International Journal of Software and Informatics, ISSN 1673-7288
http://www.ijsi.org, ijsi@iscas.ac.cn, +86-10-62661048
IJSI, 2021, 11(1): 5–29, doi: 10.21655/ijsi.1673-7288.00242

Research Article

Node Embedding Research Over Temporal Graph

Anbiao Wu (吴安彪) [1], Ye Yuan (袁野) [2], Yuliang Ma (马玉亮) [3], Guoren Wang (王国仁) [2]

[1] (School of Computer Science and Engineering, Northeastern University, Shenyang 110169, China)

[2] (School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)

[3](School of Business Administration, Northeastern University, Shenyang 110169, China)

Corresponding author: Ye Yuan, yuanye@mail.neu.edu.cn

Abstract Compared with conventional graph data analysis methods, the graph embedding algorithm provides a new graph data analysis strategy. It aims to encode graph nodes into vectors to mine or analyze graph data more effectively using neural network related technologies. Some classic tasks have been improved significantly by graph embedding methods, such as node classification, link prediction, and traffic flow prediction. Although substantial breakthroughs have been made by former researchers in graph embedding, the nodes embedding problem over temporal graph has been seldom studied. In this study, we propose an adaptive temporal graph embedding (ATGED), attempting to encode temporal graph nodes into vectors by combining previous research and the information propagation characteristics. First, an adaptive cluster method is proposed by solving the situation that nodes active frequency varies types of graph. Then, a new node walk strategy is designed in order to store the time sequence between nodes, and also the walking list will be stored in a bidirectional multi-tree in the walking process to get complete walking lists fast. Last, based on the basic walking characteristics and graph topology, an important node sampling strategy is proposed to train the satisfied neural network as soon as possible. Sufficient experiments demonstrate that the proposed method surpasses existing embedding methods in terms of node clustering, reachability prediction, and node classification in temporal graphs.

Keywords temporal graph; node embedding; importance sampling; temporal reachability; node classification

Citation Wu AB, Yuan Y, Ma YL, Wang GR. Node embedding research over temporal graph. International Journal of Software and Informatics, 2021, 11(1): 5–29. http://www.ijsi. org/1673-7288/00242.htm

With the ultra-high speed computing power of computers, models for high-dimensional data and multi-level neural networks have been designed. In the research on the graph data in deep

This is the English version of the Chinese article “时序图节点嵌入策略的研究.软件学报,2021,32(3):650-668. doi: 10.13328/j.cnki.jos.006173”.

Foundation items: National Natural Science Foundation of China (61932004, 62002054, 61732003, 61729201); Fundamental Research Funds for the Central Universities of Ministry of Education of China (N181605012); China Postdoctoral Science Foundation Funded Project (2020M670780)

Received 2020-07-19; Revised 2020-09-03; Accepted 2020-11-06; IJSI published online 2021-03-31

learning, scholars did not find an effective network model for graph data at the early stage of neural network research as they are non-Euclidean data. Therefore, it is challenging to encode high-dimensional graph data to low-dimensional and structural vectors and to train parameters for acquired node vectors by neural networks.

In 2009, Scarselli et al. [1] firstly proposed the concept of graph neural network model (GNN), and they encoded nodes to vectors by incorporating attributes of neighboring nodes. They provided a prototype of representation learning on graphs, and guided some researchers in the research on feature-based representation learning on graphs. However, it did not attract much attention at that time. Until 2014, the proposition of the DeepWalk [2] algorithm really triggered the research boom on graph (node) embedding. Inspired by the word2vec [3] algorithm in natural language processing, the DeepWalk algorithm designed a very simple yet effective approach for encoding nodes to vectors. Specifically, the DeepWalk algorithm calculated the similarity between nodes and node vectors by training the random walking list of each node (ID list of the node) through the *skip-gram* [3] model. Later, researchers have proposed node embedding algorithms such as large-scale information network embedding (LINE) [4], predictive text embedding (PTE) [5], node2vec [6], and stru2vec [7] based on the DeepWalk algorithm. However, all of these algorithms still follow the original framework of the DeepWalk algorithm. Essentially, compared with the DeepWalk algorithm, these algorithms focus more on biased node walk in terms of the node walk strategy. In other words, they define the walking probability of a node with its focus on the node topology. In 2018, Dong et al. [8] provided the matrices of the four algorithms, DeepWalk, LINE, PTE, and node2vec, which further illustrated the unified principle of these algorithms.

Such walk strategy-based representation learning on graphs can preserve the node topology to some extent, especially the methods based on biased walk strategies such as node2vec, and has achieved satisfactory results in experiments. However, the walk strategy-based algorithms have a very obvious deficiency that they completely ignore the role of attributes between nodes in experimental results. This makes them unable to achieve satisfactory results on the data sets of some attributed graphs. In 2017, Hamilton et al. [9] designed a novel sampling strategy, called GraphSAGE (SAmple and aggreGatE). First, the node v performs sampling for its neighboring node v_i (the i-th neighbor sampled by the node v). Then, the sampled node v_i performs sampling for the node v_{ij} in the next layer (the j-th neighbor sampled by the node v_i). At last, this strategy aggregates all the features of the sampled nodes from outside to inside, obtaining a new aggregated eigenvector of the node v. Although the GraphSAGE algorithm considers eigenvectors of nodes, it weakens the preserve for node topology. In the same year, Kipf [10] proposed a semi-supervised Graph Convolutional Network (GCN). Different from GraphSAGE, Ref. [10] convolves the features of all the neighboring nodes for a given node by weight sharing, which was fundamentally a weighted summation. It can be seen from $H^{(l+1)}=\sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)})$ that scholars conducted Laplace transformation for the matrix $\tilde{A}$, which is obtained by adding the identity matrix I_N and the adjacency matrix A, the node eigenvector H, and the matrix $\tilde{D}=\sum_j \tilde{A}_{i,j}$, and then obtained the new node vector by the trainable weight parameter W. In addition, it can be known that the scholars conducted a multi-layer convolution. Meanwhile, the experimental results in Ref. [10] showed that as the number of layers increased, more precise results could be obtained. However, the quality of the experimental results would drop sharply once the number of layers exceeded a certain value. This is because once too much information is aggregated by a single node, the vectors between nodes cannot have certain variabilities, and this will result in model generalization. In Ref. [10], the convolution on neighboring nodes was uniformly performed on the premise of weight sharing.

However, in practice, the influences of some specific neighboring nodes on the attribute of a given node are bigger than those of other nodes, so these neighboring nodes should be assigned higher weights. On this basis, Velickovic et al. [11] proposed the Graph ATtention networks (GAT) in 2018. Specifically, with the shared weight parameter W and the node eigenvector $\vec{h}_i$, the weight coefficient between two nodes v_i and v_j is calculated by two functions LeakyReLU and Softmax:

$$\alpha_{i,j}=\frac{\exp(LeakyReLU(\vec{a}^T[W\vec{h}_i \,||\, W\vec{h}_j]))}{\sum_{k\in N_i}\exp(LeakyReLU(\vec{a}^T[W\vec{h}_i \,||\, W\vec{h}_j]))}$$

As the eigenvectors of neighboring nodes are different, their weight coefficients should be variable. In addition, the shared weight parameter W changes during the learning process, so the weight coefficient between nodes also changes in the training process.

Temporal graphs (also known as temporal networks [12,13] and time-varying graphs [14,15]) are time-based dynamic graphs with time labels on node edges. Data analysis on temporal graphs has important applications in bioinformatic networks, online social networks, and road traffic networks. In bioinformatic networks, the connections of biological functions are not always active[16]. For example, in protein-protein interaction networks [17] and gene-regulation networks [18], the connections of biological structures are sequential. The functions of structures can be more easily confirmed through the analysis of interactions of these structures in different periods. In road traffic networks [19–21], scholars could make route recommendations or reachability queries for users by combining the historical data of the networks. In social networks [22–24], scholars could characterize the relationships between users more precisely by recording their specific interactions.

Existing research on node embedding focuses more on how to better preserve structural attributes of nodes in the vector representation of nodes. However, as connections between nodes in temporal graphs are time-varying, which indicates the propagation sequence of specific information between nodes, the node topology is dynamically changing over time. However, this situation is not fully considered in graph embedding strategies.

In network graphs, temporality is not only limited to the temporality between two connected nodes. For example, the connection between the vertex 1 and the vertex 2 in Figure 1(a) exists only at the moments t_1 and t_2. When the static edge (1, 3) is not considered, there is a reachable path $1 \rightarrow 2 \rightarrow 3$ between the vertex 1 and the vertex 2 only from the perspective of topology. However, once the temporal factor is considered, if the time stamp on the static edge (2, 3) is t_3 and $t_3 > t_2 > t_1$, then there is no reachable path between the vertex 1 and the vertex 2. In addition to connection temporality and path temporality, the attributes of nodes sometimes can be temporal as well. With the vertex 1 in Figure 1(b) as an example, the attribute of a node may vary with time. This phenomenon is prominent in e-commerce networks, and is a very difficult challenge to be solved in recommendation systems.

According to the temporal property of temporal graphs compared with static graphs, the challenges of graph neural network models on temporal graphs can be summarized as follows.

Node reachability: If two vertices v_i and v_j have only one reachable path in topology, but there is no connection possibility in practice, it is not appropriate for the node v_i to incorporate the information of the vertex v_j in sampling.

(1) Multi-edges: Two vertices have multiple time-dependent edges at different moments in a time span. For example, there are two edges between the vertex a and the vertex b in Figure 1(c) at the moment 3 and the moment 5, respectively. Then during node walk or information integration, it is necessary to consider which edge is more appropriate.

(2) Path selection: The selection of time-dependent paths will directly affect the walking length of the starting point or the information amount of aggregated nodes. With the vertex b and the vertex d in Figure 1(c) as examples, if the path between the vertex b and the vertex c at the moment 6 is selected, the vertex b will not reach the vertex d; but if the path at the moment 2 is selected, then the vertex d is reachable. Thus, the most accurate information can be incorporated only if a most correct path is selected.

(3) Time spans of paths: During vertex walk or information integration along a path, if the time span of this path is too long, the vertex near the end of the path and the vertex near the starting point should not be assigned the same weight. From the path (⟨a, b, c, e, f⟩) from the vertex a to the vertex f in Figure 1(c), it can be seen that the time span is only 7 between a and e, but when the path reaches the vertex f, the time span increases sharply to 50. At this point, the influence of the vertex f on the vertex a may be minimal. If the information of the vertex f is incorporated by the vertex a, the incorporated information of the vertex a is redundant and may even be wrong. Thus, the influences of the vertices with longer connection time on the starting point should be weakened as time goes on.

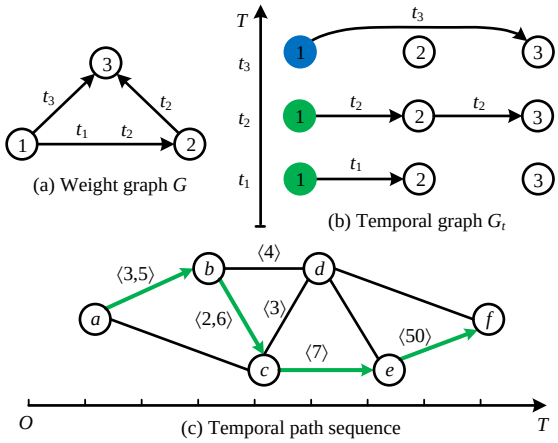


Figure 1. Example of temporal graphs

There are many research fruits in graph embedding. However, when analyzing these experimental results, we find a remarkable and general problem that the graph representation learning methods based on different strategies have variable experimental results on different types of graphs. This is because some graphs are highly sensitive to their topologies, but insensitive to their attributes. For this type of graphs, the node embedding methods based on walk strategies can get better experimental results. The graphs which are sensitive to their attributes are more suitable to convolution-based graph representation strategies. On this basis, it can be known that it is almost impossible to represent all types of graphs by one kind of graph learning models within the existing theoretical framework and at technical level. Therefore, in order to deal with the challenge of representation learning on temporal graphs, we aim to design a graph embedding learning method which is sensitive to temporality, so as to obtain a graph representation learning method that is more suitable for the characteristics of temporal graphs.

The innovations of this paper are as follows.

(1) By integrating the existing graph embedding ideas and related characteristics of temporal graphs, we design a novel embedding strategy for temporal graphs, which can satisfy the analysis of temporal graphs.

(2) In order to solve the problem that the activity between nodes in different types of temporal graphs varies much, this paper designs an adaptive node walk model that satisfied temporal reachability and preserves the temporal property that the connections between a given node and its neighboring nodes varies with time as much as possible.

(3) To obtain the walking list of nodes in different periods as fast as possible, we save the nodes in the walking process in bidirectional and temporal multi-trees. In this way, the walking list can be obtained simply and quickly after the walking was finished.

(4) In terms of the characteristics of the embedding method and the graph topology, we reduce the training time of neutral network models for single nodes by only conducting sampling for important nodes.

(5) Different experiments of temporal graphs are developed on different types of real temporal graphs, so as to verify the generality, accuracy, and efficiency of the proposed method.

Section 1 gives the basic definitions related to temporal graphs and the definition of embedding on temporal graphs. Section 2 introduces a basic and temporal walking method. Section 3 proposes a more efficient walk strategy over temporal graphs. Section 4 performs sampling for important nodes. Section 5 analyzes the experiments on temporal graphs. Section 6 describes related work.

1 Problem Definition

To organize the basic issues, this section will introduce the types of temporal graphs, which are the research objects, and define the basic concepts. The meanings of the symbols used in this paper are listed in Table 1.

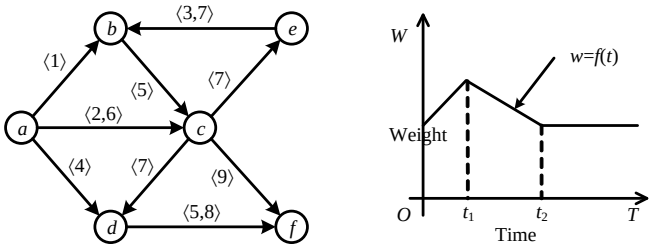
Table 1 List of symbols

Symbol	Meaning	Symbol	Meaning
u, v	Nodes in the graph	$Arr_{(u,v)}$	Time from node u to node v
G_T	Temporal graph network	ul	Walking list of node u
Inf_i	Information type in the network	WL	Set of walking lists of all the nodes
Lab_i	Label of node	Win_u	Window of walking list of node u
N_u	Set of neighboring nodes of node u	z_i	Vector representation of node v_i
$T_{(u,v)}$	Set of connection moments between node u and node v	R^d	Dimensional space of vectors

Generally, the edges of temporal graphs are discrete, as shown in Figure 2(a). The edge of the node u pointing to the node v at the moment t is denoted as (u, v, t, λ), where λ denotes the arrival time, namely that the node u departs at the moment t and arrives at the node v after time λ. This paper does not involve the time-based path query between nodes [25], and focuses more on the moment at which the node u reaches the node v. Thus, λ can be ignored. In practice, t can be treated as $t + \lambda$. In this way, the temporal graph in this paper can be simplified to (u, v, t), where $t = t + \lambda$. With the vertices a and b in Figure 2(a) as examples, if information is sent from a to b at moment 0 and arrives after $\lambda = 1$ time unit, the weight of this edge is 1. However, in social networks, λ is often set as 0 due to the immediacy of information. In another case, λ denotes the duration of the connection, namely that the connection is established between two nodes at the moment t and the connection lasts for λ time units. On the principle of arrival as fast as possible, λ is ignored and the weight of the edge between the two nodes is assigned t. It should be noted that in real datasets, the connection between two nodes in the data representation is often immediate, so λ can be ignored. This type of discrete temporal graphs is the research object of this paper.

Besides, there is a special kind of temporal graphs, which is also called as time-dependent graphs [26,27]. In this type of temporal graphs, the weights of edges are determined by the

time-dependent function $f(t)$, which are not discrete, as shown in Figure 2(b). This paper does not focus on this type of temporal graphs due to their limited applications.



(a) Temporal graph (b) Time-dependent edge weight

Figure 2. Examples of different types of temporal graphs

Definition 1 (Temporal graphs). The given temporal network $G_T(V, E, T_E, X)$ denotes a directed temporal graph with temporal relationships between nodes; V the set of nodes, $V = \{v_1, \ldots, v_n\}$; E the set of edges, and $|V| = n$, $|E| = m$. T_E represents the set of moments when there is connection between nodes in the graph, and $T_{(u,v)}$ is the set of moments when there is a connection between the node u and the node v. As shown in Figure 2(a), $T_{(a,c)} = \{2, 6\}$ and $T_{(u,v)} \in T_E$. X denotes the set of node eigenvectors, and $X = \{x_1, \ldots, x_n\}$, where x_i indicates the eigenvector of the node v_i.

Definition 2 (Arrival time). For the given temporal graph $G_T(V, E, T_E, X)$, the time for the node u reaching the node v is denoted as $Arr_{(u,v)}$, and $Arr_{(u,v)} = T_{(u,v)} + \lambda$.

As $\lambda = 0$ in this paper, with Figure 2(a) as the example, $Arr_{(a,b)} = T_{(a,b)} = \{1\}$, $Arr_{(a,c)} = T_{(a,c)} = \{2, 6\}$.

Definition 3 (Temporally reachable path). For the given temporal graph $G_T(V, E, T_E, X)$, the path $\langle v_1, v_2, \ldots, v_k\rangle$□ satisfies temporal reachability when and only when $\min(Arr_{(v_i,v_{i+1})}) \le \max(Arr_{(v_{i+1},v_{i+2})}) \mid (0 \le i \le k-2)$. The $\max(Arr_{(v_{i+1},v_{i+2})}) < \min(Arr_{(v_i,v_{i+1})})$ indicates all the connections between the node v_{i+1} and the node v_{i+2} are before the point when the node v_i is connected to the node v_{i+1}. In other words, after the node v_i reaches the node v_{i+1}, there are no connections between the two nodes v_{i+1} and v_{i+2}.

With Figure 2(a) as an example, there are three reachable paths between the vertex a and the vertex f: $\langle a, b, c, f\rangle$, $\langle a, c, f\rangle$, and $\langle a, d, f\rangle$. With the path $\langle a, d, f\rangle$ as an example, if the arrival moment from the vertex a to the vertex d changes from 4 to 9, the path $\langle a, d, f\rangle$ is not a temporally reachable path. Because after the moment 9, the two nodes v_{i+1} and v_{i+2} are disconnected.

Definition **4** (Temporal graph representation learning). For the given temporal graph $G_T(V, E, T_E, X)$, the representation learning of temporal graph nodes can be formally indicated as follows. When the sampling nodes are temporally reachable, the node v_i is mapped to a vector with the dimension of d by the learning function f, and $d << |V|$, namely

$$f: V \to Z,\ Z = \{z_1, \ldots, z_n\},\ z_i \in R^d$$

where the vector z_i is the final vector representation of the node v_i.

2 Limitations of Walk Strategy in Temporal Graphs

In light of the above definitions as well as characteristics and application scenarios of temporal graphs, we analyze the limitations and challenges of the node representation problem limited by the characteristics of temporal graphs. The first step is to analyze the difficulties of the problem as much as possible, and then we can arrive at the corresponding solutions.

Limitation 1: Walking lists cannot preserve the temporal factor.

In the node representation learning based on walk strategies, the walking lists of a node should be obtained at first. With the vertices *a* and *o* in Figure 3 as examples, when the temporal relationship between the two vertices is not considered, the vertex *a* can reach the vertex *o* through the two paths ⟨*a*, *f*, *n*, *o*⟩ and ⟨*a*, *e*, *l*, *f*, *n*, *o*⟩, and the two walking lists '*a*, *f*, *n*, *o*' and '*a*, *e*, *l*, *f*, *n*, *o*' can be obtained. The distance between the two vertices *a* and *o* indicates the distance between the two in the topology. In the temporal relationship, as the path ⟨*a*, *e*, *l*, *f*, *n*, *o*⟩ does not satisfy the temporal reachability, the vertex *a* can only reach the vertex *o* through the path ⟨*a*, *f*, *n*, *o*⟩.

In this way, the temporal property between nodes can be better preserved at the expense of some topological properties. This is certainly more friendly to the time-sensitive experimental results between nodes. In addition, the nodes which are only adjacent in the topology are not necessarily "close" to each other. When they do not satisfy the temporal reachability, the two vertices may not have many connections in practice. They just look "close" to each other constrained by the topological relationship.

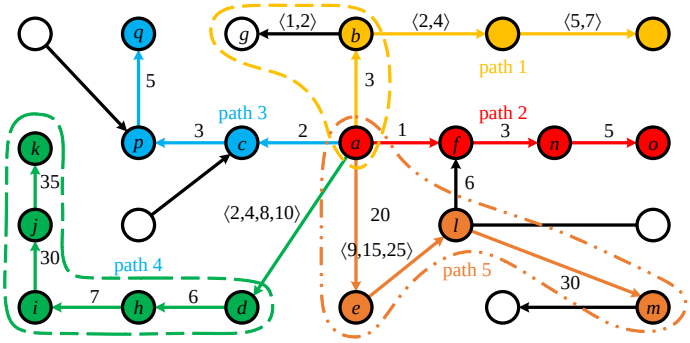


Figure 3 Selection of random walking paths in a temporal graph

Limitation 2: There are dynamic changes in local topological structures.

Sometimes in a short time span, the dynamic changes in the relationship between nodes are limited in a local scope of a graph. From the node v_i that walks at different moments, we can obtain many repeated paths. With the vertices *a* and *d* in Figure 3 as examples, the temporally reachable paths of the vertex *a* walking on the path 4 are the same at moments 2 and 4. When the time span is long, there is a different temporally reachable path at the moment 8. When there is a sudden and large change in the connection moment between nodes, there may be some "upheavals" in the related local topology. At this time, the vertex can walk after upheavals to obtain a new path, and the sampling should be performed as small as possible before upheavals.

This phenomenon is common in practical networks. In a typical road traffic network, for example, the travel time of each road section varies with moments (morning peak, evening peak, and ordinary times), which leads to changes in local connectivity. People may need to select various paths at different moments when they travel from the origin *A* to the destination *B*.

Limitation 3: The dynamic rates of change in different types of temporal networks vary greatly.

The frequency of connections between nodes varies drastically in different types of temporal networks. Some may be measured in milliseconds (such as communication service networks), while some may be measured in minutes, hours, or even days (such as mail networks). In this case,

it is a great challenge to design an adaptive node sampling strategy so that the problem stated in the Limitation 2 can be solved in different types of temporal networks.

Limitation 4: Time span should be considered in sampling paths.

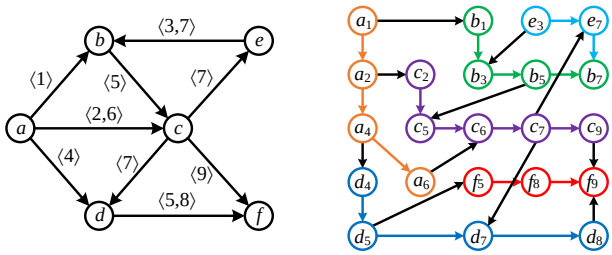
As the time variation has a big influence on the topology and the relationship between nodes in temporal graphs, the influence on a node cast by its neighboring nodes should also be limited to a certain time range. In a practical brain network (other practical networks such as traffic networks or mail networks have similar situations), a message sent from one neuron a_1 to another neuron at the moment t_1 does not spread forever in the network. After passing through many neurons, the message is sent to the neuron a_i at the moment t_i. Then after several moments, a message is sent from the neuron a_i. It may simply be sent from the neuron a_i and is no longer related to the initial neuron a_1. Thus, the temporal path after this moment should be attributed to the neuron a_i instead of a_1. This phenomenon is very common in temporal networks. Especially in social networks, where information is often instantaneous, this phenomenon is more common.

Therefore, when the initial node v_i is sampled on a path that satisfies temporal reachability, the time span of the path increases as the number of nodes on the sampled path increases. Although the node at the end of the path and the initial node v_i satisfy the temporal reachability, it is important to test whether the correlation between them is made indirectly by intermediate nodes and whether the nature of the transfer has changed.

3 Basic Temporal Node Embedding Strategy

According to the embedding method of existing walk strategies and the temporality between nodes, the simplest embedding strategy of temporal graphs is to record the arrival moment of the latest node in the walking list during the node walk and then select the next node that can be walked.

First, the temporal graph G_T should be transformed to a more convenient static graph, as shown in Figure 4. Figure 4(a) and Figure 4(b) show the original temporal graph G_T and the transformed static graph, respectively. In Figure 4(b), nodes with the same color indicate the same node at different moments. For example, the vertex a has connections with the vertices b, c, and d at the moments 1, 2, 4, and 6 in Figure 4(a). In Figure 4(b), there are four vertices with the same color, including a_1, a_2, a_4, and a_6. The earlier vertices point to the later vertices in the chronological order, as shown by the edges with the same color and vertices. The subscripts of the vertices indicate the moment when the vertex u reaches its neighboring vertex v, or the moment when the neighboring vertex v reaches the vertex u, namely the outgoing and incoming edges of a vertex at different moments, as shown in the black edges with arrows.



(a) Original temporal graph (b) Transformed static graph

Figure 4. Example of a temporal graph

Considering merely the temporal reachability between nodes, we first transform the temporal graph into a static graph, as shown in Figure 4. Then, through existing random walk strategies of nodes and the *skip-gram* model in natural language processing, we design a basic node embedding algorithm (Basic algorithm) on temporal graphs.

The basic principle of the algorithm is as follows. First, in temporal graphs, nodes lose their degree of "freedom" to walk due to the limitation of time. When the initial node v walks to the node u, it needs to select the neighboring nodes $\{u_1, u_2, \ldots, u_n\}$ of the node u to be walked. At this point, it should be determined whether the arrival moment $Arr_{(v, u)}$ of the node u is smaller than or equal to the maximum connection moment $\max(T_{(u,ui)})$ with its neighboring node u_i, and only the node u_i that satisfies the condition can be walked. Then, we can select one or more nodes from the nodes satisfying the condition to walk and sample. The walking list can be obtained after all the vertices are sampled. Then, we can obtain the vector representation of nodes with temporality by the skip-gram model.

Algorithm 1. Basic temporal embedding

Input: the temporal graph $G_T(V, E, T_E)$, the skip-gram model, and the walking step L;
Output: the representation vector Z of nodes in the temporal graph.

```
1.  WL,ul=∅;   //WL: the set of the walking lists of all the nodes; ul: the walking list of the node u
2.  FOR node u in G_T
3.    ul=u   //initialize the walking list of the node u
4.    WHILE |ul|<L   //control the walking length
5.      u=ul[−1]   // take the end node of the walking list and use it as the origin of the next walk
6.      FOR node v in N_u   //N_u: the set of the neighboring nodes of u
7.        Rand choose node v in V∈N_u and Visit_(u)≤max(T_(u,v))   //temporally reachable nodes exist
8.        Visit_(v)=t for t in T_(u,v) and t>Visit_(u)   //record the arrival time of the node v
9.        ul=ul∪v   //update the walking list ul of the node v
10.       IF ∀_(v in U) max(T_(u,v))>Visit_(u)
11.         BREAK   //fails to satisfy temporal reachability and the walk terminates
12.     END FOR
13.   END WHILE
14. WL=WL∪ul   //update the walking list WL
15. END FOR
16. z=Skip-Gram(WL)   //return the vector representation of all the nodes
```

In line 3, the node u is first used as the starting point of the walking list. The line 5 indicates selecting the node at the end of the existing walking list as the starting point of the next walking. Lines 7–11 indicate when the walking list stops at the node u, a node v satisfying the temporal reachability is randomly selected from the set of its neighboring nodes N_u and is added to the walking list ul of the node u. If there is no neighboring node that satisfies temporal reachability, the walk terminates. The codes in lines 14–16 indicate the walking lists of all the nodes are recorded at first (line 14), and the vector representation for each node is obtained by the *skip-gram* model.

However, it should be noted that this basic embedding strategy on temporal graphs cannot overcome the limitations mentioned in Section 2. In particular, it cannot automatically recognize the dynamic rates of change of different types of temporal graphs, nor can it solve the time span

problem of sampling paths. With regard to this, we improve this basic algorithm to better deal with the above limitations.

4 Improvement on Basic Temporal Node Embedding Strategy

Since the dynamic rate of change varies greatly with different types of temporal graphs, the time span of sampling paths is very different in various types of temporal graphs. As the Basic algorithm proposed in the previous section cannot effectively solve these two problems, this section proposes a new embedding strategy for adapting to the dynamic changes of temporal graphs based on the Basic algorithm.

4.1 Adaptive temporal graph embedding

To solve the problems of the basic embedding strategy proposed in the previous section, we propose an improved sampling strategy of adaptive temporal graph embedding (ATGEB). The principle of the strategy is that the dynamic changes of networks are generated by messages spreading in the networks, and messages are propagated by establishing connections between users. Messages also change with time, namely that messages have a propagation lifetime in temporal networks. The time spans of different messages Inf_i and Inf_j spreading in networks can be completely coincident, partially coincident, and completely separated. From a global perspective, it is difficult to distinguish these different messages by time. However, if the propagation of messages is analyzed specially in a single node u, it is possible to preserve the characteristics of these messages indirectly by the connection between nodes. Assuming the messages Inf_i and Inf_j are both propagated in the time span $[t_1, t_2]$, when the messages are propagated to the nodes u_i and u_j, respectively, we can distinguish the two messages indirectly by observing the nodes that have connections with the two nodes in $[t_1, t_2]$ as the nodes that different messages act on may vary.

On this principle, we can preserve the temporal relationships between the node u_i and its neighboring nodes in the propagation of different types of messages as much as possible by letting the node u_i walk under different message Inf_i. However, it should be noted that researchers generally cannot obtain the propagation paths of specific messages due to the protection of user privacy. We can study the problem from the set of activity moments of nodes in view of the propagation characteristics of messages.

The connection moment between the node u and its neighboring node N_u is $T_u = \bigcup_{v \in N_u} T_{(u,v)}$. T_u also includes the active time span of the node u: $TS_u = \max(T_u) - \min(T_u)$, the activity times $|T_u|$, and the activity frequency $AF_u = |T_u|/TS_u$. In the time span $[\min(T_u), \max(T_u)]$, the distribution of the moments $t_1, t_2, \ldots, t_{|Tu|}$ in the set T_u is not uniform as the propagated messages are different. Thus, we can distinguish the messages transferred by the node u by clustering the moment t_i with the unsupervised DBSCAN algorithm. The average active interval of nodes is set as the object radius $E = TS_u/(|T_u| - 1)$, which can be calculated as follows.

First, we sort the moments in T_u, obtaining $T_u' = [t_1, \ldots, t_{|Tu|}]$. Thus, the total interval is $\triangle_{T_u} = \sum_{i \in [1, |T_u|-1]} (t'_{i+1} - t'_i) = (t'_2 - t'_1) + (t'_3 - t'_2) + \ldots + (t'_{|T_u|} - t'_{|T_u|-1}) = \max(T_u) - \min(T_u) = TS_u$, and the number of intervals is $|T_u| - 1$. So, $E = TS_u/(|T_u| - 1)$.

We can obtain many set classes $C_1, \cdots, C_k \left| \bigcup_{i \in [1,k]} C_i = T_u \right.$ and $\bigcap_{i,j \in [1,k] | i \neq j} (C_i, C_j) = \varnothing$ by clustering T_u with the DBSCAN algorithm, where $C_i = [t^1_{C_i}, \cdots, t^{|C_i|}_{C_i}]$, which are indirectly viewed as activity ranges of messages. For example, the time span of the set C_i is $[t^1_{C_i}, t^{|C_i|}_{C_i}]$. Then, the node u walks in the time periods that are clustered, obtaining the walking list in each time period. Thus, the temporal relationships between the node u and its neighboring nodes under different messages can be saved. The specific steps are described in Algorithms 2 and 3.

This unsupervised clustering method is a good solution to the problem that the activity frequencies of nodes vary in different types of temporal graphs. Because after the object radius E

is set, the moments can be clustered automatically. If a node u sends messages regularly to its neighboring nodes, namely that AF_u remains constant, this indirectly shows the similarity of the propagated messages or even indicates that one message is being transmitted all the time. By this adaptive clustering method, the elements in T_u can be clustered to a same type. This can reduce the possibility of repeated sampling at different time periods and avoid excessive redundancy of collected data to a certain extent.

Algorithm 2. ATGEB

Input: the temporal graph $G_T(V, E, T_E)$ and the *skip-gram* model;
Output: the representation vector Z of nodes in the temporal graph.

1. $WL,ul=\varnothing$; //WL: the set of the walking lists of all the nodes; ul: the walking list of the node u
2. **FOR** node u in G_T
3. $T_u,ul=\varnothing$ //initialize the active moments (connected with its neighboring nodes) and the walking list of the node u
4. **FOR** node v in N_u //N_u: the neighboring node of the node u
5. $T_u=T_u\cup(T_{(u,v)})$
6. $T_u' = Sort(T_u), E = TS_u/(|T_u|-1)$ //sort the moments in T_u and set the object radius
7. $C_1,...,C_k = DBSCAN(T_u', E)$ //cluster the sorted T_u' in the radius E
8. **FOR** C in C_i
9. $ul=ul\cup PathTree(u,C)$ //summarize the walking lists in different time periods to update the walking list of u
10. **END FOR**
11. **END FOR**
12. $WL=WL\cup ul$ //update the walking list WL
13. **END FOR**
14. $Z=Skip\text{-}Gram(WL)$ //return the vector representation of each node

The problem that how the node u walks to its neighboring nodes at different time period C_i (as shown in line 9) is explained in Algorithm 3: The node u selects a node v from its neighboring nodes N_u; the node v has connection with u in the time period C_i, and the connection moment $T \in T_u$ approximates or equals to the initial time period $\min(C_i)$. Then, the selected node v walks to track the propagation path of messages.

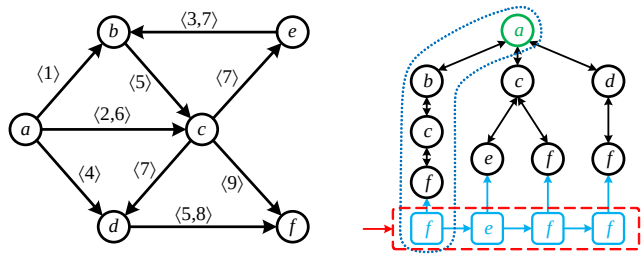
Algorithm 3. PathTree

Input: the temporal graph $G_T(V, E, T_E)$, the node u, and the time period C_i
Output: the walking list ul of the node u in the time period C_i

1. $u.prev,u.next=\varnothing,u.name=u$ //*prev* and *next* save preceding nodes and successor nodes, respectively
2. $Q.pull(u),TL=\varnothing$ //u 作为树的根节点,TL 保存树的叶子节点地址 u indicates the root node and TL saves the addresses of leaf nodes
3. **WHILE** Q is not empty //build a walking tree
4. $u\leftarrow Q.push$
5. **FOR** each node v in N_u
6. **IF** $N_u=\varnothing$ //record the node as a leaf node if there is no neighboring node and the walk terminates
7. $TL.pull(u)$ //record it as a leaf node

8. **BREAK**
9. **ELIF** $T_{(u,v)}$ exist in [min(C_i),max(C_i)] and $u.Arr_u$<max(C_i) //determine whether the node can reach within the specified time period
10. $v.Arr_v=t$ for t in $T_{(u,v)}$ and $t>=Arr_u$ //record the arrival time of walk to the new node
11. $v.prev=u$, $v.name=v$; //point out the preceding node to extract the walking list
12. $u.next.pull(v)$ //save the successor node
13. $Q.pull(v)$
14. **ELIF** $T_{(u,v)}$ not exist in [min(C_i),max(C_i)]
15. $TL.pull(u)$ //the path is unreachable in specified time, so the path ends and the node is set as the leaf node
16. **END FOR**
17. **END WHILE**
18. **FOR** tree leaves tl in TF: //search for several walking paths by a leaf node
19. $list=\varnothing$ // initialize the walking list
20. **WHILE** $tl!=\varnothing$
21. $list=list\cup tl.name$ //obtain the name of the node
22. $tl=tf.prev$ // search the node list for the root node by the leaf node
23. **END WHILE**
24. $ul=ul\cup tl$ //obtain a list set of the node u
25. **END FOR**
26. **RETURN** ul //return the walking list of the node u in the specified time period

Algorithm 3 is explained in detail with the example shown in Figure 5. First, it should be noted that the tree in Figure 5 is a multi-branch one that is temporally reachable, namely that the paths from the root node to leaf nodes are temporally reachable. In line 1, the name of the root node, the set of the successor nodes, and the preceding nodes are initialized. Since it is a multi-branch tree, the non-leaf nodes of the tree keep the set of addresses of their successor nodes, while the preceding nodes save a single address. Lines 5–8 indicate that if the node has no neighboring nodes, it is recorded as a leaf node and its address is recorded in the leaf node set *TL*. Lines 9–13 reveal if there is a neighboring node v that satisfies the requirement, namely that the path is reachable in the time period C_i, the arrival time and the addresses of the preceding and successor nodes of the node v are updated, and the node name is recorded and put into the tree. Lines 14 and 15 signal the node u has no reachable neighboring nodes. Then, the node u is used as a leaf node and its address is recorded in *TL*.



(a) Temporal graph G_t (b) Path tree for node a

Figure 5. Selection of walking paths of temporal graph

Then, lines 18–24 indicate extracting the walking list from the addresss of the leaf nodes recorded in *TL*. As shown in Figure 5(b), the node f walks to the root node a, so a node list ul_1 = 'f,

c, *b*, *a*' can be obtained. It should be noted that this list is opposite to the real list. Therefore, in the implementation of the algorithm, it is necessary to put the proposed node at the top of the list each time, so that the real list ul_1 = '*a*, *b*, *c*, *f*' can be obtained. In line 24, *ul* indicates the list set of the node in this time period, e.g., *ul* = [[*a*, *b*, *c*, *f*], [*a*, *c*, *e*], [*a*, *c*, *f*], [*a*, *d*, *f*]] in Figure 5(b). Thus, a set of walking lists of the node *u* related to the message Inf_i can be gotten in the time period C_i.

4.2 Simple analysis of embedding accuracy

From the perspective of the generation of representation vectors of nodes, the operation principle of the *skip-gram* model in node embedding is introduced. As shown in Figure 6, the *one-hot* vector of a node is weighted by the hidden layer and then classified by the output layer. The differences between the softmax layer of the nodes in the moving window *w* and the other nodes in their probabilities and the outputs are calculated and used as the losses, which are transferred backwards to update the weight of each layer. Then, the updating is continuously conducted by the moving window. At last, the vector representation *Z* of the node can be obtained by the weight of the hidden layer *H*.

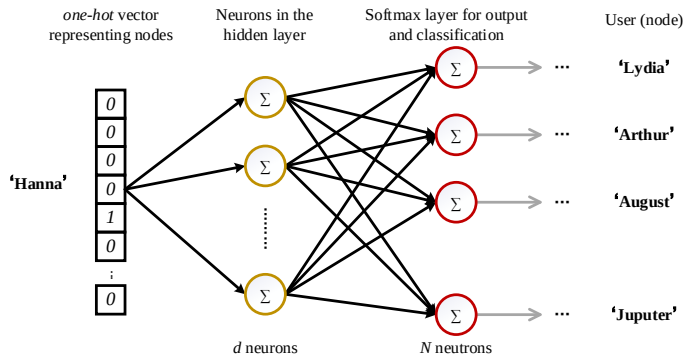


Figure 6. Diagram of node embedding

It is assumed that there are *I* messages transferred by users in a temporal network: Inf_1, Inf_2, …, Inf_I, and there are *L* node labels (classes): Lab_1, Lab_2, …, Lab_L. The users with different labels have different sensitive degrees to different messages. This indicates that labels determine that users have different probabilities of receiving and propagating different messages, namely that users contact different users through different messages. On the contrary, it can be deduced that several kinds of messages Inf_i, Inf_j, …, Inf_k can indirectly determine the labels of users, $f(Inf_i, Inf_j, \ldots, Inf_k) = Lab_i$. Thus, the label of a user can be more accurately determined by the type of the messages the user receives or propagates.

In a neural network with classified nodes, the higher similarity between the vectors of two nodes means the greater possibility that the two nodes belong to one type of nodes, namely

$$[sim(z_i, z_j) > sim(z_i, z_k)] \approx [P(u_i, u_j \in Lab_l) > P(u_i, u_k \in Lab_l)].$$

In Figure 6, the user list is indicated by *U*; the *softmax* layer by the output layer *O*; the hidden layer by *H*. In a trained *skip-gram* model, for the user nodes *u* and *v* with the same label, it is assumed that the users corresponding to the first *w* classes are $(U^{u_1}, U^{u_2}, \ldots, U^{u_w})$ and $(U^{v_1}, U^{v_2}, \ldots, U^{v_w})$ after they are output by the *softmax* layer. The embedding vectors of the two are z_u and z_v. It has been known that the value $H = W_H \times [0, 0, 0, \ldots, 1, 0]^T$ of the neuron in the hidden layer directly determines the vector of a node and the class *U* of the node after the output layer, namely $H^u \rightarrow z_u$ and $H^v \rightarrow z_v$. As it has been known that the nodes *u* and *v* have the same label, the

embedding vectors of the two nodes are similar: $z_u \approx z_v$, so $H^u \approx H^v$. As the hidden layer also determines the output layer, there is $(U^{u_1}, U^{u_2}, ..., U^{u_w}) \approx (U^{v_1}, U^{v_2}, ..., U^{v_w})$.

The above mainly describes the forward propagation mechanism in the node embedding network. The updating method of the hidden layer is further indicated based on this. It is assumed that the node u is in a walking list ul and a window $Win_u = \{u_1, \ldots, u, \ldots, u_w\}$ with a size of w is constituted with u as the central. In this window, the training error of networks $\Delta E = \sum_{v \in Win} (y' - P(u,v))^{1/2}$, where $y' = Onehot_u \times W_H \times W_O$. As the size of Win_u is limited, it is impossible to put all the nodes in it. Thus, for a node except the node u, with the probability of it is in the window Win_u drops, $P(u, v)$ decreases. As the training goes on, the classification of the *one-hot* vector $Onehot_u$ of u in the *skip-gram* network tends more towards the nodes that have a high frequency in Win_u, which can be known by the characteristics of the output layer softmax. It can be known that the frequency of the node u in the window Win_u influences its classification in the *skip-gram* model. Further, it also affects the value H^u of the neutron in the hidden layer to influence z_u according to the analysis in the last paragraph

Thus, it can be known from the above analysis that for the nodes u and v with the same label, to make their embedding vectors z_u and z_v similar, it is necessary to make the sets of high frequency nodes in the moving window Win_u and Win_v that use the nodes u and v as the central nodes in all the walking list ul coincide as much as possible, and the nodes with the same label should have the similar coincidence degree. This is because if the nodes with different labels have similar coincidence degree, they can generate the similar representation vector Z, which is contradictory to the premise that they have different labels.

For the nodes u and v with the same label, it is assumed that the sets of their possible walking nodes are S_{ul}^u and S_{ul}^v respectively, and the possible walking sets in Algorithm 2 are T_{ul}^u and T_{ul}^v respectively. It can be known that $T_{ul}^u \subseteq S_{ul}^u, T_{ul}^v \subseteq S_{ul}^v$. For the walking windows $Win_u = \{u_1, \ldots, u, \ldots, u_w\}$ and $Win_v = \{v_1, \ldots, v, \ldots, v_w\}$, if the label of the two nodes is related to Inf_i and Inf_j and it is assumed that the set of active nodes influenced by the two messages is $S_{Inf}, S_{Inf} \in T_{ul}^v \cap u, v \in S_{Inf}$, the nodes in S_{Inf} indirectly show that they are active under the impact of Inf_i and Inf_j. Thus, compared with the random walk strategy, in the walking path ul including the nodes u and v under the proposed walk strategy, the $1/2(w-1)$ nodes before and after the node $u(v)$ (namely the nodes in the window Win_u or Win_v) are more likely to be included in S_{Inf}, namely that the nodes in the window have a stronger probability to have a same label with the node $u(v)$. This indicates there is a high probability of common nodes in the windows with the two nodes as the central nodes (compared with the random walk strategy). As analyzed above, if there are more common high-frequency nodes, more similar vectors can be obtained.

5 Sampling of Important Graph Nodes

In graphs, there are usually a large number of nodes gathering in a community within a very short distance due to their complex topological structures. It can be seen from the principle of the node embedding algorithm that many nodes in a dense community will obtain several similar walking lists, which are further encoded to similar vectors. If these similar vectors belong to a same class C_i, and the classification of nodes should be completed in a period as short as possible and can be tolerant of certain model errors, is it possible to train the parameters of the neutral network by sampling several vectors with similar nodes in the same type? In this way, the parameter weights of the neutral network can be trained in a period as short as possible.

The walking lists of the nodes u_i, u_j, and u_k are u_il, u_jl, and u_kl, respectively. If the distance between u_i and u_j is $d(u_i, u_j) > d(u_j, u_k)$, the node u_i is closer to u_j than to the node u_k. Thus, generally the common neighboring nodes of the two nodes also satisfy $N_{ui} \cap N_{uj} > N_{uj} \cap N_{uk}$. Thus, for the node v walking in the range of w (the size of which is same as that of the window of the *skip-gram* model), after its walking list vl passes the node u_j, the probability of the walking list passing the node u_i is stronger than that of the walking list passing the node u_k, namely $P(vl \rightarrow u_j | vl \rightarrow u_i) > P(vl \rightarrow u_j | vl \rightarrow u_k)$. On the contrary, the probability of the walking list from the node u_i to the node u_j is also higher than that of the walking list from the node u_i to the node u_k. Therefore, for a walking list of a random node $wl \in WL$, the probability of its including both u_i and u_j in a window range with a size of w is greater than that of its containing both u_j and u_k, namely $P(u_j, u_i \in wl) > P(u_j, u_k \in wl)$. From a global perspective, $\sum_{wl \in WL} P(u_i, u_j \in wl) > \sum_{wl \in WL} P(u_j, u_k \in wl)$, so the times of u_i and u_j together in the window w is higher than that of u_j and u_k. In the *skip-gram* model, the vector of the node u is updated through the other nodes which are also in the window w with u in the walking list wl. Hence, among the vectors z_{ui}, z_{uj}, and z_{uk} of the three nodes, the first two should have greater similarity, namely $sim(z_{u_i}, z_{u_j}) > sim(z_{u_j}, z_{u_k})$.

In view of the above analysis, it can be known that we can select several dense subgraphs g_1, g_2, …, g_m in a graph as dense communities to select important nodes. The dense subgraph g_i should satisfy that for any two nodes v_i, $v_j \in g_i$, the distance between the two nodes $d_{(vi,vj)}$ should be as short as possible. The most obvious idea is to mine dense communities by the *k-core* algorithm. Although the *k-core* algorithm can be used to obtain the dense subgraph g_i by increasing the value of k, the subgraphs obtained by this method cannot make $d_{(vi,vj)}$ as small as possible. In Figure 7, for example, after the nodes *e*, *f*, and *g* are deleted, the remaining nodes constitute a complete 2-*core* subgraph. However, the distance $d_{(c,d)}$ between the nodes *c* and *d* is 6. On the small-world principle, we can know that it is a long distance in graphs. Therefore, it is difficult to meet the demands of mining communities only by satisfying the density requirement, and the distance between nodes in a community should also be limited. The *kr-Clique* algorithm stated in Reference [28] can satisfy the demands, where k indicates the degree of nodes, r the hop of nodes, namely that any two nodes in a community can reach to each other in r hops.

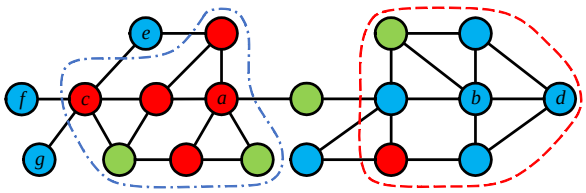


Figure 7. *k*-core communities of graph

The *kr-Clique* algorithm stated in Ref. [28] aims to find all the communities that satisfy requirements, but it is unnecessary in this paper. The sampling of important nodes is to select more representative node vectors than the node class C_i to constitute the training sample set, which does not need to select all the nodes. On this principle, several nodes v_{c1}, v_{c2}, …, v_{cn} with high degrees are selected in the graph. Then, with these nodes as the central nodes, the node $u \,|\, (u \in N_{v_{c1}} \cap d(u) > k)$ with the degree of neighboring nodes higher than k is selected. The neighboring nodes of the eligible node u are selected with the node u as the central node. These steps are repeated for r times, and then a rough *kr-Clique* algorithm about the central node v_{c2} can be obtained, which can satisfy requirements. After the *kr-Clique* communities of the central node

v_{c1} are selected, the nodes are clustered according to classes, obtaining the node sets S_{C1}, S_{C2}, …, S_{Cc}. Then we can randomly select node vectors from each cluster with a small percentage (10% for example) to constitute the training set rather than use 60% of the nodes in the graph as the training set. This is because many of the 60% nodes belong to a same class and the vectors have high similarities. Then we can select important nodes of the next central node v_{c2}.

This section selects central nodes and important nodes by an algorithm similar to the node-degree-based heuristic algorithm in the maximum influence problem [29], as shown in Algorithm 4.

Algorithm 4. Important nodes sampling (INS) algorithm

Input: the temporal graph $G_T(V, E, T_E)$, the node classes C_1, C_2, …, C_c, and the number of central nodes N;
Output: the vector set Z^{IN} of important nodes.

1. $krC=\varnothing, i=0, Z^{IN}=\varnothing$ //krC: the set of selected community nodes
2. **WHILE** ($i<N$)
3. **FOR** node u in $W=G_T \cap krC$
4. Chose u_i as central nodes if $d(u_i)>d(u_j)|(u_j \in W)$ //select the node with the maximum node degree as the central node
5. $krC_{u_i}=u_i, BN_1=u_i, BN_2=\varnothing$ //BN_1 and BN_2 save the boundary points of the community
6. **FOR** j in *range*(r) //the maximum hops between nodes in the community
7. **WHILE** $BN_1!=\varnothing$
8. $v=BN_1.pop$ //select new nodes to the community through boundary points
9. $BN_2.pull(v_i), krC_{u_i}.pull(v_i)$ for v_i in N_v and $d(v_i)>k$
10. **END WHILE**
11. $BN_1=BN_2, BN_2=\varnothing$ //update the boundary points in BN_1 through BN_2
12. **EBD FOR**
13. **END FOR**
14. Chose important nodes S in krC_{u_i} in a certain proportion //select the set S of important nodes
15. $Z^{IN}=Z^{IN} \cup S, krC=krC \cup krC_{u_i}$
16. **END WHILE**
17. **RETURN** Z^{IN} //return the vector set of important nodes

In line 1, krC indicates the set of the selected community nodes, and i controls the number of the selected central nodes in the community. Lines 3 and 4 mean the new central node u_i should not be in the already selected communities, namely that the distance between u_i and all existing central nodes should be at least 2, and the degree of u_i should be greater than any other node. In this way, the central nodes can be selected in the widest possible range of the graph rather than in a local area of the graph. For example, in Figure 7, a (3, 3)-*Clique* is selected with *a* as the central node, and *b* is selected as the central node among the remaining nodes. Lines 5–12 mean forming the krC_{ui} community with the node u_i as the central node. First, we search for the suitable neighboring nodes through BN_1 to expand the community krC_{ui}, and record the boundary points to BN_2. After these steps are completed, the boundary points of BN_2 are assigned to BN_1 for new expansions. Line 14 shows the selection of important nodes, which has been explained above.

The above-mentioned important nodes are selected based on dense communities. However, there are many nodes or new connected structures that are outside of these communities in actual

graphs. To unify the representative nodes, it is necessary to randomly select nodes from these dissociated nodes in a same proportion to add them to the training sample set Z^{IN}.

6 Experiments and evaluation

To fully verify the performance of the proposed algorithm on different types of data based on the design and verification of the algorithm test, we select four actual datasets which vary greatly in node (edge) size, distribution of temporal edges, density, and other graph structures as the input data, and verifies the algorithm from four perspectives of node clustering, link prediction, node classification, and temporal reachability of nodes.

6.1 Data and parameter setting

(1) Data

The datasets D0 [30] and D1 [31] come from two Bitcoin trading platforms Bitcoin OTC and Bitcoin Alpha, respectively. These two datasets contain financial transaction data of users, not ordinary social networks, where users are connected to their neighbors only once, which can be seen by the fact that the temporal edges are equal to the static edges. The datasets D2 [32] and D3 [32] come from temporal network data of the stack exchange websites Math Overflow and Super User, respectively. The node size, static edges, temporal edges, and activity frequency of the four datasets are listed in Table 2. In addition, it can be seen from Table 2 that the datasets vary largely in different indexes, so they satisfy the experimental requirements.

Table 2 Information of datasets

Dataset	Number of nodes	Temporal edge	Average temporal edge	Static edge	Average static edge	Time span (day)	Activity frequency
D0	6k	36k	6.05	36k	6.05	1 903	18.91
D1	4k	24k	6.39	24k	6.39	1 901	12.62
D2	25k	507k	20.41	240k	9.67	2 350	215.74
D3	194k	1443k	7.44	925k	4.77	2 773	520.38

(2) Comparison algorithms

In addition to apply the proposed algorithm to the four datasets, we also adopt the following three algorithms to the datasets for comparison. Since some algorithms do not support the temporally reachable walk strategy, the temporal graphs are viewed as static graphs when these algorithms are applied to the datasets, namely that time stamps on edges are ignored.

DeepWalk: First, the time stamps are deleted from the temporal graph G_T, and it is transformed to a static directed graph. Then, the nodes are encoded to vectors by the DeepWalk algorithm based on the static graph;

Node2vec: It is different from the DeepWalk algorithm in the following aspects. When the node t walks to the neighboring node v after it reaches the node u, the walking probability $\alpha_{pq}(t, v)$ of every neighboring node of the node u should be calculated, and then the node v walked at the next step is selected by the Alias sampling method. The walking probability is calculated by Equation (1):

$$\alpha_{pq}(t,v)=\begin{cases}1/p, & \text{if } d_{tv}=0\\ 1, & \text{if } d_{tv}=1\\ 1/q, & \text{if } d_{tv}=2\end{cases} \tag{1}$$

where d_{tv} indicates the path length between the two nodes. In this test, p is set as 0.25; q as 4. The

Alias algorithm is used to perform walk and sampling;

CTDNE [33]: The principle of this algorithm is consistent with that of the Basic algorithm in this paper and the DeepWalk algorithm. During node walk, researchers set a probability of nodes related to time, namely biased sampling, to influence the selection of neighboring nodes by the given node.

(3) Experimental environment

The experiment is conducted under the conditions including Windows 10 64-bit operating system, CPU with i5-8400@2.80Hz, 24G RAM, 500 GB hardware capacity. The program language is Python 3.7.

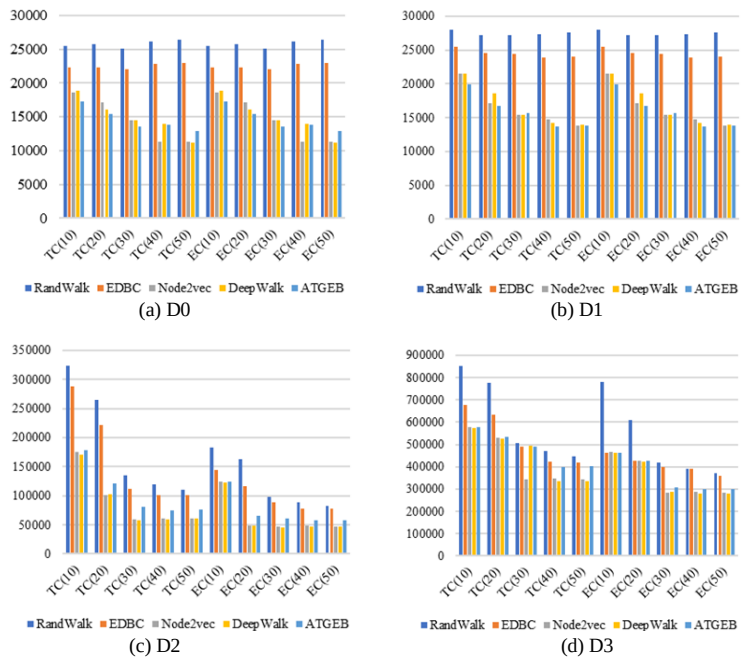
(4) Parameters

When we use the *skip-gram* model to generate vectors, the window length is set as 5; the walking steps L of nodes are set as 10, 20, 30, 40, and 50, respectively; for the vector z_u generated by the node u, the dimension d = 100; the learning rate r = 0.01. The loss function in the classification task is the cross-entropy loss function.

6.2 Node clustering

In the node clustering test, the clustering performance of each algorithm is evaluated by the number of static edges crossing clusters (*EC*) and the number of temporal edges crossing clusters (*TC*). Through the representation vectors of nodes, the nodes are clustered to N classes: C_1, C_2, …, C_N, where C^v indicates the class that the node v belongs to. Thus, $EC = \bigcup_{v \in G_T} (v,u) \mid S \in N_v \cap (\forall_{u \in S} C^u \neq C^v)$, namely that when the node v and its neighboring node u belong to different classes, the edge (v, u) crosses different clusters. $TC = \bigcup_{e \in EC} T_e$, where e indicates the edge in *EC*; T_e the set of moments of the edge e.

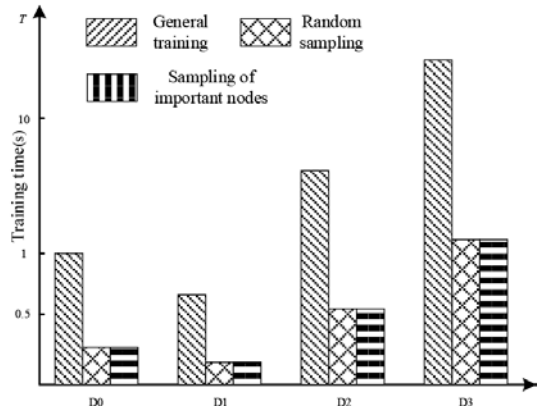
The nodes are clustered to four classes and five classes respectively to analyze the experimental results more objectively. Then the performance of each algorithm with different clustering methods is calculated, as shown in Figures 8 and 9, where i in *TC*(i) and *EC*(i) indicates the walking length of nodes and varies from 10 to 50.



(a) D0 (b) D1

(c) D2 (d) D3

Figure 8. Node clustering in a temporal graph



Dataset / Algorithm	D0		D1		D2		D3	
Basic	43.2	46.7	50.1	52.1	49.7	51.2	47.5	50.6
Deepwalk	46.5	49.4	52.4	54.5	54.9	56.3	53.8	55.9
Node2vec	47.3	49.6	52.5	54.3	55.6	57.1	52.9	55.7
CTDNE	45.8	47.6	51.0	53.1	51.4	54.2	50.1	52.6
ATGEB	46.3	49.7	51.3	53.5	53.2	55.1	51.7	54.8

Figure 9. Analysis of sampling of important nodes

For the subgraphs in Figure 8, the vertical axis indicates the number, and the horizontal axis the walking length of nodes in TC and EC, which varies from 10 to 50. It can be seen from Figures 8(a) and 8(b) that the proposed method converges faster in the initial stage, but gradually tends to be stable with the increase of the walking length. This is because once the sampling strategy is limited by time, the walking path will end automatically when it reaches a certain length. It can be seen that the proposed method is basically stable after 30 steps. Thus, the walking length should be within 30 steps. From Figure 8(a) and Figure 8(b), the ATGEB algorithm performs better in the early stage, but its performance is basically same with DeepWalk and node2vec as the walking length increases. From Figure 8(c) and Figure 8(d), the latter two algorithms perform better in the datasets D2 and D3. Through this experiment, we aim to uncover that for the traditional node clustering problem, the strategy based on temporal node embedding is more sensitive to the attributes of datasets, namely that it has some advantages for some specific types of datasets. However, it is not as versatile as the traditional algorithms such as DeepWalk and node2vec which can be applied to all types of data.

6.3 Link prediction and reachability test

In the test, sample vectors are selected for each node v respectively in the link prediction, and $1/2|N_v|$ neighboring nodes of the node v are selected as positive samples. $z^{(u,v)} = z^u+z^v$, where "+" indicates connection, and the corresponding class is 1, indicating there is an edge between the two nodes. Then, we randomly select the nodes $u_1,u_2,...,u_{1/2|N_v|}$ with the same number in the graph, and determine whether there is an edge between the node v and the node u_i. If there is an edge, the corresponding class of the vector $z^{(v,ui)}$ is 1; otherwise it is 0. In the test for detecting temporal reachability, we set the class of vectors by calculating whether the two related nodes satisfy temporal reachability. After samples are collected for all the nodes, a total of 20% of them constitute the training set and the remaining 80% make up the test set. The test structure can be seen in Table 3, and the values are all the highest ones among the results under different walking lengths.

First, it should be noted that the performance of the node embedding of temporal graphs in link prediction is limited by its walk strategy, and its performance is probably worse than that of the normal walk strategy, especially in the case of few connection moments between nodes. As shown in Table 3, the method based on temporal sampling has obvious worse performance in link prediction in the datasets D0 and D1 because the connection between nodes is single, seen from the data information in D0 and D1. Since the networks are meant for bitcoin trading, there is only one connection between users in the global range, namely that the temporal edges are the same as the static edges. This also leads to the fact that the walking length of any method based on the temporal walk strategy is largely limited by time. However, as the number of connections between users increases, the performance of the proposed algorithm is less poor than that of the DeepWalk algorithm and the node2vec algorithm. The proposed algorithm only performs slightly worse than the two algorithms in D2, and has better experimental results in D3.

Table 3 Accuracy of link prediction and temporal achievability (%)

Problem / Dataset / Algorithm	Link prediction				Temporal reachability			
	D0	D1	D2	D3	D0	D1	D2	D3
Basic	64.2	61.0	60.9	56.6	65.3	63.2	62.9	58.8
Deepwalk	79.3	**82.5**	**71.9**	80.8	76.3	78.6	70.8	78.3
Node2vec	**81.9**	81.8	70.2	79.7	79.4	79.5	68.1	77.9
CTDNE	67.9	64.7	65.9	59.4	81.7	80.7	71.9	80.3
ATGEB	78.4	76.4	71.6	**81.5**	**85.3**	**82.9**	**72.8**	**82.7**

The performance ranking of the algorithms changes in the detection of temporal reachability between nodes. In this problem, the accuracy of the traditional walk strategy plummets. This is because the traditional walk strategy does not consider the temporal relationship between nodes at all, namely that the representation vectors of nodes do not fully preserve the temporal feature of nodes. By contrast, the method based on the walk strategy in temporal graphs satisfies temporal reachability in the sampling process. It can be seen that the ATGEB algorithm performs much better in the datasets D0 and D1, while its advantages in the datasets D2 and D3 are not so obvious. This is because in the latter two datasets, nodes have higher activity frequency and more frequent connections, so the paths have a higher probability to satisfy temporal reachability than that in the first two datasets.

6.4 Node classification and sampling of important nodes

This section focuses on classification of nodes and sampling of important nodes. In the sampling of important nodes, the training time and results with different training sets and different selection methods of training sets are compared. There are three ways to select the training set: the first one is to normally select 20% of the dataset S as the training set; the second one is to randomly select 10% of the dataset S as the training set; and the last one is the sampling method of important nodes stated in Section 4, which selects the data with the same amount of the second way to constitute the training set. Then, we compare the training time and results in the same neutral network to determine whether the sampling of important nodes can train data in less time with smaller error.

In the test temporal graph, the classes that the nodes belong to are not labelled. So, in this test, we use two ways to label nodes with three classes for the simulation test. In the actual datasets, there is a smaller distance between nodes belonging to the same class. On this basis, the nodes are labelled and classified in the following two ways.

The first way is to select several nodes u_1, …, u_k that are as far away as possible, which are labelled as C_1, C_2, and C_3, respectively. Then with these nodes as the central nodes, the set S_{u1} of the neighboring nodes of these node can be obtained. A total of 60% of the nodes in the set are

labelled as the same classes as the central nodes, and the remaining nodes are labelled randomly;

The second way is that the paths should satisfy temporal reachability when central nodes select neighboring nodes, and the set $S_{u_1}^T$ is obtained. Then, the nodes are labelled in the same method as the first way.

By comparing these two ways, we aim to objectively evaluate the experimental results of the proposed method. We do not label the nodes with three classes in a random way. This is because random labelling would assign different classes to two similar vectors and only recognize only one class. So, we abandon this strategy after experiments. Nevertheless, this simulation method can still encounter this problem inevitably, we can only increase the accuracy as much as possible.

The experimental results are shown in Table 4. The performances of traditional sampling methods are getting worse when the temporality of nodes is considered. The method based on the temporal walk strategy can better preserve the temporal relationship between nodes, thus obtaining better experimental results. In Figure 9, it should be noted that as some datasets have a small size and short training time, the horizontal coordinates of the left subgraph are not plotted proportionally to ensure the visual effect of the experimental results. Meanwhile, the sampling results of important nodes are compared with the results of the first sampling strategy (without considering the temporality). Below the corresponding data, the left dataset shows the results of random selection, and the adjacent right dataset demonstrates the experimental results of nodes selected by the sampling method of important nodes. According to the two subgraphs on the left and right, the sampling strategy of important nodes can train the neutral network in a shorter time within tolerant errors (smaller than 2% in this test).

Table 4 Accuracy of node classification (%)

Problem / Dataset / Algorithm	Without considering temporality				Considering temporality			
	D0	D1	D2	D3	D0	D1	D2	D3
Basic	47.8	53.1	53.8	52.3	47.4	53.7	54.9	53.1
Deepwalk	49.7	55.6	57.6	**56.6**	48.2	54.3	55.8	54.3
Node2vec	50.2	**55.9**	**57.9**	56.4	48.6	54.5	56.5	55.2
CTDNE	48.6	54.3	55.7	54.7	49.2	54.9	54.9	53.8
ATGEB	**50.4**	54.8	56.3	56.2	**51.1**	**55.8**	**58.6**	**57.6**

7 Related work

Different from static graphs, the edges in temporal graphs have two states which can be transformable to each other: active and inactive. The vertices in temporal graphs are connected only when the edges are active. There are many examples of temporal networks in practice:

(1) Communication networks: E-mails, phone calls, and short messages are typical point-to-point temporal networks;

(2) One-to-many information transformation networks: They are characterized by a single user propagating information to many other users;

(3) Bioinformatic networks: These include brain neutron networks, metabolic networks, and protein-protein interaction networks.

In the database field, the current research on temporal graphs mainly focuses on reachability queries, shortest temporal paths, and prediction of arrival time.

The proposal of the DeepWalk [2] algorithm has brought attention to graph node embedding. The DeepWalk algorithm pioneers the combination of the random walk strategy of graphs and the *skip-gram* [3] model in natural language processing, which encodes words to vectors. The DeepWalk algorithm encodes graph nodes to vectors by this innovative method. With the

DeepWalk algorithm, researchers [4–7] tried to further improve this method by a biased sampling method, thus designing different node embedding strategies such as node2vec, LINE, PTE, and stru2vec. Although scholars claimed that these algorithms achieved better experimental results in link prediction and node classification, they made no essential improvement to the DeepWalk algorithm. In addition, different sampling strategies sometimes show significant distinctive performances on different datasets. This provides inspiration for subsequent researchers to study the node embedding problem. Specifically, with the current limited training ability for neutral networks, the study of vector representation of graph node embedding should focus on specific problems and topological characteristics of graph data. For example, it is different to perform node embedding on heterogeneous graphs from that on knowledge graphs. To solve different problems such as node classification and product recommendation, we need to propose research strategies that satisfy the specific requirements.

Community detection [28,34,35] is always a hotspot research in graph data mining. At the initial stage, researchers focused on gathering users with close topological distance to form a community. However, as the problem was further studied with some practical applications, researchers preferred to call up users with the same interest to form a community based on a common interest. In this community, users usually have common interests or similar attributes. This allows enterprises to more accurately recommend information or products to users in light of the interests.

8 Prospect

With attention given to node embedding of temporal graphs without attributes, we develop some tests and confirm the proposed algorithm is effective and expandable from different perspectives. As graph convolution cannot be used in representation learning for nodes in temporal graphs without attributes, the embedding strategy in this paper cannot encode the node attributes to vectors. The following problems will be studied in the subsequent work. (1) The node representation on attribute-based temporal graphs will be studied. We try to study the connection between the node attribute and the temporal relationship of nodes with the existing graph convolution method, so as to propose a graph representation learning strategy which can conduct convolution for attributed temporal graphs. (2) This paper has conducted the sampling of important nodes for graphs. In the subsequent work, we will consider more for sample nodes. Both node topology and the attribute relationship between nodes will be considered. We will use related mining algorithms of graphs to select the important nodes, and try to make the vectors of the selected nodes more representative and unified, so that the required weight parameters in the neutral network can be trained better and faster.

References

[1] Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G. The graph neural network model. IEEE Trans. on Neural Networks, 2009, 20(1): 61–80.

[2] Perozzi B, Al-Rfou R, Skiena S. DeepWalk: Online learning of social representations. Proc. of the 20th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2014. 701–710.

[3] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. arXiv:1301.3781v3.

[4] Tang J, Qu M, Wang MZ, Zhang M, Yan J, Mei QZ. LINE: Large-scale information network embedding. Proc. of the 24th Int'l Conf. on World Wide Web. 2015. 1067−1077.

[5] Tang J, Qu M, Mei QZ. PTE: Predictive text embedding through large-scale heterogeneous text networks. Proc. of the 21st ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2015. 1165−1174.

[6] Grover A, Leskovec J. node2vec: Scalable feature learning for networks. Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2016. 855−864.

[7] Leonardo FRR, Pedro HPS, Daniel RF. struc2vec: Learning node representations from structural identity. Proc. of the 23rd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2017. 385−394.

[8] Qiu JZ, Dong YX, Ma H, Li J, Wang KS, Tang J. Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and node2vec. Proc. of the 11th ACM Int'l Conf. on Web Search and Data Mining. 2018. 459−467.

[9] Hamilton W, Ying Z, Leskovec J. Inductive representation learning on large graphs. Advances in Neural Information Processing Systems. 2017. 1024−1034.

[10] Kipf TN, Welling M. Semi-Supervised classification with graph convolutional networks. Proc. of the ICLR (Poster). 2017.

[11] Velickovic P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y. Graph attention networks. Proc. of the ICLR (Poster). 2018.

[12] Wang YS, Yuan Y, Ma YL, Wang GR. Time-dependent graphs: Definitions, applications, and algorithms. Data Science and Engineering, 2019, 4(4): 352−366.

[13] Takaguchi T, Yano Y, Yoshida Y. Coverage centralities for temporal networks. European Physical Journal B, 2016, 89(2): 35.

[14] Frand D, Masoud TO, Jörg-Rüdiger S. Shortest paths in FIFO time-dependent networks. Algorithmica, 2012, 62(1−2): 416−435.

[15] Rossi L, Musolesi M, Torsello A. On the *k*-anonymization of time-varying and multi-layer social graphs. Proc. of the 9th Int'l Conf. on Web and Social Media. 2015. 377−386.

[16] Przytycka TM, Singh M, Slonim DK. Toward the dynamic interactome: It's about time. Briefings in Bioinformatics, 2010,11(1): 15−29.

[17] Han JD, Bertin N, Hao T, *et al*. Evidence for dynamically organized modularity in the yeast protein-protein interaction network. Nature, 2004, 430(6995): 88−93.

[18] Lèbre S, Becq J, Devaux F, *et al*. Statistical inference of the time-varying structure of gene-regulation networks. BMC Systems Biology, 2010, 4(1): 1−16.

[19] Wu H, Cheng J, Ke Y, *et al*. Efficient algorithms for temporal path computation. IEEE Trans. on Knowledge & Data Engineering, 2016, 28(11): 2927−2942.

[20] Li J, Han ZC, Cheng H, Su J, Wang PY, Zhang JF, Pan LJ. Predicting path failure in time-evolving graphs. Proc. of the 25th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining. 2019. 1279−1289.

[21] Hu JL, Yang B, Guo CJ, Jensen CS, Xiong H. Stochastic origin-destination matrix forecasting using dual-stage graph convolutional, recurrent neural networks. Proc. of the IEEE Int'l Conf. on Data Engineering. 2020. 1417−1428.

[22] Kumar S, Hamilton WL, Leskovec J, Jurafsky D. Community interaction and conflict on the Web. Proc. of the World Wide Web Conf. 2018. 933−943.

[23] Panzarasa P, Opsahl T, Carley KM. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. Journal of the American Society for Information Science and Technology, 2009, 60(5): 911–932.

[24] Bai C, Kumar S, Leskovec J, Metzger M, Nunamaker JF, Subrahmanian VS. Predicting visual focus of attention in multi-person discussion videos. Proc. of the Int'l Joint Conf. on Artificial Intelligence. 2019. 4504–4510.

[25] Wu HH, Huang YZ, Cheng J, Li JF, Ke YP. Reachability and time-based path queries in temporal graphs. Proc. of the IEEE Int'l Conf. on Data Engineering. 2016. 145–156.

[26] Yuan Y, Lian X, Wang GR, Ma YL, Wang YS. Constrained shortest path query in a large time-dependent graph. Proc. of the VLDB Endow, 2019, 12(10): 1058–1070.

[27] Yuan Y, Lian X, Wang GR, Chen L, Ma YL, Wang YS. Weight-Constrained route planning over time-dependent graphs. Proc. of the IEEE Int'l Conf. on Data Engineering. 2019. 914–925.

[28] Bron C, Kerbosch J. Finding all cliques of an undirected graph (algorithm 457). Commun. ACM, 1973, 16(9): 575–576.

[29] Chen W, Wang YJ, Yang SY. Efficient influence maximization in social networks. Proc. of the 15th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2009. 199–207.

[30] Kumar S, Spezzano F, Subrahmanian VS, Faloutsos C. Edge weight prediction in weighted signed networks. Proc. of the IEEE Int'l Conf. on Data Mining. 2016. 221–230.

[31] Kumar S, Hooi B, Makhija D, Kumar M, Subrahmanian VS, Faloutsos C. REV2: Fraudulent user prediction in rating platforms. Proc. of the ACM Int'l Conf. on Web Search and Data Mining. 2018. 333–341.

[32] Paranjape A, Benson AR, Leskovec J. Motifs in temporal networks. Proc. of the 10th ACM Int'l Conf. on Web Search and Data Mining. 2017. 601–610.

[33] Nguyen GH, Lee JB, rossi RA, Ahmed NK, Koh E, Kim S. Continuous-Time dynamic network embeddings. Companion Proc. of the Web Conf. 2018. 2018. 969–976.

[34] Wang Y, Jian X, Yang ZH. Query optimal k-plex based community in graphs. Data Science and Engineering, 2017, 2(4): 257–273.

[35] Fan WF, Hu CM. Big graph analyses: From queries to dependencies and association rules. Data Science and Engineering, 2017, 2(1): 36–55.

Anbiao Wu (1993–), PhD candidate, CCF student member, mainly engaged in graph databases and graph neural networks.

Ye Yuan (1981–), PhD, professor, PhD supervisor, CCF senior member, mainly engaged in big data management, database theory and system.

Yuliang Ma (1990–), PhD, mainly engaged in graph databases and exploration of location-based social networks (LBSN).

Guoren Wang (1966–), PhD, professor, PhD supervisor, CCF oustanding member, mainly engaged in management of uncertain data, intensive computing of data, management and analysis of visual media data, management of unstructured data, distributed query processing and analysis, and bioinformatics.