Graph Machine Learning for Anomaly Prediction in Distributed Systems

Sheyda Kiani Mehr, Wenting Sun, Xuancheng Fan, Nikita Butakov, Nicolas Ferlans
Ericsson, Santa Clara, USA
Email: sheyda.kiani.mehr@ericsson.com, wenting.sun@ericsson.com, xuancheng.fan@ericsson.com
nikita.butakov@ericsson.com, nicolas.ferland@ericsson.com

***Abstract*—Machine-learning based anomaly detection in distributed systems is a challenging task. With thousands of dynamically changing parameters, prediction of events requires considerable effort on feature selection, feature engineering, and training. For the unstructured and multi- dimensional data in IT infrastructure, many traditional machine-learning methods are unsuitable and perform poorly. Graph-based machine learning is a powerful tool capable of representing this data without losing the temporal and logical connections that are critical in making accurate predictions. Furthermore, graph-based embedding can effectively reduce the data complexity, without losing key relations. In this paper, we investigate the use of graphs for representing IT infrastructure data, in particular with the node2vec algorithm, and evaluate the performance of a random forest model. The results show that the embedded graph representation improves the precision and AUC, compared to other conventional approaches, while significantly reducing the memory needed for training and validation, thereby making it more suitable for inference on edge devices, where compute resources could be limited.**

***Index Terms*—Graph, Embedding, Node2Vec, Random Walk, Machine Learning**

I. Introduction

Any distributed system must ensure high availability and high consistency. To resolve performance degradations, which cause dissatisfaction to users, detection and prediction of anomalies can help engineers provide more reliable services, and thereby save money and time. Anomaly detection has a wide range of applications in detecting cyber-intrusion, fraud, industrial damage, and safety issues . There are various approaches including classification, clustering, statistical, information-theoretic, and spectral [1] approaches.

To effectively detect anomalies, data must accurately represent the logical dependencies, attributes, and domain characteristics [2]. Graph data structures are a powerful tool to represent data with strong inter-dependencies. For a system that can be represented as a graph, the connectivity and interaction between data points make the graph turn into a number of linked paths for finding out these relations, to detect and analyze an anomaly. Traditional graph analysis methods based on graph statistics and features are helpful, but designing them can be a time-consuming and expensive process [3].

Alternatively, a graph embedding approach can be utilized, which automatically transforms graph into feature vectors. This embedding will ideally contain all the relevant graph information. After optimizing the embedding space, the learned embedding can be used as feature inputs for downstream machine learning tasks [3].

In this work we present a graph-based machine learning anomaly detection approach for identifying anomalies in the response times of APIs in a distributed system. We propose a Node2Vec approach for graph embedding, with Random Forest Classifier method for anomaly prediction. We find the graph-based ML approach shows improved performance, compared to traditional approaches, with a significant reduction in required compute resources.

Overall the contribution of this work includes:

- The graph representation of the data that keeps all the knowledge intact.
- The graph dimensionality reduction, which enables faster development of models
- Classification and labeling for an anomaly prediction problem.
- Prediction performance improvement, compared to conventional data approaches.

In Section 2, related work is reviewed. In Section 3, our new methodology is presented. In Section 4, the implementation process is explained. In Section 5, the approach is evaluated. And in Section 6, we provide the conclusion.

II. Related Work

There has been much recent work in anomaly detection on graph-like data. Kandel et. al. [4] detect anomalies by using the node attribute information, together with the structural connectivity. To preserve closeness information, they consider similarities between node values with multiple attributes. To discover the similarity between attributes, they use discretization, distance- based similarity measures, and a k-means clustering approach.Ahmed et. al. [5] propose neighborhood-structure-assisted negative-matrix-factorization (NMF) and its application in unsupervised point anomaly detection. They consider the incorporation of the neighborhood structural similarity information, within the NMF framework, by modeling the data with a minimum spanning tree.Tosyali et. al. [6] proposed a cluster-based outlier score function to identify outliers in citation networks based on NMF. They represent citation data as a directed graph and cluster it into logical

DOI reference number: 10.18293/SEKE2020-010

groupings of nodes.Markovitz et. al. [7] propose pose clustering for anomaly detection of human actions. They apply a deep-embedded-clustering model with three parts: an encoder, decoder, and soft clustering layer. Venkatesan et. al. [8] propose graph-based unsupervised models for edge anomaly and node anomaly detection in social network data. They apply the HDBSCAN clustering algorithm for node anomaly detection with various dimensionality reduction algorithms. Lu et. al [9] proposes anomaly detection for container-based cloud environments by monitoring the response time and resource usage of components.

Our work is a supervised learning classification approach for anomaly prediction in distributed systems. We represent data with graphs and embed each to feature vectors. We define labels based on the response time thresholds. Many of these previous works use graph statistic information and dimension reduction methods to modify the data and feed it to an unsupervised learning algorithm for clustering anomaly purposes. However, so far we did not find any other work that use a graph embedding with neural networks [10] as input for ML algorithm and dimension reduction, which keep all the knowledge of the data. Most of the anomaly detection works used unsupervised learning approach, however we define it as a supervised learning problem with domain expert knowledge, help to achieve more reliable performance than unsupervised approaches.

III. Methodology

Distributed systems are time-sensitive. When a user makes a request to an API, a response should be quickly received. Identifying and predicting APIs with anomalous response times help operators quickly resolve and prevent anomalous behavior.

Our data is represented as a directed graph, with nodes being various servers and APIs, connected to each other with a weight equal to the response time (Fig. 1). Individual servers are not connected to other servers, and APIs are not connected to other APIs. As mentioned before, in this system, the definition of an anomaly is when an API fails to respond in a timely manner to a request. In the graph view, this corresponds to an edge weight which exceeds a certain threshold.

The differences between the graph embedding algorithms depend on the graph property being maintained. Different graph embedding algorithms have different definitions of the node, edge, substructure and whole-graph similarities. Node embedding represents each node as a vector in a low dimensional space. Nodes that are similar to each other in the graph have similar vector representations [3]. Our problem is a node embedding problem, as we want to identify which node is acting abnormally, and predict the anomalous behavior of that node in the future. Graph embedding techniques, such as Node2Vec, exploits the structure of the graph and can be used for transformation of graphs into the necessary feature vector space.

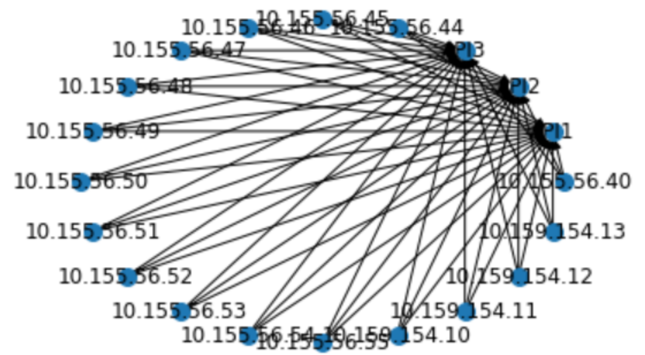

Fig. 1: A directed graph (of a two-minute snapshot) of a distributed system (three APIs, multiple servers).

A. Node2Vec

Node2Vec is one of the most common approaches for projecting nodes into feature vectors. In Node2Vec, nodes are mapped into a low-dimensional space of features that maximizes the likelihood of preserving network neighborhoods of nodes [10]. Node2Vec has two steps: random walk and word2vect. The former creates a corpus of acyclic subgraphs. The latter embeds this corpus to the feature vector space.

1) Random Walk: The parameters for the random walk step include: number of walks to be generated from each node, the number of nodes in each walk, the return parameter (p) and the out parameter (q). The random walk starts with a random node and proceeds through a path based on value of $weight{*}\alpha$ where α is $1/p$ or $1/q$, depending on if the path navigation is backward or forward.

2) Word2Vec: The output of the random walk step is a corpus of subgraphs. Each random walk corresponds to a sentence-like structure, in which each node corresponds to a word. The Word2Vec model transforms this corpus into an embedding, by using a SkipGram model with a neural-network layer into an N-dimensional embedding.

B. Anomaly Prediction

Anomaly Prediction can be implemented by either supervised classification algorithms or unsupervised clustering algorithms. Classification-based techniques can be thought of as operating in two phases. In the training phase the classifier learns using labeled training data. In the validation phase the classifier categorizes validation data as normal or abnormal. Classification based anomaly detection techniques operate under the general assumption that normal and anomalous classes can be distinguished in a given feature space [2]. Based on our data, we can define labels for each generated graph, and thereby develop a supervised algorithm for graph label classification. The random forest classifier is fast and easy to implement. It can produce highly accurate predictions, handle a very large number of input variables, and tolerate unbalanced or missing data [11].

IV. Implementation

A. Dataset

The dataset consists of two-minute snapshots, over two days of synthetic data, generated based on three months of real data, from a distributed system. In each snapshot, there are multiple servers and three APIs. Each API has its own response time threshold, based on domain knowledge. In some snapshots there may not be any response time, which means that API was not been called in the two-minute snapshot. In other words, there is no edge. For each series of graphs, their embeddings, labeling, and predictions will be separately implemented for each API.

To minimize the presence of overly-sparse graphs, we aggregate three two-minute snapshot graphs into six-minute graphs. These aggregated graphs are the input to Node2Vec, which outputs the embedded feature vectors. We feed these into the Python Sklearn Random Forest Classifier, which outputs predictions.

1) Nature of Dynamic graphs: One of the challenges in our work is the nature of time series data. The system gets a snapshot for every two minutes of the processes, requests and related latencies. In the graph representation, it will be translated to the nodes and the connectivities with weights on the edges, respectively. In each two-minute time slot, the numbers of nodes connected (involved servers and APIs) and latency of API requests (edge's weight) in the system are varying. In other words, the whole data set is a group of consecutive dynamic graphs. The graph of each time stamp might be very different compared to the graphs from the next or previous time stamp. In this case, the cosine similarity of an embedded node of the different graphs are high. With a high cosine similarity between each pair of graph embedding vectors, the prediction will not be accurate.

Graphs can be dynamic in number of nodes, connectivity/edges between nodes, edge directions, edge attributes, node attributes and etc (Fig. 2). In our dataset the changes over time in the graphs include the number of nodes, latency connectivity or weight value of edges. For example, in time stamp 00:00:00 we might have servers that are calling specific APIs with a very low latency and in the next time stamp 00:00:02, the same servers and APIs might have connection but with some different higher latency values on the edges which categorize some APIs of that graph as anomalous APIs. In addition, the variation of node numbers and connectivities in different graphs is eliminated with aggregating the graphs; therefore, all the six-minute graphs will have the same connectivity.

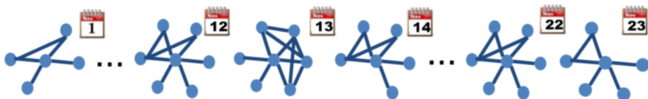

Fig. 2: Event detection in time Series of graph data [1].

Random walk is a good method if dynamic graphs with different connectivity are used as data. Weight is used for walking decision just per graph. In that case, random walks of a graph with very high weight edges might be equal to random walk of another graph with a way lower weight edges containing the same links. Therefore, the embedding of the two different graphs that has very different link weights will be the same although one might not have anomaly and the other one might have anomaly.

We make the data more informative when feeding it to the Random walk. We modify each six-minute graph, we consider an undirected graph and then we assumed all the edge weights as 1 or 0 based on the threshold values of the APIs in the data. The directed graph only guide each walk through the direction from an server to an API and will not cover all the possible nodes.

After implementing this method, we are able to lower the cosine similarity between graphs as the similarity of the graphs are not just based on the connectivity but also based on the weight on the edges, which plays an important role to capture the temporal dynamic information between graphs.

2) Two-minute graphs: We generate directed weighted graphs of two-minute snapshots and add all the nodes and edges with the assigned weights to each graph. In the case in which a graph contains multiple overlapping edges, the edge with the maximum weight (i.e. response time) is used. If any response time in the graph exceeds the threshold, it is labeled as 1, otherwise 0.

3) Six-miutes graphs: The two-minute snapshots are unnecessarily short intervals for anomaly prediction in this system. Therefore, we consider each aggregated undirected six-minute graph as the feature window, and the subsequent undirected six-minute graph as the prediction window. The label of each six-minute graph is the logistic OR of the label values in the next immediate three two-minute graphs. The weights of the edges in a six-minute graph is based on an API threshold, if the weight is more than the API threshold that the graphs have been generated and labeled for, then those edges will have weight value 1 otherwise 0.

We consider a rolling window for feature and prediction window, with no gap between feature windows and prediction windows. For example, the first six-minute feature graph will be the aggregation of three two-minute graphs from indexes 0 to 2 with the label, which is based on the prediction window from two-minute graphs with indexes 3 to 5. And the second six-minute feature graph will be the aggregation of two-minute graphs indexes from 1 to 3 with the label, which is based on the prediction window from two-minute graphs indexes from 4 to 6. And the last six-minute feature graph will be the aggregation of two-minute graphs from indexes n-5 to n-3 with the label, which is based on the prediction window from two-minute graphs indexes from n-2 to n. Mostly in the six-minute graph, we have multiple edges between the same pair of nodes, so we just consider edge with maximum weight between the same pairs.

B. Node2Vec Embedding

Node2Vec uses a random walk algorithm and then word2vec to create embedding of the random walks. We use the python Node2Vec package, which takes as input a graph, the dimensionality N for Word2Vec neural network layer, the walk length, the number of walks, the number of neural network workers, and the p and q parameter values. For the six-minute graphs, we find out the optimal values of these parameter by trying various measurements. The random walk is a flexible neighborhood sampling strategy which allows smooth interpolation between breadth-first-search and depth-first-search, in which the p and q parameters guide the walk [10].

The optimal values for parameters which we find through tuning are dimensions = 50, walklength = 20, numwalks = 20, p = 1, and q = 1. We should mention that although increasing dimension will improve the accuracy, it can also cause overfitting if the embedded dimension goes close or beyond the original dimension. Algorithm. 1 explains all the steps in more details.

V. Results and Evaluation

Data is recorded every two minutes, with seventeen servers that call three different APIs (Fig. 3). We group data by two-minute snapshots in a way that each time stamp will have one row in the data frame. To mitigate issues with sparsity, we consider a six-minute feature and prediction window. The reason we choose a window size of six minutes and not larger, is that for larger sizes, the label does not have a reliable value, and the models overfit.

	_time	ip_address	API1	API2	API3
0	8/26/2019 3:02	10.155.56.40	28103.477840	2229.161672	1993.162746
1	8/26/2019 3:02	10.155.56.44	1203.942135	4192.376419	468.450000
2	8/26/2019 3:02	10.155.56.45	8589.461385	252.956551	2461.344498
3	8/26/2019 3:02	10.155.56.46	801.239658	5530.270823	1865.235112
4	8/26/2019 3:02	10.155.56.47	9564.414603	3818.368130	223.875846

Fig. 3: Initial data from distributed system with three APIs and multiple IPs in each two-minute snapshot.

A. Original Data

We flatten the latency values of the APIs for all the servers in each two-minute window to have one row, 51 columns for each row by grouping the snapshot data (17 servers x 3 APIs = 51 columns, Table. 4). For the six-minute duration feature window, we flatten all the rows of three sequential two-minute flattened rows to generate a row of six-minute snapshot (3 rows x 51 columns = 153 columns). The label column for a six-minute snapshot row depends on all the label values in the next three sequential two-minutes data. If any of the label values of the three two-minute snapshot that show up in the predict window is 1, then the label for six-minute snapshot feature is 1 otherwise 0 (Fig . 5).

We should mention that, another reason that we consider six-minute for feature and prediction window was that the

Algorithm 1 Anomaly detection with Graph Embedding for node API1

$Params$:df=$dataframe$,k=100,d=50,num_walk=50
$length_walk$=50,p=1.5,q=0.5

Function MAIN()
$API1 \leftarrow threshold_API1$
$API2 \leftarrow threshold_API2$
$API3 \leftarrow threshold_API3$
$k \leftarrow 100$
$G2_list, label2_list \leftarrow$ 2MIN-GRAPH($API1, API2, API3, df$)
$G6_list, label6_list \leftarrow$ 6MIN-GRAPH($G2_list, label2_list$)
$EMB_list \leftarrow$ EMBED-GRAPH($G10_list$)
$Accuracy \leftarrow$ RANDOM-FOREST($EMB_list, label6_list, k$)
End Function

Function 2MIN-GRAPH($API1, API2, API3, df$)
$time_list \leftarrow$ GET-TIME(df)
For $time$ in $time_list$:
$df2 \leftarrow df[time]$
$API_list \leftarrow$ GET-APIS($df2$)
$IP_list \leftarrow$ GET-IPS($df2$)
For API in API_list:
For IP in IP_list:
$latency \leftarrow$ GET-LATENCY(IP, API)
If $latency >= API1$:
$G2 \leftarrow$ CREATE_GRAPH($IP, API, wight = 1$)
$G2_list \leftarrow$ APPEND-TOLIST($G2$)
If $AP1_threshold_2min >= API1$:
$label2 \leftarrow 1$
else:
$label2 \leftarrow 0$
$label2_list \leftarrow$ APPEND-TOLIST($label2$)
$return\ G2_list, label2_list$
End Function

Function 10MIN-GRAPH($G2_list, label2_list$)
For $(i = 0; i < size(G2_list); i++)$:
$G6 \leftarrow$ MERGE($G2_list[i]..G2_list[i+3]$)
$G6_list \leftarrow$ APPEND-TOLIST($G6$)
If $(OR(, label2_list[i].., label2_list[i+3]) == 1)$:
$label6 \leftarrow 1$
else:
$label6 \leftarrow 0$
$label6_list \leftarrow$ APPEND-TOLIST($label6$)
$return\ G6_list, label6_list$
End Function

Function EMBED-GRAPH($G6_list$)
For $G6$ in $G6_list$:
$EMB \leftarrow$ NODE2VEC($G6, d, num_walk, length_walk, p, q$)
$EMB_list \leftarrow$ APPEND-TOLIST(EMB)
$return\ EMB_list$
End Function

flattening of all the values of 10 and 20 minute feature window is a much more computationally intensive aggregation for this particular data set. In this case the number of feature columns will be 5 rows x 51 columns = 255 rows and 10 rows x 51 columns = 510 rows for 10 minutes and 20 minutes feature window, respectively.

B. Graph Represented Data

We create a graph for each grouped two-minute data, with nodes including servers and APIs (Fig. 6). So, the type of the

	0	1	2	3	...	50	label
0	12401.948060	1908.148156	3502.684315	4454.048721	...	2229.395315	0
1	19572.456250	683.115076	5302.389361	7131.114698	...	590.721805	0
2	15803.723440	639.174881	4366.975210	2763.634711	...	2475.681031	1
3	2387.346349	926.582622	2719.405802	12332.254270	...	4157.377468	0
4	5366.081955	2877.119123	616.587115	5800.050549	...	1260.454531	0

Fig. 4: Flattened two-minute data

	0	1	2	3	...	152	label
0	12401.948060	1908.148156	3502.684315	4454.048721	...	2475.681031	1
1	19572.456250	683.115076	5302.389361	7131.114698	...	4157.377468	1
2	15803.723440	639.174881	4366.975210	2763.634711	...	1260.454531	1
3	2387.346349	926.582622	2719.405802	12332.254270	...	2279.413613	1
4	5366.081955	2877.119123	616.587115	5800.050549	...	3091.551923	1

Fig. 5: Flattened six-minute data.

data is not a row of some values of response times anymore. Each row represent a two-minute graph that keeps all the connectivity and knowledge intact. All the represented links have weight value 1 and the pairs of nodes with no link have weight value 0 based on the threshold of the connected API (or there was no request between the pairs in the beginning at all). For six minutes duration, we aggregate three consecutive two-minute graphs to one graph (Fig. 7) with a reduced dimensionality compared to original flattened data (10, 20 and 50). The Six-minute graphs will be embedded by Node2Vec algorithm (Fig. 8). Then the feature vectors are ready to be fed to the prediction algorithm.

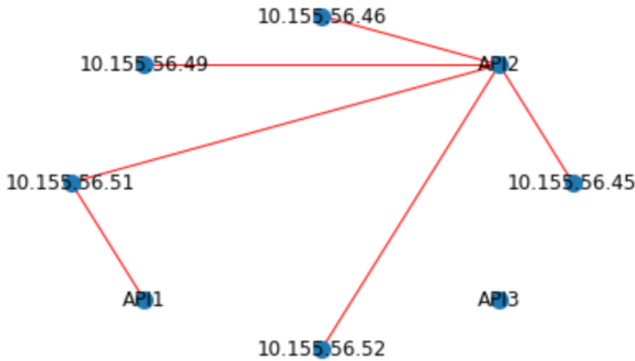

Fig. 6: A graph of a two-minute snapshot.

In the six-minute data aggregation, we shift the feature and prediction windows by 1. So, the feature windows' indices are from 0 to n-1 and the labels' index are 1 to n. Then we applied k-fold cross validation with k=100. We measure the AUC for each API (Table. I). The threshold values of API1, API2, and API3 are 16770, 2184 and 4635 milliseconds, respectively.

Approach	AUC
original	0.49514
Grpah d=10	0.54983
Grpah d=20	0.50564
Grpah d=50	0.59208

TABLE I: Evaluation of original data and graph representation of data for API1

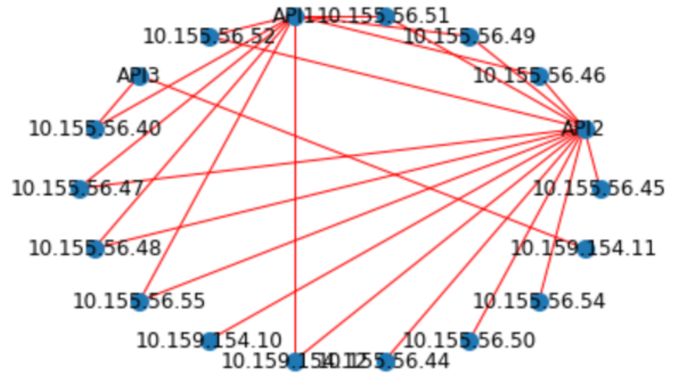

Fig. 7: A graph of a six-minute snapshot.

	0	1	2	3	...	49	label
0	-0.293229	-0.219433	-0.750952	0.772203	...	0.269326	1
1	0.355292	-0.815460	-0.256776	0.486253	...	0.030172	1
2	0.463733	-0.582479	-0.213521	0.327157	...	0.051770	1
3	0.236315	-0.038778	-0.418489	-0.050855	...	0.058645	1
4	0.796602	-0.809523	-0.327683	0.131743	...	-0.020473	1

Fig. 8: Embedding feature vectors of a six-minute graph.

The result can be improved significantly if the proposed approach is to be applied on the non-synthetic dataset. In the real dataset, the number of calls is more various in each snapshot window, and there are higher number of calls. The variations of latency are higher, and the threshold of values are more realistic.

This result also indicates that graph is a well-designed representation that improves predictability by 19.58% for graph d=50 compared to the original data (Table. I), by leveraging the temporal and dynamic additional information embedded in graph. We also present the training and inference time result for the original flattened and graph data with six minutes, ten minutes, and twenty minutes window times. The machine that is used to run these experiments has Windows 10 64-bit OS with core i7 CPU and 32GB RAM memory. For the largest dimension of the graph data, fifty, Table. II shows that the original data has higher training and inference time compared to the graph data. The consumed time has been decreased

Approach	6 minutes	10 minutes	20 minutes
Flattend	0.47	0.6	1
Grpah d=50	0.42	0.42	0.42

TABLE II: Training and inference time (seconds) of original and graph d=50 data for 6, 10 and 20 minute time window.

by 10.46% by using graph with six-minute window size. The consumed time with the ten and twenty minute window size decrease by 30% and 58% by using graph data, compared to original flattened data. The reason is that graph representation of the data always has 50 features, however, the original dataset has 153, 255 and 510 for 6 minutes, 10 minutes and 20 minutes time window, respectively.

VI. Conclusion

We implement node2vec graph embedding for our simulated distributed system data and compared it to the original data with a random forest classifier. Node2vec has two parts: random walk and word2vec. The random walk generates subgraphs of the original graph, and by considering it as a corpus, word2vec embeds the graph nodes to a desired dimension size. We achieve the dimensionality reduction in a way that all the relevant knowledge of the data remains intact. We used classification machine learning algorithm to predict a future anomalous event. The embedded graph shows better AUC than raw data. For future work, we could use graph representation to discover potential causality given that a proper DAG (directed acyclic graph) can be derived based on domain knowledge. In addition, edge and graph embedding and prediction can be further explored to identify the system dynamics from a different perspective. These can be interesting areas for future work. Based on our experiment, the random walk algorithm is not the best way to traverse a weight-based dynamic graph and the anomalous event has a direct relationship to an edge weight. We may be able to consider additional data with different features other than response time such as CPU, memory, and database event logs to enrich the graph design, in hope of improving both adaptability and prediction effectiveness.

References

[1] L. Akoglu, H. Tong, and D. Koutra, "Graph-based anomaly detection and description: A survey," 2014.

[2] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[3] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[4] P. R. Kandel, *Node Similarity for Anomaly Detection in Attributed Graphs*. PhD thesis, Tennessee Technological University, 2019.

[5] I. Ahmed, X. B. Hu, M. P. Acharya, and Y. Ding, "Neighborhood structure assisted non-negative matrix factorization and its application in unsupervised point anomaly detection," *arXiv preprint arXiv:2001.06541*, 2020.

[6] A. Tosyali, J. Kim, J. Choi, Y. Kang, and M. K. Jeong, "New node anomaly detection algorithm based on nonnegative matrix factorization for directed citation networks," *Annals of Operations Research*, pp. 1–18.

[7] A. Markovitz, G. Sharir, I. Friedman, L. Zelnik-Manor, and S. Avidan, "Graph embedded pose clustering for anomaly detection," *arXiv preprint arXiv:1912.11850*, 2019.

[8] M. Venkatesan and P. Prabhavathy, "Graph based unsupervised learning methods for edge and node anomaly detection in social network," in *2019 IEEE 1st International Conference on Energy, Systems and Information Processing (ICESIP)*, pp. 1–5, IEEE, 2019.

[9] C. Lu, K. Ye, W. Chen, and C.-Z. Xu, "Adgs: Anomaly detection and localization based on graph similarity in container-based clouds," in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 53–60, IEEE, 2019.

[10] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, ACM, 2016.

[11] G. Biau, "Analysis of a random forests model," *Journal of Machine Learning Research*, vol. 13, no. Apr, pp. 1063–1095, 2012.