Densest Periodic Subgraph Mining on Large Temporal Graphs

Hongchao Qin, Rong-Hua Li, Ye Yuan, Yongheng Dai, and Guoren Wang

Abstract—Densest subgraphs are often interpreted as *communities*, based on a basic assumption that the connections inside a community are much denser than those between communities. In a graph with temporal information, a densest periodic subgraph is the most densely connected periodic behavior which needs to be captured. Unfortunately, the existing work do not model the densest periodic subgraph in temporal graphs, and the current algorithms for mining the densest subgraph cannot be applied to detect the densest periodic subgraph in the temporal networks. To tackle this problem, we propose a novel model, called the densest σ-periodic subgraph, which presents the densest periodic subgraph whose period size is σ. We prove that finding the densest σ-periodic subgraph can be solved in polynomial time, but it is still challenging because the naive algorithm needs to repeatedly invoke a maximum flow algorithm for many periodic subgraphs. To compute the densest σ-periodic subgraph efficiently, we first develop an effective pruning technique based on the degeneracy of the graph to significantly prune the number of the periodic subgraphs. Then, we present a more efficient algorithm that can reduce the computations for the degeneracy and maximum flow. Next, we develop a greedy algorithm that can compute the approximate densest σ-periodic subgraph and achieve an approximation ratio of 1/2. Finally, the results of extensive experiments on several real-life datasets demonstrate the efficiency, scalability, and effectiveness of our algorithms.

Index Terms—Densest subgraph, periodic subgraph, temporal graph

1 Introduction

Temporal networks, in which each edge is associated with an interaction time t, are ubiquitous in real life. For example, in a temporal social network, each edge (u, v, t) denotes a contact between user u and user v at time t. In a temporal communication network, each edge shows information of a sender, a receiver and their communication time. In fact, most networks are temporal due to at least one creation time of each edge associated. In the temporal graphs, some graph mining problems become more difficult to solve since we need to define new concepts and design more non-trivial methods by considering the temporal information. There are lots of works considering temporal information, to name but a few, Holme [1], [2] makes a colloquium about the novel definitions of paths, centrality measures, cyclic patterns motifs and so on in temporal network. He states that the methods and models developed for static networks could be inapplicable or could need non-trivial generalizations; Han et al. [3] build a temporal graph analysis system by exploring the interesting interplay among locality, parallelism, and incremental computation in supporting common mining tasks on temporal graph.

Periodicity is a frequently happening phenomenon for communications in temporal networks. Animal migration in the animal connection network [4], cell activation in the brain neuron network [5], and biological clock in the human activity network all exhibit periodic behaviors. In a static network, a densest subgraph can be related to community structure because a densest subgraph is a typical interactive behavior in a network. However, in a temporal network, a densest subgraph cannot represent a periodic interactive behavior since the dense subgraph model do not consider the temporal information on the edges. Therefore, we cannot use the densest subgraph model to find the periodic behavior such as animal migration, cell activation etc. In this manuscript, we seek for the periodic densest subgraph which is a dense part and also connected periodically. This model is helpful for finding the most densely connected and significant periodic activities in the temporal graphs, which can be described in the following applications.

Mining the Periodic Medical Behavior. It usually takes several treatments to cure a disease, so the treatments for a patient must be periodical. The hospital dataset [6] is a temporal network of face-to-face contacts between patients and health-care workers (including nurses, doctors, and administrative staffs). Inside this temporal network, there are interactions of patient-doctor, nurse-doctor, patient-nurse and so on. Each periodic subgraph in the temporal network can be a possible treatment interactions. The densest periodic subgraphs in this temporal network refer to the most complex treatment process in this hospital, since the density of this subgraph is highest. Therefore, we can seek the densest periodic subgraphs to mine the periodic medical behavior in a hospital interaction temporal network.

- *Hongchao Qin, Rong-Hua Li, Ye Yuan, and Guoren Wang are with the Department of Computer Science, Beijing Institute of Technology, Beijing 100811, China. E-mail: {hcqin, rhli, yuan-ye, wanggr}@bit.edu.cn.*
- *Yongheng Dai is with the China Academy of Electronic and Information Technology, Beijing 100190, China. E-mail: toyhdai@163.com.*

Manuscript received 27 December 2021; revised 29 November 2022; accepted 17 December 2022. Date of publication 4 January 2023; date of current version 6 October 2023.
This work was supported in part by the National Key R&D Program of China under Grant 2020AAA0108503; in part by NSFC under Grants 62202053, 62072034, U1809206, 61932004, 62225203, U21A20516, 61732003 and U2001211; and in part by CCF-Huawei Populus Grove Fund.
(Corresponding authors: Rong-Hua Li and Guoren Wang.)
Recommended for acceptance by S.S. Bhowmick.
Digital Object Identifier no. 10.1109/TKDE.2022.3233788

Predicting the Future Co-Operation. A densest periodic subgraph is more likely to model a periodic activity in the temporal network. Once we identify a periodic activity, we may predict that the same activity will appear within a regular periodic interval. Based on this observation, we are capable of inferring the future interactions of a group of individuals in a co-operation temporal network such as DBLP. For example, if four authors co-author papers continuously at years 2020, 2021, 2022. Then, we can infer that these four researchers are likely to coauthor papers in year 2023.

In this manuscript, we define a particular periodic and dense pattern on temporal networks, called densest periodic subgraph, which is the densest subgraph pattern that occurs periodically. In the literature, a few solutions for mining periodic subgraphs in temporal graphs have been proposed in the past years. For example, Lahiri [7] proposes an algorithm based on enumeration trees to list all the maximal periodic subgraphs in temporal networks. However, it can only list the maximal periodic subgraphs, but can not find the periodic and dense subgraphs. Qin [8] puts forward an efficient algorithm with several pruning strategies to seek all maximal periodic cliques in temporal networks. Although clique is a model of the dense subgraph, it is still hard to apply the periodic clique model into practical applications since the number of the maximal periodic cliques is exponential and the algorithm is not scalable (the problem of mining periodic clique is NP-hard [8]). Zhang [9] models the activity of communities as a mixture of hidden periodic signals and proposes an optimization approach to solve it. However, this model is difficult to optimize, so it can only handle the graph with thousands of nodes. Thus, a better algorithm is demanded for searching periodic and dense subgraphs on large temporal networks.

Based on a basic assumption that the connections inside a community are much denser than those between communities, we study the problem of mining densest periodic subgraphs (*DPS*) in the temporal graph, and invent an algorithm which can solve the *DPS* problem in polynomial time. To the best of our knowledge, we are the first to study the densest periodic subgraphs and propose a scalable algorithm to search them.

Contributions. In this paper, we formulate and provide efficient solutions to find the densest periodic subgraphs in a temporal graph. In particular, we make the following main contributions.

Novel Model. We propose a novel concept, called densest σ-periodic subgraph, to characterize the densest periodic subgraph in temporal graphs. A σ-periodic subgraph is the subgraph that occurs periodically in the temporal graph with the number of occurrences is σ. And a densest σ-periodic subgraph is the densest subgraph among all the σ-periodic subgraphs. As for the density of periodic subgraphs, we define it properly so that the problem can be solved in polynomial time. Moreover, the densest σ-periodic subgraph is a σ-periodic subgraph which has the highest density, hence it is more likely to be related to the periodic behavior in the temporal networks.

Scalable Algorithms. To search the densest σ-periodic subgraph, the basic algorithm is to enumerate the maximal periodic subgraphs first, and then invoke the maximum flow algorithm in each maximal periodic subgraph. However, this naive algorithm may produce numerous redundant computations while enumerating. In order to improve the efficiency, we first introduce σ-periodic degeneracy to prune the periodic timestamps while generating the periodic subgraphs. Then, we use k-core concepts to reduce the number of candidate nodes. Next, we propose a generated graph G^* in which we can search the densest σ-periodic subgraph by calling the maximum flow algorithm only once. Finally, we put forward a more scalable approximation algorithm that can process a ten-million level temporal graph in seconds.

Extensive Experiments. We conduct extensive experiments using several real-life temporal graphs to evaluate the proposed algorithm under different parameter settings. The results indicate that our algorithms significantly outperform the baselines in terms of community quality. In addition, through extensive experiments, we find that our methods are highly efficient. For instance, on a large-scale temporal graph with more than 3.2 million nodes and 12.2 million edges, our best exact algorithm and approximate algorithm can find the densest periodic subgraphs in 312 and 32 seconds, respectively.

Organization. Section 2 introduces the preliminaries and formulations of our problem. The basic algorithm of mining densest periodic subgraphs is proposed in Section 3. Furthermore, the algorithms with several pruning strategies and the approximate algorithm are proposed in Sections 4 and 5. Experimental studies are presented in Section 6, and the related work is discussed in Section 7. Section 8 draws the conclusion of this paper.

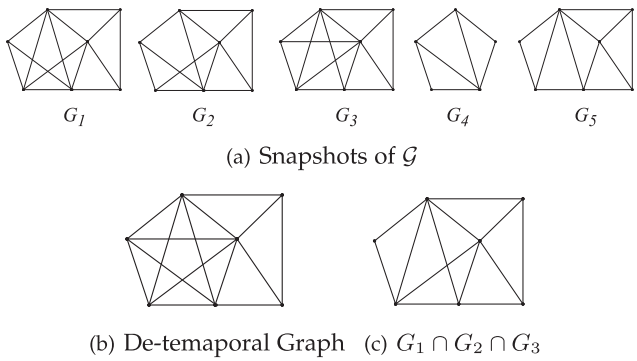
2 Preliminaries

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected temporal graph, where $\mathcal{V}$ and $\mathcal{E}$ denote the set of nodes and the set of temporal edges respectively. Each temporal edge $e \in \mathcal{E}$ is a triplet (u, v, t), where u, v are nodes in $\mathcal{V}$, and t is the interaction time between u and v. We assume that t is an integer, since the timestamp is an integer in practice.

We can extract $\mathcal{G}$ into a series of snapshots based on the timestamps. Let $\mathcal{T} = [\{t|(u, v, t) \in \mathcal{E}\}]$ be the set of timestamps, and it is a sequence $[t_1 : t_{|\mathcal{T}|}]$. Therefore, $\mathcal{G} = \{G_1, G_2...G_{|\mathcal{T}|}\}$ such that each snapshot $G_i = (V_i, E_i)$ where $V_i = \{u|(u, v, t_i) \in \mathcal{E}\}$ and $E_i = \{(u, v)|(u, v, t_i) \in \mathcal{E}\}$.

The *de-temporal graph* of $\mathcal{G}$, denoted by $G = (V, E)$, is a static graph that ignores all the timestamps associated with the temporal edges. More formally, for the de-temporal graph G of $\mathcal{G}$, $V = \mathcal{V}$ and $E = \{(u, v)|(u, v, t) \in \mathcal{E}\}$. Let $N_u(G) = \{v|(u, v) \in E\}$ be the set of neighbor nodes of u, and $d_u(G) = |N_u(G)|$ be the degree of u in G. A graph $G' = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A subgraph $G_S = (V_S, E_S)$ is referred to as an induced subgraph of G if $E_S = \{(u, v)|u, v \in V_S, (u, v) \in E\}$. Similarly, a temporal subgraph $\mathcal{G}_\mathcal{S} = (\mathcal{V}_\mathcal{S}, \mathcal{E}_\mathcal{S})$ is referred to as an induced temporal subgraph of $\mathcal{G}$ if $\mathcal{V}_\mathcal{S} \subseteq \mathcal{V}$ and $\mathcal{E}_\mathcal{S} = \{(u, v, t)|u, v \in \mathcal{V}_\mathcal{S}, (u, v, t) \in \mathcal{E}\}$. For convenience, we use the notion $S \subseteq G$ ($S \subset G$ if $S \neq G$) to indicate that S is a subgraph of G.

Definition 1 (σ-periodic time support set). *Given a temporal graph $\mathcal{G}$, de-temporal graph G of $\mathcal{G}$ and parameter σ, a σ-periodic time support set of a subgraph $S \subseteq G$ can be denoted by $\mathbb{PT}^\sigma(S) = \{t_{j_1}, \cdots, t_{j_\sigma}\}$ $(j_1 <= j_2 <= ...j_\sigma)$, satisfying*



(a) Snapshots of $\mathcal{G}$

(b) De-temaporal Graph (c) $G_1 \cap G_2 \cap G_3$

Fig. 1. Running example.

$t_{j_{i+1}} - t_{j_i}$ is a constant which is the periodic interval and $S \subseteq G_{t_{j_i}}$ for all $i = 1, \cdots, \sigma - 1$.

By Definition 1, we can see that the timestamps in a σ-periodic time support set form an arithmetic sequence and the cardinality of a σ-periodic time support set is exactly equal to σ. We do not care about the periodic interval, but focus on the period size σ, since different periodic behavior may occur with different periodic interval. Clearly, there may exist many periodic support sets of size σ for a subgraph S. Derived from Definition 1, we define the σ-periodic subgraph below.

Definition 2 (σ-periodic subgraph). *Given a temporal graph $\mathcal{G}$, de-temporal graph G of $\mathcal{G}$ and parameter σ, a subgraph $S \subseteq G$ is a σ-periodic subgraph in $\mathcal{G}$ if there exists a σ-periodic time support set $\mathbb{PT}^{\sigma}(S)$.*

By Definition 2, any σ-periodic subgraph $S \in G$ has at least one σ-periodic time support set, and a subgraph S is a maximal σ-periodic subgraph if there is no other σ-periodic subgraph S' that satisfies $S \subset S'$. Below, we define the density of the subgraph.

Definition 3 (density). *The density of a graph $G = (V, E)$, denoted by $\rho(G)$, equals to the ratio between number of the edges and number of nodes in G, i.e., $\rho(G) = \frac{|E|}{|V|}$.*

Based on Definition 2 and Definition 3, the model of the densest periodic subgraph can be defined as follows.

Definition 4 (densest σ-periodic subgraph). *A densest σ-periodic subgraph S (abbreviated as $\sigma-$DPS) is a σ-periodic subgraph which has the largest density such that there exists no σ-periodic subgraph S' satisfying $\rho(S') > \rho(S)$.*

Example 1. Fig. 1 a illustrates all the five snapshots of a temporal graph $\mathcal{G}$. Fig. 1b illustrates the de-temporal graph G of $\mathcal{G}$ in Fig. 1a. For the subgraph $G' = G_1 \cap G_2 \cap G_3$ in Fig. 1c, we can see that the time support set of G' in $\mathcal{G}$ is [1,2,3], such that G' is a 3-periodic subgraph. However, the density $\rho(G') = 13/7$ and we cannot find a subgraph $S \subset G'$ satisfying $\rho(S) > \rho(G')$. So, G' is the densest subgraph of the 3-periodic subgraph in time [1,2,3].

Problem. Given a temporal graph $\mathcal{G}$, an integer $\sigma \geq 2$, the goal of mining the densest periodic subgraph is to compute the densest σ-periodic subgraph in $\mathcal{G}$.

Challenges. One straightforward method to solve the problem is enumerating the maximal periodic subgraphs first, and then invoking the traditional parametric flow algorithm to find the densest subgraph in each maximal periodic subgraph. Finally, picking the densest solution among all maximal periodic subgraphs to be the $\sigma-$DPS. Since the number of the maximal periodic subgraph is bounded by $|\mathcal{T}|^2$, the problem can be solved in polynomial time. However, this method is not efficient and redundant, since we need to invoke the maximum flow algorithm on the periodic subgraphs for $O(|\mathcal{T}|^2)$ times, and many of the periodic subgraphs are similar or totally same. Another potential approach is finding a subgraph with highest density first, and then checking whether the subgraph is periodic. Clearly, this approach is impracticable, because choosing such graph with highest density is tough and we need non-polynomial time to try all the subgraphs.

Therefore, the challenge of our problem is how to efficiently enumerate all periodic subgraphs with less redundant computations. In the following sections, we will develop several novel graph reduction techniques and an efficient enumeration algorithm to identify the densest σ-periodic subgraph.

3 The Basic Algorithm

To find the $\sigma-$DPS in a temporal graph, a straightforward way is enumerating all the σ-periodic subgraph first, and then invoking Goldberg's parametric flow algorithm [10], [11] to search the densest subgraphs in each σ-periodic subgraph. Combining all the results of densest subgraphs in all σ-periodic subgraphs, a subgraph with the maximum density is the $\sigma-$DPS. Clearly, this basic algorithm is costly because the number of σ-periodic subgraphs is numerous and the procedure of densest subgraph mining algorithm is ineffective in large graphs. Therefore, we conduct several powerful pruning rules which can reduce the size of the temporal graph before performing the subgraph enumeration and densest subgraph mining.

Below, we introduce a simple periodic subgraph enumeration method using Goldberg's algorithm [11] to find $\sigma-$DPS in Algorithm 1. The implementation detail is shown as follows.

Algorithm 1 first devises a new data structure, $PerioSub$, to represent the set of σ-periodic time support set in Definition 1. Each item PS in $PerioSub$ is a four-tuple $[s, i, l, G']$, in which s is the start time, i refers to the time interval, l represents the current confirmed length and G' is the current periodic subgraph. Based on this data structure, the algorithm makes use of $PerioSub$ to maintain all the candidates of the arithmetic sequences, $StrtT$ to maintain the starting timestamps set and DPS to record the densest σ-periodic subgraph. Initially, $PerioSub$, $StrtT$ and DPS are set to be empty (line 1). Then, the algorithm enumerates all the timestamps from 1 to $|\mathcal{T}|$ (line 2). For each timestamp t, the algorithm explores all the candidate arithmetic sequences in $PerioSub$ (line 3). For each candidate $PS \in PerioSub$, if $(t - PS.s)\%PS.i \neq 0$, we can continue the loop, because t is absolutely not in the arithmetic sequence PS (line 4). Otherwise, the algorithm can augment the arithmetic sequence PS by adding t into it. In this case, we increase $PS.l$ by 1, and compute the intersection of the periodic snapshots (line 5). Next, if $PS.l = \sigma$, there exists a valid periodic subgraph which is

recorded by $PS.G'$. Then the algorithm invokes procedure DensestSub to compute the densest subgraph in $PS.G'$ (line 7). If the density of the densest subgraph in $PS.G'$ is larger than the recorded value, the DPS will be updated (line 8). After that, the current PS is popped from $PerioSub$ since it has been considered (line 9). The algorithm also applies the current timestamp t to generate a new starting timestamp which will be used for the next iterations (lines 10-11). Since Algorithm 1 explores all the possible arithmetic sequences, it will search all the possible periodic subgraphs. And in each periodic subgraph, it will call procedure DensestSub once.

Algorithm 1. DPS$-B(\mathcal{G}, \sigma)$

Input: Temporal graph $\mathcal{G} = \{G_1, ...G_{|\mathcal{T}|}\}$, parameter σ
Output: $\sigma-$DPS in $\mathcal{G}$

1: $PerioSub \leftarrow \emptyset$; $StrtT \leftarrow \emptyset$; $DPS \leftarrow \emptyset$;
2: **for** $t \leftarrow 1 : |\mathcal{T}|$ **do**
3: **for** $PS \leftarrow [s, i, l, G'] \in PerioSub$ **do**
4: **if** $(t - PS.s)\%PS.i \neq 0$ **then continue**;
5: $PS.l \leftarrow PS.l + 1$; $PS.G' \leftarrow PS.G' \cap G_t$;
6: **if** $PS.l = \sigma$ **then**
7: $DPS' \leftarrow$ DensestSub$(PS.G', 0, |E_{PS.G'}|)$;
8: **if** $\rho(DPS') > \rho(DPS)$ **then** $DPS \leftarrow DPS'$;
9: $PerioSub.pop(PS)$; **continue**;
10: **for** $s \in StrtT$ **do** $PerioSub.push([s, t-s, 2, G_s \cap G_t])$;
11: $StrtT \leftarrow StrtT \cup \{t\}$;
12: **return** DPS;
13: **Procedure** DensestSub(G, l, u)
14: $DS \leftarrow \emptyset$;
15: **while** $u - l \geq 1/|V_G|^2$ **do**
16: $\rho = (l + u)/2$; $G_F \leftarrow V_G \cup \{s\} \cup \{t\}$ in which s, t are new nodes;
17: **for** *each edge* $(u, v) \in E_G$ **do**
18: add edges (u, v) with capacity 1 into G_F ;
19: **for** *each node* $w \in V_G$ **do**
20: add an edge (s, w) with capacity $|E_G|$ into G_F;
21: add an edge (w, t) with capacity $(|E_G| + 2 \times \rho - d_w(G))$ into G_F;
22: Compute the minimum s-t cut in G_F, denoted by S and T;
23: **if** $S \setminus \{s\} \neq \emptyset$ **then** $\{DS \leftarrow S \setminus \{s\}; l \leftarrow \rho;\}$ **else** $u \leftarrow d$;
24: **return** DS;

Procedure DensestSub uses the Goldberg's parametric flow algorithm, and it can find the maximal densest subgraph in polynomial time. The general idea of procedure DensestSub is as follows: it first uses a binary search process to find the optimal density (lines 15-16). In each step of the procedure, the algorithm guesses a density ρ in a binary search manner, and tries to find a subgraph G with density larger than ρ (lines 15-23). Such a subgraph can be identified by computing the minimum s-t cut in a flow network G_F (lines 16-21). The binary search procedure can terminate in $O(\log n)$ iterations (lines 15-16) [11]. The detailed description and correctness analysis of Goldberg's algorithm can be found in [10].

Example 2. Fig. 2 shows the process of generating the 3-periodic subgraphs and computing the densest subgraph while $t = 2 \rightarrow 4$. When $t = 2$, the start time set $StrtT$ is [1,2], and the set $PerioSub$ only has one item.

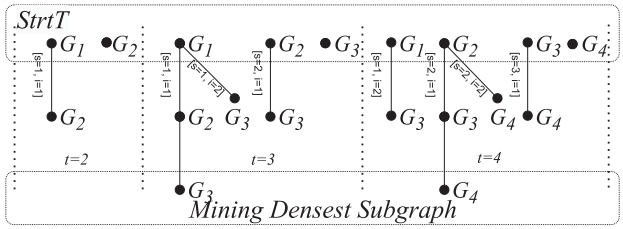


Fig. 2. Illustration for Algorithm 1.

When t comes to 3, the algorithm finds one 3-periodic subgraph in time [1,2,3] and it invokes Proc. DensestSub to find the densest subgraph. Subsequently, the algorithm enumerates all the 3-periodic subgraphs such as subgraph in time $[1, 2, 3], [2, 3, 4], [1, 3, 5], [3, 4, 5]$ and so on.

Theorem 1 (Complexity of Algorithm 1). *For a temporal graph $\mathcal{G}$ with $|\mathcal{T}|$ timestamps and de-temporal graph $G = (V, E)$ of $\mathcal{G}$, the time and space complexity of Algorithm 1 are $O(|\mathcal{T}|^2\sigma^{-1}mn\log(\frac{n^2}{m}))$ and $O(m + n|T|^2\sigma^{-1})$ respectively, in which $n = |V|, m = |E|$.*

Proof. Recall that each timestamps of a periodic is a σ-term arithmetic sequence which can be represented as $\{t_{i+p}, t_{i+2p}, \cdots, t_{i+\sigma p}\}$, where $0 < i \leq |\mathcal{T}| - (\sigma - 1)p$ and $p \geq 1$ is a common difference. Clearly, the maximum p is $\left\lfloor \frac{|\mathcal{T}|-1}{\sigma-1} \right\rfloor$. Since $i + \sigma p \leq |\mathcal{T}|$, we have $i \leq |\mathcal{T}| - \sigma p$. As a result, the total number of arithmetic sequences can be bounded by $\sum_{p=1}^{\left\lfloor \frac{|\mathcal{T}|-1}{\sigma-1} \right\rfloor} (|\mathcal{T}| - \sigma p)$. By relaxing this formula, we can easily derive that the number of is bounded by $O(|\mathcal{T}|^2\sigma^{-1})$.

Since there are $O(|\mathcal{T}|^2\sigma^{-1})$ periodic subgraphs, in each enumeration, we need total $O(\sigma m)$ time to compute the interactions of the snapshots for σ times (line 5). According to [12], [13], the maximum flow of G_F for all possible ρ can be computed in $O(nm\log(\frac{n^2}{m}))$ time, such that we need $O(nm\log(\frac{n^2}{m}))$ to perform Proc. DensestSub. Therefore, the total time complexity is $O(|\mathcal{T}|^2\sigma^{-1}(\sigma m + nm\log(n^2/m))) = O(|\mathcal{T}|^2\sigma^{-1}nm\log(\frac{n^2}{m}))$.

Besides, the algorithm need to store the graph and the storage for each periodic graph G' can be released after computing the densest subgraph G'. However, the maximum size of set $PerioSub$ is $O(|\mathcal{T}|^2\sigma^{-1})$, and we can only store the nodes in G' to present the graph. So, the space complexity is $O(m + n|T|^2\sigma^{-1})$. □

4 The Improved Algorithms

As described in Algorithm 1, we need to enumerate all the snapshots (line 2) in the temporal graph to generate the periodic subgraph and invoke the maximum flow computation for many times (line 18). In this section, in order to improve the efficiency, we propose several powerful techniques to prune the unpromising nodes and even periodic subgraph which are totally not connected with the $\sigma-$DPS. Our key idea for reduction is based on the concept of degeneracy. Before proceeding further, we first give the definition of degeneracy as follows.

Definition 5 (Degeneracy). *The degeneracy of a graph G is the minimum integer δ such that each subgraph $S \subseteq G$ contains a node v with degree no larger than δ.*

The degeneracy of G has many properties, two of which are listed as following lemmas:

Lemma 1. *For two graph G and G', the degeneracy δ of graph $G' \cap G$ is no larger than the degeneracy of G' or G.*

Lemma 2 (Ref. [14]). *The degeneracy δ of a static graph G, is 1/2-approximation for the density ρ_{max} of the densest subgraph in G, i.e. $\delta/2 \leq \rho_{max} < \delta$.*

The classic degeneracy, however, cannot be directly used to bound the density of $\sigma-$DPS in temporal graphs, since we need to consider all the σ-periodic subgraphs. Below, we introduce a novel concept, called σ-periodic degeneracy, which will be applied to bound the density of $\sigma-$DPS.

Definition 6 (σ-periodic degeneracy). *Given a temporal graph $\mathcal{G}$ and parameter σ, the σ-periodic degeneracy of $\mathcal{G}$ is the smallest integer $\hat{\delta}$ such that every σ-periodic subgraph contains a node with degree at most $\hat{\delta}$.*

Theorem 2. *The σ-periodic degeneracy $\hat{\delta}$ of a temporal graph $\mathcal{G}$, is 1/2-approximation for the density of $\sigma-$DPS $\hat{\rho}_{max}$ in $\mathcal{G}$, i.e. $\hat{\delta}/2 \leq \hat{\rho}_{max} < \hat{\delta}$.*

Proof. Suppose that $\hat{G}$ is the σ-periodic subgraph which has the largest $\hat{\delta}$. According to Definition 6, $\hat{\delta}$ is the smallest integer so there exists a subgraph $\hat{S} \subseteq \hat{G}$ satisfying $d_v(\hat{S}) = \hat{\delta}$ for each $v \in \hat{S}$. Since $\rho(\hat{S}) = \frac{\hat{\delta}}{2}$, we can have $\hat{\delta}/2 \leq \hat{\rho}_{max}$. Furthermore, for any σ-periodic subgraph S in $\mathcal{G}$ and each node $v \in S$, $d_v(S) \leq \hat{\delta}$. So, any subgraph S satisfies that $\rho(S) < \hat{\delta}$. Therefore, $\hat{\delta}/2 \leq \hat{\rho}_{max} < \hat{\delta}$. □

Recall that in Algorithm 1, we set a lower bound and an upper bound for the density of the $\sigma-$DPS in temporal graph $\mathcal{G}$ (line 7), and then try to compute the maximum flow with using parameter of $(l+u)/2$ (line 15). However, according to Theorem 2, the lower bound and upper bound of the maximum density of $\sigma-$DPS can be $\hat{\delta}/2$ and $\hat{\delta}$ respectively, in which $\hat{\delta}$ is the σ-periodic degeneracy. In this section, we introduce an effective method to compute $\hat{\delta}$. At first, we introduce a new concept, k$-core$, which can be applied into computing $\hat{\delta}$.

Definition 7 (k$-core$). *Given a de-temporal graph G of $\mathcal{G}$ and a parameter k, a k$-core$ is the maximal subgraph S of G in which the degree of each node is at least k, i.e., $d_u(S) \geq k$ for $u \in G$.*

Given a graph G, the degeneracy δ is the number of the maximum k satisfying kCore $\neq \emptyset$. The detail of procedure Degeneracy δ is shown at Algorithm 2. It first initializes the considering subsets S with V_G, the deleting nodes queue Q with $\emptyset$, and $Deg[u]$ with the degree of u inside G_S (line 1). Then, the algorithm loops until the S is deleted to be $\emptyset$ (line 9). In each loop, it captures the nodes set D with the minimal degree, and then pushes all nodes in D into Q (lines 3-4). Subsequently, the algorithm iteratively processes the nodes in Q. In each iteration, the algorithm pops a node v from Q and uses D to maintain all the deleted nodes (line 6). For each w which is the neighbor of v in G_S, if $Deg[w] \geq d$, the algorithm reduces $Deg[w]$. If the revised $Deg[w]$ is less than d, w will be peeled from the considering nodes set S and then be pushed into Q (line 8). Next, S is updated by the deleting nodes in D. If S is $\emptyset$, then the algorithm returns d, which is the maximal number of kCore, i.e., the degeneracy of G.

Algorithm 2. Degeneracy(G)

Input: Temporal graph $\mathcal{G} = \{G_1, ...G_{|\mathcal{T}|}\}$, parameter σ
Output: $\sigma-$DPS in $\mathcal{G}$
1: $S \leftarrow V_G$; $Q \leftarrow \emptyset$; **for** $u \in S$ **do** $Deg[u] \leftarrow d_u(S)$;
2: **while** *True* **do**
3: $d \leftarrow$ the minimal degree of nodes in the graph G_S;
4: $D \leftarrow \{u | d_u(G_S) = d\}$; **for** $u \in D$ **do** $Q.push(u)$;
5: **while** $Q \neq \emptyset$ **do**
6: $v \leftarrow Q.pop()$; $D \leftarrow D \cup \{v\}$;
7: **for** $w \in N_v(G_S)$ *s.t.* $Deg[w] \geq d$ **do**
8: $Deg[w] \leftarrow Deg[w] - 1$; **If** $Deg[w] < d$ **then** $Q.push(w)$;
9: $S \leftarrow S \setminus D$; **If** $S = \emptyset$ **then return** d;

Theorem 3 (Complexity of Computing Degeneracy). *For a graph $G = (V, E)$, the time and space complexity of computing G's degeneracy by Algorithm 2 are both $O(m)$, in which $m = |E|$.*

Proof. In line 1, Algorithm 2 needs $O(m)$ to record the degree of all the nodes. The minimal degree of nodes can be found in $O(\log|V|)$ (line 3). For lines 4-8, we can observe that each edge will be considered once, so the total time complexity of this process is $O(m)$. Besides, the algorithm needs to maintain the graph and sets S, Q and Deg, which all require $O(m)$ memory. In conclude, the time and space complexity of Algorithm 2 are all $O(m)$. □

To compute the exact σ-periodic degeneracy, we enumerate the periodic subgraphs and invoke Algorithm 2 to compute the degeneracy δ of each periodic subgraph, and the maximum δ is the σ-periodic degeneracy $\hat{\delta}$. As it needs $O(|\mathcal{T}|^2 m)$ (see the proof of Theorem 1) to generate all the σ-periodic subgraphs, the whole process of computing $\hat{\delta}$ needs time complexity of $O(|\mathcal{T}|^2 m^2)$.

4.1 Algorithm of Pruning Invalid Subgraphs

In the previous subsection, we mentioned that in order to compute σ-periodic degeneracy, we need to enumerate all the σ-periodic subgraphs first, which is obviously inefficient. To avoid listing all the periodic subgraphs, we propose a pruning algorithm which considers the periodic property on-demand. The details are provided in Algorithm 3, which is abbreviated as DPS$-P$.

Algorithm 3 uses the similar enumeration method like Algorithm 1 to compute the periodic subgraphs. However, it adds timestamps set $DelT$ to record the time t where snapshot G_t will not contain the $\sigma-$DPS (line 1). We use $PerioSub$ to record the temp results, $StrtT$ to record the candidate start time, DPS to record the candidate $\sigma-$DPS and $\hat{\delta}$ to record the computed σ-periodic degeneracy (line 1), then enumerate t from 1 to $|\mathcal{T}|$ to generate all the periodic subgraphs (line 2). In this algorithm, if the current t is in $DelT$, it will not be considered to construct a periodic subgraph. Subsequently, it checks the value of start, interval and

length in PS to identify a periodic subgraph. Note that, it needs to check whether the current $\frac{t-PS.s}{PS.i}$ equals $PS.l$, because some timestamps may be added into $DelT$ before, and they will be skipped (line 6). If we have found a periodic subgraph, the algorithm uses Algorithm 2 to compute the max periodic degeneracy δ of the current periodic subgraph $PS.G$ (line 9). If δ is larger than the computed σ-periodic degeneracy $\hat{\delta}$, then $\hat{\delta}$ will be updated to be equaled to δ (line 11). Concurrently, the algorithm checks whether the degeneracy of the snapshots $G_{t'}$ is less than $\hat{\delta}/2$ with t' ranges from t to $|\mathcal{T}|$. If $\mathsf{Degeneracy}(G_{t'}) < \frac{\hat{\delta}}{2}$, according to Theorem 1, any periodic subgraph contains timestamps t' will not have a subgraph whose density is larger than $\frac{\hat{\delta}}{2}$, so t' will be added into $DelT$ (lines 12-13). Note that, we can compute once and store the degeneracy of each snapshots G_t, so it will not repeatedly invoke Algorithm 2. Next, in line 14, if $\delta \geq \frac{\rho(DPS)}{2}$, the current periodic subgraph may have a subgraph with density larger than $\hat{\delta}$. Then, the algorithm invokes procedure DensestSub to compute the densest subgraph in the current periodic subgraph. According to Theorem 2, it invokes DensestSub with the initialized lower and upper bound of $\frac{\delta}{2}$ and δ. If the computed densest subgraph DPS' is denser than DPS, the algorithm update DPS to be DPS' (line 16). In the end, it removes the considered periodic subgraph PS (line 17), and applies the current timestamp t to generate a new starting timestamp (line 18). Finally, after t ranges from 1 to $|\mathcal{T}|$, it returns DPS.

Algorithm 3. DPS$-P(\mathcal{G}, \sigma)$

Input: Temporal graph $\mathcal{G} = \{G_1, ...G_{|\mathcal{T}|}\}$, parameter σ
Output: $\sigma-$DPS in $\mathcal{G}$

1: $PerioSub \leftarrow \emptyset$; $StrtT, DelT \leftarrow \emptyset, \emptyset$; $DPS \leftarrow \emptyset$; $\hat{\delta} \leftarrow 0$;
2: **for** $t \leftarrow 1 : |\mathcal{T}|$ **do**
3: **If** $t \in DelT$ **then continue**;
4: **for** $PS \leftarrow [s, i, l, G'] \in PerioSub$ **do**
5: **if** $(t - PS.s)\%PS.i \neq 0$ **then continue**;
6: **if** $\frac{t-PS.s}{PS.i} \neq PS.l$ **then** {$PerioSub.pop(PS)$; **continue**;}
7: $PS.l \leftarrow PS.l + 1$; $PS.G' \leftarrow PS.G' \cap G_t$;
8: **if** $PS.l = \sigma$ **then**
9: $\delta \leftarrow \mathsf{Degeneracy}(PS.G)$;
10: **if** $\delta > \hat{\delta}$ **then**
11: $\hat{\delta} \leftarrow \delta$;
12: **for** $t' \leftarrow t : |\mathcal{T}|$ **do**
13: **if** $\mathsf{Degeneracy}(G_{t'}) < \frac{\hat{\delta}}{2}$ **then** $DelT.add(t')$;
14: **if** $\delta \geq \rho(DPS)/2$ **then**
15: $DPS' \leftarrow \mathsf{DensestSub}(PS.G', \frac{\delta}{2}, \delta)$;
16: **if** $\rho(DPS') > \rho(DPS)$ **then** $DPS \leftarrow DPS'$;
17: $PerioSub.pop(PS)$; **continue**;
18: **for** each $s \in StrtT$ **do** $PerioSub.push([s, t-s, 2, G_s \cap G_t])$;
19: **return** DPS;

Theorem 4 (Complexity of Algorithm 3). *The worst case time and space complexity of Algorithm 3 are the same as those of Algorithm 1.*

In the worst case, Algorithm 3 needs to invoke Procedure DensestSub to compute the densest subgraph of each periodic subgraph. However, the pruning rule based on Algorithm 3 can reduce the computation time greatly. The running time of Algorithm 3 is shown in Section 6.

4.2 Algorithm of Putting All Together

Although Algorithm 3 is efficient in practice, it still has three limitations. (i) It needs to perform the maximum flow computation for many times (line 15). In the worse case, the number of the calls of procedure DensestSub is $O(\frac{\mathcal{T}^2}{\sigma})$. (ii) It needs to invoke Algorithm 2 to compute the degeneracy for the σ-periodic subgraphs (line 9) and the snapshots (line 13). In the worse case, the number of the calls of Algorithm 2 is also $O(\frac{\mathcal{T}^2}{\sigma})$. (iii) It needs to compute the intersection of the snapshots G_t for σ times (line 7) to generate one σ-periodic subgraph. Only this step will take $O(\mathcal{T}^2 m)$ time.

Algorithm 4. DPS$-P+(\mathcal{G}, \sigma)$

Input: Temporal graph $\mathcal{G} = \{G_1, ...G_{|\mathcal{T}|}\}$, parameter σ
Output: $\sigma-$DPS in $\mathcal{G}$

1: $\bar{\delta} \leftarrow$ guess a value for the σ-periodic degeneracy of $\mathcal{G}$;
2: $G_c = \{V_c, E_c\} \leftarrow \mathsf{kCore}(G, \frac{\bar{\delta}}{2})$;
3: $PerioN \leftarrow \emptyset$; $StrtT, DelT \leftarrow \emptyset, \emptyset$; $DPS \leftarrow \emptyset$; $G^* \leftarrow \emptyset$;
4: **for** $t \leftarrow 1 : |\mathcal{T}|$ **do**
5: **If** $T \in DelT$ **then continue**;
6: **for** *each* $u \in V_c$ s.t. $d_u(G_t) \geq \frac{\bar{\delta}}{2}$ **do**
7: **for** $PN \leftarrow [s, i, l] \in PerioN[u]$ **do**
8: **if** $(t - PN.s)\%PN.i \neq 0$ **then continue**;
9: **if** $\frac{t-PN.s}{PN.i} \neq PN.l$ **then** {$PerioN[u].pop(PN)$; **continue**;}
10: $PN.l \leftarrow PN.l + 1$;
11: **if** $PN.l = \sigma$ **then**
12: add one node $\{[u, PN.s, PN.i]\}$ into G^*;
13: **for** $s \in StrtT$ **do**
14: $PerioN[u].push([s, t-s, 2, \min(d_u(G_s), d_u(G_t))])$;
15: **for** *each node* $[u, s, i] \in V_{G^*}$ **do**
16: **for** $v \in N_u(G)$ *s.t.* $[v, s, i] \in V_{G^*}$ **do**
17: **if** edge (u, v) in $\{G_s, G_{s+i}, ...G_{s+i(\sigma-1)}\}$ **then**
18: add edge $([u, s, i], [v, s, i])$ into G^*;
19: $\bar{\delta} \leftarrow \mathsf{Degeneracy}(G^*)$; $G^* \leftarrow \mathsf{kCore}(G^*, \frac{\bar{\delta}}{2})$;
20: **return** $\mathsf{DensestSub}(G^*, \frac{\bar{\delta}}{2}, \bar{\delta})$;
21: **Procedure** kCore(G, d)
22: **return** k$-core$ of G in which $k = d$;

To overcome those limitations, we propose an improved algorithm called DPS$-P+$, and the striking features are as follows. *First*, it does not need to enumerate the σ-periodic subgraphs, but generate one new graph with the promising periodic nodes to record the periodic information. *Second*, DPS$-P+$ only needs to compute the σ-periodic degeneracy and to invoke the Proc. DensestSub to compute the densest subgraph for only once, which greatly save the computation time. *Third*, to generate the new graph, it does not need to compute any interactions of the snapshot, which makes the process efficient. Before introducing DPS$-P+$, we show a property of the $\sigma-$DPS which can be applied to reduce the temporal networks at the beginning of the algorithm.

In the last section, we prune some snapshots to speed up the enumerations for the periodic subgraphs. However, according to the theorem below, we can actually prune some nodes that are definitely not contained in the $\sigma-$DPS at the beginning.

Theorem 5. *Given a temporal graph $\mathcal{G}$, its de-temporal graph G, parameter σ and the σ-periodic degeneracy $\hat{\delta}$, all nodes in $\sigma-$DPS need to be contained in $\frac{\hat{\delta}}{2}$-Core of G.*

Proof. According to the definition of the de-temporal graph G, any σ-periodic subgraph G' in $\mathcal{G}$ must be contained in G. So for any node u and any G', $d_u(G') \leq d_u(G)$. We can know from Theorem 2 that the density of the $\sigma-$DPS in one σ-periodic subgraph G' is at least $\hat{\delta}/2$, so the density of the induced graph from the $\sigma-$DPS in G is no less than $\hat{\delta}/2$. Since $\frac{\hat{\delta}}{2}$-Core is a maximal subgraph in which the degree of each node is at least $\hat{\delta}/2$, if the $\sigma-$DPS is not contained in $\frac{\hat{\delta}}{2}$-Core then one node u' in the $\sigma-$DPS will have density less than $\hat{\delta}/2$. However, we can remove u' from the $\sigma-$DPS to get a subgraph with higher density, which violates the definition of the $\sigma-$DPS. Therefore, all the $\sigma-$DPS $\subseteq \frac{\hat{\delta}}{2}$-Core. □

According to Theorem 5, we can invoke the kCore algorithm to reduce the graph size first, and then find $\sigma-$DPS in the reduced graph. Furthermore, the kCore structure in the graph holds the property that each k+1-Core must be contained in the kCore. So, we can store the intermediate subgraph, and delete the nodes or unnecessary periodic subgraphs by a peeling algorithm. We generate a new graph which records the periodic information from the reduced graph. The generated graph can be defined as follows.

Definition 8 (G-star). *Given a temporal graph $\mathcal{G}$, parameter σ and σ-periodic degeneracy $\hat{\delta}$ of $\mathcal{G}$, G-star (G^*) is a static graph in which each node is a triple $[u, s, i]$ with $d_u(G_j) \geq \frac{\hat{\delta}}{2}$, and each edge $([u, s, i], [v, s, i])$ satisfies edge $(u, v, j) \in \mathcal{G}$, for all $j = s, s + i, ..., s + i \times (\sigma - 1)$.*

We can generate the G^* of the temporal graph $\mathcal{G}$ to compute the densest subgraphs. The detailed process is shown as follows.

According to Theorem 5, Algorithm 4 first invokes modified Algorithm 3 (see Algorithm 5 in next section) to compute a lower bound for the σ-periodic degeneracy of $\mathcal{G}$ (line 1). Then, it computes the kCore $(k = \frac{\hat{\delta}}{2})$ of the de-temporal graph G to determine the candidate nodes (line 2). Algorithm 4 uses $StrtT$ to record the candidate start time, DPS to record the candidate $\sigma-$DPS, $DelT$ to record the deleted timestamps set and a new data structure $PerioN$ to record the periodic nodes (line 1). Next, it enumerates t from 1 to $|\mathcal{T}|$ to generate a new static graph G^* which contains all the periodic nodes (lines 4-14). In line 6, it checks whether the degree of the current node u is no less than $\frac{\hat{\delta}}{2}$. If u's degree meets the constraint, then the algorithm constructs a periodic degree sequence PN for node u (lines 7-14). Note that, it not need to compute any interactions of subgraphs. Each periodic node u is recorded by $[u, PN.s, PN.i]$ and formated as a new node in G^*, which means that u has degree of no less than $\frac{\hat{\delta}}{2}$ at periodic time $[PN.s, PN.s + PN.i...PN.s + PN.i \times (\sigma - 1)]$ (line 12). After all the periodic nodes are detected, the algorithm checks whether the periodic nodes can be connected in the periodic timestamps, and then adds the detecting edges into G^* (lines 15-18). Finally, it computes the degeneracy in G^*, reduce G^* to be kCore $(k = \frac{\hat{\delta}}{2})$ of G^* (line 19), and invokes Proc. DensestSub to mine the densest subgraphs in G^* (line 20). The following theorem shows that the connection between the degeneracy and densest subgraphs in G^* with the σ-periodic degeneracy and $\sigma-$DPS in $\mathcal{G}$.

Example 3. Algorithm 4 first computes the σ-periodic degeneracy δ of $\mathcal{G}$ in Fig. 1a. Then, it checks the kCore of de-temporal graph G with $k = \delta$ to find all the periodic nodes and generates the G^*. However, all the periodic nodes whose degrees are no less than δ is shown at Fig. 3 c. Then, it checks whether those nodes are connected in the corresponding timestamps (lines 15-18) and generates the final G^*. We can invoke Proc. DensestSub once to find a densest subgraph, which is the $\sigma-$DPS according to Theorem 6.

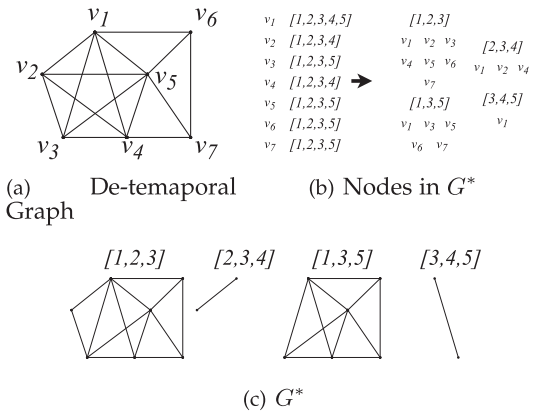


Fig. 3. Illustration for Algorithm 4 (σ=3).

Theorem 6. *Given a temporal graph $\mathcal{G}$, parameter σ and a generated graph G^* of $\mathcal{G}$, the degeneracy and densest subgraphs in G^* equals the σ-periodic degeneracy and the $\sigma-$DPS in $\mathcal{G}$, respectively.*

Proof. We prove the theorem by demonstrating that the $\frac{\hat{\delta}}{2}$-Core of G^* equals the combinations of $\frac{\hat{\delta}}{2}$-Core of all the σ-periodic subgraphs $\{G' \cup G'' \cup ...\}$. Let kCore$(G^*, \frac{\hat{\delta}}{2}) = \{V^*, E^*\}$ and kCore$(\{G' \cup G'' \cup ...\}, \frac{\hat{\delta}}{2}) = \{V', E'\}$, according to the definitions of G^* and G', any node $[u, s, i]$ in V^* will be contained in V' with time support $\{t_s, t_{s+i}...t_{s+i(\sigma-1)}\}$. So, $V^* \subseteq V'$. On the contrary, for a σ-periodic subgraph G' which contains the $\sigma-$DPS, since the $\sigma-$DPS will be contained in $\frac{\hat{\delta}}{2}$-Core of G', any node u in the $\sigma-$DPS satisfies that $d_u(G_j) > \hat{\delta}/2$ for $j = s, s + 1...s + i(\sigma - 1)$. So, $V' \subseteq V^*$. Based on Definition 8, since $V' = V^*$, E^* equals E'. Therefore, kCore$(G^*, \frac{\hat{\delta}}{2})$ = kCore$(\{G' \cup G'' \cup ...\}, \frac{\hat{\delta}}{2})$.

Based on Theorem 5, the $\sigma-$DPS will be in $\frac{\hat{\delta}}{2}$-Core of the periodic subgraph and the degeneracy δ is the number of the maximum k satisfying kCore $\neq \emptyset$. Therefore, the densest graph and the degeneracy in kCore$(G^*, \frac{\hat{\delta}}{2})$ and kCore$(\{G' \cup G'' \cup ...\}, \frac{\hat{\delta}}{2})$ will both be the same. □

According to Theorem 6, Algorithm 4 returns the densest graph in G^*, which is the $\sigma-$DPS of $\mathcal{G}$ in practice.

Theorem 7 (Complexity of Algorithm 4). *For a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{T}|$ timestamps, the de-temporal graph of $\mathcal{G}$ is $G = (V, E)$, the time and space complexity of Algorithm 4 is $O(m'n' \log(\frac{n'^2}{m'}))$ and $O(m')$, where $n' = \frac{|\mathcal{T}|^2}{\sigma}|V|$, $m' = \frac{|\mathcal{T}|^2}{\sigma}|E|$.*

Proof. It needs $O(|\mathcal{T}|^2|E|^2)$ to compute the σ-periodic degeneracy (line 1), and $O(|E|)$ to compute the $\frac{\hat{\delta}}{2}$-Core of G (line 2). In lines 4-14, in worst case, the number of periodic nodes $[u, PN.s, PN.i]$ is $O(\frac{|\mathcal{T}|^2}{\sigma}|V|)$, but it needs σ times (line 11) to determine a σ-periodic node. So the time complexity of constructing the σ-periodic nodes are

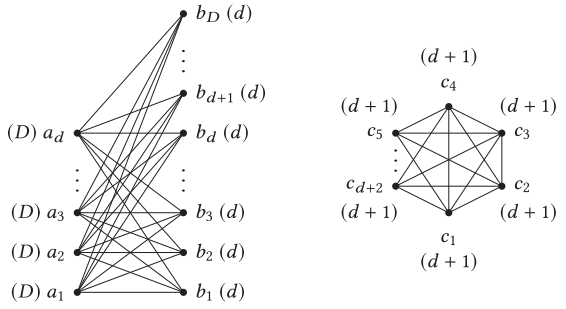


Fig. 4. Special case which achieves the 1/2 approximation [16].

$O(|\mathcal{T}|^2|V|)$. Then, the time complexity of constructing the edges in G^* is $O(|E|)$. Furthermore, computing the densest graph in G^* needs $O(m'n'\log(\frac{n'^2}{m'}))$ according to [12], [13]. Putting all together, the the time complexity of Algorithm 4 is $O(m'n'\log(\frac{n'^2}{m'}))$. Besides, the algorithm needs to maintain the graph G^*, sets $PerioN$ and $StrtT$, which require total $O(m')$ memory. □

5 Approximate Algorithms

Similar to the problem of traditional densest subgraph mining, the problem of mining $\sigma-$DPS can also be computed by an approximation method. The *EDS* [14] method follows the peeling paradigm and achieves an approximation ratio of $1/2$ to find densest subgraph in a static graph. Also, Samir Khuller and Barna Saha [15] show an example where the $1/2$ approximation is tight for greedy peeling. Next, we introduce approximate algorithms for seeking $\sigma-$DPS in the temporal graph $\mathcal{G}$.

According to Lemma 2, the kCore in G with the maximum k will be the $1/2$-approximation answer for the densest subgraph. Recall theorem 2, there holds the following lemma.

Lemma 3. *Given a temporal graph G, the maximum k of the non-empty k$-core$ in all σ-periodic subgraphs, will be 1/2 approximation for density of $\sigma-$DPS.*

Proof. Suppose that $\hat{G}$ is a σ-periodic subgraph which has the largest $\hat{\delta}$. According to Theorem 2, we have $\hat{\delta}/2 \leq \rho(\sigma-\text{DPS}) < \hat{\delta}$. Since k is maximum among the non-empty k$-core$ in all σ-periodic subgraphs, it holds $k \geq \hat{\delta}$. So, $\rho(\sigma-\text{DPS}) < k$. Besides, $\sigma-$DPS is a σ-periodic subgraph with the maximum density, we have $\rho(\sigma-\text{DPS}) \geq \rho(\text{k}-core) \geq k/2$. Therefore, $k/2 \leq \rho(\sigma-\text{DPS}) < k$. □

Based on Lemma 3, there is a question that whether the approximation ratio is theoretically tight. The case in Fig. 4 show that the model of the k$-core$ with maximum k is a tight approximation algorithm in general.

Example 4. As shown in Fig. 4, suppose that the σ-periodic subgraph $G' = B \cup C_1 \cup C_2 ... C_k$ in which B is a $d \times D$ bipartite graph and C_i is a clique of size $d+2$. Consider that $d << D, k \to +\infty$, the density of G' will be $\frac{2dD+(d+1)(d+2)k}{2d+2D+2k(d+2)} \to \frac{d+1}{2}$. However, the density of B is $\frac{dD}{d+D} \approx d$, which is in fact the optimal solution. The approximation algorithm will output the whole graph G', since it starts eliminating nodes of degree d from B, and by doing this, it never sees a subgraph with higher density. Therefore, this example illustrates that the $1/2$ approximation is tight.

Based on theorem 2, we can search the periodic subgraphs to find the kCore with the maximum k to be the optimal approximation for $\sigma-$DPS with an approximation ratio of $1/2$. The process of the algorithm can be modified by changing lines 8-16 in Algorithm 3, as shown in Algorithm 5 below.

Algorithm 5 first repeats the process of lines 1-7 of Algorithm 3 to enumerate the periodic subgraphs. Note that, according to lemma 1, the degeneracy of graph G is no less than that of subgraph S which satisfies $S \subseteq G$. So different from Algorithm 3, in line 7 of Algorithm 5, the $DelT$ will be added into time T' if the degeneracy of $G_{t'}$ is less than $\bar{\delta}$. Finally, the algorithm searches all the possible periodic subgraphs and finds one σ-periodic subgraph which has the largest degeneracy $\bar{\delta}$, then it returns the kCore of this subgraph with $k = \bar{\delta}$.

Recall Section 4.2, we use a generated graph G^* to search the $\sigma-$DPS. However, can we use G^* to find the kCore with the maximum k in all periodic subgraphs? The answer is yes and Algorithm 6 shows the process of using G^* to search the $\sigma-$DPS. Similar as the proof of Theorem 6, we can prove that given a temporal graph $\mathcal{G}$, parameter σ and a generated graph G^* of $\mathcal{G}$, the maximum kCore in G^* equals the maximum kCore in all the σ-periodic subgraphs.

Algorithm 6 not need to compute σ-periodic degeneracy in line 1 (it is the final answer of the algorithm). It may guess an integer in the range of 5-10 in practice for the σ-periodic degeneracy of $\mathcal{G}$. Then, the algorithm computes the candidate nodes set G_c by kCore$(G, \bar{\delta})$ and generates periodic nodes from G_c into G^*, which are similar as lines 4-18 in Algorithm 4. Note that, all parameter $\frac{\delta}{2}$ are modified into $\bar{\delta}$ according to lemma 1. Finally, the algorithm returns the kCore in G^* with the maximum k.

Algorithm 5. DPS$-A_1(\mathcal{G}, \sigma)$

Input: Temporal graph $\mathcal{G} = \{G_1, ...G_{|\mathcal{T}|}\}$, parameter σ
Output: Approximate $\sigma-$DPS in $\mathcal{G}$
1: Lines 1-7 in Algorithm 3;
2: **if** $PS.l = \sigma$ **then**
3: $\quad \rho \leftarrow$ Degeneracy$(PS.G)$;
4: $\quad$ **if** $\rho > \bar{\delta}$ **then**
5: $\quad\quad \bar{\delta} \leftarrow \rho$; $DPS \leftarrow PS.G$;
6: $\quad\quad$ **for** $t' \leftarrow t : |\mathcal{T}|$ **do**
7: $\quad\quad\quad$ **if** Degeneracy$(G_{t'}) < \bar{\delta}$ **then** $DelT.add(t')$;
8: Lines 17-18 in Algorithm 3;
9: **return** kCore$(DPS, \bar{\delta})$;

Algorithm 6. DPS$-A_2(\mathcal{G}, \sigma)$

Input: Temporal graph $\mathcal{G} = \{G_1, ...G_{|\mathcal{T}|}\}$, parameter σ
Output: Approximate $\sigma-$DPS in $\mathcal{G}$
1: $\bar{\delta} \leftarrow$ guess an integer for the σ-periodic degeneracy of $\mathcal{G}$ (5-10 in practice);
2: $G_c = \{V_c, E_c\} \leftarrow$ kCore$(G, \bar{\delta})$;
3: Lines 4-18 in Algorithm 4 (modify $\frac{\bar{\delta}}{2}$ to $\bar{\delta}$ in line 6);
4: **return** kCore$(G^*,$ Degeneracy$(G^*))$;

Theorem 8 (Complexity of Algorithms 5 and 6). *For a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{T}|$ timestamps, the de-temporal graph of $\mathcal{G}$ is $G = (V, E)$, the time complexity of Algorithm 5*

TABLE 1
Statistics of Datasets

Dataset	$\lvert V\rvert = n$	$\lvert E\rvert = m'$	$\lvert\mathcal{E}\rvert = m$	$d_{\max}$	$\lvert\mathcal{T}\rvert$	Time scale
Chess	7,301	55,899	63,689	233	101	month
Lkml	26,885	159,996	328,092	14,172	96	month
Enron	86,836	296,952	501,510	2,156	87	month
DBLP	1,729,816	8,546,306	12,007,380	5,980	78	year
YTB	3,223,589	9,376,594	12,218,755	129,819	225	day
FLK	2,302,925	22,838,276	24,690,648	28,276	197	day
MO	24,759	187,986	294,293	5,556	2,351	day
AU	157,222	455,691	549,914	7,325	2,614	day
WT	1,094,018	2,787,967	4,010,611	214,518	2,321	day
MO2	24,759	187,986	350,798	6,500	56,409	hour
AU2	157,222	455,691	631,151	8,373	62,732	hour
WT2	1,094,018	2,787,967	4,702,689	233,313	55,690	hour

and Algorithm 6 are $O(|T|^2 m)$ *and* $O(\frac{|T|^2}{\sigma} m)$*, respectively. The space complexity of Algorithm 5 and Algorithm 6 are both* $O(m + \frac{|T|^2}{\sigma})$*, in which* $n = |V|, m = |E|$.

Proof. In Algorithm 5, line 1 needs $O(|T|^2 m)$ time, and lines 2-7 also require $O(|T|^2 m)$ time. Besides, it maintains the temporal graph $\mathcal{G}$, sets $PerioSub$ and $StrtT$, so its space complexity is $O(m + \frac{|T|^2}{\sigma})$.

In Algorithm 6, line 2 takes $O(m)$ time, and line 3 requires $O(\frac{|T|^2}{\sigma} m)$ since the generated graph G^* may have $O(\frac{|T|^2}{\sigma} m)$ edges. Besides, it maintains the generated graph G^*, sets $PerioN$ and $StrtT$, thus its space complexity is $O(\frac{|T|^2}{\sigma} m + \frac{|T|^2}{\sigma}) = O(\frac{|T|^2}{\sigma} m)$. However, we can release memories of some periodic nodes who are not in the k$-core$ so its memory overhead is acceptable in practice.□

6 Experiments

In this section, we conduct extensive experiments to evaluate the proposed algorithms. We implement seven different algorithms for comparison:

- MPC [8] is a comparison algorithm for computing the maximum clique in the σ-periodic subgraphs.
- PERC [9] is a comparison algorithm that searches periodic communities in the temporal networks and can optimize a model based on periodic behavior.
- DPS$-B$ is a baseline which computes the $\sigma-$DPS using the framework shown in Algorithm 1, but it enumerates all periodic subgraphs to find the densest subgraphs.
- DPS$-P$ is the implementation of Algorithm 3 which searches the $\sigma-$DPS and uses the σ-periodic degeneracy to prune the considering periodic subgraphs.
- DPS$-P+$ is the implementation of Algorithm 4 which finds the $\sigma-$DPS by searching a generated graph and requires less computation of set intersection and maximum flow mining.
- DPS$-A_1$ is the implementation of Algorithm 5 which uses the pruning algorithm similar to Algorithm 3 for finding the approximate $\sigma-$DPS.
- DPS$-A_2$ is the implementation of Algorithm 6 which uses a generated graph similar to Algorithm 4 for finding the approximate $\sigma-$DPS.

All algorithms are implemented in Python and all the experiments are conducted on a server of Linux kernel 4.4 with Intel Core(TM) i5-8400@3.80GHz and 32 GB Memory. When quantity measures are evaluated, the test was repeated over 5 times and the average is reported here.

Datasets. We evaluate our algorithms on 12 different real-world temporal networks. The detailed statistics of datasets are summarized in Table 1, where $d_{\max}$ denotes the maximum number of temporal edges associated with a node, and $|\mathcal{T}|$ represents the number of snapshots. All the snapshots are simple, undirected and unweighted graphs. Chess[1] is a network that represents each pair of chess players playing game together from 1998 to 2006. Lkml[1] is a communication network of the Linux kernel mailing list from 2001 to 2011. Enron[1] is an email communication network between employees of Enron from 1999 to 2003. DBLP[2] is a collaboration network of authors in DBLP from 1940 to Feb. 2018. Youtube[3] (YTB for short) and Flickr[1] (FLK) are friendship networks of users in Youtube and Flickr, respectively. MathOverflow[3] (MO), AskUbuntu[3] (AU) are temporal networks of interactions on the stack exchange web site mathoverflow.net and askubuntu.com, respectively. WikiTalk[3] (WT) is a temporal network representing the interactions among Wikipedia users. Unlike MO, AU, WT collect snapshots by day, MO2, AU2, WT2 are transformed graphs of MO, AU, WT which generate snapshots by hour, so their $|\mathcal{T}|$ are much larger.

Goodness Metrics. Most existing metrics (e.g., modularity) for measuring the dense subgraph quality are tailored for traditional graphs. Motivated by *density*, *cohesiveness*, *clustering coefficient* and *separability* [17], we introduce two goodness metrics evaluating communities for temporal graphs. Let C be a dense subgraph computed by different algorithms.

Average Separability (AS) captures the intuition that a good community is well-separated from the rest of the network, meaning that they have relatively few across edges between C and the rest of the network: $\text{AS} \triangleq \lfloor \frac{|\{(u,v,t)\in\mathcal{E}:u\in C,v\in C\}|/|C|}{|S=\{(u,v,t)\in\mathcal{E}:u\in C,v\notin C\}|/|S|} \rfloor$, which measures the ratio between the internal average density and external average density.

1. http://konect.cc/
2. https://dblp.uni-trier.de/xml/
3. http://snap.stanford.edu/data/index.html

TABLE 2
Evaluation Methods for the Dense Subgraphs in Temporal Graph

Metric	Formulation	Intuition
AS	$\left[\frac{\lvert\{(u,v,t)\in\mathcal{E}:u\in C,v\in C\}\rvert/\lvert C\rvert}{\lvert S=\{(u,v,t)\in\mathcal{E}:u\in C,v\notin C\}\rvert/\lvert S\rvert}\right]$	#temporal edges inside community C/ #temporal edges outside C
AD	$\left[\frac{\sum_{v_i\in C} deg_{\mathcal{G}_C}(v_i)}{\lvert C\rvert}\right]$	sum of nodes' degrees inside community C / #nodes in C
AC	$\sum_{C_i\in\mathcal{C}} max_{S\subseteq C_i}\phi(S)$	maximum cut-edges which can split community C into S and $C\setminus S$
ACC	$\sum_{v_j\in C}\frac{\#edge(N(v_j,C))}{d_C(v_j)}/\lvert\mathcal{C}\rvert$	avg$_{v_j\in C}$(#common neighbors of v_j/ #temporal degree of v_j inside C)

*Average Density (*AD*)* builds on intuition that good communities are well connected. It measures the fraction of the temporal edges that appear between the nodes in C: AD $\triangleq [\frac{\sum_{v_i\in C} deg_{\mathcal{G}_C}(v_i)}{\lvert C\rvert}]$, where $deg_{\mathcal{G}_C}(v_i)$ denotes the number of temporal edges that are associated with v_i in the community C.

*Average Cohesiveness (*AC*)* characterizes the internal structure of a community. Intuitively, a good community should be internally well and evenly connected, i.e., it should be relatively hard to split a community into two sub communities. We characterize this by the conductance of the internal cut and adapt it into temporal graph: AC $\triangleq \sum_{C_i\in\mathcal{C}} max_{S\subseteq C_i}\phi(S)$, where $\phi(S)$ is the conductance of S measured in the induced temporal subgraph by S.

*Average Clustering Coefficient (*ACC*)* is based on the premise that network communities are manifestations of locally inhomogeneous distributions of edges, because pairs of nodes with common neighbors are more likely to be connected with each other: ACC $\triangleq \sum_{v_j\in C}\frac{\#edge(N(v_j,C))}{d_C(v_j)}/\lvert\mathcal{C}\rvert$, where $\#edge(N(v_j,C))$ is the number of temporal edges in C whose two end nodes are v_j's neighbors and $d_C(v_j)$ denotes the number of temporal edges that are associated with v_j in the dense subgraph C.

Table 2 shows the brief introductions of the above evaluation metrics. As shown, AS measures the ratio between the number of temporal edges inside community C and the number of temporal edges outside C; AD measures the ratio between the sum of nodes' degrees inside community C and the number of nodes in C; AC is the maximum number of cut edges which can split community C into S and $C\setminus S$; ACC is the average value of the ratio between the number of common neighbors of v_j in C and the number of temporal degree of v_j inside C. Unless otherwise specified, in the following effectiveness testings, the evaluation values are normalized so each maximum single value is 1.

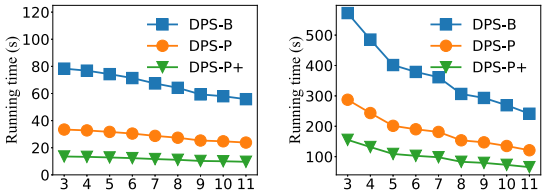
TABLE 3
Running Time (s) of Different Algorithms (INF: $>$ 3 hours)

Dataset	MPC	PERC	DPS$-B$	DPS$-P$	DPS$-P+$	DPS$-A_1$	DPS$-A_2$
Chess	11.45	21.45	8.32	1.32	**0.78**	**0.30**	0.40
Lkml	35.06	45.23	20.32	10.4	**9.21**	3.23	**2.36**
Enron	56.19	104.2	78.32	33.41	**13.54**	4.21	**3.25**
DBLP	405.23	1602.32	572.54	287.32	**155.34**	42.28	**28.95**
YTB	306.53	2653.23	1123.13	559.52	**226.92**	72.28	**67.43**
FLK	417.53	3234.23	1323.32	766.4	**322.21**	172.28	**78.52**
MO	835.06	6713.241	2445.14	1000.23	**434.19**	30.15	**13.71**
AU	1203.32	10232.23	3121.31	1599.78	**766.89**	53.32	**23.36**
WT	2203.32	inf	8021.31	4865.87	**1445.23**	130.15	**57.65**
MO2	inf	inf	inf	4508.23	**1834.19**	430.15	**116.45**
AU2	inf	inf	inf	8599.78	**3458.89**	613.32	**238.24**
WT2	inf	inf	inf	inf	**10543.32**	1030.15	**557.65**

6.1 Efficiency Evaluation

Exp-1. Running Time of All the Algorithms. Table 3 evaluates the running time of MPC, PERC, DPS$-B$, DPS$-P$, DPS$-P+$, DPS$-A_1$ and DPS$-A_2$ with parameters $\delta = 3$. Similar results can also be observed with the other parameter settings. From Table 3, it is obvious that DPS$-P+$ is much more efficient than MPC, PERC on all datasets. This is because DPS$-P+$ can compute the $\sigma-$DPS in $O((n')^2 \log n')$ where $n' = \frac{\lvert T\rvert^2}{\sigma} \times \lvert V\rvert$, which is much quicker than the current algorithms MPC and PERC. Compared with DPS$-B$ and DPS$-P$, DPS$-P+$ is also much faster because it prunes the enumerations of the periodic subgraphs and it overcomes some limitations as discussed in Section 4.2. However, since the time complexity of the algorithm DPS$-P+$ is proportional to the square of $\lvert T\rvert$, the algorithm still has some performance deficiencies when $\lvert T\rvert$ is too large. We can see that it needs about 10 thousands seconds to run DPS$-P+$ in WT2, but it only needs about 1 thousand seconds in WT (WT2 have same nodes as WT, but WT2 generate snapshots by hour so it has lager $\lvert T\rvert$). Moreover, the two approximation algorithms DPS$-A_1$ and DPS$-A_2$, perform better than the best exact algorithm DPS$-P+$. It is the reason that DPS$-A_1$ and DPS$-A_2$ not need to invoke the maximum flow computation, which is quite time-consuming in practice. However, DPS$-A_2$ is much faster than DPS$-A_1$ on almost all datasets, because DPS$-A_1$ needs to compute the interactions of the snapshots. According to Table 3, our proposed algorithms are efficient to mine the $\sigma-$DPS in large temporal networks. For example, on YTB, DPS$-P+$ takes 226.92 seconds and the approximation algorithm DPS$-A_2$ only consumes 67.43 seconds. On WT, we can see that MPC takes 2,203 seconds, and PERC takes more than 3 hours, but our proposed DPS$-P+$ takes merely 1,445 seconds and MBC+ only takes 57 seconds. These results confirm that our proposed algorithms are very efficient on large real-life temporal networks.

Exp-2. Running Time of Varying the Parameter σ. Fig. 5 shows the running time of DPS$-B$, DPS$-P$ and DPS$-P+$ during varying parameter σ on Enron and DBLP. Similar results can also be observed on other datasets. It can be seen from Fig. 5 that DPS$-P+$ is faster than DPS$-B$ and DPS$-P$ under all parameter settings. In Fig. 5b, we can see that the running time of DPS$-P+$ decreases faster when σ ranges from 3 to 7 than σ ranges from 7 to 9. This is because that



(a) Enron (b) DBLP

Fig. 5. Running time of varying the parameter σ.

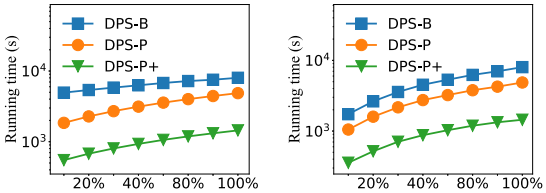
the time complexity of the algorithm DPS$-P+$ is inversely proportional to σ, and the pruning technique will be less powerful when σ is smaller. With the increase of σ, the running time of all the three algorithms decreases. These results confirm that the time complexity of DPS$-B$, DPS$-P$ and DPS$-P+$ are inversely proportional to σ.

Exp-3. Scalability of All the Algorithms. Fig. 6 shows the scalability of DPS$-B$, DPS$-P$ and DPS$-P+$ on WT dataset. Similar results can also be observed on other datasets. We generate ten temporal subgraphs by randomly selecting 10%-100% temporal edges or 10%-100% timestamps, and evaluate the running times of DPS$-B$, DPS$-P$ and DPS$-P+$ on these subgraphs. As shown in Fig. 5, the running time increases smoothly with increasing number of edges or size of $|\mathcal{T}|$. These results suggest that our proposed algorithms are scalable when handling large temporal graphs.

Exp-4. Memory Overhead. Table 4 shows the memory usage of DPS$-P$ and DPS$-P+$ on different datasets. We can see that the memory usage of DPS$-P+$ is higher than the size of DPS$-P$, because DPS$-P$ only needs to store *PerioSub* in Algorithm 1 and 3 but DPS$-P+$ needs to store all the periodic nodes *PerioN* in Algorithm 4. In practice, when executing Algorithm 4, we can release memories of some periodic nodes who are not in the k$-$*core* (line 19). Therefore, on large datasets, the memory usage of DPS$-P+$ is typically smaller than ten times the size of the temporal graph. For instance, on WT, DPS$-P+$ consumes 2,923.2MB memory while the graph needs 354.3MB. These results indicate that DPS$-B$, DPS$-P$ and DPS$-P+$ all achieve nearly linear space complexity, which confirms our theoretical analysis in Section 4.

6.2 Effectiveness Evaluation

Exp-5. Effectiveness of MPC, PERC *and* DPS$-P+$ Fig. 7 shows the four goodness results of the model MPC, PERC and DPS$-P+$ with $\sigma = 3$ on all the datasets. We only choose DPS$-P+$ for comparison because DPS$-B$, DPS$-P$ and DPS$-P+$ all output $\sigma-$DPS so they have the same results. Intuitively, a good periodic community should have high



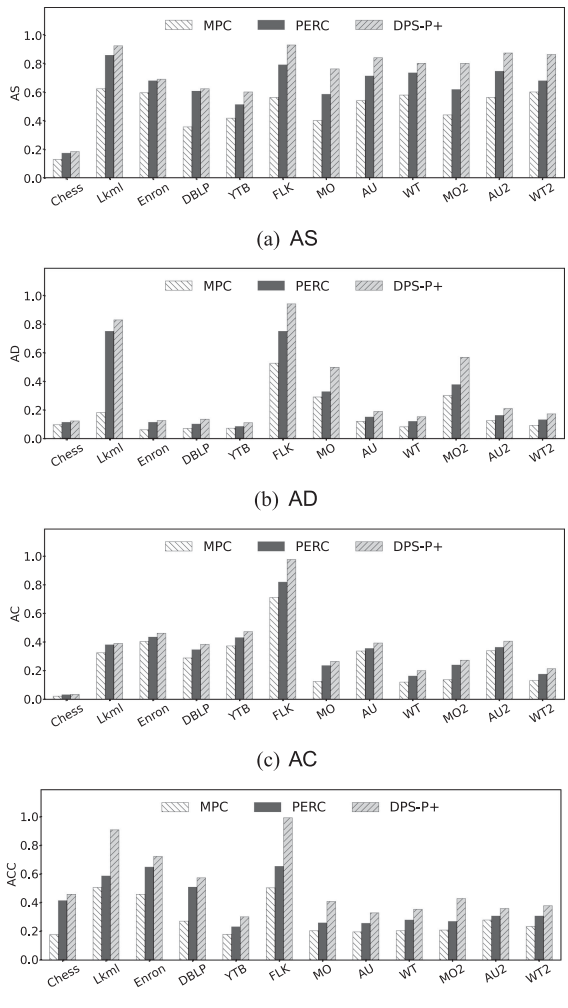
(a) percents of edges (b) percents of $|\mathcal{T}|$

Fig. 6. Scalability of the algorithms.

TABLE 4
Memory Overhead of DPS$-P$ and DPS$-P+$

	Graph in Memory	Memory of DPS$-P$	Memory of DPS$-P+$
Chess	3.5MB	13.2MB	58.2MB
Lkml	20.1MB	47.4MB	146.2MB
Enron	53.3MB	127.6Mb	353.2MB
DBLP	1,064.5MB	2,532.2MB	4,324.4MB
YTB	718.5MB	1,561.5MB	3,425.8MB
FLK	1,412.1MB	3,424.2MB	5,802.1MB
MO	14.32MB	46.68MB	108.45MB
AU	64.22MB	167.50MB	580.8MB
WT	354.3MB	1074.56MB	2,923.2MB
MO2	51.53MB	45.23MB	92.75MB
AU2	155.96MB	150.32MB	459.2MB
WT2	1342.3MB	1023.23MB	2,923.2MB

AS, AD, AC and ACC values. As can be seen, DPS$-P+$ is obviously better than the two baselines in terms of AS, AD, AC and ACC metrics. The reason is that DPS$-P+$ seeks the most densest periodic subgraphs which certainly have the best average density (AD). In addition, with the higher inner density, DPS$-P+$ is hard to be separated and it has higher AS, AC and ACC. We can also find that PERC is much better than MPC in terms of all the metrics. This is because that the real



(a) AS (b) AD (c) AC (d) ACC

Fig. 7. Effectiveness of the algorithms.

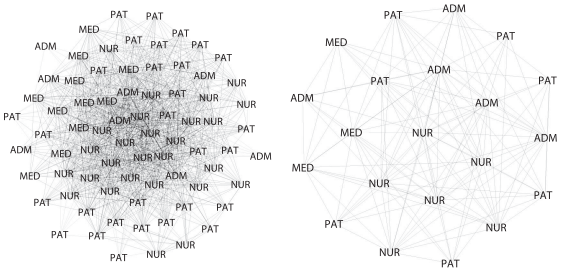
TABLE 5
Effectiveness of DPS$-P+$, DPS$-A_1$ and DPS$-A_2$ ($\sigma = 3$)

	$\rho_1 = \rho$(DPS$-P+$)	$\rho_2 = \rho$(DPS$-A_1$/DPS$-A_2$)	$\frac{\rho_2}{\rho_1}$
Chess	5.6	4.5	80.4%
Lkml	9.4	8.4	89.4%
Enron	8.3	7.2	86.8%
DBLP	13.2	11.2	84.9%
YTB	15.3	13.6	88.9%
FLK	14.8	12.4	83.8%
MO	8.5	6.9	81.2%
AU	7.6	5.3	69.7%
WT	9.4	7.3	77.7%
MO2	4.5	3.5	77.8%
AU2	3.7	3.1	83.7%
WT2	7.5	6.0	80.0%

periodic cliques in the temporal networks are always too small, so the MPC can not output a subgraph of large size. As shown in Figs. 7a, 7b, 7c, and 7d, we can observe that the AS, AD, AC and ACC values in FLK are larger than in the other datasets, this is because the dataset FLK ranks top in terms of data size and density. Interestingly, the AS, AD and ACC in Lkml are obviously better than in the other datasets (expect in FLK), but the AC value in Lkml is slightly larger. It is the reason that although Lkml has larger density, the graph size in Lkml is small such that the conductance of the internal cut is not large in Lkml.

Exp-6. Effectiveness of DPS$-P+$, DPS$-A_1$ *and* DPS$-A_2$. Table 5 shows the density results of the approximation algorithms DPS$-A_1$ and DPS$-A_2$ V.S. the exact algorithm DPS$-P+$. The first column ρ_1 is the density of the output of algorithm DPS$-P+$, the second column ρ_2 is the density of the output of algorithm DPS$-A_1$ (or DPS$-A_2$ since they output the same results), and the third column is the percentage value of ρ_2/ρ_1. We can see that in all the testings, the density of the output of DPS$-A_1$/DPS$-A_2$ achieve $1/2$ approximation since all ρ_2/ρ_1 are inside the range of $50\% - 100\%$. In practice, the approximation algorithms perform great since they almost achieve the 80% percentage of the optimal answer. The above results indicate the theoretical analysis in Section 5 and show that the approximation algorithms achieve better than the theoretical approximation ratio in the real datasets.

Exp-7. Case Study on HOSPITAL. The dataset HOSPITAL[4] is the temporal network of face-to-face contacts between patients and health-care workers in a hospital at Lyon, France, from Monday, December 6, 2010 at 1:00 pm to Friday, December 10, 2010 at 2:00 pm. It includes 46 health-care workers (the doctors, nurses and administrative staffs are labeled as *MED*, *NUR* and *ADM*, respectively) and 29 patients (labeled as *PAT*). In this experiment, we set the time interval to be one hour. Next, we will show mining the densest periodic subgraph can help to seek the periodic treatments in the HOSPITAL dataset. In real-life, the treatment of a disease often takes several courses of treatments, so the interactions between doctors and patients are often periodic. Therefore, mining the periodic and densest subgraph can help to find the information about these periodic



(a) The densest subgraph in the de-temporal graph (b) The σ-DPS ($\sigma = 5$)

Fig. 8. Case study on HOSPITAL.

treatments. At first, as can be seen in Fig. 8a, we seek the densest subgraph in the de-temporal graph, which is a most possible community in the interaction networks between the health-care workers and the patients. In Fig. 8a, as we do not consider the temporal information, the resulting community involves almost all the users (the dataset contains 75 users but Fig. 8a contains 70 users), so we are hard to identify the activity of the possible periodic treatments in the networks. However, in Fig. 8b we seek the densest σ-periodic subgraph with $\sigma = 5$, which shows a subgraph with 5 nodes of *ADM*, 3 nodes of *MED*, 6 nodes of *NUR* and 7 nodes of *PAT*. We also find that the health-care workers and the patients are periodically contacted at December 8, 2010 from 10:00 am to 14 pm. Therefore, the densest σ-periodic subgraph in Fig. 8b is more likely to be a periodic treatment at December 8, 2010 among doctors, nurses administrative staffs and patients in the hospital. The results indicates that our model can find the periodic medical behavior in the hospital contact temporal network.

7 Related Work

In this section, we summarize the existing related algorithms for identifying densest periodic subgraphs in the temporal graph, which is related to the references below.

Densest Subgraph in Static Graph. Finding the densest subgraphs in static graph is a well-studied graph mining problem. It is well known that Goldberg's parametric flow algorithm [10] can find the maximal densest subgraph in polynomial time by invoking $O(\log\ n)$ max-flow computations. Moreover, as shown in [14], a linear time greedy algorithm proposed by Asashiro et al. [18] can obtain a $1/2$-approximation densest subgraph. However, when we restrict the size of the densest subgraph [15], [19] or redefine the considered density [20], the problem becomes NP-hard. For example, one recent work [21] proposes a nearly-linear-time algorithm to appropriately restrict the size of the densest subgraph by defining a concave function. Moreover, if the density is redefined as $d' = |E|/C^2_{|V|}$, finding the subgraph with the largest d' is to find the maximum clique, which is known to be NP-hard [22]. Another interesting variant of the densest subgraph model, termed optimal quasi-clique, based on a new definition of the density function, is also NP-hard [23]. In addition to the above studies, there are related work on other models based on various graph properties [11], [13], [16], [24], [25], [26], [27], [28]. In our work, we study the

4. http://www.sociopatterns.org/datasets/hospital-ward-dynamic-contact-network/

problem of mining the densest subgraphs which occur in temporal graphs periodically and proposed unprecedented targeted solutions.

Densest Subgraph in Non-Static Graph. Recently, some researches study the problem of mining densest subgraph in non-static graph. Some of the studies maintain the densest subgraph in a highly dynamic graphs [29], [30], [31]. By capturing the dynamic property, we can have fast algorithms for approximation factors better than 1/2. One approach towards this is to sparsify the graph in a way that maintains subgraph densities within a factor of $1-\epsilon$, and run the exact algorithm on the sparsifier [32], [33]. A second approach is via numerical methods to solve positive LPs approximately [34], [35] which can find an $(1-\epsilon)$-approximate solution. Others, including our work, focus on mining the densest subgraph with temporal features [36], [37], [38], [39], [40], [41]. For example, Liu et al. [36] studied the problem of finding densest lasting-subgraphs in large dynamic graphs, which considered the time duration of the subgraph pattern. Ma et al. [37] investigated the densest subgraph mining problem in weighted temporal graphs. Rozenshtein et al. [39] searched for a partition of the timestamps into k non-overlapping intervals, so that the intervals span subgraphs with maximum total density. Miyauchi et al. [40] proposed the model of robust densest subgraph with sampling oracle in temporal networks. Chu et al. [41] devised an algorithm which can find a subgraph that accumulates its density at the fastest speed in temporal networks. Unlike all these studies, in our work, we propose a model which aims to find densest periodic subgraphs in temporal graphs. Based on our model, we are able to identify all the periodic and densest regions of a temporal graph in polynomial time, which cannot be found by the above models.

Other Community Models in Temporal Graph. In addition to the above two categories, there are several related works on other cohesive subgraph mining models in temporal graphs [42], [43], [44], [45], [46], [47], for example: *(i) Temporal Core Model:* Galimberti et al. [42] proposed temporal span-cores, in which each node has minimum degree in a specific time interval; Wu et al. [48] studied the core decomposition problem in temporal networks; Yu et al. [49] computed the historical k-cores in the graph snapshots over the time window; Li et al. [43] developed an algorithm to detect persistent cores in a temporal graph. *(ii) Temporal Clique Model:* Qin et al. [8] proposed a model for seeking periodic cliques in a temporal graph. Yang et al. [50] studied a problem of finding a set of diversified quasi-cliques from a temporal graph. *(iii) Temporal Subgraph Model:* Yang et al. [51] proposed an algorithm to detect frequent changing components in temporal graph; Huang et al. [52] investigated the minimum spanning tree problem in temporal graphs; Gurukar et al. [53] presented a model to identify the recurring subgraphs that have similar sequence of information flow. However, the above works do not study the problem of mining densest periodic subgraph in temporal graphs.

8 Conclusion

In this work, we study the issues of mining densest periodic subgraphs, which is the most densely connected periodic behavior in a temporal graph. We propose a novel model, called densest σ-periodic subgraph, to characterize the densest periodic subgraph in a temporal graph. To find it, we first develop a basic algorithm which can enumerate all the periodic subgraph and then invoke maximum flow algorithm to search the densest subgraph. Then, we present an improved algorithm with several novel pruning techniques to improve the efficiency. Subsequently, we develop a greedy algorithm which can compute the approximate densest σ-periodic subgraph. The experimental results on several real-life temporal networks demonstrate the efficiency, scalability and effectiveness of our algorithms.

There are a lot of problems to be solved in the future, to name but a few: *(i)* We can observe that a slight insertion or deletion of edge may not change the periodic subgraph. As a result, in the future we can study the problem of maintaining the densest periodic subgraph in fully dynamic graphs. *(ii)* We can consider to find the densest subgraph which is quasi-periodic. In real life, the events are always quasi-periodically, which means the time intervals between two adjacent events are near to a constant. For example, every-year birthday parties are usually celebrated with an interval of 365 days, but there also has an interval of 366 days when considering the leap years; monthly meetings are usually held at intervals of 28 to 31 days, because the number of days included in each month is not fixed, but we usually hold monthly meetings on a fixed day. As a result, it is significant to mining the quasi-periodic densest subgraph in the temporal network, since it can represent more periodic behavior in real life.

References

[1] P. Holme, "Modern temporal network theory: A colloquium," *Eur. Phys. J. B*, vol. 88, no. 9, pp. 1–30, 2015.

[2] P. Holme, "Analyzing temporal networks in social media," *Proc. IEEE*, vol. 102, no. 12, pp. 1922–1933, Dec. 2014.

[3] W. Han et al., "Chronos: A graph engine for temporal graph analysis," in *Proc. Eur. Conf. Comput. Syst.*, 2014, pp. 1:1–1:14.

[4] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye, "Mining periodic behaviors for moving objects," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2010, pp. 1099–1108.

[5] J. K. Bizley, K. M. M. Walker, A. J. King, and J. W. H. Schnupp, "Neural ensemble codes for stimulus periodicity in auditory cortex," *J. Neurosci.*, vol. 30, no. 14, pp. 5078–5091, 2010.

[6] P. Vanhems et al., "Estimating potential infection transmission routes in hospital wards using wearable proximity sensors," *PLoS One*, vol. 8, 2013, Art. no. e73970.

[7] M. Lahiri and T. Y. Berger-Wolf, "Periodic subgraph mining in dynamic networks," *Knowl. Inf. Syst.*, vol. 24, no. 3, pp. 467–497, 2010.

[8] H. Qin, R. Li, G. Wang, L. Qin, Y. Cheng, and Y. Yuan, "Mining periodic cliques in temporal networks," in *Proc. IEEE Int. Conf. Data Eng.*, 2019, pp. 1130–1141.

[9] L. Zhang, A. Gorovits, and P. Bogdanov, "Perceids: Periodic community detection," in *Proc. IEEE Int. Conf. Des. Mining*, 2019, pp. 816–825.

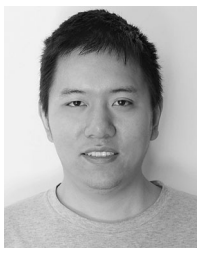
[10] A. V. Goldberg, "Finding a maximum density subgraph," EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/CSD-84-171, 1984. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1984/5956.html

[11] L. Qin, R. Li, L. Chang, and C. Zhang, "Locally densest subgraph discovery," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2015, pp. 965–974.

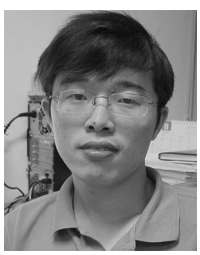
[12] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan, "A fast parametric maximum flow algorithm and applications," *SIAM J. Comput.*, vol. 18, no. 1, pp. 30–55, 1989.

[13] L. Chang and M. Qiao, "Deconstruct densest subgraphs," in *Proc. Int. World Wide Web Conf.*, 2020, pp. 2747–2753.

[14] M. Charikar, "Greedy approximation algorithms for finding dense components in a graph," in *Proc. Int. Workshop Approximation Algorithms Combinatorial Optim.*, 2000, pp. 84–95.
[15] S. Khuller and B. Saha, "On finding dense subgraphs," in *Proc. Int. Colloq. Automata, Lang. Program.*, 2009, pp. 597–608.
[16] D. Boob et al., "Flowless: Extracting densest subgraphs without flow computations," in *Proc. Int. World Wide Web Conf.*, 2020, pp. 573–583.
[17] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proc. Int. Conf. Des. Mining*, 2012, pp. 745–754.
[18] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama, "Greedily finding a dense subgraph," *J. Algorithms*, vol. 34, no. 2, pp. 203–221, 2000.
[19] R. Andersen and K. Chellapilla, "Finding dense subgraphs with size bounds," in *Proc. Int. Workshop Algorithms Models Web-Graph*, 2009, pp. 25–37.
[20] A. Faragó and Z. R. Mojaveri, "In search of the densest subgraph," *Algorithms*, vol. 12, no. 8, 2019, Art. no. 157.
[21] Y. Kawase and A. Miyauchi, "The densest subgraph problem with a convex/concave size function," *Algorithmica*, vol. 80, no. 12, pp. 3461–3480, 2018.
[22] Z. Ertem, E. Lykhovyd, Y. Wang, and S. Butenko, "The maximum independent union of cliques problem: Complexity and exact approaches," *J. Glob. Optim.*, vol. 76, no. 3, pp. 545–562, 2020.
[23] C. E. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. A. Tsiarli, "Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees," in *Proc. Int. Conf. Knowl. Discov. Data Mining*, 2013, pp. 104–112.
[24] B. Sun, M. Dansich, T. H. Chan, and M. Sozio, "KClist++: A simple algorithm for finding k-clique densest subgraphs in large graphs," *Proc. VLDB Endowment*, vol. 13, no. 10, pp. 1628–1640, 2020.
[25] Y. Fang, K. Yu, R. Cheng, L. V. S. Lakshmanan, and X. Lin, "Efficient algorithms for densest subgraph discovery," *Proc. VLDB Endowment*, vol. 12, no. 11, pp. 1719–1732, 2019.
[26] A. E. Sariyüce and A. Pinar, "Fast hierarchy construction for dense subgraphs," *Proc. VLDB Endowment*, vol. 10, no. 3, pp. 97–108, 2016.
[27] A. E. Sariyüce, C. Seshadhri, and A. Pinar, "Local algorithms for hierarchical dense subgraph discovery," *Proc. VLDB Endowment*, vol. 12, no. 1, pp. 43–56, 2018.
[28] C. Ma, Y. Fang, R. Cheng, L. V. S. Lakshmanan, W. Zhang, and X. Lin, "Efficient algorithms for densest subgraph discovery on large directed graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 1051–1066.
[29] A. Epasto, S. Lattanzi, and M. Sozio, "Efficient densest subgraph computation in evolving graphs," in *Proc. Int. World Wide Web Conf.*, 2015, pp. 300–310.
[30] A. Angel, N. Koudas, N. Sarkas, and D. Srivastava,, "Dense subgraph maintenance under streaming edge weight updates for real-time story identification," *Proc. VLDB Endowment*, vol. 5, no. 6, pp. 574–585, 2012.
[31] S. Sawlani and J. Wang, "Near-optimal fully dynamic densest subgraph," in *Proc. Annu. ACM SIGACT Symp. Theory Comput.*, 2020, pp. 181–193.
[32] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak, "Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture," in *Proc. Annu. ACM SIGACT Symp. Theory Comput.*, 2015, pp. 21–30.
[33] S. Bhattacharya, M. Henzinger, D. Nanongkai, and C. E. Tsourakakis, "Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams," in *Proc. Annu. ACM SIGACT Symp. Theory Comput.*, 2015, pp. 173–182.
[34] B. Bahmani, A. Goel, and K. Munagala, "Efficient primal-dual graph algorithms for MapReduce," in *Proc. Int. Workshop Algorithms Models Web-Graph*, 2014, pp. 59–78.
[35] H. Su and H. T. Vu, "Distributed dense subgraph detection and low outdegree orientation," in *Proc. Int. Symp. Distrib. Comput.*, 2020, pp. 15:1–15:18.
[36] X. Liu, T. Ge, and Y. Wu, "Finding densest lasting subgraphs in dynamic graphs: A stochastic approach," in *Proc. IEEE Int. Conf. Data Eng.*, 2019, pp. 782–793.
[37] S. Ma, R. Hu, L. Wang, X. Lin, and J. Huai, "Fast computation of dense temporal subgraphs," in *Proc. IEEE Int. Conf. Data Eng.*, 2017, pp. 361–372.
[38] Y. Wu, R. Jin, X. Zhu, and X. Zhang, "Finding dense and connected subgraphs in dual networks," in *Proc. IEEE Int. Conf. Data Eng.*, 2015, pp. 915–926.
[39] P. Rozenshtein, F. Bonchi, A. Gionis, M. Sozio, and N. Tatti, "Finding events in temporal networks: Segmentation meets densest-subgraph discovery," in *Proc. Int. Conf. Des. Mining*, 2018, pp. 397–406.
[40] A. Miyauchi and A. Takeda, "Robust densest subgraph discovery," in *Proc. IEEE Int. Conf. Des. Mining*, 2018, pp. 1188–1193.
[41] L. Chu, Y. Zhang, Y. Yang, L. Wang, and J. Pei, "Online density bursting subgraph detection from temporal graphs," *Proc. VLDB Endowment*, vol. 12, no. 13, pp. 2353–2365, 2019.
[42] E. Galimberti, A. Barrat, F. Bonchi, C. Cattuto, and F. Gullo, "Mining (maximal) span-cores from temporal networks," in *Proc. Int. Conf. Inf. Knowl. Manage.*, 2018, pp. 107–116.
[43] R. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai, "Persistent community search in temporal networks," in *Proc. IEEE Int. Conf. Data Eng.*, 2018, pp. 797–808.
[44] K. Zhu, G. H. L. Fletcher, N. Yakovets, O. Papapetrou, and Y. Wu, "Scalable temporal clique enumeration," in *Proc. Int. Symp. Spatial Temporal Databases*, 2019, pp. 120–129.
[45] A. P. Appel, R. L. de Freitas Cunha, C. C. Aggarwal, and M. M. Terakado, "Temporally evolving community detection and prediction in content-centric networks," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, 2018, pp. 3–18.
[46] R. Aktunc, I. H. Toroslu, and P. Karagoz, "Event detection by change tracking on community structure of temporal networks," in *Proc. IEEE Int. Conf. Adv. Social Netw. Anal. Mining*, 2018, pp. 928–931.
[47] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos, "TimeCrunch: Interpretable dynamic graph summarization," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2015, pp. 1055–1064.
[48] H. Wu et al., "Core decomposition in large temporal graphs," in *Proc. IEEE Int. Conf. Big Data*, 2015, pp. 649–658.
[49] M. Yu, D. Wen, L. Qin, Y. Zhang, W. Zhang, and X. Lin, "On querying historical k-cores," *Proc. VLDB Endow.*, vol. 14, no. 11, pp. 2033–2045, 2021.
[50] Y. Yang, D. Yan, H. Wu, J. Cheng, S. Zhou, and J. C. S. Lui, "Diversified temporal subgraph pattern mining," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 1965–1974.
[51] Y. Yang, J. X. Yu, H. Gao, J. Pei, and J. Li, "Mining most frequently changing component in evolving graphs," *World Wide Web*, vol. 17, no. 3, pp. 351–376, 2014.
[52] S. Huang, A. W. Fu, and R. Liu, "Minimum spanning trees in temporal graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 419–430.
[53] S. Gurukar, S. Ranu, and B. Ravindran, "COMMIT: A scalable approach to mining communication motifs from dynamic networks," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 475–489.



Hongchao Qin received the BS degree in mathematics, the ME and PhD degrees in computer science from Northeastern University, China in 2013, 2015 and 2020, respectively. He is currently a postdoc in Beijing Institute of Technology, China. His current research interests include social network analysis and data-driven graph mining.



Rong-Hua Li received the PhD degree from the Chinese University of Hong Kong in 2013. He is currently a professor with the Beijing Institute of Technology, Beijing, China. His research interests include graph data management and mining, social network analysis, graph computation systems, and graph-based machine learning.

Ye Yuan received the BS, MS, and PhD degrees in computer science from Northeastern University, in 2004, 2007, and 2011, respectively. He is now a professor with the School of Computer Science & Technology, Beijing Institute of Technology, China. His research interests include graph databases, probabilistic databases, and social network analysis.

Yongheng Dai received the PhD degree from The Chinese University of Hong Kong in 2011. Now he works as a R&D engineer in the areas of data modeling, knowledge formalization, and knowledge-driven machine learning with the China Academy of Electronics and Information Technology (CAEIT). From 2011 to 2013, he worked on optical fiber communication and digital signal processing in Huawei. After that, he worked on OFDM-based visible light communication and OCC-based indoor positioning until 2017 in CAEIT. He has published 33 papers in international journals and conferences, and holds 10 pending patents.

Guoren Wang received the BSc, MSc, and PhD degrees from the Department of Computer Science, Northeastern University, China, in 1988, 1991 and 1996, respectively. Currently, he is a professor with the Department of Computer Science, Beijing Institute of Technology, Beijing, China. His research interests include XML data management, query processing and optimization, bioinformatics, high dimensional indexing, parallel database systems, and cloud data management.