempting to avoid systems that cannot adapt to these new varieties of intrusions. With the rising numbers of network attacks, Network Intrusion Detections Systems will be more important to notice those subtle nuances in the network behavior.

Although certain clustering algorithms have been developed and improved throughout time, there has been little focus as to how to parallelize the process to even further improve the detection speed. With faster detection speeds, action can be taken much sooner to prevent possible attacks from outside entities. Even though accuracy is an important measure of how well an algorithm performs, speed is also essential to this study because the ultimate goal is to develop a detection system to be implemented for real time use. To replicate that flow of data in this study, the 1999 KDD Cup Dataset, the most widely used dataset for network intrusion detection, was read from a text file instead of coming from some network directly. To maximize the quickness of anomaly detection in the data, Hadoop MapReduce was used to parallelize the process for faster computations.

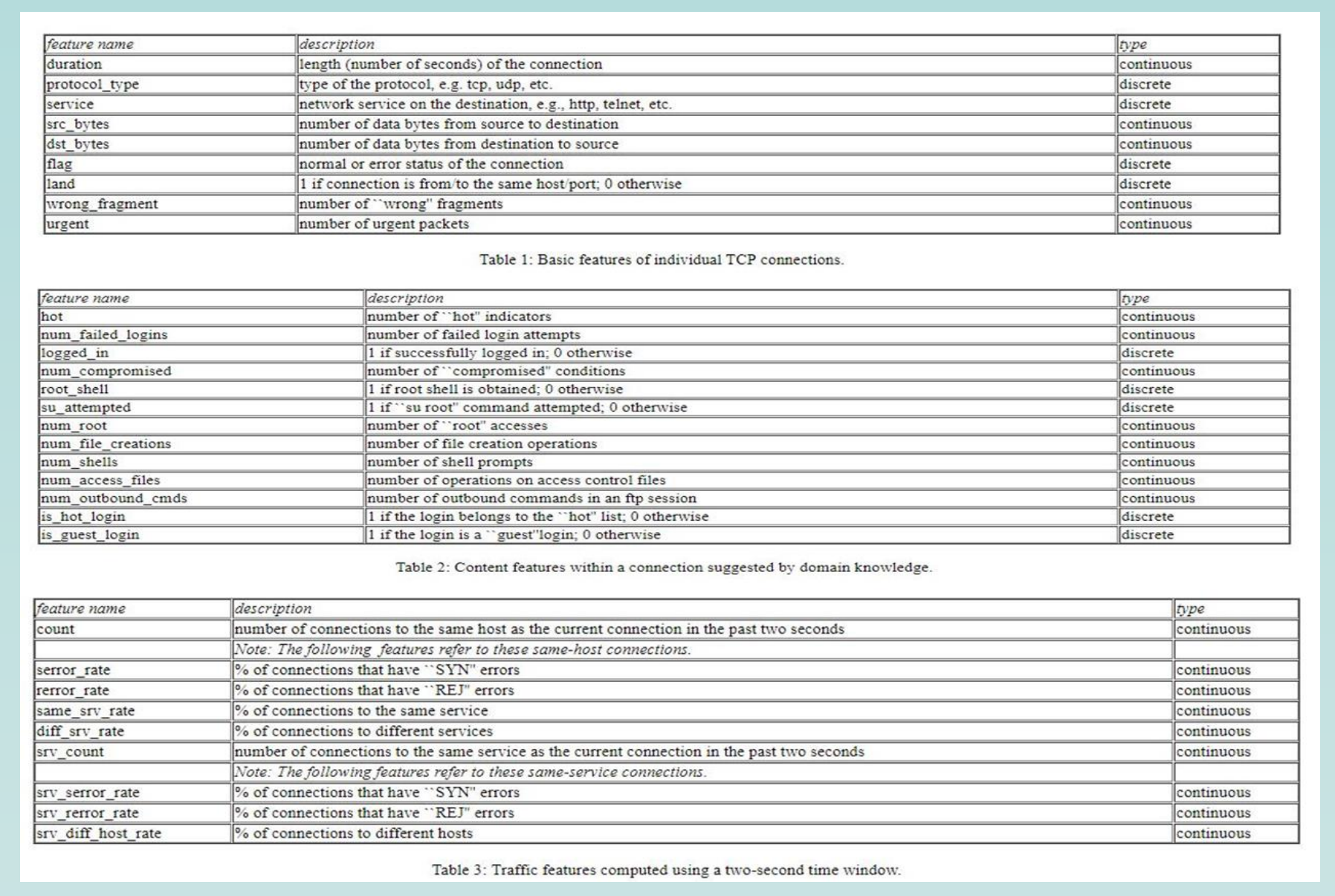
Hadoop MapReduce

For this project, the Hadoop MapReduce framework was used to help with parallelizing the overall process of clustering the network data. This framework works well with processing large data sets in a distributed way. Normally a user would need to manage every aspect of parallelization, but MapReduce is unique in the way it handles all those background operations with the help of a handful of helpful commands. As the name suggests, MapReduce has two primary functions: Map and Reduce. MapReduce works with data sets by converting it into the format of <Key, Value> whenever reading input or emitting pairs as output records. The Map function reads in some entries from input and creates an intermediate data set which is then shuffled before the Reduce function to group the similar entries. Then the Reducer function takes that shuffled intermediate data, compacts the similar entries into a single entry, then outputs the collection of compacted entries which could possibly serve as the input for another MapReduce process. Although MapReduce allows for easier parallelized implementation of code, it consequently has some overhead with every Map and Reduce function.



Network Dataset

All experiments carried out throughout the program used the 1999 KDD Cup Dataset for computational tasks and analysis. This is because the 1999 KDD Cup Dataset has been the standard for network data experiments by scientists in this field of study. Each record in the dataset has 41 unique features that are either continuous or discrete. Some of those features are *protocol_type* (type of protocol), *src_bytes* (number of bytes going from the source to the destination), and *srv_diff_host_rate* (percentage of connections to different hosts) [9]. The combination of discrete and continuous features in addition to the scale of the data poses a challenge to the clustering operation to handle both types of values in each network record.



feature name	*description*	*type*
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of ``wrong'' fragments	continuous
urgent	number of urgent packets	continuous

Table 1: Basic features of individual TCP connections.

feature name	*description*	*type*
hot	number of ``hot'' indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of ``compromised'' conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if ``su root'' command attempted; 0 otherwise	discrete
num_root	number of ``root'' accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the ``hot'' list; 0 otherwise	discrete
is_guest_login	1 if the login is a ``guest''login; 0 otherwise	discrete

Table 2: Content features within a connection suggested by domain knowledge.

feature name	*description*	*type*
count	number of connections to the same host as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-host connections.	
serror_rate	% of connections that have ``SYN'' errors	continuous
rerror_rate	% of connections that have ``REJ'' errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-service connections.	
srv_serror_rate	% of connections that have ``SYN'' errors	continuous
srv_rerror_rate	% of connections that have ``REJ'' errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

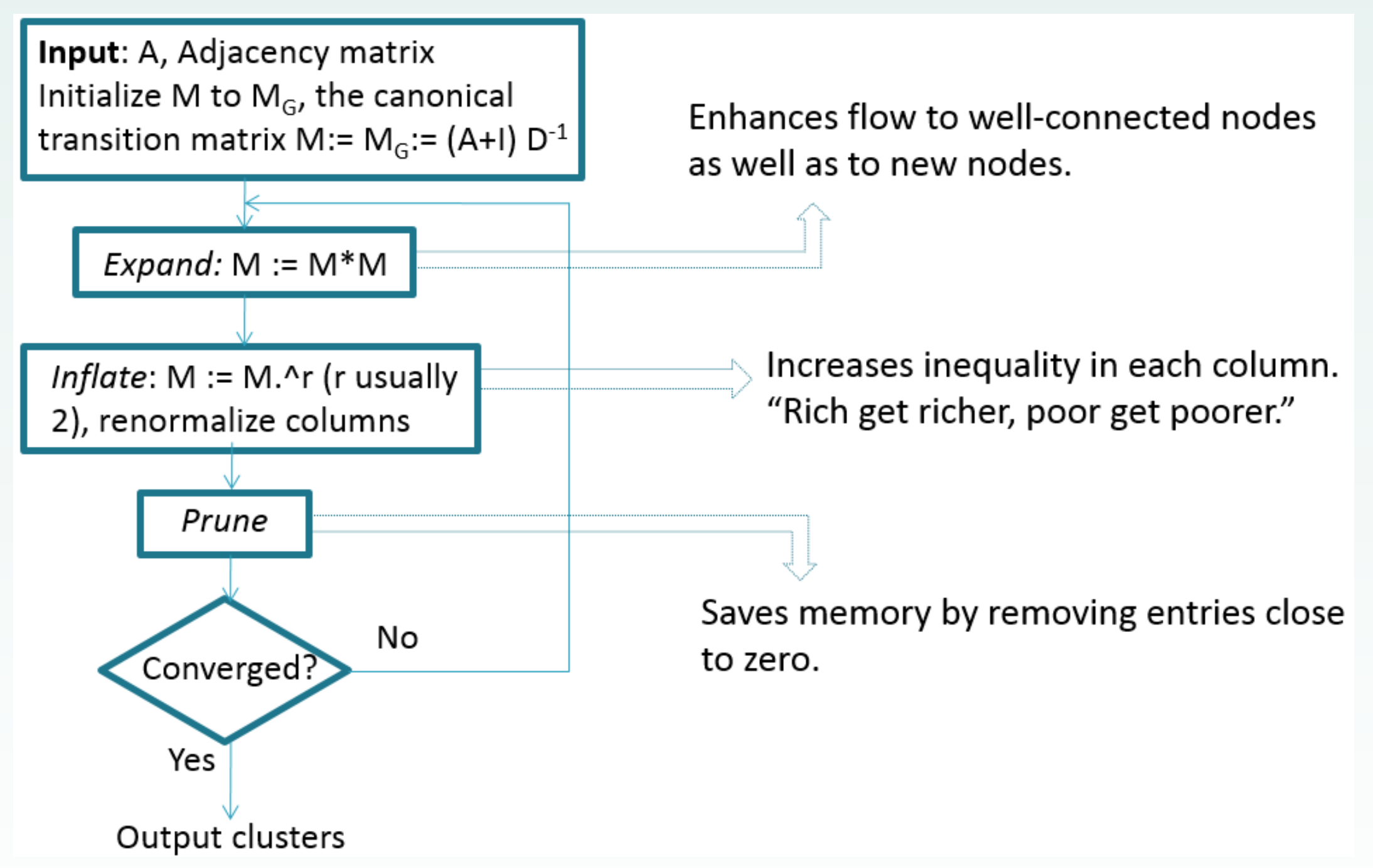
Table 3: Traffic features computed using a two-second time window.

Barycentric Clustering

The Barycentric Clustering Algorithm is influenced by Hooke's Law which is based on physical spring behavior. Hooke's Law states that there is a linear proportionality between some force needed to compress or extend a spring and that resulting distance the spring will compress or extend. Barycentric Clustering considers that concept and provides some additions to Hooke's Law to be performed on undirected graphs. Firstly, all the edges connecting the vertices of the graph are imagined to be springs. By doing so, the locations of all the vertices can be randomized and then released to see where they move after each iteration based on the forces they experience from their neighbors. Once each trial is done, the length between each vertex is measured and recorded before another trial is started. After all the runs are completed, the average length for each edge is calculated. Those average edge lengths are then compared to one another and contribute to a score for each edge. Edges with high scores are deemed inter-cluster edges and are then erased. After all the inter-cluster edges are deleted, only the clusters will remain and the process will terminate [2].

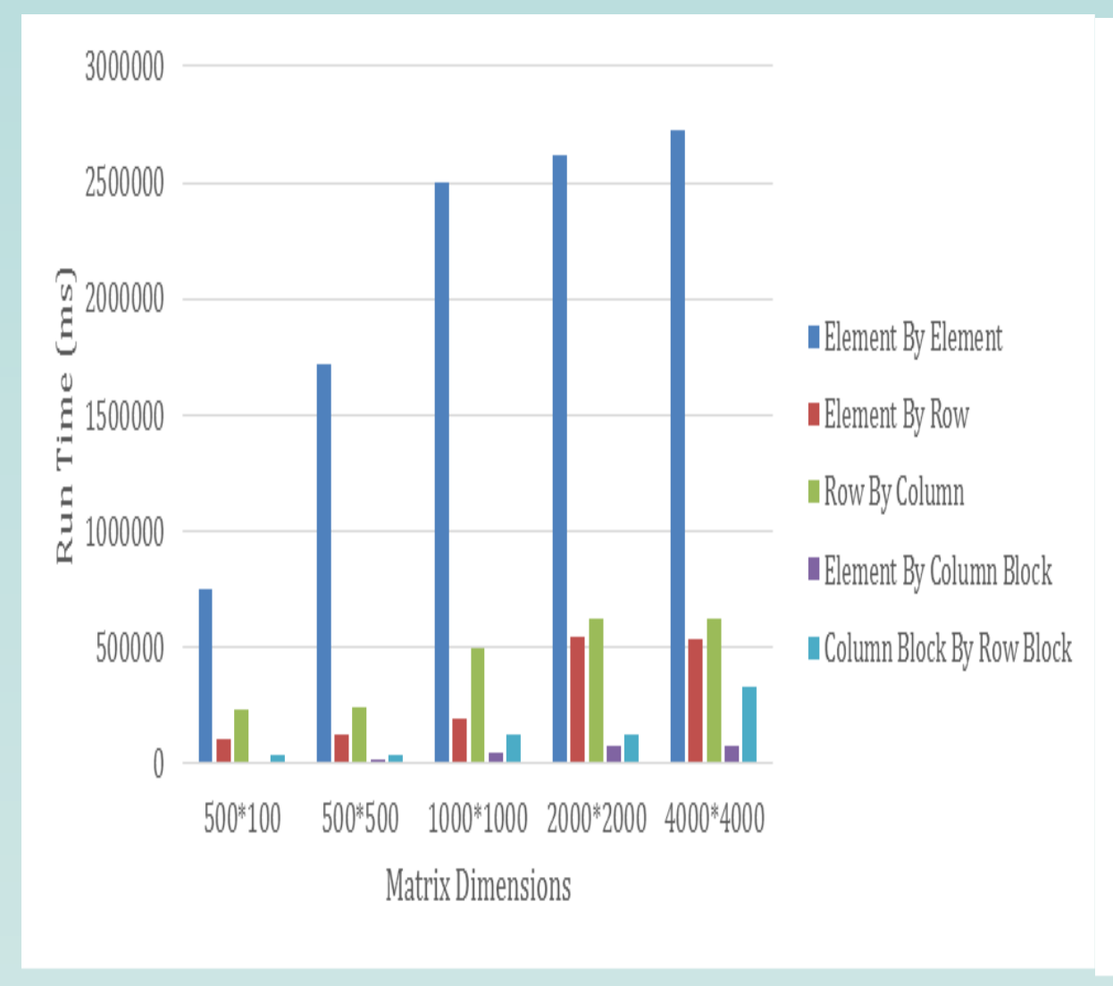
Markov Clustering

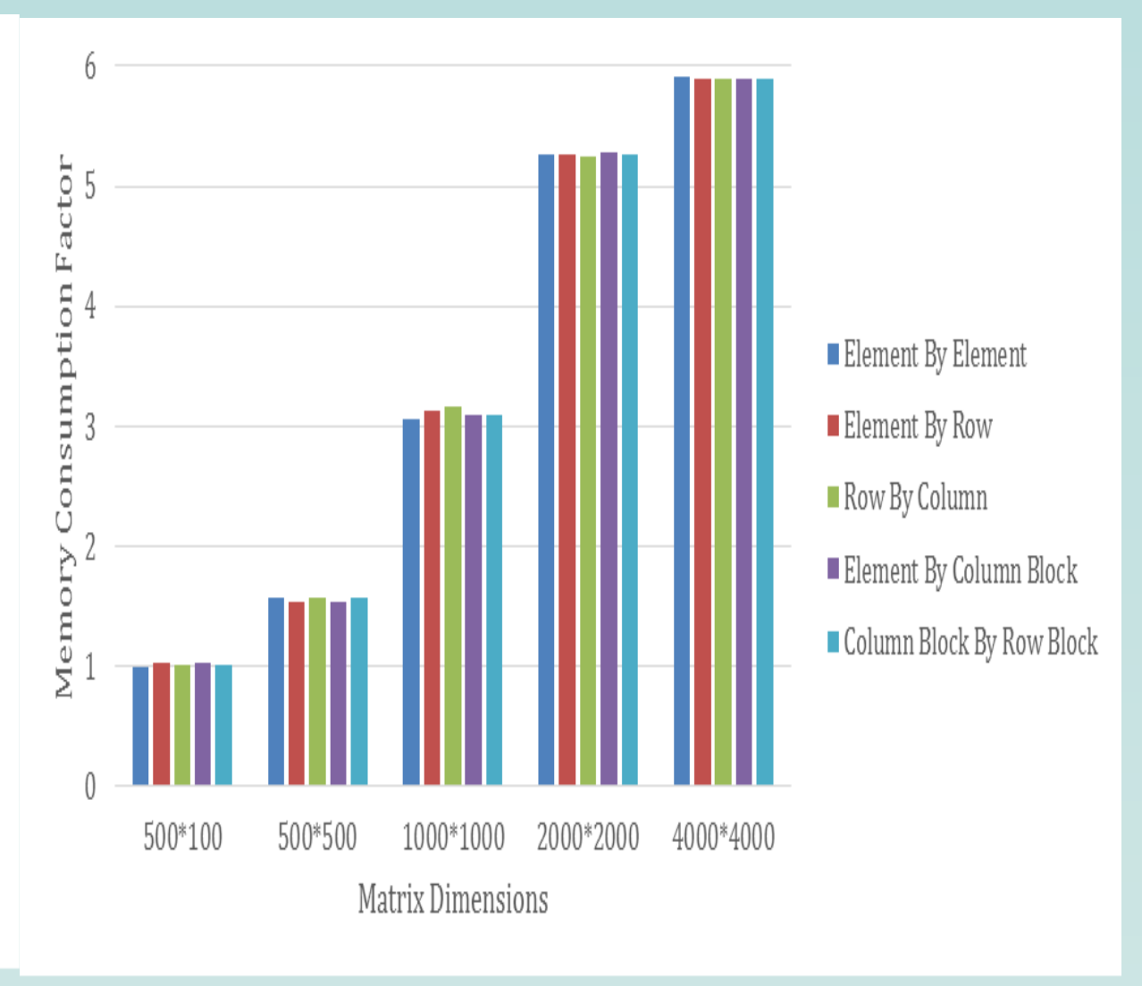
Unlike the Barycentric Clustering Algorithm, Markov Clustering uses a network flow approach. Each node is given some flow to start with. After each iteration, a percentage of flow from each node travels to each of its neighbors based on weighted probability that the flow will travel from one vertex to another. These interactions are represented as Markov Matrices and over time become idempotent. If the matrix is idempotent at the end of an iteration, then the matrix has converged and clustering can stop. An image is included below to visually represent the algorithm.


Input: A, Adjacency matrix
Initialize M to M_G, the canonical transition matrix M:= M_G:= (A+I) D^{-1}
Enhances flow to well-connected nodes as well as to new nodes.
Expand: M := M*M
Inflate: M := M.^r (r usually 2), renormalize columns
Increases inequality in each column. "Rich get richer, poor get poorer."
Prune
Saves memory by removing entries close to zero.
Converged?
No
Yes
Output clusters

Matrix Multiplication In Hadoop

With demand for constant improvements for speed and memory management, Kadhum et al. [5] proposed a relatively new approach to matrix multiplication in addition to prior work done by Deng & Wu [3] to reduce the amount of MapReduce jobs down from two to one. Instead of following the traditional format of having the first job process the data and the second one do all of the computational tasks, the authors proposed that a preprocessing step replace the first MapReduce job to reduce the overall amount of data and I/O processing overhead. In addition, they also observed from previous work that element-by-element matrix multiplication was not efficient and took a considerable amount of time and in response proposed that an element-by-block scheme be implemented instead to reduce the amount of necessary mappers. Using this scheme, Kadhum et al. proposed that with the single MapReduce job, the Map task would read the preprocessed data from a file and conduct the matrix multiplication and the Reduce task would aggregate all of the computations into a resulting matrix.





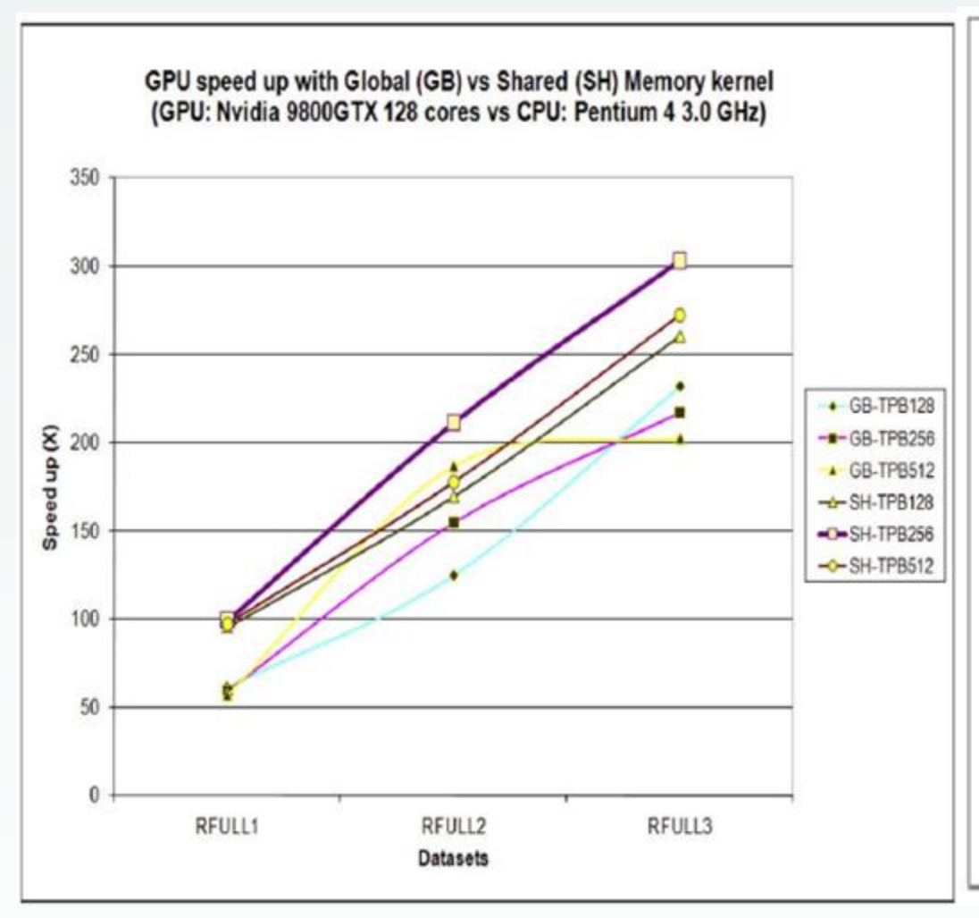
Matrix Multiplication In CUDA

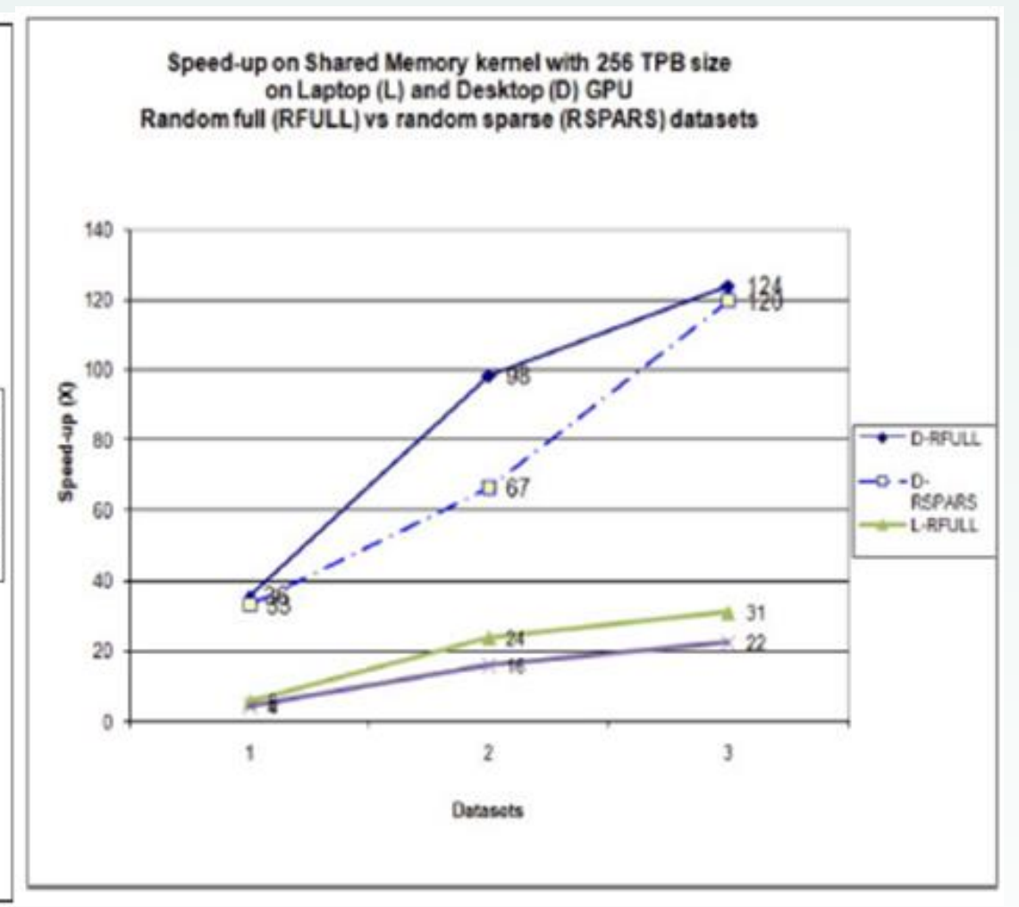
An implementation of MCL uses CUDA where kernels are used in parallel to execute the required tasks. Bustamam et al. [1] proposed that a kernel be designated to each computational task within MCL. One kernel would be responsible for executing the expansion step, another for the inflation step, and the final one to compute the chaos value for the system to determine whether the clustering is done. Before, it was mentioned that clustering was completed if the resulting matrix was idempotent. However, Bustamam et al. calculated a global chaos value by aggregating the chaos values of the columns to support the theory that an idempotent matrix would describe the completion of the clustering operation. Chaos values were calculated by using the equations below.

$$(chaos)_k = \frac{max((\Gamma_r M)_{ik}, i = 1 \ldots m)}{\sum_{i=1}^{m} (\Gamma_r M)_{ik}^2}$$

$$glb\_chaos = max((chaos)_k, k = 1 \ldots n)$$

Bustamam et al. then tested their CUDA-MCL implementation on a dataset of protein-protein interactions represented as a graph where each protein is a vertex and the interaction between two proteins is the edge connecting the vertices. They first conducted preliminary tests to determine whether global or shared memory and the number of threads per block (TPB) in each kernel would be the most efficient to maximize the speedup between GPU and CPU computation. By using the 128 core GPU SC-9800GTX machine, the authors showed that a 300x speedup was obtained by using 256 TPB in shared memory. The results of all combinations of memory kernels and TPB are shown in Graph 3 below.





Measuring Performance

To analyze how well the clustering algorithm performed, there is a need for some metrics to be calculated. Commonly, records are classified as either normal or intrusive after the clustering process has been completed. However, since clustering is an unsupervised method, there are bound to be both correctly and incorrectly classified records whether they be either normal or intrusive. This gives rise to labeling each of them as *true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN)* [4]. True says that a record was correctly classified while false says it was incorrectly identified. In combination with either true or false, positive determines a record to be an intrusion and negative classifies a record as normal. There are many metrics mentioned in [4] which could be used to examine the performance based on the various classification of records, but the one which mostly matters is the accuracy calculated with the use of equation 3 below

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Conclusion

For the entirety of the program, an investigation of the comparison between Barycentric Clustering and was thoroughly conducted through the consultation of many articles. Because the Barycentric Clustering Algorithm had already been implemented and tested by previous REU attendees, the focus then turned on what the Markov Clustering Algorithm offered. However , by the time all preparations were made, the program reached a stage when there was not going to be enough time implement and test any changes. Therefore the attention shifted towards theoretical thought of possible future approaches which may yield a better accuracy than that from Barycentric Clustering. One of those ideas was to replace the first MapReduce cycle with a preprocessing step to reduce the overall overhead. This showed a significant speedup with sparse graphs of varying dimensions and shows promise of a possible approach for dense matrices like those used in the Markov Clustering Algorithm. Another way to minimize overhead would be to use CUDA to implement the Markov Clustering Algorithm. Because CUDA uses the memory within the GPU to do computations, there is less latency overall. By suggesting some ways to reduce the run time for a possible clustering algorithm which is more accurate, it is safe to conclude that implementing MCL on CUDA would be the most efficient algorithm for speed and accuracy over Barycentric Clustering.

References

[1] A. Bustamam, K. Burrage, and N. A. Hamilton, "Fast parallel Markov clustering in bioinformatics using massively parallel computing on GPU with CUDA and ELLPACK-R sparse format," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB),* vol. 9, no. 3, pp. 679-692, 2012.
[2] J. Cohen, "Barycentric graph clustering," *Oregon Health Science University,* 2008.
[3] S. Deng and W. J. I. Wu, "Efficient Matrix Multiplication in Hadoop," vol. 13, no. 1, pp. 93-104, 2016.
[4] B. Joyce, E. Lu, and M. K. Gobbert, "Graph Based Anomaly Detection using MapReduce on Network Records," ed, 2017.
[5] M. Kadhum, M. H. Qasem, A. Sleit, and A. Sharieh, "Efficient MapReduce matrix multiplication with optimized mapper set," in *Computer Science On-line Conference*, 2017, pp. 186-196: Springer.
[6] C. McNeill, E. Lu, and M. Gobbert, "Distributed Graph-Based Clustering for Network Intrusion Detection," in *Extended Abstract, Companion of SC: IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SuperComputing)*, 2016.
[7] C. C. Noble and D. J. Cook, "Graph-based anomaly detection," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 631-636: ACM.
[8] J. Norstad, "A mapreduce algorithm for matrix multiplication," ed, 2009.
[9] Kdd.ics.uci.edu. (1999). *KDD-CUP-99 Task Description*. http://kdd.ic.uci.edu/databases/kddcup99/task.html.
[10] Zhang, A. (n.d). *Markov Cluster Algorithm.*

Acknowledgements

This work is funded by the National Science Foundation under the Research Experience for Undergraduate Students by the CCF-1757017 grant.