IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 20 June 2022, accepted 13 September 2022, date of publication 21 September 2022, date of current version 28 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3208360

RESEARCH ARTICLE

An Efficient Vertex-Driven Temporal Graph Model and Subgraph Clustering Method

HANLIN ZHANG[1], LINLIN DING[1], GANG ZHANG[1], YISHAN PAN[2], AND BAOYAN SONG[1], (Member, IEEE)

[1]School of Information Institution, Liaoning University, Shenyang, Liaoning 110036, China
[2]School of Environment Institution, Liaoning University, Shenyang, Liaoning 110036, China

Corresponding author: Baoyan Song (bysong@lnu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 62072220 and Grant 61502215, in part by the Central Government Guides Local Science and Technology Development Foundation Project of Liaoning Province under Grant 2022JH6/100100032, and in part by the Natural Science Foundation of Liaoning Province under Grant 2022-KF-13-06.

ABSTRACT The temporal graph can represent a temporal relationship widely used in compound synthesis analysis, biological gene analysis, etc. However, the temporal graph would embody vertex updates frequently, high time resolution, and not enumerated rules. The construction and update of some temporal graph models are too dependent on the graph operation sequence, which leads to a lack of an effective model. Simultaneously, the temporal subgraph clustering of the temporal graph with frequent updating for the lack of an effective model leads to low accuracy. Therefore, we propose an efficient and frequently updated temporal graph model as vertex driven and corresponding temporal subgraph clustering method. First, we propose a temporal graph construction algorithm and set two thresholds to divide the temporal graph on a timeline to obtain temporal subgraphs. Next, an enhancement strategy based on the sliding window is proposed to accelerate the construction process. Third, we offer a double-standard temporal subgraph clustering method based on community comparison and temporal distance. The temporal subgraph can be effectively distinguished in temporal and structure dimensions. Lastly, experimental results on both real and synthetic datasets show that the temporal graph model proposed in this work can reduce the time overhead of construction compared to other existing models. The cluster method improves the clustering accuracy of temporal subgraphs. The clustering results show through the hierarchical clustering at the same time.

INDEX TERMS Temporal Graph, temporal graph model, subgraph clustering, sliding window, hierarchical clustering.

I. INTRODUCTION

The Temporal graph [1], [2] is a dynamic network [3], [4] with frequent updates of vertices and edges and high time resolution, unenumerable nature of rules can describe complex objects and their relationships in the real world. The temporal graph generation model [5] is that defines the meaning of the vertices and edges, in the time axis according to the reality of the uninterrupted evolution process, according to certain rules to construct a temporal graph, which can solve the problem of temporal graph built in the real world. It is widely used in various practical scenarios and is an important basis and premise of data modeling and data analysis, such as compound evolution analysis, traffic model evolution analysis, and disaster monitoring evolution analysis. In a disaster monitoring analysis scenario, for example, we can express a vertex as a sensor, taking abnormal moment difference between sensors as edge generation or not, between the sensors will generate more than one edge determined by the properties with the time information. When users want to describe the relationship between the sensors, they can use the temporal graph generation model and build a sensor relationship graph between the sensors, forming a set of multiple temporal graphs. When the user wants to analyze the rule of events in a certain period, the clustering results of multiple temporal graphs can be quickly presented, which provides

The associate editor coordinating the review of this manuscript and approving it for publication was Zhigao Zheng.

accurate data reference for disaster analysis and subsequent disaster prevention. It is very effective for sensor relationship modeling and mining, which are critical methods in big data analysis.

Existing temporal graph models focus on graph streaming and graph generation model, in which the graph streaming model uses known changes in graph vertices and edges to construct and update the temporal graph [6], [7], [8], graph generation model is similar to it, which generates temporal graphs generally through the attributes of vertices or edges. For example, Purohit S. *et al.* [9] calculated the probability of the arrival of edges and vertices, and need know the sequence of the generation or disappearance of the edges and vertices, but the relevant temporal information is difficult to get directly in the real scene, so whether the emergence or disappearance of vertices and edges should be developed rules. For example, the graph generation model proposed by Bai Y. *et al.* [10] requires an additional calculation of the similarity between different objects in the video to construct the temporal graph, which is too costly and reduces the efficiency of subsequent data analysis for the scene of real-time construction of the temporal graph. Multiple graphs clustering work is mostly concentrated in the field of machine learning [11], [12], [13], most of the work involves the classification of the temporal graph, extracting the characteristic subgraphs of graphs or the characteristic matrix to learn. It's equivalent to clustering, but this way of classification to be separate training on each data set leading to these methods has some limitations, such as the training cost is high, the labels being difficult to obtain, and the model is difficult to update. The other method is to compare the similarity between two temporal graphs by extracting the graph core feature [14] for multiple graph clustering. Although this method can obtain the similarity of multiple graphs, this similarity does not consider time differences, only structural differences, which is not suitable for temporal multi-graph clustering.

Therefore, this paper proposes a Vertex-Driven Temporal Subgraph Clustering(**VDTSC**) Problem and a corresponding Vertex-Driven Temporal Subgraph Clustering method. Firstly, a basic temporal graph construction method is proposed, which determines the generation of inner vertices edges by setting two thresholds. Secondly, the construction process of the temporal graph is optimized using a sliding window. Then, the community detection method is used to extract communities from a static graph transforming from a temporal graph. The total distance between the two temporal graphs was determined by calculating the community comparing distance and the distance of time, and hierarchical clustering was used for clustering. Finally, the effectiveness of the method proposed in this paper was verified through experiments. Our main contributions are as follows.

- We propose a vertex-driven temporal graph construction algorithm and set two thresholds to divide the temporal graph on a timeline to obtain various temporal subgraphs.
- An enhancement strategy based on a sliding window is proposed to speed up the construction process and reduce the number of comparisons.
- We offer a double-standard temporal subgraph clustering method based on community comparison and temporal editing distance to realize the temporal subgraph clustering. The temporal subgraph can be distinguished in temporal and structure dimensions.
- Experimental results on 5 real and synthetic datasets show that the temporal graph model in this work can reduce the time overhead of construction compared with other existing models. The cluster method improves the clustering accuracy of temporal subgraphs. The clustering results can be shown directly through hierarchical clustering at the same time.

This paper is organized as follows. Section II introduces the current work in this direction. Section III illustrates definitions and VDTSC problems. Section IV constructs temporal graphs with two thresholds. Section V uses 3 steps to compute temporal subgraphs' distances and cluster them. Section VI compares our model and algorithms. Section VII summarizes this work.

II. RELATED WORKS

We introduce several aspects relevant to the work of this paper, streaming graph, community detection, community detection on the temporal graph, and multi-graph clustering, respectively.

1) STREAMING GRAPH

C. Linhares *et al.* [15] proposed a visualization method of time series graph to decompose large networks into small parts and classify community activity models better. Yin, Siwen *et al.* [16] proposed a novel incremental community detection method based on modularity optimization for node-grained streaming networks. This method takes one vertex and its connecting edges as a processing unit and equally treats edges involved by the same node, finding the temporal communities on the timeline. Ferrari, André *et al.* [17] devised an online change-point detection algorithm that fully benefits from the recent advances in graph signal processing to exploit the characteristics of the data that lie on irregular supports. Sariyüce, A. *et al.* [18] proposed a find-and-merge type of community detection algorithm that can efficiently handle streaming updates incorporating two additional techniques to speed up the incremental merge-min-hashing and inverted indexes. Y. Wu *et al.* [19] introduced a simple model for networks growing over time, which they refer to as the streaming stochastic block model (StSBM). Within this model, they proved that voting algorithms have fundamental limitations. Feng Sheng *et al.* [20], they use components as units, edges are added and deleted after vertex classification, and use a load balancing mechanism. However, this method does not have to scale with frequent updates on a large graph. Zhang Jianpeng *et al.* [21] proposed an appropriate streaming clustering model and designed two new core components:

streaming reservoir and a cluster manager, which handles the edge additions/deletions. They cluster vertices into clusters and do not consider multi-graph clustering.

2) COMMUNITY DETECTION

B. Das *et al.* [22] proposed an online community detection method based on user behavior. Pan Xiaohui *et al.* [23] composed by the similarity of adjacent nodes and then formed communities by connecting these small pieces. Ma Tinghuai *et al.* [24] proposed a community discovery method for static graphs based on global (k-shell entropy) and local information. Bilal Saoud *et al.* [25] proposed a method of small community detection merging based on local clustering. Fang Hu *et al.* [26] proposed a framework of vertex continuous feature representation and executed graph clustering by spectral clustering. You Xuemei *et al.* [27] finds communities through central vertex recognition, tag propagation, and community combination based on global and local information. Feng Zeng *et al.* [28] proposed a relationship between social properties and an interesting measurement vertex relationship and maintained the result of cluster clustering between vertices. Li Tianpeng *et al.* [29] proposed a method of dynamic temporal community discovery, which is like the view in this paper, based on the internal feature correlation of each snapshot. They both consider the structure of a graph but do not consider the multi-graph problem.

3) COMMUNITY DETECTION ON TEMPORAL GRAPH

Qin H. *et al.* [4] studied the problem of seeking periodic communities in a temporal network, where each edge is associated with a set of timestamps and proposed novel models, including σ-periodic k-core and σ-periodic k-clique, representing periodic communities in temporal networks. A. Hollocou *et al.* [30] found communities of graphs from an edge perspective. M.A.K. Patwary *et al.* [31] segments graph vertices online, ensuring load balance between partitions, but it is not suitable for constructing sequential graphs. An adaptive clustering algorithm in [21] is proposed to ensure the priority of graph stream data construction. M. Mariappan *et al.* [32] also proposed an adaptive data processing model based on a streaming graph, both of which consider edge creation and deletion. Feng Sheng *et al.* [33] proposed a technique of loss processing of incremental graphs to illustrate the optimization process of edge creation and deletion. Yao Junjie *et al.* [34] proposed a new approach to detect burst tagging events, which captures the relationships among a group of correlated tags where the tags are burst or associated with bursts tag co-occurrence. They both consider the streaming or graph data and not the construction perspective.

4) MULTI-GRAPH CLUSTERING

Current temporal graph clustering mainly focuses on vertex clustering, not the muti-graph clustering. Most of them focus on the multi-graph classification.Multi-graph classification, such as [13], [14] [11], [12]. Wu, Jia *et al.* [11] used the gspan method to mine the subgraph, which has a unique code, developed a bScore function to compute the similarity of each graph. They use a weak classifier to get the $(t+1)^{\text{th}}$ informative subgraph and two classifiers to adjust the final consequence. Wu Jia *et al.* [12]'s work is similar to [11]. Pang Jun *et al.* [13] mainly developed a frequent subgraph mining score function, calculates the top-K frequent subgraphs to get the feature subgraph on the multi-graph. They complete multi-graph clustering with a combination of the ELM classification model and MapReduce framework, which is also in the static graph clustering. Ma Guixiang *et al.* [14] used interior-node clustering and the multi-graph clustering, which can finally achieve a refined multi-graph clustering result. It studies the static brain networks. Wang Haishuai *et al.* [35] calculated the shaplet of each time vary graph and classified them by converting the time-varying graph into time-series data, but it counts the operation counts without computing temporal information. At present, there is still a lack of an efficient multi-temporal graph clustering method.

III. PRELIMINARY

In this section, we introduce the basic concepts and definitions of a vertex-driven temporal subgraph $G_t = (M, E, T)$ is introduced according to Example 1. They form a set of temporal graph denotes as $\mathcal{G} = (\mathcal{M}, \mathcal{E}, \mathcal{T})$, whereas $G_t \subseteq \mathcal{G}$. M and E just denote the temporal subgraph on the layer of notation, $\mathcal{T}$ contains temporal information. We call $\mathcal{G}$ vertex-driven temporal graph set.

Definition 1 (Vertex): The vertex set is denoted as $M = \{m_1, m_2, m_3, \ldots, m_l, \ldots | l \in [1, |M|]\}$, can easily infer $|M| \geq 1$, each vertex has an abnormal time instance $P_{m_i}, i \in M$, and each vertex can be connected or disconnected by temporal edges according to abnormal status.

Definition 2 (Edge): The relation between two vertices uses an edge to connect in graph, each edge is existence or not decided by the vertex's abnormal status and instance. For simply, vertex 1 has abnormal time P_{m_1}, and vertex 2 has abnormal time P_{m_2}, the time difference (edge time information) is $\Delta t = |P_{m_2} - P_{m_1}|$. We denote the edge set as $E = \{e_{ij} | i, j \in [0, |M|]\}$, and $\mathcal{T} = \{\Delta t_{ij} | i, j \in [0, |M|]\}$.

Definition 3 (Immediate Neighbor): The vertex A's immediate neighbor indicates a node set apart from node A, denotes the set of vertices directly connected to A. We use $N(A)$ to denote and use $N[A]$ to represent a node set including A and its immediate neighbor.

Definition 4 (Structure Similarity): The similarity of vertices uses the balanced performance method, called structure similarity [36], the computational formula is as follows.

$$sim_{ij} = (N[i] \cap N[j])/(\sqrt{|N[i]| \times |N[j]|}), \quad i, j \in M \quad (1)$$

Example 1: We obtain the set of abnormal instances corresponding to different sensors through the abnormal detection method when mining a tunnel in the mountain. The temporal graph can express these relationships to analyze digging from structure and time. Although a cluster of sensors [37] can obtain the classification sensors, it is the integrated

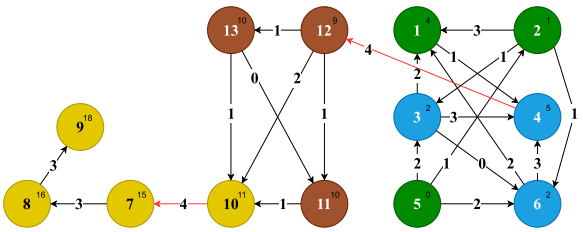


FIGURE 1. An Example of Temporal Directed Graph.

relationship of all sensors between each monitoring event. Therefore, there needs a temporal graph model in the process of analysis.

As shown in Figure 1, we can see that vertex 5 is the first becoming an anomalous state. Its immediate neighbors are vertices 2, 3, and 6, vertex 5's anomaly instances is 1, 2, 2, respectively. We set the local threshold to $t = 3$ and the global threshold to $\sigma = 20$, which can be determined based on the specific domain's prior knowledge. However, there are no edges between the vertices between 5 and 1, and between vertices 2 and 4. We assume that if the time difference between the two vertices is greater than the local threshold t, then the directed edge will not appear. When vertex 7 appears in this temporal graph, it does not meet the local threshold t. We will force connecting vertex 7 to 10 to ensure that the temporal graph is connected within the global threshold, similarly, vertex 4 and 12. We use the global threshold σ to facilitate the partition of the temporal graph, forming a temporal subgraph set, or forming an integrated temporal graph with t.

Definition 5 (Connectivity and Direct Connectivity): If $\exists m_i \in M$ where the vertices set M correspond to a temporal subgraph of $\mathcal{G}$, the first abnormal vertex is $m_{first} = arg_m min(P_M)$, so if the node m_i and m_{first} satisfy the following different conditions, respectively, we will get different conclusions.

Condition 1: $0 \leq P_{m_i} - P_{m_{first}} \leq \sigma$, all nodes in this range are connected containing m_i, denotes as $\gamma_n = \{m_i | 0 \leq P_{m_i} - P_{m_{first}} \leq \sigma, i \in [1, |\gamma_n|]\}$.

Condition 2: $0 \leq P_{m_i} - P_{m_{first}} \leq t$, we set m_i to be direct connected with each other, denotes as ${}_n\gamma_l^* = \{m_i | P_{m_i} - P_{m_{first}} \leq t, i \in [1, |\gamma_l^*|]\}$.

Condition 3: $t < P_{m_i} - P_{m_{first}} \leq \sigma$, we consider the node m_i and m_{first} are not connectivity, the first $m_i \in \gamma_{n+1}$ is the new initial point m'_{first} in the next temporal internal subgraph which indicates the node m_i is not contained in the current sliding window with length t.

Condition 4: $P_{m_i} - P_{m_{first}} \geq \sigma$, we separate m_i into a new temporal level keep away from the former temporal graph, which indicates the node m_i is not contained in the current sliding window with length σ.

When a new m_i emerge on the timeline, we have:

If condition 1 is satisfied, we consider that the m_i are in a connectivity state with the nodes in this range. If conditions 1 and 2 are met simultaneously, we think that they possess direct connectivity and will fully connect the nodes to m_i. If condition 4 is met, we consider that they are separated on the timeline and form a new temporal graph vertex set γ_{n+1}, the first node is m_i.

If condition 1 and condition 3 satisfied, we will force connect the latest vertex $m_{i_{last}}$ in γ_i^* and the earliest vertex $m_{j_{first}}$ in γ_j^*, that is to insert an edge $e_{\gamma_{i_{last}}^* \gamma_{j_{first}}^*}$ between them. So, the different temporal scope of Δt will connect together, and construct a connected temporal graph with length σ.

Recall Example 1, we set an extensive threshold of σ, rather than only a small threshold t to segment the temporal graph and further analyze it in different periods. If only one small threshold is used, all the temporal information will integrate into a large graph, which is not conducive to subsequent analysis, so the global threshold is used here. This reason can easily get in Example 2.

Definition 6 (Vertex-Driven Temporal Graph (Connected Graph)): When the number of nodes reaches $|\gamma| \geq \alpha, \alpha \geq 2$, and they all satisfy connectivity (Definition 5(1)), we consider that there form a temporal graph, they have properties as follows.

i. ${}_n\gamma_m^* \subseteq \gamma_n \subseteq \mathcal{M}, {}_n\mathcal{E}_m^* \subseteq \mathcal{E}_n \subseteq \mathcal{E}$, i.e. $G_{tn}^* \subseteq G_{tn}$.

ii. $\forall m_i \in \gamma_i$ and $\forall m_i \in \gamma_i^*$, they are all reachable, i.e., each node can reach another through a series of nodes and edges. We call this graph vertex-driven temporal graph, denotes as $G_{tn} = (\gamma_n, \mathcal{E}_n, \mathcal{T}_n)$, whereas $\gamma_n = \{{}_n\gamma_1^*, {}_n\gamma_2^*, \ldots, {}_n\gamma_l^*, \ldots\}$, $\mathcal{E}_n = \{{}_n\mathcal{E}_1^*, {}_n\mathcal{E}_2^*, \ldots, {}_n\mathcal{E}_l^*, \ldots\}$, $\mathcal{T}_n = \{{}_n\mathcal{T}_1^*, {}_n\mathcal{T}_2^*, \ldots, {}_n\mathcal{T}_l^*, \ldots\}$. The graph we obtained by combining the ${}_n\gamma_m^*$ and ${}_n\mathcal{E}_m$ is a vertex-driven temporal internal subgraph, which is denoted as $G_{tn}^* = ({}_n\gamma_l^*, {}_n\mathcal{E}_l^*, {}_n\mathcal{T}_l^*)$. These temporal graphs will form a temporal subgraph set denoted as $\mathcal{G} = (\mathcal{M}, \mathcal{E}, \mathcal{T}) = \{G_{t1}, G_{t2}, \ldots, G_{tn}, \ldots\}$, whereas $\mathcal{M} = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$, $\mathcal{E} = \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n\}$,$\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n\}$.

To briefly state, we use the terms "temporal graph" referring to a vertex-driven temporal graph, and "temporal subgraph," referring to a vertex-driven temporal subgraph. According to the definitions above, we obtain the property of a temporal graph as follows.

i. $P_{m_i} - P_{m_{first}} \geq \sigma$, $m_{first} \in \gamma_i, m_i \in \gamma_j$ or $m_{first} \in \gamma_j$, $m_i \in \gamma_i$

ii. $|\mathcal{M}| \geq |G_t| \times \alpha$

iii. $|\gamma_i| \geq 2 \wedge |\gamma_j| \geq 2$

iiii. $|{}_n\gamma_l^*| \leq |\gamma_n|$, with $m \in [1, |\gamma_n|]$

iiiii. $\forall m_i \in \gamma_i$ and $\forall m_j \in \gamma_j$ in the temporal graph set is unreachable.

Definition 7 (Vertex-Driven Temporal Subgraph Clustering(VDTSC) Problem): A user provides a query period $T_{u_i} \in [t_i, t_j]$, a temporal directed graph can be formally represented as $\mathcal{G} = (\mathcal{M}, \mathcal{E}, \mathcal{T})$, where $\mathcal{M}$ is the set of vertices, $\mathcal{E} \subset \mathcal{M} \times \mathcal{M}$ is the set of edges, and $\mathcal{T}$ is the temporal weight or temporal information that denotes the difference between a pair of nodes becoming abnormal status. Given a set of such graphs $\mathcal{G} = \{G_{t1}, G_{t2}, \ldots G_{ti}\}$, which construct a temporal graph set according to the above Definitions, they do not have

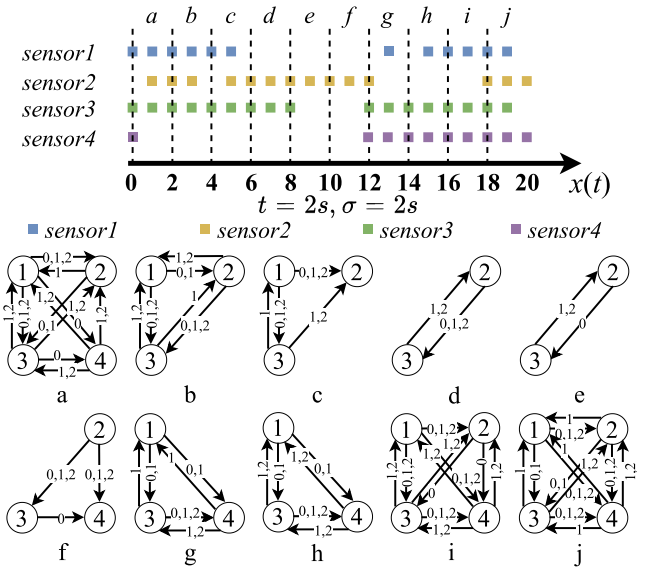


FIGURE 2. An example of temporal subgraphs clustering.

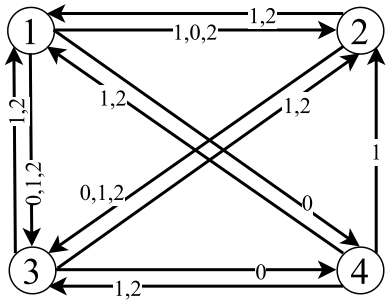


FIGURE 3. A temporal graph generates under the condition of $t = 4s$.

TABLE 1. Symbolic meanings.

Symbol	Meaning
$G_t = (M, E, T)$	Temporal subgraph.
$\mathcal{G} = (\mathcal{M}, \mathcal{E}, \mathcal{T})$	The temporal graph; $\mathcal{M}$ and $\mathcal{E}$ are the vertex set and edge set in temporal graph set. $\mathcal{T}$ provides temporal information.
$M = \{m_l \mid l \in [1, \lvert M\rvert]\}$	vertex set of G_t.
$E = \{e_{ij} \mid i, j \in [0, \lvert M\rvert\}$	edge set of G_t.
$\mathcal{T} = \{\mathbf{t}_{ij} \mid i, j \in [0, \lvert M\rvert]\}$	edge's time information set of G_t.
P_{m_i}	A vertex m_i's abnormal time instance.
$N[A]$, $N(A)$	A's immediate neighbor and A; immediate neighbor of A.
sim_{ij}	The similarity between vertex i and j.
σ	The global threshold means the graph segmented by it.
t	The local threshold, which means that the vertices fully connect in it.
h	The distance threshold of the hierarchical cluster after z-normalization.
Δt	The time difference $\lvert P_{m_i} - P_{m_j}\rvert$ between the two vertex's abnormal time instance.
$\gamma^* \subseteq \gamma$	The node set of one temporal internal subgraph; The node set is one temporal graph.
$\mathcal{E}_i^* \subseteq \mathcal{E}_i$	The edge set of one temporal subgraph; The edge set of one temporal graph.
α	The number constraints of temporal graph generating.
s_i	A vertex sequence of one temporal graph.
$distance_{ij}, d_e, d_d$	Distance between G_{ti} and G_{tj}; community comparing distance; temporal editing distance.
$o_i \subseteq O_i$	A community of one temporal graph; all communities of one temporal graph.
$\mathcal{F}_{m_i}$	The map function of vertex m_i and m_j between o_i and o_j.
$\mathcal{D}_{e_{ij}}$	The time difference of the temporal subgraph's edge sets $\mathcal{E}_i^*$ and $\mathcal{E}_j^*$.
$\mathbf{C} = \{c_1, c_2, \ldots, c_n\}$	The cluster consequence includes multiple temporal subgraphs in each c_i.

any properties themselves, so we must compute the distance $distance_{ij}$ between two temporal graphs and using the hierarchical clustering method, the goal of multi-graph clustering is to cluster the graphs in $\mathcal{G}$ into $C = \{c_1, c_2, \ldots, c_n\}$ set.

Example 2: There are 4 sensors on the timeline. Their status changes from normal to abnormal frequently. As shown in Figure 2, there are many dots with different colors denoting the different sensors. The axis denotes the timeline, which indicates that these sensors turn into the abnormal status at different time instances in seconds, and the *sequence*1 - *sequence*4 will together form a temporal graph set through our graph model, which is shown at the bottom of this Figure. The dashes denote the partition of the temporal graph, which is determined by σ. We set $\sigma = t = 2s$, which will merge the first four temporal graphs when $\sigma = 8s$, but they cannot form a new structure of a temporal graph. We sort them in ascending order according to vertex series numbers for some abnormal nodes simultaneously. These temporal graphs form a new set as $\mathcal{G}$. And their clustering denotes as $C = \{\{A, I, J\}, \{B, C, D, E, F, G, H\}\}$. We will illustrate clustering in section V-D. This example illustrates that the purpose of we set a threshold to segment a temporal graph into a series of temporal subgraphs to differentiate them from $\sigma = 10s$,$t = 2s$ situation shown in Figure 3, which generates at $t = 4s$ and does not vary in the period of $4s$-$10s$. The operation is boring for a little bigger σ.

We can easily find the different periods corresponding to a different temporal graph. The distance among them will help us cluster and seek the rule of temporal graphs to avoid accidents. The table 1 shows the meaning of all symbols.

IV. VERTEX DRIVEN TEMPORAL GRAPH MODEL

The temporal graph model should be a model that has two procedures of construction and partition on the timeline. Some vertices and edges update omit, unavailable, and unreasonable if we use the snapshot strategy [5]. It would also be impractical to update a temporal graph with every vertex and edge updated with one vertex at frequent update mode.

So we set two thresholds (see section III Definition 5) to segment the temporal graph on the timeline and call this model the vertex-driven temporal graph model. Its primary purpose is to construct a set of temporal subgraphs $\mathcal{G}$.

A. VERTEX-DRIVEN TEMPORAL GRAPH CONSTRUCTION

We construct the temporal graph $\mathcal{G}$ according to the abnormal time instance of vertices, recalling the Definition 1, Definition 2, Definition 5 and Definition 6 from the above.

The abnormal instance determines whether some edges exist or not between different vertices. The main thought of temporal graph construction is to wait for the abnormal vertex. We insert edges following the constraints, connect the vertices in the time sequence. The edge e_{ij} is determined by the time vertex arrival and created one by one abiding by the thresholds t and σ. We will count the abnormal frequency of the same vertex, when it becomes abnormal status from normal between σ and determine the importance of this vertex. The vertex will be connected all the time according to the threshold t when $\sigma = 0$. The pseudo-code of this method is shown in Algorithm 1.

Algorithm 1 Temporal Graph Construction(TGC)

Require: M, t, edge construction threshold $\sigma, t, T_{u_i} \in [t_i, t_j]$
Ensure: $\mathcal{G} = \{G_{t1}, G_{t2}, \ldots, G_{tn}\}$
1: *startPoint* $\leftarrow \emptyset$,*queue* $\leftarrow \emptyset$,*nodeList* $\leftarrow \emptyset$,*existEdge* $=$ *False*
2: **while** True **do**
3: **if** m_{new} is motivated and falls in the time interval T_{u_i} **then**
4: *startPoint* $\leftarrow m_{new}$ if the procedure execute first circle.
5: *nodeList*.*insert*(m_{new})
6: $d \leftarrow P_{m_{new}} - P_{startPoint}$
7: **if** $d > \sigma$ and $\sigma \neq 0$ **then**
8: $\mathcal{G} = \mathcal{G} \cup G_{ti}$, G_{ti} construction according to *nodeList*
9: *startPoint* $\leftarrow m_{new}$
10: *nodeList* $\leftarrow \emptyset$
11: *continue*
12: **else**
13: **if** $d \leq \sigma$ and $\sigma \neq 0$ **then**
14: **for** each node i in *nodeList* **do**
15: **if** $|P_{m_{new}-P_i}| \leq t$ **then**
16: $i \rightarrow m_{new}$,$e_{i,m_{new}} = d$
17: *existEdge* $=$ *True*
18: **end if**
19: **end for**
20: **if** *existEdge* $==$ *False* **then**
21: *nodeList*.*last* $\rightarrow m_{new}$,$e_{nodeList.last,m_{new}} = d$
22: **end if**
23: *existEdge* $=$ *False*
24: **end if**
25: **if** $\sigma == 0$ **then**
26: The same as line 14-24.
27: **end if**
28: **end if**
29: **if** m_{new} not in *queue* **then**
30: $nodeList_{m_{new}}.count++$
31: **end if**
32: **end if**
33: **end while**
34: **return** $\mathcal{G}$

Line 1 initials the variable. Line 2 monitors the situation of vertex abnormal status through an endless loop. It entrances the edge construction process according to a new vertex that occurs at abnormal status at line 3. Line 4 confirms the first vertex with abnormal status, and lines 5-6 compute the time difference d of *startPoint* and a new abnormal vertex m_{mew} in each iteration, where *nodeList* stores the frequency and node set of the range of σ period, we will save to form a temporal graph and clear it after G_{ti} construction. Line 8 merges the current G_{ti} into the $\mathcal{G}$. Lines 9-10, re-initialize *startPoint* and *nodeList* ready to save the next temporal graph. If $d \leq \sigma$ and $\sigma \neq 0$, then this node will locate in σ period at line 13. At lines 28-31, we statistic the number of each vertex and save them into the current *nodeList* between σ when they appear more than once. Lines 14-19 traverse each node in *nodeList*, find the nodes that satisfy the condition $\Delta t < t$ and construct the edge between i and m_{new} with weight d under constraint Definition 5 (2). We should connect the *nodeList*.*last* in the last G^*_{ti} and m_{new} in $G^*_{t(i+1)}$ that construct a connected G_{tn} with Definition 5 (4). We construct G_t through the above method one by one on the timeline. At line 25, we obtain an integrated temporal graph with $\sigma = 0$. The algorithm returns the $\mathcal{G}$ at line 34.

Complexity Analysis: The vertex set is $\mathcal{M}$ be allocated by ourself, and the time complexity is $|\mathcal{M}| \times (|\mathcal{M}| - 1)$, the space complexity is $|\mathcal{E}| + |\mathcal{M}| \times (|\mathcal{M}| - 1)$.

B. SLIDING WINDOW ENHANCED

Though we can get temporal graphs $\mathcal{G}$ in section IV-A, we can also optimize the construction process according to the sliding window as IV-B.

Corollary 1 (Period Partition): We set the run-up time $startTime = P_{Sfirst}$ and the end time $startTime + \sigma$ as shown in Figure 4, the current period is $[P_i - t, P_i]$. Its temporal subgraph vertex set is *PSET*, the next period is determined by $[P_{i+1} - t, P_{i+1}]$, when the next abnormal vertex emerges. There have $NSET = PSET \backslash \Delta k \cup m_{i+1}$, where Δk denotes a period in vertices definitely does not connect with the current vertex in it. Therefore, the algorithm only eliminates vertices from *PSET*, and diagnoses each node in *NSET* each time, which can reduce the computation cost.

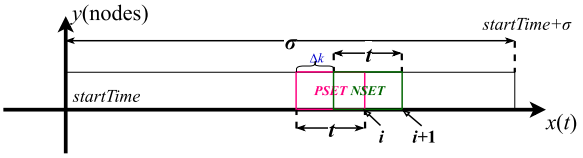


FIGURE 4. Time period partition illustration.

We will not compare with all nodes from the start time of one temporal graph in the generation process. However, constructing edges to the vertex set is shown in Figure 4(time difference $\Delta t < t$) to reduce the complexity of online temporal graph construction according to IV-B. This method's pseudo-code is shown in Algorithm 2.

Line 1 initializes the variables, here, we delete the *existEdge* and add another variable *isFirst* we set as a flag to show *PSET* is built accomplish or not in this method. Lines 3-10 are the same as the *TGC* so we omit this code paragraph. Lines $12 - 14$ illustrate *PSET* and G_{t1} construction, line 16 sets the *isFirst* to *False* so that the next iteration cannot execution code segment 11-14. Line 16 adds a node to Δk that satisfies the distance from nodes to the first node in *PSET* smaller than the distance from the m_{new} to the last motivating node in *PSET*. Lines 18-21 denote the motivation of m_{new} is greater than t when the $|\Delta k|$ equals to the $|PSET|$, then we connect the *PSET*.*last* to m_{new} with weight d to promise

Algorithm 2 Fast Temporal Graph Construction(FTGC)

Require: M, t, edge construction threshold σ, t, $T_{u_i} \in [t_i, t_j]$
Ensure: $\mathcal{G} = \{G_{t1}, G_{t2}, \ldots, G_{tn}\}$
1: $startPoint \leftarrow \emptyset$,$queue \leftarrow \emptyset$,$isFirst = True$, $PSET \leftarrow \emptyset$, $NEST \leftarrow \emptyset$
2: **while** True **do**
3: **if** m_{new} is motivated and falls in the time interval T_{u_i} **then**
4: $startPoint \leftarrow m_{new}$ if the procedure execute first circle.
5: $nodeList.insert(m_{new})$
6: $d \leftarrow P_{m_{new}} - P_{startPoint}$
7: **if** $d > \sigma$ and $\sigma \neq 0$ **then**
8: It's the same to the Algorithm 1 except for the line 10, we set $isFirst = False$ extra.
9: **else**
10: **if** $d \leq \sigma$ and $\sigma \neq 0$ **then**
11: **if** $isFirst$ **then**
12: **if** $P_{m_{new}} - P_{startPoint} \leq t$ **then**
13: Construct $PSET$
14: connect all vertices emerge in the first $PSET$.
15: **else**
16: $isFirst = False$
17: get Δt according to IV-BIV-B
18: **if** $|\Delta k| == |PSET|$ **then**
19: $nodeList.last \rightarrow m_{new}$
20: $e_{nodeList.last,m_{new}} = d$, $queue \leftarrow m_{new}$
21: **end if**
22: **for** each node i in $PSET \backslash \Delta k$ **do**
23: $NSET.insert(m_{new})$
24: $i \rightarrow m_{new}$;$e_{i,m_{new}} = d$, $queue \leftarrow m_{new}$
25: **end for**
26: $PSET = NSET$
27: **end if**
28: **else**
29: The same as line 14 to 23 in Algorithm 1.
30: **end if**
31: **end if**
32: **end if**
33: **if** $\sigma == 0$ **then**
34: The same as line 11 to 30 in Algorithm 2.
35: **end if**
36: **if** m_{new} in $queue$ **then**
37: $nodeList_{m_{new}}.count++$
38: **end if**
39: **end if**
40: **end while**
41: **return** $\mathcal{G}$

connectivity, and push m_{new} into the *queue*. Lines 21-24 connect each node in $PSET \backslash \Delta k$ to the m_{new} with distance d and push m_{new} into the *queue*. We reset $PSET$ to the current variable value $NSET$ at line 26. And line 29 ensures the $PSET$ and $NSET$ updating in the subsequent process. Lines 34 is the same as line 10 to 29 to construct an integrated temporal graph. Line 36-38 statistic the node emerging in the σ period. Line 41 returns the $\mathcal{G}$.

Complexity Analysis: We do not consider the while circle participating time complexity computation as well, the time complexity of Algorithm 2 is decided by length of the first $|PSET|$, and the others decided by the remaining length of $PSET$ in each period t that decided by motivated frequency of these nodes. So the total time complexity is $|first(PSET)| + |PSET| \times ((t_j - t_i)/\sigma)$. The space complexity is consist of $\mathcal{G}$, G_t, $PSET$, and $NSET$ in each iteration, and $nodeList$ in each G_{ti}, and the total space complexity is $|\mathcal{M}| + |\gamma_n| + (|PSET| + |NSET|) + |nodeList|$.

The algorithm above will construct the temporal graph in real-time. The edge between two vertices is the time difference, which directly represents the temporal relationship between two points at the period of $[t_i, t_j]$. The temporal edge can judge the similarities and differences between two temporal subgraphs when they have a similar structure. Although we can find the difference between temporal graphs through the combination of structure matching algorithm and time information, it requires high computational overhead, and a large amount of temporal subgraph matching is not suitable for the traditional subgraph matching algorithm [38], so community detection algorithm is used in this work as a compromise.

V. VERTEX-DRIVEN TEMPORAL SUBGRAPHS CLUSTERING

The above algorithm will construct the temporal graph in real-time with period $[t_i, t_j]$. We can distinguish a temporal graph with the same time relationship through the temporal edge. In contrast, the graph structure matching methods can find their differences. Still, it requires high time complexity and many temporal graph distance computation tasks. It is unsuitable for a large amount of subgraph matching, so we decide to use community detection as a compromise. This step enhances the morphology discrimination and prepares a temporal graph set for subsequent clustering.

A. VERTEX COMPUTATION TREE

This section will change the temporal graph to a static graph to conduct community detection to find the structure of the temporal graph. So we propose two notations of the graph: local impact and overlap level before entering the community detection. Hit counts can decide the local effect of a vertex in its range of two-hop neighbors, and we define the hit counts as Definition 8.

We erase the temporal information of vertices and edges. If there is a one-way edge between two vertices in the current graph, which will change into an undirected edge, do the same operation for the bi-edge, and the whole graph becomes a static graph.

Definition 8 (Hit Counts): The range of two-hop neighbors and the current vertex is the entire scope, which the current vertex is considered the center we count. Naturally, the hit count of vertex i is computed by the equation 2.

$$\begin{aligned} HIT(i) &= \{hit_1, hit_2, \ldots, hit_n\}, \quad n \in [1, |V|], \\ hit_i &= |twoHop(i)|, \\ twoHop(i) &= N[i] \cup N[N[i]], \quad i \in [1, |V|] \end{aligned} \tag{2}$$

where the $|twoHop(i)|$ denotes the occurrence number of i in the two-hop range, the hit_i is a list of all vertices that exist with vertex i. We devise a self-adaptive function to decide the vertex is reserving or not. The function of self-adaptive is shown in equation 3.

$$hit_i^{ad} = \frac{\Sigma_{j=1}^{|N[N[i]]|} hit_i - max(HIT(i)) - min(HIT(i))}{|HIT(i)| - 2} \tag{3}$$

A vertex remained or not can be decided by the hit_i^{ad} when the hit_i of nodes in $\mathcal{G}$ is greater than this value or not through

the piecewise function as equation 4.

$$f(x) = \begin{cases} 0, & \text{if } hit_i > hit_i^{ad} \\ 1, & \text{otherwise.} \end{cases} \tag{4}$$

The hit counts can represent the impact of the current node, but it only decides the current node is the representation between two-hop and cannot determine the classification with other vertices. Therefore, we should consider the overlap notation of a vertex, which can decide to merge them. The definition of overlap level denotes as Definition 9.

Definition 9 (Overlap Level): The number of intersections between the current node i and each neighboring node in two hops can show their correlation, which requires a threshold corresponding to the current node in the two-hop neighbor set. The ratio of overlap vertices will be used to decide if they merge not. First, one-hop neighbors of a vertex n_i denotes as $N[n_i] = \{n_i, N(n_i)\}$, so we define the overlap level threshold is as equation 5.

$$\begin{aligned} W_i &= \{w_{i1}, w_{i2}, \ldots, w_{ij}\} \\ w_{ij} &= \frac{N[n_i] \cap N[n_j]}{min(|N[n_j]|, |N[n_j]|)}, \quad j \in [1, |N(i)|] \end{aligned} \tag{5}$$

We adopt the same method of adaptive hit counts $hit_{m_i}^{ad}$ computation and define the adaptive overlap level w_i^{ad} as equation 6.

$$w_i^{ad} = \frac{\Sigma_{j=1}^{|N(i)|} w_{ij} - max(W_i) - min(W_i)}{|w_i| - 2} \tag{6}$$

The same as hit counts, the piecewise function will select the vertex to determine whether it will become the candidate set through $f(x) = 1$ selected or $f(x) = 0$ unchosen in the two-hop range.

$$f(x) = \begin{cases} 0, & \text{if } w_i > w_i^{ad} \\ 1, & \text{otherwise.} \end{cases} \tag{7}$$

We construct two sets-hit candidate set and overlap candidate set denotes as *hitCandidate* and *wCandidate* according to the Definition 8 and Definition 9, respectively.

Definition 10 (Vertex Computation Tree): We will construct a vertex computation tree to speed up the process of the solution procedure while the algorithm is executing for participating vertex $i \in G_{ti}$. The computation tree divided into two hierarchical layers, the neighboring set of the current vertex i forms the first layer denoted as $N(i) = \{1, 2, 3, \ldots, j, \ldots\}$, the neighboring of the current vertex i's neighbor set forms the second layer denoted as $N(N(i)) = \{1, 2, 3, \ldots, k, \ldots\}$, each edge denotes the hit counts($|hitCandidate|$) of two vertices vertex i and vertex j between the first layer and the second layer, whereas i and j denote as two vertices, respectively.

The common neighbor $CN(i, j) + 2$ is the hit counts. We simply compute the $|N(i)|$ will hit i only once. And i will hit itself. So $hit_i = i.degree + 1$, and $\forall j \in N(i)$, hit_j determined by the common neighbor among i and j since others cannot hit j, plus i and j itself. So $hit_j = CN(i, j) + 2$. The third layer is the rest of the non-repeating vertex because we removed the repeat elements from the first layer, so the $hit_k = count(k)$.

The hit count of current node i is $hit_i = i.degree + 1$, the second layer of each node is set to $hit_j = CN(i, j) + 2$, also, as the edge between the root and its children as well. And we also consider the counts during the construction process, see section IV. So the hit counts will become $(i.degree + 1) \times i.count$ and $hit_j = (CN(i, j) + 2) \times j.count$. The third layer is $hit_k = count(k) \times k.count$ apart from the elements emerging in the second layer. The emerged counts of the vertex k must be 1 in each node in the second layer after eliminating the repeated elements in the second layer, and each element k in the third layer hit_k is computed by summation all the same elements in the third layer. So we can easily deduct that the edge between the third and second layers must be 1. The overlap level between node i and node j denotes as $w_{ij} = CN(i, j)/CN(i, j) + count(k)$.

B. VERTEX COMPUTATION TREE UPDATE

We set the corresponding tree to null when a computation tree's root vertex i is classified, or we set the second layer node j's subtree to null if the node in the second layer is classified as well and set the corresponding vertex's edge weight to 1. Its child node is excluded from the number of a hit anymore, and $w_{ij} = null$, not calculated, too. An update will happen between the two-hop neighboring set when some visited nodes in the vertex are visited.

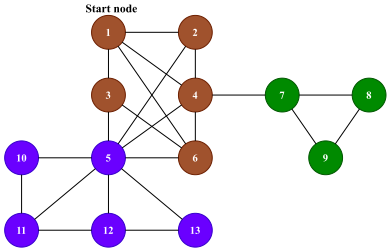
The construction of a vertex computation tree can speed up the community detection process, and the pseudo-code is shown as Algorithm 3 as follows.

Algorithm 3 Computing Tree Construction(**CTC**)

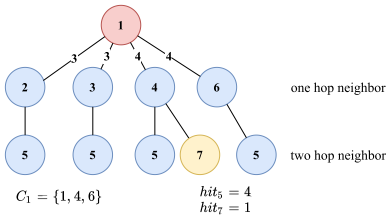
Require: M, a node i need to compute
Ensure: The tree of node i
1: Set the current node i as the root node of computation tree.
2: **for** each node j in $N(i)$ **do**
3: $CN(i, j) \leftarrow$ compute the common neighbors between i and j
4: Insert a child between i and j in the tree named j, and its edge is set to the $(CN(i, j) + 2) \times j.count$
5: Insert $N[j] \backslash N[j] \cap N[i]$ set in the tree named by its node's name, and do not compute any value between j and k.
6: **end for**
7: **return** The computing tree of node i

Line 1 sets the vertex i as the root node of the i computation tree. Lines 2-6 compute the number of common neighbors of each node in the immediate neighbor $N(i)$, then we insert a new vertex with the weight $(CN(i, j) + 2) \times j.count$ as a child. Insert a vertex that does not emerge in the first layer of the computation tree and does not have any weight between the third and second layers. The algorithm is over when we traverse all vertices between two-hop neighbors of the current node i.

Example 3: We give a static graph example to help understand. As shown in Figure 5, we will compute the red vertex first. The blue node denotes the one-hop neighbor set, including several vertices described by Definition 3, the yellow nodes denote the two-hop neighbor set of the current node.



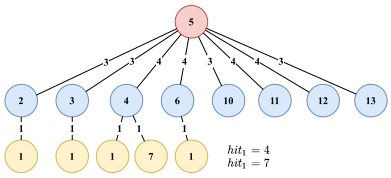
(a) The whole static graph.



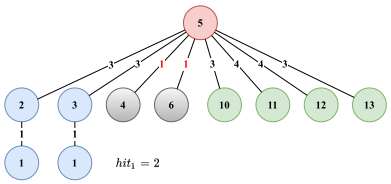
(b) Computation tree of vertex 1.

1:2,3,4,6——len(1)=5
2:5——len(2)=3+1=4——w=3/4=0.75
3:5——len(3)=3+1=4——w=3/4=0.75
4:5,7——len(4)=4+2=6——w=4/5=0.8
6:5——len(6)=4+1=5——w=4/5=0.8
w_1^{ad}=(0.8+0.75)/2=0.775
146 falls into 1 category

(c) The first consequence.



(d) Before computation tree of vertex 5.



(e) Updated computation tree of vertex 5 after vertex was allocated.

5:2,3,4,6,10,11,12,13——len(5)=9
2:1——len(2)=3+1=4——w=3/4=0.75
3:1——len(3)=3+1=4——w=3/4=0.75
4:null
6:null
10:coincide——len(10)=3——w=3/3=1
11:coincide——len(11)=4——w=4/4=1
12:coincide——len(12)=4——w=4/4=1
13:coincide——len(13)=3——w=3/3=1
w_5^{ad}=(0.75+1+1+1)/4=3.75/4=0.9375
5,10,11,12,13 fall into 1 category.

(f) The second consequence.

FIGURE 5. **An toy example for community detection on static graph.**

We reject the repeated nodes that emerge in the one-hop nodes' neighbor set to eliminate the redundancy and do not list the current node when we construct the computation tree. We compute the hit counts and overlap level through the edge without node count.

For $N(1) = \{2, 3, 4, 6\}$, and $N(2) = \{4, 5\}$, we remove the repeated elements by comparing with vertex 1, get the final list as shown in Figure 5 (c) according to the rules we propose of the above. And the root is vertex 1, and its child is 2, 3, 4, 6, respectively. The third layer is 5, 5, 5, 7, 5, we compute the overlap level according to the equation 6 in V-A Definition 9, the first cluster will be $c_1 = \{1, 4, 6\}$. We suppose the vertex 5 is the next node that should compute as a center. And the computation tree of vertex 5 will become Figure 5 (e) from (d) according to the updated strategy described above since the range of two-hop of node 5 has node 1 visited before.

C. DE-TEMPORAL GRAPH COMMUNITY DETECTION

We can quickly get the hit candidate set and overlap the candidate set and compute them are simple through computation tree. We will introduce the community detection algorithm below, which is sensitive to the dense edge vertex and detects communities in a de-temporal graph. This algorithm only sets a threshold called τ will be introduced in Algorithm 5 to merge the vertices according to τ. The pseudo-code is shown in Algorithm 4.

Algorithm 4 Graph Community Detection(**GCD**)

Require: M, τ,at least one De-temporal direct graph G_{ti}
Ensure: Community of G
1: $nodeList \leftarrow k_{shell} \leftarrow getRank(G_n.nodes())$
2: $diagFirst \leftarrow False, q \leftarrow \emptyset$
3: **for** each node i in $nodeList$ **do**
4: **if** $i.degree > 1 \wedge i.visit == False$ **then**
5: $q \leftarrow i, i.stacked == True$
6: **while** $q! = null$ **do**
7: $current \leftarrow q.pop$
8: **if** $current.degree > 1 \wedge current.visit == False$ **then**
9: **if** $diagFirst \leftarrow DND(G_n current, t) == True$ **then**
10: Return
11: **end if**
12: $computationTree \leftarrow CTC(G_n, current)$
13: $hitCandidate \leftarrow$ get the hit number according to Definition 8.
14: **if** $|hitCandidate| < 3$ **then**
15: $current.cluster \leftarrow$ its neighbor's cluster series with τ.
16: **else**
17: $wCandidate \leftarrow$ extract nodes according to Definition 9.
18: $interaction \leftarrow hitCandidate \cap wCandidate$
19: **if** $interact.length() < 3$ **then**
20: $current.cluster \leftarrow N(current)$'s series with τ.
21: **else**
22: $maxCluster \leftarrow$The maximum number of clusters.
23: **if** $interact.length \geq |maxCluster|$ **then**
24: $interact$ form a new cluster and cluster series.
25: **else**
26: $current.cluster \leftarrow maxCluster$
27: **end if**
28: **end if**
29: **end if**
30: **end if**
31: **if** $current.cluster! = -1$ **then**
32: $current.visit = True$
33: **PushtoQueue**($computationTree, G, q, \tau, current, interaction$)
34: **end if**
35: **end while**
36: **end if**
37: **end for**
38: Tidy nodes not allocated, according to their neighbors' cluster.
39: **return** Community of G

Lines 1-2 initialize the variables that the algorithm needs. In 2011, Kitsak *et al.* proposed the K-shell method [39].

The algorithm considers the nodes of a network hierarchical. The higher the k-shell layer of a node, the more likely it is to be the core of the network. Lines 3-4 will circle in the list of nodes whose degree is greater than 1 and have not been visited before in the list. Line 5 pushes into the queue q that prepares to process subsequently. Line 8 controls the requirements of the current node i. Lines 9-11, the algorithm diagnoses whether the node is divided into a cluster or not, and decides the algorithm is too early to give up. Lines 12-13 generate a computation tree of vertex i and compute the hit candidate set according to Definition 8. Line 14, the current node will be classified by its neighbors' cluster series if the number of hit candidates set smaller than 3, skips subsequent calculations. τ is a value that computes through $|allocatedvertices|/N(current)$. Lines 17-18 continue to compute the *wCandidate* set, and get the interaction between *hitCandidate* and *wCandidate*. Line 19 are the same as line 14 when $|interact| \leq 3$. When $|interact| \geq 3$, we seek the maximum cluster of the current node's immediate neighbor set. If $|maxCluster| > |interact|$, then we set all vertices in *interact* to the cluster series of *maxCluster*, or we set the nodes in *interact* to a new cluster series at lines 22-27. Lines 31-34 sort the vertices in the two-hop range of the current node which has not been visited, and push them to a queue according to a priority rule we define, see algorithm 6. We cluster the nodes which are not clustered in the iteration for the specific graph structure based by the principle of the majority rule when their neighbor is not allocated or create a new cluster for each node and form clusters that have $|\gamma_i|$ at line 32. The algorithm returns a community of G_{tn} at line 39.

Complexity Analysis: The time complexity is $|G_{tn}| \times |q| \times O(CTC) + O(DND) + O(PushtoQueue)$, and space complexity is $|q| \times (|hitCandidate| + |wCandidate| + |interact| + |computationTree| + |maxCluster|)$.

The affiliation we diagnose first when a new vertex is traversed. Suppose the vertex is classified as a cluster. In that case, we will not compute the remaining vertices and divide them directly to the cluster and call this solution the *DND* method, the threshold τ represented illustrated above here. The pseudo-code is shown in Algorithm 5.

Algorithm 5 Diagnose Node Division(**DND**))

Require: G, $current$, τ
Ensure: updated queue
1: $cluster \leftarrow$ get all nodes have been allocated to one cluster.
2: $clusterFD \leftarrow$ sorted the cluster according to the frequency of emerging.
3: $maxCluster \leftarrow$ get the node emerges most.
4: **if** $|cluster| > 0 \wedge \frac{|cluster|}{|N(current)|} > \tau$ **then**
5: $current.cluster \leftarrow maxCluster$
6: **return** true
7: **end if**
8: **return** false

When the procedure pushes the remaining vertices into the queue q, we should consider a sort strategy to move these vertices to the queue in a specific order or affect the subsequent results a lot. We use the total similarity of the current node with the whole graph to divide the number of allocated vertices, which can get the average value of the entire impact of the current node acting on the allocated nodes for the effect of the current node is its compactness to the entire graph. The larger the value is, the smaller the number of allocated vertices is, which shows that the existing vertices are less likely to be integrated. It will be assigned separately. The weight of each edge can be a time difference or other meanings. There, we use the structure similarity [36] to denote our temporal graph more precisely, see Definition 4.

Algorithm 6 Push to Queue (**PQ**)

Require: $computationTree$, G, q, τ, $current$, $interaction$
Ensure: updated queue
1: $priority \leftarrow \emptyset$,$FD \leftarrow$ transfer the $computationTree$ to FD array and delete unqualified nodes.
2: **for** each i in FD **do**
3: **if** i in interaction **then**
4: delete i
5: **end if**
6: **end for**
7: **for** each i in FD **do**
8: signed=0;unsigned=0
9: **for** each j in N(i) **do**
10: sim=compute the similarity between i and j
11: **if** $j.visit == True$ **then**
12: $signed++$
13: $allSim += sim$
14: **end if**
15: **end for**
16: **if** $allSim == 0$ or $signed == 0$ **then**
17: $priority[i] = 0$
18: **else**
19: $priority[i] = allSim/signed$
20: **if** $priority[i] == 0$ **then**
21: delete $priority[i]$
22: **end if**
23: **end if**
24: **end for**
25: $prioritySorted$ = sorted priority by ASC.
26: **for** each i in $prioritySorted$ **do**
27: **if** $i.degree > 1$ and $i.visit == False$ and $i.stack == False$ **then**
28: $q.put(i)$
29: $i.stack = True$
30: **end if**
31: **end for**
32: **return** Updated queue

So we can sort this value in descending order to optimize the order in which the graphs are assigned, called the enqueue optimization strategy. The pseudo-code is shown as Algorithm 6.

Line 1 initializes variables, lines 2-6 delete the nodes in the *interact* since they are visited and clustered to a cluster, see algorithm 4. Line 8 initializes the statistical variables. Lines 9-15 statistic the number of allocated node *signed* and the strength of the connection with the graph *allSim* (the total similarity of all neighbors) of each node in *FD* after the delete operation. Lines $16-17$ adjust the priority of nodes to 0 which $allSim = 0$ or $signed = 0$. Line 19 apportions the similarity *allSim* to each allocated node equally. Lines 20-21 delete the node whose priority is 0. Line 25 sorts all vertices as ASC. Lines 26-31 will push the nodes not visited before, not included in the queue before, and the degree greater than 1. The algorithm returns the updated queue to promise the order visit.

Complexity Analysis: The total time complexity is $FD + FD \times |N(i)| + |priority|$, the total space complexity is $|FD| + |N(i)| \times |FD| + |priority| + l$.

There exists a problem that each temporal graph has significant differences at different periods. The temporal graph constructed may be much smaller than the original graph, the consequences of community detection also have a tremendous difference, so we define an equation to describe the relationship among different temporal graph as Definition 12 to describe the temporal graph.

D. DISTANCE COMPUTATION AND HIERARCHICAL CLUSTERING

We cluster these G_t using a hierarchical clustering algorithm with our own defined distance, which transforms the whole temporal graph into a chain following the time sequence and is defined as follows.

Definition 11 (Temporal Graph Chain): G_{ti} can be transformed to a form of $s_i = \{m_1 \to m_2 \to \cdots \to m_i \to \ldots\}$, $i \in [1, |\mathcal{E}_i|]$, which ignores the other edges in G_{ti} aiming to simplify the distance computation process, s.t. $|s_i| \geq |\gamma_i|$, $P_{m_j} - P_{m_i} \geq 0$, or $P_{m_j} - P_{m_i} = 0, i < j$. The vertex emerges not only once, following the abnormal order and series number.

Definition 12 (Temporal Graph Distance): There are two sequences of temporal graphs on G_{ti}, G_{tj}, we consider not only the community comparing distance but also, consider the distance of time, we define it as equation 8.

$$distance_{ij} = (d_e + d_d)/2 \tag{8}$$

The d_e denotes the community comparing distance from the community detection consequence. The character comparison will count the corresponding position letter one by one, taking the communities as a unit according to the vertex's abnormal status sequence. If they are different in the corresponding position, then add 1. Otherwise, we do not execute any operation and arrange them with time sequence, ignoring other edges as equation 9.

$$d_e = \Sigma^{max(|O_i|,|O_j|)} \Sigma_{n=1}^{max(|o_i|,|o_j|)} |\mathcal{F}_{m_n}| \tag{9}$$

whereas o_i and o_j denote two communities in G_{ti} and G_{tj}, respectively, O_i and O_j denote all communities in G_{ti} and G_{tj}, respectively. $\mathcal{F}_{m_n}$ denotes the nodes map of each community between $|o_i|$ and $|o_j|$, the different map set of vertex summation gets community comparing distance. The equation is defined as equation 10.

$$d_d = \Sigma_{n=1}^{max(|\mathcal{E}_i|,|\mathcal{E}_j|)} |\mathcal{D}_{e_{ij}}| \tag{10}$$

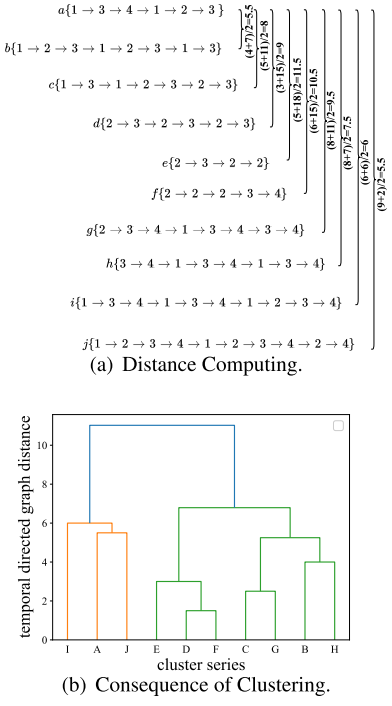
whereas $\mathcal{D}_{m_n}$ denotes the time difference of the temporal subgraph's edge set $\mathcal{E}_i$ and $\mathcal{E}_j$. Each different edge time difference summation in all o_i and o_j on G_{ti} and G_{tj} gets the temporal distance.

Last, we will cluster them through the hierarchical cluster method of *distance* on the date set comprising a set of the temporal subgraph, which if and only if $|\mathcal{G}| \geq 2$ satisfied can be computed. Each consequence of clustering must be within a period of user provision. Because we didn't improve the hierarchical clustering algorithm, the pseudo-code omits in this part, and we list the other details of the algorithm as shown in Algorithm 7.

Algorithm 7 Vertex-Driven Temporal Subgraphs Clustering(**VDTSC**)

Require: *community*, $\mathcal{G}$
Ensure: cluster **C**
1: **for** $i = 1$ to $|\mathcal{G}|$ **do**
2: **for** $j = i + 1$ to $|\mathcal{G}| - 1$ **do**
3: $d_e \leftarrow$ compute character edit distance between G_{ti} and G_{tj}.
4: $d_d \leftarrow$ compute directed edge distance between G_{ti} and G_{tj}.
5: $distance_{ij} = (d_e + d_d)/2$
6: **end for**
7: **end for**
8: Cluster subgraphs in $\mathcal{G}$ using hierarchical clustering with average linkage.
9: $C \leftarrow$ divide the clustering tree according to h.
10: **return** C

Lines 1-7 compute the distance for every two subgraphs G_{ti} and G_{tj}, line 8 enters the hierarchical clustering procedure to calculate the final clustering tree and divides according to the distance threshold h user setting.



(a) Distance Computing.

(b) Consequence of Clustering.

FIGURE 6. The process of Hierarchical Clustering computation.

Example 4: We illustrate the process of hierarchical clustering. First, we should compute the distance between every two temporal graphs according to the Definition 12 as shown in Figure 2 mentioned before. There have 10 graphs are formed to compute from G_{ta} to G_{ti}, and they will generate 45 distances, which as shown in Figure 6 (a), we only list only the distance between G_{ta} and the others; First, we compute the $distance_{ab}$, which comprise d_e and d_d since $s_a = 1 \to 3 \to 4 \to 1 \to 2 \to 3$, $s_b = 1 \to 2 \to 3 \to 1 \to 2 \to 3 \to 1 \to 3$ having the 2, 3, 7, 8 positions are different, so $d_e = 4$. Meanwhile, we compute d_d between them by adding all time differences in the edges,

so $d_d = 7$, and $distance_{ab} = (4 + 7)/2 = 5.5$. Likewise, we can compute a and d, which a has 3 positions are different with d, so $d_e = 3$, and G_{ta} has the total time cost of 21, the total time cost of D is 6, so $d_d = 21 - 6 = 15$, and the final $distance_{ad} = (3 + 15)/2 = 9$. The rest can be done in the same manner. The clustering consequence is shown in Figure 6 (b), which clusters into 2 clusters intuitively for our model by not setting a threshold to segment them. Actually, the threshold can be set according to the specific requirements of users.

VI. EXPERIMENTS

A. EXPERIMENTAL SETUP

1) ENVIRONMENT

This experiment was implemented using Python 3.6.6 on CPU I5 and 512G SSD platform with NVIDIA 750ti graphics card.

2) REAL DATASETS AND COMPARING METHODS

We need to synthesize the abnormal state time into the networks to test our online **TGC** and **FTGC** algorithm's performance. And we will set two parameters of t, σ to make each period have many instances to ensure a new temporal graph can be constructed definitely in 10 minutes at all datasets corresponding to the 10 minutes range. Each 3 second in the file has 1 abnormal time instances of each vertex randomly for the linear way to promise the motivation consistent with the real situation as far as possible. We will test the time cost under different t and σ values shown in table 2, and evaluate the size of each temporal subgraph with each value of variables. We set the cluster threshold with h to evaluate the cluster method. These network details are listed in table 3 and illustrated as follows.

a: EU-CORE

is an email network among members of the research institution. Nodes of the network can spread anonymized information. A member sends at least one email to other members.

b: DOLPHIN

is a network, whose construction is based on the observation of 62 bottlenose dolphins from 1994 to 2001. Each node of the network is a dolphin, edges represent the relation between dolphins.

c: KARATE

is a network, which comprises players, a coach, and a manager. It focused on the coach and the manager, and the network can split into two communities.

d: POLBOOK

is a network of books about US politics. The books were published around the 2004 presidential election and are sold on the Amazon website. Nodes of the network represent books, and an edge means that the same buyer purchases the books illustrated by the two nodes connecting the edge.

e: FOOTBALL

is a network of American College football games. Each node denotes a team, and an edge between two nodes means at least one match between the two teams.

Here, we select three algorithms **BSC** Algorithm [21], a streaming graph processing frame, **GraPu** Algorithm [20], a distributed algorithm for graph partition. **GShaplet** Algorithm [35], an algorithm borrows from time series. And we set $\alpha = 20 \leq min(|M|), M \iff \gamma_n$, which smaller than the smallest datasets for generating graphs correctly. Here, a user can adjust through the prior knowledge to avoid obtaining noise.

This method should consider the parameters t and σ in real situations when used. That's not our focus. We must build these temporal edges following our temporal graph model using FTGC and TGC algorithms. For example, a vertex m_i has 20 abnormal time instances in one minute, and the network Football has 115 vertices, we will simulate the node motivating in timeline incrementally, which needs to arrange all nodes in a row or column according to an abnormal time instance in a file. The first abnormal time instance of m_i as the *startNode* in Algorithm 1 of all vertices means m_i are uncertain, while generate abnormal instances in each minute can promise the number of $|\mathcal{G}|$ locate in a controlled range. So these experimental methods can thoroughly verify the algorithm performance.

We select **MGTC** and **ME-MGC** as Clustering comparison algorithms. MGTC [14] uses the interior-node clustering and the multi-graph clustering, which can finally achieve a refined multi-graph clustering result. It studies the static brain networks.

ME-MGC [13] mainly devices a frequent subgraph mining score function, calculates the top-K frequent subgraph to get the feature subgraph on the multi-graph. They complete multi-graph clustering with a combination of the ELM classification model and MapReduce framework, which is also in the static graph clustering.

3) OTHER FACTORS

a: MULTI-GRAPHS LABEL TAG

A similar temporal graph will be allocated to the same cluster for verifying the accuracy of clustering, if and only if a kind of temporal graph, which has a similar structure and temporal information. So we have an equation of labeling for temporal graph as: $SIM_{\gamma_i,\gamma_j} = (\frac{\sum_{k=1}^{|c_{i1}|} D(k)}{\sum_{k=1}^{|\gamma_i|} D(k)} \times \frac{\sum_{i=1}^{|c_{j1}|} D(k)}{\sum_{k=1}^{|\gamma_j|} D(k)} \cdots \times \frac{\sum_{k=1}^{|c_{in}|} D(k)}{\sum_{k=1}^{|\gamma_i|} D(k)} \times \frac{\sum_{k=1}^{|c_{jn}|} D(k)}{\sum_{k=1}^{|\gamma_j|} D(k)}) \times (\frac{\sum_{k=1}^{|c_{i1}|} T(k)}{\sum_{k=1}^{|\gamma_i|} T(k)} \times \frac{\sum_{k=1}^{|c_{j1}|} T(k)}{\sum_{k=1}^{|\gamma_j|} T(k)} \times \cdots \times \frac{\sum_{k=1}^{|c_{in}|} T(k)}{\sum_{k=1}^{|\gamma_i|} T(k)} \times \frac{\sum_{k=1}^{|c_{jn}|} T(k)}{\sum_{k=1}^{|\gamma_i|} T(k)})$, where c_i is got by K-Clique method [40], and with no loss generation, $T(i)$ and $D(i)$ are the total period and the total degree of one community. The same cluster

series is allocated to two temporal graphs with $SIM_{\gamma i,\gamma j} \leq 0.3$ through experiments, then we label all the graphs in the dataset generated by FTGC to evaluate the performance of the algorithms.

b: EFFECT OF STATIC GRAPH TRANSFORMATION

The dense of edge temporal graph is higher than two G^*_{ti} and $G^*_{t(i+1)}$ for each node will connect with the newest node. The variable t is small enough will lead to the graph transfer into a chain because each node in it cannot connect with any node between t, the community detection can also find community according to this structure, see section V-C. Additionally, the σ only decides the range of each G_{ti} on the timeline, so it cannot affect community detection.

4) EVALUATION INDICATORS

We adopt the Accuracy for the cluster method evaluation, adopt two measures NMI and Modularity for community detection and use time cost to evaluate the temporal graph construction.

a: NMI

Normalized mutual information is used to measure the similarity between the true community structure and the community structure obtained by community detection algorithms. The higher the value of NMI, the more accurate the community detection algorithm is. NMI is formulated as: $NMI = \frac{-2\sum_{C_A}^{i=1}\sum_{C_n}^{j=1} N_{ij}log(N_{ij}N/N_{i.}N_{.j})}{\sum_{C_A}^{i=1} N_{i.}log(N_i/N)+\sum_{C_n}^{j=1} N_{.j}log(N_{.j}/N)}$), where C_A is the number of real communities, C_B denotes the number of found communities. The matrix N represents the confusion matrix, where N_{ij} is simply the number of nodes in the real community i that appear in the detected community j.$N_{i.}$ and $N_{j.}$ are the sum over row i and column j of the confusion matrix, respectively. N is the number of nodes. When NMI is equal to 1, the community structure detected by the algorithm is the same as the real community structure. Conversely, if $NMI = 0$, the detected community structure is entirely independent of the real, and the entire network comprises of one community. It indirectly proves the effectiveness of the algorithm.

b: MODULARITY

Modularity is widely used to measure the quality of communities. The overlapping modularity is expressed as $Q = 1/2m\sum_K^{c=1}\sum_{v\in C_c} 1/O_iO_j(A_{ij} - k_ik_j/2m)$, where m is the number of edges in the entire graph, k_i, k_j are respectively node i and j, A_{ij} is the adjacency matrix of the graph and O_i and O_j respectively denote the number of communities which node i and j belong to the same cluster and 0 otherwise. The higher the value Q is, the more accurate the community results.

c: ACCURACY

Let c_i represent the clustering label result of a multi-graph clustering algorithm and y_i represent the corresponding ground truth label of the graph G_ti. Then Accuracy is defined as: $Accuracy = \sum_n^{i=1}\delta(y_i, map(c_i))/n$, where δ is the Kronecker delta function, and $map(c_i)$ is the best mapping function that permutes clustering labels to match the ground truth labels using the KuhnMunkres algorithm [41]. Larger Accuracy indicates better clustering performance.

TABLE 2. Experimental variables.

Variables	Values
t	$\{1s, 2s, \mathbf{3}s, 4s, 5s, 6s, 7s, 8s, 9s, 10s\}$
σ	$\{3s, 4s, 5s, 6s, 7s, 8s, 9s, \mathbf{1}0s, 11s, 12s\}$
α	20
h	$\{0, 0.1, 0.2, \mathbf{0}.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$
τ	$\{0, 0.01, 0.02, \ldots, \mathbf{0}.23, \ldots, 0.04, \ldots, 0.99, 1\}$

TABLE 3. Datasets information.

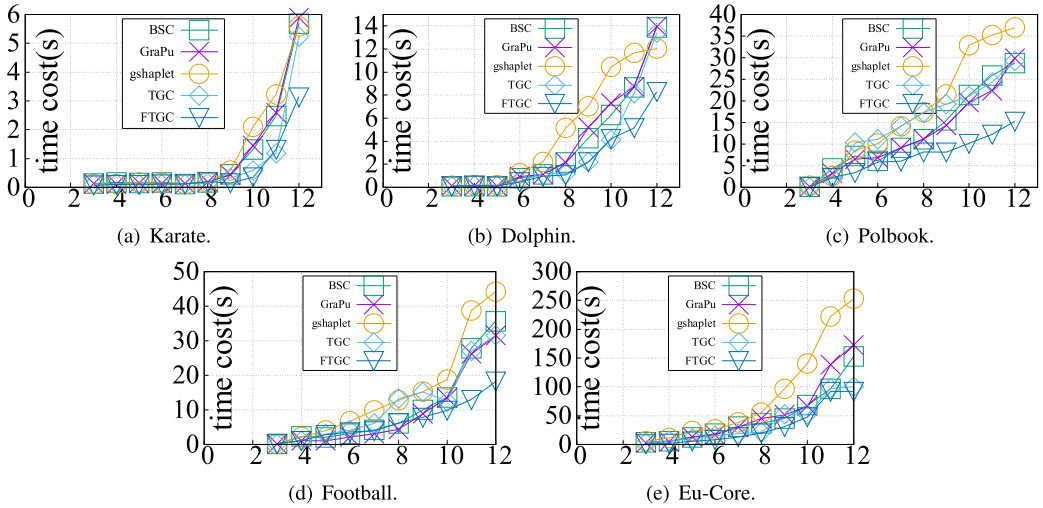
Networks	n	m	community number
Karate Club	34	78	2
Dolphin	62	159	4
Polbook	105	441	3
Football	115	613	12
Eu-Core	1005	25571	42

B. PERFORMANCE

1) EFFECT OF σ

We count each node transfer from normal status to abnormal status and add the cost of edges construction in seconds. Add each period cost to get the final time cost at the end of 10 minute, construct temporal graphs with different σ, set fixed $t = 3s$ for initialization for studying the performance of each algorithm. Observe the effect of the temporal subgraph threshold's construction time cost on different datasets. All nodes need not wait for abnormal status, which is an offline model.

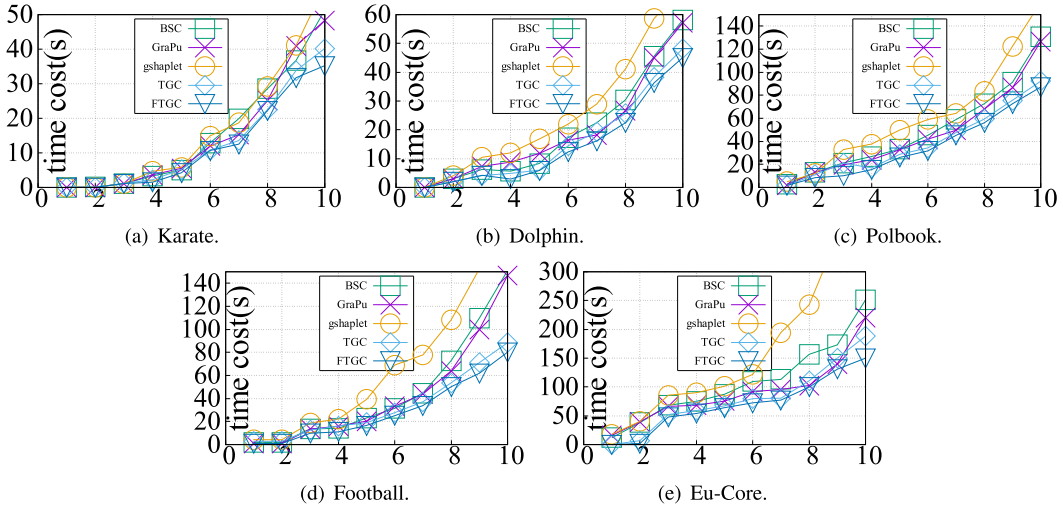
As shown in Figure 7, x axis denotes threshold σ from $3s$ to $12s$, y axis denotes time cost in seconds, where the overall trend of the computational overhead of all algorithms is up. We use the sequence of graph operations generated by our build rules required by the three algorithms. It can be seen from the figure, four kinds of algorithms of the total computing time spending growth trends, when the sigma in [3, 9]. the time cost does not increase a lot, but since the 9th second, while the number of increased significantly, the reason is the σ not only determines the number of $\mathcal{G}$, and determines the number of temporal edges, when σ increases, while t is fixed, the number of edges increase since σ contains many windows of length t, so the build time increases rapidly. As we can see from the figure, the BSC and GraPU algorithms are more expensive than ours because they have extra operations on edge insertion, which makes them spend more time. BSC uses cluster manager to manage constraints and merge components directly, which uses the streaming reservoir to ensure conformity and maximality. We save time by inserting edges directly through the two thresholds. Additionally, the clipping process of vertices and edges and distribution



(a) Karate. (b) Dolphin. (c) Polbook.

(d) Football. (e) Eu-Core.

FIGURE 7. Temporal Graph Construction Time Cost with Different σ.



(a) Karate. (b) Dolphin. (c) Polbook.

(d) Football. (e) Eu-Core.

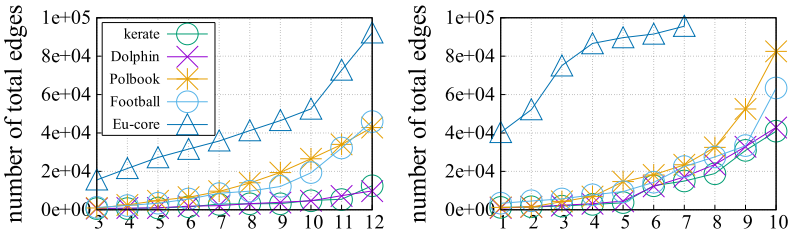
FIGURE 8. Temporal Graph Construction Time Cost with Different t(s).

of GraPu algorithms are not used to avoid additional time overhead.

2) EFFECT OF t

We evaluate different t to observe the effect of the temporal subgraph G_t^* threshold's construction time cost on 5 datasets under $\sigma = 10s$ as shown in Figure 8.

As Figure 8 shows, the x axis denotes the connectivity (local) threshold t from 1s to 10s, and the y axis indicates the time cost of the 5 methods we are comparing. Here, we can see that our algorithm outperforms others for construction procedures at different connectivity thresholds t. The 5 kinds of algorithms' time spending increased following the increase in t. The curve grows up rapidly at $t = 5s$, the reason is that the abnormal vertex number grows at the local threshold with t increasing. We recall Definition 5 (2), the number of temporal edge is $n \times (n-1)/2$, whereas n is $|\gamma^*|$, so the build time grows rapidly, among them. In the Kerate, Dolphin, EU-Core data set, the slope of the curve reduces at $t = 6s$, because the number of abnormal vertices decrease between $t = 6s$ and $t = 7s$, so the construction time also decreases accordingly. Therefore, the slope of these experimental curves can partly reflect the increase or decrease of the number of abnormal vertices. It can be seen from Figure 8 (e) that the construction time of TGC algorithm and FTGC algorithm is less than that of the other three algorithms. There is a large difference between them, mainly because the larger the value of t is, the fewer times of comparison after filtering through $PSET$, and the smaller the time cost of FTGC algorithm is. However, when the data set is small, this advantage cannot be reflected. There is little difference in the time cost between Figure (a) and Figure (b) because the number of vertices in the first two data sets is not large, so the advantage of the algorithm is not reflected. As shown



(a) different σ. (b) different t.

FIGURE 9. Number of edge of $|\mathcal{G}|$.

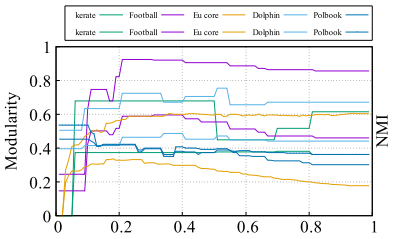


FIGURE 10. Effect of τ to modularity and NMI.

in the figure, the current change of t has a greater impact on the time cost than σ because the increase of σ does not affect the large growth of edges, according to Definition 5. But the growth of t directly affects the number of edges.

3) EFFECT OF σ AND t FOR EDGES

We further evaluate the effect of σ and t on the number of edges $|\mathcal{E}|$ that we count all edges generated of construction in each G_{tn} on five datasets using FTGC method. Figure 9 (a) uses $\sigma \in [3s, 12s]$ with $t = 3s$, Figure 9 (b) uses $t \in [1s, 10s]$ with $\sigma = 10s$.

As shown in Figure 9, x axis denotes different σ in Figure 9 (a), and denotes different t in Figure 9 (b), y axis denotes the number of edges in minutes. It can be seen from Figure 9 (a) that when σ increases, $|\mathcal{E}|$ increases but not as fast as Figure 9 (b), because the change of t directly affects $|\mathcal{E}|$. The EU-core dataset has more original vertices, so it has more temporal edges in the same condition. It can be seen from Figure 9 (b) that the number of temporal edges in EU-core is stable when $t \in [4s, 7s]$, because there are fewer vertex anomalies, so the $|\mathcal{G}|$ also decreases.

4) COMMUNITY DETECTION

We use GCD to detect the community in 5 real networks, and the community is shown in Figure 11. Our algorithm needs to set a parameter τ to promise the NMI index and Modularity maximum at each dataset. Next, we developed a group experiment to instruct our GCD algorithm under the original graph to verify and compare it with other methods.

As shown in Figure 10, x axis denotes different τ we set from 0 to 1, y axis denotes Modularity and NMI, respectively. The dashed line denotes the modularity. We can see different

TABLE 4. The Modularity of community detection.

Networks	Dolphin	Football	Eu-core
Link	0.149	0.083	0.127
OCDLCE	0.368	0.565	0.225
OCDSSE	0.377	0.41	0.07
SECD	0.401	0.595	0.137
AOCCM	0.437	0.372	0.179
CAMAS	0.396	0.401	0.172
LGIEM	**0.491**	0.572	0.292
GCD	0.485	**0.597**	**0.326**

TABLE 5. The NMI index of community detection.

Networks	Dolphin	Football	Eu-core
Link	0.392	0.716	0.279
OCDLCE	0.487	0.725	0.327
OCDSSE	0.064	0.632	0.316
SECD	0.489	0.641	0.395
AOCCM	0.872	0.795	0.263
CAMAS	0.809	0.809	0.412
LGIEM	**0.890**	0.665	0.504
GCD	0.671	**0.919**	**0.562**

τ have a different degree influence on Modularity and NMI, when the $\tau = 0.23$, the four datasets produced the maximum of the two indexes except for Polbook. Because this dataset has a boundary and fewer communities within each community when merging some vertices using a smaller τ. So we set $\tau < 0.1$ for the Polbook dataset to obtain a higher index. The subsequent experiments will use the τ setting here.

As shown in Figure 11, the community of each original real network detected using our algorithm GCD, each network has its structure to be clustered. So we then compare with other 7 community detection algorithms to test our community detection method using the consequence of the paper [24]. we compute NMI and Modularity to evaluate our method, the result is shown as table 4 and table 5. We can see that our method performs much better in a larger dataset Eu-core. The reason is that our approach ranks vertices in each step through the compactness of the graph, see section V. So our method is more adaptive to a large dataset on sparse and dense graphs. OCDLCE cluster graph is based on edges, so it is not an adaptive dataset that has too many edges for a vertex. Not Different from OCDLCE, both OCDSSE and SECD are overlapping community detection algorithms using a seed set expansion based on nodes rather than edges, which

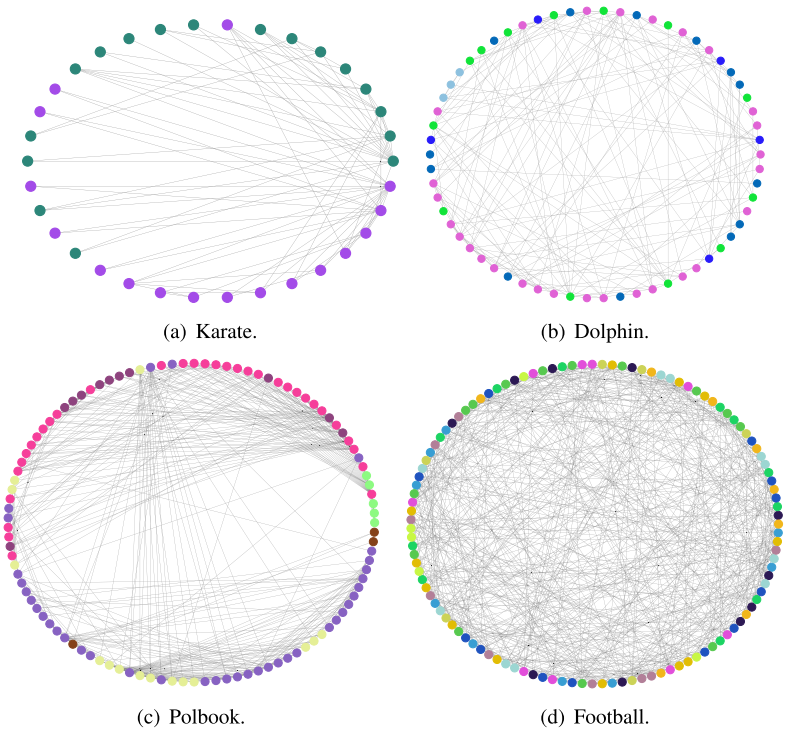


FIGURE 11. Community Detection Consequence of 5 networks.

have a limitation when local information differs from global information. Link divides a large community into several smaller communities, ignoring the connections among nodes within a large community. So links cannot detect reasonable communities. However, AOCCM and CAMAS have the same drawback that all nodes of the network cannot be in full coverage, so the community found are not precise. LGIEM does not consider the sequence processing between local information, and it is also a method that focuses on extension, so it cannot detect communities on large datasets like Eu-core.

5) CLUSTERING CONSEQUENCE

We set $\sigma = 3s$ to $12s$ to get the most suitable temporal graph for Football and Eu-core datasets, suppose a user selects $t \in [3min, 5min]$ and we get 60, 45, 36, 30, 26, 23, 20, 18, 17, 15 temporal subgraphs within 3 minutes, also, we cluster these temporal subgraphs to form C, and we use our VDTSC algorithm to get the final clusters, which are shown as Figure 12 and Figure 13. It can be easily deduced that the distance increases among different communities following the increasing number of vertices in each community, so the distance will become larger, difficult to set the threshold that distinguishes different clusters in hierarchical clusters. We use Z-score normalization to normalize all distances to express every two subgraphs to promise within the same range.

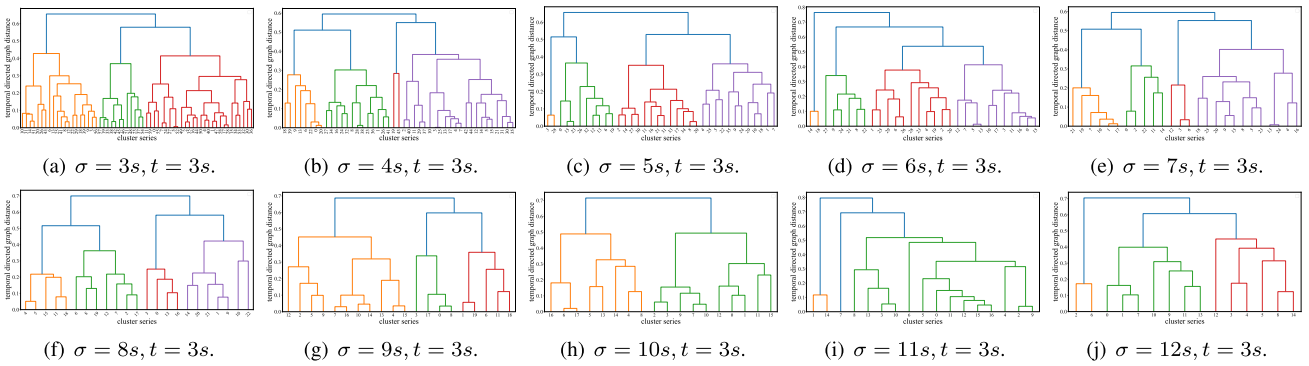
As shown in Figure 12 and 13, the x axis is series of temporal subgraphs G_m, y axis is the distance among them, they form an integrated clustering tree. Still, the distance threshold h we set will affect the final clustering. That's why we use Z-core normalization to promise distances among these clusters located in a range of [0, 1], which is easy for user settings.

6) EFFECT OF h AND τ

h decides the cluster allocation, τ decides the structure of each temporal graph, so we evaluate the two parameters to observe our method's stability in football and Eu-core. Figure 14 shows the consequence.

We can see that the τ locate in the scope 0.4-0.6 achieves higher accuracy because our *GCD* method arrives at a higher level on two datasets in Figure 14 (a) and (b). And, h directly affects the accuracy. The accuracy will drop if we set a smaller value of h. The reverse has the same effect since hierarchical clustering is sensitive to distance threshold.

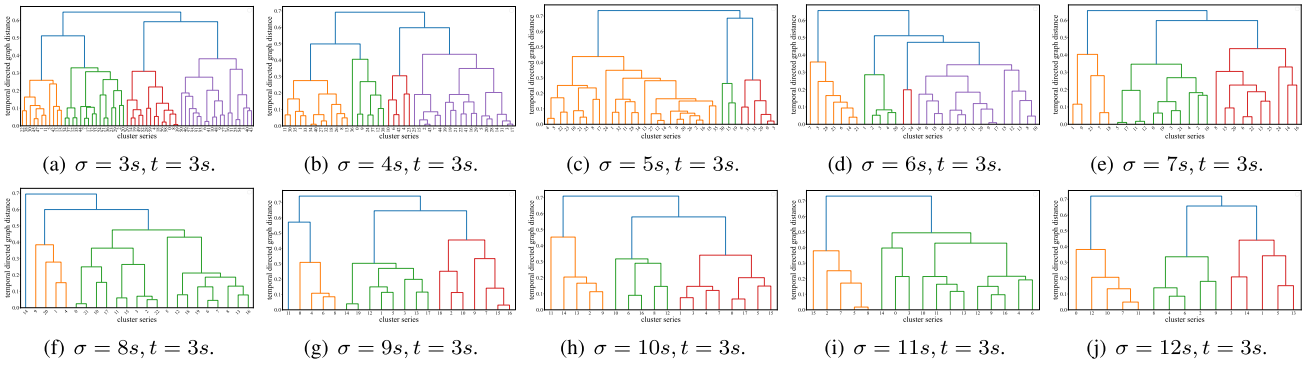
Then we compare the accuracy with two methods MGTC and ME-MGC, at $h = 0.3$ and $\tau = 0.23$, respectively. We train these two classifiers using Football and Eu-core datasets above, considering they need labeled graphs, so we set the σ and t is $3s$ simultaneously. The accuracy of these 3 methods is shown in Table 6.



(a) $\sigma = 3s, t = 3s$. (b) $\sigma = 4s, t = 3s$. (c) $\sigma = 5s, t = 3s$. (d) $\sigma = 6s, t = 3s$. (e) $\sigma = 7s, t = 3s$.

(f) $\sigma = 8s, t = 3s$. (g) $\sigma = 9s, t = 3s$. (h) $\sigma = 10s, t = 3s$. (i) $\sigma = 11s, t = 3s$. (j) $\sigma = 12s, t = 3s$.

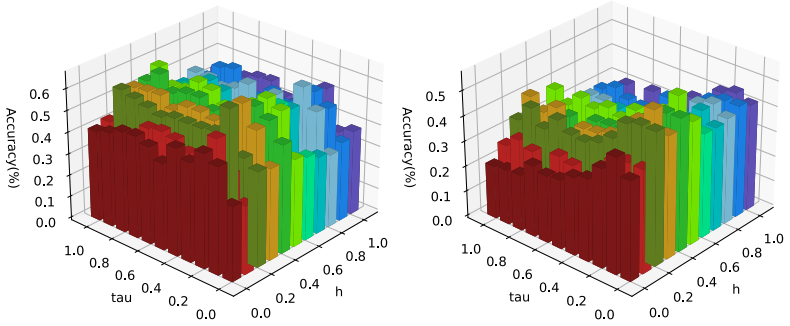
FIGURE 12. Cluster Consequence on Football.



(a) $\sigma = 3s, t = 3s$. (b) $\sigma = 4s, t = 3s$. (c) $\sigma = 5s, t = 3s$. (d) $\sigma = 6s, t = 3s$. (e) $\sigma = 7s, t = 3s$.

(f) $\sigma = 8s, t = 3s$. (g) $\sigma = 9s, t = 3s$. (h) $\sigma = 10s, t = 3s$. (i) $\sigma = 11s, t = 3s$. (j) $\sigma = 12s, t = 3s$.

FIGURE 13. Cluster Consequence on Eu-core.



(a) Football Dataset. (b) Eu-core Dataset.

FIGURE 14. Effect of h and τ to Clustering Accuracy.

TABLE 6. The Accuracy of Temporal Graph Clustering.

Methods	Accuracy	
	Football	**Eu-core**
VDTSC	**0.68**	**0.57**
MTGC	0.52	0.49
ME-MGC	0.31	0.27

We can find that our algorithm has a much better performance on both datasets of Football and Eu-core from Table 6. Because the above two algorithms do not consider temporal information, our algorithm considers the distance of the time dimension, and its accuracy is more than 30% higher than MTGC for football and more 16% higher than MTGC on Eu-core. This proves our algorithm's validity. ME-MGC algorithm is weaker than MGTC algorithm, which the main reason is MGTC algorithm performs clustering alternately of internal nodes and global structures to verify each other. MGTC algorithm more accurately compared with ME-MGC only looks for structural features from the perspective of subgraphs to find structures. Another main reason for the accuracy of ME-MGC algorithm is that the number of label graphs is not enough, and the classification of graphs may produce over-fitting.

We can find that these temporal subgraphs are clustered, and the results are clear for analysis users. This method can also cluster other temporal subgraph clustering problems. We also get different results if we set different thresholds for t and σ. We provide a novel cluster for the **VDTSC** problem.

VII. CONCLUSION

This paper proposes a temporal graph construction model, which generates a temporal graph based on abnormal streaming data. We further offer a hierarchical clustering method on a set of temporal graph, calculating the distance based on community comparing distance and time distance according to their properties. We verify the effectiveness and efficiency of temporal graph model, verify the accuracy and show classification results on 5 real network datasets.

In addition, on the one hand, the temporal graph building model applies to the area where the exception time can be obtained, but the two thresholds are required according to the prior knowledge. Graph structure information is hidden due to add temporal information, if only use the temporal information is as a basis for the structure partition, which leads to the structure of the temporal graph partition prefer to temporal perspective since the temporal information is added. So we use structural similarity to replace the temporal information, i.e., removing the temporal information of temporal graph do community detection.

On the other hand, the temporal graph on the structure and temporal information is varied after the complete building. If we mine frequent subgraph directly, where there may not get effective temporal graph structure, or difficult to go through frequent subgraphs analysis the internal information of temporal graph. At the same time, expertise acts an essential role when label temporal subgraphs. It's nontrival for its cost is large and difficult to be applied in practice. As a result, we use a way of through a double-standard temporal subgraphs clustering method based on community comparison and temporal distance. And our method provide process them in a novel manner.

REFERENCES

[1] P. Holme and J. Saramäki, "Temporal networks," *Phys. Rep.*, vol. 519, no. 3, pp. 97–125, 2012.

[2] N. Masuda and P. Holme, "Small inter-event times govern epidemic spreading on networks," *Phys. Rev. Res.*, vol. 2, no. 2, p. 23163, May 2020.

[3] C. Song, T. Ge, C. Chen, and J. Wang, "Event pattern matching over graph streams," *Proc. VLDB Endowment*, vol. 8, no. 4, pp. 413–424, 2014.

[4] H. Qin, R. Li, Y. Yuan, G. Wang, W. Yang, and L. Qin, "Periodic communities mining in temporal networks: Concepts and algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 1, no. 8, pp. 3927–3945, Sep. 2020.

[5] I. Maduako and M. Wachowicz, "A space-time varying graph for modelling places and events in a network," *Int. J. Geographical Inf. Sci.*, vol. 33, no. 10, pp. 1915–1935, Oct. 2019, doi: 10.1080/13658816.2019.1603386.

[6] M. Latapy, T. Viard, and C. Magnien, "Stream graphs and link streams for the modeling of interactions over time," *Social Netw. Anal. Mining*, vol. 8, no. 1, pp. 1–29, Dec. 2018.

[7] L. Zhang, L. Zhao, S. Qin, and D. Pfoser, "TG-GAN: Continuous-time temporal graph generation with deep generative models," 2020, *arXiv:2005.08323*.

[8] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "GraphRNN: Generating realistic graphs with deep auto-regressive models," 2018, *arXiv:1802.08773*.

[9] S. Purohit, L. Holder, and G. Chin, "Temporal graph generation based on a distribution of temporal motifs," Pacific Northwest Nat. Lab. (PNNL), Richland, WA, USA, Tech. Rep. PNNL-SA-134797, 2018.

[10] Y. Bai, Y. Wang, Y. Tong, Y. Yang, Q. Liu, and J. Liu, "Boundary content graph neural network for temporal action proposal generation," in *Proc. Eur. Conf. Comput. Vis.* Glasgow, U.K.: Springer, 2020, pp. 121–137.

[11] J. Wu, S. Pan, X. Zhu, and Z. Cai, "Boosting for multi-graph classification," *IEEE Trans. Cybern.*, vol. 45, no. 3, pp. 416–429, Mar. 2015.

[12] J. Wu, X. Zhu, C. Zhang, and P. S. Yu, "Bag constrained structure pattern mining for multi-graph classification," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 10, pp. 2382–2396, Oct. 2014.

[13] J. Pang, Y. Gu, J. Xu, X. Kong, and G. Yu, "Parallel multi-graph classification using extreme learning machine and mapreduce," *Neurocomputing*, vol. 261, pp. 171–183, Oct. 2017.

[14] G. Ma, L. He, B. Cao, J. Zhang, P. S. Yu, and A. B. Ragin, "Multi-graph clustering based on interior-node topology with applications to brain networks," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Riva del Garda, Italy: Springer, 2016, pp. 476–492.

[15] C. Linhares, J. R. Ponciano, F. Pereira, L. Rocha, J. Paiva, and B. Travencolo, "A scalable node ordering strategy based on community structure for enhanced temporal network visualization," *Comput. Graph.*, vol. 84, pp. 185–198, Nov. 2019.

[16] S. Yin, S. Chen, Z. Feng, K. Huang, D. He, P. Zhao, and M. Y. Yang, "Node-grained incremental community detection for streaming networks," in *Proc. IEEE 28th Int. Conf. Tools Artif. Intell. (ICTAI)*, Nov. 2016, pp. 585–592.

[17] A. Ferrari and C. Richard, "Non-parametric community change-points detection in streaming graph signals," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 5545–5549.

[18] A. E. Sariyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and V. Çatalyürek, "Sonic: Streaming overlapping community detection," *Data Mining Knowl. Discovery*, vol. 30, no. 4, pp. 819–847, 2016.

[19] Y. Wu, M. Bateni, A. Linhares, F. M. G. de Almeida, A. Montanari, A. Norouzi-Fard, and J. Tardos, "Streaming belief propagation for community detection," 2021, *arXiv:2106.04805*.

[20] F. Sheng, Q. Cao, H. Cai, J. Yao, and C. Xie, "GraPU: Accelerate streaming graph analysis through preprocessing buffered updates," in *Proc. ACM Symp. Cloud Comput.* New York, NY, USA: Association for Computing Machinery, 2018, pp. 301–312, doi: 10.1145/3267809.3267811.

[21] J. Zhang, Y. Pei, G. Fletcher, and M. Pechenizkiy, "A bounded-size clustering algorithm on fully-dynamic streaming graphs," *Intell. Data Anal.*, vol. 22, no. 5, pp. 1039–1058, Sep. 2018.

[22] B. C. Das, M. M. Anwar, and M. A.-A. Bhuiyan, "Attribute driven temporal active local online community detection," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining (ASONAM)*. Los Alamitos, CA, USA: IEEE Computer Society, Dec. 2020, pp. 619–622, doi: 10.1109/asonam49781.2020.9381442.

[23] X. Pan, G. Xu, B. Wang, and T. Zhang, "A novel community detection algorithm based on local similarity of clustering coefficient in social networks," *IEEE Access*, vol. 7, pp. 121586–121598, 2019.

[24] T. Ma, Q. Liu, J. Cao, Y. Tian, A. Al-Dhelaan, and M. Al-Rodhaan, "LGIEM: Global and local node influence based community detection," *Future Gener. Comput. Syst.*, vol. 105, pp. 533–546, Apr. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X19310210

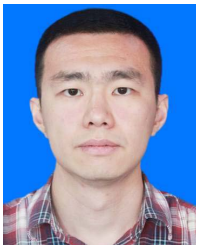
[25] B. Saoud and A. Moussaoui, "Node similarity and modularity for finding communities in networks," *Phys. A, Stat. Mech. Appl.*, vol. 492, pp. 1958–1966, Feb. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378437117311883

[26] F. Hu, J. Liu, L. Li, and J. Liang, "Community detection in complex networks using Node2vec with spectral clustering," *Phys. A, Stat. Mech. Appl.*, vol. 545, May 2020, Art. no. 123633. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378437119320254

[27] X. You, Y. Ma, and Z. Liu, "A three-stage algorithm on community detection in social networks," *Knowl.-Based Syst.*, vol. 187, Jan. 2020, Art. no. 104822. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705119302977

[28] F. Zeng, J. Peng, and W. Li, "An effective clustering routing algorithm based on social-interest similarity in mobile opportunistic networks," in *Proc. 10th EAI Int. Conf. Mobile Multimedia Commun.*, 2017, pp. 41–46, doi: 10.4108/eai.13-7-2017.2270328.

[29] T. Li, W. Wang, X. Wu, H. Wu, P. Jiao, and Y. Yu, "Exploring the transition behavior of nodes in temporal networks based on dynamic community detection," *Future Gener. Comput. Syst.*, vol. 107, pp. 458–468, Jun. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X19326901
[30] A. Hollocou, J. Maudet, T. Bonald, and M. Lelarge, "A streaming algorithm for graph clustering," in *Proc. NIPS Wokshop Adv. Model. Learn. Interact. Complex Data*, Long Beach, CA, USA, Dec. 2017, pp. 1–12. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01639506
[31] M. A. K. Patwary, S. Garg, and B. Kang, "Window-based streaming graph partitioning algorithm," in *Proc. Australas. Comput. Sci. Week Multiconference*, Jan. 2019, pp. 1–10.
[32] M. Mariappan, J. Che, and K. Vora, "DZiG: Sparsity-aware incremental processing of streaming graphs," in *Proc. 16th Eur. Conf. Comput. Syst.* New York, NY, USA: Association for Computing Machinery, Apr. 2021, pp. 83–98, doi: 10.1145/3447786.3456230.
[33] F. Sheng, Q. Cao, and J. Yao, "Exploiting buffered updates for fast streaming graph analysis," *IEEE Trans. Comput.*, vol. 70, no. 2, pp. 255–269, Feb. 2021.
[34] J. Yao, B. Cui, Y. Huang, and Y. Zhou, "Bursty event detection from collaborative tags," *World Wide Web*, vol. 15, no. 2, pp. 171–195, 2012, doi: 10.1007/s11280-011-0136-2.
[35] H. Wang, J. Wu, X. Zhu, Y. Chen, and C. Zhang, "Time-variant graph classification," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 50, no. 8, pp. 2883–2896, 2020.
[36] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger, "Scan: A structural clustering algorithm for networks," in *Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2007, pp. 824–833, doi: 10.1145/1281192.1281280.
[37] A. Zaeemzadeh, M. Joneidi, N. Rahnavard, and G.-J. Qi, "Co-SpOT: Cooperative spectrum opportunity detection using Bayesian clustering in spectrum-heterogeneous cognitive radio networks," *IEEE Trans. Cognit. Commun. Netw.*, vol. 4, no. 2, pp. 206–219, Jun. 2018.
[38] X. Sun, Y. Tan, Q. Wu, and J. Wang, "Hasse diagram based algorithm for continuous temporal subgraph query in graph stream," in *Proc. 6th Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT)*, Oct. 2017, pp. 241–246.
[39] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse, "Identification of influential spreaders in complex networks," *Nature Phys.*, vol. 6, pp. 888–893, Aug. 2010, doi: 10.1038/nphys1746.
[40] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, no. 7043, pp. 814–818, Jun. 2005.
[41] H. W. Kuhn, "The Hungarian method for the assignment problem," *Nav. Res. Logistics*, vol. 2, nos. 1–2, pp. 83–97, Mar. 1955.



HANLIN ZHANG was born in Shenyang, Liaoning, China, in 1993. He received the B.S. degree in CST from Liaoning University, Shenyang, in 2020, where he is currently pursuing the Ph.D. degree in big data statistics and intelligent.

He was a Program Developer with the Mine Safety Engineering Laboratory for three years. His research interests include big data processing techniques and graph data processing techniques.

LINLIN DING received the M.S. and Ph.D. degrees in computer science and technology from Northeastern University, Shenyang, China, in July 2008 and 2013, respectively.

She is currently an Associate Professor with the School of Information, Liaoning University, Shenyang. She published more than 40 research articles in international conference proceedings and journals. Her research interests include big data management, uncertain data management, high-dimensional data, and distributed data management.

GANG ZHANG was born in Anyang, Henan, China, in 1998. He is currently pursuing the M.S. degree in software engineering with Liaoning University, Shenyang, Liaoning, China.

He was a Program Developer at the Mine Safety Engineering Laboratory for two years. His research interests include big data processing techniques and graph data processing techniques.

YISHAN PAN was born in Donggang, Liaoning, China, in 1964. He received the M.S. degree in mining engineering from Liaoning Project Technology University, Fuxin, Liaoning, in 1986, and the Ph.D. degree in solid mechanics from Tsinghua University, Beijing, China, in 1997.

He is currently a Professor with the School of Environment, Liaoning University, Shenyang. He published more than 80 research paper in international conference proceedings and journals. His research interests include coal mine rock burst, gas outburst, and coalbed methane exploitation.

Mr. Pan's awards and honors include the China Coal Industry Science and Technology First Award, the National Science and Technology Progress Second Award, and the China Coal Industry Association Special Award.

BAOYAN SONG (Member, IEEE) received the B.Eng. and M.Sc. degrees in computer software and theory from Northeastern University, Shenyang, in 1996, where she received the Ph.D. degree, in 2002.

From 1988 to 1993, she was a Research Assistant at the School of Information Science and Engineering, Northeastern University. Since 2005, she has been a Professor with the School of Information, Liaoning University. She is the author of three books, more than 100 articles, and four inventions. Her research interests include database theory and techniques, RFID event stream processing techniques, massive data processing techniques, and graph data processing techniques. She is a member of the China Computer Federation.