Applications of Graph Embedding Techniques to Malware Detection

Andrew Rippy and Shyla Sharma*
Wabash College, St. Edward's University

This research looks at how we can express program binaries as graph embeddings in an effort to find malware. We considered three graph embedding methods: graph2vec, GL2Vec, and LDP, in addition to three clustering methods, K-Means, DBSCAN, and Agglomerative Clustering. From the experiments conducted, we determined that graph2vec coupled with the K-means clustering method, was the best overall graph embedding method. These results showed clusters of program binaries that, despite the randomness added in to the base program, retained defining feature components.

INTRODUCTION

Security attacks in the form of malware continue to increase exponentially with ever increasing frequency and sophistication. Malware is considered software that is specifically designed to disrupt, damage, or gain unauthorized access to a computer system. A widely used method of malware detection is comparing potential malware to databases of known malware. This method is known as malware hashing, where the hashing algorithm produces a unique hash for the malware, a kind of malware fingerprint. Most commonly, the Message-Digest Algorithm 5 (MD5)[1] hash function is used for malware analysis, though the Secure Hash Algorithm 1 (SHA-1)[2] is another widely used algorithm. Though databases of malware fingerprints are useful, they are also limited. For example, if a malware binary is edited or changed, the fingerprint database may not be able to identify the new variant. With this problem, we hope to apply graph clustering and anomaly detection as a method to detect these variants. Clusters of malware binaries can stand out against the indistinct and stochastic composition of benign binaries present on a computer system. This is due to the distinct and unique structure of malware, which presents itself as an anomaly, or clusters closely to known malware binaries. By converting programs to control flow graphs using angr[3], a Control Flow Graph (CFG) generator from program binaries, workable unique graph structures for the binaries can be generated. This graph structure is then used with a graph embedding method, such as graph2vec[4], to convert the binary's CFG to a high dimensional feature vector. With a set of vectors collected from a computer system's binaries, we examine the binaries' clusterability using the Hopkins statistic[5]. Based on these results, clustering algorithms such as K-means[6], DBSCAN[6], or Agglomerative[6] clustering can be run to aggregate the data and look for clusters of functionally equivalent binaries. These clusters or anomalies can then indicate the presence malware in the analyzed binaries.

BACKGROUND

1. Angr CFG Generator[3]

Angr is a multi-architecture binary analysis toolkit with the capability to perform various static analyses on binaries. One of these analysis techniques is generating a CFG, which is a graph structure that represents basic blocks as nodes and operations (jumps, calls, etc.) as edges. Angr has the ability to generate two types of CFGs: a static CFG, using CFGFast, and a dynamic CFG, using CFGEmulated. CFGFast generates a CFG utilizing static analysis, which is faster, but theoretically less accurate because some control-flow transitions can only be resolved at execution-time. CFGEmulated uses symbolic execution and is theoretically more accurate, but also significantly slower. The generated CFG is a NetworkX[7] directed graph, which makes all of the normal NetworkX APIs are available.

2. Graph Embedding Algorithms

The graph2vec[4] embedding method is an unsupervised machine learning technique that is used to learn data-driven distributed representations of arbitrary sized graphs. The algorithm takes a data set of graphs as input to learn their representations. Graph2vec considers the set of all rooted sub-graphs around every node, and uses it to train a skip-gram model[8]. A unique label is then assigned to each of the rooted sub-graphs, and a WL[9] (Weisfeiler-Leman Algorithm) relabeling strategy is deployed.

Using graph2vec brings the advantages of using unsupervised representation learning, task-agnostic embeddings, and data-driven embeddings. Task-agnostic embeddings are able to be used across all analytical tasks involving whole graphs, because they don't utilize any task specific information. Data-driven embeddings means that the embeddings are derived from a large amount of graph data, and consequently is not limited by poor generalization. Graph2vec also provides the ability to capture structural equivalence, meaning that it considers non-linear substructures. The research[4] completed regarding this embedding technique demonstrates that graph2vec is an excellent resource for graph classification and graph clustering, as it achieves significant improvements in classification and clustering accuracy over substructure embedding methods, such as node2vec[10] and sub2vec[11]. Graph2vec outperforms state-of-the-art substructure embedding approaches on malware detection and malware familial clustering.

The GL2Vec[12] embedding method was designed to improve upon Graph2Vec. It is capable of preserving the structural information by exploiting the line graphs, which are edge-to-vertex dual graphs of input graphs. The algorithm achieves this by converting the edge features in the original graph to node features of the line graph. Further, GL2Vec concatenates the embedding of the original graph into a line graph in order to account for node label similarity. In the experiments run on benchmark data sets, GL2Vec has improved performance for graph classification when compared to graph2vec.

The Local Degree Profile[13] (LDP) graph embedding method calculates histograms of degree profiles of the graphs, with those calculated histograms then forming the graph embedding representations. The number of bins for the histograms can be changed to control the graph embedding output.

3. *Clustering Algorithms*

We utilized three different clustering algorithms, K-means, DBSCAN, and Agglomerative Clustering, in conjunction with different graph embedding methods to detect anomalies in graphs.

K-means[6] clustering aggregates data based on centroids that are used to define clusters. The scikit-learn K-means implementation requires the number of clusters to be specified as input, and then separates the data into n groups of equal variance based on their proximity to centroids.

DBSCAN[6] defines clusters as areas of high density separated by areas of low density. As input the user must supply ϵ, which is the maximum distance between two samples for one to be considered as in the neighborhood of the other, and the minimum number of samples. Based on these two inputs, core samples (samples that are in high density areas) are defined. A cluster is then generated as a set of core samples that is recursively built through finding the neighbors of core samples.

Agglomerative Clustering[6] is a form of hierarchical clustering, which builds nested clusters by merging or splitting them successively. In the case of agglomerative clustering, each data point begins in its own cluster, and then each cluster is successively merged together based on linkage criteria. The scikit-learn algorithm requires the user to specify the number of clusters to find, and then uses recursion to merge clusters together based on a linkage distance.

In addition to the clustering algorithms, the Hopkins Statistic[5] was utilized to determine the clusterability of data. The pyclustertend[14] algorithm received the data set as input, and returned the Hopkins Statistic of the data, which is a value between 0 and 1. A Hopkins Statistic of 0.5 indicates a uniformly distributed data set, meaning that a value closer to 1 indicates clusterable data.

Performance Metrics

Normalized Mutual Information[6] (NMI) was utilized to determine the accuracy of the clustering algorithms.

The sklearn (scikit-learn) implementation requires the user to input two arrays, one containing the true values (the correct target values) and the other containing predicted values (estimated targets as returned by a clustering algorithm). NMI then measures the agreement of the clustered and explicit data assignments and returns a value between 0 and 1, with 1 indicating perfect clustering.

DATA

For our study, we utilized the AndMal[15][16] data set, as well as two data sets of our own creation. These two data sets were created with Python to take 5 different core benign programs and then add extra for loops and if statements at random places to create variants of the original program binary.

Procedure

To test if clustering was a viable method of malware detection, preliminary tests were run on pre-embedded graphs from the AndMal data set, a collection of feature vectors for various families of Android malware. The tests consisted of the previously aforementioned three clustering algorithms run on these features vectors to see if clustering was a viable way to find malware amidst benign code. These initial tests proved promising, with distinct clusters of malware appearing and being detected by the clustering algorithms. An example of the outcome of one of the initial tests is shown below:

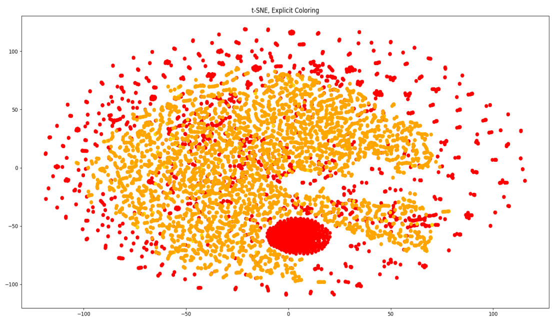

FIG. 1. An Explicit Coloring of Adware (Red) vs. Benign Code (Orange) for the AndMal data set

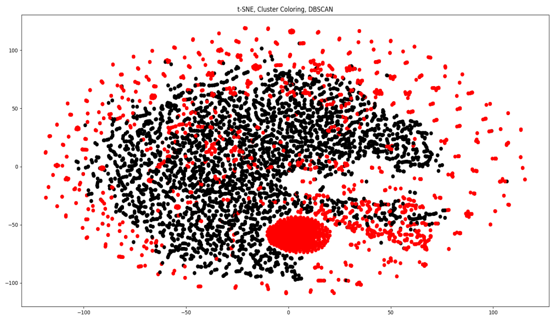

FIG. 2. A DBSCAN Cluster Coloring of AndMal data set Adware vs. Benign Code. Red is considered a cluster (and therefore malware) and black is not considered a cluster and is ignored.

Much of the benign code was ignored, and due to the similar nature of the adware, many of the malicious binaries were detected by the clustering algorithm DBSCAN.

With these promising preliminary results from the initial tests, we moved to graph embedding and clustering techniques. To test the impact of graph embeddings, two data sets with five core programs each were created, Similar Benign Code (SBC) and Distinct Benign Code (DBC). In each data set, five core programs were developed, and 50 variants of each program were made by randomly adding code to the base program to test how the embedding methods would embed similar programs. The difference between SBC and DBC is that the programs in SBC were designed to be more homogeneous, whereas the programs in DBC distinctly fall into five categories based on the core programs. These variants contained random numbers of extra for loops and if statements to create different, yet similar enough programs. For each embedding type, the dimensionality or number of histogram bins were varied with the following number of feature components being evaluated: 2, 4, 8, 16, 32, 64, 128, 256, 512, and 1024. For each embedding, DBSCAN, Kmeans, and Agglomerative clustering were run on the embeddings, then the NMI metric score was recorded for each instance. The SBC and DBC data sets can be found at the github link https://github.com/rippy1849/anomdetect

Experimental Results

Three different graph embedding techniques were tested for the purposes of this experiment: Graph2Vec, GL2Vec, and LDP. In addition to testing these different embedding methods, the parameter settings for each algorithm were manipulated and compared using different types of clustering algorithms. The NMI scores of the optimal parameter settings and clustering algorithm for each graph embedding method were then compared. To help visualize the outputs of each type of embedding, t-SNE (t-distributed stochastic neighbor embedding) was used to reduce the dimensionality of each vector representation of each graph to 2. We then plot this 2-dimensional representation of our data to help us build up intuition for how well the embedding is capturing graph structures for various program types.

Graph2Vec Embedding Method

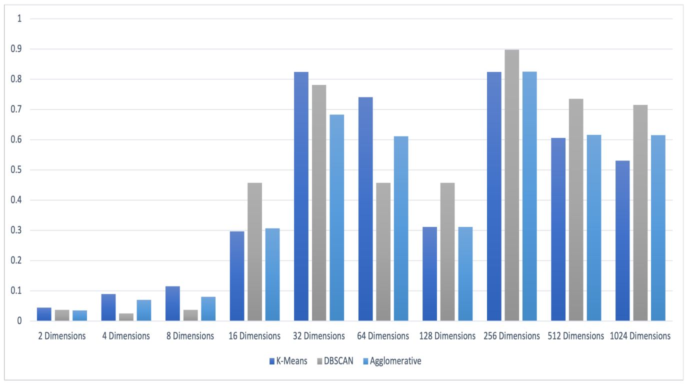

FIG. 3. A clustered column chart of the overall NMI score for each type of clustering method and number of dimensions of graph2vec, using the SBC data set.

As depicted in Figure 3, we see the overall performance for each of the clustering methods for graph2vec using the SBC data set. We found a dimension of 256 for graph2vec paired with the DBSCAN clustering algorithm yielded the overall highest NMI score of 0.8976. For this best outcome, Fig. 4 visualizes the embedding that graph2vec produced using t-SNE to lower the 256 dimensions down to 2. The coloring is based off the DBSCAN clustering outputs.

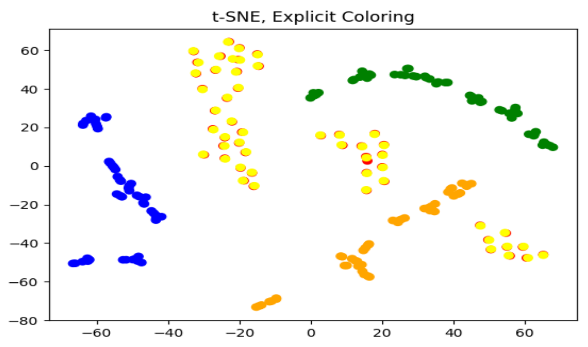

FIG. 4. A t-SNE Visualization of graph2vec with 256 Dimensions and using the DBSCAN clustering method and SBC data set. This combination resulted in an NMI score of 0.8976.

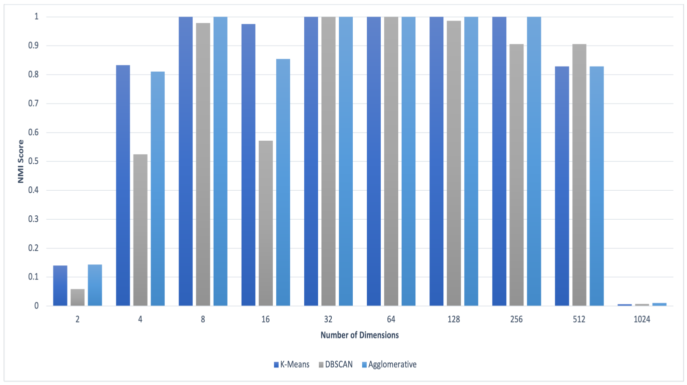

FIG. 5. A clustered column chart of the overall NMI score for each number of dimensions of graph2vec, using the DBC data set.

For the DBC data set, there were several combinations of dimensions for graph2vec and clustering algorithms that resulted in a perfect NMI score of 1; 32 dimensions paired with any of the clustering algorithms, 64 dimensions paired with any of the clustering algorithms, 128 dimensions paired with either K-Means or Agglomerative Clustering, and 256 dimensions paired with either K-Means or Agglomerative Clustering. The overall scoring for each dimension and clustering method is shown in Fig. 5. A visualization of one of these optimal scenarios, 64 dimensions with K-means clustering, is provided in Fig. 6.

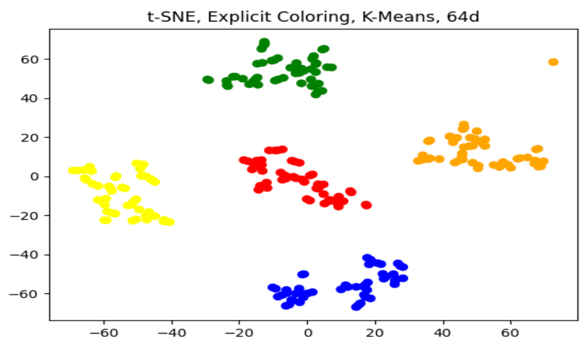

FIG. 6. A t-SNE Visualization of graph2vec with 64 Dimensions for the DBC data set and using the K-Means clustering method. This combination resulted in an NMI score of 1.

GL2Vec Embedding Method

As depicted in Fig 7, we see the overall performance for each of the clustering methods for GL2Vec using the SBC data set. We found for GL2Vec, the best pairing was 2 dimensions with either DBSCAN or Agglomerative clustering, both of which resulted in an NMI score of 0.700. A t-SNE visualization of one of the optimal scenarios, Agglomerative clustering with 2 dimensions for GL2Vec, is included in Fig. 8.

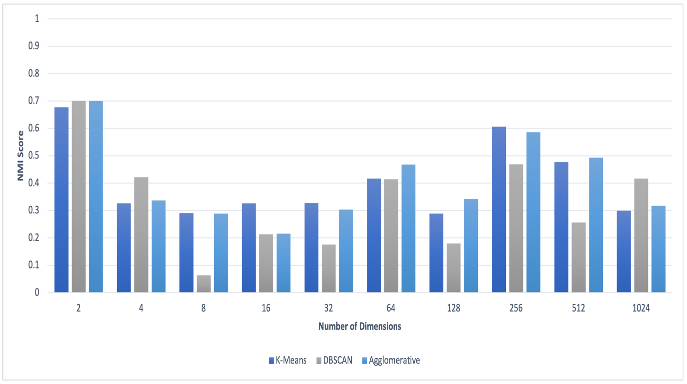

FIG. 7. A clustered column chart of the overall NMI Score for each clustering method and number of dimension of GL2Vec, using the SBC data set.

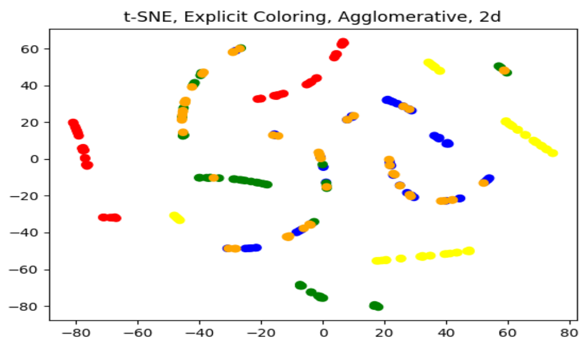

FIG. 8. An Agglomerative Cluster Coloring of GL2Vec with 2 Dimensions for the SBC data set. This combination resulted in an NMI score of 0.700.

For the DBC data set, the ideal combination for GL2Vec was 1024 dimensions with K-Means clustering, which had an NMI score of 0.5555. The overall scoring for each dimension and clustering method is shown in Fig. 9. A t-SNE visualization of the optimal scenario, 1024 dimensions with K-Means clustering, is provided in Fig. 10.

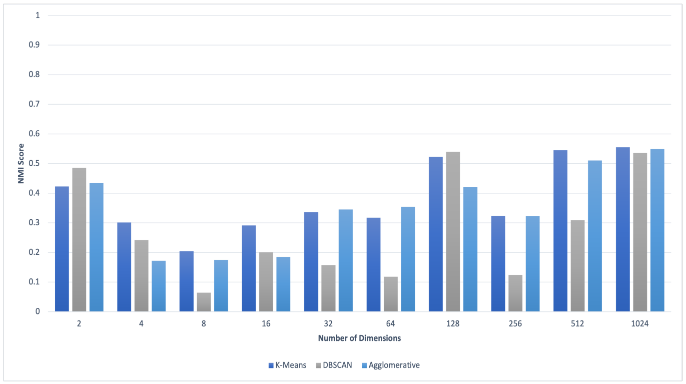

FIG. 9. A clustered column chart of the overall NMI Score for each clustering method and number of dimension of GL2Vec, using the DBC data set.

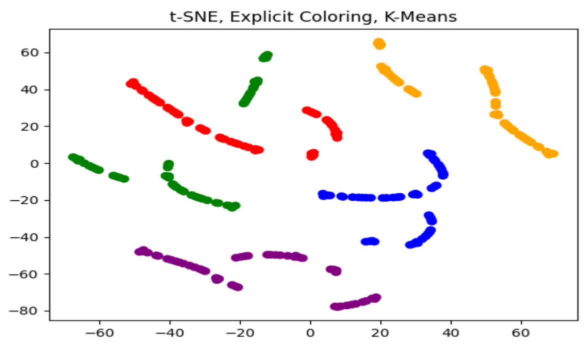

FIG. 10. A t-SNE Visualization of GL2Vec with 1024 Dimensions and using the K-Means clustering method. This combination resulted in an NMI score of 0.5555.

LDP Graph Embedding

As depicted in Fig 11, we see the overall performance for each of the clustering methods with LDP using the SBC data set. Agglomerative clustering paired with dimensions 2, 4, or 256 results in the highest NMI score for LDP in SBC, 0.0205, which is quite low, indicating very little, if any clusterability. A t-SNE visualization of one of the optimal scenarios, LDP with 2 dimensions and Agglomerative clustering, is included in Fig. 12.

For the DBC data set, the ideal combination for LDP was a dimension of 2, 4, or 256 paired with the Agglomerative clustering algorithm yielded the highest NMI score of 0.0205, which is quite low, indicating very little, if any clusterability. The overall scoring for each dimension and clustering method is shown in Fig. 13. A t-SNE visualization of one of the optimal scenarios, Agglomerative clustering with a LDP dimension of 256, is provided in Fig. 14.

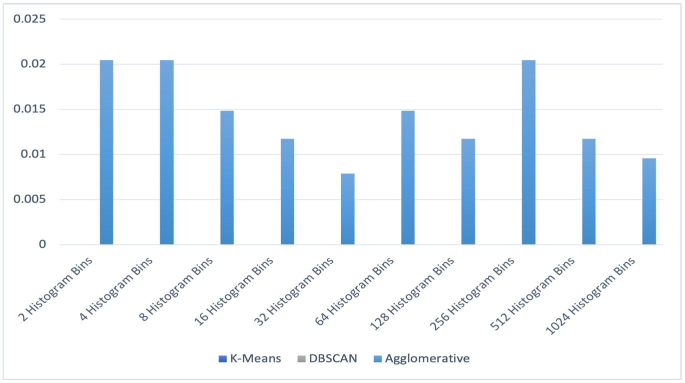

FIG. 11. A clustered column chart of the overall NMI score for each clustering method and number of histogram bins of LDP, using the SBC data set.

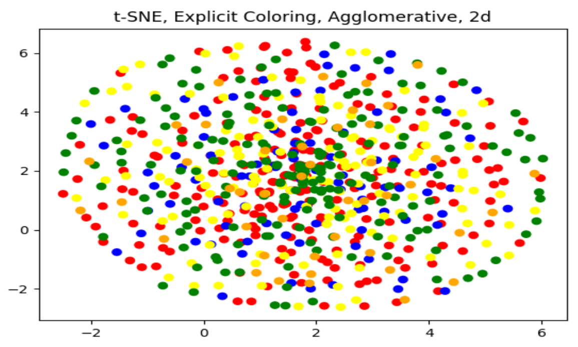

FIG. 12. A t-SNE Visualization of LDP for the SBC data set with 2 dimensions and using the Agglomerative clustering method. This combination resulted in an NMI score of 0.0205.

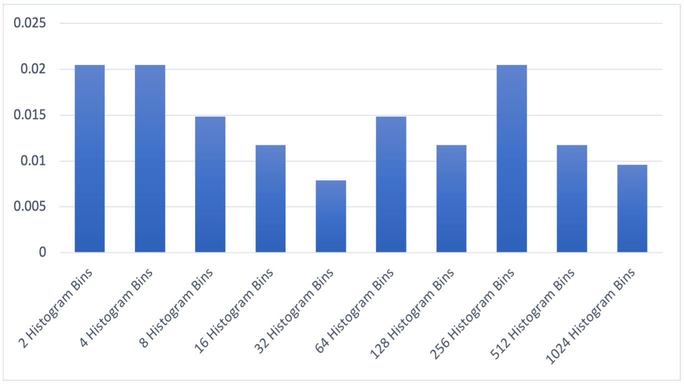

FIG. 13. A clustered column chart of the overall NMI score for each clustering method and number of histogram bins of LDP, using the DBC data set.

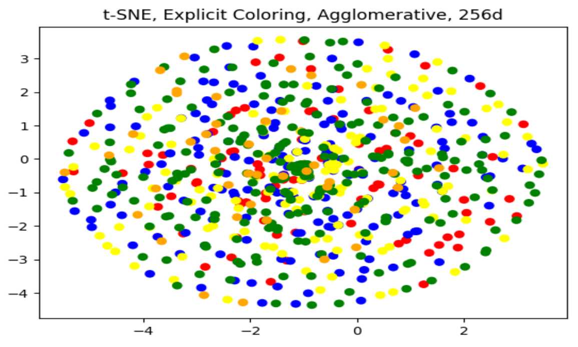

FIG. 14. A Agglomerative Cluster Coloring of LDP with 256 Dimensions. This combination resulted in an NMI score of 0.0205.

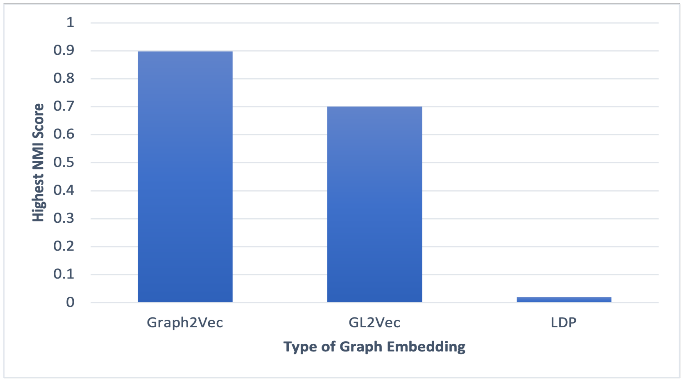

FIG. 15. A clustered column chart of the highest NMI Score for each graph embedding method in SBC.

Overall Graph Embedding Comparisons

The best NMI scores for each embedding method are shown in Fig. 15. We see that for the SBC data set, graph2vec has the highest NMI score of 0.8976, which outperforms that of GL2Vec and LDP. However, GL2Vec still had a high NMI score of 0.700, but both graph2vec and GL2Vec greatly surpass LDP, which had a best NMI score of only 0.02046. For the DBC data set, the overall best performance for each embedding method is illustrated in Fig. 16. Graph2Vec had a perfect NMI score of 1, which exceeds that of GL2Vec and LDP, with NMI scores of 0.5555 and 0.02046 respectively. Again, graph2vec and GL2Vec significantly out performed LDP. One of best scenarios for graph2vec for the DBC data set is shown in Fig. 17.

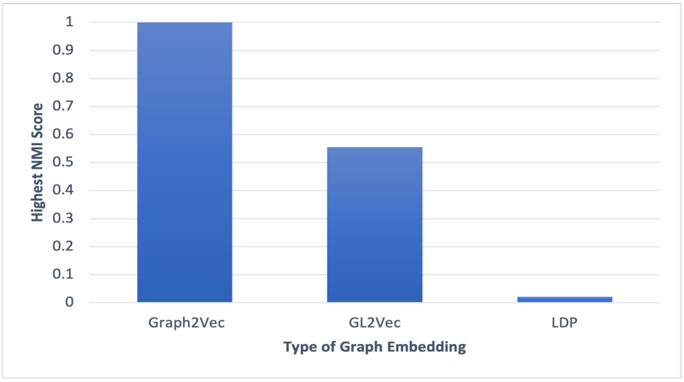

FIG. 16. A clustered column chart of the highest NMI Score for each graph embedding method in DBC.

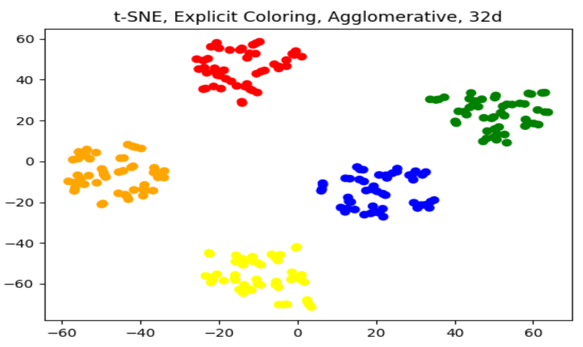

FIG. 17. A t-SNE Visualization of Graph2Vec for DBC with 32 Dimensions and using the Agglomerative clustering method. This combination resulted in a perfect NMI score of 1.

CONCLUSION

From the experiments conducted, we determined that graph2vec was overall the best graph embedding method based on how well the clustering structure of benign code data samples was preserved in the embedding space. This resulted in clusters of programs that, despite the randomness added in to the base program, retained defining feature components. The optimal settings for each embedding method are 32 and 64 dimensions for graph2vec, 1024 dimensions for GL2Vec, and 2, 4, and 256 histogram bins for LDP. The embeddings were shown to preserve the unique structure of a CFG, and despite the variance introduced through random generation, the core features of a program are retained, indicating promising results for application on malware binaries. This is important because even though malware's structure can be obfuscated, we have indication its unique CFG structure would still be retained when it is embedded into higher dimensional space, like the benign programs we tested.

The ideal clustering algorithm for both Graph2Vec and GL2Vec was K-means. For LDP, Agglomerative Clustering had the highest average NMI score, but still scored very low. Overall, we found that the choice of clustering algorithm was not as important as the choice of embedding method.

FUTURE WORK

Future research should further develop and confirm these initial findings by utilizing an available malware binary data set for the experiments, as finding publicly available well documented live binaries is quite difficult. Additional experiments should include a broader range of graph embedding methods, such as family of graph spectral distances (FGSD). In addition, a wider range of parameter settings, other than dimensionality, should also be tested in order to determine the optimal combination for each graph embedding method. For example, a greater range of values for DBSCAN's epsilon parameter setting can be tested, as this parameter could have skewed the DBSCAN NMI results due to it being incredibly sensitive. Similarly, work on improving the CFG's generated for this experiment, as some could have been inaccurate due to problems with angr, such as occasionally accessing address locations it should not in CFG generation. Anomaly detection methods, such as isolation forest, could also be utilized in further experiments to pick out malware from benign code.

ACKNOWLEDGEMENTS

We would like to acknowledge the National Science Foundation and Montana State University for supporting our work through an NSF REU grant.

---

* anrippy22@wabash.edu; shylasharma1@gmail.com

[1] F. A. Sagar, "Cryptographic hashing functions - md5," (2016).
[2] F. I. T. Laboratory, "Secure hash standard (shs)," (2015).
[3] Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel, and G. Vigna, "SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis," (2016).
[4] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning distributed representations of graphs," (2017), arXiv:1707.05005 [cs.AI].
[5] A. Adolfsson, M. Ackerman, and N. C. Brownstein, Pattern Recognition **88**, 13–26 (2019).
[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," (2011).
[7] D. A. S. Aric A. Hagberg and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," (2008).
[8] C. Zhang, X. Liu, and D. Bis, 2019 International Joint Conference on Neural Networks (IJCNN) (2019), 10.1109/ijcnn.2019.8852182.
[9] F. Fuhlbrück, J. Köbler, I. Ponomarenko, and O. Verbitsky, "The weisfeiler-leman algorithm and recognition of graph properties," (2020), arXiv:2005.08887 [math.CO].
[10] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," (2016), arXiv:1607.00653 [cs.SI].
[11] B. Adhikari, Y. Zhang, N. Ramakrishnan, and B. A. Prakash, "Distributed representation of subgraphs," (2017), arXiv:1702.06921 [cs.SI].
[12] K. H. Chen H., "Gl2vec: Graph embedding enriched by line graphs with edge features. in: Gedeon t., wong k., lee m. (eds) neural information processing. iconip 2019."

(2019).

[13] C. Cai and Y. Wang, “A simple yet effective baseline for non-attributed graph classification,” (2019), arXiv:1811.03508 [cs.LG].

[14] I. Lachheb, (2021), 10.5281/zenodo.5761766.

[15] G. K. A. H. L. F. G. F. M. David Sean Keyes, Beiqi Li, “”entroplyzer: Android malware classification and characterization using entropy analysis of dynamic characteristics” reconciling data analytics, automation, privacy, and security: A big data challenge (rdaaps),” (2021).

[16] G. K. L. T. F. G. Abir Rahali, Arash Habibi Lashkari and F. Massicotte, “”didroid: Android malware classification and characterization using deep image learning”, 10th international conference on communication and network security (iccns2020),” (2020).