Embedding Global and Local Influences for Dynamic Graphs

Meng Liu
2191438@s.hlju.edu.cn
Heilongjiang University & National University of Defense Technology
Harbin, China

Jiaming Wu
2191439@s.hlju.edu.cn
Heilongjiang University
Harbin, China

Yong Liu*
liuyong123456@hlju.edu.cn
Heilongjiang University
Harbin, China

ABSTRACT

Graph embedding is becoming increasingly popular due to its ability of representing large-scale graph data by mapping nodes to low-dimensional space. Current research usually focuses on transductive learning, which aims to generates fixed node embeddings by training the whole graph. However, dynamic graph changes constantly with new node additions and interactions. Unlike transductive learning, inductive learning attempts to dynamically generate node embeddings over time even for unseen nodes, which is more suitable for real-world applications. Therefore, we propose an inductive dynamic graph embedding method called AGLI by aggregating global and local influences. We propose an aggregator function that integrates global influence with local influence to generate node embeddings at any time. We conduct extensive experiments on several real-world datasets and compare AGLI with several state-of-the-art baseline methods on various tasks. The experimental results show that AGLI achieves better performance than the state-of-the-art baseline methods.

CCS CONCEPTS

• **Information systems** → **Data mining**; • **Computing methodologies** → **Machine learning**; **Artificial intelligence**.

KEYWORDS

Dynamic Graph Embedding, Inductive Learning, Global and Local Inlfuences

ACM Reference Format:
Meng Liu, Jiaming Wu, and Yong Liu. 2022. Embedding Global and Local Influences for Dynamic Graphs. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3511808.3557594

1 INTRODUCTION

In the real world, graph data is ubiquitous such as communication graphs, citation graphs, etc. In the fields of machine learning and data mining, learning from graph data has gained much attention [12, 13]. As a popular graph learning method, graph embedding (GE), aims to represent a graph by mapping nodes to a low-dimensional space [2, 15]. The generated node embeddings can be used for downstream tasks [1, 16, 17, 29, 30] like node classification/clustering, community detection, and link prediction, etc.

Related work. Based on the different datasets, we can divide GE methods into static graph learning and dynamic graph learning. A ***static graph*** is a graph where neither topological structure nor node attributes change over time. In the early stages of NE, researchers usually focus on the topological structure of graphs. For example, DeepWalk performs a random walk procedure over the network and then employs the Skip-Gram [18] model to learn node embeddings [22]. node2vec proposes a biased random walk procedure to balance the breadth-first and depth-first search strategy [4]. GraphSAGE learns a function to generate embeddings by sampling and aggregating features from a node's local neighborhood [5].

Unlike the static graph, a ***dynamic graph*** contains a graph's dynamic changes. Dynamic changes in graph can help researchers learn the evolution of a graph structure and obtain more effective embeddings. For example, HTNE uses the Hawkes process to capture historical neighbors' influence on the current node to obtain node embeddings [32]. DyREP proposes a two-time scale deep dynamic point process model, which captures the interleaved dynamics of the observed processes [26]. EvolveGCN captures the dynamic change of graph sequence through using an RNN to evolve the GCN parameters [21].

In addition, current research usually focuses on ***transductive learning***, which generates node embeddings once by training the whole graph [23]. However, graphs change frequently, with new nodes being added and new interactions happening constantly. In this case, transductive learning will have to retrain the whole graph to obtain new node embeddings, which is not feasible for real-world datasets. Unlike transductive learning, ***inductive learning*** [14, 26] no longer focuses on the graph's final node embeddings but attempts to learn a model that can dynamically generate node embeddings over time even for unseen nodes.

A few works have started to focus on inductive learning, but they tend to capture the changes in node structure over time. In other words, the evolution of the graph environment over time has been neglected in dynamic graph.

Our contributions. In this paper, we propose an inductive dynamic graph embedding method called AGLI to learn node embeddings. AGLI can effectively capture graph changes to obtain node embeddings at any time, by **a**ggregating **g**lobal and **l**ocal **i**nfluences in dynamic graphs.

We conduct experiments in several real-world datasets and compare AGLI with several SOTA methods. The results show that AGLI

*Corresponding Author.

CIKM '22, October 17–21, 2022, Atlanta, GA, USA

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557594

can achieve better performance than baselines on various tasks. We summarize our main contributions as follows:

(1) We propose an inductive dynamic graph embedding method AGLI. We model the local neighborhood influence with several perspectives, and propose a global graph influence based on the idea of the Linear Threshold model.

(2) We propose a new aggregator function to combine global and local influences inspired by simulated annealing algorithm.

(3) We empirically evaluate AGLI for multiple tasks on several real-world datasets and show superior performance.

2 METHOD

In this part, we introduce our method AGLI. First, we give the problem definition to explain the background, and then discuss the framework and details of AGLI.

2.1 Problem Definition

According to the interaction between nodes, we can formally define the dynamic graph as follows.

Definition 1: Dynamic Graph. *When an interaction is established between two nodes in the graph, the interaction will be accompanied by a clear timestamp. The dynamic graph can be defined as $G = (V, E, T)$, where V is the set of nodes, $E \subseteq V \times V$ is the set of edges, and T is the set of interaction timestamps. For each edge $e = (u, v)$ between node u and v, there is at least one interaction matching e, i.e., $T_{u,v} = \{(u, v, t_1), (u, v, t_2), \cdots, (u, v, t_n)\}$.*

In a dynamic graph, a node will interact with other nodes multiple times, and these interactions can be ordered by timestamp. When two nodes interact, we call them neighbors. The historical neighbor sequence of a node can be defined as follows.

Definition 2: Historical Neighbor Sequence. *Given a node u, we can obtain its historical neighbor sequence H_u, i.e., $H_u = \{(v_1, t_1), (v_2, t_2), \cdots, (v_n, t_n)\}$. Each tuple in the sequence represents an event, i.e., node v_i interacts with u at time t_i.*

Given a dynamic graph defined above, our goal is to learn an aggregator function that can capture both global and local influences to update node embeddings inductively.

2.2 Overall Framework

Here we present the framework of our method AGLI, which can be divided into four parts, i.e., global influence, local influence, aggregator, and loss function.

For ***global graph influence***, we draw the idea of the Linear Threshold (LT) model [3] in the information propagation field [11, 25], and generate a dynamic embedding for the global graph. For ***local neighborhood influence***, we model the local neighborhood influence from several perspectives: affinity, temporal information, and self-correlation mechanism.

Afterwards, we devise a new ***aggregator function*** to obtain node embeddings inspired by simulated annealing algorithm [24]. Note that the generation of a node embedding requires only a combination of its previous embedding and current influences. It means that giving a new node does not require retraining of the whole dataset, thus inductive learning is achieved.

In the following, we will discuss the details of above parts. Note that we first introduce local influence as it is useful for understanding global influence[1].

2.3 Local Neighborhood Influence

In a dynamic graph, after an interaction occurs between node u and v, node v will influence the future interactions of node u, and vice versa. We will analyze the local influence from several perspectives: affinity, temporal information, and self-correlation mechanism, which work together to form the local influence.

Affinity. We argue that there is an affinity between any two nodes, which reflects the closeness of their relationship. Given a node u and its neighbor node i, we can calculate their affinity $a_{u,i}$ through their node embeddings z_u and z_i as follows.

$$a_{u,i} = \sigma(z_u \odot z_i) \tag{1}$$

Here $\odot$ denotes element-wise multiplication, σ is the variant of sigmoid function to normalize its value to between [0, 1].

Temporal Information. As shown in [7], the Hawkes process is used to model discrete sequential events by assuming historical events will influence the occurrence of future events. According to the process, a node's historical neighbors will influence its future interactions. The closer the interaction time, the greater is the influence on future interactions. Therefore, we can calculate the time weight $k_{u,i}$ of neighbor i on node u as follows.

$$k_{u,i} = \delta_u^t \cdot e^{-|t_c - t_i|} \tag{2}$$

Here t_i is the timestamp when neighbor i interacts with u, t_c is the current time, δ_u^t is a learnable weight parameter that regulates the neighbor nodes' time weight on u.

Self-Correlation Mechanism. After exploring the neighborhood information, we further introduce a self-correlation mechanism [27] to capture the neighborhood sequence's internal relationship. In particular, let L be the length of the historical neighbor sequence, given a node u's neighbor embedding matrix $Z_u \in \mathbb{R}^{L \times d}$, it's normalized self-correlation weight matrix $S \in \mathbb{R}^{L \times L}$ can be calculated as follows.

$$S_{i,j} = \frac{e^{(Z_u Z_u^{\mathrm{T}})_{ij}}}{\sum_{k \in H_u} e^{(Z_u Z_u^{\mathrm{T}})_{ik}}} \tag{3}$$

$$Z_{u'} = S Z_u \tag{4}$$

With S as coefficients, the node u's neighbor embedding matrix Z_u can be updated as above, and the updated neighbor embedding $z_{i'} \in Z'_u$ can be used in the calculation of local influence.

Local Influence Embedding. Combining above two weights, the local neighborhood influence embedding $l_u^{t_n}$ of u at time t_n can be obtained as follows.

$$l_u^{t_n} = \sum_{i \in H_u} \omega_{u,i} \cdot z_{i'}^{t_{n-1}} \tag{5}$$

$$\omega_{u,i} = \frac{a_{u,i} \cdot k_{u,i}}{\sum_{i' \in H_u} a_{u,i'} \cdot k_{u,i'}} \tag{6}$$

[1]The local and global influences of a node are calculated every time it interacts, so we omit the time superscript for each variable by default.

Here $z_i^{t_{n-1}}$ is the embedding of u's neighbor i at time t_{n-1}. To calculate the influence of the next timestamp, we need to use the node embedding of the previous timestamp.

2.4 Global Graph Influence

Activation Threshold. The global graph environment will influence the interaction between nodes. When a node u joins a graph, it is not sensitive to changes in the global environment. Only when the cumulative affinity of u exceeds a certain threshold, u will become sensitive to global graph changes and can easily capture the latest changes on the graph. In this case, we call u an active node.

For the definitions of the node's active status and cumulative affinity, we draw the idea of the Linear Threshold (LT) model [3] in the information propagation field [11, 25]. The LT model's basic idea is that a node can switch its status from inactive to active if its neighbors' total influence on the node reaches a certain value.

Let ϵ be an activation threshold for all nodes, which is a hyperparameter. Let ϵ_u be the cumulative affinity of node u, which means the sum of the affinity between u and u's neighbors. The cumulative affinity ϵ_u of node u can be calculated as follows.

$$\epsilon_u = \sum_{i \in H_u} \sigma(z_u \odot z_i) = \sum_{i \in H_u} \frac{sigmoid(z_u \odot z_i) + 1}{2} \tag{7}$$

At any time, if ϵ_u is larger than ϵ, i.e., the cumulative affinity of u exceeds the activation threshold, u will enter an active status. We believe that the cumulative affinity of u with u's neighbors can reflect the relationship between u and the graph to a certain extent. The higher the cumulative affinity of u with its neighbors, the more easily the global graph will influence u.

Also, the active status is irreversible, i.e., when node u starts to be influenced by the global graph, the global influence on u will always exist. For example, once a scholar has a good understanding of a field, he will always be sensitive to this field's latest ideas. Therefore, after node u enters an active status, we should calculate the global graph influence embedding $g_u^{t_n}$ of node u at time t_n.

Whole Graph Embedding. Here we discuss the whole graph embedding which will exert the global influence on each node. First, we generate a graph embedding z_{graph} which aggregates all node embeddings as follows.

$$z_{graph} = \sum_{u \in V} z_u^{t_0} / |V| \tag{8}$$

Here $|V|$ is the number of nodes, and $z_u^{t_0}$ is the initial node embedding of node u at time t_0. In a dynamic graph, the graph structure and features will evolve over time. Therefore, we need to update the graph embedding over time. Similarly, in the updating process, we also believe that only nodes in the active status can be "noticed" by the whole graph and thus become part of the graph embedding. In this way, only an active node u's embedding is updated from $z_u^{t_{n-1}}$ to $z_u^{t_n}$, the graph embedding z_{graph} will be updated as follows.

$$z_{graph} := (|V| \times z_{graph} - z_u^{t_{n-1}} + z_u^{t_n}) / |V| \tag{9}$$

Global Influence Embedding. We assume that a node's global influence is not only related to the graph embedding, but also to its dynamics. We first divide a node u's dynamics into c_u^- and c_u^+ based on its activation time t_a. We define c_u^- as the embedding change per unit time **before** u is activated. c_u^+ is the embedding change per unit time **after** u is activated. c_u^- and c_u^+ are calculated based on the embedding difference and the time interval as follows.

$$c_u^- = \frac{z_u^{t_a} - z_u^{t_0}}{t_a - t_0}, \qquad c_u^+ = \frac{z_u^{t_{n-1}} - z_u^{t_a}}{t_{n-1} - t_a} \tag{10}$$

Here $z_u^{t_a}$ is the embedding when node u enters an active status, $z_u^{t_0}$ is u's initial embedding, and $z_u^{t_{n-1}}$ is the embedding of u at time t_{n-1}. The global influence of the graph on a node is exerted in a proportion that depends on how relevant the node is to the graph after its active status. Combining Eq. (9) and (10), we finally calculate the global graph influence embedding $g_u^{t_n}$ of node u at time t_n as follows.

$$g_u^{t_n} = (\frac{1}{d} \sum_{i=1}^{d} \frac{c_{u,i}^+ - c_{u,i}^-}{c_{u,i}^+}) \cdot \delta_u^g \cdot z_{graph} \tag{11}$$

Here $c_{u,i}^+$ is the component of c_u^+ in the i^{th} dimension, $c_{u,i}^-$ is the component of c_u^- in the i^{th} dimension, and d is the dimension size.

The first part $(\frac{1}{d} \sum_{i=1}^{d} \frac{c_{u,i}^+ - c_{u,i}^-}{c_{u,i}^+})$ in Eq. (11) is used to measure the degree to which a node is influenced by the graph. In other words, it determines in what proportion the graph embedding is incorporated into a node embedding.

The difference of embedding change degree before and after activation can reasonably reflect the role of global graph influence. Thus, by calculating the difference between c_u^+ and c_u^-, we can obtain the degree of embedding change influenced by the graph. The larger the percentage of this difference in c_u^+, the deeper node u is influenced by the global graph. It is worth noting that this difference is presented in the form of a vector, i.e., there is a percentage in each dimension of the vector. Thus, we average this percentage in all dimensions to obtain the final percentage.

The second part δ_u^g in Eq. (11) is a learnable weight parameter that regulates u's global graph influence embedding. By multipling the weights of the first two parts by the graph embedding z_{graph}, we can finally calculate the global graph influence.

2.5 Aggregator Function

In this part, we design an annealing aggregator function to combine global and local influences with node embeddings. Note that the goal of inductive learning is achieved by aggregator functions, which will be described in detail below.

Considering the global influence is not well-updated at the early stage, we expect the effect of global influence embedding g to be smaller at first and gradually grow with the number of trained interactions s. To this end, we introduce simulated annealing algorithm [24] to control the ratio of global influence as follows.

$$z_u^{t_n} = z_u^{t_{n-1}} + \delta_i [\gamma(s) \cdot l_u^{t_n} + (1 - \gamma(s)) \cdot g_u^{t_n}] \tag{12}$$

$$\gamma(s) = -\frac{s}{|E| + s} \tag{13}$$

Here $z_u^{t_n}$ and $z_u^{t_{n-1}}$ are node u's embedding at time t_n and t_{n-1}, $l_u^{t_n}$ and $g_u^{t_n}$ are local influence embedding and global influence embedding at time t_n. δ_i is a learnable parameter that controls the ratio of both local and global influences. $|E|$ denotes the number of interactions, which is the upper bound of s.

Given a new node, instead of re-training the whole graph to generate its node embedding, we only need to calculate its local and global influences. In this way, we can generate node embeddings inductively.

2.6 Loss Function

To learn the effective node embeddings in a fully unsupervised setting [31], we apply a graph-based loss function to node embedding $z_u^{t_n}$, and optimize it with Adam [8] method. The graph-based loss function encourages nearby nodes to have similar embeddings while enforcing that the embeddings of disparate nodes are highly distinct. We use negative squared Euclidean distance to measure the similarity between two embeddings, the loss function can be defined as follows.

$$\begin{aligned}\log L = \sum_{u \in V} \sum_{v \in H_u} \Bigg[\log \sigma \left(- \left\| z_u^{t_n} - z_v^{t_n} \right\|^2 \right) \\ - Q \cdot E_{v_n \sim P_n(v)} \log \sigma \left(- \left\| z_u^{t_n} - z_{v_n}^{t_n} \right\|^2 \right) \Bigg]\end{aligned} \tag{14}$$

Due to the enormous computation cost of the loss function, we use negative sampling [19] to optimize the loss. $P_n(v)$ is a negative sampling distribution, and Q defines the number of negative samples. We sample negative nodes which have not occured in node u's historical neighbor sequence.

3 EXPERIMENTS

3.1 Experimental Setup

3.1.1 Datasets. We conduct experiments on five real-world datasets as shown in Table 1. *DBLP* is a co-authorship graph of computer science [32]. *BITotc* and *BITalpha* are two datasets from two bitcoin trading platforms [9, 10]. *AMms* is taken from the magazine subscription [20]. *Yelp* is a Yelp Challenge dataset [32].

Table 1: Description of the datasets

Datasets	DBLP	BITotc	BITalpha	AMms	Yelp
Nodes	28,085	5,881	3,783	74,526	424,450
Edges	236,894	35,592	24,186	89,689	2,610,143
Labels	10	7	7	5	5
Timestamps	25	22,115	981	5,082	70

3.1.2 Baselines. We compare AGLI with five state-of-the-art baseline methods, i.e., Deepwalk, node2vec, GraphSAGE, HTNE, DyREP, and EvoloveGCN. These methods are described in Section 1.

3.1.3 Parameter Settings. For all methods, we set the embedding dimension size d, the learning rate, the batch size, the number of negative samples Q, and the activation threshold ϵ to be 128, 0.001, 128, 10, and 1, respectively. Also, we use default values for other parameters in baselines.

3.2 Results

Here we conduct link prediction experiments on six datasets and take the Area Under the ROC Curve (AUC) [6] to measure the prediction performance.

First, we sort all interactions in a dataset in order of interaction time. We select the top 80% of each dataset as our training set, and the rest 20% as the test set. If the same timestamp interactions are assigned to both training set and test set, we assign all interactions at this timestamp to the training set. We obtain all node embeddings by applying AGLI and baselines in training set. In the test set, we sample a certain number of node pairs connected by edges as positive samples and sample the same number of node pairs without edge as negative samples.

For AUC, we calculate the dot product of their embeddings for each pair of nodes and use the sigmoid function to normalize the dot product as the interaction probability. Then we sort all interaction probabilities in descending order and assume that there are edges between each node pair of the top-half. By comparing the truth on node pairs, we can obtain the AUC score.

As shown in Table 2, it can be seen that AGLI has the best performance on all datasets, which demonstrates the ability of AGLI to capture interactive information in the graph. It also shows that for prediction tasks, a method such as HTNE that exploits the interaction time are more effective than a method such as Deepwalk that only focuses on graph structure.

Table 2: Link prediction results of all methods on all datasets

Metric	AUC				
Method	DBLP	BITotc	BITalpha	AMms	Yelp
DeepWalk	0.8253	0.5199	0.5558	0.5483	0.7740
node2vec	0.8173	0.5799	0.6245	0.5228	0.8426
GraphSAGE	0.8452	0.5967	0.6964	0.5554	0.8553
HTNE	0.8868	0.7145	0.7401	0.5741	0.8821
DyREP	0.8763	0.7112	0.7342	0.5807	0.8664
EvolveGCN	0.8554	0.7179	0.7164	0.5143	0.7953
AGLI	**0.8954**	**0.7359**	**0.7451**	**0.5934**	**0.8929**

4 CONCLUSIONS

In this paper, we propose an inductive graph embedding method AGLI that captures both global and local influences to generate node embeddings at any time. Extensive experiments on several real-world datasets demonstrate that AGLI significantly outperforms state-of-the-art baseline methods. In the future, we will study the influence of attribute learning [28] or contrastive learning [31] on node embeddings.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (No. 61972135), and the Natural Science Foundation of Heilongjiang Province in China (No. LH2020F043).

REFERENCES

[1] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral Networks and Locally Connected Networks on Graphs. *ICLR* (2014).
[2] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2019. A Survey on Network Embedding. *TKDE* (2019).
[3] Mark Granovetter. 1978. Threshold Models of Collective Behavior. *Amer. J. Sociology* (1978).

[4] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. *KDD*, 855–864.

[5] L. William Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. *NeurIPS*, 1024–1034.

[6] A. James Hanley and J. Barbara McNeil. 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* (1982), 29–36.

[7] G. A. Hawkes. 1971. Point spectra of some mutually exciting point processes. (1971).

[8] P. Diederik Kingma and Lei Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *ICLR* (2015).

[9] Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian. 2018. Rev2: Fraudulent user prediction in rating platforms. In *WSDM*. 333–341.

[10] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. 2016. Edge weight prediction in weighted signed networks. In *ICDM*. 221–230.

[11] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. 2018. Influence Maximization on Social Graphs: A Survey. *TKDE* (2018), 1852–1872.

[12] Ke Liang, Sifan Wu, and Jiayi Gu. 2021. MKA: A Scalable Medical Knowledge-Assisted Mechanism for Generative Models on Medical Conversation Tasks. (2021).

[13] Weixuan Liang, Xinwang Liu, Sihang Zhou, Jiyuan Liu, Siwei Wang, and En Zhu. 2022. Robust Graph-based Multi-view Clustering. In *AAAI*.

[14] Meng Liu and Yong Liu. 2021. Inductive representation learning in temporal networks via mining neighborhood and community influences. In *SIGIR*.

[15] Meng Liu, Zi-Wei Quan, Jia-Ming Wu, Yong Liu, and Meng Han. 2022. Embedding temporal networks inductively via mining neighborhood and community influences. *Applied Intelligence* (2022).

[16] Yue Liu, Wenxuan Tu, Sihang Zhou, Xinwang Liu, Linxuan Song, Xihong Yang, and En Zhu. 2022. Deep Graph Clustering via Dual Correlation Reduction. In *AAAI*.

[17] Yue Liu, Xihong Yang, Sihang Zhou, and Xinwang Liu. 2022. Simple Contrastive Graph Clustering. *arXiv:2205.07865* (2022).

[18] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* (2013).

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *NeurIPS* (2013), 3111–3119.

[20] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Fined-Grained Aspects. *EMNLP* (2019), 188–197.

[21] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and E. Charles Leisersen. 2020. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. *AAAI* (2020).

[22] Bryan Perozzi, Rami Al-Rfou', and Steven Skiena. 2014. DeepWalk: online learning of social representations. *KDD* (2014), 701–710.

[23] Balasubramaniam Srinivasan and Bruno Ribeiro. 2020. On the Equivalence between Node Embeddings and Structural Graph Representations. *ICLR* (2020).

[24] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. 1997. Heuristic and randomized optimization for the join ordering problem. In *The VLDB Journal*.

[25] Tang Jia Tian, WANG Yi-Tong, and FENG Xiao-Jun. 2011. A new hybrid algorithm for influence maximization in social networks. *Chinese Journal of Computers* (2011), 1956–1965.

[26] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep - Learning Representations over Dynamic Graphs. *ICLR* (2019).

[27] Wenxuan Tu, Sihang Zhou, Xinwang Liu, Xifeng Guo, Zhiping Cai, En Zhu, and Jieren Cheng. 2021. Deep fusion clustering network. In *AAAI*.

[28] Wenxuan Tu, Sihang Zhou, Xinwang Liu, Yue Liu, Zhiping Cai, En Zhu, Zhang Changwang, and Jieren Cheng. 2022. Initializing Then Refining: A Simple Graph Attribute Imputation Network. In *IJCAI*.

[29] Siwei Wang, Xinwang Liu, Li Liu, Wenxuan Tu, Xinzhong Zhu, Jiyuan Liu, Sihang Zhou, and En Zhu. 2022. Highly-efficient Incomplete Large-scale Multi-view Clustering with Consensus Bipartite Graph. In *CVPR*.

[30] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community Preserving Network Embedding. *AAAI* (2017), 203–209.

[31] Xihong Yang, Xiaochang Hu, Sihang Zhou, Xinwang Liu, and En Zhu. 2022. Interpolation-Based Contrastive Learning for Few-Label Semi-Supervised Learning. *TNNLS* (2022).

[32] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. 2018. Embedding Temporal Network via Neighborhood Formation. *KDD*.

APPENDIX

In the appendix part, we analyze the complexity of AGLI. The procedure is shown in Algo. 1.

Algorithm 1 AGLI procedure

Require: Dynamic Graph $G = (V,\ E,\ T)$.
Ensure: Node embeddings; Graph embedding.
1: Initialize embeddings for each node randomly;
2: Initialize graph embedding based on Eq. (8);
3: **repeat**
4: **for** each *batch* in network **do**
5: **for** each node u in *batch* **do**
6: Calculate $l_u^{t_n}$ based on Eq. (5);
7: Calculate $g_u^{t_n}$ based on Eq. (11);
8: Calculate $z_u^{t_n}$ based on Eq. (12);
9: **end for**
10: Update graph embedding based on Eq. (9);
11: Optimize the loss function based on Eq.(14);
12: **end for**
13: **until** Convergence

Suppose that the number of nodes and edges in the graph are $|V|$ and $|E|$, respectively. Let t be the number of epochs, s be the number of batches ($s = |E|/|B|$), $|B|$ be the size of each batch, d be the embedding size, L be the length of the historical neighbor sequence, and Q be the number of negative sample nodes.

According to Algo. 1, we can divide AGLI into three parts: Initialization, Batch Training, Updating and Optimization. Thus the time complexity can be calculated as follows.

(1) Initialization (lines 1-2). In this part, we initialize node embeddings and global embedding, the time complexity of this part is $O(d|V| + d) = O(d|V|)$.

(2) Batch Training (lines 5-9). According to Eq. (1)-(6), Eq. (7)-(11), and Eq. (12)-(13), we discuss the complexity of calculating $l_u^{t_n}$, $g_u^{t_n}$, and $z_u^{t_n}$, respectively. The time complexity of traing one batch data is $O(|B|(L^2d^2 + Ld^2 + d)) = O(|B|L^2d^2)$.

(3) Updating and Optimization (line 9-10). According to Eq. (9) and (14), the time complexity is $O(|B|d + |B|LQd) = O(|B|LQd)$.

In summary, considering the number of epochs t and the number of batches s, the total time complexity of AGLI can be calculated as $O(d|V| + ts(|B|L^2d^2 + |B|LQd))$. Considering that L and Q are small constants, the time complexity of AGLI can be simplified as $O(d|V| + ts|B|d^2)$.