

Enhancing Signed Graph Attention Network by Graph Characteristics: An Analysis

Panatda Kaewhit¹, Chanun Lewchalermvongs¹, Phakaporn Lewchalermvongs²

¹Department of Mathematics, Faculty of Science, Mahidol University, Bangkok, Thailand

²Mahidol University, Nakhon Pathom, Thailand

Abstract – A graph neural network (GNN) is one of successful methods for handling tasks on a graph data structure, e.g. node embedding, link prediction and node classification. GNNs focus on a graph data structure that must aggregate messages on nodes in the graph to retain a graph-structured information in new node's message and proceed tasks on a graph. One of modifications on the propagation step in GNNs by adopting attention mechanism is a graph attention network (GAT). Applying this modification to signed graphs generated by sociological theories is called signed graph attention network (SiGAT). In this research, we utilize SiGAT and create novel graphs using graph characters to assess the performance of SiGAT models embedded in nodes across various characteristic graphs. The primary focus of our study was linked prediction, which aligns with the task employed in the previous research on SiGAT. We propose a method using graph characteristics to improve the time spent on the learning process in SiGAT.

Keywords – Graph neural network, graph attention network, signed graph attention network, graph characteristics, graph theory.

DOI: 10.18421/TEM132-05

<https://doi.org/10.18421/TEM132-05>

Corresponding author: Chanun Lewchalermvongs,
Department of Mathematics, Faculty of Science, Mahidol University, Bangkok, Thailand


Email: chanun.lew@mahidol.edu

Received: 03 November 2023.

Revised: 16 February 2024.

Accepted: 22 February 2024.

Published: 28 May 2024.

 © 2024 Panatda Kaewhit, Chanun Lewchalermvongs & Phakaporn Lewchalermvongs; published by UIKTEN. This work is licensed under the Creative Commons Attribution-NonCommercial-NoDeriv 4.0 License.

The article is published with Open Access at <https://www.temjournal.com/>

1. Introduction

A graph is a discrete structure that comprises vertices (also known as nodes) and edges that connect these vertices. Graph models can be used to solve problems in various disciplines since they can represent complex data as graphs of relationships between objects. A graph neural network (GNN) is a type of artificial neural network (ANN) that applies ANNs to graphs. The power of GNNs in modelling the dependencies between nodes in a graph has resulted in breakthroughs in research related to graph analysis. Recently, GNNs have been adapted with the attention mechanism to weight node features in the aggregating task, leading to the creation of graph attention networks (GATs). Unlike general weighting of neural networks, GATs use self-attention, which weights the nodes' features.

To address the limitations of GATs in handling signed graphs, which are graphs with positive and negative edges, signed graph attention networks (SiGATs) were developed by incorporating graph characters generated by sociological theories such as balance theory and status theory. SiGAT is a variant of graph convolutional networks that incorporates attention mechanisms, enabling it to selectively attend to positive or negative edges in signed graphs. SiGAT is a relatively new concept in GNN that has gained popularity in recent years. While SiGAT has shown great promise in various applications, it is not without limitations. One of the main limitations of SiGAT is its sensitivity to the graph structure. SiGAT performs well when the graph is fully connected, but its performance degrades as the sparsity of the graph increases. Additionally, SiGAT can be computationally expensive, especially for large graphs. Graph characteristics are important in GNNs because they provide valuable information about the structure and properties of the graph. GNNs are designed to operate on graph-structured data, where the nodes and edges represent entities and relationships between them.

By considering graph characteristics, GNNs can effectively leverage the inherent structure of the graph to perform various tasks, such as node classification, link prediction, and graph-level prediction [4], [9]. A subgraph is a smaller graph that is formed by selecting a subset of the vertices and edges from a larger graph. In other words, a subgraph is a graph that can be obtained from another graph by deleting some of its vertices and edges. The subgraph retains the structure and properties of the original graph, but may have fewer vertices and edges. A subgraph can be used to analyze a specific subset of a graph, or to simplify a graph by removing some of its complexity. Subgraphs are an important concept in graph theory and are used in many applications such as network analysis, data mining, and social network analysis. Subgraphs provide a localized view of the graph by focusing on a subset of nodes and edges. GNNs operate on these subgraphs, enabling them to capture local information and dependencies. By considering subgraphs, GNNs can leverage the neighborhood information around each node, allowing for localized information propagation and learning. Working with the entire graph can be computationally expensive, especially for large-scale graphs. Subgraphs provide a way to reduce the computational complexity by considering only a subset of the graph. This allows GNNs to scale to larger graphs and makes training and inference more efficient. In this study, we employed SiGAT and generated new graphs using graph characters to evaluate the node-embedded performance of SiGAT models across different characteristic graphs. We used the characteristic graph list from a previous study in [12] and the Bitcoin Alpha dataset [15], [21] to evaluate SiGAT's performance. We also generated a new characteristic graph list based on balance theory to assess the time spent on the models. Finally, we created new datasets to train the balance-theory-generated models to examine the types of graphs suitable for SiGAT models and to confirm the performance of characteristic graph lists. Our main task was linked prediction, which is the same task used in the previous study on SiGAT.

The remainder of this article is organized as follows. The related work and background are discussed in Section 2. The proposed methods and illustrated examples are presented in Sections 3, 4 and 5. Finally, we conclude and discuss future work in Section 6.

2. Preliminaries

ANNs are computational systems inspired by biological neural networks, closely resembling the human brain's operations, and excel in various problem-solving tasks.

They utilize a directed graph structure with nodes representing neurons and edges denoting connections. Nodes are organized into input, hidden, and output layers, facilitating data flow. ANNs operate through interconnected units that simulate neural signal transmission. The perceptron, the basic unit, has evolved into the multi-layer perceptron to address computational limitations. Forward and backward propagation processes are inspired by biological neural networks' action potentials, facilitating learning. ANNs have proven effective in various domains, such as estimating vapor pressures, stock price analysis, gas price forecasting, and bankruptcy prediction, highlighting their versatility and accuracy in different applications [11], [6], [18], [3]. GNNs, a subset of ANNs, are designed for graph data. Adapting them to diverse graph types is a current challenge. GNNs use message-passing models to aggregate node information, making them suitable for tasks like link prediction and classification. The concept was initially introduced by Gori [8] and Scarselli [17]. Different aggregation methods have led to various GNN models, including the influential Graph Convolutional Network (GCN) by Kipf and Welling in 2017 [14]. GCN applies spectral graph theory to perform convolutions on graph-structured data, capturing both local and global information. Extensions like GAT [20] and diffusion convolutional neural network (DCNN) [2] have followed. GNNs have practical applications in fields like quantum chemistry and reinforcement learning [7], [13], [16]. However, challenges remain regarding expressiveness, efficiency, and model robustness.

2.1. Graph Attention Network

A GAT, introduced by Velickovic *et al.* [7], is a model that leverages the attention mechanism to perform graph aggregation. While attention is typically used in artificial neural networks to assign weights to the inputs, GATs utilize self-attention, which assigns weights to the feature elements of each node based on their importance. By doing so, the self-attention mechanism helps to identify the most relevant features of each node, which can be used for downstream tasks such as classification or regression.

In a GAT, the input data for each node is represented as a vector $\vec{h}_i \in \mathbb{R}^F$, where F is the number of features. The goal of the GAT is to aggregate this input data in a way that captures important relationships between nodes. This is done by computing a new feature vector $\vec{h}'_i \in \mathbb{R}^{F'}$ for each node $i = \{1, 2, \dots, N\}$, where F' is the number of new features after aggregation. The aggregation process involves a linear transformation parametrized by a weight matrix $W \in \mathbb{R}^{F' \times F}$, followed by an attention operation.

The attention operation assigns weights to each of the neighboring nodes, based on the relevance of their features to the target node. Finally, the aggregated features are smoothed by an activation function σ , which adds non-linearity to the model. Equation 1 represents an aggregation function that combines features of all neighboring nodes j of a given node i , where N_i is the neighbors of node i .

$$\vec{h}'_i = \sigma(\sum_{j \in N_i} \alpha_{ij} W \vec{h}_j) \quad (1)$$

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{j \in N_i} \exp(e_{ij})} \quad (2)$$

$$e_{ij} = \text{LeakyReLU}(\vec{a}^T [W \vec{h}_i || W \vec{h}_j]) \quad (3)$$

This aggregation process involves a linear transformation, parameterized by the weight matrix $W \in \mathbb{R}^{F' \times F}$, and an attention mechanism based on α_{ij} . The output of this aggregation is then smoothed by an activation function σ .

In Equation 3, the attention mechanism \vec{a}^T acts as the coefficient of the LeakyReLU function. This means that the alignment model score e_{ij} is influenced by the vector element obtained by concatenating the transformed feature vectors \vec{h}_i and \vec{h}_j . The LeakyReLU nonlinearity function is an activation function that applies the rectified linear unit (ReLU) function with a negative slope (Figure 1). Finally, to prepare for the aggregation step as in Equation 2, the alignment model score e_{ij} is normalized using the softmax function to obtain α_{ij} .

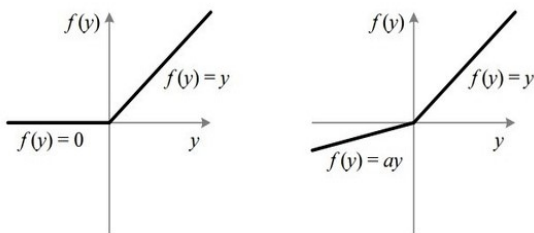


Figure 1. The left is ReLU function and the right is LeakyReLU function

2.2. Signed Graph Attention Network

Signed graph attention network (SiGAT) is a type of graph neural network that has been gaining attention in recent years. SiGAT is a variant of graph convolutional networks that incorporates attention mechanisms, enabling it to selectively attend to positive or negative edges in signed graphs.

One of the earliest works on SiGAT was proposed by Derr *et al.* [5].

The authors introduced SiGAT, which is a graph convolutional network that uses both node-level and edge-level attention mechanisms. The node level attention mechanism enables SiGAT to selectively attend to positive or negative nodes, while the edge-level attention mechanism enables it to selectively attend to positive or negative edges. The authors also proposed a graph diffusion process to enable SiGAT to learn from multi-hop graph structures. In 2019, Huang *et al.* [12] proposed a modified SiGAT architecture that uses both global and local attention mechanisms. The global attention mechanism considers all the nodes and edges in the graph, while the local attention mechanism focuses only on the immediate neighbors of each node. The authors also introduced a signed diffusion process to enable SiGAT to learn from multi-hop graph structures. SiGAT has been applied to various real-world applications, including sentiment analysis, recommendation systems, and fraud detection. Signed graphs have been utilized to model and interpreted voting patterns [1]. The signed graph analysis is valuable in interpreting voting behavior. In the following sections, we will provide a detailed explanation of how GAT operates with characteristic graphs.

2.2.1. Balance Theory and Status Theory

SiGAT utilizes sociological theories, such as balance and status theories, to create a characteristic graph for its operations. The status theory establishes a directed relationship between objects (nodes), with one object being higher or lower than the other. Since SiGAT operates within a signed graph domain, the status is labeled as positive or negative. For instance, a positive relation between object A and object B means that B has a higher positive status than A. Conversely, a negative relation of object A to object B means that B has a higher negative status than A. In the second row of Figure 2, all triads satisfy the status theory by fulfilling “status(j) > status(i)” and “status(k) > status(j)”, resulting in “status(k) > status(i)” and forming a complete circle triad. All status triads in Figure 2 are combinations of balance triads, which are further elaborated below. Balance theory, which originated in social psychology during the mid-20th century, was initially developed as a model for undirected signed networks [10]. According to the balance theory, a triad in an undirected signed network is considered balanced when all signs within the triad are positive or when only one sign is positive, and the others are negative. Suppose a positive refers to a friendship relationship and a negative refers to an enemy relationship.

Then the first two triads in the first row of Figure 2 can be described as “the friend of my friend is my friend” and “the enemy of my enemy is my friend,” respectively. These two relationships are considered balanced. Conversely, the last two triads in the first row of Figure 2 are unbalanced.

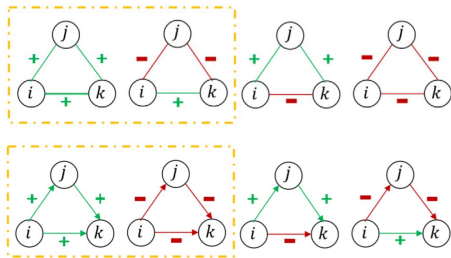


Figure 2. All triads in the first row are balance triads and all triads in the second row are status triads [12]

SiGAT leverages the balance and status theories to generate characteristic graphs in its method. There are 38 characteristic graphs, which are shown in Figure 3.

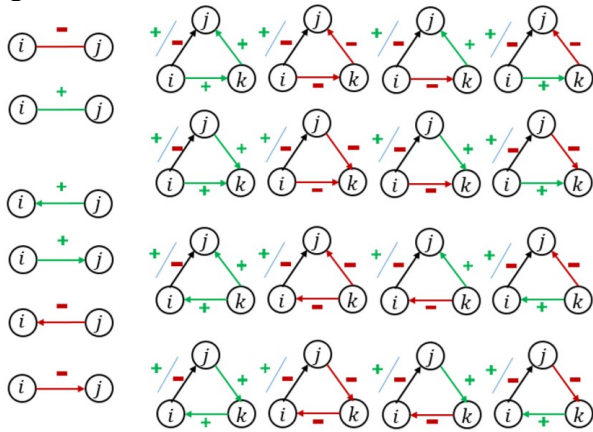


Figure 3. The 38 characteristic graphs in SiGAT [12]

2.2.2. SiGAT Process

Let $G(V, E, s)$ be a signed directed graph, let $\vec{h}_{m_i}(u) \in \mathbb{R}^F$ be the feature vector of a node $u \in V$, and let $\vec{h}'_{m_i}(u) \in \mathbb{R}^{F'}$ be the new feature vector of u after aggregation, where F is the number of features and F' is the number of new features after aggregation according to each characteristic graph in the list M and $i \in \{1, 2, \dots, |M|\}$. The SiGAT algorithm is shown below. The SiGAT algorithm is an algorithm that embeds the node $u \in V$ of $G(V, E, s)$ into a representation Z_u . This purpose is inherited from GAT. The starting section explains the characteristic graphs involved in the method, which is different from GAT. The characteristic graphs $m_i \in M$ are generated and fed into the characteristics graph extract function F_{m_i} .

The characteristic graphs extract function is used to extract a subgraph of the graph G that contains characters similar to the characteristic graphs, as shown in Figures 4 and 5.

Algorithm SiGAT embedding generating algorithm

Input:

- 1: Signed directed graph $G(V, E, s)$
- 2: Characteristic graphs list M
- 3: Characteristic graphs extract function $F_{m_i}; \forall m_i \in M$
- 4: $GAT - AGGREGATOR_{m_i}$ with parameters $W_{m_i}, a_{m_i}; \forall m_i \in M$
- 5: Non-linear function Tanh with parameters weight matrices W_1, W_2 and bias b_1, b_2

Output:

- 1: Node representation $Z_u, \forall u \in V$

Initialization:

- 1: Iterations T
- 2: Batch Size B
- 3: $\vec{h}_u \leftarrow \text{random}(0,1), \forall u \in V$
- 4: $G_{m_i} \leftarrow F_{m_i}(G), \forall m_i \in M$
- 5: $N_{m_i}(u) \leftarrow \{v | (u, v) \in G_{m_i}\}, \forall m_i \in M, \forall u \in V$

SiGAT:

- 1: **for** iteration = 1, ..., T **do**
- 2: **for** batch = 1, ..., $|V|/B$ **do**
- 3: $B \leftarrow V_{(batch-1) \times B + 1 : batch \times B}$
- 4: **for** $u \in B$ **do**
- 5: **for** m_i in M **do**
- 6: $h'_{m_i}(u) \leftarrow GAT - AGGREGATOR_{m_i}(\{h(v), \forall v \in N_{m_i}(u)\})$
- 7: **end for**
- 8: $h'(u) \leftarrow CONCATENATE(h(u), h'_{m_1}(u), \dots, h'_{m_{|M|}}(u))$
- 9: $Z_u \leftarrow W_2 \cdot \text{Tanh}(W_1 \cdot h'(u) + b_1) + B_2$
- 10: **end for**
- 11: **end for**
- 12: **end for**

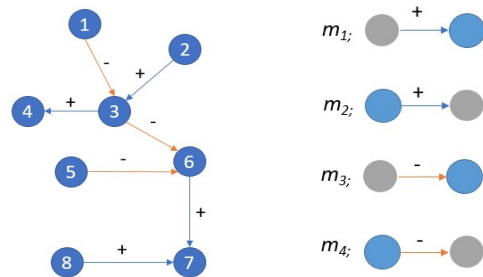


Figure 4. On the left is an example of graph G , and on the right is an example of graph characters

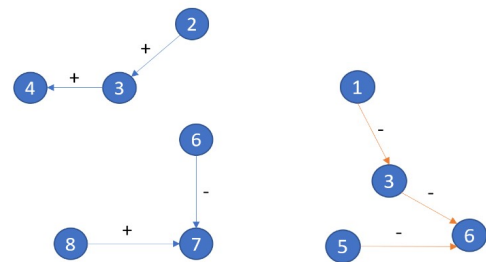


Figure 5. Extracting a graph is required for each graph character. On the left, the graph is extracted from m_1 or m_2 , and on the right, the graph is extracted from m_3 or m_4

For each iteration, SiGAT works on a batch in \mathcal{B} . In Figure 6, the operation of nodes in the batch on SiGAT gives new represented features of nodes in the batch. In Figure 7, $h(u)$ is fed into aggregators with $N_{m_i}(u)$ and gives $\vec{h}_{m_i}(u)$ back. After that, the concatenated thing $\vec{h}(u) \in \mathbb{R}^{|M|F'+F}$ is formed from $\vec{h}_{m_i}(u)$ for all $m_i \in M$, and $\vec{h}(u)$ is input terms of the nonlinearity tanh function with parametrized weight matrices W_1, W_2 and bias b_1, b_2 .

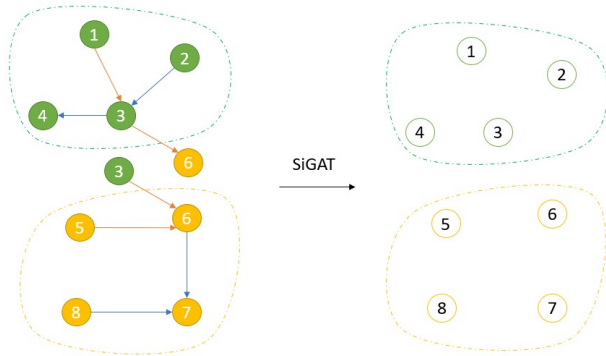


Figure 6. SiGAT operates on each batch by grouping nodes of graph G . In the figure, the green batch and yellow batch are examples of batches that SiGAT operates on

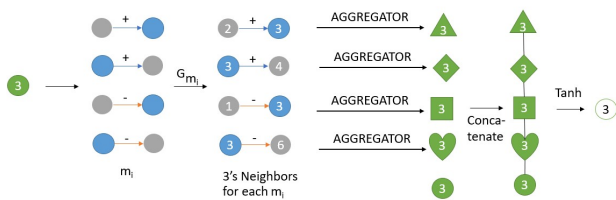


Figure 7. SiGAT's operation on the example node-3 in the green batch

$GAT - AGGREGATOR_{m_i}$ is formed by equations (1), (2), and (3). Subsequently, backward propagation computes the loss, which is defined as an unsupervised loss function below.

$$J(Z_u) = -\sum_{v^+ \in N(u)^+} \log(\sigma(Z_u^T Z_{v^+})) - Q \sum_{v^- \in N(u)^-} \log(\sigma(Z_u^T Z_{v^-})), \quad (4)$$

where σ is a sigmoid function that maps real numbers to the range (0,1), given by the equation $f(x) = \frac{1}{1+e^{-x}}$, $x \in \mathbb{R}$. $N(u)^+$ is the set of positive neighbors of node u , and $N(u)^-$ is the set of negative neighbors of node u . The balance parameter Q is defined as the ratio of the number of positive neighbors to the number of negative neighbors, i.e., $= \frac{|N(u)^+|}{|N(u)^-|}$. This loss function reflects the idea that “friend embeddings are similar, and enemy embeddings are dissimilar.”

To decrease the time required for learning in SiGAT, a proposed framework involves conducting experiments on diverse characteristic graph lists and utilizing specific graph datasets to analyze the properties that impact the SiGAT model. By examining the effects of various graph characteristics on the model's performance, this approach aims to identify key factors that can improve the learning process.

3. Diverse Characteristic Graph Lists

Status theory deals with hierarchical relationships, where individuals or entities can be ranked higher or lower than others. For this reason, status theory is particularly suitable for directed graph datasets, which have sources and sinks that represent the origins and destinations of these relationships. The original SiGAT model, which had 38 characteristic graph lists (SiGAT-38), was revised to incorporate both status theory and balance theory. However, we are particularly interested in the characteristic graphs from the balance theory, which are neither as complex as SiGAT-38 nor as simple as the two characteristic graphs from status theory, which consist of arrowheads pointing in and out without any sign. In the next section, we will present two subsections detailing the process of generating characteristic graph lists and the experimental results obtained using SiGAT-38 and the generated lists. Characteristic graph lists are generated from two concepts, the first of which is based on SiGAT-38's undirected concept, and the second one is the base concept which takes into account the societal connections within the network.

3.1. Undirected Concept

The *Undi(8)* graph list is similar to SiGAT-38's approach, but with no direction on edges. In this list, the blue target node i collects orange neighbor nodes j that have a triad relation with gray nodes. This list uses both balanced and unbalanced triads so that the machine can learn from all cases. There are 8 characters in this list, as shown in Figure 8.

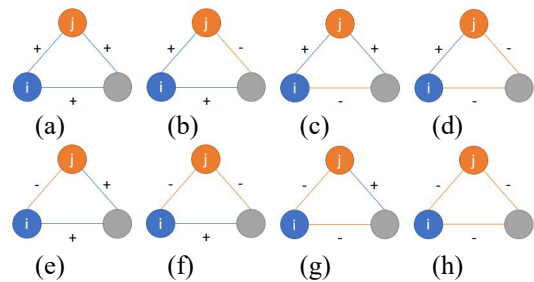


Figure 8. The *Undi(8)* graph list

The *Undi(6)* graph list extends from *Undi(8)* and includes gray nodes as neighbors in addition to orange nodes. In *Undi(8)*, the target node *i* collects only orange neighbor nodes *j*, while *Undi(6)* also collects gray nodes as neighbors. In *Undi(6)*, the gray node is the same as the orange node *k* in *Undi(8)*, as shown in Figure 8. Since the triad with the target node is a symmetrical structure, it can be observed that the two triads (c) and (e) are similar to (d) and (f) in Figure 8. Therefore, this list has only 6 characters instead of 8.

The *Undi(8+6)* list is a combination of the *Undi(8)* and *Undi(6)* lists. The *Undi(8)** list is similar to the *Undi(8)* list but includes positive and negative signs, as shown in Figure 9. The *Undi(6)** list is similar to the *Undi(8)** list but based on the *Undi(6)* list. The *Undi(8+6)** list adds positive and negative signs to the *Undi(8+6)* list.

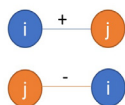


Figure 9. Positive and negative signs

3.2. Base Concept

In the initialization part of the SiGAT algorithm, a new graph is extracted for each characteristic graph. This function works by focusing on one node and finding neighbors of its node that match with the characteristic graph. After that, it explores like this with all nodes in the graph. The focused node is called the target node. SiGAT-38 has a matching characteristic graph between the target node and its neighbors. The second concept considers the matching among only the target's neighbors. However, the interested society is only in triad form that looks like a tetrahedron, where its spire is the target node and its base is the neighbors. Therefore, the second concept is called the base concept.

The base list uses balanced and unbalanced triads to match among the target's neighbors. However, it only collects neighbors that construct triad bases whose sum of signs of edges on the base equals the possible summation. A positive sign is 1, and a negative sign is -1. Therefore, the possible summations of triad bases are 3, 1, -1, and -3, as shown in Figure 10. There are 4 characteristic graphs contained in the *Bbse* list.

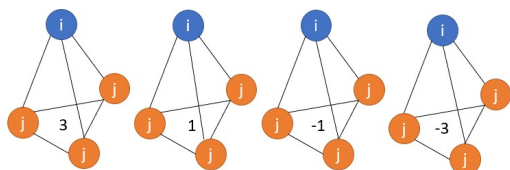


Figure 10. The Base list

The *fixedBase* list collects the neighbors *j* that form a triad base as depicted in Figure 11. The signs of the edges between the neighbors *j* and the gray nodes are denoted as *s1* and *s2*, while the sign of the edge between the two gray nodes is denoted as *s3*, as shown in Figure 12. The characteristic graphs are obtained by considering the values of *s1+s2* and *s3*, which result in six possible cases: *s1+s2* can be 2, 0, or -2, and *s3* can be 1 or -1.

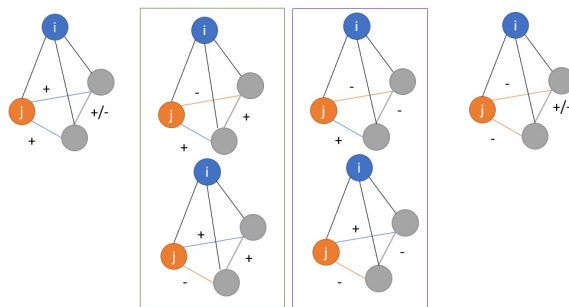


Figure 11. The *fixedBase* list

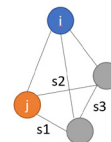


Figure 12. The *fixedBase* concept

The *Base-Side1* list represents the characteristic graph on the base and side of the tetrahedron. It employs the base characters from the base list and the side characters from the triad summation of the base list, resulting in 8 possible characters as shown in Figure 13.

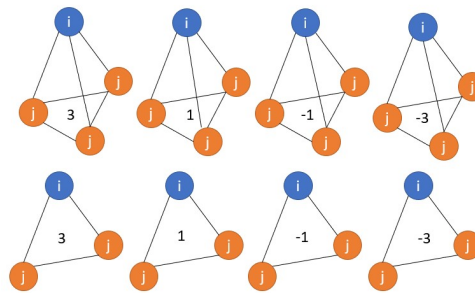


Figure 13. The *Base-Side1* list

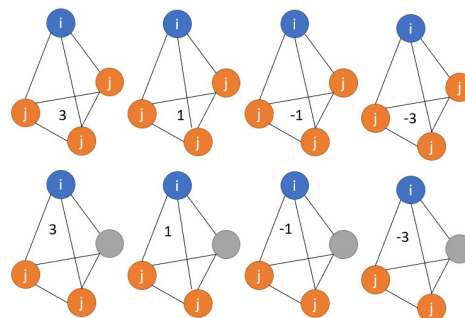


Figure 14. The *Base-Side2* List

The Base-Side2 list also contains the characteristic graph of the base and side, similar to Base-Side1. However, the side characters in Base-Side2 are different. Base-Side1 collects all neighbors j , regardless of whether they construct base triads. In contrast, Base-Side2 only collects the neighbors j that must construct base triads, as shown in Figure 14.

The Base-Side1* and Base-Side2* lists are similar to Undi(8)* and Undi(6)*, respectively, and include both positive and negative signs with the Bas-Side1 list and Base-Side2 list, respectively.

3.3. Illustrated Example: Bitcoin Alpha Dataset

We utilized the Bitcoin Alpha dataset, obtained from the UCI machine learning repository [2] to demonstrate our method and compare it with other datasets. The results of our analysis are summarized in Table 1. Our findings indicate that SiGAT-38 has the minimum loss, while base has the least average time over 102 iterations of learning. Notably, Base-Side2 has the least time ratio, even though its average time is not the lowest. We observed that the time spent on learning is not solely dependent on the number of characteristic graphs in each list, except for cases where a list includes characters from another list. For instance, the Undi(8+6)* list contains positive and negative signs in addition to the Undi(8+6) list, resulting in a higher number of characteristic graphs compared to Undi(8+6). As a result, the average time for Undi(8+6)* is greater than that of Undi(8+6). To optimize time spent during learning, we recommend identifying relevant characteristic graphs that are specific to the dataset and using only those for optimal results.

Table 1. The loss and time results of various characteristic graphs lists' experiments conducted on Bitcoin Alpha.

Lists	Minimum loss	Average time	#graphs in list	Time ratio
SiGAT-38	1105.76	13.38	38	0.35
Undi(8)	1589.20	5.82	8	0.73
Undi(8)*	1206.94	5.67	10	0.57
Undi(6)	1756.49	5.11	6	0.85
Undi(6)*	1481.30	6.09	8	0.76
Undi(8+6)	1446.06	5.46	10	0.55
Undi(8+6)*	1159.20	10.38	12	0.86
Base	7613.17	1.45	4	0.36
fixedBase	7087.21	1.75	4	0.44
Base-Side1	7266.56	2.14	8	0.27
Base-Side1*	7119.32	2.88	10	0.29
Base-Side2	7286.15	2.12	8	0.27
Base-Side2*	7150.22	2.87	10	0.29

Remark - for binary classification problems with two classes, positive and negative, the accuracy value is equivalent to the micro F1-score. This means that if we have a balanced dataset with equal numbers of positive and negative samples, the accuracy score and micro F1-score will be the same. However, in cases where the dataset is imbalanced, the micro F1-score may be a better metric for evaluating model performance than accuracy. This is because the micro F1-score takes into account both precision and recall, providing a more balanced evaluation of the model's performance. Therefore, in situations where we have a highly imbalanced dataset, it is important to consider other evaluation metrics, such as F1-score, that provide a more comprehensive picture of the model's performance.

The Bitcoin Alpha dataset has a pos-ratio of 0.93649, indicating that it is highly imbalanced. This means that the area under the curve (auc) value cannot be used as a reliable metric for evaluating the performance of models trained on this dataset. Therefore, other metrics such as F1-score of positive (F1-score(+)), F1-score of negative (F1-score(-)), macro-F1-score, and micro-F1-score are more appropriate for evaluating model performance. These metrics are calculated using the confusion matrix, which provides information on the model's ability to accurately classify positive and negative samples. By evaluating the model on both macro and micro levels, we can get a better understanding of the model's performance, and we can focus on the performance of each class using F1-score(+) and F1-score(-).

When analyzing the undirected concept in Table 2, we can see that both Undi(8) and Undi(6) outperform Undi(8+6). Although Undi(8) and Undi(6) have similar performance, the macro-F1-score, micro-F1-score, and F1-score(+) of Undi(6) are better than those of Undi(8) due to the highly imbalanced nature of the Bitcoin Alpha dataset. The performance of classifying negative samples is higher, resulting in better F1-score(+) and micro F1-score values for Undi(6). However, Undi(8) outperforms Undi(6) in terms of F1-score(-). Moreover, Undi(8)* and Undi(8+6) perform better than Undi(8) and Undi(8+6), respectively. On the other hand, Undi(6)* performs worse than Undi(6). These findings illustrate the importance of choosing appropriate metrics when evaluating models trained on imbalanced datasets.

In Table 3, Base-Side2 outperforms fixedBase for the base concept. Base-Side1* performs better than the original but not as well as Base-Side2. Base and Base-Side1 have the lowest F1-score(-) due to their inability to predict any negative signs correctly.

This can be seen from the recall(+) value of 1, indicating that all actual positive signs were predicted correctly, while the recall(-) value of 0 indicates that no actual negative signs were predicted. Figure 15 illustrates this situation.

Table 2. The results of analyzing the undirected concept's experiments conducted on the Bitcoin Alpha datasets.

Bitcoin Alpha	Undi(8)	Undi(8)*	Undi(6)
precision(+)	0.95	0.96	0.95
precision(-)	0.58	0.60	0.64
recall(+)	0.99	0.99	0.99
recall(-)	0.27	0.32	0.26
F1-score(+)	0.97	0.97	0.97
F1-score(-)	0.37	0.42	0.36
Macro-F1-score	0.67	0.69	0.67
Micro-F1-score	0.94	0.95	0.95
AUC score	0.87	0.89	0.85

Bitcoin Alpha	Undi(6)*	Undi(8+6)	Undi(8+6)*
precision(+)	0.95	0.95	0.94
precision(-)	0.59	0.57	0.95
recall(+)	0.99	0.99	0.57
recall(-)	0.26	0.24	0.26
F1-score(+)	0.97	0.97	0.97
F1-score(-)	0.36	0.33	0.36
Macro-F1-score	0.67	0.65	0.66
Micro-F1-score	0.94	0.94	0.94
AUC score	0.87	0.87	0.88

Next, we focus on four characteristic graph lists: Undi(8), Undi(6), Base, and fixedBase. While Undi(8) and Undi(6) demonstrate solid performance, Undi(8+6) is simply a combination of the two. Both Base and fixedBase only consider triads in the neighborhood of the target node, whereas Base-Side1 and Base-Side2 consider triads of the target node and its neighbors. Comparing these concepts with SiGAT-38, SiGAT-38 exhibits the best performance. The undirected concept is closer to SiGAT38 than the base concept, and the minimum loss of the undirected concept is also similar to SiGAT-38. However, the average computation time of the base concept is much lower than that of SiGAT-38.

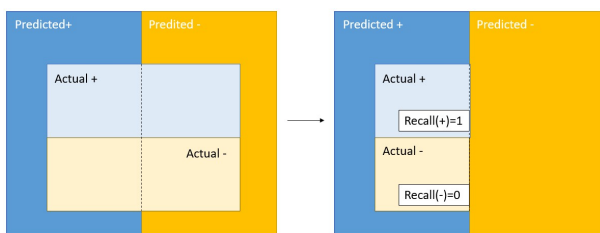


Figure 15. A special case where recall(+) equals 1 and recall(-) equals 0

Table 3. The results of analyzing the base concept's experiments conducted on the Bitcoin Alpha datasets

Bitcoin Alpha	Base	fixedBase	Base-S1
precision(+)	0.94	0.94	0.94
precision(-)	0.00	0.33	0.00
recall(+)	1.00	1.00	1.00
recall(-)	0.00	0.01	0.00
F1-score(+)	0.97	0.97	0.97
F1-score(-)	0.00	0.01	0.00
Macro-F1-score	0.48	0.49	0.48
Micro-F1-score	0.94	0.94	0.94
AUC score	0.71	0.74	0.73

Bitcoin Alpha	Base-S1*	Base-S2	Base-S2*
precision(+)	0.94	0.94	0.94
precision(-)	0.63	0.60	0.00
recall(+)	1.00	1.00	1.00
recall(-)	0.02	0.01	0.00
F1-score(+)	0.97	0.97	0.97
F1-score(-)	0.03	0.02	0.00
Macro-F1-score	0.50	0.49	0.48
Micro-F1-score	0.94	0.94	0.94
AUC score	0.74	0.75	0.71

4. Specific Graphs Datasets

It is not possible to develop an approach that can solve or learn all the various properties of datasets. Therefore, this section focuses on specific graph datasets that we have generated to evaluate the SiGAT model using interesting characteristic graph lists. In the previous section, four characteristic graph lists were discussed, namely Undi(8), Undi(6), Base, and fixedBase. However, for this section, we are interested in two characteristic graph lists, namely Undi(8) and fixedBase. The reason for this is that these two lists only collect the neighbors of a target node once for each target's neighbor finding, which is different from Undi(6) and Base.

In the next subsections, we divide the generated graphs used for experimentation into three groups. All generated graphs are in the signed directed domain for experimentation, and the models ignore the directions of nodes on edges for undirected graphs, except for the SiGAT-38 indicator.

4.1. Random Graphs

The random graph group refers to graphs generated with random edges, signs, and directions. Within this group, we created four datasets: graph1, graph2, graph1+2, and graph3.

Graph1 contains 1,632 nodes and 8,260 edges, created as a connected component graph, similar to graph2, which has 1,648 nodes and 8,444 edges. Graph1+2 is a combination of graph1 and graph2, resulting in a graph with 3,280 nodes, 16,704 edges, and two connected components. Graph3 is also a combination of graph1 and graph2 but with an additional edge to link the components. This results in a graph with 3,281 nodes and 16,705 edges.

As the graphs used in this study are randomly generated, it resulted in the loss of extracted graphs on fixedBase. These generated graphs lack the characteristics of fixedBase’s characteristic graphs, which means that no experiments could be conducted on fixedBase.

Regarding Table 4, the minimum loss of graph1, graph1+2, and graph3 are likely decreasing in that order. This is interesting because graph1 is a part of graph1+2 and graph3, yet the minimum loss of these two graphs is less than graph1’s loss. The minimum loss of Undi(8) increases from SiGAT-38, while the time spent on Undi(8) decreases. The average time spent is rising with the graph1, graph3, and graph3 sequences. For graph1+2 and graph3, which have a different part consisting of only one edge, the difference in average time is larger than the difference in the part. It is necessary to evaluate the values, such as AUC and F1-score, as they are around 0.5. This means that the SiGAT-38 and Undi(8) models cannot classify the signs of edges, and hence, they are not effective.

Table 4. The loss and time results of random graph’s experiments conducted on Bitcoin Alpha

Loss minimum			
Dataset	SiGAT-38	Undi(8)	fixedBase
Graph1	88.15	2435.80	-
Graph1+2	62.01	872.55	-
Graph3	62.08	872.51	-
Time average			
Dataset	SiGAT-38	Undi(8)	fixedBase
Graph1	4.95	3.11	-
Graph1+2	20.74	7.46	-
Graph3	19.16	9.10	-

4.2. Triangle Component Graph

The triangle component graph is created to check the effect of the number of connected components on the graph. This group is created by connecting triangles to construct connected components, where all the characteristic graphs are triangles. We generate these graphs to determine the extent to which the number of triangles affects the models.

We generated graph-c(n), which has around 2,500 nodes and 8,500 edges for n in {1,2,3,4,5}. However, the evaluation values for these graphs are approximately 0.5, similar to the random graph, indicating that the models are unclassified. To test on larger graphs, we generate graph-c2(n), which has 3,000 nodes and approximately 62,500 edges, but the evaluation values are still around 0.5. We observe that the minimum loss of graph-c(n) and graph-c2(n) increases for Undi(8) when compare with SiGAT-38, but decrease for fixedBase. The results presented in Table 5 indicate that the average time is consistently the best for fixedBase, followed by Undi(8) and SiGAT-38. However, the primary purpose of generating these graphs is to check the effect of the number of connected components and the number of triangles, but the effect of the number of connected components is not evident. Furthermore, the effect of the number of triangles involves time spent since there are more nodes and neighbors to consider.

Table 5. The loss and time results of graph-c(n) and graph-c2(n) experiments conducted on Bitcoin Alpha

Minimum loss			
Dataset	SiGAT-38	Undi(8)	fixedBase
Graph-c(1)	39.83	121.25	0.69
Graph-c(2)	39.83	114.66	3.11
Graph-c(3)	33.72	113.65	9.84
Graph-c(4)	67.25	149.27	14.03
Graph-c(5)	145.16	200.16	52.48
Graph-c2(1)	21976.22	45043.43	356.67
Graph-c2(2)	19031.85	43311.19	864.20
Graph-c2(3)	18665.73	43198.74	2136.00
Graph-c2(4)	19568.09	43978.39	3579.60
Graph-c2(5)	18088.79	42866.53	6440.54

Average time			
Dataset	SiGAT-38	Undi(8)	fixedBase
Graph-c(1)	13.02	5.84	0.23
Graph-c(2)	13.02	4.43	0.47
Graph-c(3)	11.35	5.12	0.56
Graph-c(4)	11.77	4.83	0.52
Graph-c(5)	11.00	5.53	0.56
Graph-c2(1)	24.22	9.41	1.07
Graph-c2(2)	24.15	9.49	1.64
Graph-c2(3)	23.28	9.58	2.55
Graph-c2(4)	24.67	9.84	3.44
Graph-c2(5)	36.69	9.58	4.82

We also generate graph-c-p0.8(n) to simulate the imbalance of Bitcoin Alpha. The positive probability is set to 0.8.

The results presented in Table 6 indicate that the minimum loss and average time were similar to graph-c(n) and graph-c2(n), but the evaluation values are weird, with a high F1-score(+) and a low F1-score(-) for SiGAT-38 and Undi(8), and zero for fixedBase because the model has no negative predictions, as observed in the Bitcoin Alpha experiment. To address this issue, we generate graph-c-p0.8(n)* by removing the edge with the same nodes tuple. For instance, if edge e_1 goes from node a to node b with a positive sign, and edge e_2 goes from node b to node a, with a negative sign, we cut one of them. Consequently, graph-c-p0.8(n)* has no same nodes tuple of edges. The data presented in Table 5 suggests that the F1-score(-) is likely to be higher than the original, resulting in an increase in the AUC score to around 0.7 for both SiGAT-38 and Undi(8). However, for fixedBase, only components 4 and 5 have characteristics, resulting in the graph having only 4 and 5 components. Similar to graph-c-p0.8(n), it does not classify the negative class.

Table 6. The loss and time results of graph-c-p0.8(n) and graph-c-p0.8*(n) experiments conducted on Bitcoin Alpha

Minimum loss			
Dataset	SiGAT-38	Undi(8)	fixedBase
Graph-c-p0.8(1)	25.27	81.53	14.25
Graph-c-p0.8(2)	36.22	101.69	20.25
Graph-c-p0.8(3)	40.20	98.03	17.24
Graph-c-p0.8(4)	79.06	139.55	57.47
Graph-c-p0.8(5)	65.23	135.37	45.48
Graph-c-p0.8*(1)	14.34	80.12	-
Graph-c-p0.8*(2)	13.94	77.06	-
Graph-c-p0.8*(3)	15.12	75.57	-
Graph-c-p0.8*(4)	25.24	68.40	5.61
Graph-c-p0.8*(5)	15.47	80.02	11.81

Average time			
Dataset	SiGAT-38	Undi(8)	fixedBase
Graph-c-p0.8(1)	10.81	10.81	0.50
Graph-c-p0.8(2)	9.93	4.26	0.49
Graph-c-p0.8(3)	9.94	4.28	0.52
Graph-c-p0.8(4)	11.43	4.27	0.55
Graph-c-p0.8(5)	10.78	6.31	0.51
Graph-c-p0.8*(1)	10.60	4.12	-
Graph-c-p0.8*(2)	9.91	4.13	-
Graph-c-p0.8*(3)	9.85	4.53	-
Graph-c-p0.8*(4)	12.50	4.37	0.19
Graph-c-p0.8*(5)	10.60	4.67	0.42

4.3. Tree Graph

In our experimentation with planar graph datasets, we specifically focused on trees. A planar graph is a graph that can be embedded in the plane without any edges crossing each other. A tree is an undirected graph that is connected and acyclic, which means it contains no cycles (loops). We generate two tree datasets: tree-d2 and tree-d10, each consisting of 3,000 nodes and 2,999 edges. The tree-d2 dataset is a tree where each node has a degree of at most 2, while the tree-d10 dataset has a degree of at most 10. In this study, the degree is defined as the combination of in-degree and out-degree.

However, since trees are connected graphs that contain no cycles, Undi(8) and fixedBase models are not able to learn these graph structures. On the other hand, for the SiGAT-38 model, we use only six non-triad characters, named SiGAT-38*, (Table 7). Although the results are not shown, the model is able to learn the tree structures effectively, indicating its potential for learning complex planar graphs.

Table 7. The loss and time results of tree-d2 and tree-d10 experiments conducted on Bitcoin Alpha

SiGAT-38*	Minimum loss	Average time
tree-d2	2.74	3.21
tree-d10	2.80	3.02

5. Subgraph-based Learning Approach

In this part, we aim to improve the performance of our model on the Bitcoin Alpha dataset by using a new training and testing approach. Specifically, we focus on a subgraph-based learning strategy where we extract a subgraph from a larger graph, which is simulated from the Bitcoin Alpha dataset, by identifying a domination set. A domination set is a subset of vertices in a graph such that each vertex in the graph is either a part of the subset or adjacent to a vertex in the subset. Dominating sets are important in graph theory because they can be used to model a variety of real-world problems, such as facility location, sensor placement, and network routing. The domination number of a graph is the minimum size of a dominating set for the graph. Finding the domination number is an NP-hard problem.

By constructing a subgraph from the domination set, we can simplify the original graph while preserving its structural properties. This allows us to train and test our model on a smaller and more manageable subgraph, which can improve the efficiency and accuracy of our approach.

Overall, we expect this new subgraph-based learning approach to yield better results on the Bitcoin Alpha dataset than previous methods. To extract the subgraph, we first determine the domination number of the graph, which in this case is 1,706. Then the nodes are ranked in the graph by

their degree and the top nodes and their neighbors are selected to form the first subgraph, which we refer to as subgraph1. This subgraph contains a total of 3,623 nodes, which is close to the size of the original Bitcoin Alpha graph, which has 3,783 nodes. We then split subgraph1 into training and testing sets, with the testing set consisting of the difference between the Bitcoin Alpha graph and subgraph1. This allows us to evaluate the performance of our model on the remaining nodes in the graph. The resulting dataset is called Bitcoin Alpha 2, and the training set accounts for 80% of the total data.

However, in order to further improve the efficiency and accuracy of our model, we extract another subgraph called subgraph2. This subgraph contains nodes with a degree of less than 100 and their neighbors. We split subgraph2 into training and testing sets, with the testing set again consisting of the difference between the Bitcoin Alpha graph and subgraph2. The resulting dataset is called Bitcoin Alpha 3, and the training set accounts for 70% of the total data.

By using subgraph-based learning on these new datasets, we aim to better understand the underlying structure of the Bitcoin Alpha graph and improve the accuracy of our model. The use of multiple subgraphs with different node selection criteria can provide insights into which nodes and edges are most important for predicting node behavior, and can help us to develop more efficient and accurate machine learning algorithms for graph data.

The experimental results presented in Tables 8 and 9 show that the Bitcoin Alpha 2 dataset outperforms subgraph2 in terms of prediction accuracy, although this comes at the cost of increased time spent on training and testing the model. The Bitcoin Alpha 2 dataset contains 23,996 edges, which is comparable to the number of edges in the original Bitcoin Alpha dataset, which has 24,186 edges. In contrast, the Bitcoin Alpha 3 dataset has only 20,344 edges, which may explain its relatively poor performance. The lower number of edges, combined with the smaller percentage of nodes in the training set, may have resulted in insufficient information for the model to accurately predict node behavior. Overall, these results suggest that the choice of subgraph and its size can have a significant impact on the performance of machine learning algorithms for graph data, and highlight the importance of carefully selecting appropriate subgraphs for training and testing.

Table 8. The loss and time results of experiments conducted on Bitcoin Alpha 2 and Bitcoin Alpha 3

SiGAT-38	Minimum loss	Average time
Bitcoin Alpha 2	1002.89	13.50
Bitcoin Alpha 3	898.72	8.27

Table 9. The results of analyzing experiments conducted on Bitcoin Alpha 2 and Bitcoin Alpha 3

SiGAT-38	Bitcoin Alpha 2	Bitcoin Alpha 3
pos-ratio(test)	0.94	0.94
accuracy	0.95	0.95
precision(+)	0.96	0.96
precision(-)	0.68	0.66
recall(+)	0.99	0.99
recall(-)	0.36	0.30
F1-score(+)	0.97	0.98
F1-score(-)	0.47	0.42
Macro-F1-score	0.72	0.70
Micro-F1-score	0.95	0.95
AUC score	0.93	0.88

6. Conclusion and Discussion

SiGAT is a model that embeds nodes on a graph to learn tasks on graphs. This study focuses on linked prediction. Initial experiments on Bitcoin Alpha using various characteristic graph lists demonstrate that the number of graph characters is not relevant to the time spent on the SiGAT model. However, characteristic graph lists in an undirected concept provide evaluation results close to those of SiGAT-38, and they require only one third of the learning time. They are a good choice for further study on other tasks. Additionally, characteristic graph lists in a base concept have a time spent around one sixth of SiGAT-38 or one half of the undirected concept, even though their evaluating values are lower than the undirected concept. Users can weigh their advantages, performance, or time spent.

Next, a dataset is generated to test on chosen lists. Since the time spent is not relevant to the number of graphs in the list, it is assumed to be related to the number of neighbors on each graph in the list for each target node. Therefore, Undi(8) and fixedBase, which have good performance, are chosen. They only pick one neighbor for each target's exploration.

There are three groups of graphs to test on SiGAT-38, Undi(8), and fixedBase. Firstly, these models cannot classify signs on edges of random graphs. Secondly, testing is performed on triangle component graphs since most characters in the lists are in triangle form, and the effect of the number of components is checked.

Similarly, the models cannot classify on a triangle component graph. Next, the interest is in planar graphs. A tree graph is one of the planar graphs. Since a tree has no cycle, it can only learn on non-triad of SiGAT-38 called SiGAT-38*. Once again, the models cannot classify on a tree.

Furthermore, we perform experiments on new splitting dataset. Bitcoin Alpha is split into a new training and testing set to be learned. The new datasets involve the domination number and the addition of in-degree and out-degree of nodes. Datasets from the domination number have better performance than the original Bitcoin Alpha. In conclusion, SiGAT is a powerful concept in graph neural networks that has the potential to provide a more accurate representation of real-world phenomena. While SiGAT has some limitations, researchers continue to explore its potential applications and improve its performance. This study provides valuable insights into SiGAT's effectiveness on linked prediction tasks and various graph structures. The results suggest that undirected characteristic graphs with fewer learning times are suitable for extended studying of other tasks. We recommend further studies on the effect of nodes that are neighbor's any target on the same characters and properties of features.

Acknowledgements

The authors would like to thank the academic editors and reviewers for their comments and suggestions on the manuscript. This work was financially supported by Office of the Permanent Secretary, Ministry of Higher Education, Science, Research and Innovation (Grant No. RGNS 63-174).

References:

- [1]. Arinik, N., Figueiredo, R., & Labatut, V. (2018). Signed graph analysis for the interpretation of voting behavior. *arXiv preprint*, arXiv:1712.10157.
- [2]. Atwood, J. & Towsley, D. (2016). *Diffusion convolutional neural networks*. Advances in Neural Information Processing Systems, 29, 1993–2001.
- [3]. Bagheri, M., Valipour, M., & Amin, V. (2012). The bankruptcy prediction in tehran share holding using neural network and its comparison with logistic regression. *Journal of Mathematics and Computer Science*, 5, 219–228.
- [4]. Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., & Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks, *arXiv preprint*, arXiv:1806.01261.
- [5]. Derr, T., Ma, Y., & Tang, J. (2018). Signed graph convolutional network. *arXiv preprint*, arXiv:1808.06354.
- [6]. Ghezelbash, A. (2012). Predicting changes in stock index and gold prices to neural network approach. *The Journal of mathematics and computer science*, 4(2), 227-236.
- [7]. Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, 1263-1272. PMLR.
- [8]. Gori, M., Monfardini, G., & Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005*, 2, 729-734. IEEE.
- [9]. Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.
- [10]. Heider, F. (1946). Attitudes and cognitive organization. *The Journal of psychology*, 21(1), 107-112.
- [11]. Honarmand, M., Sanjari, E., & Badihi, H. (2014). Prediction of saturated vapor pressures using non-linear equations and artificial neural network approach. *Journal of Mathematics and computer science*, 8, 343-358.
- [12]. Huang, J., Shen, H., Hou, L., & Cheng, X. (2019). Signed graph attention networks. In *Artificial Neural Networks and Machine Learning–ICANN 2019: Workshop and Special Sessions: 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17–19, 2019, Proceedings 28*, 566-577. Springer International Publishing.
- [13]. Jiang, J., Dun, C., Huang, T., & Lu, Z. (2018). Graph convolutional reinforcement learning. *arXiv preprint arXiv:1810.09202*.
- [14]. Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.
- [15]. Kumar, S., Spezzano, F., Subrahmanian, V. S., & Faloutsos, C. (2016). Edge weight prediction in weighted signed networks. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 221-230. IEEE.
- [16]. Nie, M., Chen, D., & Wang, D. (2023). Reinforcement learning on graphs: A survey. *IEEE Transactions on Emerging Topics in Computational Intelligence*.
- [17]. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE transactions on neural networks*, 20(1), 61-80.
- [18]. Sotoudeh, M., & Farshad, E. (2012). Application of neural network for forecasting gas price in America. *J Math Comp Sci*, 4(2), 216-226.
- [19]. Tang, J., Lou, T., & Kleinberg, J. (2012). Inferring social ties across heterogeneous networks. In *Proceedings of the fifth ACM international conference on Web search and data mining*, 743-752.
- [20]. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.
- [21]. Zhang, K., Pan, L., & Liu, S. (2023). A rating prediction model with cross projection and evolving GCN for bitcoin trading network. *Personal and Ubiquitous Computing*, 27(4), 1561-1571.