Zhu CX, Lin LL, Yuan PP *et al.* Discovering cohesive temporal subgraphs with temporal density aware exploration. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 37(5): 1068–1085 Sept. 2022. DOI 10.1007/s11390-022-2431-z

Discovering Cohesive Temporal Subgraphs with Temporal Density Aware Exploration

Chun-Xue Zhu (朱春雪), Long-Long Lin (林隆龙), *Member, CCF*
Ping-Peng Yuan* (袁平鹏), *Senior Member, CCF, Member, ACM, IEEE*, and
Hai Jin (金　海), *Fellow, CCF, IEEE, Life Member, ACM*

National Engineering Research Center for Big Data Technology and System, Huazhong University of Science and Technology, Wuhan 430074, China

Service Computing Technology and System Laboratory, Huazhong University of Science and Technology Wuhan 430074, China

Cluster and Grid Computing Laboratory, Huazhong University of Science and Technology, Wuhan 430074, China

School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

E-mail: cxzhu@hust.edu.cn; longlonglin@swu.edu.cn; {ppyuan, hjin}@hust.edu.cn

Received April 16, 2022; accepted September 8, 2022.

Abstract Real-world networks, such as social networks, cryptocurrency networks, and e-commerce networks, always have occurrence time of interactions between nodes. Such networks are typically modeled as temporal graphs. Mining cohesive subgraphs from temporal graphs is practical and essential in numerous data mining applications, since mining cohesive subgraphs gets insights into the time-varying nature of temporal graphs. However, existing studies on mining cohesive subgraphs, such as Densest-Exact and k-truss, are mainly tailored for static graphs (whose edges have no temporal information). Therefore, those cohesive subgraph models cannot indicate both the temporal and the structural characteristics of subgraphs. To this end, we explore the model of cohesive temporal subgraphs by incorporating both the evolving and the structural characteristics of temporal subgraphs. Unfortunately, the volume of time intervals in a temporal network is quadratic. As a result, the time complexity of mining temporal cohesive subgraphs is high. To efficiently address the problem, we first mine the temporal density distribution of temporal graphs. Guided by the distribution, we can safely prune many unqualified time intervals with the linear time cost. Then, the remaining time intervals where cohesive temporal subgraphs fall in are examined using the greedy search. The results of the experiments on nine real-world temporal graphs indicate that our model outperforms state-of-the-art solutions in efficiency and quality. Specifically, our model only takes less than two minutes on a million-vertex DBLP and has the highest overall average ranking in EDB and TC metrics.

Keywords temporal network, temporal feature distribution, cohesive subgraph, convex property

1 Introduction

Mining cohesive subgraphs is very important for network analysis. Previous research proposed many models and algorithms for mining cohesive subgraphs on static graphs, where edges do not evolve with time [1]. For example, one of cohesive subgraph models is the average-degree density, namely the average number of edges induced per node [2]. However, real-world networks are often time-stamped, indicating when the interactions between vertices occurred [3–6]. For example, in social networks like Facebook or LinkedIn, each edge has a timestamp to indicate when a user follows another [6]. In neuroscience, the brain network, a complex system of interconnected neurons, is also a temporal network, in which nerve cells interact by passing electrical impulses [7].

Regular Paper
Special Section on Scalable Data Science
The work was supported by the National Key Research and Development Program of China under Grant No. 2018YFB1402802 and the National Natural Science Foundation of China under Grant Nos. 61932004 and 62072205.
*Corresponding Author

Existing cohesive models, such as average-degree density on non-temporal graphs, may not work when handling such temporal networks. For instance, Fig.1 illustrates a mobile phone-call network, in which each timestamp on edges indicates when two persons had a phone call. The subgraph H_1 is the densest subgraph when ignoring the timestamps on edges[2]. However, if we want to detect persons who interact frequently and intensively, H_1 is not a good candidate because Jan and Bob are both frequently and tightly connected to Lisa but are loosely connected to the other persons in H_1. In addition, the phone calls between vertices of H_1 span a longer gap and thus the communication in H_1 is less frequent and intensive than that in H_3 (more details in Section 2). Therefore, it is important to define the temporal cohesive subgraph to suit the real-life applications. For example, the telecom operators can provide more guaranteed QoS (e.g., more bandwidth) to the recognized frequent and dense calling users. Other examples include: recognizing the most active group in social temporal networks to promote advertising and friend recommendation, and finding the densely and tightly connected neurons to facilitate the in-depth research on the mechanisms of brain networks. In addition, it can also be applied to discover the congested networks, scientific cooperation groups, and other meaningful hidden patterns.

Recently, some studies have emerged on mining cohesive subgraphs in temporal networks. For example, Rozenshtein *et al.*[8] proposed the KGAPPROX algorithm for mining cohesive temporal subgraphs based on average-degree density[2]. As mentioned above, the model ignores the frequency of interactions, and thus may discover some unsuitable subgraphs. As an improvement, Chu *et al.*[9] proposed a bursting community on temporal networks, which should satisfy the constraints on both density and duration. However, mining bursting communities is NP-hard, wherefore the bursting community mining algorithm OL faces an intractable computational bottleneck. FIDES$^+$[10] focuses on finding dense subgraphs in special temporal networks whose structure remains fixed, and the weights of edges change over time. Therefore, FIDES$^+$ is limited by its application scenarios. Besides, some studies model temporal networks as a special case of multilayer graphs to search the meaningful dense subgraphs. For instance, MiMAG[11], DCCS[12] and FirmCore[13] define a dense subgraph in discrete layers based on the γ-quasi-clique[14,15] and k-core[16,17], which have large gaps when the number of layers is large. As a result, these models cannot capture the structural characteristics in consecutive time intervals.

To this end, we model the temporal cohesive subgraph as a temporal densest subgraph (for brief, TDS),

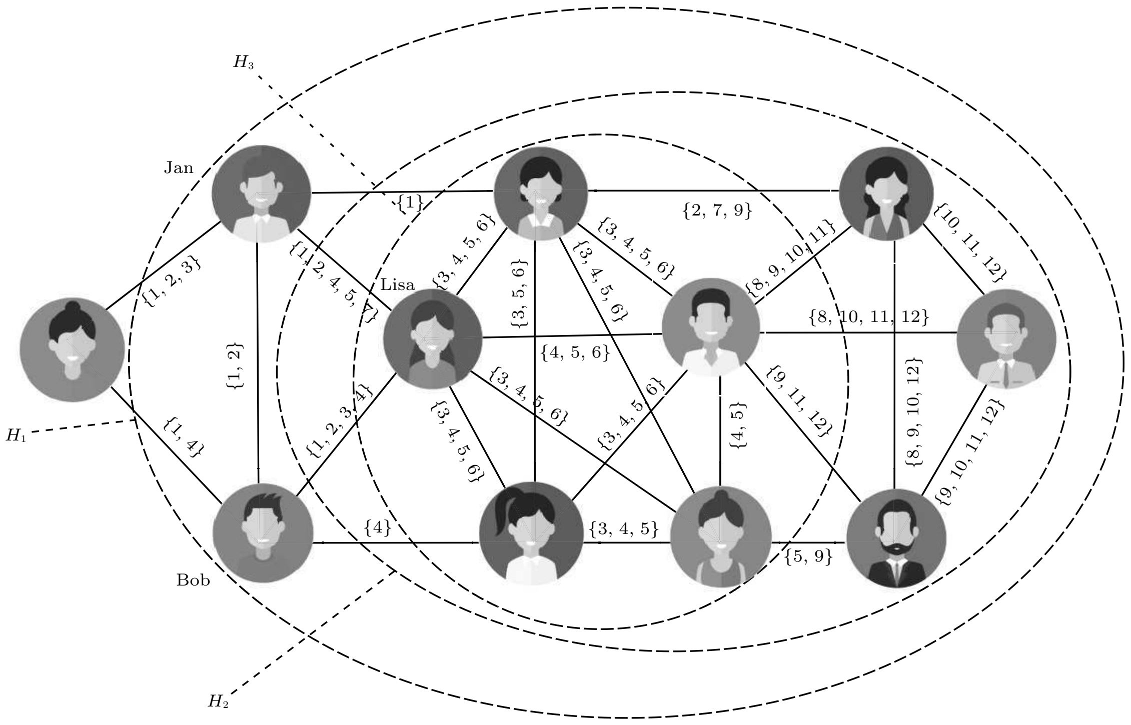


Fig.1. Example phone-call network.

which is defined as the densely-connected and tightly-interacting temporal subgraph. Specifically, TDS should have the most frequent interactions among vertices within a given time interval (more details in Section 2). However, solving TDS is technically challenging because the total number of time sub-intervals is quadratic with respect to the time span of the temporal graph, resulting in inefficiency in massive temporal networks. Therefore, we exploit the temporal feature distribution for identifying candidate time intervals. As a result, we only need to search TDS on the candidate intervals instead of the quadratic intervals. Our contributions are reported as follows.

- *Novel Model.* We propose a novel temporal subgraph model named TDS. The model is based on the condensed density for the temporal graph, instead of average density. The condensed density is defined as the ratio between the count of temporal edges per unit time and the volume of vertices. In this way, our model can capture both the structural and the temporal characteristics of temporal cohesive subgraphs.
- *Efficient Algorithms.* To mine temporal densest subgraphs, we first propose a baseline algorithm BA (refer to Section 3) to identify an exact solution. However, the algorithm has high overhead. Thus, to further improve the efficiency, we decompose the temporal graph into the sequence of snapshots, and show the temporal feature distribution of the snapshot sequence that is convexizable. With the convex property, we propose the ITKT algorithm (refer to Subsection 4.1) to extract constant qualified candidate time intervals from the quadratic time intervals with powerful strategies.
- *Extensive Experiments.* We evaluate the efficiency and effectiveness of our solutions on nine real-world temporal graphs. Specifically, the efficiency tests show that our algorithms outperform others in terms of the running time. For example, on a million-vertex DBLP[①], the heuristic algorithm GFDS+ITKT, consisting of the GFDS algorithm (refer to Subsection 4.2) and the ITKT algorithm (refer to Subsection 4.1), requires only about two minutes to return the solution, while some competitors need more than two days. Additionally, we compare the effectiveness of TDS with the existing models through qualitative analyses and case studies. We find that our model can mine some meaningful patterns with a higher quality than other models.

Roadmap. Some basic concepts and the formulation of our problem TDS are presented in Section 2. An exact solution for TDS is proposed in Section 3, and another heuristic but efficient temporal density aware exploration algorithm ITKT is presented in Section 4 with several pruning techniques. Experimental results of our model will be analyzed in Section 5. The closely related work and conclusions will be discussed in Section 6 and Section 7, respectively.

2 Preliminaries

2.1 Basic Concepts

Let $\mathcal{G} = (V, E, \mathit{\Gamma})$ denote the undirected temporal network, in which $\mathit{\Gamma}$ is the ordered and continuous integer list of timestamps when some connections between vertices occur. $E = \{(u, v, t)|u, v \in V, t \in \mathit{\Gamma}\}$ collects the temporal edges for $\mathcal{G}$. We refer to $d_{\mathcal{G}}(v) = |\{(v, u, t) \in E\}|$ as the temporal degree of v. Let $\lfloor \mathit{\Gamma} \rfloor$ and $\lceil \mathit{\Gamma} \rceil$ be the smallest timestamp and the largest timestamp of $\mathit{\Gamma}$, respectively. And the time span of $\mathcal{G}$ is $I(\mathit{\Gamma}) = [\lfloor \mathit{\Gamma} \rfloor, \lceil \mathit{\Gamma} \rceil]$. It indicates that the first temporal edge of $\mathcal{G}$ begins at timestamp $\lfloor \mathit{\Gamma} \rfloor$ and the last temporal edge of $\mathcal{G}$ ends at $\lceil \mathit{\Gamma} \rceil$. In most cases, time intervals are generally closed. We also use open or half-open time intervals to denote those intervals which do not include their endpoint(s). Since $\mathit{\Gamma}$ is ordered and continuous, it is easy to transfer an open or half-open interval to a closed interval by moving out the endpoint(s). $|I(\mathit{\Gamma})| = \lceil \mathit{\Gamma} \rceil - \lfloor \mathit{\Gamma} \rfloor + 1$, denoting the number of timestamps. Fig.2(a) displays a temporal graph $\mathcal{G}$ consisting of six nodes and 34 temporal edges with $\mathit{\Gamma} = \{1, 2, 3, 4, 5, 6\}$. Therefore, its time interval $I(\mathit{\Gamma}) = [1, 6]$, $|I(\mathit{\Gamma})| = 6$.

We denote $G(V, \bar{E})$ as the static network of $\mathcal{G}$, in which $\bar{E} = \{(u, v)|\exists(u, v, t) \in E\}$. Fig. 2(b) shows the static graph G of $\mathcal{G}$. Vertex set S induces a static subgraph $G_S = (S, \bar{E}_S)$ of G, where $S \subseteq V$ and $\bar{E}_S = \{(u, v) \in E|u, v \in S\}$. $\mathcal{G}_S = (S, E_S, \mathit{\Gamma}_S)$ is a temporal subgraph of $\mathcal{G}$ if $S \subseteq V$, $E_S \subseteq E$, and $\mathit{\Gamma}_S \subseteq \mathit{\Gamma}$. Fig. 2(c) exhibits a temporal subgraph $\mathcal{G}_1$ of $\mathcal{G}$, which has five vertices, and 18 temporal edges with $\mathit{\Gamma}_S = [1, 3]$. Given a temporal interval I, the corresponding temporal subgraph for $\mathcal{G}$ during interval I is defined as $\mathcal{G}_I = (V_{\mathcal{G}_I}, E_{\mathcal{G}_I}, I)$, where $E_{\mathcal{G}_I} = \{(u, v, t) \in E| \lfloor I \rfloor \leqslant t \leqslant \lceil I \rceil\}$. Specially, let $G_t = \{(u, v, t) \in E\}$ be a snapshot of $\mathcal{G}$ at the timestamp t and let $deg(t) = \frac{1}{2}\sum_{v \in G_t} d_{G_t}(v)$. We refer to $\mathcal{G}_{(S,I)} = (S, E_{(S,I)}, I)$ as the subgraph of $\mathcal{G}$ induced by the vertex set S and the time interval I, where $S \subseteq V$, $I \subseteq \mathit{\Gamma}$ and $E_{(S,I)} = \{(u, v, t) \in E|t \in I \text{ and } u, v \in S\}$.

①http://konect.cc/networks/dblp_coauthor/, Aug. 2021.

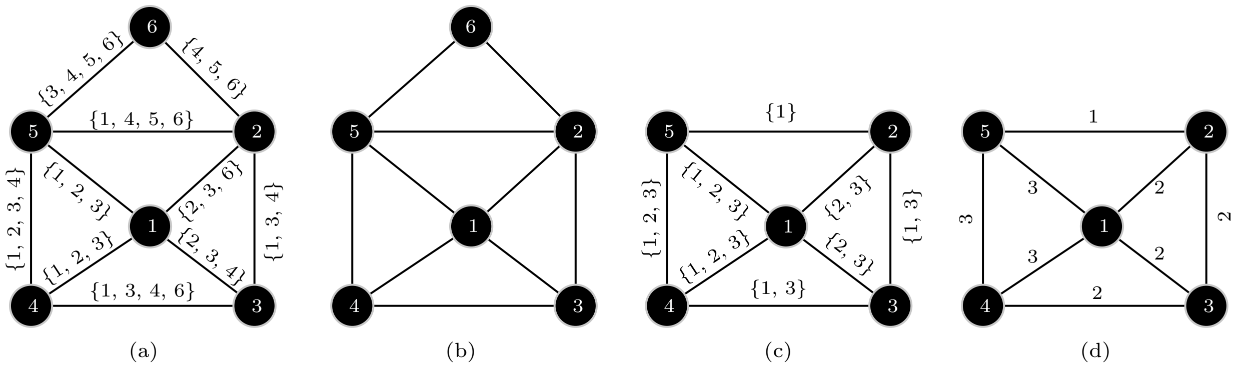


Fig.2. (a) Temporal graph $\mathcal{G}$. (b) Static graph G of $\mathcal{G}$. (c) Temporal subgraph $\mathcal{G}_1$ of $\mathcal{G}$. (d) Condensed graph $\hat{\mathcal{G}}_1$ of $\mathcal{G}_1$.

To help formalize our problem, we give the following two definitions.

Definition 1 (Condensed Graph). *For a temporal graph $\mathcal{G} = (V, E, \Gamma)$, its condensed graph $\hat{\mathcal{G}} = (V, \hat{E}, w)$, in which $\hat{E} = \{(u, v)|\exists(u, v, t) \in E\}$. $w : \hat{E} \to \mathbb{R}$ is a function to assign each $(u, v) \in \hat{E}$ a weight. Formally, $w(u, v) = |\{(u, v, t) \in E\}|$.*

We use $d_{\hat{\mathcal{G}}}(v) = \sum_{(v,u)\in\hat{E}} w(v, u)$ to represent the weighted degree of v. Furthermore, let $W(\hat{\mathcal{G}}) = \sum_{e\in\hat{E}} w(\hat{e}) = \frac{1}{2}\sum_{v\in V} d_{\hat{\mathcal{G}}}(v)$.

Definition 2 (Condensed Density). *Given a temporal graph $\mathcal{G} = (V, E, \Gamma)$ and its condensed graph $\hat{\mathcal{G}}$, the condensed density (cdensity) of $\hat{\mathcal{G}}$, represented as $cdensity(\hat{\mathcal{G}})$, is defined as:*

$$cdensity(\hat{\mathcal{G}}) = \frac{W(\hat{\mathcal{G}})}{|V||I(\Gamma)|} = \frac{1}{2}\frac{\sum_{v\in V} d_{\hat{\mathcal{G}}}(v)}{|V||I(\Gamma)|}.$$

The condensed graph $\hat{\mathcal{G}}_1$ of the temporal subgraph $\mathcal{G}_1$ is shown in Fig. 2(d), whose condensed density is $cdensity(\hat{\mathcal{G}}_1) = 1.2$.

Proposition 1. *For a temporal graph $\mathcal{G}$ and its subgraph $\mathcal{G}_{(S,I)}$, let $\hat{S} = \hat{\mathcal{G}}_{(S,I)}$. $\forall v_1,\ v_2 \in \hat{S}$, if $d_{\hat{S}}(v_1) \leqslant d_{\hat{S}}(v_2)$, and then $cdensity(\hat{S}\backslash v_1) \geqslant cdensity(\hat{S}\backslash v_2)$.*

Proof.

$$\begin{aligned} cdensity(\hat{S}\backslash v_1) &= \tfrac{W(\hat{S}\backslash v_1)}{|S\backslash v_1||I|} = \tfrac{W(\hat{S})-d_{\hat{S}}(v_1)}{(|S|-1)|I|} \\ &= \tfrac{W(\hat{S})}{(|S|-1)|I|} - \tfrac{d_{\hat{S}}(v_1)}{(|S|-1)|I|}. \end{aligned}$$

Similarly, we have

$$cdensity(\hat{S}\backslash v_2) = \frac{W(\hat{S})}{(|S|-1)|I|} - \frac{d_{\hat{S}}(v_2)}{(|S|-1)|I|}.$$

Thus, the following equation holds:

$$\begin{aligned} & cdensity(\hat{S}\backslash v_1) - cdensity(\hat{S}\backslash v_2) \\ = & \frac{d_{\hat{S}}(v_2) - d_{\hat{S}}(v_1)}{(|S|-1)|I|} \\ \geqslant & \ 0. \end{aligned}$$

Namely, $cdensity(\hat{S}\backslash v_1) \geqslant cdensity(\hat{S}\backslash v_2)$. □

2.2 Problem Formulation

As mentioned in Section 1, the densest subgraph models, such as average-degree density [2, 18–24], find the subgraphs with the maximal average density and capture the density of the structure well. However, they do not take into account the tightness of the interaction time, resulting in a sub-optimal solution. Inspired by this, to mine a densely-connected and tightly-interacting cohesive temporal subgraph, we incorporate time intervals into the inner temporal characteristics of temporal graphs. One natural way is to extend the average degree to the average number of temporal edges induced per node. However, the model may tend to find the subgraphs with a larger time span. Considering the example in Fig.1, H_2 is the subgraph with the maximal average-temporal-degree, in which the phone calls between vertices still span a longer gap than H_3, which has more edges in unit time. The reason is that the number of temporal edges increases with time (monotonic-increasing). Therefore, the average number of temporal edges induced per node of H_2 is larger. However, edges of subgraphs (e.g., H_3) occurring in a short time may be more significant. Based on this observation, we introduce the average-temporal-degree per unit time (cdensity) as an indicator of cohesive temporal subgraphs. And the definition is as follows.

Definition 3 (Temporal Densest Subgraph, TDS). *Given a temporal graph $\mathcal{G}$ and a positive integer L, we say a temporal subgraph $\mathcal{G}_{(S,I)}$ is the temporal densest subgraph of $\mathcal{G}$ if these following conditions are satisfied: 1) $|I| \geqslant L$; 2) there is no other node set $S' \subseteq V$ such that $cdensity(\hat{\mathcal{G}}_{(S',I)}) > cdensity(\hat{\mathcal{G}}_{(S,I)})$; 3) there is no other interval I' such that $|I'| \geqslant L$ and $cdensity(\hat{\mathcal{G}}_{(S,I')}) > cdensity(\hat{\mathcal{G}}_{(S,I)})$.*

Condition 1 is a limit on the time span of the TDS solution. It is practical and essential because a subgraph with a large average-degree but a very small time span can also have a large cdensity. Such temporal subgraphs may not be really cohesive, that is, they may not be cohesive in a real situation. For instance, in Fig. 2(a), if there is no limit on the time span, then its snapshot G_3 whose cdensity is 1.33 is the solution. This is unreasonable as it only has strong connections in structure, but any two nodes interact at most once, violating the "tightly-interacting" property of cohesive temporal subgraphs. Besides, the temporal subgraphs have the largest cdensity but their time span is less than the limit, and they may not be cohesive in a real situation. The reason is that they violate users' personalized time constraints (which only keep cohesive in a shorter time interval). In this way, some unqualified temporal subgraphs can be pruned using this limit. Condition 2 and condition 3 require the vertex set and the time interval to be optimal, respectively, wherefore we can find the cohesive temporal subgraph with the maximum cdensity.

Example 1. Considering the temporal graph $\mathcal{G}$ in Fig. 2(a), its static graph G is displayed in Fig. 2(b). According to the definition of the densest subgraph, G itself is a densest subgraph [2] whose density is 1.67. By Definition 3, when $L = 3$, we can find that the temporal subgraph $\mathcal{G}_1$ in Fig. 2(c) is the temporal densest subgraph of $\mathcal{G}$, where $S = \{1, 2, 3, 4, 5\}$, $I = [1, 3]$ and $cdensity(\hat{\mathcal{G}}_1) = 1.2$.

Regarding TDS, we have the following proposition.

Proposition 2. *For a temporal graph $\mathcal{G}$ and its temporal densest subgraph $\mathcal{G}_{(S,I)}$, we let d_l be the minimum weighted degree of $\hat{\mathcal{G}}_{(S,I)}$, and then $cdensity(\hat{\mathcal{G}}_{(S,I)}) \leqslant d_l$.*

Proof. Let $\hat{S} = \hat{\mathcal{G}}_{(S,I)}$. Assuming $\exists v \in S$ and $d_{\hat{S}}(v) = d_l < cdensity(\hat{S}) = W(\hat{S})/|S||I|$. Let $S' = S\backslash v$, $\hat{S}' = \hat{\mathcal{G}}_{(S',I)}$, and then

$$\begin{aligned} & cdensity(\hat{S}') \\ = & \frac{W(\hat{S}')}{|S'||I|} = \frac{W(\hat{S}) - d_{\hat{S}}(v)}{(|S|-1)|I|} > \frac{W(\hat{S}) - \frac{W(\hat{S})}{|S||I|}}{(|S|-1)|I|} \\ = & \frac{\frac{W(\hat{S})(|S||I|-1)}{|S||I|}}{(|S|-1)|I|} = \frac{\frac{W(\hat{S})(|S||I|-1)}{|S||I|}}{|S||I|-|I|}. \end{aligned}$$

Since $|I| \geqslant 1$, then $|S||I| - 1 \geqslant |S||I| - |I|$, and thus $cdensity(\hat{S}') \geqslant cdensity(\hat{S})$ conflicts with preconditions that S is the temporal densest subgraph. □

In other words, Proposition 2 shows that the minimum weighted degree of a TDS should be maximal among all temporal subgraphs containing the TDS. Based on this, we can approach the TDS problem by pruning the nodes whose weighted degree is less than a given value. In this way, the process to search TDS can speed up.

Our temporal densest subgraph model (Definition 3) is significantly different from previous densest subgraphs [2, 18–24]. Specifically, cdensity can capture the structural and the dynamic characteristics of temporal graphs by incorporating density and the time interval. Next, we show these using two metrics for temporal cohesive subgraphs: edge density burstiness [9] (EDB in short) and temporal conductance [25] (TC in short). EDB denotes the speed that a temporal subgraph accumulates edge density. For a vertex set S and the time interval I, $EDB = |\{(u, v, t)|u, v \in S, (u, v, t) \in E, t \in I\}|/|S||S-1||I|$. $TC = Tcut(S, V\backslash S)/\min\{Tvol(S), Tvol(V\backslash S)\}$, in which $Tcut(S, V\backslash S) = |\{(u, v, t)|(u, v, t) \in E, u \in S, v \in V\backslash S, t \in I\}|$ is the set of the edges between S and $V\backslash S$. $Tvol(S) = |\{(u, v, t)|(u, v, t) \in E, u \in S, v \in V, t \in I\}|$, namely all edges induced by S during I. TC reveals the separability of S from the rest of a temporal graph. Generally, the subgraph with a smaller TC has more inner connections and less outgoing connections. As Fig.3 shows, TDS and TDS- (TDS without considering the monotonic-increasing of temporal edges) have better performance than DS in both EDB and TC. In other words, taking the time interval into account improves the connection strength and weakens the separability of the solution. Besides, we can learn that considering the monotonic-increasing of temporal edges makes the solution preferable to ignoring it. In the following, we will define the TDS discovery problem.

Our Problem—Discovering Temporal Densest Subgraph (TDS). Given a temporal graph $\mathcal{G}$ and an integer L, the TDS problem aims to find a temporal densest subgraph $\mathcal{G}_{(S,I)}$ from $\mathcal{G}$, where $|I| \geqslant L$.

Although the temporal densest subgraph is similar to the densest subgraph [2] in a sense, we cannot directly solve our problem using the methods for the densest subgraph. The reason is that our model has to consider edge variance over a time span instead of only the degree. Therefore, it is much more time-consuming to find a naive TDS (refer to Section 3). Fortunately, some convex properties exist to facilitate the design of the efficient algorithm (refer to Section 4).

3 Baseline Algorithm

When given the time interval I, the TDS problem for the temporal subgraph $\mathcal{G}_I$ of $\mathcal{G}$ is actually to find a

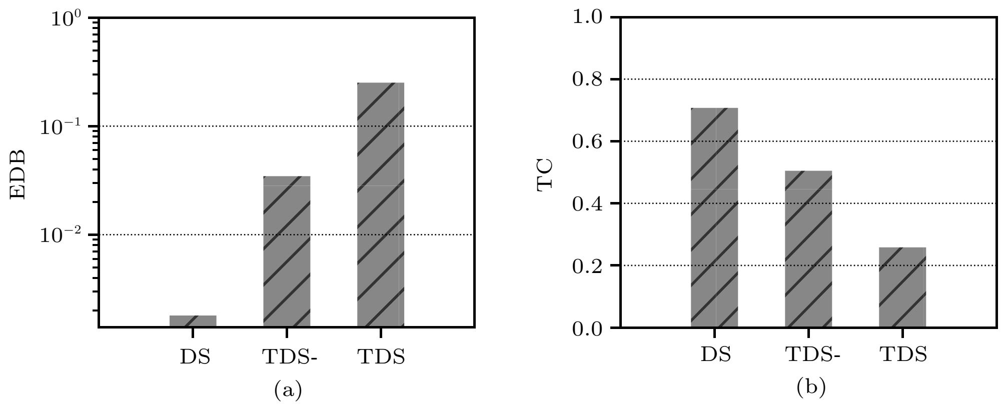


Fig.3. EDBs, and TCs of cohesive subgraph models on Rmin[2]. Compared with DS (the densest subgraph), and TDS- (TDS without considering the monotonic-increasing of temporal edges), TDS (the temporal densest subgraph) has the largest EDB and the smallest TC. (More details are shown in Section 5.) (a) EDB. (b) TC.

densest-like subgraph for $\hat{\mathcal{G}}_I$. An intuitive baseline algorithm (BA in short) for it is outlined in Algorithm 1. We search for the temporal densest subgraph in all possible time sub-intervals longer than L (lines 2–9), i.e., $|I(\varGamma)|\times(|I(\varGamma)|+1)/2$ time intervals. For each fixed time interval $I=[t_s,t_e]$, the algorithm Temp-Densest-Exact adapted from the algorithm in [2] is employed to return a densest-like subgraph for $\hat{\mathcal{G}}_I$ (line 6). Finally, Algorithm 1 returns the densest temporal subgraph from $|I(\varGamma)|\times(|I(\varGamma)|+1)/2$ candidate subgraphs.

Algorithm 1. Baseline Algorithm (BA)

```
Input: temporal graph $\mathcal{G}=(V,E,\varGamma)$, and integer $L$;
Output: subgraph $S$ and interval $I$;
 1: Initial $S=\emptyset$ , $d^*=0$;
 2: for $t_s \leftarrow 1$ upto $\lceil\varGamma\rceil-L+1$ do
 3:     for $t_e \leftarrow t_s+L-1 : \lceil\varGamma\rceil$ do
 4:         $\hat{\mathcal{G}}_{[t_s,t_e]} = Aggregate(t_s,t_e)$;
 5:         Execute the MWDP strategy for $\hat{\mathcal{G}}_{[t_s,t_e]}$;
 6:         $H = Temp\text{-}Densest\text{-}Exact(\hat{\mathcal{G}}_{[t_s,t_e]})$;
 7:         if $d^* < cdensity(\hat{\mathcal{G}}_{(H,[t_s,t_e])})$ then
 8:             $S=H, I=[t_s,t_e]$;
 9:             $d^* = cdensity(\hat{\mathcal{G}}_{(H,[t_s,t_e])})$;
10: return the subgraph $S$ and the interval $I$;
11: procedure $Aggregate(t_s,t_e)$
12:     for $(u,v)$ in $G_{t_e}$ do
13:         $d_{\hat{\mathcal{G}}_{[t_s,t_e]}}(u) = d_{\hat{\mathcal{G}}_{[t_s,t_{e-1}]}}(u)+1$;
14:         $d_{\hat{\mathcal{G}}_{[t_s,t_e]}}(v) = d_{\hat{\mathcal{G}}_{[t_s,t_{e-1}]}}(v)+1$;
```

However, Temp-Densest-Exact is very costly due to the high overhead in the calculation of the maximum flow. Therefore, according to Proposition 2, we design a pruning strategy—Minimum Weighted Degree Pruning (MWDP) to exclude the impossible nodes. Concretely, we record a current optimal solution S with $cdensity(\hat{S})=d^*$ during the search process (lines 7–9). Based on Proposition 2, if temporal subgraph $\hat{\mathcal{G}}_I$ contains a TDS (let it be H) that is more optimal than S, then for $\forall v\in\hat{\mathcal{G}}_I/H$, their weighted degree must be larger than d^*. And thus we can quickly and safely skip those nodes without searching (line 5). After that, the Temp-Densest-Exact algorithm is called for a pruned subgraph to return a sub-solution (line 6) for the current time interval.

Theorem 1. *The baseline algorithm runs within $O(|I(\varGamma)|^2\times|V||\bar{E}|\log(|V|^2/|\bar{E}|))$ time.*

Proof. The complexity of Temp-Densest-Exact is $O(|V||\bar{E}|\log(|V|^2/|\bar{E}|))$. It is called for $|I(\varGamma)|^2$ candidate time intervals. Additionally, procedure *Aggregate* has to access all the temporal edges $|I(\varGamma)|$ times. Therefore, the complexity of BA is $O(|I(\varGamma)|^2\times|V||\bar{E}|\log(|V|^2/|\bar{E}|))$. □

Obviously, BA has quadratic time complexity relative to the time span, and even for a fixed time sub-interval it spends $|V||\bar{E}|\log\frac{|V|^2}{|\bar{E}|}$ time. In the worst case, it is time-consuming if both the time span and the graph are large. Besides, there are $|I(\varGamma)|^2$ time sub-intervals, leading to $|I(\varGamma)|^2$ candidate subgraphs. Therefore, it is prohibitively expensive on both the memory consumption and the running time to choose the best one from $|I(\varGamma)|^2$ candidate subgraphs. In order to speed up the densest temporal subgraph mining, the temporal density aware algorithm is designed in Section 4.

4 Temporal Density Aware Exploration Algorithm

A temporal density aware algorithm is designed to speed up the procedure of mining the temporal dens-

[2]http://konect.cc/networks/mit/, Aug. 2021.

est subgraph (TDS), which consists of two stages. At the first stage, we analyze the features of temporal networks and identify the candidate time intervals where the temporal densest subgraphs fall in. Then, a greedy algorithm with a new proposed pruning technique is adapted to explore the densest subgraphs in the possible target time intervals.

4.1 Candidate Dense Time Intervals Extraction

A temporal graph is actually a temporal sequence of snapshots. Some snapshots have more edges while fewer edges occur in other snapshots. To some extent, a time interval including snapshots with more edges is denser than the interval including snapshots with fewer edges. For example, in traffic peaks (e.g., morning peak and evening peak), traffic networks tend to contain temporal cohesive subgraphs. Due to it, we try to extract some good candidate intervals by discovering evolving trends between sequences of snapshots. By this way, we do not need to search all permutations of time intervals to find the temporal densest subgraph.

We model the temporal sequence of snapshots as the edge distribution cumulative function that describes how many edges occur before timestamp t. Let $S(t) = \frac{1}{2}\sum_{v\in V} d_{\hat{\mathcal{G}}_{[1,t]}}(v) = \sum_{x\in[1,t]} deg(x)$. It is easy to know $S(t)$ is a strictly increasing function. To this end, it is more useful to investigate the increase of edges in a given interval, e.g., $I:(t_l,t_u]$.

$$f(I:(t_l,t_u]) = \frac{S(t_u)-S(t_l)}{|I|} = \frac{S(t_u)-S(t_l)}{|t_u-t_l|},$$

where $f((t_l,t_u])$ denotes the slope of $S(t)$ over the interval $[t_l,t_u]$, that is, $f((t_l,t_u]) = \mathit{slope}(S(t_l),S(t_u))$. According to Definition 2, it is surprised to find that:

$$f(I:(t_l,t_u]) = \frac{W(\hat{\mathcal{G}}_I)}{|I|}.$$

In other words, our condensed density is highly relevant to the slope of $S(t)$. Therefore, the interval with a greater slope of $S(t)$ may be a better candidate interval for the TDS problem. Inspired by this, we try to extract k time intervals with the top-k value of function f, and take them as the target intervals to solve the TDS problem. An intuitive way to find the top-k time intervals is to calculate and compare the slope of $S(t)$ for all permutations of time sub-intervals. As mentioned in Section 3, it is time-consuming. It is known that the slope of the tangent line to a convex function g increases in a non-negative manner[26]. That is, if $\mathcal{S} = \{(0,S(0)),\ (1,S(1)),\ ...,\ (t,S(t)),\ ...,\ (\lceil\mathit{\Gamma}\rceil,S(\lceil\mathit{\Gamma}\rceil))\}$ is a convex sequence of condensed graphs, we can easily determine the sub-sequence ending at $(t,S(t))$ (for simplicity, we denote $(t,S(t))$ as s_t) with the maximum slope. In this way, it is possible to compare only $|I(\mathit{\Gamma})|$ time intervals and extract the top-k from these intervals. Unfortunately, $\mathcal{S}$ is not convex. Thus, we construct a convex sequence $\mathcal{P}$ by adding points from $\mathcal{S}$ and maintain its convexity dynamically. Concretely, some convex nodes may turn to concave after adding a new node. Therefore, to maintain the convexity of $\mathcal{P}$, it is necessary to check whether it is still convex when a new node is added. If not, we remove concave points so that the sequence is convex. The process is sketched in ITKT (Algorithm 2). In the following, we will discuss ITKT in detail.

According to Definition 3, the time span of the temporal densest subgraph is not less than L. For each point $s_t \in \mathcal{S}$, where $t \geqslant L$, if we find the sequence $\{s_{e_t},...,s_t\}$ that starts with point s_{e_t} has the maximum slope, namely $e_t = \{x|f((x,t]) = \max(f((x_0,t]),x_0 \in [0,t-L])\}$. Then, we can get the corresponding time interval $(e_t,t]$ which has the maximum value of function f in all intervals ending with t. To quickly find such a sequence for each point $s_t \in \mathcal{S}$, we maintain a convex point sequence $\mathcal{P}$ for s_t, and let it end with s_{t-L}. Before adding the point s_{t-L} to $\mathcal{P}$, we consider the following situations. First, if $\mathcal{P}$ is empty, point s_{t-L} will be added directly to $\mathcal{P}$. If $\mathcal{P}$ is not empty and $\mathcal{P}$ is still convex after adding s_{t-L}, then it will also be inserted into the sequence directly. In many cases, after adding s_{t-L}, $\mathcal{P}$ is not convex again. Therefore, we need to remove some nodes from $\mathcal{P}$ to keep its convexity (line 12 in Algorithm 2). In more details, if $\mathit{slope}(\mathcal{P}_j,s_{t-L}) \leqslant \mathit{slope}(\mathcal{P}_{j-1},\mathcal{P}_j)$, Algorithm 2 will remove the concave node $\mathcal{P}_j$ until there is no concave point (lines 7–12), where j is the end index of $\mathcal{P}$, and $\mathcal{P}_j$ denotes the end point of $\mathcal{P}$. In this way, we can maintain a convex sequence $\mathcal{P}$ for current point s_t. Second, to find the corresponding interval $(e_t,t]$, we need to check the points again in $\mathcal{P}$ from $\mathcal{P}_i$, where i is the first index of $\mathcal{P}$, and $\mathcal{P}_i$ denotes the first point of $\mathcal{P}$. Specifically, we compute the slope of the sub-sequence $\{\mathcal{P}_i,...,\mathcal{P}_j,s_t\}$. Since $\mathcal{P}$ is convex, if $\mathit{slope}(\mathcal{P}_i,s_t) \geqslant \mathit{slope}(\mathcal{P}_i,\mathcal{P}_{i+1})$, then $\mathcal{P}_i$ cannot maximize the slope of the sub-sequence ending with s_t, and thus Algorithm 2 will remove the point $\mathcal{P}_i$ (line 19) and update $\mathcal{P}$ to check again until the maximum slope is obtained (lines 14–19). Finally, we can get the start point $s_{e_t} = \mathcal{P}_i$.

Algorithm 2. Identifying the Top-k Time Intervals (ITKT)

```
Input: temporal graph $\mathcal{G} = (V, E, \Gamma)$, and the integers $L$ and $k$;
Output: top-$k$ time intervals;
1: $\mathcal{P}$ is a convex sequence, $\mathcal{P} = \emptyset$;
2: $y = S(t)$ is the edge distribution function, and $s_t$ denotes the
   node in $\mathcal{S}$, and $\mathcal{S}$ is the node sequence for $S(t)$;
3: let $\mathcal{I}$ be a maximum heap to save $k$ candidate intervals;
4: $AccumulateDeg(\mathcal{G})$;
5: Initial $i = 0, j = 0$;
6: for $t \leftarrow L$ upto $\lceil \Gamma \rceil$ do
7:     while $(j - i) > 1$ do
8:         $slope1 \leftarrow slope(\mathcal{P}_j, s_{t-L})$;
9:         $slope2 \leftarrow slope(\mathcal{P}_{j-1}, \mathcal{P}_j)$;
10:        if $slope1 > slope2$ then
11:            break;
12:        $j \leftarrow j - 1$;
13:    $j \leftarrow j + 1$; $\mathcal{P}_j \leftarrow s_{t-L}$;
14:    while $(j - i) > 1$ do
15:        $slope3 \leftarrow slope(\mathcal{P}_i, s_t)$;
16:        $slope4 \leftarrow slope(\mathcal{P}_i, \mathcal{P}_{i+1})$;
17:        if $slope3 < slope4$ then
18:            break;
19:        $i \leftarrow i + 1$;
20:    if $\mathcal{I}.size() < k$ then
21:        $\mathcal{I}.push(\mathcal{P}_i, s_t, slope(\mathcal{P}_i, s_t))$;
22:        continue;
23:    if $\mathcal{I}.top().slope < slope(\mathcal{P}_i, s_t)$ then
24:        $\mathcal{I}.pop()$;
25:        $\mathcal{I}.push(\mathcal{P}_i, s_t, slope(\mathcal{P}_i, s_t))$;
26: return $\mathcal{I}$;
27: procedure $AccumulateDeg(\mathcal{G})$)
28:    for $t \leftarrow 1$ upto $\lceil \Gamma \rceil$ do
29:        $S(t) = S(t) + |E_{G_t}|$;
30: procedure $slope(s_i, s_j)$
31:    return $(S(j) - S(i))/(j - i)$;
```

Having computed the corresponding interval for the current candidate point s_t in $\mathcal{S}$, we update the candidate intervals based on the current value of $slope(s_{e_t}, s_t)$ (lines 20–25) to extract k time intervals with the top-k value of function f. Actually, the experiment conducted in Section 5 (experiment 7) confirms that the top-k intervals returned by ITKT are good candidates for the TDS problem.

Example 2. Considering the temporal graph $\mathcal{G}$ in Fig. 2(a), and letting $L = 2$, we show the process of identifying the top-2 time intervals by the ITKT algorithm in Fig. 4. It is easy to get $\mathcal{S} = \{(0,0),(1,6),(2,11),(3,19),(4,26),(5,29),(6,34)\} = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$. Since $L = 2$, the first candidate point is s_2. As $\mathcal{P}$ is empty, we directly add s_0 to $\mathcal{P}$, then $\mathcal{P} = \{s_0\}$, $s_{e_2} = s_0$, and $slope(s_0, s_2) = 5.5$. For s_3, adding s_1 would not change the convexity of $\mathcal{P}$, and thus $\mathcal{P} = \{s_0, s_1\}$. However, $slope(s_0, s_3) = (19 - 0)/(3 - 0) = 6.33$ is greater than $slope(s_0, s_1) = (6 - 0)/(1 - 0) = 6$, which indicates that s_0 cannot maximize the slope. Therefore, s_0 is removed from $\mathcal{P}$, and then $\mathcal{P} = \{s_1\}$, $s_{e_3} = s_1$ and $slope(s_1, s_3) = 6.5$. In terms of s_4, there are similar processes for s_4 and s_3, and then $\mathcal{P} = \{s_2\}$. Thus, we can get that $s_{e_4} = s_2$ and $slope(s_2, s_4) = 7.5$. As for s_5, point s_3 is directly added to $\mathcal{P}$, and then $slope(s_2, s_5) < slope(s_2, s_3)$, and thus $\mathcal{P} = \{s_2, s_3\}$. Therefore, $s_{e_5} = s_2$ and $slope(s_2, s_5) = 6$. When dealing with s_6, there is a

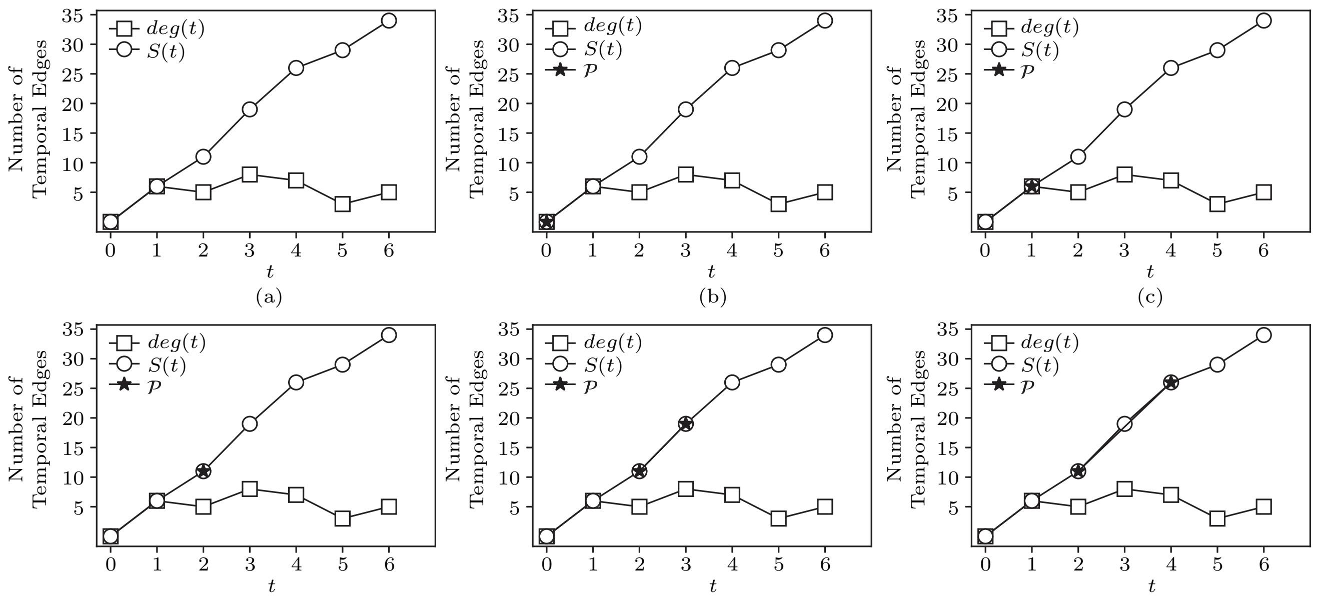


Fig. 4. Running example of ITKT on Fig. 2(a) with $L = 2$. $S(t)$ is the edge distribution cumulative function denoting how many edges occur before timestamp t, and its corresponding point sequence is $\mathcal{S} = \{(0,0),(1,6),(2,11),(3,19),(4,26),(5,29),(6,34)\}$, briefly denoted as $\{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$. $\mathcal{P}$ describes the constructed concave point sequence. (a) Curves of $S(t)$ and $deg(t)$. (b) $t = 2$ (s_2). (c) $t = 3$ (s_3). (d) $t = 4$ (s_4). (e) $t = 5$ (s_5). (f) $t = 6$ (s_6).

fact that $slope(s_3, s_4) = 7$ is less than $slope(s_2, s_3) = 8$, which means that s_3 would become a concave node after adding s_4 to $\mathcal{P}$, and thus we need to remove s_3 to protect the convexity of $\mathcal{P}$. Finally, we obtain a set $\mathcal{I}$ including two intervals with the top-2 slope of $S(t)$, where $\mathcal{I} = \{(1, 3], (2, 4]\}$.

Theorem 2. *The time complexity of Algorithm 2 is $O(|I(\Gamma)|)$.*

Proof. 1) Computing $y = S(t)$ chart line costs $O(|I(\Gamma)|)$ time. 2) Algorithm 2 takes $O(|I(\Gamma)|)$ time to get the top-k time intervals. Specifically, for a candidate node s_t, Algorithm 2 needs to remove concave nodes and delete unsatisfactory nodes. In general, all nodes may be convex and each node will be deleted at most once as a concave node (or an unsatisfactory node), and thus Algorithm 2 takes $O(|I(\Gamma)|)$ time to get the start point s_{e_t} for each $s_t \in \mathcal{S}$. And then, Algorithm 2 needs $O(|I(\Gamma)| \times \log k)$ time to get the top-k time intervals, where k is a small constant, and thus the time complexity of Algorithm 2 is $O(|I(\Gamma)|)$. $\square$

4.2 Greedy Search with a Pruning Strategy

After analyzing the features of the temporal graph, we can extract a set $\mathcal{I}$ including k candidate time intervals. For each $I \in \mathcal{I}$, we first compute the condensed graphs $\hat{\mathcal{G}}_I$ of $\mathcal{G}_I$. And then we employ an iterative algorithm to greedily find a densest-like subgraph (GFDS). GFDS borrows some ideas from the greedy Peeling algorithm [21] that is an approximation algorithm for mining the densest subgraphs on static networks. Inspired by this, following Proposition 1, GFDS greedily deletes the node with the minimum weighted degree step by step, and then updates the cdensity of the new condensed subgraph. When the iteration terminates, we can get the subgraph with the largest cdensity. However, the iterative process may be costly. To speed up the search process, we first introduce Proposition 3.

Proposition 3. *Supposing I is a time interval, $\hat{\mathcal{G}}_I$ is the condensed graph of the subgraph $\mathcal{G}_I$ of $\mathcal{G}$, and then $cdensity(\hat{\mathcal{G}}_I) \leqslant \frac{\max(d_{\hat{\mathcal{G}}_I}(v))}{2|I|}$.*

It is easy to know Proposition 3 is true from the definition of cdensity. With Proposition 3, we can prune some unqualified vertices according to the maximum weighted degree. Concretely, for all candidate time intervals, we maintain the optimal solution with optimal cdensity as d^*. Let GFDS stop when the upper-bound of the current condensed graph is not greater than d^*. In other words, if $\max(d_{\hat{\mathcal{G}}_I}(v)) < 2d^*|I|$, there is no optimal solution for the current condensed graph. By this way, we can skip these unnecessary searches. In the following, we analyze the time complexity of our algorithm.

Theorem 3. *The time complexity of GFDS is $O(|\bar{E}| + |V| \log |V|)$.*

Proof. GFDS greedily deletes the nodes with the minimum weighted degree. For a subgraph of $\mathcal{G}$, at most $|V|$ nodes need to be removed. In this way, the delete operation needs $O(|V| \log |V|)$ time. In addition to this, GFDS accesses each static edge at most once to update the weighted degree, and thus the update operation needs $O(|\bar{E}|)$ time. Therefore, the GFDS algorithm takes $O(|\bar{E}| + |V| \log |V|)$ time for a given time interval. $\square$

5 Experimental Evaluation

Several comprehensive experiments are carried out to assess efficiency, scalability, and effectiveness of the proposed methods. These experiments are executed on a server with an Xeon 2.00 GHz and 256 GB memory running Ubuntu 18.04.

5.1 Experimental Setup

Datasets. We evaluate the proposed methods on nine real-word temporal networks with different types and sizes (Table 1). Specifically, Rmin[3], Primary[4], Lyon[5] and Thiers[5] are temporal face-to-face networks, in which each node represents a student (or teacher) and each temporal edge records the physical contact timestamp of the corresponding persons. Facebook[6] and Twitter[7] are both temporal social networks that record the interactions among users via social platforms. Enron[8] and Lkml[9] are temporal email networks for Enron company and Linux kernel,

[3] http://konect.cc/networks/mit/, Aug. 2021.
[4] http://www.sociopatterns.org/datasets/primary-school-temporal-network-data/, Aug. 2021.
[5] http://www.sociopatterns.org/datasets/co-location-data-for-several-sociopatterns-data-sets/, Aug. 2021.
[6] http://konect.cc/networks/facebook-wosn-wall/, Aug. 2021.
[7] http://snap.stanford.edu/data/higgs-twitter.html, Aug. 2021.
[8] http://konect.cc/networks/enron-rm/, Aug. 2021.
[9] http://konect.cc/networks/lkml-reply/, Aug. 2021.

respectively. DBLP[10] is a million-vertex temporal collaboration network.

Table 1. Dataset Statistics

Dataset	$\|V\|$	$\|E\|$	$\|\bar{E}\|$	$\|I(\Gamma)\|$	TS
Rmin	96	76 551	2 539	5 576	Hour
Primary	242	26 351	8 317	20	Hour
Lyon	242	218 503	26 594	20	Hour
Thiers	328	352 374	43 496	50	Hour
Twitter	304 198	464 653	452 202	7	Day
Facebook	45 813	461 179	183 412	223	Week
Enron	86 978	697 956	297 456	177	Week
Lkml	26 885	328 092	159 996	98	Month
DBLP	1 729 816	12 007 380	8 546 306	72	Year

Note: TS is the time scale of the timestamp.

Models. We choose several state-of-the-art competitors that are most similar to our problem. Densest-Exact [2] is an exact algorithm that can mine the densest subgraph from non-temporal networks. KGAPPROX [8] dynamically searches temporal subgraphs with the maximum total density. OL [9] models the bursting temporal subgraphs according to their density and duration. FIDES$^+$ [10] finds a dense temporal subgraph with the heaviest total weight. To make our datasets adapt to FIDES$^+$, we assign the weight of an edge as $+1$ if it exists at that timestamp; otherwise, we set its weight as -1. MiMAG [11] aims to mine the clusters of vertices that are densely connected by edges with similar labels. To accommodate its input, we treat each timestamp as an attribute. DCCS [12] and FirmCore [13] are both defined based on k-core, and explore the densest subgraph with different constraints in multilayer networks. GFDS+TQ and GFDS+ITKT are our methods which differ in the volume of time intervals to search in GFDS. In the former, GFDS runs on $|I(\Gamma)|^2$ time intervals while the later only executes GFDS on k time intervals. According to the experimental situation, comprehensively considering all datasets, parameters L and k are set to 6 and 10, respectively (unless otherwise specified, their values are not changed). All these algorithms are implemented in C++[11].

Experiment 1: *Running Time of Various Temporal Methods on Different Datasets.* Table 2 reports the running time of eight temporal methods on nine datasets. Clearly, GFDS+ITKT is consistently faster than the other methods on most datasets. It only costs less than five seconds on most datasets except DBLP, where it requires less than two minutes. GFDS+TQ ranks the third and is slightly slower than the heuristics algorithm FIDES$^+$. FirmCore, DCCS and MiMAG take more time than GFDS+TQ as they need to search for the base model (core, clique) and integrate the results to return the solution. OL ranks the sixth because it is NP-hard and runs several iterations to return the solutions. As for KGAPPROX, it gets the worst performance, because it dynamically calculates the time intervals to obtain the optimal solutions. These results suggest that the proposed pruning strategies (in Section 4) are effective in practice.

Experiment 2: *Scalability.* We generate the synthetic networks by resampling nodes and timestamps to evaluate the scalability of our methods. In more detail, we generate eight artificial networks by randomly selecting 20%, 40%, 60%, and 80% of the nodes or varying TS to change $|I(\Gamma)|$ on DBLP. The running time of GFDS+TQ and GFDS+ITKT is plotted in Fig.5. As it shows, the running time of GFDS+TQ varies nonlinearly when changing $|I(\Gamma)|$, as its time complexity is quadratic with the time span. But in other cases, the running time of GFDS+TQ and GFDS+ITKT varies near-linearly. Additionally, GFDS+ITKT performs better than GFDS+TQ, indicating that it can handle the large-scale temporal networks.

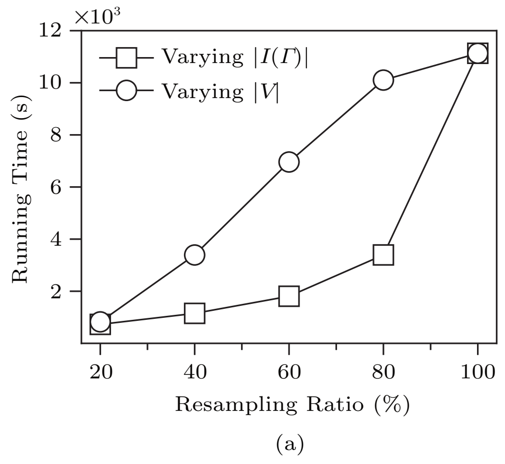
Table 2. Running Time (s) of Temporal Methods on Different Datasets

Dataset	KGAPPROX [8]	OL [9]	FirmCore [13]	MiMAG [11]	FIDES$^+$ [10]	DCCS [12]	GFDS+TQ	GFDS+ITKT
Rmin	526.708	47.121	4 112.281	470.670	1.065	$*$	463.201 2	**0.023 1**
Primary	425.443	69.515	0.959	12 300.100	0.318	4.094	0.693 7	**0.062 4**
Lyon	1 312.427	1 635.670	3.587	7 560.000	0.914	19.335	3.820 5	**0.380 2**
Thiers	3 111.475	2 077.790	16.084	8 340.370	1.402	175.428	17.001 2	**0.600 3**
Twitter	95 504.369	35 151.700	26.036	2 943.954	84.024	8.323	6.300 0	**3.340 1**
Facebook	127 741.870	51.213	2 413.352	6 389.240	18.643	3 796.355	1 974.004 1	**1.090 0**
Enron	$\geqslant$172 800.000	2 679.060	2 073.825	11 460.400	13.514	4 913.610	2 545.017 7	**2.210 1**
Lkml	32 014.886	3 130.130	448.820	31 860.371	10.727	1 638.877	286.011 0	**0.390 4**
DBLP	$\geqslant$172 800.000	2 345.710	8 080.973	$*$	**43.093**	5 236.058	11 132.614 1	111.600 1
Avg. rank	8	6	4	7	2	5	3	1

Note: $*$ denotes the methods cannot return a solution in 4 days. Avg. rank is the average rank of each method in the test datasets. The best result on the current dataset is in bold.

[10] http://konect.cc/networks/dblp_coauthor/, Aug. 2021.

[11] https://gitee.com/Dawn-snow/tds.git/, Jul. 2022.



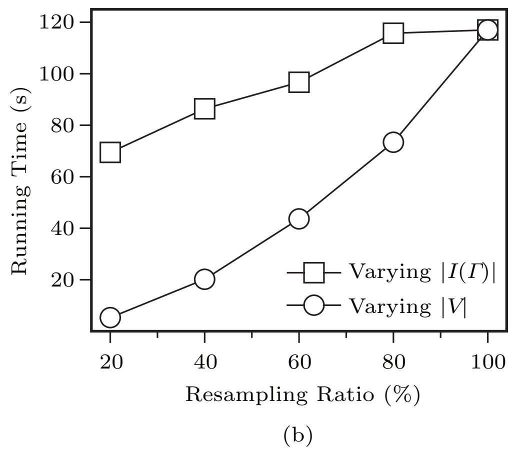


Fig. 5. Scalability test on DBLP (L = 6 and k = 10). (a) GFDS+TQ. (b) GFDS+ITKT.

Experiment 3: *Running Time with Varying k and L.* Figs.6(a)–6(d) display how the running time of GFDS+TQ and GFDS+TQ varies with L. Specifically, the running time of GFDS+TQ decreases as L increases. This is because a larger L results in fewer time intervals and thus fewer calls to GFDS. However, the running time of GFDS+ITKT is stable when L varies. Fig. 6(e) describes the effect of the parameter k on the running time for the GFDS+ITKT method. As expected, though a larger k requires a little more time to find a TDS as it calls GFDS more times, overall the running time is insensitive to k. In other words, the GFDS+ITKT method is stable regardless of varying k or L. Based on our update procedure Aggregate (in Algorithm 1), closer intervals would take less time to generate the temporal subgraph. For example, the temporal subgraph $\mathcal{G}_{[4,9]}$ can be obtained from $\mathcal{G}_{[3,8]}$, but $\mathcal{G}_{[10,15]}$ must be rebuilt. The results indicate that our two methods are insensitive to parameters.

Experiment 4: *Average Memory Consumption of Our Methods on Different Datasets.* Table 3 reports the average memory consumption of GFDS+TQ and GFDS+ITKT under the default parameter settings. Our two methods consume memory less than 1x–3x the raw size of input graphs except for Twitter, in which memory consumption is up to about 5x the raw size. Furthermore, GFDS+ITKT consumes less memory than GFDS+TQ on eight of the nine datasets. The

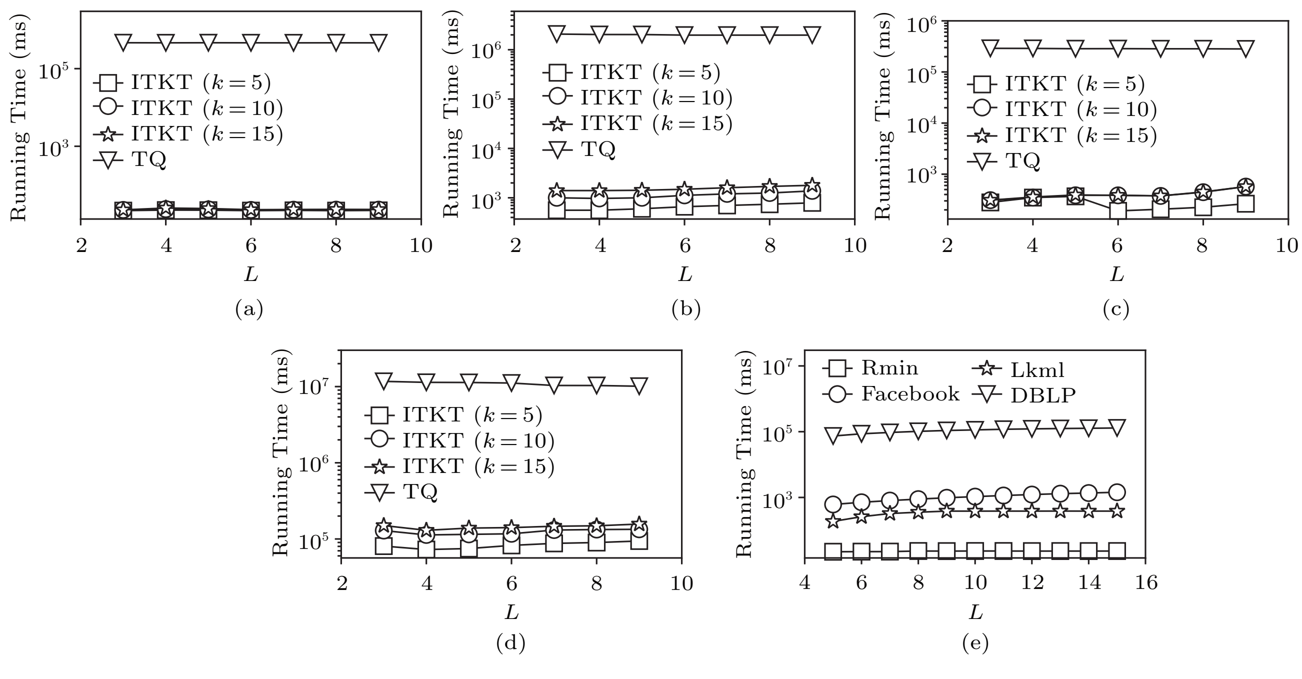


Fig. 6. Running time with varying parameters. (a) Varying L on Rmin. (b) Varying L on Facebook. (c) Varying L on Lkml. (d) Varying L on DBLP. (e) GFDS+ITKT's running time with varying k (L = 6).

Table 3. Average Memory Consumption (MB) of Our Methods on Different Datasets

Method	Rmin	Primary	Lyon	Thiers	Twitter	Facebook	Enron	Lkml	DBLP
Raw size	1.492 5	0.403 2	3.335 3	5.379 7	7.090 4	7.055 0	10.663 3	5.012 0	183.222 3
GFDS+TQ	1.578 0	0.688 6	4.212 5	6.807 8	34.229 0	14.858 6	23.708 4	11.391 4	528.111 7
GFDS+ITKT	1.928 4	0.604 6	4.121 4	6.235 4	34.075 4	9.604 5	19.893 6	6.038 2	381.848 7

Note: Raw size denotes the memory consumption of the original temporal graph.

reason is that GFDS+ITKT only executes GFDS for k time intervals, while GFDS+TQ needs to access all $|I(\mathit{\Gamma})|^2$ time intervals. On Rmin, GFDS+ITKT consumes a little more memory than GFDS+TQ, since it needs more memory to identify the top-k intervals. In a word, these results indicate GFDS+TQ and GFDS+ITKT can achieve near-linear space complexity.

5.2 Effectiveness

In this subsection, to evaluate the effectiveness of our methods and model, we first choose cdensity (refer to Definition 2) as a metric to evaluate whether our methods work well. Then, we employ edge density burstiness EDB[9] and temporal conductance TC[25] (refer to Subsection 2.2) as the common metrics.

Experiment 5: *Effectiveness of Various Algorithms on Different Datasets.* Table 4 reports the effectiveness of the eight methods on nine datasets. For the metric cdensity, GFDS+TQ outperforms the others on seven datasets and GFDS+ITKT is the runner-up. DCCS ranks the third, since it finds dense cores, with a large average temporal degree. KGAPPROX and FIDES$^+$ rank the fifth and the sixth, respectively, be-

Table 4. Effectiveness of Various Algorithms

Metric	Dataset	Densest-Exact [2]	KGAPPROX [8]	OL [9]	FirmCore [13]	MiMAG [11]	FIDES$^+$ [10]	DCCS [12]	GFDS+TQ	GFDS+ITKT
cdensity	Rmin	0.155 1	0.305 2	2.812 5	0.303 8	0.002 1	0.357 5	*	**3.522 2**	1.629 6
	Primary	5.363 0	5.658 2	2.750 0	5.374 8	1.329 3	5.823 9	4.676 6	7.406 3	**7.267 4**
	Lyon	45.028 0	54.000 0	42.006 6	44.101 3	7.484 4	46.056 0	48.532 8	**54.658 4**	**54.658 4**
	Thiers	21.619 4	18.666 9	39.297 5	20.490 7	6.255 0	23.292 8	15.479 4	**43.501 6**	**43.501 6**
	Twitter	2.402 0	**10.410 0**	1.500 0	2.020 2	1.227 4	0.097 8	3.971 7	3.125 0	3.125 0
	Facebook	0.198 3	0.371 5	1.312 5	0.292 8	0.002 8	0.354 8	1.105 7	**1.333 3**	1.083 3
	Enron	1.060 1	3.664 5	4.680 0	1.574 2	0.748 5	1.862 1	4.988 1	**6.862 6**	5.826 1
	Lkml	3.409 0	7.422 1	3.500 0	5.342 8	1.600 0	5.653 5	7.158 4	**9.533 3**	**9.533 3**
	DBLP	0.801 5	*	6.350 0	2.444 9	*	0.333 3	4.154 2	**29.230 3**	**29.230 3**
	Avg. rank	8	5	4	7	9	6	3	1	2
EDB	Rmin	0.001 8	0.004 4	**0.401 7**	0.037 8	0.000 3	0.357 5	*	0.251 5	0.203 7
	Primary	0.027 7	0.031 1	**0.275 0**	0.024 8	0.215 0	0.026 7	0.117 1	0.054 0	0.056 0
	Lyon	0.187 6	0.229 8	0.453 5	0.202 3	**0.499 0**	0.196 0	0.216 2	0.259 0	0.259 0
	Thiers	0.066 7	0.060 6	**0.499 1**	0.074 5	0.367 9	0.076 9	0.238 3	0.430 7	0.430 7
	Twitter	0.011 8	0.043 7	**0.375 0**	0.067 3	0.288 3	0.000 1	0.136 3	0.040 0	0.039 0
	Facebook	0.000 2	0.000 5	0.283 3	0.292 8	0.000 7	0.000 1	0.368 5	**0.444 4**	0.361 1
	Enron	0.001 1	0.037 0	**0.460 6**	0.021 9	0.187 1	0.002 0	0.038 3	0.294 0	0.264 8
	Lkml	0.008 3	0.040 6	**0.467 6**	0.082 2	0.400 0	0.049 6	0.052 2	0.113 5	0.113 5
	DBLP	0.025 1	*	**0.341 7**	0.006 9	*	0.166 6	0.000 9	0.083 5	0.083 5
	Avg. rank	9	8	1	6	2	7	5	3	4
TC	Rmin	0.707 4	0.793 4	0.466 7	0.544 8	0.917 3	**0.244 4**	*	0.258 3	0.432 3
	Primary	0.843 2	0.763 6	0.812 8	0.761 0	0.855 1	0.904 7	0.762 3	**0.673 6**	0.683 4
	Lyon	1.000 0	0.990 5	**0.665 3**	0.893 4	0.933 4	0.995 9	0.973 9	0.877 2	0.877 2
	Thiers	1.000 0	0.996 8	0.183 0	0.943 0	0.870 4	0.981 9	0.808 2	**0.146 4**	**0.146 4**
	Twitter	0.962 1	0.965 7	0.998 8	0.966 0	0.992 6	**0.720 4**	0.984 4	0.977 2	0.977 2
	Facebook	0.546 6	0.604 7	0.655 7	0.579 3	0.947 3	0.648 1	0.560 3	0.623 5	**0.434 8**
	Enron	0.943 7	0.757 4	0.904 5	0.861 9	0.973 1	**0.531 5**	0.845 0	0.756 3	0.813 1
	Lkml	0.731 9	0.720 5	0.964 6	0.800 2	0.981 3	0.726 2	0.774 2	**0.717 1**	**0.717 1**
	DBLP	0.110 3	*	0.793 2	0.110 3	*	**0.000 0**	0.789 8	0.053 5	0.0535
	Avg. rank	9	6	8	5	7	4	3	1	2

Note: $*$ denotes the methods cannot return a solution in 4 days. Avg. rank is the average rank of each method in the test datasets. The best result on the current dataset is in bold.

cause they prefer to return a solution having the large sum of density. Although FirmCore is also based on the core model, it considers a trade-off between the number of layers and the edge density. Densest-Exact and MiMAG have the lowest cdensity score on average. The reason is that MiMAG is a best-first search algorithm for mining multi-layer coherent subgraph (MLCS), in which the temporal layers in a coherent subgraph may not be continuous. Regarding EDB, OL and MiMAG are the top two methods (but have poor cdensity and TC). The reason is that they are clique-based dense subgraphs, which have more connections between their nodes. Thus, their subgraphs have a large EDB. Our methods rank the third and the fourth because they aim to find the densest temporal subgraph having both a high interaction frequency and a tighter interval. DCCS and FirmCore only concern about the frequency of the interaction. Therefore, it may require a long time to accumulate the structural density. FIDES$^+$ and KGAPPROX always return large subgraphs, and thus the average speed of edge accumulation is slow. In terms of the TC metric, GFDS+TQ reaches the top ranking and GFDS+ITKT ranks the second since our model can capture more accurate temporal proximity from interactions between nodes. DCCS, FirmCore and MiMAG do not perform as well as our model, because they tend to find the cohesive subgraphs in any subset of the layers, which may not be coherent and span a large gap. Densest-Exact's performance is the worst because its returned subgraphs are not distinctly separated from the rest of the network (e.g., its TCs are 1 in Lyon and Thiers). All in all, our model is effective for finding meaningful cohesive subgraphs which have more denser inner connections and less outgoing temporal interactions. Furthermore, our methods are effective for our model.

Experiment 6: *GFDS+ITKT's cdensity with Varying k and L.* Figs.7(a)–7(d) are heatmaps describing how the cdensity of GFDS+ITKT changes with k and L on four datasets. Our solution relies on L, and in most case, cdensity decreases as L increases. Besides, the sensitivity of cdensity to L varies with different temporal graphs. For example, when increasing L from 3 to 9, the cdensity changes at most by 13% in Lkml, but about 60% for DBLP. Furthermore, as L increases to 6, the change in cdensity decreases and gradually sta-

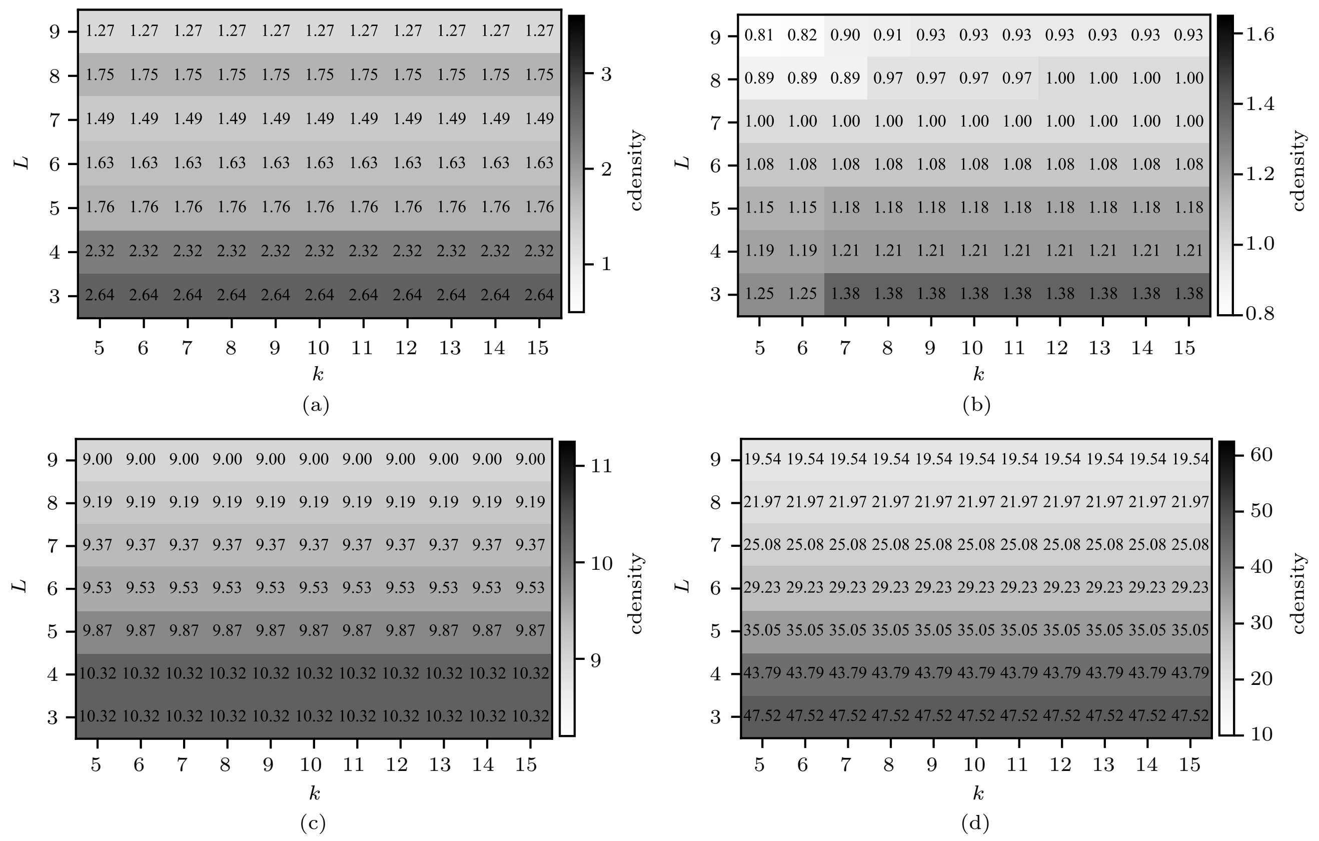


Fig.7. GFDS+ITKT's cdensity with varying k and L. (a) Rmin. (b) Facebook. (c) Lkml. (d) DBLP.

bilizes. Considering all datasets, we take $L = 6$ as the default setting. However, as for k, it is obvious that cdensity is independent of k, except that it changes about 10% on Facebook when $k < 12$. But taking all temporal graphs and efficiency into consideration, we regard $k = 10$ as the default setting. In summary, our GFDS+ITKT method is insensitive to k in cdensity.

Experiment 7: *Quality Comparison with the Exact Algorithm.* Fig.8 reports the ability of GFDS+TQ and GFDS+ITKT for solving the TDS problem. We employ BA (Algorithm 1) to derive the exact solution for the TDS problem. And then we compare BA's cdensity with GFDS+TQ's and GFDS+ITKT's. Since BA has poor scalability, we only report the results on two small-scale datasets Primary and Lyon. The following observations can be obtained. 1) The cdensity values of the subgraphs returned by GFDS+TQ and GFDS+ITKT are pretty close, and even become equal when L gets larger. 2) The quality of solutions returned by GFDS+ITKT is insensitive to k. 3) GFDS+TQ's and GFDS+ITKT's cdensity are very close to BA's cdensity. And no matter how the parameters k and L change, the cdensity of GFDS+ITKT is very close to the exact solution for the TDS problem. As a result, it proves that GFDS+ITKT can mine better candidate time intervals for the TDS problem. Besides, GFDS+TQ and GFDS+ITKT are comparable to TDS. The heuristic algorithm GFDS+ITKT works well to strike a balance between efficiency and quality.

Experiment 8: *Case Study on Rmin.* Fig.9(a) and Fig.9(b) show the results of OL and GFDS+ITKT on the Rmin dataset under their default settings, respectively. OL identifies seven persons and involves five different crowds (Fig.9(a)). Specifically, three students are "1styeargrad" and other persons are "sloan", "mlfrosh", "mlurop" and "mlgrad". GFDS+ITKT finds a subgraph (Fig. 9(b)) consisting of nine persons involving four different crowds, in which three persons are "mlgrad", four persons are "1styeargrad" and the other two persons are "sloan" and "mlfrosh". Intuitively, our model returns a larger subgraph but involves fewer identities. From this perspective, our model can find subgraphs with stronger connections in terms of crowds. Furthermore, there is a fact that graduate and undergraduate students have different courses. As a result, there may be more interactions within undergraduates (graduates). Based on this common sense, from Fig.9(b), we can see that the subgraph returned by GFDS+ITKT mainly consists of "mlgrad" and "1styeargrad" that are both "Graduate Student" and makes up 7/9 of the whole solution, which may be a reflection of the previous conjecture. Therefore, to a certain extent, it proves that our model is closer to the real situation. Besides, as we can see, OL returns a 7-clique with strong structural constraints. But a strict structural constraint may not always be good, since it may ignore some hidden meaningful patterns. However, in GFDS+ITKT's solution, although the two persons "sloan" and "mlfrosh" have no interactions, they are both in the final solution as they are closely associated with many of the same people. It has exhibited the potential relationship, which may be more interesting and meaningful. To be more convincing, we exhibit the solution for GFDS+ITKT with the parameter $L = 3$ in Fig.9(c) (as the solution is insensitive to k, we set k to the default value). As shown in Fig.9(c), there are similar observations with the solution under the default settings (Fig.9(b)). Specifically, when $L = 3$, though

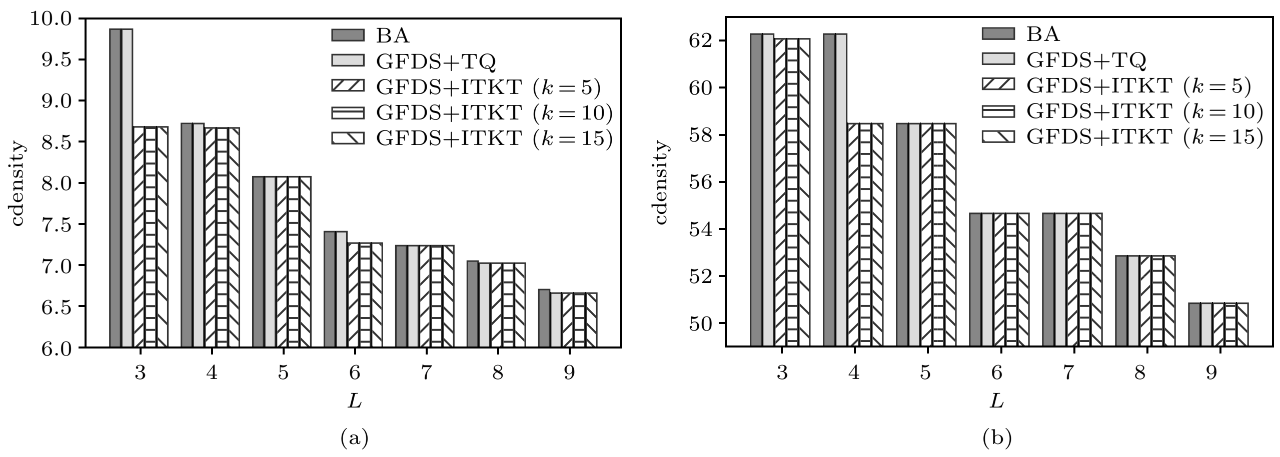


Fig.8. Quality comparison with exact algorithm. (a) Primary. (b) Lyon.

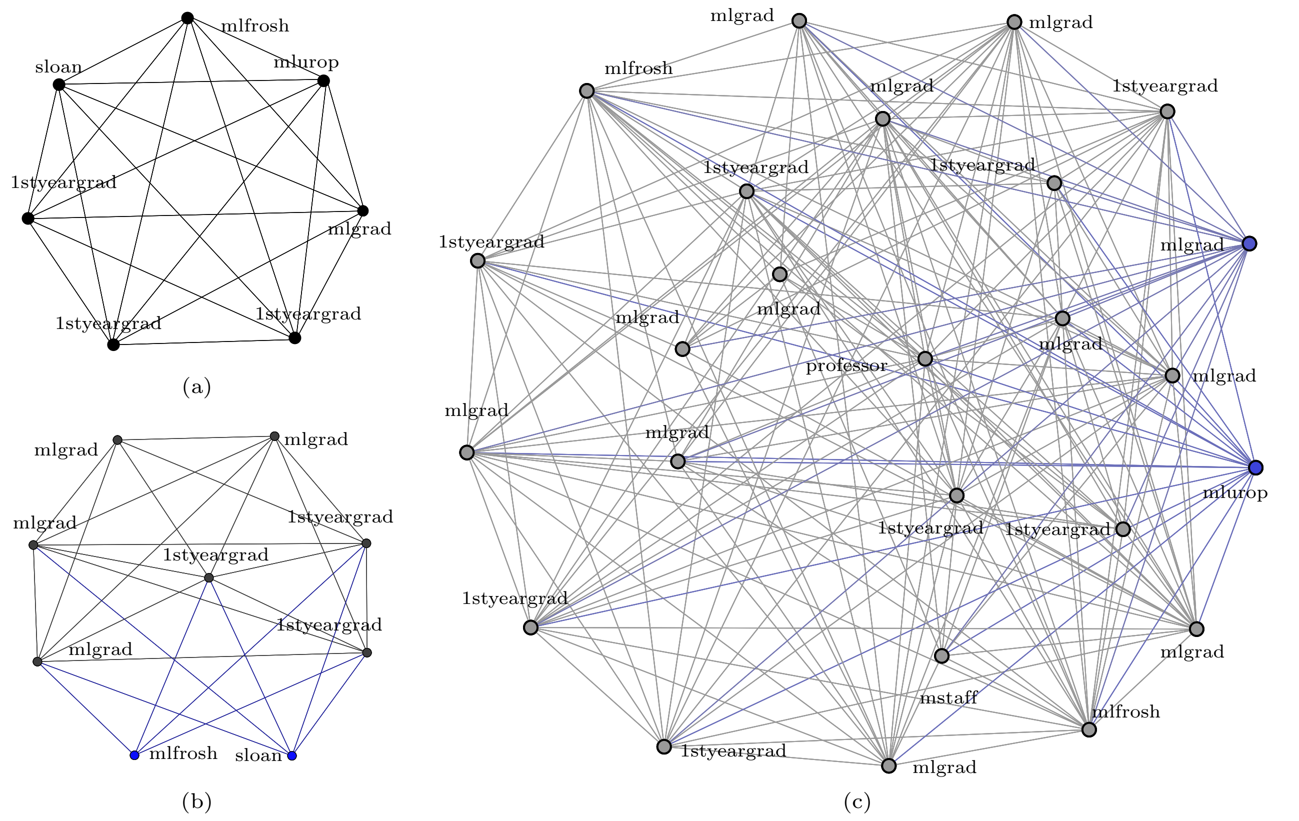


Fig.9. Case study on Rmin. "1styeargrad" and "mlgrad" represent first-year and not-first-year Media Lab graduate student respectively. "mlfrosh" and "mlurop" represent first-year and not-first-year Media Lab undergraduate student respectively. "sloan" represents the student of Sloan Business School, "professor" represents the Media Lab Professor, and "mlstaff" represents the Media Lab Staff (see more details in [27]). (a) OL. (b) GFDS+ITKT ($L = 6$ and $k = 10$). (c) GFDS+ITKT ($L = 3$ and $k = 10$).

there are more nodes in the solution (as it has a more lenient time constraint), its solution also mainly consists of "1styeargrad" and "mlgard" crowds. Furthermore, we can also find the meaningful hidden pattern, that is, two persons have no interactions but they are in the final solution (e.g., two nodes in blue in Fig.9(c)). Based on these observations, we can conclude that different parameters have no significant effect on the final results. These experimental results indicate that our model is more practical for finding meaningful patterns than OL.

6 Related Work

6.1 Cohesive Subgraph Mining

Many cohesive subgraph models have been proposed to suit many application scenarios. For instance, clique [28, 29], γ-quasi-clique [14, 15], k-core [16, 17] and k-truss [30], all these are common cohesive subgraph models. The closest one to our work is the densest subgraph model [2, 18–20]. Goldberg [2] applied the maximum flow technology to devise an exact solution with polynomial time. However, the computation of the maximum flow has a prohibitively high overhead and is therefore inefficient for massive real-world graphs. Charikar [21] developed a greedy Peeling algorithm which can generate a 2-approximation solution for the densest subgraph problem on the undirected static graphs. And he further designed a linear 2-approximation algorithm in directed graphs. Andersen and Chellapilla [22] studied to search the densest subgraph with a size restriction. Depending on whether the limit is set on the upper or lower boundary, it can be divided into the DalkS and DamkS problems, respectively. They found a 3-approximation solution for DalkS and demonstrated the difficulty of finding an approximate method for DamkS. Tsourakakis [19] studied to search a k-clique densest subgraph, which has the maximum average number of k-cliques participants. Epasto *et al.* [23] proposed approaches to dynamically keep the densest subgraph when some edges are deleted or added. Recently, a tutorial on densest subgraph computation has been presented in [24]. However, all these approaches are tailored for static graphs and thus ignore the temporal information of networks, resulting in sub-optimal results. Thus, some studies model the static graphs with different attributes (timestamps) as multilayer networks to find the cohesive subgraphs, for instance, finding top-k diversified cores covering the largest number of vertices [12], defining clusters of densely-connected vertices whose induced edges have the similar labels in a subset of layers [11], and extracting dense subgraphs sharing the same vertex set [13, 31] from a set of multiple graphs. However, these studies search for the subgraphs in discontinuous layers [32] and cannot capture the continuous temporal information.

6.2 Temporal Network Analysis

Nowadays, temporal networks are widely used to model real-life applications. For instance, Wu *et al.* [33] defined four different temporal path models by considering different temporal constraints, making these models more realistic. Besides, they also proposed an approach to keep the time-order by transforming temporal networks into de-temporal networks. In [34], Wu *et al.* presented an efficient indexing scheme TopChain equipped with the graph transformation approach of [33] to investigate the real time query of temporal reachability and time-based path. It is worth mentioning that TopChain is able to update the index quickly and dynamically. Rozenshtein and Gionis [35] explored the PageRank model for temporal graphs, in which the importance of vertices changes along with the time. Additionally, the classic problems of the vertex covering [36] and the graph coloring [37] also have been studied on temporal graphs. Recently, some studies have attempted to detect cohesive subgraphs in time-varying networks. Specifically, Ma *et al.* [10] explored how to find a heavy subgraph in a special class of temporal networks, whose structure remains unchanged but the weight of edges changes over time. Lin *et al.* [38] investigated the diversified lasting cohesive subgraphs on temporal networks. Li *et al.* [5] introduced a model called persistent community, and they also devised efficient algorithms for identifying a persistent community with the largest size. Qin *et al.* [39] did some research on the periodicity of cohesive subgraphs. Zhang *et al.* [40] studied the problem of identifying the significant engagement community to which the user-specified query belongs. However, the closest work to ours is [8, 9], which expanded the concept of static density and attempted to find temporal subgraphs with the maximum total density and the largest burstiness, respectively.

7 Conclusions

In this work, we proposed the novel model TDS and three algorithms BA, GFDS+TQ and GFDS+ITKT, which can efficiently mine more meaningful and practical temporal cohesive subgraphs. In efficiency tests, the heuristic algorithm GFDS+ITKT outperforms the others. And it takes less than two minutes to return a solution from the million-vertex DBLP. Besides, GFDS+ITKT gets the best average ranking when taking the well-known EDB and TC as the metrics. These experimental results revealed the importance of capturing the temporal characteristics and the limitations of using overly strong structural constraints. Considering all the factors mentioned above, TDS is more practical in finding the meaningful potential patterns when exploring the cohesive temporal subgraphs. In future work, we will try to extend the concept of the personalized queries to TDS to explore the potential of finding the cohesive temporal subgraphs with the specified query nodes.

References

[1] Chang L, Qin L. Cohesive subgraph computation over large sparse graphs. In *Proc. the 35th IEEE International Conference on Data Engineering*, April 2019, pp.2068-2071. DOI: 10.1109/ICDE.2019.00241.

[2] Goldberg A V. Finding a maximum density subgraph. Technical Report, University of California Berkeley, 1984. https://digicoll.lib.berkeley.edu/record/136696, July 2022.

[3] Rozenshtein P, Gionis A. Mining temporal networks. In *Proc. the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2019, pp.3225-3226. DOI: 10.1145/3292500.3332295.

[4] Lin L, Yuan P, Li R H, Wang J, Liu L, Jin H. Mining stable quasi-cliques on temporal networks. *IEEE Trans. Systems, Man, and Cybernetics: Systems*, 2022, 52(6): 3731-3745. DOI: 10.1109/TSMC.2021.3071721.

[5] Li R H, Su J, Qin L, Yu J X, Dai Q. Persistent community search in temporal networks. In *Proc. the 34th IEEE International Conference on Data Engineering*, April 2018, pp.797-808. DOI: 10.1109/ICDE.2018.00077.

[6] Yang Y, Yan D, Wu H, Cheng J, Zhou S, Lui J C S. Diversified temporal subgraph pattern mining. In *Proc. the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2016, pp.1965-1974. DOI: 10.1145/2939672.2939848.

[7] Bassett D S, Yang M, Wymbs N F, Grafton S T. Learning-induced autonomy of sensorimotor systems. *Nature Neuroscience*, 2015, 18(5): 744-751. DOI: 10.1038/nn.3993.

[8] Rozenshtein P, Bonchi F, Gionis A, Sozio M, Tatti N. Finding events in temporal networks: Segmentation meets densest subgraph discovery. *Knowledge and Information Systems*, 2020, 62(4): 1611-1639. DOI: 10.1007/s10115-019-01403-9.

[9] Chu L, Zhang Y, Yang Y, Wang L, Pei J. Online density bursting subgraph detection from temporal graphs. *Proceedings of the VLDB Endowment*, 2019, 12(13): 2353-2365. DOI: 10.14778/3358701.3358704.

[10] Ma S, Hu R, Wang L, Lin X, Huai J. Fast computation of dense temporal subgraphs. In *Proc. the 33rd IEEE International Conference on Data Engineering*, April 2017, pp.361-372. DOI: 10.1109/ICDE.2017.95.

[11] Boden B, Gunnemann S, Hoffmann H, Seidl T. MiMAG: Mining coherent subgraphs in multi-layer graphs with edge labels. *Knowledge and Information Systems*, 2017, 50(2): 417-446. DOI: 10.1007/s10115-016-0949-5.

[12] Zhu R, Zou Z, Li J. Diversified coherent core search on multi-layer graphs. In *Proc. the 34th IEEE International Conference on Data Engineering*, April 2018, pp.701-712. DOI: 10.1109/ICDE.2018.00069.

[13] Hashemi F, Behrouz A, Lakshmanan L V S. FirmCore decomposition of multilayer networks. In *Proc. the 2022 ACM Web Conference*, April 2022, pp.1589-1600. DOI: 10.1145/3485447.3512205.

[14] Liu G, Wong L. Effective pruning techniques for mining quasi-cliques. In *Proc. the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases*, September 2008, pp.33-49. DOI: 10.1007/978-3-540-87481-2_3.

[15] Pei J, Jiang D, Zhang A. On mining cross-graph quasi-cliques. In *Proc. the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, August 2005, pp.228-238. DOI: 10.1145/1081870.1081898.

[16] Cheng J, Ke Y, Chu S, Özsu M. T. Efficient core decomposition in massive networks. In *Proc. the 27th IEEE International Conference on Data Engineering*, April 2011, pp.51-62. DOI: 10.1109/ICDE.2011.5767911.

[17] Li R H, Qin L, Yu J X, Mao R. Influential community search in large networks. *Proceedings of the VLDB Endowment*, 2015, 8(5): 509-520. DOI: 10.14778/2735479.2735484.

[18] Khuller S, Saha B. On finding dense subgraphs. In *Proc. the 36th International Colloquium on Automata, Languages, and Programming*, July 2009, pp.597-608. DOI: 10.1007/978-3-642-02927-1_50.

[19] Tsourakakis C. The k-clique densest subgraph problem. In *Proc. the 24th International Conference on World Wide Web*, May 2015, pp.1122-1132. DOI: 10.1145/2736277.2741098.

[20] Tsourakakis C, Bonchi F, Gionis A, Gullo F, Tsiarli M. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In *Proc. the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2013, pp.104-112. DOI: 10.1145/2487575.2487645.

[21] Charikar M. Greedy approximation algorithms for finding dense components in a graph. In *Proc. the 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization*, September 2000, pp.84-95. DOI: 10.1007/3-540-44436-X_10.

[22] Andersen R, Chellapilla K. Finding dense subgraphs with size bounds. In *Proc. the 6th International Workshop on Algorithms and Models for the Web-Graph*, February 2009, pp.25-37. DOI: 10.1007/978-3-540-95995-3_3.

[23] Epasto A, Lattanzi S, Sozio M. Efficient densest subgraph computation in evolving graphs. In *Proc. the 24th International Conference on World Wide Web*, May 2015, pp.300-310. DOI: 10.1145/2736277.2741638.

[24] Gionis A, Tsourakakis C E. Dense subgraph discovery: KDD 2015 tutorial. In *Proc. the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2015, pp.2313-2314. DOI: 10.1145/2783258.2789987.

[25] Silva A, Singh A, Swami A. Spectral algorithms for temporal graph cuts. In *Proc. the 27th International Conference on World Wide Web*, April 2018, pp.519-528. DOI: 10.1145/3178876.3186118.

[26] Peajcariaac J E, Tong Y L. Convex Functions, Partial Orderings, and Statistical Applications (1st edition). Academic Press, 1992.

[27] Eagle N, Pentland A. Reality mining: Sensing complex social systems. *Personal and Ubiquitous Computing*, 2006, 10(4): 255-268. DOI: 10.1007/s00779-005-0046-3.

[28] Yuan L, Qin L, Lin X, Chang L, Zhang W. Diversified top-k clique search. *The VLDB Journal*, 2016, 25(2): 171-196. DOI: 10.1007/s00778-015-0408-z.

[29] Lu C, Yu J X, Wei H, Zhang Y. Finding the maximum clique in massive graphs. *Proceedings of the VLDB Endowment*, 2017, 10(11): 1538-1549. DOI: 10.14778/3137628.3137660.

[30] Wang J, Cheng J. Truss decomposition in massive networks. arXiv:1205.6693, 2012. https://doi.org/10.48550/arXiv.1205.6693, July 2022.

[31] Galimberti E, Bonchi F, Gullo F, Lanciano T. Core decomposition in multilayer networks: Theory, algorithms, and applications. *ACM Trans. Knowledge Discovery from Data*, 2020, 14(1): Article No. 11. DOI: 10.1145/3369872.

[32] Galimberti E, Ciaperoni M, Barrat A, Bonchi F, Cattuto C, Gullo F. Span-core decomposition for temporal networks: Algorithms and applications. *ACM Trans. Knowledge Discovery from Data*, 2020, 15(1): Article No. 2. DOI: 10.1145/3418226.

[33] Wu H, Cheng J, Huang S, Ke Y, Lu Y, Xu Y. Path problems in temporal graphs. *Proceedings of VLDB Endowment*, 2014, 7(9): 721-732. DOI: 10.14778/2732939.2732945.

[34] Wu H, Huang Y, Cheng J, Li J, Ke Y. Reachability and time-based path queries in temporal graphs. In *Proc. the 32nd IEEE International Conference on Data Engineering*, May 2016, pp.145-156. DOI: 10.1109/ICDE.2016.7498236.

[35] Rozenshtein P, Gionis A. Temporal PageRank. In *Proc. the European Conference on Machine Learning and Knowledge Discovery in Databases*, September 2016, pp.674-689. DOI: 10.1007/978-3-319-46227-1_42.

[36] Akrida E C, Mertzios G B, Spirakis P G, Zamaraev V. Temporal vertex cover with a sliding time window. *Journal of Computer and System Sciences*, 2020, 107: 108-123. DOI: 10.1016/j.jcss.2019.08.002.

[37] Mertzios G B, Molter H, Zamaraev V. Sliding window temporal graph coloring. In *Proc. the 33rd AAAI Conference on Artificial Intelligence*, January 27-February 1, 2019, pp.7667-7674. DOI: 10.1609/aaai.v33i01.33017667.

[38] Lin L, Yuan P, Li R, Jin H. Mining diversified top-r lasting cohesive subgraphs on temporal networks. *IEEE Trans. Big Data*. DOI: 10.1109/TBDATA.2021.3058294.

[39] Qin H, Li R, Yuan Y, Wang G, Yang W. Periodic communities mining in temporal networks: Concepts and algorithms. *IEEE Trans. Knowledge and Data Engineering*, 2022, 34(8): 3927-3945. DOI: 10.1109/TKDE.2020.3028025.

[40] Zhang Y, Lin L, Yuan P, Jin H. Significant engagement community search on temporal networks. In *Proc. the 27th International Conference on Database Systems for Advanced Applications*, April 2022, pp.250-258. DOI: 10.1007/978-3-031-00123-9_20.

Chun-Xue Zhu received her B.E. degree in computer science and technology from Chongqing University, Chongqing, in 2020. She is currently pursuing her Master's degree in School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), Wuhan. Her current research interest is temporal network mining.

Long-Long Lin received his Ph.D. degree in theory of computer software from Huazhong University of Science and Technology (HUST), Wuhan, in 2022. He is currently an associate professor in the College of Computer and Information Science, Southwest University, Chongqing. His current research interests include social network analysis and temporal network mining.

Ping-Peng Yuan received his Ph.D. degree in computer science from Zhejiang University, Hangzhou, in 2002. He is now a professor in the School of Computer Science and Technology at Huazhong University of Science and Technology (HUST), Wuhan. His research interests include databases, knowledge representation and reasoning, and natural language processing, with a focus on high-performance computing. He is the principle developer in multiple system prototypes, including TripleBit, PathGraph and SemreX.

Hai Jin is a chair professor of computer science and engineering at Huazhong University of Science and Technology (HUST), Wuhan. Jin received his Ph.D. degree in computer engineering from HUST, Wuhan, in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. Jin worked at The University of Hong Kong, Hong Kong, between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. Jin is a fellow of CCF and IEEE, and a life member of ACM. He has co-authored more than 20 books and published over 900 research papers. His research interests include computer architecture, parallel and distributed computing, big data processing, data storage, and system security.