Graph Summarization for Human-Understandable Visualization towards CVE Data Analysis

Ji Sun Park
Hanyang University
Seoul, South Korea
sgs04023@hanyang.ac.kr

Mingu Kang
Zhejiang University
Hangzhou, P.R. China
jiangmq@zju.edu.cn

Sungryoul Lee
The Affiliated Institute of ETRI
Daejeon, South Korea
srlee0525@nsr.re.kr

Dong-Kyu Chae*
Hanyang University
Seoul, South Korea
dongkyu@hanyang.ac.kr

2022 IEEE International Conference on Big Data and Smart Computing (BigComp) | 978-1-6654-2197-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/BIGCOMP54360.2022.00068

***Abstract*—Graph is one of the data structures used to represent various real-world data effectively. It is easy to analyze complex relationships between objects through graphs, and the structure of data can be understood at a glance through visualization. As the size of the data increases, the scale of the graph representing the data also increases. However, for large-scale graphs, not only is it difficult to analyze patterns, but it is also expensive in terms of storage space and computational speed. In this case, by applying a graph summary method that can extract only the core information of the graph, the aforementioned problem can be solved. This paper studies various graph summarization techniques and applied them to the CVE data. For effective analysis, we first converted the CVE data into a graph, then summarized it with graph embedding and clustering methods. By visualizing the results, we confirmed that we could successfully derive a brief view of the data through this process.**

I. Introduction

A graph is a data structure composed of two elements: nodes and edges. A node represents an object, and an edge represents a relationship between the objects. By analyzing the graph, it is possible to understand the relationship between complex objects within the data and derive the unique characteristics of each graph. A graph can be classified according to various criteria. For example, it can be classified into weighted graph or unweighted graph depending on the presence of weights on edges. Alternatively, it can also be classified into directed graph or undirected graph depending on whether the orientation of the edges exists or not. Graphs can be used to model our surroundings, such as internet networks, road networks, compounds, social networks, genetic information, telecommunication networks, and power grids [1]. Furthermore, graphs are widely used in the field of social science, including sociology, economics, and business administration, for the purpose of analyzing the relationship or interaction between each component [2].

Recently, due to the continuous accumulation of data, the scale of graphs is getting larger [3]. Accordingly, it is becoming increasingly difficult to obtain useful and intuitive information from large-scale graphs [4]. In addition, huge storage space and scalable analysis technology are required to store and analyze large-scale graphs. As one of the ways to solve such problem, a technique called "*graph summarization*" is emerging. Graph summarization is a method for generating relatively small-size graphs that require much less storage while inheriting the unique characteristics of the original, large-scale graphs [5]. The main idea of graph summarization is not only to simply reduce the size of the graph, but also to identify objects and edges that have the most useful information, and then derive a summarized graph based on those nodes and edges. A summarized graph may enable us to analyze a graph much faster, to grasp important information, to understand data patterns at a glance, and so forth. In addition, the *visualization* of the summarized graph will further help users to understand the overall structure of the data more intuitively.

Graph summarization methods have often been utilized in various fields such as bioinformatics, social networks, and citation networks [6]. On the other hand, graph summarization methods have rarely been used in the field of intelligent military and defense systems, even though it can also be used as an effective tool for this area. For example, data gathered from various IoT and sensor networks can be expressed as a graph. However, the large size of the data makes it difficult for defense experts to intuitively obtain meaningful information of potential attacks and aware of situations. Thus, if graph summary methods are applied, it could solve these issues and better assist the military by providing an abstract view of possible attack scenarios.

In this paper, we collected CVE (*Common Vulnerabilities and Exposure*) data, and performed graph summarization based on graph embedding and clustering on this data to provide a view in a form that makes it easier for security experts to understand large-scale data at a glance. CVE data was created for the purpose of sharing publicly known software security vulnerabilities. Each unique CVE stores the characteristics of a particular software vulnerability (e.g. the types of operating system with identified vulnerabilities, a number indicating the degree of vulnerability, etc.). We built a graph with each CVE as a node where the similarity between the features of each node (CVE) is calculated and if the value is greater than 0, the two nodes are connected. Then, we embedded the graph into a vector space using two factorization methods: *HOPE* (*High-Order Proximity preserved Embedding*) [7] and *Laplacian Eigenmaps* [8]. We then employed networkx[1] to visualize

*Corresponding author.

[1]https://pypi.org/project/networkx

2375-9356/22/$31.00 ©2022 IEEE
DOI 10.1109/BigComp54360.2022.00068

TABLE I
EXAMPLES OF FEATURE VALUES.

V2_CVSS	VULNERABILITY	STRUCT	CWE	IMPACT	VENDER	DRIVER	SOURCECODE	SOFTWARE	FUNCTION
6.8	execcode	NaN	CWE-362	2.9	jenkins	NaN	article_list.php	fiyo cms	submitrequestlist
7.5	sql	race condition	CWE-89	6.4	NaN	NaN	zinflate.c	book walker	NaN
9.3	priv	NaN	CWE-426	10.0	sitecore	libavcodec/	NaN	crypto	insert_note_steps
5.0	unknown	webkit	CWE-125	2.9	apache	NaN	item_details.php	fs cms clone	swf_deletefilter
7.5	dos	multi-touch	CWE-89	6.4	huawei	app/helpers/	NaN	walker office	NaN

the embedded results, which showed that the data had its own clusters. Finally, through K-Means clustering algorithm to cluster the data in the embedding space, we could confirm that the nodes with similar feature values belong to the same cluster. As a result, we succeeded in summarizing the CVE data in a form that makes it convenient for security experts to understand the data at a glance.

II. CVE GRAPH CONSTRUCTION

A. Data Description

CVE is a publicly disclosed data set that details the features of software security vulnerabilities. A CVE ID is created by combining a unique number with the year in which the corresponding vulnerability was identified. CVE LIST is available in JSON format for each year [2]. Each CVE ID has various features that describe the corresponding vulnerability, and each feature would contain some value, or NaN if it is missing. To further elaborate on some features, V2_CVSS is a value that quantifies the vulnerability risk evaluated on a scale of 0 to 10. CWE (Common Weakness Enumeration) represents a list of weaknesses of a particular software. CWE is a broader concept than CVEs, and multiple CVEs can belong to one CWE. Impact is a numerical value between 0 and 10 that indicates the ripple effect when the security issue is exploited. Examples of possible values for each feature are summarized in Table I. We collected a total of 453 CVEs.

B. Graph Construction

In this section, we provide further explanation on the graph construction process of the CVE data. First, each CVE is expressed as a node. CVE IDs become the labels of the node. Then, the similarity of the features between each node is calculated, and if the similarity is greater than 0.1, an edge is created between the two nodes and the similarity value is assigned as the weight of the edge. In the case of "CVSS", since it is a value between 0 and 10, these values are divided into four categorical values:[0, 2.5), [2.5, 5.0), [5.0, 7.5), [7.5, 10.0] as part of the preprocessing prior to measuring the similarity. In addition, "Driver", "Sourcecode", and "Function" features are ignored since there are many missing values and the variance between the CVEs is large. The Jaccard index, also known as the Jaccard similarity coefficient [9] is used to calculate the similarity between the CVEs. The Jaccard coefficient measures similarity between finite sample sets, and

[2]https://nvd.nist.gov/vuln/data-feeds

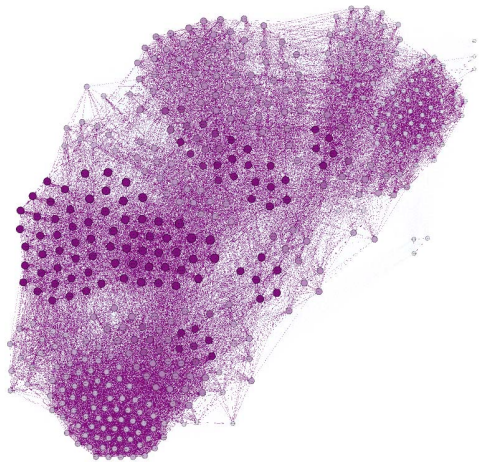

Fig. 1. Graph generated with 453 CVE data. It consists of 453 nodes and 57459 edges.

is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (1)$$

$J(A,B)$ has a value from 0 to 1, and has a value of 1 if the two combinations are identical and 0 if there is no common element.

A graph was built by creating nodes and edges following the steps mentioned above, and the constructed CVE graph was visualized with Gephi[3]. Gephi is an open source software package for network analysis and visualization. It is implemented in Java on the NetBeans platform. The result of the visualized graph is shown in Figure 1.

III. NODE EMBEDDING METHODS

This section elaborates on node embedding methods. We employ an embedding technique for effective graph summarization [10]. Graph embedding algorithms embed a graph into a latent vector space where the structure and the inherent properties of the graph are preserved [7]. We tested various graph embedding approaches and selected *HOPE* [7] and *Laplacian Eigenmaps* [8], which are factorization-based methods. We provide brief explanations of each method in the following subsections. In addition, the results of the embedded CVE graph using HOPE and Laplacian Eigenmaps are visualized

[3]https://gephi.org

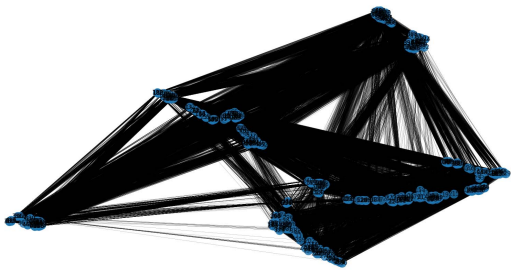

Fig. 2. Visualiztion of CVE graph embedded with HOPE.

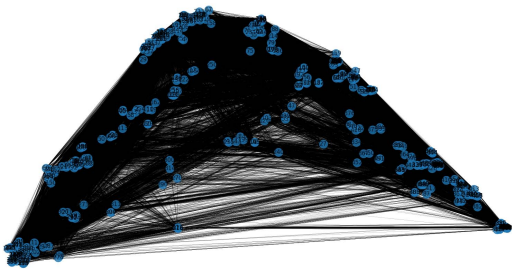

Fig. 3. Visualiztion of CVE graph embedded with Laplacian Eigenmaps.

in Figures 2 and 3, respectively. networkx package was used for visualization.

A. HOPE

HOPE (High-Order Proximity preserved Embedding) [7] is a method of creating a summary graph while preserving the high-order proximities of a large-scale graph. HOPE can be applied to both directed and undirected graphs, and allows for edge weights to be reflected in embeddings, making it suitable for our research. HOPE is trained with the objective of minimizing the loss function of the L2-norm as follows:

$$\min \left\| S - Y^s \cdot {Y^t}^\top \right\|_F^2 \tag{2}$$

where S is a similarity matrix with jaccard weights between each CVE pair. Y is a node embedding matrix and Y^s and Y^t are embeddings for an arbitrary source-target node pairs on the graph. HOPE employs the idea of generalized SVD [11] to solve the above optimization problem.

B. Laplacian Eigenmaps

The objective of Laplacian Eigenmaps [8] is to embed a pair of two nodes i, j in such a way that the distance between i and j in the latent space becomes close if their weight w_{ij} are high. This can be achieved by minimizing the following objective function:

$$\phi(Y) = \frac{1}{2} \sum_{i,j} |Y_i - Y_j|^2 w_{ij} = \operatorname{tr}\left(Y^T L Y\right) \tag{3}$$

where L is graph G's Laplacian matrix. From the Laplacian matrix $L = D - W$ (or in the case of normalized Laplacian matrix: $L_{norm} = D^{-1/2} L D^{-1/2}$), we extract d Eigenvectors corresponding to the d smallest eigenvalues, which are then considered as embeddings.

Algorithm 1 K-Means Clustering Algorithm

Input: the number of clusters k,
a set of node vectors $X = \{x_1, x_2, ..., x_n\}$.

procedure
randomly choose k initial centroids $C = \{c_1, c_2, ..., c_k\}$.
repeat
1. Find closest centroids:
$S_i \leftarrow \{x : \|x - c_i\|^2 \leq \|x - c_j\|^2 \, \forall j, 1 \leq i, j \leq k\}$
2. Update centroids:
$c_i \leftarrow \frac{1}{|S_i|} \sum_{x_j \in S_i} x_j$

until assignments of S_i do not change.

IV. Clustering

In this section, we discuss the clustering of nodes embedded in the latent space as the next step. This is done to help security experts to grasp the overall structure of the data at a glance by putting nodes adjacent to the latent space in the same cluster. Among many others, we chose the K-means clustering [12] .

K-means clustering is an algorithm that groups the given data into k clusters where the variance of the difference in distance between each cluster is minimized. It separates the data into several clusters without overlapping. Formally, given n of d-dimensional data $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, K-means clustering divides n data into $k(\leq n)$ sets $S = \{S_1, S_2, ..., S_k\}$. Assuming that $\mathbf{c}_i$ is the centroid of the set S_i, the objective is to find the set S that minimizes the distances from the center point of each set to the elements in the set:

$$\arg\min_S \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \mathbf{c}_i\|^2 \tag{4}$$

Algorithm 1 shows the overall mechanism of K-means clustering. First, the number of clusters k and the set X of n embedded node vectors are input. Then, k data from node set X is randomly extracted and used as values for initializing centroids for each cluster. Next, for each node of X, the Euclidean distance to k centroids is calculated, and identify which centroid is closest to each node. The node is allocated into the cluster where the nearest centroid to the node belongs. If this cluster allocation process for all nodes in X is complete, then the centroids of each cluster is recalculated. The above process repeats until the clusters no longer change.

V. Visualization Results

We applied K-Means clustering described in Section IV to the node embeddings obtained by the two methods introduced in Section III. As a result of analyzing the embedded visualization results shown in Figures 2 and 3 prior to clustering, the number of clusters k was set to 6. We used seaborn[4] based on matplotlib as a visualization tool. Figure 4 is the clustering result for HOPE-based embedding, and Figure 5 corresponds to the clustering result for Laplacian Eigenmaps-based embedding. By comparing the results, it can be confirmed that the

[4]https://pypi.org/project/seaborn

TABLE II
INFORMATION ON SOME OF THE CVE DATA ALLOCATED TO CLUSTER 1 IN FIGURE 4. WE CAN OBSERVE HIGH SIMILARITY IN THEIR FEATURE VALUES.

LABLE	V2_CVSS	VULNERABILITY	STRUCT	CWE	IMPACT	VENDER	SOFTWARE
CVE-2017-1000386	3.5	xss	NaN	CWE-79	2.9	jenkins	active choices plugin
CVE-2017-15892	3.5	xss	slash command creator	CWE-79	2.9	NaN	synology chat
CVE-2017-9979	4.3	xss	NaN	CWE-79	2.9	osnexus	quantastor v4 virtual appliance
CVE-2017-8953	3.5	xss	NaN	CWE-79	2.9	hpe	loadrunner
CVE-2017-7634	4.3	xss	crafted link	CWE-79	2.9	qnap	nas application media streaming add-on
CVE-2017-5542	4.3	xss	NaN	CWE-79	2.9	NaN	symphony cms
CVE-2017-9336	4.3	xss	NaN	CWE-79	2.9	wordpress	wp editor.md plugin
CVE-2017-16681	4.3	xss	NaN	CWE-79	2.9	sap	business intelligence promotion management application
CVE-2017-7437	4.3	xss	NaN	CWE-79	2.9	netiq	privileged account manager

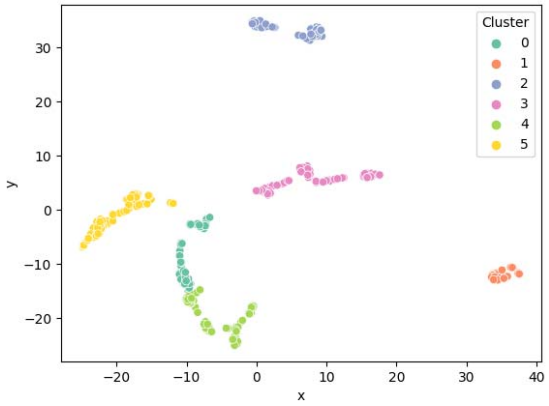

Fig. 4. Clustering results for embeddings based on HOPE.

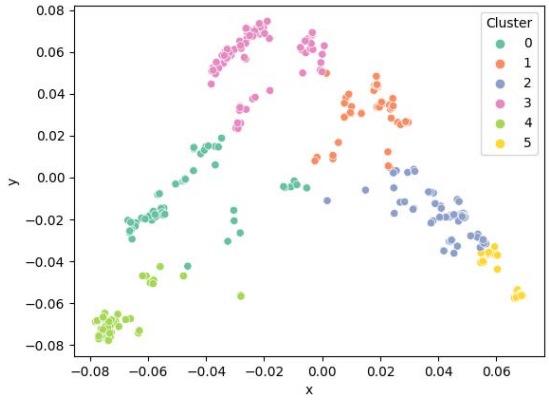

Fig. 5. Clustering results for embeddings based on Laplacian Eigenmaps.

embedding results based on Laplaican Eigenmaps are more scattered, while the results from HOPE are embedded in a grouped form where there are several recognizable clusters. We analyzed the clusters in Figure 4 in more detail. For example, we checked the actual feature values of the nodes belonging to cluster 1 in Figure 4 to see if they are actually similar to each other. Table II summarizes the feature values of some of the nodes belonging to cluster 1. We found that the feature values of the nodes were highly similar to each other. This achievement is expected to help security experts who want to analyze not only CVE data but also various graphs in the security domain to easily understand the data at a glance.

VI. CONCLUSION

This paper presents a study that provides a compact view of security vulnerability data for security experts by graphing and summarizing CVE data. We created a graph with each CVE data as a node and weighted edges with similarity above a certain threshold between each CVE node. Then, the graph was embedded in the latent vector space using HOPE and Laplacian Eigenmaps, and the nodes were clustered using the K-Means clustering algorithm. When we visualized both the embedding and clustering results, we could identify meaningful cluster structures in the data. Through further inspection of CVE data belonging to the same cluster, the actual feature values were found to be similar to each other.

ACKNOWLEDGEMENT

This work was partly supported by Institute of Information & communications Technology Planning Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2020-0-01373,Artificial Intelligence Graduate School Program(Hanyang University)) and by commissioned research project supported by the affiliated institute of ETRI[2021-120].

REFERENCES

[1] F. Chen, Y.-C. Wang, B. Wang, and C.-C. J. Kuo, "Graph representation learning: A survey," *APSIPA Transactions on Signal and Information Processing*, vol. 9, 2020.

[2] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Computing Surveys (CSUR)*, vol. 40, no. 1, pp. 1–39, 2008.

[3] R. Chen, X. Weng, B. He, and M. Yang, "Large graph processing in the cloud," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 1123–1126.

[4] P. Harish and P. J. Narayanan, "Accelerating large graph algorithms on the gpu using cuda," in *International conference on high-performance computing*. Springer, 2007, pp. 197–208.

[5] Y. Liu, T. Safavi, A. Dighe, and D. Koutra, "Graph summarization methods and applications: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–34, 2018.

[6] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[7] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1105–1114.

[8] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering." in *Nips*, vol. 14, no. 14, 2001, pp. 585–591.

[9] P. Jaccard, "The distribution of the flora in the alpine zone. 1," *New phytologist*, vol. 11, no. 2, pp. 37–50, 1912.

[10] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[11] C. F. Van Loan, "Generalizing the singular value decomposition," *SIAM Journal on numerical Analysis*, vol. 13, no. 1, pp. 76–83, 1976.

[12] H. Steinhaus *et al.*, "Sur la division des corps matériels en parties," *Bull. Acad. Polon. Sci*, vol. 1, no. 804, p. 801, 1956.