Periodic Communities Mining in Temporal Networks: Concepts and Algorithms

Journal:	*Transactions on Knowledge and Data Engineering*
Manuscript ID	TKDESI-2020-04-0326
Manuscript Type:	ICDE 2019
Keywords:	Periodic Community, Temporal Networks, Maximal Clique, Maximum Clique, k-Core

Summary of changes

Periodic Communities Mining in Temporal Networks: Concepts and Algorithms

Hongchao Qin, Rong-Hua Li, Ye Yuan, Guoren Wang, Weihua Yang, and Lu Qin

This journal submission is based on our paper titled "Mining Periodic Cliques in Temporal Networks" published in ICDE 2019. We extended our previous work substantially. We summarize the major differences below.

1) In the previous version, we only study the problem of mining maximal periodic cliques in temporal networks. In this journal submission, except the periodic clique model, we introduce several new periodic community models such as periodic k-core, periodic k-ECC, and periodic k-truss. We also present detailed discussions on the intuitions, cohesiveness, efficiency and relations of those periodic community models (see Section 2).
2) We generalize our previously-proposed periodic clique mining algorithms to a new algorithmic framework for mining periodic communities based on different periodic community models. Our framework includes three steps. The first step is to reduce the temporal graph by two effective graph reduction techniques (Section 3). The second step is to transform the temporal graph into a static graph so that mining the periodic communities in the original temporal graph is equivalent to mining communities in the transformed graph (Section 4). In the third step, we propose a decomposition algorithm to search maximal periodic k-core and a Bron-Kerbosch style algorithm to enumerate all maximal periodic k-cliques (Section 5).
3) We also study the maximum periodic clique search problem in temporal networks, and propose a new branch-and-bound algorithm to compute the maximum periodic clique. To improve the efficiency, we also develop several non-trivial lower-bounding techniques and an early termination technique based on the degeneracy of the graph. All these materials are shown in Section 5.3.
4) We redesign and conduct comprehensive experiments to evaluate the newly-proposed maximal periodic k-core search and maximum periodic clique search algorithm. The new experimental results are shown in Figs.8-9, and Fig. 11. The detailed analyses of these results are presented in Exps.2-4, 8-10 in Section 6.
5) We give all the missing proofs of the theorems and lemmas in our conference version (See Theorems 1-6 and lemmas 1-8). We have also carefully revised the abstract, introduction, related work, and other sections. Many new materials are added compared to the conference version of this paper.
6) For reproducibility purpose, the source code of the algorithms is released at https://github.com/VeryLargeGraph/MPC/.

Periodic Communities Mining in Temporal Networks: Concepts and Algorithms

Hongchao Qin, Rong-Hua Li, Ye Yuan, Guoren Wang, Weihua Yang, and Lu Qin

Abstract—Periodicity is a frequently happening phenomenon for social interactions in temporal networks. Mining periodic communities are essential to understanding periodic group behaviors in temporal networks. Unfortunately, most previous studies for community mining in temporal networks ignore the periodic patterns of communities. In this paper, we study the problem of seeking periodic communities in a temporal network, where each edge is associated with a set of timestamps. We propose novel models, including σ-periodic k-core and σ-periodic k-clique, that represent periodic communities in temporal networks. Specifically, a σ-periodic k-core (or σ-periodic k-clique) is a k-core (or clique with size larger than k) that appears at least σ times periodically in the temporal graph. The problem of searching periodic core is efficient but the resulting communities may be not enough cohesive; the problem of enumerating all periodic cliques is not efficient (NP-hard) but the resulting communities are very cohesive. To compute all of them efficiently, we first develop two effective graph reduction techniques to significantly prune the temporal graph. Then, we transform the temporal graph into a static graph and prove that mining the periodic communities in the temporal graph equals mining communities in the transformed graph. Subsequently, we propose a decomposition algorithm to search maximal σ-periodic k-core, a Bron-Kerbosch style algorithm to enumerate all maximal σ-periodic k-cliques, and a branch-and-bound style algorithm to find the maximum σ-periodic clique. The results of extensive experiments on five real-life datasets demonstrate the efficiency, scalability, and effectiveness of our algorithms.

Index Terms—Periodic Community, Temporal Networks, Maximal Clique, Maximum Clique, k-Core.

1 Introduction

In many real-life networks, such as communication networks, scientific collaboration networks, and social networks, the links are often associated with temporal information. For example, in a face-to-face contact network [1], [2], each edge (u, v, t) denotes a contact between two individuals u and v at time t. In an email communication network, each email contains a sender and a receiver, as well as the time when the email was sent. In a scientific collaboration network (e.g., DBLP), each edge (u, v, t) represents that two authors u and v coauthored a paper at time t. The networks that involve temporal information are typically termed as temporal networks [3], [4], [5].

Periodicity is a frequently happening phenomenon for social interactions in temporal networks. Weekly group meeting, monthly birthday party, and yearly family reunions – these are regular and significant patterns in temporal interaction networks. Mining such periodic group patterns are essential to understanding and predicting group behaviors in a temporal network. In this paper, we investigate a novel data mining problem for temporal networks: periodic community mining, or the detection of all communities that occur at regular time intervals, and show that the proposed technique can be applied to discover the inherent periodicity of communities in a temporal network. Mining the periodic community patterns could be very useful for many practical applications, two of which are listed as follows.

Periodic movement behavior discovery. Consider an application in studying the collective movement behaviors of wild herds of animals [6]. It is well known that the movement behavior of wild herds of animals often exhibits periodic group patterns. In practice, ecologists can tag the animals with tracking sensors to study the collective movement patterns of the animals. In this application, the interactions of the animals (e.g., two animals within a short distance may be considered as an interaction) can be modeled as a temporal network. By mining periodic communities in this temporal network, we are able to identify periodic group movement behaviors of wild animals. Mining such periodic group movement behavior of wild animals can be of ecological interests [6]. For example, if a herd of animals fail to follow the periodic mitigation behavior, it could be a signal of abnormal environment change.

Predicting future activities. Periodic pattern is a predictable pattern, because it repeatedly occurs at regular time intervals. Once we identify a periodic activity, we may predict the same activity will appear within a regular time interval. Based on this observation, we are capable of inferring the future interactions of a group of individuals in a temporal network by mining periodic communities. Taking a temporal scientific collaboration network DBLP as an example, suppose that four researchers A, B, C, and D in DBLP have

- *H.Qin is with the Department of Computer Science, Northeastern University, Shenyang, China. E-mail: qhc.neu@gmail.com*
- *R.Li, Y.Yuan and G.Wang are with the Department of Computer Science, Beijing Institute of Technology, Beijing, China. E-mail: lironghuascut@gmail.com; yuanye@mail.neu.edu.cn; wanggrbit@126.com*
- *W. Yang is with the Taiyuan University of Technology, Taiyuan, China. email: yangweihua@tyut.edu.cn*
- *L.Qin is with the University of Technology Sydney, Sydney, Australia email: lu.qin@uts.edu.au*

Manuscript received 2020

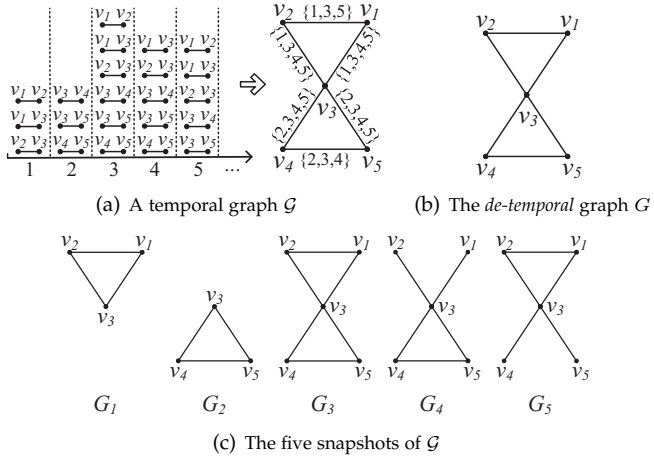
collaborated with each other in 2015, 2016, and 2017 years. Then, we can infer that these four researchers are likely to coauthor papers in 2018 year.

Recently, the problem of mining communities on temporal graphs has attracted much attention due to numerous applications [4], [5], [7]. For example, Wu et al. [7] proposed a temporal k-core model to find cohesive subgraphs in a temporal network. Ma et al. [4] devised a dense subgraph mining algorithm to identify cohesive subgraphs in a temporal network. Li et al. [5] developed an algorithm to detect persistent communities in a temporal graph. All these community mining algorithms do not consider the periodic patterns of communities, thus cannot be applied to identify periodic communities. To the best of our knowledge, we are the first to study the periodic community mining problem, and propose efficient solutions to detect periodic communities in temporal graphs. The main contributions of our work are summarized as follows.

Novel models. We propose several models based on the cohesive subgraph models to characterize periodic communities in a temporal graph. They are motivated by the concepts such as k-core, k-ECC, k-truss and k-clique. The containment relation is that k-clique $\subseteq$ k-truss$\subseteq$ k-ECC $\subseteq$ k-core. In this paper, we study the algorithms of mining maximal σ-periodic k-core, maximal σ-periodic k-clique and maximum σ-periodic clique, since the other models can be computed by the same methods as them.

New algorithms. First, we develop two novel graph reduction methods, called PNCluster and PECluster, based on the concept of k-core [9]. On the basis of the PNCluster and PECluster, we develop two efficient and powerful graph reduction techniques to prune the input temporal graph. We show that both PNCluster and PECluster can be computed in near-linear time and space complexity. Second, we use the variables in the processing of finding PECluster to transform the temporal graph into a static graph. We have proved that mining the periodic communities in the temporal graph equals mining communities in the transformed graph. Third, we propose a decomposition algorithm to search maximal σ-periodic k-core, a Bron-Kerbosch style algorithm to enumerate all maximal σ-periodic k-cliques, and a branch-and-bound style algorithm to find maximum σ-periodic clique. In addition, we present theoretical analyses for all those algorithms. Although the problems of enumerating all maximal σ-periodic k-cliques and finding maximum σ-periodic clique are NP-hard, they are fixed-parameter tractable with respect to a newly-proposed concept called σ-periodic degeneracy $\hat{\delta}$, which is often very small in practice as confirmed in our experiments.

Extensive experiments. We conduct comprehensive experiments on five real-life temporal networks. The results show that our best algorithm is much faster than the baselines on all datasets under most parameter settings. For example, our best algorithm can identify all maximal σ-periodic k-cliques in around 400 seconds on a large temporal graph with more than 1.7 million nodes and 12 million edges. We also examine case studies to evaluate the effectiveness of our model. The results show that our model is indeed able to identify many interesting periodic communities that can not be found by the other models.



(a) A temporal graph $\mathcal{G}$ (b) The *de-temporal* graph G

(c) The five snapshots of $\mathcal{G}$

Fig. 1. Basic concepts of the temporal graph

Organization. Section 2 introduces the models and formulates our problem. The graph reduction techniques are proposed in Section 3. Section 4 introduces the transforming method which can change the temporal graph into a static graph. Section 5 proposes the algorithms for mining the periodic communities. The experiments are shown in Section 6. We review the related work in Section 7, and conclude this work in Section 8.

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. Let $n = |\mathcal{V}|$ and $m = |\mathcal{E}|$ be the number of nodes and 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, because the timestamp is an integer in practice. For a temporal graph $\mathcal{G}$, the *de-temporal graph* of $\mathcal{G}$ denoted by $G = (V, E)$ is a graph that ignores all the timestamps associated with the temporal edges. More formally, for the de-temporal graph G of $\mathcal{G}$, we have $V = \mathcal{V}$ 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 called 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 represent that S is a subgraph of G.

Given a temporal graph $\mathcal{G}$, we can extract a series of snapshots based on the timestamps. Let $\mathcal{T} = \{t|(u, v, t) \in \mathcal{E}\}$ be the set of timestamps. For each $t_i \in \mathcal{T}$, we can obtain a snapshot $G_i = (V_i, E_i)$ where $V_i = \{u|(u, v, t_i) \in \mathcal{E}\}$ and $E_i = \{(u, v)|(u, v, t_i) \in \mathcal{E}\}$. In the rest of this paper, we assume without loss of generality that all the timestamps are sorted in a chronological order, i.e., $t_1 < t_2 < \cdots < t_{|\mathcal{T}|}$. Fig. 1(a) illustrates a temporal graph $\mathcal{G}$ with 5 nodes and 22 temporal edges. Figs.1(b) and (c) illustrates the de-temporal graph of G and all the five snapshots of $\mathcal{G}$ respectively.

TABLE 1
Periodic community models based on the cohesive subgraph models

Models	Intuitions (all are σ-periodic subgraphs C)	Cohesiveness	Efficiency	Complexity (static model)
σ-periodic k-core	each node in C has degree at least k	★	★★★★	$O(\|E\|)$ [10]
σ-periodic k-ECC	C is still connected after removing $k-1$ edges	★★	★★★	$O(hl\|E\|)$ $(l, h << \|V\|)$ [11]
σ-periodic k-truss	each edge in C supports at least $k-2$ triangles	★★	★★★	$O(\|E\|^{1.5})$ [12]
σ-periodic k-clique	a clique with size no less than k	★★★	★	$O(\|V\|^2 3^{\|V\|/3})$ (NP-hard) [8]

Definition 1 (time support set). Given a temporal graph $\mathcal{G}$, the time support set of a subgraph S is defined as $\mathbb{TS}(S) \triangleq \{t_i | S \subseteq G_i\}$, where G_i is the i-th snapshot of $\mathcal{G}$.

Definition 2 (σ-periodic time support set). Given a temporal graph $\mathcal{G}$ and a parameter σ, a σ-periodic time support set of a subgraph S, denoted by $\mathbb{PT}^{\sigma}(S)$, is a subset of $\mathbb{TS}(S)$ such that (1) $\mathbb{PT}^{\sigma}(S) = \{t_{j_1}, \cdots, t_{j_\sigma}\}$, and (2) $t_{j_{i+1}} - t_{j_i} = p$ for all $i = 1, \cdots, \sigma - 1$ with any constant p.

By Definition 2, we can see that the timestamps of a σ-periodic time support set forms an arithmetic sequence and the cardinality of a σ-periodic time support set is exactly equal to σ. Clearly, there may exist many σ-periodic time support sets for a subgraph S. Based on Definition 2, we define the σ-periodic subgraph below.

Definition 3 (σ-periodic subgraph). Given a temporal graph $\mathcal{G}$, its de-temporal graph 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 $\mathbb{PT}^{\sigma}(S)$ which is not empty.

By Definition 3, any σ-periodic subgraph $S \subseteq G$ has at least one σ-periodic time support set $\mathbb{PT}^{\sigma}(S)$. A subgraph S is a maximal σ-periodic subgraph if there is no other σ-periodic subgraph S' that satisfies $S \subset S'$. Intuitively, a periodic community should be a periodic densely-connected subgraph. We propose several novel models to define the periodic communities as follows.

Definition 4 (σ-periodic k-core). A σ-periodic k-core C is a subgraph of the de-temporal graph G such that (1) every node inside C has degree at least k, and (2) C is a σ-periodic subgraph.

Definition 5 (σ-periodic k-clique). A σ-periodic k-clique C is a subgraph of the de-temporal graph G such that (1) C is a clique in G with $|C| > k$, and (2) C is a σ-periodic subgraph.

Fig. 2 shows the comparison of the widely used cohesiveness models. Those models have the following properties: (1) a k-ECC must be a k-core since considering any node u, u will not be disconnected by removing $k-1$ edges so u has degree no less than k; (2) a k-truss must be a k-ECC since considering any edge (u, v), it will be contained in $k-2$ triangles so node u and v can not be disconnected by removing $k-1$ edges; (3) a k-clique must be a k-truss since nodes in clique are connected with each other. Based on the widely-used k-truss and k-ECC model, we can modify the first condition in Definition 4 and 5 to define the σ-periodic k-truss and σ-periodic k-ECC.

Table 1 shows the intuitions, cohesiveness, efficiency and complexity (static model) of those periodic community models. As σ-periodic k-core is the worst cohesive and best efficient model, and σ-periodic k-clique is the best cohesive and worst efficient model, we study the problem of mining σ-periodic core and σ-periodic clique in this paper due to the limit of space.

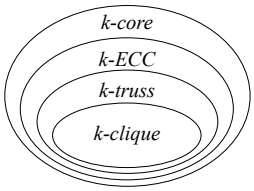


Fig. 2. Comparison of the cohesiveness models

Note that for a typical temporal graph, many σ-periodic cliques are small and may not be interesting to the users. Therefore, it will be more useful to find large σ-periodic cliques for practical applications. As a result, we focus mainly on mining the σ-periodic cliques with size larger than k as defined in Definition 4 and 5. Intuitively, a σ-periodic k-core (or σ-periodic k-clique) C is maximal if there is no other σ-periodic k-core (or σ-periodic k-clique) C' meeting $C \subset C'$.

Below, we use an example to illustrate the above definitions and summarizes the problems below.

Example 1. Consider a temporal graph in Fig. 1(a). Suppose that σ = 3, k= 2. For the subgraph $S = \{(v_1, v_3), (v_2, v_3)\}$, the time support set of S is $\{1, 3, 4, 5\}$. Clearly, by Definition 2, the set $\{1, 3, 5\}$ is a σ-periodic time support set of S. Therefore, the subgraph S is a σ-periodic subgraph by Definition 3. Note that S is not a maximal σ-periodic subgraph because there is a σ-periodic subgraph $C = \{(v_1, v_3), (v_2, v_3), (v_1, v_2)\}$ that contains S. By Definition 4 and 5, we can see that C is both a σ-periodic k-core and σ-periodic k-clique with $\mathbb{PT}^{\sigma}(C) = \{1, 3, 5\}$. Moreover, $C' = \{(v_3, v_4), (v_3, v_5), (v_4, v_5)\}$ is another σ-periodic k-core and σ-periodic k-clique with $\mathbb{PT}^{\sigma}(C') = \{2, 3, 4\}$.

Problem 1. Given a temporal graph $\mathcal{G}$ and parameters σ and k, the goal is to find all the maximal σ-periodic k-core (MPCore) in $\mathcal{G}$.

Problem 2. Given a temporal graph $\mathcal{G}$ and parameters σ and k, the goal is to enumerate all the maximal σ-periodic k-clique (MPClique) in $\mathcal{G}$.

In problem 2, it is hard to set the parameter k to control the lower size of the clique in some applications, so we study the problem below to find the largest k.

Problem 3. Given a temporal graph $\mathcal{G}$ and parameters σ and k, the goal is to find the maximum σ-periodic clique of the largest size (MAXPClique) in $\mathcal{G}$.

Note that, if we have enumerated all the maximal σ-periodic k-cliques in Problem 2, then Problem 3 will be solved. However, there are several pruning rules which can speed up the process of finding maximum σ-periodic clique.

So the key issues in this paper are to mine MPCore and MPClique in $\mathcal{G}$.

NP-hardness. As shown in Table 1, the k-core can be found by linear time in terms of $|E|$, but the maximal clique enumeration problem is NP-hard. Below, we prove that the maximal σ-periodic k-clique enumeration problem is also NP-hard.

We can show that the traditional maximal clique enumeration problem is a special case of the maximal σ-periodic k-clique enumeration problem. Consider a temporal graph $\mathcal{G}$ that contains a set of snapshots $G = G_i = G_2 =, \cdots, = G_{|\mathcal{T}|}$. Clearly, in this temporal graph $\mathcal{G}$, every subgraph of G is periodic. As a result, the problem of enumerating all maximal σ-periodic k-cliques is equivalent to the problem of enumerating all maximal cliques (with size larger than k) in the *de-temporal* graph G. So our problem is NP-hard.

Although there is a close connection between our problem and the maximal clique enumeration problem, the existing maximal clique enumeration algorithms cannot be directly applied to solve our problem. The reason is that the traditional maximal clique enumeration algorithms, such as the Bron-Kerbosch algorithm [13] can only identify maximal cliques in a snapshot G_i for the timestamp t_i. It is not clear to apply this algorithm to derive maximal periodic cliques.

Challenges. To solve our problems, a possible solution is first to enumerate all maximal cores/cliques in the *de-temporal* graph, and then checks which of them is periodic. However, this method is quite complicated and even intractable, because a core/clique in a snapshot may contain a maximal periodic core/clique with less nodes in a periodic time support set. Therefore, we need to check each subgraph of a maximal core/clique in each snapshot, which is very costly.

Another potential approach is first to enumerate all periodic subgraphs, and then applies traditional algorithms to identify all MPCores/MPCliques in each periodic subgraph. Clearly, this approach may involve numerous redundant computations for subgraphs with the same nodes, because the number of periodic subgraphs may be very large and the same cores or cliques could be repeatedly enumerated in many different periodic subgraphs. Therefore, the challenge of our problem is how to efficiently enumerate all periodic communities with less redundant computations. In the following sections, we will develop several novel graph reduction techniques and an efficient enumeration algorithm to identify them.

3 Reduction by Periodic Nodes and Edges

In this section, we propose several powerful techniques to prune the unpromising nodes which are definitely not contained in any periodic communities. Our key idea for graph reduction is based on the concept of k-core [10]. Before proceeding further, we first give the definition of k-core (abbreviated as KCore) as follows.

Definition 6 (*KCore*). Given a de-temporal graph G of $\mathcal{G}$ and a parameter k, a KCore is a maximal subgraph of G in which every node has degree at least k, i.e., $d_u(G) \geq k$ for $u \in G$.

It is easy to check that if a node is contained in a maximal σ-periodic k-core, this node will have at least k neighbors in the de-temporal graph G of $\mathcal{G}$. Hence, if a node is not included in the KCore of G, it must be not contained in any maximal σ-periodic k-core. As a consequence, we can first prune all nodes that are not contained in the KCore of G. This prune rule is simple, but it may be not very effective, because it does not consider the periodic property for pruning. Below, we develop novel concepts called PNCluster and PECluster which can capture the periodic property for pruning.

3.1 The PNCluster pruning rule

By Definitions 4 and 5, we can easily derive that every node u in periodic communities satisfies a *periodic degree property*: there must exist a σ-periodic subgraph in which u has degree no less than k. Therefore, if a node is not contained in any σ-periodic subgraph, it can be safely pruned. Since the deletion of an unpromising node may trigger its neighbors that violate the periodic degree property, we can iteratively prune the graph until all nodes meet the periodic degree property. Below, we give a definition, called (σ, k)-periodic node, to describe a node that satisfies the periodic degree property.

Definition 7 (*(σ, k)*-periodic node). Given a temporal graph $\mathcal{G}$, a subgraph $S \subseteq G$, and parameters σ and k, a node v is called a (σ, k)-periodic node in S if and only if there exists a σ-periodic subgraph of S in which v has degree at least k.

Recall that by Definition 3, a σ-periodic subgraph may have many σ-periodic time support sets. Therefore, there may also exist many σ-periodic time support sets for a (σ, k)-periodic node v in which v has degree no less than k. Below, we give a definition to describe all σ-periodic time support sets for a (σ, k)-periodic node.

Based on Definition 7, we define the (σ, k)-periodic time support set for a (σ, k)-periodic node as follows.

Definition 8 ((σ, k)-periodic time support set). Given a temporal graph $\mathcal{G}$, a subgraph $S \subseteq \mathcal{V}$ and a (σ, k)-periodic node v, the (σ, k)-periodic time support set of v in S is $\mathbb{PT}_k^\sigma(S, v) \triangleq [t_{j_1}, \cdots, t_{j_\sigma}]$ that satisfies (1) $t_{j_{i+1}} - t_{j_i} = p$ for each $i = 1, \cdots, \sigma - 1$ with a constant p, and (2) $d_v(S \cap G_{t_i}) \geq k$ for each $i = 1, \cdots, \sigma - 1$.

By Definition 8, for any (σ, k)-periodic node v, there is a σ-periodic subgraph S with $\mathbb{PT}^\sigma(S) = \mathbb{PT}_k^\sigma(S, v)$ in which $d_v(S) \geq k$. Since a σ-periodic subgraph may have many σ-periodic time support sets, there also exist many (σ, k)-periodic time support sets for a (σ, k)-periodic node v. For convenience, when there is no confusion of S, σ and k, we let PT_v be the set of all those (σ, k)-periodic time support sets in subgraph S for the node v. Clearly, a node v is a (σ, k)-periodic node if and only if $\mathsf{PT}_v \neq \emptyset$ in a subgraph S. Based on the above definitions, we present a new periodic cohesive subgraph model, called (σ, k)-periodic node cluster (abbreviated as PNCluster), which will be applied to prune unpromising nodes in the periodic communities. The PNCluster is defined as follows.

Definition 9 ((σ, k)-periodic node cluster). Given a temporal graph $\mathcal{G}$, two integer parameters σ and k, a subset of nodes S in $\mathcal{G}$ is called a (σ, k)-periodic node cluster if it meets the following constraints.

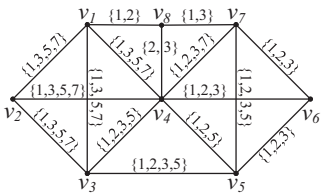


Fig. 3. Running example

(1) ***Periodic degree constraint:*** each node $u \in S$ is a (σ, k)-periodic node of the temporal subgraph induced by S;
(2) ***Maximal constraint:*** there does not exist a subset of nodes S' in $\mathcal{G}$ that satisfies (1) and $S \subset S'$.

Lemma 1. Given a temporal graph $\mathcal{G}$, parameters σ and k, the PNCluster is unique in $\mathcal{G}$ if it exists.

Proof: We can prove this lemma by a contradiction. Suppose that there exist two different PNCluster in $\mathcal{G}$, denoted by S_1 and S_2 respectively ($S_1 \neq S_2$). Let us consider the node set $S = S_1 \cup S_2$. Since every node in S_1 (S_2) is a (σ, k)-periodic node, each node in S is also a (σ, k)-periodic node by Definition 7. As a result, every node in S meets the periodic degree property in Definition 9. Since $S_1 \neq S_2$, we have $S_1 \subset S$ and $S_2 \subset S$ which contradicts to the fact that S_1 (S_2) satisfies the maximal property. □

The following example illustrates the above definitions.

Example 2. Consider a temporal graph $\mathcal{G}$ shown in Fig. 3. Note that in Fig. 3, each temporal edge is associated with a set of integers denoting the set of timestamps of that edge. Clearly, the de-temporal graph G of $\mathcal{G}$ is a 3-core, as every node in G has at least 3 neighbors. For node v_4, we can see that it has degree no less than 3 in timestamps $\{1, 2, 3, 5, 7\}$. Suppose that $\sigma = 3, k = 3$. Then, we can derive that v_4 is a (σ, k)-periodic node. This is because there exists a σ-periodic subgraph $S = \{(v_4, v_3), (v_4, v_6), (v_4, v_7)\}$ in which $d_{v_4}(S) \geq 3$, and the corresponding (σ, k)-periodic time support set for v_4 is $[1, 2, 3]$ (i.e., $\mathbb{PT}^{\sigma}_{k}(S, v_4) = [1, 2, 3]$). It is easy to check that there are three (σ, k)-periodic time support sets for v_4 in G, which are $[1, 2, 3]$, $[1, 3, 5]$ and $[3, 5, 7]$. Thus, we have $\mathsf{PT}_{v_4} = \{[1, 2, 3], [1, 3, 5], [3, 5, 7]\}$ in G. Also, we can find that v_8 is not a (σ, k)-periodic node, because no σ-periodic subgraph contains v_8. By Definition 9, we can obtain that $\{v_1, \cdots, v_7\}$ is a PNCluster. □

Based on Lemma 1, the PNCluster is unique in $\mathcal{G}$ so it can be computed by a decomposition framework. Below, we develop two efficient algorithms to efficiently calculate the PNCluster.

The basic PNCluster **algorithm.** Similar to the traditional k-core algorithm [9], a basic solution to compute the PNCluster is to peel the nodes from $\mathcal{G}$ that violate the periodic degree property. Since the deletion of a node u may result in u's neighbors no longer satisfying the periodic degree property, we need to iteratively process u's neighbors. Such an iterative peeling procedure terminates until no node can be deleted. When the algorithm completes, the remaining nodes form the PNCluster. The detailed description of our algorithm is shown in Algorithm 1.

Algorithm 1 first computes the KCore $G_c = (V_c, E_c)$ in the de-temporal graph (lines 1-2), because the PNCluster is clearly contained in the KCore. Then, for each node u in V_c, the algorithm invokes Algorithm 2 to checks whether u is a (σ, k)-periodic node or not (lines 4-6). If a node u is not a (σ, k)-periodic node, it will be pushed into a queue $\mathcal{Q}$ (lines 7-8). Subsequently, the algorithm iteratively processes the nodes in $\mathcal{Q}$. In each iteration, the algorithm pops a node v from $\mathcal{Q}$ and uses a set D to maintain all the deleted nodes (line 10). For each neighbor node w of v, the algorithm updates $d_w(G_c)$ (lines 12). If the revised $d_w(G_c)$ is smaller than k, w is clearly not a (σ, k)-periodic node. As a consequence, the algorithm pushes it into $\mathcal{Q}$ which will be deleted in the next iterations (line 13). Otherwise, the algorithm invokes Algorithm 2 to determine whether w is a (σ, k)-periodic node (lines 14-15). If w is not a (σ, k)-periodic node, the algorithm sets $d_w(G_c)$ to 0, and pushes it into $\mathcal{Q}$. The algorithm terminates when $\mathcal{Q}$ is empty. At this moment, the remaining nodes $V_c \setminus D$ is the PNCluster of G. Below, we describe the implementation details of Algorithm 2.

Recall that we need to compute the set of (σ, k)-periodic time support set in $G_{V_c \setminus D}$ for a node v, i.e., PT_v, to check

Algorithm 1: PNCluster $(\mathcal{G}, \sigma, k)$

Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parameters σ and k
Output: The PNCluster V_w.
1 Let $G = (V, E)$ be the de-temporal graph of $\mathcal{G}$;
2 Let $G_c = (V_c, E_c)$ be the KCore of G;
3 $\mathcal{Q} \leftarrow \emptyset$; $D \leftarrow \emptyset$;
4 **for** $u \in V_c$ **do**
5 $d_u(G_c) \leftarrow$ compute the degree of u in G_c;
6 $(flag, \mathsf{PT}_u) \leftarrow$ComputePeriod $(\mathcal{G}, \sigma, k, u, V_c)$;
7 **if** $flag = 0$ **then**
8 $d_u(G_c) \leftarrow 0$; $\mathcal{Q}.push(u)$;
9 **while** $\mathcal{Q} \neq \emptyset$ **do**
10 $v \leftarrow \mathcal{Q}.pop()$; $D \leftarrow D \cup \{v\}$;
11 **for** $w \in N_v(G_c)$, *s.t.* $d_u(G_c) \geq k$ **do**
12 $d_w(G_c) \leftarrow d_w(G_c) - 1$;
13 **if** $d_w(G_c) < k$ **then** $\mathcal{Q}.push(w)$;
14 **else**
15 $(flag, \mathsf{PT}_w) \leftarrow$ComputePeriod $(\mathcal{G}, \sigma, k, w, V_c \setminus D)$;
16 **if** $flag = 0$ **then**
17 $d_w(G_c) \leftarrow 0$; $\mathcal{Q}.push(w)$;
18 **return** $V_w \leftarrow V_c \setminus D$;

Algorithm 2: ComputePeriod $(\mathcal{G}, \sigma, k, u, F)$

Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parameters σ, k, node u, and node set F
Output: A boolean variable $flag$ and PT_u
1 $PQ \leftarrow \emptyset$; $StartS \leftarrow \emptyset$; $\mathsf{PT}_u \leftarrow \emptyset$; $flag \leftarrow 0$;
2 **for** $t \leftarrow t_1 : t_{|\mathcal{T}|}$ **do**
3 Let G_t be the snapshot of $\mathcal{G}$ at timestamp t;
4 $d_u \leftarrow |N_u(G_t) \cap F|$;
5 **if** $d_u \geq k$ **then**
6 **for** *each* $TS \leftarrow [s, i, l, ArrD] \in PQ$ **do**
7 **if** $(t - TS.s)\%TS.i = 0$ **then**
8 **if** $(t - TS.s)/TS.i \neq TS.l$ **then**
9 $PQ.pop(TS)$; **continue**;
10 $TS.l \leftarrow TS.l + 1$; $TS.ArrD \leftarrow TS.ArrD \cup \{d_u\}$;
11 **if** $TS.l = \sigma$ **then**
12 $\mathsf{PT}_u \leftarrow \mathsf{PT}_u \cup \{TS\}$; $flag \leftarrow 1$;$PQ.pop(TS)$;
13 /* For PNCluster, the algorithm can early terminate. */
14 **if** $t - TS.s > (\sigma - 1)TS.i$ **then** $PQ.pop(TS)$;
15 **for** $start \leftarrow [s, d] \in StartS$ **do**
16 $PQ.push([start.s, t - start.s, 2, \{start.d, d_u\}])$;
17 $StartS \leftarrow StartS \cup \{[t, d_u]\}$;
18 **return** $(flag, \mathsf{PT}_u)$;

whether v is a (σ, k)-periodic node or not. The node v is a (σ, k)-periodic node if and only if PT_v is nonempty. By Definition 8, a (σ, k)-periodic time support set can be represented as an arithmetic sequence of the timestamps. In Algorithm 2, we record PT_v as a set where each element TS (Time Support) in PT_v is a four-tuple $[s, i, l, ArrD]$ representing an arithmetic sequence. In the four-tuple $[s, i, l, ArrD]$, s denotes the starting timestamp of the arithmetic sequence, i is the common difference, l represents the number of terms of the arithmetic sequence, and $ArrD$ (Array of Degree) is an array that stores the degree of u at each timestamp of the arithmetic sequence.

Based on this data structure, the algorithm makes use of a queue PQ to maintain all the candidates of the arithmetic sequences. The algorithm also uses a set $StartS$ to store all the starting timestamps of the arithmetic sequences. Each element in $StartS$ is a two-tuple $[s, d]$, where s denotes the starting timestamp and d denotes the degree of u at s (lines 15-17). Initially, both PQ and $StartS$ are set to empty sets (line 1). Then, the algorithm enumerates all the timestamps from t_1 to $t_{|\mathcal{T}|}$ (line 2). For each timestamp, the algorithm calculates the number of neighbors of u (denoted by d_u) that are both in G_t (the snapshot at the timestamp t) and the node set F (lines 3-4), i.e., $|N_u(G_t) \cap F|$. If $d_u \geq k$, the algorithm explores all the candidate arithmetic sequences in PQ (lines 5-6). For each candidate $TS \in PQ$, if $(t - TS.s)\%TS.i = 0$, we may extend the arithmetic sequence TS by t (line 7). If $(t - TS.s)/TS.i \neq TS.l$, we know that t cannot extend the current arithmetic sequence TS. Since the remaining timestamps are no less than t, they also cannot extend TS. Therefore, we can safely delete the candidate TS (lines 8-9). Otherwise, the algorithm can augment the arithmetic sequence TS by adding t into TS. In this case, we increase $TS.l$ by 1, and add d_u into the array $TS.ArrD$ (line 8). If the augmented arithmetic sequence TS has σ terms, TS represents a valid (σ, k)-periodic time support set for u (line 11). As a result, the algorithm adds TS into PT_u and set $flag$ to 1, denoting that u is a (σ, k)-periodic node (line 12). At this moment, the algorithm can early terminate. Note that Algorithm 2 can also be applied to compute the complete set of (σ, k)-periodic time support sets for u. Clearly, if $t - TS.s > (\sigma - 1)TS.i$, t cannot grow the current arithmetic sequence TS, and TS is no longer to be a valid (σ, k)-periodic time support set. Therefore, the algorithm deletes TS from PQ (line 14). For each starting timestamp $start.s$, the algorithm makes use of the current timestamp t and $start.s$ to generate an initial arithmetic sequence (lines 15-16). The algorithm also applies the current timestamp t to generate a new starting timestamp which will be used for the next iterations (line 17). Since Algorithm 2 explores all the possible arithmetic sequences, it is able to correctly compute PT_u. The following example illustrates how Algorithm 2 works.

Example 3. Reconsider the temporal graph in Fig. 3. Suppose that $\sigma = 3, k = 3$. It is easy to derive that v_4 has degree no less than 3 at the timestamps $\{1, 2, 3, 5, 7\}$. Fig. 4 illustrates the candidate arithmetic sequences when the algorithm processes a timestamp in $\{1, 2, 3, 5, 7\}$. The first row in Fig. 4 shows the starting timestamp of the candidate arithmetic sequences. When $t = 1$, the starting timestamp is $\{1\}$, and the set $StartS = \{[1, 6]\}$ (since $d_{v_4} = 6$ at timestamp 1). When $t = 2$, there is a candidate arithmetic sequence $\{1, 2\}$, and the queue PQ has an element $[1, 1, 2, \{6, 5\}]$. Similarly, when $t = 3$ there are three candidates which are $\{1, 2, 3\}$, $\{1, 3\}$, and $\{2, 3\}$. Clearly, $\{1, 2, 3\}$ is a valid (σ, k)-periodic time support set for v_4. When $t = 5$, the timestamp 5 cannot extend $\{2, 3\}$, thus $\{2, 3\}$ is deleted. It is easy to check that the timestamp 5 can extend $\{1, 3\}$, $\{1\}$, $\{2\}$, and $\{3\}$. As a result, we can obtain four candidates $\{1, 3, 5\}$, $\{1, 5\}$, $\{2, 5\}$, and $\{3, 5\}$. Likewise, when $t = 7$, we have seven candidates which are $\{3, 5, 7\}$, $\{1, 5\}$, $\{2, 5\}$, $\{1, 7\}$, $\{2, 7\}$, $\{3, 7\}$ and $\{5, 7\}$. Note that our algorithm cannot delete the candidate $\{1, 5\}$ when $t = 7$, because $\{1, 5\}$ could be extended by $t > 7$ (similar for $\{2, 5\}$). Clearly, we can obtain three (σ, k)-periodic time support sets for v_4 which are $[1, 2, 3]$, $[1, 3, 5]$, and $[3, 5, 7]$. □

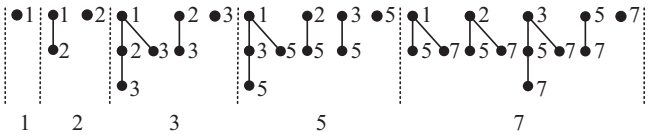


Fig. 4. Illustration of using Algorithm 2 to compute $\mathsf{PT}(v_4)$

Analysis of Algorithm 1. Below, we analyze the correctness and complexity of Algorithm 1.

Theorem 1. Algorithm 1 correctly computes the PNCluster.

Proof: Let $S = V_c \setminus D$. Clearly, by Algorithm 1, each node in S is a (σ, k)-periodic node of the temporal subgraph induced by S. To prove that S is a PNCluster, we need to show the maximal property of S. Suppose to the contrary that there is a set S' such that (1) every node in S' is a (σ, k)-periodic node of the temporal subgraph induced by S', and (2) $S \subset S'$. Since $S \subset S'$, there exists a (σ, k)-periodic node $u \in S' \setminus S$ in the temporal subgraph induced by S'. Note that by our assumption, every node in S' has degree no less than k in a σ-periodic subgraph of the temporal graph induced by S'. Thus, Algorithm 1 cannot delete the node u. Therefore, $u \in V_c \setminus D$ which is a contradiction. □

The complexity of Algorithm 1 is shown as follows.

Lemma 2. For a temporal graph $\mathcal{G}$ with $|\mathcal{T}|$ timestamps, there are at most $O(|\mathcal{T}|^2\sigma^{-1})$ (σ, k)-periodic time support sets for each node in $\mathcal{G}$.

Proof: Recall that each (σ, k)-periodic time support set for a node 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 (σ, k)-periodic time support sets is bounded by $O(|T|^2\sigma^{-1})$. □

Based on Lemma 2, we have the following results.

Corollary 1. The time and space complexity of Algorithm 2 for computing PT_u is $O(|\mathcal{T}|d_u(G) + |\mathcal{T}|^2\sigma^{-1})$ and $O(|\mathcal{T}|^2)$ respectively.

Proof: First, Algorithm 2 needs to compute the degree of u at each timestamp which consumes $|\mathcal{T}|d_u(G)$ time in the worst case. Since there are at most $O(|\mathcal{T}|^2\sigma^{-1})$ (σ, k)-periodic time support sets for u by Lemma 2, the total number of TS can be bounded by $O(|\mathcal{T}|^2\sigma^{-1})$. Therefore, the total time complexity of Algorithm 2 is $O(|\mathcal{T}|d_u(G) + |\mathcal{T}|^2\sigma^{-1})$. For the space complexity, each TS uses $O(\sigma)$ space, thus the total space complexity is $O(|\mathcal{T}|^2)$. □

Theorem 2. The time and space complexity of Algorithm 1 is $O(m|\mathcal{T}|^2\sigma^{-1})$ and $O(m + n + |\mathcal{T}|^2)$ respectively.

Proof: We first analyze the time complexity of Algorithm 1. First, the algorithm takes $O(m + n)$ time to compute the k-core $G_c = (V_c, E_c)$ (line 2). Then, for each node u in V_c, the algorithm invokes Algorithm 2 to compute PT_u which takes $O(|\mathcal{T}|^2\sigma^{-1})$ time. Therefore, in lines 4-8, the algorithm consumes $O(n|\mathcal{T}|^2\sigma^{-1})$ time. In lines 9-17, for each node v, the algorithm explores all neighbors of v at most once. For each neighbor w of v, the algorithm needs to invoke Algorithm 2 to compute PT_w which consumes $O(|\mathcal{T}|^2\sigma^{-1})$ time. Therefore, the total time complexity in lines 9-17 is $O(m|\mathcal{T}|^2\sigma^{-1})$. Putting it all together, the time complexity of Algorithm 1 is $O(m|\mathcal{T}|^2\sigma^{-1})$. For the space complexity, the algorithm needs to maintain the graph, the queue $\mathcal{Q}$, and PT_u for a node $u \in V_c$ which consumes $O(m + n + |\mathcal{T}|^2)$ space in total. □

Note that $|\mathcal{T}|$ (the number of snapshots) is typically not very large in practical temporal graphs. For example, in DBLP temporal network, there are at most 60 snapshots if we extract a snapshot by year (each snapshot represents a co-authorship network in one year). Hence, the worst-case time complexity of our algorithm is near linear w.r.t. the size of the temporal graph. Moreover, the practical performance of Algorithm 1 should be much better than the worst-case time complexity. This is because Algorithm 1 is integrated with a degree pruning rule (see lines 12-13 in Algorithm 1), which significantly decreases the number of calls of the ComputePeriod procedure. In addition, the ComputePeriod procedure can early terminate once the algorithm find a valid (σ, k)-periodic time support set, which can further reduce the time cost of Algorithm 1.

An improved PNCluster+ algorithm. Although Algorithm 1 is efficient in practice, it still has two limitations. First, Algorithm 1 needs to invoke Algorithm 2 to compute PT_u for every node $u \in V_c$ (line 6), which is very costly for high-degree nodes. Second, when deleting a node u, Algorithm 1 has to call Algorithm 2 to re-compute PT_w for each neighbor node w of u (see line 15 in Algorithm 1), which clearly results in significant amounts of redundant computations.

To overcome these limitations, we propose an improved algorithm called PNCluster+. The striking features of PNCluster+ are twofold. On the one hand, PNCluster+ does not compute PT_u for every node u in advance. Instead, it calculates PT_u for the node u *on-demand*. PNCluster+ processes the nodes based on an increasing order by their degrees. Specifically, the algorithm first explores the low-degree nodes and applies the degree pruning rule to delete nodes. This is because the low-degree nodes are more likely to be deleted by the degree pruning rule. Moreover, compared to the high-degree nodes, the time costs for computing PT_u for low-degree nodes are much cheaper. If a node u cannot be removed by the degree pruning rule, the PNCluster+ algorithm invokes Algorithm 2 to compute PT_u on-demand. Note that based on this on-demand computing paradigm, we can substantially reduce the computational costs for the high-degree nodes. The reason is as follows. When processing a high-degree node u, many low-degree neighbors of u may have already been pruned which will significantly decrease the degree of u, thus reducing the cost for computing PT_u. On the other hand, when deleting a node u, PNCluster+ does not re-compute PT_w for a neighbor node w of u. Instead, PNCluster+ dynamically updates the computed PT_w for each node w, thus substantially avoiding redundant computations. The detailed description of PNCluster+ is shown in Algorithm 3.

Algorithm 3: PNCluster+ $(\mathcal{G}, \sigma, k)$

Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parameters σ, and k
Output: The PNCluster V_w
1 Let $G = (V, E)$ be the de-temporal graph of $\mathcal{G}$;
2 Let $G_c = (V_c, E_c)$ be the KCore of G;
3 $\mathcal{Q} \leftarrow \emptyset$; $D \leftarrow \emptyset$;
4 Let $d_u(G_c)$ be the degree of u in G_c;
5 **for** $u \in V_c$ *in an increasing order by* $d_u(G_c)$ **do**
6 **if** $u \in D$ **then continue**;
7 $\mathsf{PT}_u \leftarrow$ComputePeriod $(u, \mathcal{G}, \sigma, k, V_c \setminus D)$;
8 **if** $\mathsf{PT}_u = \emptyset$ **then** $\mathcal{Q}.push(u)$;
9 $\mathsf{IPT}_u \leftarrow$InvertIndex (PT_u);
10 **while** $\mathcal{Q} \neq \emptyset$ **do**
11 $v \leftarrow \mathcal{Q}.pop()$; $D \leftarrow D \cup \{v\}$;
12 **for** $w \in N_v(G_c)$ **do**
13 **if** $d_w(G_c) \geq k$ **then**
14 $d_w(G_c) \leftarrow d_w(G_c) - 1$;
15 **if** $d_w(G_c) < k$ **then** $\mathcal{Q}.push(w)$; **continue**;
16 **if** PT_w *has already been computed* **then**
17 UpdatePeriod $(\mathsf{PT}_w, \mathsf{IPT}_w, v, k)$;
18 **if** $\mathsf{PT}_w = \emptyset$ **then** $\mathcal{Q}.push(w)$;
19 **return** $V_w \leftarrow (V_c \setminus D)$;
20 **Procedure** InvertIndex (PT_u)
21 $\mathsf{IPT}_u \leftarrow \emptyset$; $L \leftarrow \emptyset$; $h \leftarrow 1$;
22 Let $\mathsf{PT}_u(j) \leftarrow [s, i, \sigma, ArrD]$ be the j-th element in PT_u;
23 **for** $j \leftarrow 1 : |\mathsf{PT}_u|$ **do**
24 **for** $t \leftarrow 0 : (\sigma - 1)$ **do**
25 $L(h) \leftarrow [\mathsf{PT}_u(j).s + t \times i,\ j]$; $h \leftarrow h + 1$;
26 **for** $h \leftarrow 1 : |L|$ **do**
27 $[t,\ j] \leftarrow L(h)$; $\mathsf{IPT}_u(t).push(j)$;
28 **return** IPT_u;
29 **Procedure** UpdatePeriod $(\mathsf{PT}_w, \mathsf{IPT}_w, v, k)$
30 **for** *each temporal edge* $(w, v, t) \in \mathcal{E}$ **do**
31 $\mathsf{PTS}(t) \leftarrow \mathsf{IPT}_w(t)$;
32 **while** $\mathsf{PTS}(t) \neq \emptyset$ **do**
33 $j \leftarrow \mathsf{PTS}(t).pop()$;
34 $\mathsf{PT}_w(j).ArrD[t] \leftarrow \mathsf{PT}_w(j).ArrD[t] - 1$;
35 **if** $\mathsf{PT}_w(j).ArrD[t] < k$ **then**
36 $\mathsf{PT}_w \leftarrow \mathsf{PT}_w \setminus \{\mathsf{PT}_w(j)\}$;

Algorithm 3 first computes the KCore $G_c = (V_c, E_c)$ in the de-temporal graph (line 2), and then explores the nodes in V_c based on an increasing order by the degrees in G_c (line 5). When processing a node u, the algorithm first checks whether u has been deleted or not (line 6). If u has not been removed, PNCluster+ invokes Algorithm 2 to compute PT_u (line 7). If PT_u is an empty set, u is not a (σ, k)-periodic node. Thus, the algorithm pushes it into the queue $\mathcal{Q}$ (line 8). Subsequently, the algorithm iteratively deletes the nodes in $\mathcal{Q}$ (lines 10-18). When removing a node v, PNCluster+ explores all v's neighbors (line 12). For a neighbor node

w, PNCluster+ first updates the degree of w (line 14), i.e., $d_w(G_c)$. If the updated degree is less than k, u is not a (σ, k)-periodic node (line 15). In this case, the algorithm pushes it into $\mathcal{Q}$ and continues to process the next node in $\mathcal{Q}$ (the degree pruning rule). Otherwise, if PT_w has already been computed, the algorithm invokes UpdatePeriod to update PT_w (line 17). If the updated PT_w becomes empty, w is not a (σ, k)-periodic node and the algorithm pushes w into $\mathcal{Q}$ (line 18). Note that if PT_w has not been computed yet, the algorithm does not need to update PT_w. In this case, PT_w will be calculated in the next iterations (see line 7).

To efficiently implement the UpdatePeriod procedure, we develop an inverted index structure called IPT_u to organize all (σ, k)-periodic time support sets maintained in PT_u. Specifically, for the j-th arithmetic sequence (corresponding to a (σ, k)-periodic time support set) $\{t_{j_i}, t_{j_i+p}, \cdots, t_{j_i+(\sigma-1)\times p}\}$ in PT_u, we insert an element j into $\mathsf{IPT}_u(t_{j_i+h\times p})$ for each $0 \leq h \leq \sigma - 1$, i.e., $\mathsf{IPT}_u(t_{j_i+h\times p}).push(j)$. Based on PT_u, we can easily construct the inverted index IPT_u by invoking the InvertIndex procedure (lines 20-28). By our construction, $\mathsf{IPT}_u(t)$ keeps all arithmetic sequences that contain the timestamp t. Therefore, once we have an invert index IPT_u, we can quickly retrieve the arithmetic sequences containing t.

The UpdatePeriod procedure explores all the temporal edges (w, v, t) to update PT_w after deleting v (line 30). For each (w, v, t), the algorithm identifies all the arithmetic sequences (the elements in PT_w) that contain the timestamp t based on the inverted index structure (lines 31-33). For each arithmetic sequence, the algorithm decreases the degree of w at timestamp t by 1 (line 34). If the updated degree is smaller than k, the algorithm deletes the arithmetic sequence from PT_w (lines 35-36), because it is no longer a valid (σ, k)-periodic time support set. Since our algorithm correctly computes and maintains PT_w for every node w, the correctness of Algorithm 3 can be guaranteed. Below, we analyze the time and space complexity of Algorithm 3.

Theorem 3. The time and space complexity of Algorithm 3 is $O(\alpha m + n(\alpha\sigma + \mathcal{T}^2\sigma^{-1})$ and $O(m + n\alpha\sigma)$ respectively, where $\alpha = \max_{u\in V_c}\{|\mathsf{PT}_u|\}$.

Proof: First, in line 2, the algorithm computes the k-core on the de-temporal graph which takes $O(m + n)$ time. In line 7, the algorithm takes $O(n\mathcal{T}^2\sigma^{-1})$ time in the worst case by Lemma 2. In line 9, the algorithm spends $O(\alpha\sigma)$ time to construct the inverted index for each node u in the worst case. Therefore, the total time cost for computing the inverted index can be bounded by $O(\alpha\sigma n)$. For each temporal edge (w, v, t), the algorithm updates PT_w at most once, and the cost for each update can be bounded by $O(\alpha)$ in lines 31-35. Therefore, the total cost for updating all PT_w is bounded by $O(\alpha m)$. Putting it all together, the time complexity of Algorithm 3 is $O(\alpha m + n(\alpha\sigma + \mathcal{T}^2\sigma^{-1})$.

Note that for each node u, the space overhead of the inverted index IPT_u is equal to that of PT_u. Since the algorithm uses $O(\alpha\sigma)$ space for storing PT_u, the total space overhead for maintaining both IPT_u and PT_u for all $u \in V_c$ is bounded by $O(n\alpha\sigma)$. The algorithm also needs to store the temporal graph and the queue $\mathcal{Q}$ which consumes $O(m + n)$ space. Therefore, the space complexity of Algorithm 3 is $O(m + n\alpha\sigma)$. □

Note that the time complexity of Algorithm 3 is lower than that of Algorithm 1, as α is smaller than $\mathcal{T}^2\sigma^{-1}$. In practice, the space usage of Algorithm 3 is much smaller than the worst-case bound, because our algorithm only work on the k-core subgraph which is typically significantly smaller than the original temporal graph.

3.2 The PECluster pruning rule

Although PNCluster can prune many unpromising nodes, it is not very effective for pruning unpromising edges. For example, in Fig. 3, the edge (v_4, v_5) is clearly not a σ-*periodic edge* with $\sigma = 3$, as the timestamps associated with this edge cannot form an 3-term arithmetic sequence. As a result, such an edge cannot be contained in any σ-periodic k-clique. To overcome this defect, we propose a novel (σ, k)-periodic edge cluster (abbreviated as PECluster) pruning technique which combines both σ-periodic nodes and edges for pruning. Below, we give a definition of σ-periodic edge.

Definition 10 (σ-periodic edge). Given a temporal graph $\mathcal{G}$, its de-temporal graph G and parameter σ, an edge $(u, v) \in G$ is called a σ-periodic edge if there exists a σ-periodic time support set for the subgraph $\{(u, v)\}$.

It is easy to see that a σ-periodic edge is also a special σ-periodic subgraph, because we can treat an edge (u, v) as a special subgraph. Therefore, every σ-periodic edge also has a set of σ-periodic time support sets. For convenience, we let EPT_{uv} be the set of all those σ-periodic time support sets for a σ-periodic edge (u, v). Clearly, an edge is a σ-periodic edge if and only if $\mathsf{EPT}_{uv} \neq \emptyset$. Note that to determine whether an edge (u, v) is a σ-periodic edge, we can make use of a similar algorithm as shown in Algorithm 2 to compute EPT_{uv}, which takes $O(|\mathcal{T}|^2\sigma^{-1})$ time in the worst case. Based on Definition 10, we define the σ-periodic-link graph in the following.

Definition 11. A subgraph $\tilde{G}_c = (\tilde{V}_c, \tilde{E}_c)$ of the de-temporal graph G is called a σ-periodic-link graph if every edge $(u, v) \in \tilde{E}_c$ is a σ-periodic edge.

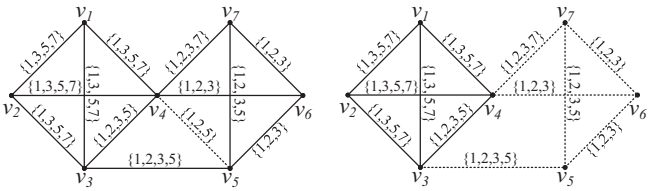
By Definition 11, we can obtain the maximum σ-periodic-link graph by removing all the *non-periodic* edges from G (i.e., only retain all the σ-periodic edges in G). The following example illustrates the above definitions.

Example 4. Reconsider the temporal graph $\mathcal{G}$ shown in Fig. 3. Suppose that $\sigma = 3, k = 3$. Then, we can see that the edge (v_4, v_5) has three timestamps $\{1, 2, 5\}$ which clearly cannot form a 3-term arithmetic sequence. Therefore, we have $\mathsf{EPT}_{v_4v_5} = \emptyset$, indicating that (v_4, v_5) is not a σ-periodic edge. We can easily derive that all the other edges in the de-temporal graph G (except (v_4, v_5)) are σ-periodic edges. Hence, the maximum σ-periodic-link graph is a subgraph by removing edge (v_4, v_5) in G which is shown in Fig. 5(a). □

Based on the maximum σ-periodic-link graph, we define the (σ, k)-periodic edge cluster (PECluster) as follows.

Definition 12 ((σ, k)-periodic edge cluster). A subgraph S of the maximum σ-periodic-link graph $\hat{G}_c = (\hat{V}_c, \hat{E}_c)$ is called a (σ, k)-periodic edge cluster if it satisfies the following constraints.

(1) ***Periodic edge constraint:*** for any edge (u, v) in S,



(a) σ-periodic-link graph (b) Subgraph reduced by PECluster

Fig. 5. Illustration of the PECluster pruning ($\sigma = 3, k = 3$)

$\mathsf{PT}_u \cap \mathsf{PT}_v \cap \mathsf{EPT}_{uv} \neq \emptyset$;

(2) ***Maximal constraint:*** there does not exist a subgraph S' of $\hat{G}_c$ that satisfies (1) and $S \subset S'$.

Based on Definition 12, we are able to derive several useful properties for the (σ, k)-periodic edge cluster.

Lemma 3. Given a temporal graph $\mathcal{G}$, its de-temporal graph G, parameters σ and k, the PECluster is unique in G if it exists.

Proof: The proof is similar to the proof of Lemma 1, thus we omit for brevity. □

Lemma 4. Let G be the de-temporal graph, G_w be the subgraph induced by the PNCluster, and G_s is the PECluster. Then, we have $G_s \subseteq G_w$.

Proof: By Definition 12, each edge $(u, v) \in G_s$ meets $\mathsf{PT}_u \cap \mathsf{PT}_v \cap \mathsf{EPT}_{uv} \neq \emptyset$, indicating that both $\mathsf{PT}_u \neq \emptyset$ and $\mathsf{PT}_v \neq \emptyset$. As a result, all the nodes in G_s is a (σ, k)-periodic node. Since PNCluster G_w is a maximal subgraph that consists of all (σ, k)-periodic nodes, we have $G_s \subseteq G_w$. □

As shown in Lemma 4, such a PECluster pruning technique is more powerful than the PNCluster pruning technique, since it may prune more edges and nodes of the original temporal graph. Below, we develop an algorithm to efficiently compute the PECluster.

The basic idea of PECluster algorithm is that we first compute the subgraph induced by the PNCluster, denoted by G_w. Then, we identify all the edges in G_w that do not satisfy the periodic edge constraint in Definition 12 (i.e., find the edge (u, v) meeting $\mathsf{PT}_u \cap \mathsf{PT}_v \cap \mathsf{EPT}_{uv} = \emptyset$). After that, we delete all those unpromising edges from G_w. Note that the deletion of an edge (u, v) may trigger u and v's outgoing edges that violate the periodic edge constraint. Therefore, we need to iteratively perform this edge peeling procedure, until no edge can be removed.

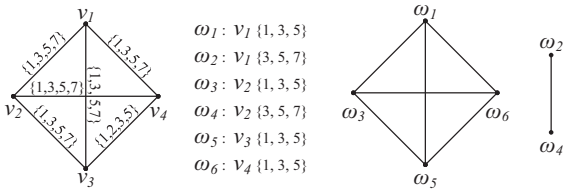
The detailed description of PECluster algorithm is shown in Algorithm 4. In line 1, the algorithm first invokes Algorithm 3 to calculate the PNCluster V_w. Note that by Algorithm 3, we are able to obtain PT_u and IPT_u for each $u \in V_w$ (line 2). Also, we can easily get the subgraph $G_w = (V_w, E_w)$ induced by V_w. The algorithm uses a queue EQ and a set ED to maintain all the unpromising edges (line 5). For each $(u, v) \in E_w$, the algorithm computes the set $\mathsf{EPT}(u, v)$ (lines 6-7), which is denoted by EPT_{uv} in Algorithm 4. Then, if (u, v) violates the periodic edge constraint ($\mathsf{PT}_u \cap \mathsf{PT}_v \cap \mathsf{EPT}_{uv} = \emptyset$)), the algorithm pushes it into the queue EQ (lines 8-9). Subsequently, the algorithm iteratively deletes the element in EQ (lines 10-19). For each $(u, v) \in EQ$, the algorithm needs to update PT_u and PT_v by invoking the UpdatePeriod procedure (lines 12 and 16). This is because the deletion of an edge (u, v) decreases the degrees of both u and v by 1 which may further result in the updating of PT_u and PT_v. Since PT_u (or PT_v) may update, the algorithm has to verify each edge (u, x) (or edge (v, x)) for $x \in N_u(G_w)$ whether it satisfies the periodic edge constraint or not (lines 13-15 and lines 17-19). If the edge (u, x) (or edge (v, x)) does not satisfy the periodic edge constraint, the algorithm pushes it into EQ (lines 15 and 19). The algorithm terminates when $EQ = \emptyset$. At this moment, the subgraph comprises all the remaining edges is a PECluster. Since all the edges that violate the periodic edge constraint are deleted and every remaining edge meets the periodic edge constraint, Algorithm 4 can correctly compute the PECluster. The following example illustrates how Algorithm 4 works.

Algorithm 4: PECluster $(\mathcal{G}, \sigma, k)$

Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parameters σ, and k
Output: The PECluster $G_s = (V_s, E_s)$

1 $V_w \leftarrow$ PNCluster+ $(\mathcal{G}, \sigma, k)$;
2 PT_u and IPT_u have already been computed in PNCluster+ for each $u \in V_w$;
3 Let $G = (V, E)$ be the de-temporal graph of $\mathcal{G}$;
4 Let $G_w = (V_w, E_w)$ be the subgraph induced by V_w in G;
5 $EQ \leftarrow \emptyset$; $ED \leftarrow \emptyset$;
6 **for** *each* $(u, v) \in E_w$ **do**
7 Compute EPT_{uv};
8 **if** $\mathsf{PT}_u \cap \mathsf{PT}_v \cap \mathsf{EPT}_{uv} = \emptyset$ **then**
9 $EQ.push((u, v))$;
10 **while** $EQ \neq \emptyset$ **do**
11 $(u, v) \leftarrow EQ.pop()$; $ED \leftarrow ED \cup \{(u, v)\}$;
12 UpdatePeriod $(\mathsf{PT}_u, \mathsf{IPT}_u, v, k)$;
13 **for** $x \in N_u(G_w)$ **do**
14 **if** $(u, x) \notin EQ$ *and* $(u, x) \notin ED$ **then**
15 **if** $\mathsf{PT}_u \cap \mathsf{PT}_x \cap \mathsf{EPT}_{ux} = \emptyset$ **then** $EQ.push((u, x))$;
16 UpdatePeriod $(\mathsf{PT}_v, \mathsf{IPT}_v, u, k)$;
17 **for** $x \in N_v(G_w)$ **do**
18 **if** $(v, x) \notin EQ$ *and* $(v, x) \notin ED$ **then**
19 **if** $\mathsf{PT}_v \cap \mathsf{PT}_x \cap \mathsf{EPT}_{vx} = \emptyset$ **then** $EQ.push((v, x))$;
20 **return** $G_s \leftarrow$ the subgraph comprises all edges in $E_w \setminus ED$;

Example 5. Reconsider the temporal graph shown in Fig. 3. Suppose that $\sigma = 3, k = 3$. First, by computing the PNCluster, the algorithm can obtain an induced subgraph $G_w = (V_w, E_w)$ where $V_w = \{v_1, \cdots, v_7\}$. Then, the algorithm calculates EPT_{uv} for each $(u, v) \in E_w$ (lines 6-7). Clearly, we have $\mathsf{EPT}_{v_4v_5} = \emptyset$, thus the algorithm pushes (v_4, v_5) into EQ (lines 8-9). Also, the algorithm pushes (v_3, v_5) into EQ. The reason is that $\mathsf{PT}_{v_3} = \{[1, 3, 5]\}$, $\mathsf{PT}_{v_5} = \{[1, 2, 3]\}$, $\mathsf{EPT}_{v_3v_5} = \{[1, 2, 3]\}$, thus $\mathsf{PT}_{v_3} \cap \mathsf{PT}_{v_5} \cap \mathsf{EPT}_{v_3v_5} = \emptyset$ (lines 8-9). Subsequently, the algorithm pops (v_4, v_5) from EQ and updates PT_{v_4} and PT_{v_5}. Since PT_{v_4} and PT_{v_5} do not change after deleting (v_4, v_5), the algorithm continues to pop (v_3, v_5) from EQ. After removing (v_3, v_5), PT_{v_5} is updated to be an empty set. Thus, the algorithm will pushes (v_5, v_6) and (v_5, v_7) into EQ, and then iteratively processes these two edges. When the algorithm terminates, we can obtain the PECluster as shown in Fig. 5(b) (the subgraph induced by the nodes $\{v_1, \cdots, v_4\}$). Compared to the PNCluster pruning, the PECluster pruning can prune many additional nodes and edges, indicating that the PECluster pruning is indeed much more powerful than the PNCluster pruning. □



(a) Transformed nodes (b) Transformed graph $\tilde{G}$

Fig. 6. Illustration of the graph transformation method

Theorem 4. The time and space complexity of Algorithm 4 is $O(m|\mathcal{T}|^2\sigma^{-1})$ and $O(m|\mathcal{T}|^2\sigma^{-1})$ respectively.

Proof: First, the algorithm takes $O(\alpha m + n(\alpha\sigma + |\mathcal{T}|^2\sigma^{-1})$ to compute the PNCluster, where $\alpha = \max_{u\in V_c}\{|\mathsf{PT}_u|\}$. Second, the algorithm consumes $O(|\mathcal{T}|^2\sigma^{-1})$ time to compute $\mathsf{EPT}(u,v)$ for each $(u,v) \in E_w$. Thus, the total time costs to compute $\mathsf{EPT}(u,v)$ for all edges in E_w can be bounded by $O(m|\mathcal{T}|^2\sigma^{-1})$. Third, for each deleted edge (u,v), the algorithm takes $O(\alpha)$ time to update PT_u and PT_v. Therefore, the total time complexity of lines 10-17 can be bounded by $O(\alpha m)$. Putting it all together, the time complexity of Algorithm 4 is $O(m|\mathcal{T}|^2\sigma^{-1})$. For the space complexity, it is easy to show that the total space usage of our algorithm can be bounded by $O(m|\mathcal{T}|^2\sigma^{-1})$. □

Note that since our algorithm only works on the PNCluster (not the original temporal graph), the time cost of Algorithm 4 is much less than the worst case bound in practice, which is also confirmed in our experiments.

4 Transforming the Temporal Graph into Static Graph by Periodic Nodes and Edges

In this section, we present the approach of transforming the temporal graph into static graph by periodic nodes and edges. Recall that in PECluster $G_s = (V_s, E_s)$, each node u has a set of (σ,k)-periodic time support sets, i.e., PT_u, and each edge (u,v) also has a set of σ-periodic time support sets, i.e., EPT_{uv}. Since every node u and every edge (u,v) in the periodic community shares at least one periodic time support set, we can decompose the periodic community into a set of nodes and edges which are associated with the same periodic time support sets. This motivate us to construct a graph $\tilde{G} = (\tilde{V}, \tilde{E})$ as follows. For each node $v \in V_s$ and an element PT_v^s in PT_v, we construct a node (v, PT_v^s) for $\tilde{V}$. As a result, for each node $v \in V_s$, we can obtain $|\mathsf{PT}_v|$ nodes in $\tilde{V}$. For any two nodes (u, PT_u^s) and (v, PT_v^s) in $\tilde{V}$, we create an edge $(u, v, \mathsf{EPT}_{uv}^s)$ if and only if $\mathsf{EPT}_{uv}^s = \mathsf{PT}_u^s = \mathsf{PT}_v^s$ (i.e., the same arithmetic sequence), where EPT_{uv}^s is an element in EPT_{uv}. This is because for any edge (u,v) in a periodic community, the nodes u, v and the edge (u,v) shares the same periodic time support set. Clearly, by this construction, each node in the transformed graph is a two-tuple (a node and a periodic time support set), and each edge is a three-tuple (an edge and a periodic time support set). The following example illustrates our graph transformation method.

Example 6. Consider the temporal graph shown in Fig. 3. Suppose that σ=3, k=3. Then, the reduced graph by PECluster is shown in Fig. 5(b). Based on the reduced graph, we can obtain the transformed graph $\tilde{G}$ shown in Fig. 6. Specifically, Fig. 6(a) depicts the reduced graph and the transformed nodes. For example, for the node v_1, we have $\mathsf{PT}(v_1) = \{[1,3,5],[3,5,7]\}$. Therefore, we construct two nodes $\omega_1 = (v_1, [1,3,5])$ and $\omega_2 = (3,5,7)$ in $\tilde{G}$. Similarly, we can obtain four other nodes in $\tilde{G}$ which are $\omega_3, \cdots, \omega_6$ as shown in Fig. 6(a). Since the nodes v_1, v_2, and edge (v_1, v_2) are associated with the same periodic time support sets $[1,3,5]$ and $[3,5,7]$, we can obtain two edges (ω_1, ω_3) and (ω_2, ω_4) in the transformed graph $\tilde{G}$. Likewise, we can get all the other edges in $\tilde{G}$. The final transformed graph $\tilde{G}$ is shown in Fig. 6(b) which contains 6 nodes and 7 edges. □

Below, we show the relation of PECluster, MPCore, MPClique and MAXPClique.

Lemma 5. Any node and edge in a MPCore must be contained in the PECluster.

Proof: According to Definition 4, a MPCore C is a σ-periodic subgraph, and any node v in C has at least k neighbors. By Definition 7, any node v is a (σ,k)-periodic node, and therefore C satisfies the periodic degree constraint. Since the PNCluster is a maximal subgraph meeting the periodic degree constraint, C must be contained in the PNCluster.

Consider an edge (u,v) in a MPCore C. Since C is a σ-periodic subgraph, there exist at least one σ-periodic time support set TS for C. Since $u, v \in C$, TS is also a (σ,k)-periodic time support set for both nodes u and v by Definitions 7 and 8. Similarly, TS is also a σ-periodic time support set for the edge (u,v), as $(u,v) \in C$. As a result, we have $\mathsf{PT}_u \cap \mathsf{PT}_v \cap \mathsf{EPT}_{uv} \neq \emptyset$. Since each edge $(u,v) \in C$ meets $\mathsf{PT}_u \cap \mathsf{PT}_v \cap \mathsf{EPT}_{uv} \neq \emptyset$, (u,v) must be contained in the PECluster by Definition 12. □

Lemma 6. Any node and edge in a MPClique must be contained in the MPCore.

Proof: The proof can be easily get by Definition 4 and 5, thus we omit it for brevity. □

Corollary 2. Any node and edge in σ-periodic k-ECC must be contained in the maximal σ-periodic k-core, σ-periodic k-truss must be in σ-periodic k-ECC, σ-periodic k-clique must be in σ-periodic k-truss.

Based on Lemmas 5 and 6, we know that MPCore and MPClique are contained in the PECluster of G. Therefore, we can first compute the PECluster to prune unpromising nodes and edges, and then mine MPCore and MPClique on the reduced graph. The following lemma shows that any MPCore in temporal graph $\mathcal{G}$ is an unique corresponding core in the transformed graph $\tilde{G}$.

Lemma 7. For any σ-periodic k-core $C = (V_c, E_c)$ in the reduced graph G_s, there exists a k-core $\tilde{C} = (\tilde{V}_c, \tilde{E}_c)$ in the transformed graph $\tilde{G}$ such that the node set $\tilde{V}_c$ of $\tilde{C}$ is equal to V_c. For any maximal k-core $\tilde{C} = (\tilde{V}_c, \tilde{E}_c)$ with node set $\tilde{V}_c$ sharing the same σ-periodic time support set in $\tilde{G}$, the subgraph induced by $\tilde{V}_c$ is a maximal σ-periodic k-core C in G_s.

Proof: First, by Definition 5, each σ-periodic k-core $C = (V_c, E_c)$ in G_s has at least one σ-periodic time support set TS. According to the graph transformation method, we can easily derive that the set of nodes $\{(v, TS) | v \in V_c\}$ form a k-core $\tilde{C}$ in $\tilde{G}$. Clearly, by our definition, we have $V_c = \tilde{V}_c$.

Algorithm 5: MPCore $(\mathcal{G}, \sigma, k)$

Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parameters σ and k
Output: Set of maximal σ-periodic k-core $\mathcal{C}$

1 $G_s = (V_s, E_s) \leftarrow$ PECluster$(\mathcal{G}, \sigma, k)$;
2 PT_u has been computed in PECluster for each $u \in V_s$;
3 EPT_{uv} has been calculated in PECluster for each $(u, v) \in E_s$;
4 $\tilde{G} = (\tilde{V}, \tilde{E}) \leftarrow$ the transformed graph based on PT_u and EPT_{uv};
5 Let $d_{(u,\mathsf{PT}^s_u)}(\tilde{G})$ be the degree of node (u, PT^s_u) in $\tilde{G}$; $\mathcal{Q} \leftarrow \emptyset$; $D \leftarrow \emptyset$;
6 **for** $(u, \mathsf{PT}^s_u) \in \tilde{V}$ *in an increasing order by* $d_{(u,\mathsf{PT}^s_u)}(\tilde{G})$ **do**
7 **if** $u \in D$ **then continue**;
8 **if** $d_{(u,\mathsf{PT}^s_u)}(\tilde{G}) < k$ **then** $\mathcal{Q}.push((u, \mathsf{PT}^s_u))$;
9 **while** $\mathcal{Q} \neq \emptyset$ **do**
10 $(v, \mathsf{PT}^s_v) \leftarrow \mathcal{Q}.pop()$; $D \leftarrow D \cup \{(v, \mathsf{PT}^s_v)\}$;
11 **for** $(w, \mathsf{PT}^s_w) \in N_{(v,\mathsf{PT}^s_v)}(\tilde{G})$, *s.t.* $d_{(w,\mathsf{PT}^s_w)}(\tilde{G}) \geq k$ **do**
12 $d_{(w,\mathsf{PT}^s_w)}(\tilde{G}) \leftarrow d_{(w,\mathsf{PT}^s_w)}(\tilde{G}) - 1$;
13 **if** $d_{(w,\mathsf{PT}^s_w)}(\tilde{G}) < k$ **then** $\mathcal{Q}.push(w)$;
14 $\mathcal{C} \leftarrow \tilde{G}_{\tilde{V} \setminus D}$; // $\mathcal{C}$ can be grouped into different maximal σ-periodic k-core by PT^s_u in each node (u, PT^s_u)
15 **return** $\mathcal{C}$;

Second, for each maximal k-core $\tilde{C} = (\tilde{V}_c, \tilde{E}_c)$ in which nodes in $\tilde{V}_c$ share the same σ-periodic time support set, we have $\mathsf{EPT}^s_{uv} = \mathsf{PT}^s_u = \mathsf{PT}^s_v \neq \emptyset$ for any two nodes $u, v \in \tilde{V}_c$ by the graph transformation approach. As a result, the node set $\tilde{V}_c$ forms a k-core C, and all nodes and edges share the same σ-periodic time support set EPT^s_{uv}. Since $\tilde{C}$ is a maximal clique, no node can be added into $\tilde{C}$ while maintaining the property of $\mathsf{EPT}^s_{uv} = \mathsf{PT}^s_u = \mathsf{PT}^s_v \neq \emptyset$. Therefore, the maximal k-core $\tilde{C}$ in $\tilde{G}$ is a MPCore in G_s. □

Corollary 3. Any MPClique in temporal graph $\mathcal{G}$ is an unique corresponding maximal clique in the transformed graph $\tilde{G}$.

Proof: We can prove that (1) For any MPClique $C' = (V'_c, E'_c)$ in the reduced graph G_s, there exists a maximal clique $\tilde{C} = (\tilde{V}_c, \tilde{E}_c)$ in the transformed graph $\tilde{G}$ such that the node set $\tilde{V}_c$ is equal to V'_c. (2) For any maximal clique $\tilde{C} = (\tilde{V}_c, \tilde{E}_c)$ with node set $\tilde{V}_c$ in $\tilde{G}$, the subgraph induced by $\tilde{V}_c$ is a MPClique in G_s. The detail is similar to the proof of Lemma 7. □

5 Mining the Periodic Communities

5.1 Searching maximal σ-periodic k-core

Based on Lemma 7, any MPCore in temporal graph $\mathcal{G}$ is an unique corresponding core in the transformed graph $\tilde{G}$. Therefore, we can first compute the PECluster and transform the temporal graph by the construction method in section 4, and then perform a decomposition algorithm in the transformed graph. The detail of the MPCore algorithm is shown as follows.

Algorithm 5 invokes the PECluster pruning technique (Algorithm 4) to prune the temporal graph (line 1). Note that in this pruning procedure, we can also obtain PT_u for each $u \in V_s$ and EPT_{uv} for each $(u, v) \in E_s$ (lines 2-3). Based on PT_u and EPT_{uv}, the algorithm can construct the transformed graph $\tilde{G}$ (line 4). We can see that each node in $\tilde{G}$ is a two-tuple like (u, PT^s_u) in which u is the node id in G_s and PT^s_u is a σ-periodic time support set of node u. Then, the algorithm performs a decomposition algorithm to search the maximal σ-periodic k-core. It maintains $\mathcal{Q}$ to store the deleting nodes and D to store the deleted nodes (line 5). Then the algorithm pushes the node with degree less than k into $\mathcal{Q}$ and checks whether their neighbors meet the degree constrict (lines 8-13). After all the nodes in $\tilde{V}$ have been checked in line 6, the remained node set $\tilde{V} \setminus D$ is the set of all nodes in the maximal σ-periodic k-core. Furthermore, $\mathcal{C}$ can be grouped into different maximal σ-periodic k-core by PT^s_u in each node (u, PT^s_u). The node set in $\mathcal{C}$ with different PT^s_u will be in different maximal σ-periodic k-core.

It is easy to see that Algorithm 5 only needs $O(|\tilde{V}|)$ time to find the maximal σ-periodic k-core, so the time and space complexity of Algorithm 5 is $O(m|\mathcal{T}|^2\sigma^{-1})$ and $O(m|\mathcal{T}|^2\sigma^{-1})$ respectively, same as that in Algorithm 4.

5.2 Enumerating maximal σ-periodic k-clique

Recall that the MPClique enumeration problem is NP-hard. Thus, there does not exist a polynomial-time algorithm to solve our problem unless *P=NP*. Moreover, most existing maximal clique enumeration algorithms (e.g., the classic cBron-Kerbosh algorithm [13]) can only work on static graphs, it is not clear how to apply them to identify periodic cliques in temporal graphs. To circumvent this problem, we propose a new Bron-Kerbosch style enumeration algorithm, called MPClique, which can efficiently compute the complete set of all MPCliques.

The MPClique **algorithm.** Based on Corollary 3, we are able to obtain the complete set of MPCliques by enumerating all maximal cliques in $\tilde{G}$. Since $\tilde{G}$ is a static graph, we make use of a Bron-Kerbosch style algorithm to identify all maximal cliques in $\tilde{G}$. The detailed description of our algorithm is shown in Algorithm 6. It first invokes Algorithm 5 to find the set of MPCliques, because any node and edge in a MPClique must be contained in the MPCore based on Lemma 6. Then, the algorithm performs a Bron-Kerbosch algorithm with pivoting technique to identify all maximal cliques in $\tilde{G}$ (line 3). Specifically, the set $\tilde{R}$ denotes the current clique, $\tilde{P}$ denotes the set of candidate nodes, and $\tilde{X}$ denotes the set of nodes that have already been processed. Note that each node in $\tilde{P}$, $\tilde{R}$, and $\tilde{X}$ is a two-tuple (v, PT^s_v). In line 10, the algorithm adopts a similar pivoting technique developed in [14] to speed up the enumeration procedure. Note that the operator $N_{(v,\mathsf{PT}^s_v)}(\tilde{G})$ is to take the neighbors of the node (v, PT^s_v) in the transformed graph $\tilde{G}$. The correctness of Algorithm 6 can be guaranteed by [14] and Corollary 3.

Number of MPCliques**.** Below, we analyze the number of MPCliques in the temporal graph $\mathcal{G}$ based on a novel concept of σ-periodic degeneracy. The classic degeneracy is a well-known metric for measuring the sparsity of a static graph [8]. Many real-life networks are often very sparse, thus having a small degeneracy [8]. Below, we give the definition of degeneracy.

Definition 13 (Degeneracy). The degeneracy of a static graph G is the minimum integer δ such that each subgraph S of G contains a node v with degree no larger than δ.

Eppstein et al. [8] proved that the number of maximal cliques in a static graph is bounded by $(|V| - \delta)3^{\delta/3}$. They also developed an efficient maximal clique enumeration algorithm with time complexity $O(\delta|V|3^{\delta/3})$ based on the

Algorithm 6: MPClique $(\mathcal{G}, \sigma, k)$

Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parameters σ and k
Output: Set of maximal σ-periodic k-clique $\mathcal{C}$

$\tilde{G} = (\tilde{V}, \tilde{E}) \leftarrow$ MPCore $(\mathcal{G}, \sigma, k)$;
global $\mathcal{C} \leftarrow \emptyset$;
EnumClique $(\tilde{V}, \emptyset, \emptyset, k)$;
return $\mathcal{C}$;

Procedure EnumClique $(\tilde{P}, \tilde{R}, \tilde{X}, k)$
if $|\tilde{P}| + |\tilde{R}| < k$ **then return**;
if $\tilde{P} \cup \tilde{X} = \emptyset$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{\tilde{R}\}$;
$(v', \mathsf{PT}^s_{v'}) \leftarrow \arg \max_{(v,\mathsf{PT}^s_v) \in \tilde{P} \cup \tilde{X}} |\tilde{P} \cap N_{(v,\mathsf{PT}^s_v)}(\tilde{G})|$;
for $(v, \mathsf{PT}^s_v) \in \tilde{P} \setminus N_{(v',\mathsf{PT}^s_{v'})}(\tilde{G})$ **do**
  $\tilde{R}' \leftarrow \tilde{R}' \cup (v, \mathsf{PT}^s_v)$;
  $\tilde{P}' \leftarrow \tilde{P} \cap N_{(v,\mathsf{PT}^s_v)}(\tilde{G})$; $\tilde{X}' \leftarrow \tilde{X} \cap N_{(v,\mathsf{PT}^s_v)}(\tilde{G})$;
  EnumClique $(\tilde{P}', \tilde{R}', \tilde{X}', k)$;
  $\tilde{P} \leftarrow \tilde{P} \setminus (v, \mathsf{PT}^s_v)$; $\tilde{X} \leftarrow \tilde{X} \cup (v, \mathsf{PT}^s_v)$;

degeneracy ordering. The classic degeneracy, however, cannot be directly used to bound the number of MPCliques in temporal graphs. Below, we introduce a novel concept, called σ-periodic degeneracy, which will be applied to bound the number of MPCliques.

Definition 14 (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 $\hat{\delta}$.

Since the degeneracy-based bound for the number of maximal cliques is tailored for static graph [8], it is not clear how to use the σ-periodic degeneracy to bound the number of MPCliques in temporal graph. To circumvent this problem, we resort to bound the number of maximal cliques in the transformed graph $\tilde{G}$. The rationale is that the number of maximal cliques in $\tilde{G}$ is no less than the number of MPCliques in the temporal graph $\mathcal{G}$ by Lemma 3. Since the transformed graph $\tilde{G}$ is a static graph, we are capable of applying the results developed by Eppstein et al. [8] to bound the number of maximal cliques in $\tilde{G}$. Let $\tilde{\delta}$ be the degeneracy of the transformed graph $\tilde{G}$. Then, the following lemma shows that $\tilde{\delta}$ is bounded by $\hat{\delta}$.

Lemma 8. For any temporal graph $\mathcal{G}$ and the transformed graph $\tilde{G}$ of the PECluster of $\mathcal{G}$, we have $\tilde{\delta} \leq \hat{\delta}$.

Proof: First, we consider a connected subgraph $\tilde{S}$ of the transformed graph $\tilde{G}$. By our graph transformation method, the connected subgraph $\tilde{S}$ must be a σ-periodic subgraph in $\mathcal{G}$. Therefore, each connected subgraph $\tilde{S}$ of $\tilde{G}$ can be mapped to a σ-periodic subgraph S' in $\mathcal{G}$. By definition, there exists a node u in $\tilde{S}$ having degree at most $\tilde{\delta}$. Hence, the degree of u in the σ-periodic subgraph S' is also no larger than $\tilde{\delta}$. By Definition 14, any σ-periodic subgraph in $\mathcal{G}$ must have a node with degree at most $\hat{\delta}$. Therefore, we have $\tilde{\delta} \leq \hat{\delta}$. Second, for any disconnected subgraph $\widetilde{DS}$ in $\tilde{G}$, we consider two cases: (1) each connected component of $\widetilde{DS}$ has the same σ-periodic time support set, and (2) the connected components of $\widetilde{DS}$ are associated with different σ-periodic time support sets. For the case (1), we can easily check that $\widetilde{DS}$ can be mapped to a σ-periodic subgraph $\widetilde{DS}'$ in $\mathcal{G}$. Therefore, the above argument can also be used to prove $\tilde{\delta} \leq \hat{\delta}$. For the case (2), since $\widetilde{DS}$ cannot be mapped to a σ-periodic subgraph, we do not need to bound $\hat{\delta}$. Putting it all together, we can derive that $\tilde{\delta} \leq \hat{\delta}$. □

Based on Lemma 8, we can leverage $\hat{\delta}$ to bound the number of MPCliques in $\mathcal{G}$ as shown in the following theorem.

Theorem 5. Given a temporal graph $\mathcal{G}$, parameters σ and k, the number of maximal σ-periodic k-cliques (MPCliques) in $\mathcal{G}$ is less than $(4m^2k^{-2}\sigma^{-1} - \hat{\delta})3^{\hat{\delta}/3}$.

Proof: Note that if a node $u \in \mathcal{G}$ is a (σ, k)-periodic node, the largest cardinality of the time support set of u in which u has degree at least k is less than $d_u(\mathcal{G})/k$. By Lemma 2, the number of (σ, k)-periodic time support sets of u is bounded by $\sum_{p=1}^{\lfloor \frac{(d_u(\mathcal{G})/k)-1}{\sigma-1} \rfloor}(d_u(\mathcal{G})/k - \sigma p) < (d_u(\mathcal{G})/k)^2\sigma^{-1}$. Therefore, the total number of (σ, k)-periodic time support sets for all nodes in $\mathcal{G}$, denoted by $\tilde{N}$, is bounded by $\sum_{u \in V}(d_u(\mathcal{G})/k)^2\sigma^{-1} < 4m^2k^{-2}\sigma^{-1}$. Recall that by our graph transformation approach, the number of nodes in the transformed graph $\tilde{G}$ can be bounded by $\tilde{N}$. According to the results developed by Eppstein et al. [8], the number of maximal cliques in a static graph with degeneracy δ is at most $(|V| - \delta)3^{\delta/3}$, where $|V|$ is the number of nodes of the graph. As a consequence, the number of maximal cliques in the transformed graph $\tilde{G}$ is no larger than $(4m^2k^{-2}\sigma^{-1} - \tilde{\delta})3^{\tilde{\delta}/3}$. Since the number of maximal cliques in $\tilde{G}$ is no less than the number of MPCliques in $\mathcal{G}$ and $\tilde{\delta} \leq \hat{\delta}$, the number of MPCliques in $\mathcal{G}$ can be bounded by $(4m^2k^{-2}\sigma^{-1} - \hat{\delta})3^{\hat{\delta}/3}$. □

Based on the results developed by Eppstein et al. [8], we can also bound the worst-case time complexity of the MPClique enumeration problem by the σ-periodic degeneracy of $\mathcal{G}$, i.e., $\hat{\delta}$. Specifically, we have the following results.

Theorem 6. Given a temporal graph $\mathcal{G}$, parameters σ and k, there exists an algorithm to enumerate all MPCliques in $\mathcal{G}$ in $O(\hat{\delta}m^2k^{-2}\sigma^{-1}3^{\hat{\delta}/3})$ time, where $\hat{\delta}$ is the σ-periodic degeneracy and m is the number of temporal edges in $\mathcal{G}$.

Proof: Since the transformed graph $\tilde{G}$ is a static graph, the algorithm proposed by Eppstein et al. [8] with worst-case time complexity $O(\delta|V|3^{\delta/3})$ can also be applied to enumerate all maximal cliques in $\tilde{G}$. Note that the degeneracy and the number of nodes of $\tilde{G}$ is bounded by $\hat{\delta}$ and $m^2k^{-2}\sigma^{-1}$ respectively. The time complexity of the algorithm devised by Eppstein et al. is $O(\hat{\delta}m^2k^{-2}\sigma^{-1}3^{\hat{\delta}/3})$ to enumerate all maximal cliques in $\tilde{G}$, thus the theorem is established. □

Not that Theorem 6 indicates that enumerating all MPCliques in a temporal graph $\mathcal{G}$ is *fixed-parameter tractable* with respect to the parameter σ-periodic degeneracy $\hat{\delta}$ of $\mathcal{G}$. Since the σ-periodic degeneracy of $\mathcal{G}$ is typically very small in real-life temporal graphs, the proposed algorithm can be very efficient in practice.

5.3 Finding maximum σ-periodic clique

We can also use the transformed graph with a branch-and-bound algorithm framework to find the maximum periodic clique [15]. In the worst case, finding the maximum periodic clique need to enumerate all the maximal cliques. However, we can modify the search space of Algorithm 6 to search the maximum periodic clique. Specifically, we do not need the flag set $\tilde{X}$ to check whether the clique is maximal and we

need to design several pruning rules for the candidate maximum periodic clique. Intuitively, we maintain $MaxSize$ to record the size of the MAXPClique, $Size$ to record the size of the candidate clique in current loop and $\mathcal{C}$ to record the candidate nodes which will be checked to add into the candidate clique. Below, we will introduce several pruning rules which can make termination for the enumeration.

Pruning rule 1: the current max size lower bound. Suppose that we have maintained the current $MaxSize$ of the MAXPClique, this pruning rule is based on that the newly joined node must have degree larger than $MaxSize$ to form a clique of size $MaxSize+1$. This pruning rule will perform when the search space is rebuilt.

Pruning rule 2: the color-based upper bound. This pruning rule performs when we have the current $Size$ for the MAXPClique, and there comes a set of nodes $\mathcal{C}$ which will be added to form a larger MAXPClique. Intuitively, if the current $Size$ plus the size of $\mathcal{C}$ is not larger than the maintained $MaxSize$, in this loop $MaxSize$ will certainly not changed. However, the upper bound based on the candidate set size $|\mathcal{C}|$ is not very tight, because $Size + |\mathcal{C}|$ is often larger than $MaxSize$. A tighter bound can be easily derived by a coloring algorithm. In particular, we assign a color to each node in $\mathcal{C}$ using a degree-ordering based greedy coloring algorithm [16] so that no two adjacent nodes have the same color. The colors of the nodes in MAXPClique must be different. Therefore, let $\mathsf{color}(\mathcal{C})$ be the number of colors of the candidate nodes in $\mathcal{C}$, $\mathsf{color}(\mathcal{C}) + Size$ is an upper bound of the size of a maximum clique in the current loop. For an efficient implement, we only invoke the greedy coloring algorithm once, and compute the upper bound $\mathsf{color}(\mathcal{C})+Size$ in each search subspace $\mathcal{C}$ based on the same coloring result. Note that the greedy coloring algorithm can be implemented in linear time w.r.t. the uncertain graph size [16] and compute the upper bound in each search subspace can be done in $O(|\mathcal{C}|)$ time, thus such a basic color-based pruning technique is very efficient.

Pruning rule 3: early termination based on degeneracy. We can have that the size of MAXPClique $MaxSize$ will be no larger than the degeneracy $\tilde{\delta}$ of the transformed $\tilde{G}$. Because we can find a maximum clique in which every node's degree is $MaxSize$, and according to Definition 13 $MaxSize < \tilde{\delta}$. Computing $\tilde{\delta}$ only needs $O(|\tilde{V}||\tilde{E}|)$ time, so this pruning technique is efficient.

Algorithm 7 details the pseudo-code for computing the size of the MAXPClique. In line 1, it first initializes $MaxSize$ as 3, since we do not consider the MAXPClique with size less than 3. Then, it invokes Algorithm 5 to compute the MPCore, because MAXPClique must be in MPClique and MPClique are in MPCore according to Lemma 6. Next, it computes the degeneracy $\tilde{\delta}$ in transformed graph $\tilde{G}$ and it can be used in line 17 to make an early termination based on pruning rule 3. Furthermore, the algorithm visits node $u_i \in \tilde{V}$ to search the maximum clique (lines 4-10). $\mathcal{C}$ records the candidate nodes which will be checked to add into the candidate clique and pruning rule 1 performs in line 9. Procedure FindClique can search the maximum clique by the given candidate nodes $\mathcal{C}$ and parameter $MaxSize$. In lines 13-15, if $\mathcal{C} = \emptyset$, the current $Size$ is the maximum size of clique in this loop, and it updates $MaxSize$ if $Size > MaxSize$.

Algorithm 7: MAXPClique $(\mathcal{G}, \sigma)$

Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and parameter σ
Output: Size of the maximum σ-periodic clique

```
1  global MaxSize ← 3;
2  G̃ = (Ṽ, Ẽ) ← MPCore (G, σ, MaxSize);
3  δ̃ ← degeneracy of the transformed graph G̃; // each node is a two-tuple
   like (u, PT_u^s)), for convenient, we use u to represent one node here
4  for i ← 1 : |Ṽ| in a descending order w.r.t. the degree of the node u_i ∈ Ṽ do
5      if |N_{u_i}(G̃)| > MaxSize then
6          C ← ∅;
7          for node u_j ∈ N_{u_i}(G̃) do
8              if j > i and |N_{u_j}(G̃)| > MaxSize then
9                  C ← C ∪ (v, PT_v^s);                    [Pruning rule 1]
10         FindClique (C, MaxSize);
11 return MaxSize;
12 Procedure FindClique (C, Size)
13 if C = ∅ then
14     if Size > MaxSize then MaxSize ← Size;
15     return;
16 while |C| > 0 do
17     if MaxSize = δ̃ then return;                         [Pruning rule 3]
18     if color(C) + Size ≤ MaxSize then return;           [Pruning rule 2]
19     for node u ∈ C do
20         C' ← (C \ u) ∩ N_u(G̃);
21         C' ← {v|v ∈ C'; |N_v(G̃) ∩ C'| ≥ MaxSize};       [Pruning rule 1]
22         FindClique (C', Size + 1);
```

While $\mathcal{C}$ is not empty set, it first uses pruning rule 3 and 2 to check whether terminate the loop or not (lines 17-18), and then checks each node in $\mathcal{C}$ to form a larger clique (lines 19-22). The whole algorithm finished after all the nodes are checked in line 4. Finally, it returns the size of the MAXPClique. Note that, if we add a stored node set, it can also output one clique of the maximum size.

The problem of mining maximum clique is also NP-hard [16], due to the proposed pruning rules, the time complexity of Algorithm 7 can be bounded by that of Algorithm 6 in Theorem 6. However, the pruning rules can reduce the computation time greatly. We will show the running time in practice at Section 6.

6 Experiments

In this section, we conduct extensive experiments to evaluate the efficiency and effectiveness of the proposed algorithms. In our experiments, we implement various algorithms for comparison.

- MPCO-KC is a baseline algorithm integrated with k-core reduction techniques. It first computes PT_u and EPT_{uv} for nodes and edges in KCore of G using Algorithm 2, and then constructs a transformed graph $\tilde{G}$. It uses the core decomposition algorithm to search MPCore on $\tilde{G}$.
- MPCO-NC denotes the MPCO-KC algorithm integrated with the PNCluster reduction rule.
- MPCO-EC denotes the MPCO-KC algorithm with the PECluster reduction rule, i.e., Algorithm 5.
- MPCL-KC is a baseline algorithm with k-core reduction techniques. It constructs $\tilde{G}$ by PT_u and EPT_{uv} in KCore of G. Then it uses the Bron-Kerbosch algorithm with a pivoting technique to enumerate all MPClique on $\tilde{G}$.
- MPCL-NC denotes the MPCL-KC algorithm integrated with the PNCluster reduction rule.
- MPCL-EC denotes the MPCL-KC algorithm with the PECluster reduction rule, i.e., Algorithm 6.

TABLE 2
Datasets

Dataset	$\|V\|$	$\|E\|$	$\|\mathcal{E}\|$	$d_{\max}$	$\|\mathcal{T}\|$	Time scale
HS	327	5,818	20,448	322	101	hour
PS	242	8,317	26,351	393	34	hour
LKML	26,885	159,996	328,092	14,172	96	month
Enron	86,978	297,456	499,983	4,311	48	month
DBLP	1,729,816	8,546,306	12,007,380	5,980	59	year

- MAXPCL denotes the MAXPClique algorithm with all the pruning rules, i.e., Algorithm 7.
- MAXPCL-B denotes the MAXPClique algorithm without the three pruning rules in Section 5.3.

To evaluate the effectiveness of the proposed maximal σ-periodic k-clique model, we also use PNCluster and PECluster as two intuitive baseline models. The reasons are as follows. First, to the best of our knowledge, there is no existing community model that can be used to model periodic communities in temporal networks. Second, by Definitions 9 and 12, both PNCluster and PECluster can capture periodic and cohesive properties of a community in temporal graphs, thus PNCluster and PECluster can serve as two baselines for modeling periodic communities.

All algorithms are implemented in Python and the source code is available at https://github.com/VeryLargeGraph/MPC/. All the experiments are conducted on a server of Linux kernel 4.4 with Intel Core(TM) i5-8400 @ 3.20GHz and 32 GB main memory.

Datasets. We use five different types of real-life temporal networks in the experiments. The detailed statistics of our datasets are summarized in Table 2. In Table 2, the first two datasets are human contact temporal networks which are download from (http://www.sociopatterns.org/datasets/). Specifically, HS is a temporal network of face-to-face contacts between students in a French high school [2], and PS is a temporal network of contacts between the children and teachers in a French primary school [2]. Each snapshot of these temporal networks is extracted in a hour. Both LKML and Enron are temporal communication networks downloaded from (http://konect.uni-koblenz.de), where each temporal edge (u, v, t) represents an email communication from a user u to v at time t. Each snapshot of these temporal networks is extracted in a month. DBLP is a temporal collaboration network of authors in DBLP downloaded from (http://dblp.uni-trier.de/xml/), where each temporal edge (u, v, t) denotes that two authors u and v co-authored one paper at time t. Each snapshot of DBLP is extracted in a year. In Table 2, $d_{\max}$ is the maximum number of temporal edges associated with a node, and $|\mathcal{T}|$ denotes the number of snapshots.

Parameter settings. There are two parameters k, σ in our algorithm. For the parameter k, we vary it from 3 to 7 with a default value of 3. We also vary σ from 3 to 7 with a default value of 3. Unless otherwise specified, the value of the other parameter are set to its default value when varying a parameter.

6.1 Efficiency Testing

Exp-1: Comparison between PNCluster **and** PNCluster+**.** Fig. 7 evaluates the running time of PNCluster (Algorithm 1) and PNCluster+ (Algorithm 3) for computing

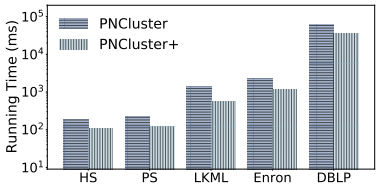
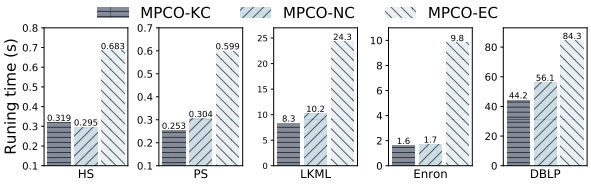
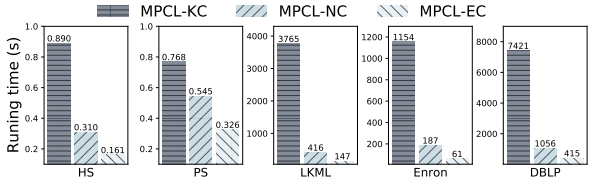


Fig. 7. Running time of PNCluster V.S. PNCluster+



(a) Time of searching MPCore



(b) Time of enumerating MPCliques

Fig. 8. Running time of different algorithms on various datasets

(σ, k)-periodic node cluster under the default parameter setting. As can be seen, PNCluster+ is much faster than PNCluster on all datasets. The running time of PNCluster+ is around a half of the running time of PNCluster. For example, on Enron, PNCluster+ takes 1.1 seconds and PNCluster consumes 2.3 seconds to identify (σ, k)-periodic node cluster. The reason is that PNCluster+ is based on an on-demand computing paradigm which can substantially reduce redundant computations. These results are consistent with our theoretical analysis presented in Section 3.1. In the following experiments, we will use PNCluster+ to compute (σ, k)-periodic node cluster.

Exp-2: Efficiency of various MPCore **and** MPClique **mining algorithms.** Fig. 8 shows the running time of MPCO-KC, MPCO-NC, MPCO-EC and MPCL-KC, MPCL-NC, MPCL-EC on different datasets with parameters $\sigma = 4$, $k = 4$. Similar results can also be observed under the other parameter settings. From Fig. 8(a), we can see that MPCO-KC is faster than the other competitors on all datasets. This is because that in MPCO-NC and MPCO-EC, the process of invoking ComputePeriod in Algorithm 2 spends lots of extra time. However, from Fig. 8(b), we can see that MPCL-EC is much faster than the others on all datasets. For example, on DBLP, MPCL-EC takes around 7 minutes to enumerate all MPCliques which cuts the running time over MPCL-NC and MPCL-KC by 154% and 1,688% respectively. These results indicate that the PECluster pruning rule is indeed very powerful in practice which are consistent with our analysis in Section 3.2.

Exp-3: Efficiency with varying parameters. Table. 3 reports the running time of MPCO-KC, MPCO-NC, MPCO-EC and MPCL-KC, MPCL-NC, MPCL-EC with varying parameters on DBLP. Similar results can also be observed on the other datasets. At the above part of the table, it can be observed that MPCO-KC is quicker than MPCO-NC and MPCO-EC under most parameter settings, but the performance gap is

TABLE 3
Running time (s) of different algorithms with varying parameters (on DBLP)

	$\sigma=3$			$\sigma=4$			$\sigma=5$			$\sigma=6$			$\sigma=7$		
	MPCO-KC	MPCO-NC	MPCO-EC	MPCO-KC	MPCO-NC	MPCO-EC	MPCO-KC	MPCO-NC	MPCO-EC	MPCO-KC	MPCO-NC	MPCO-EC	MPCO-KC	MPCO-NC	MPCO-EC
$k=3$	**27.5**	36.2	127.3	**16.5**	24.4	98.5	**13.3**	23.6	66.3	**9.6**	20.6	44.1	**9.0**	16.5	33.0
$k=4$	**20.5**	29.7	97.2	**13.1**	21.7	64.5	**11.1**	19.8	40.5	**8.8**	18.0	30.2	**8.2**	15.8	21.6
$k=5$	**17.4**	23.1	74.8	**11.1**	18.5	40.1	**8.4**	16.7	25.8	**6.6**	15.5	19.8	**5.7**	14.0	15.0
$k=6$	**15.9**	19.0	44.7	**7.0**	15.4	26.5	**6.3**	11.6	17.9	**6.8**	9.3	10.8	**5.3**	9.6	10.2
$k=7$	**10.0**	16.0	30.4	**6.0**	13.6	17.1	**5.5**	11.8	12.4	**6.6**	8.3	9.5	**5.5**	9.5	9.6
	$\sigma=3$			$\sigma=4$			$\sigma=5$			$\sigma=6$			$\sigma=7$		
	MPCL-KC	MPCL-NC	MPCL-EC	MPCL-KC	MPCL-NC	MPCL-EC	MPCL-KC	MPCL-NC	MPCL-EC	MPCL-KC	MPCL-NC	MPCL-EC	MPCL-KC	MPCL-NC	MPCL-EC
$k=3$	*INF*	32,213	**4,339**	24,517	12,313	**1,237**	8,321	4,567	**936**	4,235	3,456	**804**	2,145	1,023	**144**
$k=4$	23,100	3,574	**580**	7,421	1,056	**415**	3,441	774	**326**	1,960	114	**71**	1,467	48	**38**
$k=5$	9,770	736	**280**	3,428	801	**75**	1,771	417	**45**	1,220	63	**35**	1,023	**33**	37
$k=6$	4,464	621	**112**	2,035	585	**45**	1,643	142	**32**	980	32	**27**	576	**14**	16
$k=7$	2,382	534	**24**	1,292	51	**23**	1,201	44	**27**	620	21	**20**	231	**10**	11

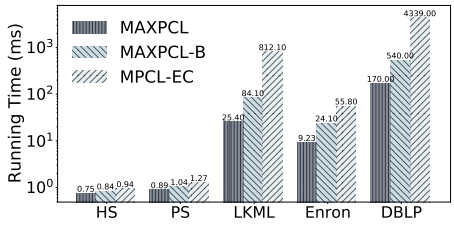


Fig. 9. Running time of MAXPCL V.S. MAXPCL-B

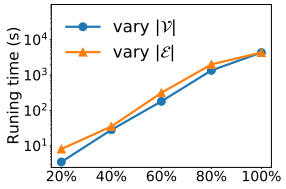


Fig. 10. Scalability testing of MPCL-EC (DBLP)

decreasing with increasing k and σ. It is the reason that MPCO-NC and MPCO-EC need to invoking ComputePeriod and the processing time is decreasing when k and σ increase. At the below part of the table, we can see MPCL-EC is faster than all the other algorithms under almost all parameter settings. In general, the running time of MPCL-KC, MPCL-NC and MPCL-EC decrease with increasing k and σ, because the size of the transformed graph decreases as k or σ increases. Note that when $\sigma = 7$ and $k \geq 5$, MPCL-NC is slightly faster than MPCL-EC. The reason could be that for a large σ and k, the original temporal graph can be reduced to a very small graph by PNCluster, thus the benefit of PECluster may be not significant.

Exp-4: Efficiency of MAXPCL **V.S.** MAXPCL-B**.** Fig. 9 shows the running time of MAXPCL, MAXPCL-B. We also put the running time of MPCL-EC here for comparison. It can be observed that MAXPCL is much faster than MPCL-EC on all datasets, which means that the branch-and-bound style framework is efficient in practice. We can also see that on LKML, Enron and DBLP, MAXPCL-B needs about triple time as much as MAXPCL. Those results indicate that the proposed pruning rules in Section 5.3 are effective.

Exp-5: Scalability testing. Fig. 10 shows the scalability of MPCL-EC on DBLP. Similar results can also be observed on the other datasets or other algorithms. We generate four temporal subgraphs by randomly picking 20%-80% of the nodes (temporal edges), and evaluate the running time of MPCL-EC on those subgraphs. As can be seen, the running time increases smoothly with increasing $|\mathcal{V}|$ and $|\mathcal{E}|$. These results suggest that the MPCL-EC algorithm is scalable when handling large temporal networks.

Exp-6: Memory overhead. The most uncontrollable memory usage in the above algorithms is the storage of PT_u and EPT_{uv} for MPCL-EC in Algorithm 4. Table 4 shows the memory usage of MPCL-EC on different datasets. We can see that the memory usage of MPCL-EC is higher than the size of the temporal graph, because MPCL-EC needs to store

TABLE 4
Memory overhead of MPCL-EC

Memory	Graph	PT + EPT	Memory (all)
HS	5.2MB	25.2MB	45MB
PS	2.8MB	15.8MB	35MB
LKML	20.1MB	35.4MB	101MB
Enron	53.3MB	98.6Mb	198MB
DBLP	678.5MB	2,398MB	3,234MB

TABLE 5
Number of nodes in the reduced graph

	KCore		PNCluster		PECluster	
HS	326	99%	280	86%	165	51%
PS	242	100%	233	96%	211	87%
LKML	9,773	36%	1,785	6.6%	926	3.4%
Enron	18,591	21%	3,314	3.8%	2,315	2.7%
DBLP	1,258,540	73%	126,357	7.3%	73,109	4.2%

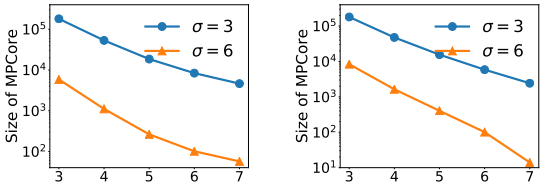
PT_u and EPT_{uv} (for each node and edge). However, on large datasets, it is typically lower than five times of the size of the temporal graph. For instance, MPCL-EC consumes 3,234MB memory on DBLP while the temporal graph uses 678.5MB memory. These results indicate that MPCL-EC achieves near linear space complexity which confirms our theoretical analysis in Sections 3.2 and 5.2.

6.2 Effectiveness Testing

Exp-7: Number of nodes in the reduced graph. Table 5 shows the number of remaining nodes in de-temporal graph G obtained by KCore, PNCluster and PECluster on all datasets under the default parameter setting. In columns 2-4 of Table 5, the left integer is the number of remaining nodes and the right value is the percentage of the remaining nodes over all nodes in the graph. As can be seen, both PNCluster and PECluster can prune a large number of unpromising nodes on large datasets. For example, on DBLP, the number of remaining nodes obtained by PNCluster and PECluster is only 7.3% and 4.2% of the original graph respectively. These results confirm that our graph reduction techniques are indeed very effective on large real-life temporal networks.

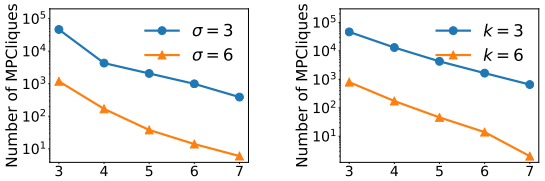
TABLE 6
The size of the transformed graph reduced by PECluster

	$\lvert\tilde{V}\rvert$	$\lvert\tilde{E}\rvert$	#MPCliques	$\tilde{\delta}$	|MAXPClique|
HS	1,946	3,388	18	24	10
PS	3,508	12,174	475	47	11
LKML	149,385	505,514	17,382	12	6
Enron	37,869	173,914	10,203	53	13
DBLP	353,557	1,028,598	45,442	120	21



(a) Vary k (DBLP) (b) Vary σ (DBLP)

Fig. 11. Size of MPCore with varying parameters



(a) Vary k (DBLP) (b) Vary σ (DBLP)

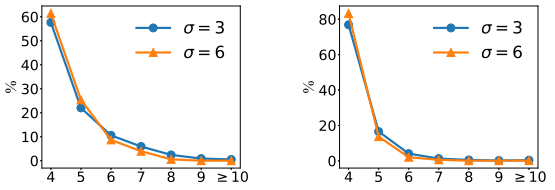
Fig. 12. Number of MPCliques with varying parameters

Exp-8: Size of the transformed graph. Table 6 reports the size of the transformed graph $\tilde{G} = (\tilde{V}, \tilde{E})$ generated by PECluster under the default parameter setting. We can observe that the size of $\tilde{G}$ scales linearly w.r.t. the original graph size. Moreover, the degeneracy $\tilde{\delta}$ of $\tilde{G}$ is very small in all datasets. The number of MPCliques is clearly less than $(4m^2k^{-2}\sigma^{-1} - \tilde{\delta})3^{\tilde{\delta}/3}$, and the size of MAXPClique is less than $\tilde{\delta}$, which confirms our theoretical analyses in Section 5.2 and 5.3.

Exp-9: Size of MPCore **and number of** MPCliques **with varying** σ, k**.** Fig. 11 shows the size of MPCore with varying σ, k on DBLP. Fig. 12 shows the number of MPCliques with varying σ, k on DBLP. The results on the other datasets are consistent. As shown in Fig. 11(a) and 12(a), the size of MPCore and number of MPCliques drop sharply with an increasing k. Likewise, we can observe from Fig. 11(b) and 12(b) that the size of MPCore and number of MPCliques decreases with a growing σ. The reason is that with a large k or σ, the periodic constraint will be strong, thus the size of MPCore and number of MPCliques decreases. These results confirm the definitions of MPCore, MPCliques and our theoretical analysis in Theorem 6.

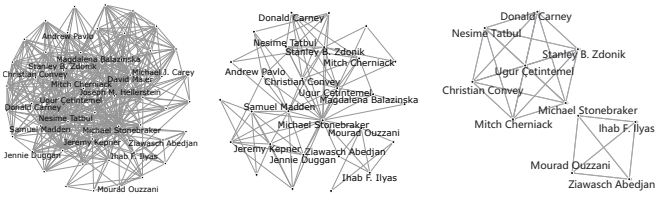
Exp-10: Distribution of the size of MPCliques **with varying parameters.** Fig. 13 shows the distribution of the size of MPCliques on Enron and DBLP with parameters $\sigma = 3$ (or $\sigma = 6$) and $k = 3$. Similar trends can also be observed on the other datasets and using other parameter settings. We can see that most MPCliques has a small size on Enron and DBLP, and very few MPCliques have a size no less than 10. This is because a MPClique must satisfy the periodic clique constraint which may rule out large cliques.

Exp-11: Case study on DBLP**.** We conduct a case study using DBLP to further evaluate the effectiveness of various models. As MPClique is the most cohesive model, we use



(a) Enron (b) DBLP

Fig. 13. Distribution of the size of MPCliques ($k = 3$)



(a) PNCluster (b) PECluster (c) MPClique

Fig. 14. Case study on DBLP

MPClique to compare with other baselines here. Fig. 14 shows three communities of Prof. Michael Stonebraker obtained by PNCluster, PECluster and MPClique respectively, using default parameters. As can be seen in Fig. 14(c), the community obtained by MPClique contains two cliques, and each clique comprises the close and long-term collaborators of Prof. Michael Stonebraker. Moreover, we find that each clique appears in 2015, 2016, and 2017 year, suggesting that there are two periodic communities containing Prof. Michael Stonebraker in recent years. From Figs. 14(a-b), we can see that the communities obtained by PNCluster and PECluster not only contain two MPCliques in Fig. 14(c), but they also include some short-term collaborators of Prof. Michael Stonebraker who did not collaborate with him periodically, which indicates that both PNCluster and PECluster models cannot fully capture the periodic patterns of a community. These results further confirm that MPClique is more effective than the baselines to detect periodic communities in temporal graphs.

7 Related Work

Temporal graph analysis. Our work is related to the studies on temporal graph analysis. Yang et al. [17] proposed an algorithm to detect frequent changing components in the temporal graph. Huang et al. [18] investigated the minimum spanning tree problem in temporal graphs. Gurukar et al. [19] presented a model to identify the recurring subgraphs that have similar sequence of information flow in temporal graphs. Wu et al. [20] proposed an efficient algorithm to answer the reachability and time-based path queries in temporal graphs. Yang et al. [3] studied a problem of finding a set of diversified quasi-cliques from a temporal graph. Wu et al. [7] proposed a temporal k-core model based on the counts of temporal edges. Ma et al. [4] investigated a dense subgraph problem in temporal graphs. Li et al. [5] developed an efficient algorithm to identify persistent communities in temporal graphs. To the best of our knowledge, our work is the first to study the problem of mining periodic communities in temporal graphs.

Community detection in dynamic graphs. There is a number of studies for mining communities on dynamic

networks [21]. Most of them aim to identify and analyze the community structures that evolve over time. Lin et al. [22] proposed a probabilistic generative model to analyze evolving communities. Chen et al. [23] developed an algorithm for tracking community dynamics. Agarwal et al. [24] studied how to find dense clusters for dynamic microblog streams. Li et al. [25] devised an algorithm to maintain the k-core in large dynamic graphs. Rossetti et al. [26] proposed an online iterative algorithm for tracking the evolution of communities. Unlike these studies, our work focuses mainly on detecting periodic communities in temporal graphs.

Maximal cliques enumeration. Our work is also related to the maximal clique enumeration problem. Notable algorithms for enumerating maximal clique include the classic Bron-Kerbosch algorithm [13] and its variants [8], [14], [27]. Tomita et al. [14] proved that the Bron-Kerbosch algorithm with a pivoting technique is essentially optimal according to the worst-case bound. Eppstein et al. [8] developed an algorithm which is fixed-parameter tractable w.r.t. the degeneracy of the graph. Cheng et al. [27] proposed an external-memory algorithm for clique enumeration in massive graphs. More recently, Himmel [28] developed a Bron-Kerbosch style algorithm for enumerating maximal cliques in temporal graph. Their work, however, cannot be used to enumerate periodic cliques.

Periodic behavior mining. The studies of periodic behavior mining are also related to our work. Notable examples are summarized below. Li et al. [29] addressed the problem of mining periodic behaviors for moving objects. Kurashima et al. [30] modeled the periodic actions in real-world (e.g., eating, sleep, and exercise) to make predictions for future actions. Radinsky et al. [31] also developed a temporal model to predict the periodic actions. Lahiri et al. [32] investigated a problem of mining periodic subgraphs in dynamic social networks. Their work, however, does not consider the communities in the periodic subgraphs, thus cannot be used for mining periodic communities.

8 Conclusion

In this work, we study a problem of mining periodic communities in temporal graph. We propose novel models, including σ-periodic k-core and σ-periodic k-clique, that represents a periodic community in temporal networks. We first develop several new pruning techniques to substantially reduce the size of the temporal graph. Then, we transform the reduced temporal graph into a static graph. Next, we propose a decomposition algorithm to search maximal σ-periodic k-core, a Bron-Kerbosch style algorithm to enumerate all maximal σ-periodic k-cliques, and a branch-and-bound style algorithm to find maximum σ-periodic clique. Comprehensive experiments on five real-life temporal networks demonstrate the efficiency, scalability and effectiveness of our algorithms.

Acknowledgement

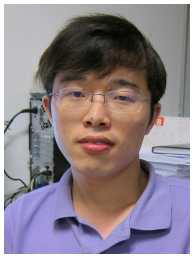
This work was partially supported by (i) NSFC Grants 61772346, 61732003, U1809206, 61932004, 61702435; (ii) Fundamental Research Funds for the Central Universities Grant N181605012.

References

[1] P. Vanhems, A. Barrat, C. Cattuto, J.-F. Pinton, N. Khanafer, C. Regis, B. a Kim, B. Comte, and N. Voirin, "Estimating potential infection transmission routes in hospital wards using wearable proximity sensors," *PLoS ONE*, vol. 8, p. e73970, 2013.
[2] J. Fournet and A. Barrat, "Contact patterns among high school students," *PLOS ONE*, vol. 9, p. e107878, 2014.
[3] Y. Yang, D. Yan, H. Wu, J. Cheng, S. Zhou, and J. C. S. Lui, "Diversified temporal subgraph pattern mining," in *KDD*, 2016.
[4] S. Ma, R. Hu, L. Wang, X. Lin, and J. Huai, "Fast computation of dense temporal subgraphs," in *ICDE*, 2017.
[5] R.-H. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai, "Persistent community search in temporal networks," in *ICDE*, 2018.
[6] I. R.Fischhoff, S. R.Sundaresan, J. Cordingley, H. M.Larkin, and M.-J. Sellier, "Social relationships and reproductive state influence leadership roles in movements of plains zebra, equus burchellii," *Animal Behaviour*, vol. 73, no. 5, pp. 825–831, 2007.
[7] H. Wu, J. Cheng, Y. Lu, Y. Ke, Y. Huang, D. Yan, and H. Wu, "Core decomposition in large temporal graphs," in *IEEE International Conference on Big Data*, 2015.
[8] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," *ACM Journal of Experimental Algorithmics*, vol. 18, 2013.
[9] V. Batagelj and M. Zaversnik, "An O(m) algorithm for cores decomposition of networks," *CoRR cs.DS/0310049*, 2003.
[10] S. B. Seidman, "Network structure and minimum degree," *Social networks*, vol. 5, no. 3, pp. 269–287, 1983.
[11] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang, "Efficiently computing k-edge connected components via graph decomposition," in *SIGMOD*, 2013.
[12] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," *SIGMOD*, 2014.
[13] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.
[14] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical Computer Science*, vol. 363, no. 1, pp. 28–42, 2006.
[15] Q. Wu and J. Hao, "A review on algorithms for maximum clique problems," *European Journal of Operational Research*, vol. 242, no. 3, pp. 693–709, 2015.
[16] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Ordering heuristics for parallel graph coloring," in *SPAA*, 2014, pp. 166–177.
[17] 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.
[18] S. Huang, A. W. Fu, and R. Liu, "Minimum spanning trees in temporal graphs," in *SIGMOD*, 2015.
[19] S. Gurukar, S. Ranu, and B. Ravindran, "COMMIT: A scalable approach to mining communication motifs from dynamic networks," in *SIGMOD*, 2015.
[20] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke, "Reachability and time-based path queries in temporal graphs," in *ICDE*, 2016.
[21] G. Rossetti and R. Cazabet, "Community discovery in dynamic networks: A survey," *ACM Comput. Surv.*, vol. 51, no. 2, pp. 35:1–35:37, 2018.
[22] Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng, "Facetnet: A framework for analyzing communities and their evolutions in dynamic networks," in *WWW*, 2008.
[23] Z. Chen, K. A. Wilson, Y. Jin, W. Hendrix, and N. F. Samatova, "Detecting and tracking community dynamics in evolutionary networks," in *ICDMW*, 2010.
[24] M. K. Agarwal, K. Ramamritham, and M. Bhide, "Real time discovery of dense clusters in highly dynamic graphs: Identifying real world events in highly dynamic environments," *PVLDB*, vol. 5, no. 10, 2012.
[25] R. H. Li, J. X. Yu, and R. Mao, "Efficient core maintenance in large dynamic graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 10, pp. 2453–2465, 2014.
[26] G. Rossetti, L. Pappalardo, D. Pedreschi, and F. Giannotti, "Tiles: an online algorithm for community discovery in dynamic social networks," *Machine Learning*, vol. 106, no. 8, pp. 1213–1241, 2017.
[27] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks," *ACM Trans. Database Syst.*, vol. 36, no. 4, pp. 21:1–21:34, 2011.

[28] A.-S. Himmel, H. Molter, R. Niedermeier, and M. Sorge, "Adapting the bron–kerbosch algorithm for enumerating maximal cliques in temporal graphs," *Social Network Analysis and Mining*, vol. 7, no. 1, pp. 7–35, 2017.
[29] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye, "Mining periodic behaviors for moving objects," in *KDD*, 2010.
[30] T. Kurashima, T. Althoff, and J. Leskovec, "Modeling Interdependent and Periodic Real-World Action Sequences," in *WWW*, 2018.
[31] K. Radinsky, K. Svore, S. Dumais, J. Teevan, A. Bocharov, and E. Horvitz, "Modeling and predicting behavioral dynamics on the web," in *WWW*, 2012.
[32] M. Lahiri and T. Y. Berger-Wolf, "Periodic subgraph mining in dynamic networks," *Knowledge and Information Systems*, vol. 24, no. 3, pp. 467–497, 2010.

Hongchao Qin is currently a Ph.D. Candidate in Northeastern University, China. He received the B.S. degree in mathematics and M.E. degree in computer science from Northeastern University in 2013 and 2015, respectively. His current research interests include social network analysis and data-driven graph mining.

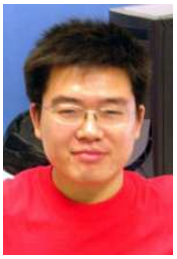


Rong-Hua Li received the Ph.D. degree from the Chinese University of Hong Kong in 2013. He is currently an Associate Professor at Beijing Institute of Technology, Beijing, China. His research interests include graph data management and mining, social network analysis, graph computation systems, and graph-based 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 in the Department of Computer Science, Northeastern University, China. His research interests include graph databases, probabilistic databases, and social network analysis.

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 in 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. He has published more than 100 research papers.

Weihua Yang is currently a professor at Taiyuan University of Technology, Taiyuan, China. He received his Ph.D degree at Laboratoire de Recherche en Informatique-CNRS, Department of computer science, Pairs-Sud University, Paris, France, in 2013. His research interests are in the area of graph theory, connectivity and hamiltonicity of graphs, design and analysis of graph algorithm.



Lu Qin received his bachelor degree from department of Computer Science and Technology in Renmin University of China in 2006, and PhD degree from department of Systems Engineering and Engineering Management in the Chinese University of Hong Kong in 2010. He is now an associate professor in the Centre of Quantum Computation and Intelligent Systems (QCIS) in University of Technology, Sydney (UTS). His research interests include parallel big graph processing, I/O efficient algorithms on massive graphs, and keyword search in graph data.

2019 IEEE 35th International Conference on Data Engineering (ICDE)

Mining Periodic Cliques in Temporal Networks

Hongchao Qin[†], Rong-Hua Li[‡], Guoren Wang[‡], Lu Qin[#], Yurong Cheng[‡], Ye Yuan[†]

[†]*Northeastern University, China;* [‡]*Beijing Institute of Technology, China;* [#]*University of Technology Sydney, Australia*
{qhc.neu;lironghuascut}@gmail.com; wanggrbit@126.com; Lu.Qin@uts.edu.au; yrcheng@bit.edu.cn; yuanye@mail.neu.edu.cn

***Abstract*—Periodicity is a frequently happening phenomenon for social interactions in temporal networks. Mining periodic communities are essential to understanding periodic group behaviors in temporal networks. Unfortunately, most previous studies for community mining in temporal networks ignore the periodic patterns of communities. In this paper, we study a problem of seeking periodic communities in a temporal network, where each edge is associated with a set of timestamps. We propose a novel model, called maximal σ-periodic k-clique, that represents a periodic community in temporal networks. Specifically, a maximal σ-periodic k-clique is a clique with size larger than k that appears at least σ times periodically in the temporal graph. We show that the problem of enumerating all those periodic cliques is NP-hard. To compute all of them efficiently, we first develop two effective graph reduction techniques to significantly prune the temporal graph. Then, we present an efficient enumeration algorithm to enumerate all maximal σ-periodic k-cliques in the reduced graph. The results of extensive experiments on five real-life datasets demonstrate the efficiency, scalability, and effectiveness of our algorithms.**

I. Introduction

In many real-life networks, such as communication networks, scientific collaboration networks, and social networks, the links are often associated with temporal information. For example, in a face-to-face contact network [1], [2], each edge (u, v, t) denotes a contact between two individuals u and v at time t. In an email communication network, each email contains a sender and a receiver, as well as the time when the email was sent. In a scientific collaboration network (e.g., DBLP), each edge (u, v, t) represents that two authors u and v coauthored a paper at time t. The networks that involve temporal information are typically termed as temporal networks [3]–[5].

Periodicity is a frequently happening phenomenon for social interactions in temporal networks. Weekly group meeting, monthly birthday party, and yearly family reunions – these are regular and significant patterns in temporal interaction networks. Mining such periodic group patterns are essential to understanding and predicting group behaviors in a temporal network. In this paper, we investigate a novel data mining problem for temporal networks: periodic community mining, or the detection of all communities that occur at regular time intervals, and show that the proposed technique can be applied to discover the inherent periodicity of communities in a temporal network. Mining the periodic community patterns could be very useful for many practical applications, two of which are listed as follows.

Periodic movement behavior discovery. Consider an application in studying the collective movement behaviors of wild herds of animals [6]. It is well known that the movement behavior of wild herds of animals often exhibits periodic group patterns. In practice, ecologists can tag the animals with tracking sensors to study the collective movement patterns of the animals. In this application, the interactions of the animals (e.g., two animals within a short distance may be considered as an interaction) can be modeled as a temporal network. By mining periodic communities in this temporal network, we are able to identify periodic group movement behaviors of wild animals. Mining such periodic group movement behavior of wild animals can be of ecological interests [6]. For example, if a herd of animals fail to follow the periodic mitigation behavior, it could be a signal of abnormal environment change.

Predicting future activities. Periodic pattern is a predictable pattern, because it repeatedly occurs at regular time intervals. Once we identify a periodic activity, we may predict the same activity will appear within a regular time interval. Based on this observation, we are capable of inferring the future interactions of a group of individuals in a temporal network by mining periodic communities. Taking a temporal scientific collaboration network DBLP as an example, suppose that four researchers A, B, C, and D in DBLP have collaborated with each other in 2015, 2016, and 2017 years. Then, we can infer that these four researchers are likely to coauthor papers in 2018 year.

Recently, the problem of mining communities on temporal graphs has attracted much attention due to numerous applications [4], [5], [7]. For example, Wu et al. [7] proposed a temporal k-core model to find cohesive subgraphs in a temporal network. Ma et al. [4] devised a dense subgraph mining algorithm to identify cohesive subgraphs in a temporal network. Li et al. [5] developed an algorithm to detect persistent communities in a temporal graph. All these community mining algorithms do not consider the periodic patterns of communities, thus cannot be applied to identify periodic communities. To the best of our knowledge, we are the first to study the periodic community mining problem, define the periodic clique model and propose efficient solutions to find periodic cliques in temporal graphs. The main contributions of our work are summarized as follows.

Novel model. We propose a novel maximal σ-periodic k-clique model to characterize periodic communities in a temporal graph, since the clique is the most cohesive structure in a graph and it is a traditional model for community. We show that the traditional maximal clique problem is a special case of the problem of enumerating all maximal σ-periodic k-cliques in a temporal graph. Since the problem of enumerating maximal cliques is NP-hard [8], our problem is also NP-hard.

New algorithms. First, we develop two novel *relaxed* periodic clique models, called WPCore and SPCore, based on the concept of k-core [9]. On the basis of the WPCore and SPCore, we develop two efficient and powerful graph reduction techniques to prune the input temporal graph. We show that both WPCore and SPCore can be computed in near-linear time and space complexity. Second, we propose an enumeration algorithm based on a carefully-design graph transformation technique to efficiently enumerate all maximal σ-periodic k-cliques. In addition, we present a theoretical analysis for the

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00104

number of such cliques in a temporal graph based on a newly-proposed concept called σ-periodic degeneracy $\hat{\delta}$. We also show that the problem of enumerating all maximal σ-periodic k-cliques is fixed-parameter tractable with respect to $\hat{\delta}$, which is often very small in practice as confirmed in our experiments.

Extensive experimental results. We conduct comprehensive experiments on five real-life temporal networks. The results show that our best algorithm is much faster than the baselines on all datasets under most parameter settings. For example, our best algorithm can identify all maximal σ-periodic k-cliques in around 400 seconds on a large temporal graph with more than 1.7 million nodes and 12 million edges. The baseline algorithm, however, cannot get the results in 2 days. We also examine two case studies to evaluate the effectiveness of our model. The results show that our model is indeed able to identify many interesting periodic communities that can not be found by the other models.

Organization. Section II introduces the model of maximal σ-periodic k-clique and formulates our problem. The graph reduction techniques are proposed in Section III. Section IV proposes the enumeration algorithm and presents an analysis of the number of those cliques. The experiments are shown in Section V. We review the related work in Section VI, and conclude this work in Section VII.

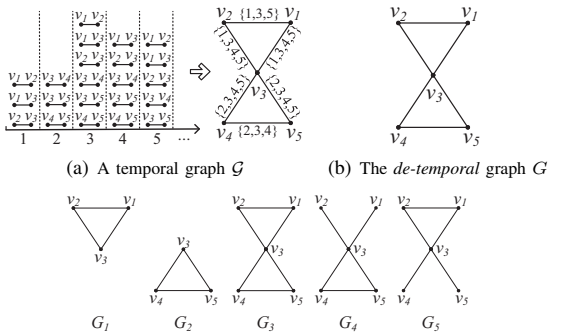
II. Preliminaries

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. Let $n = |\mathcal{V}|$ and $m = |\mathcal{E}|$ be the number of nodes and 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, because the timestamp is an integer in practice. For a temporal graph $\mathcal{G}$, the *de-temporal* graph of $\mathcal{G}$ denoted by $G = (V, E)$ is a graph that ignores all the timestamps associated with the temporal edges. More formally, for the *de-temporal* graph G of $\mathcal{G}$, we have $V = \mathcal{V}$ 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 $S = (V_S, E_S)$ is called a subgraph of $G = (V, E)$ if $V_S \subseteq V$ and $E_S \subseteq E$. A subgraph $S = (V_S, E_S)$ 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 $\mathcal{S} = (\mathcal{V}_S, \mathcal{E}_S)$ is referred to as an induced temporal subgraph of $\mathcal{G}$ if $\mathcal{V}_S \subseteq \mathcal{V}$ and $\mathcal{E}_S = \{(u, v, t)|u, v \in \mathcal{V}_S, (u, v, t) \in \mathcal{E}\}$. For convenience, we use the notion $S \subseteq G$ ($S \subset G$ if $S \neq G$) to represent that S is a subgraph of G.

Given a temporal graph $\mathcal{G}$, we can extract a series of snapshots based on the timestamps. Let $\mathcal{T} = \{t|(u, v, t) \in \mathcal{E}\}$ be the set of timestamps. For each $t_i \in \mathcal{T}$, we can obtain a snapshot $G_i = (V_i, E_i)$ where $V_i = \{u|(u, v, t_i) \in \mathcal{E}\}$ and $E_i = \{(u, v)|(u, v, t_i) \in \mathcal{E}\}$. In the rest of this paper, we assume without loss of generality that all the timestamps are sorted in a chronological order, i.e., $t_1 < t_2 < \cdots < t_{|\mathcal{T}|}$. Fig. 1(a) illustrates a temporal graph $\mathcal{G}$ with 5 nodes and 22 temporal edges. Figs.1(b) and (c) illustrates the de-temporal graph of G and all the five snapshots of $\mathcal{G}$ respectively.

Definition 1 (time support set): Given a temporal graph $\mathcal{G}$, the time support set of a subgraph S is defined as $T(S) \triangleq \{t_i|S \subseteq G_i\}$, where G_i is the i-th snapshot of $\mathcal{G}$.

Definition 2 (σ-periodic time support set): Given a temporal graph $\mathcal{G}$ and a parameter σ, a σ-periodic time support set of a subgraph S, denoted by $\pi_\sigma(S)$, is a subset of $T(S)$ such that (1) $\pi_\sigma(S) = \{t_{j_1}, \cdots, t_{j_\sigma}\}$, and (2) $t_{j_{i+1}} - t_{j_i} = p$ for all $i = 1, \cdots, \sigma - 1$ with any constant p.



(a) A temporal graph $\mathcal{G}$ (b) The *de-temporal* graph G

(c) The five snapshots of $\mathcal{G}$

Fig. 1. Basic concepts of the temporal graph

By Definition 2, we can see that the timestamps of a σ-periodic time support set forms an arithmetic sequence and the cardinality of a σ-periodic time support set is exactly equal to σ. Clearly, there may exist many σ-periodic time support sets for a subgraph S. Based on Definition 2, we define the σ-periodic subgraph below.

Definition 3 (σ-periodic subgraph): Given a temporal graph $\mathcal{G}$, its de-temporal graph G, and a parameter σ, a subgraph S of G is called a σ-periodic subgraph if there exists a σ-periodic time support set $\pi_\sigma(S)$ for S.

By Definition 3, any σ-periodic subgraph $S \subseteq G$ has at least one σ-periodic time support set $\pi_\sigma(S)$. A subgraph S is a maximal σ-periodic subgraph if there is no other σ-periodic subgraph S' that satisfies $S \subset S'$. Based on the widely-used clique model, we propose a novel σ-periodic k-clique model to define the periodic communities as follows.

Definition 4 (σ-periodic k-clique): A σ-periodic k-clique C is a subgraph of the de-temporal graph G such that (1) C is a clique in G with $|C| > k$, and (2) C is a σ-periodic subgraph.

Note that for a typical temporal graph, many σ-periodic cliques are small and may not be interesting to the users. Therefore, it will be more useful to find large σ-periodic cliques for practical applications. As a result, we focus mainly on mining the σ-periodic cliques with size larger than k as defined in Definition 4. Based on Definition 4, we define the maximal σ-periodic k-clique as follows.

Definition 5 (maximal σ-periodic k-clique): A σ-periodic k-clique C is maximal if there is no other σ-periodic k-clique C' meeting $C \subset C'$.

For convenience, in the rest of this paper, the maximal σ-periodic k-clique is abbreviated as MPClique. Below, we use an example to illustrate the above definitions.

Example 1: Consider a temporal graph in Fig. 1(a). Suppose that $\sigma = 3, k = 2$. For the subgraph $S = \{(v_1, v_3), (v_2, v_3)\}$, the time support set of S is $\{1, 3, 4, 5\}$. Clearly, by Definition 2, the set $\{1, 3, 5\}$ is a σ-periodic time support set of S. Therefore, the subgraph S is a σ-periodic subgraph by Definition 3. Note that S is not a maximal σ-periodic subgraph because there is a σ-periodic subgraph $C = \{(v_1, v_3), (v_2, v_3), (v_1, v_2)\}$ that contains S. By Definition 4, we can see that C is a σ-periodic k-clique. Moreover, it is easy to check that C is also a maximal σ-periodic k-clique.

Based on Definition 5, we formulate the periodic community mining problem as follows.

Problem formulation. Given a temporal graph $\mathcal{G}$ and parameters σ and k, the goal of the periodic community mining problem is to enumerate all the maximal σ-periodic k-cliques (MPCliques) in $\mathcal{G}$.

Hardness and challenges. We can show that the traditional maximal clique enumeration problem is a special case of the maximal σ-periodic k-clique enumeration problem. Consider a temporal graph $\mathcal{G}$ that contains a set of snapshots $G = G_i = G_2 =, \cdots, = G_{|\mathcal{T}|}$. Clearly, in this temporal graph $\mathcal{G}$, every subgraph of G is periodic. As a result, the problem of enumerating all maximal σ-periodic k-cliques is equivalent to the problem of enumerating all maximal cliques (with size larger than k) in the *de-temporal* graph G. Since the traditional maximal clique enumeration problem is known to be NP-hard, our problem is also NP-hard.

Although there is a close connection between our problem and the maximal clique enumeration problem, the existing maximal clique enumeration algorithms cannot be directly applied to solve our problem. The reason is that the traditional maximal clique enumeration algorithms, such as the Bron-Kerbosch algorithm [10] can only identify maximal cliques in a snapshot G_i for the timestamp t_i. It is not clearly how to apply this algorithm to derive maximal periodic cliques. To solve our problem, a possible solution is first to enumerate all maximal cliques in the *de-temporal* graph, and then checks which of them is periodic. However, this method is quite complicated and even intractable, because a clique in a snapshot may contain a maximal periodic clique with less nodes in a periodic time support set. Therefore, to identify all MPCliques, we need to check each sub-clique of a maximal clique in each snapshot, which is very costly. Another potential approach is first to enumerate all periodic subgraphs, and then applies a traditional maximal clique enumeration algorithm to identify all MPCliques in each periodic subgraph. Clearly, this approach may involve numerous redundant computations for cliques with the same nodes, because the number of periodic subgraphs may be very large and the same MPClique could be repeatedly enumerated in many different periodic subgraphs. Therefore, the challenge of our problem is how to efficiently enumerate all MPCliques with less redundant computations. In the following sections, we will develop several novel graph reduction techniques and an efficient enumeration algorithm to identify all MPCliques.

III. Reduction by Periodic Cores

In this section, we propose several powerful techniques to prune the unpromising nodes which are definitely not contained in any maximal periodic clique. Our key idea for graph reduction is based on the concept of k-core [11]. Before proceeding further, we first give the definition of k-core (abbreviated as KCore) as follows.

*Definition 6 (*KCore*):* Given a de-temporal graph G of $\mathcal{G}$ and a parameter k, a KCore is a maximal subgraph of G in which every node has degree at least k, i.e., $d_u(G) \geq k$ for $u \in G$.

It is easy to check that if a node is contained in a MPClique, this node will have at least k neighbors in the de-temporal graph G of $\mathcal{G}$. Hence, if a node is not included in the KCore of G, it must be not contained in any MPClique. As a consequence, we can first prune all nodes that are not contained in the KCore of G. This prune rule is simple, but it may be not very effective, because it does not consider the periodic property of the MPClique for pruning. Below, we develop a novel concept called WPCore which can capture the periodic property for pruning.

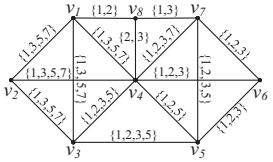


Fig. 2. Running example

A. The WPCore pruning rule

By Definitions 4 and 5, we can easily derive that every node u in a MPClique satisfies a *periodic degree property*: there must exist a σ-periodic subgraph in which u has degree no less than k. Therefore, if a node is not contained in any σ-periodic subgraph, it can be safely pruned. Since the deletion of an unpromising node may trigger its neighbors that violate the periodic degree property, we can iteratively prune the graph until all nodes meet the periodic degree property. Below, we give a definition, called (σ, k)-periodic node, to describe a node that satisfies the periodic degree property.

Definition 7 ((σ, k)-periodic node): Given a temporal graph $\mathcal{G}$, its de-temporal graph G, and parameters σ and k, a node v is called a (σ, k)-periodic node if and only if there exists a σ-periodic subgraph of G in which v has degree at least k.

Recall that by Definition 3, a σ-periodic subgraph may have many σ-periodic time support sets. Therefore, there may also exist many σ-periodic time support sets for a (σ, k)-periodic node v in which v has degree no less than k. Below, we give a definition to describe all σ-periodic time support sets for a (σ, k)-periodic node.

Based on Definition 7, we define the (σ, k)-periodic time support set for a (σ, k)-periodic node as follows.

Definition 8 ((σ, k)-periodic time support set): Given a temporal graph $\mathcal{G}$ and a (σ, k)-periodic node v, the (σ, k)-periodic time support set of v is $\pi^k_\sigma(v) \triangleq [t_{j_1}, \cdots, t_{j_\sigma}]$ that satisfies (1) $t_{j_{i+1}} - t_{j_i} = p$ for each $i = 1, \cdots, \sigma - 1$ with a constant p, and (2) $d_v(S) \geq k$, where S is a subgraph of the snapshot G_{t_i} for each $i = 1, \cdots, \sigma - 1$.

By Definition 8, for any (σ, k)-periodic node v, there is a σ-periodic subgraph S with $\pi_\sigma(S) = \pi^k_\sigma(v)$ in which $d_v(S) \geq k$. Since a σ-periodic subgraph may have many σ-periodic time support sets, there also exist many (σ, k)-periodic time support sets for a (σ, k)-periodic node v. For convenience, we let $\mathsf{PT}(v)$ be the set of all those (σ, k)-periodic time support sets for a node v. Clearly, a node v is a (σ, k)-periodic node if and only if $\mathsf{PT}(v) \neq \emptyset$. Based on the above definitions, we present a new periodic cohesive subgraph model, called σ-periodic weak k-core (abbreviated as WPCore), which will be applied to prune unpromising nodes in enumerating all MPCliques. The WPCore is defined as follows.

Definition 9 (σ-periodic weak k-core): Given a temporal graph $\mathcal{G}$, two integer parameters σ and k, a subset of nodes S in $\mathcal{G}$ is called a σ-periodic weak k-core if it meets the following constraints.

(1) ***Periodic degree constraint:*** each node $u \in S$ is a (σ, k)-periodic node of the temporal subgraph induced by S;

(2) ***Maximal constraint:*** there does not exist a subset of nodes S' in $\mathcal{G}$ that satisfies (1) and $S \subset S'$.

The following example illustrates the above definitions.

Example 2: Consider a temporal graph $\mathcal{G}$ shown in Fig. 2. Note that in Fig. 2, each temporal edge is associated with a set of integers denoting the set of timestamps of that

Algorithm 1: WPCore $(\mathcal{G}, \sigma, k)$

Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parameters σ and k
Output: The WPCore V_w.

```
1  Let G = (V, E) be the de-temporal graph of G;
2  Let G_c = (V_c, E_c) be the k-core of G;
3  Q ← ∅; D ← ∅;
4  for u ∈ V_c do
5      d_u(G_c) ← compute the degree of u in G_c;
6      (flag, PT̃_u) ←ComputePeriod (G, σ, k, u, V_c);
7      if flag = 0 then
8          d_u(G_c) ← 0; Q.push(u);
9  while Q ≠ ∅ do
10     v ← Q.pop(); D ← D ∪ {v};
11     for w ∈ N_v(G_c), s.t. d_u(G_c) ≥ k do
12         d_w(G_c) ← d_w(G_c) − 1;
13         if d_w(G_c) < k then Q.push(w);
14         else
15             (flag, PT̃_w) ←ComputePeriod (G, σ, k, w, V_c \ D);
16             if flag = 0 then
17                 d_w(G_c) ← 0; Q.push(w);
18 return V_w ← V_c \ D ;
```

edge. Clearly, the de-temporal graph G of $\mathcal{G}$ is a 3-core, as every node in G has at least 3 neighbors. For node v_4, we can see that it has degree no less than 3 in timestamps $\{1, 2, 3, 5, 7\}$. Suppose that $\sigma = 3, k = 3$. Then, we can derive that v_4 is a (σ, k)-periodic node. This is because there exists a σ-periodic subgraph $S = \{(v_4, v_3), (v_4, v_6), (v_4, v_7)\}$ in which $d_{v_4}(S) \geq 3$, and the corresponding (σ, k)-periodic time support set for v_4 is $[1, 2, 3]$ (i.e., $\pi^k_\sigma(v_4) = [1, 2, 3]$). It is easy to check that there are three (σ, k)-periodic time support sets for v_4 which are $[1, 2, 3]$, $[1, 3, 5]$ and $[3, 5, 7]$. Thus, we have $\mathsf{PT}(v_4) = \{[1, 2, 3], [1, 3, 5], [3, 5, 7]\}$. Also, we can find that v_8 is not a (σ, k)-periodic node, because no σ-periodic subgraph contains v_8. By Definition 9, we can obtain that $\{v_1, \cdots, v_7\}$ is a WPCore. ■

Below, we show two important properties of the WPCore which will be used for pruning in enumerating all MPCliques.

Lemma 3.1: Any node in a MPClique must be contained in the WPCore of $\mathcal{G}$.

Lemma 3.2: Given a temporal graph $\mathcal{G}$, parameters σ and k, the WPCore is unique in $\mathcal{G}$ if it exists.

Based on Lemmas 3.1 and 3.2, we can first compute the WPCore S of $\mathcal{G}$, and then enumerate all MPCliques on the temporal subgraph induced by the nodes in S. The remaining question is how can we efficiently compute the WPCore. Below, we develop two efficient algorithms to efficiently calculate the WPCore.

The basic WPCore **algorithm.** Similar to the traditional k-core algorithm [9], a basic solution to compute the WPCore is to peel the nodes from $\mathcal{G}$ that violate the periodic degree property. Since the deletion of a node u may result in u's neighbors no longer satisfying the periodic degree property, we need to iteratively process u's neighbors. Such an iterative peeling procedure terminates until no node can be deleted. When the algorithm completes, the remaining nodes form the WPCore. The detailed description of our algorithm is shown in Algorithm 1.

Algorithm 1 first computes the KCore $G_c = (V_c, E_c)$ in the de-temporal graph (lines 1-2), because the WPCore is clearly contained in the KCore. Then, for each node u in V_c, the algorithm invokes Algorithm 2 to checks whether u is a (σ, k)-periodic node or not (lines 4-6). If a node u is not a (σ, k)-periodic node, it will be pushed into a queue $\mathcal{Q}$ (lines 7-8). Subsequently, the algorithm iteratively processes the nodes in $\mathcal{Q}$. In each iteration, the algorithm pops a node v from $\mathcal{Q}$ and uses a set D to maintain all the deleted nodes (line 10). For each neighbor node w of v, the algorithm updates $d_w(G_c)$ (lines 12). If the revised $d_w(G_c)$ is smaller than k, w is clearly not a (σ, k)-periodic node. As a consequence, the algorithm pushes it into $\mathcal{Q}$ which will be deleted in the next iterations (line 13). Otherwise, the algorithm invokes Algorithm 2 to determine whether w is a (σ, k)-periodic node (lines 14-15). If w is not a (σ, k)-periodic node, the algorithm sets $d_w(G_c)$ to 0, and pushes it into $\mathcal{Q}$. The algorithm terminates when $\mathcal{Q}$ is empty. At this moment, the remaining nodes $V_c \setminus D$ is the WPCore of G. Below, we describe the implementation details of Algorithm 2.

Recall that we need to compute the set of (σ, k)-periodic time support set for a node v, i.e., $\mathsf{PT}(v)$, to check whether v is a (σ, k)-periodic node or not. The node v is a (σ, k)-periodic node if and only if $\mathsf{PT}(v)$ is nonempty. By Definition 8, a (σ, k)-periodic time support set can be represented as an arithmetic sequence of the timestamps. In Algorithm 2, we devise a new data structure $\widetilde{\mathsf{PT}}_v$ to represent the set of (σ, k)-periodic time support set for v (i.e., $\mathsf{PT}(v)$). Specifically, $\widetilde{\mathsf{PT}}_v$ is a set where each element TS in $\widetilde{\mathsf{PT}}_v$ is a four-tuple $[s, i, l, ArrD]$ representing an arithmetic sequence. In the four-tuple $[s, i, l, ArrD]$, s denotes the starting timestamp of the arithmetic sequence, i is the common difference, l represents the number of terms of the arithmetic sequence, and $ArrD$ (Array of Degree) is an array that stores the degree of u at each timestamp of the arithmetic sequence.

Based on this data structure, the algorithm makes use of a queue PQ to maintain all the candidates of the arithmetic sequences. The algorithm also uses a set $StartS$ to store all the starting timestamps of the arithmetic sequences. Each element in $StartS$ is a two-tuple $[s, d]$, where s denotes the starting timestamp and d denotes the degree of u at s (lines 15-17). Initially, both PQ and $StartS$ are set to empty sets (line 1). Then, the algorithm enumerates all the timestamps from t_1 to $t_{|\mathcal{T}|}$ (line 2). For each timestamp, the algorithm calculates the number of neighbors of u (denoted by d_u) that are both in G_t (the snapshot at the timestamp t) and the node set F (lines 3-4), i.e., $|N_u(G_t) \cap F|$. If $d_u \geq k$, the algorithm explores all the candidate arithmetic sequences in PQ (lines 5-6). For each candidate $TS \in PQ$, if $(t - TS.s)\%TS.i = 0$, we may extend the arithmetic sequence TS by t (line 7). If $(t - TS.s)/TS.i \neq TS.l$, we know that t cannot extend the current arithmetic sequence TS. Since the remaining timestamps are no less than t, they also cannot extend TS. Therefore, we can safely delete the candidate TS (lines 8-9). Otherwise, the algorithm can augment the arithmetic sequence TS by adding t into TS. In this case, we increase $TS.l$ by 1, and add d_u into the array $TS.ArrD$ (line 8). If the augmented arithmetic sequence TS has σ terms, TS represents a valid (σ, k)-periodic time support set for u (line 11). As a result, the algorithm adds TS into $\widetilde{\mathsf{PT}}_u$ and set $flag$ to 1, denoting that u is a (σ, k)-periodic node (line 12). At this moment, the algorithm can early terminate. Note that Algorithm 2 can also be applied to compute the complete set of (σ, k)-periodic time support sets for u. Clearly, if $t - TS.s > (\sigma - 1)TS.i$, t cannot grow the current arithmetic sequence TS, and TS is no longer to be a valid (σ, k)-periodic time support set. Therefore, the algorithm deletes TS from PQ (line 14). For each starting timestamp $start.s$, the algorithm makes use of

Algorithm 2: ComputePeriod $(\mathcal{G}, \sigma, k, u, F)$

Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parameters σ, k, node u, and node set F
Output: A boolean variable $flag$ and $\mathsf{PT}(u)$

1 $PQ \leftarrow \emptyset$; $StartS \leftarrow \emptyset$; $\widetilde{\mathsf{PT}}_u \leftarrow \emptyset$; $flag \leftarrow 0$;
2 **for** $t \leftarrow t_1 : t_{|\mathcal{T}|}$ **do**
3 Let G_t be the snapshot of $\mathcal{G}$ at timestamp t;
4 $d_u \leftarrow |N_u(G_t) \cap F|$;
5 **if** $d_u \geq k$ **then**
6 **for** *each* $TS \leftarrow [s, i, l, ArrD] \in PQ$ **do**
7 **if** $(t - TS.s)\%TS.i = 0$ **then**
8 **if** $(t - TS.s)/TS.i \neq TS.l$ **then**
9 $PQ.pop(TS)$; **continue**;
10 $TS.l \leftarrow TS.l + 1$; $TS.ArrD \leftarrow TS.ArrD \cup \{d_u\}$;
11 **if** $TS.l = \sigma$ **then**
12 $\widetilde{\mathsf{PT}}_u \leftarrow \widetilde{\mathsf{PT}}_u \cup \{TS\}$; $flag \leftarrow 1$;$PQ.pop(TS)$;
13 /* For WPCore, the algorithm can early terminate. */
14 **if** $t - TS.s > (\sigma - 1)TS.i$ **then** $PQ.pop(TS)$;
15 **for** $start \leftarrow [s, d] \in StartS$ **do**
16 $PQ.push([start.s, t - start.s, 2, \{start.d, d_u\}])$;
17 $StartS \leftarrow StartS \cup \{[t, d_u]\}$;
18 **return** $(flag, \widetilde{\mathsf{PT}}_u)$;

the current timestamp t and $start.s$ to generate an initial arithmetic sequence (lines 15-16). The algorithm also applies the current timestamp t to generate a new starting timestamp which will be used for the next iterations (line 17). Since Algorithm 2 explores all the possible arithmetic sequences, it is able to correctly compute $\widetilde{\mathsf{PT}}_u$. The following example illustrates how Algorithm 2 works.

Example 3: Reconsider the temporal graph in Fig. 2. Suppose that $\sigma = 3, k = 3$. It is easy to derive that v_4 has degree no less than 3 at the timestamps $\{1, 2, 3, 5, 7\}$. Fig. 3 illustrates the candidate arithmetic sequences when the algorithm processes a timestamp in $\{1, 2, 3, 5, 7\}$. The first row in Fig. 3 shows the starting timestamp of the candidate arithmetic sequences. When $t = 1$, the starting timestamp is $\{1\}$, and the set $StartS = \{[1, 6]\}$ (since $d_{v_4} = 6$ at timestamp 1). When $t = 2$, there is a candidate arithmetic sequence $\{1, 2\}$, and the queue PQ has an element $[1, 1, 2, \{6, 5\}]$. Similarly, when $t = 3$ there are three candidates which are $\{1, 2, 3\}$, $\{1, 3\}$, and $\{2, 3\}$. Clearly, $\{1, 2, 3\}$ is a valid (σ, k)-periodic time support set for v_4. When $t = 5$, the timestamp 5 cannot extend $\{2, 3\}$, thus $\{2, 3\}$ is deleted. It is easy to check that the timestamp 5 can extend $\{1, 3\}$, $\{1\}$, $\{2\}$, and $\{3\}$. As a result, we can obtain four candidates $\{1, 3, 5\}$, $\{1, 5\}$, $\{2, 5\}$, and $\{3, 5\}$. Likewise, when $t = 7$, we have seven candidates which are $\{3, 5, 7\}$, $\{1, 5\}$, $\{2, 5\}$, $\{1, 7\}$, $\{2, 7\}$, $\{3, 7\}$ and $\{5, 7\}$. Note that our algorithm cannot delete the candidate $\{1, 5\}$ when $t = 7$, because $\{1, 5\}$ could be extended by $t > 7$ (similar for $\{2, 5\}$). Clearly, we can obtain three (σ, k)-periodic time support sets for v_4 which are $[1, 2, 3]$, $[1, 3, 5]$, and $[3, 5, 7]$. ■

Analysis of Algorithm 1. Below, we analyze the correctness and complexity of Algorithm 1.

Theorem 3.1: Algorithm 1 correctly computes the WPCore.

Proof: Let $S = V_c \backslash D$. Clearly, by Algorithm 1, each node in S is a (σ, k)-periodic node of the temporal subgraph induced by S. To prove that S is a WPCore, we need to show the maximal property of S. Suppose to the contrary that there is a set S' such that (1) every node in S' is a (σ, k)-periodic node of the temporal subgraph induced by S', and (2) $S \subset S'$. Since $S \subset S'$, there exists a (σ, k)-periodic node $u \in S' \setminus S$ in the temporal subgraph induced by S'. Note that by our assumption, every node in S' has degree no less than k in a σ-periodic subgraph of the temporal graph induced by S'. Thus, Algorithm 1 cannot delete the node u. Therefore, $u \in V_c \setminus D$ which is a contradiction. □

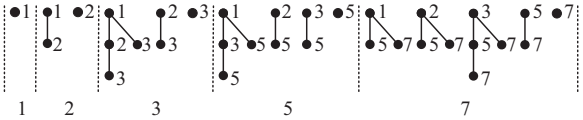


Fig. 3. Illustration of using Algorithm 2 to compute $\mathsf{PT}(v_4)$

The complexity of Algorithm 1 is shown as follows.

Lemma 3.3: For a temporal graph $\mathcal{G}$ with $|\mathcal{T}|$ timestamps, there are at most $O(|T|^2\sigma^{-1})$ (σ, k)-periodic time support sets for each node in $\mathcal{G}$.

Based on Lemma 3.3, we have the following results.

Corollary 3.1: The time and space complexity of Algorithm 2 for computing $\widetilde{\mathsf{PT}}_u$ is $O(|\mathcal{T}|d_u(G) + |\mathcal{T}|^2\sigma^{-1})$ and $O(|\mathcal{T}|^2)$ respectively.

Theorem 3.2: The time and space complexity of Algorithm 1 is $O(m|\mathcal{T}|^2\sigma^{-1})$ and $O(m + n + |\mathcal{T}|^2)$ respectively.

Note that $|\mathcal{T}|$ (the number of snapshots) is typically not very large in practical temporal graphs. For example, in DBLP temporal network, there are at most 60 snapshots if we extract a snapshot by year (each snapshot represents a co-authorship network in one year). Hence, the worst-case time complexity of our algorithm is near linear w.r.t. the size of the temporal graph. Moreover, the practical performance of Algorithm 1 should be much better than the worst-case time complexity. This is because Algorithm 1 is integrated with a degree pruning rule (see lines 12-13 in Algorithm 1), which significantly decreases the number of calls of the ComputePeriod procedure. In addition, the ComputePeriod procedure can early terminate once the algorithm find a valid (σ, k)-periodic time support set, which can further reduce the time cost of Algorithm 1.

An improved WPCore **algorithm.** Although Algorithm 1 is efficient in practice, it still has two limitations. First, Algorithm 1 needs to invoke Algorithm 2 to compute $\widetilde{\mathsf{PT}}_u$ for every node $u \in V_c$ (line 6), which is very costly for high-degree nodes. Second, when deleting a node u, Algorithm 1 has to call Algorithm 2 to re-compute $\widetilde{\mathsf{PT}}_w$ for each neighbor node w of u (see line 15 in Algorithm 1), which clearly results in significant amounts of redundant computations.

To overcome these limitations, we propose an improved algorithm called WPCore+. The striking features of WPCore+ are twofold. On the one hand, WPCore+ does not compute $\widetilde{\mathsf{PT}}_u$ for every node u in advance. Instead, it calculates $\widetilde{\mathsf{PT}}_u$ for the node u *on-demand*. WPCore+ processes the nodes based on an increasing order by their degrees. Specifically, the algorithm first explores the low-degree nodes and applies the degree pruning rule to delete nodes. This is because the low-degree nodes are more likely to be deleted by the degree pruning rule. Moreover, compared to the high-degree nodes, the time costs for computing $\widetilde{\mathsf{PT}}_u$ for low-degree nodes are much cheaper. If a node u cannot be removed by the degree pruning rule, the WPCore+ algorithm invokes Algorithm 2 to compute $\widetilde{\mathsf{PT}}_u$ on-demand. Note that based on this on-demand computing paradigm, we can substantially reduce the computational costs for the high-degree nodes. The reason is as follows. When processing a high-degree node u, many low-degree neighbors of u may have already been pruned which will significantly decrease the degree of u, thus

Algorithm 3: WPCore+ $(\mathcal{G}, \sigma, k)$

Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parameters σ, and k
Output: The WPCore V_w

1 Let $G = (V, E)$ be the de-temporal graph of $\mathcal{G}$;
2 Let $G_c = (V_c, E_c)$ be the k-core of G;
3 $\mathcal{Q} \leftarrow \emptyset$; $D \leftarrow \emptyset$;
4 Let $d_u(G_c)$ be the degree of u in G_c;
5 **for** $u \in V_c$ *in an increasing order by* $d_u(G_c)$ **do**
6 **if** $u \in D$ **then continue**;
7 $\widetilde{\mathsf{PT}}_u \leftarrow$ ComputePeriod $(u, \mathcal{G}, \sigma, k, V_c \setminus D)$;
8 **if** $\widetilde{\mathsf{PT}}_u = \emptyset$ **then** $\mathcal{Q}.push(u)$;
9 $\widetilde{\mathsf{IPT}}_u \leftarrow$ InvertIndex $(\widetilde{\mathsf{PT}}_u)$;
10 **while** $\mathcal{Q} \neq \emptyset$ **do**
11 $v \leftarrow \mathcal{Q}.pop()$; $D \leftarrow D \cup \{v\}$;
12 **for** $w \in N_v(G_c)$ **do**
13 **if** $d_w(G_c) \geq k$ **then**
14 $d_w(G_c) \leftarrow d_w(G_c) - 1$;
15 **if** $d_w(G_c) < k$ **then** $\mathcal{Q}.push(w)$; **continue**;
16 **if** $\widetilde{\mathsf{PT}}_w$ *has already been computed* **then**
17 UpdatePeriod $(\widetilde{\mathsf{PT}}_w, \widetilde{\mathsf{IPT}}_w, v, k)$;
18 **if** $\widetilde{\mathsf{PT}}_w = \emptyset$ **then** $\mathcal{Q}.push(w)$;
19 **return** $V_w \leftarrow (V_c \setminus D)$;
20 **Procedure** InvertIndex $(\widetilde{\mathsf{PT}}_u)$
21 $\widetilde{\mathsf{IPT}}_u \leftarrow \emptyset$; $L \leftarrow \emptyset$; $h \leftarrow 1$;
22 Let $\widetilde{\mathsf{PT}}_u(j) \leftarrow [s, i, \sigma, ArrD]$ be the j-th element in $\widetilde{\mathsf{PT}}_u$;
23 **for** $j \leftarrow 1 : |\widetilde{\mathsf{PT}}_u|$ **do**
24 **for** $t \leftarrow 0 : (\sigma - 1)$ **do**
25 $L(h) \leftarrow [\widetilde{\mathsf{PT}}_u(j).s + t \times i,\ j]$; $h \leftarrow h + 1$;
26 **for** $h \leftarrow 1 : |L|$ **do**
27 $[t,\ j] \leftarrow L(h)$; $\widetilde{\mathsf{IPT}}_u(t).push(j)$;
28 **return** $\widetilde{\mathsf{IPT}}_u$;
29 **Procedure** UpdatePeriod $(\widetilde{\mathsf{PT}}_w, \widetilde{\mathsf{IPT}}_w, v, k)$
30 **for** *each temporal edge* $(w, v, t) \in \mathcal{E}$ **do**
31 $\mathsf{PTS}(t) \leftarrow \widetilde{\mathsf{IPT}}_w(t)$;
32 **while** $\mathsf{PTS}(t) \neq \emptyset$ **do**
33 $j \leftarrow \mathsf{PTS}(t).pop()$;
34 $\widetilde{\mathsf{PT}}_w(j).ArrD[t] \leftarrow \widetilde{\mathsf{PT}}_w(j).ArrD[t] - 1$;
35 **if** $\widetilde{\mathsf{PT}}_w(j).ArrD[t] < k$ **then**
36 $\widetilde{\mathsf{PT}}_w \leftarrow \widetilde{\mathsf{PT}}_w \setminus \{\widetilde{\mathsf{PT}}_w(j)\}$;

reducing the cost for computing $\widetilde{\mathsf{PT}}_u$. On the other hand, when deleting a node u, WPCore+ does not re-compute $\widetilde{\mathsf{PT}}_w$ for a neighbor node w of u. Instead, WPCore+ dynamically updates the computed $\widetilde{\mathsf{PT}}_w$ for each node w, thus substantially avoiding redundant computations. The detailed description of WPCore+ is shown in Algorithm 3.

Algorithm 3 first computes the KCore $G_c = (V_c, E_c)$ in the de-temporal graph (line 2), and then explores the nodes in V_c based on an increasing order by the degrees in G_c (line 5). When processing a node u, the algorithm first checks whether u has been deleted or not (line 6). If u has not been removed, WPCore+ invokes Algorithm 2 to compute $\widetilde{\mathsf{PT}}_u$ (line 7). If $\widetilde{\mathsf{PT}}_u$ is an empty set, u is not a (σ, k)-periodic node. Thus, the algorithm pushes it into the queue $\mathcal{Q}$ (line 8). Subsequently, the algorithm iteratively deletes the nodes in $\mathcal{Q}$ (lines 10-18). When removing a node v, WPCore+ explores all v's neighbors (line 12). For a neighbor node w, WPCore+ first updates the degree of w (line 14), i.e., $d_w(G_c)$. If the updated degree is less than k, u is not a (σ, k)-periodic node (line 15). In this case, the algorithm pushes it into $\mathcal{Q}$ and continues to process the next node in $\mathcal{Q}$ (the degree pruning rule). Otherwise, if $\widetilde{\mathsf{PT}}_w$ has already been computed, the algorithm invokes UpdatePeriod to update $\widetilde{\mathsf{PT}}_w$ (line 17). If the updated $\widetilde{\mathsf{PT}}_w$ becomes empty, w is not a (σ, k)-periodic node and the algorithm pushes w into $\mathcal{Q}$ (line 18). Note that if $\widetilde{\mathsf{PT}}_w$ has not been computed yet, the algorithm does not need to update $\widetilde{\mathsf{PT}}_w$. In this case, $\widetilde{\mathsf{PT}}_w$ will be calculated in the next iterations (see line 7).

To efficiently implement the UpdatePeriod procedure, we develop an inverted index structure called $\widetilde{\mathsf{IPT}}_u$ to organize all (σ, k)-periodic time support sets maintained in $\widetilde{\mathsf{PT}}_u$. Specifically, for the j-th arithmetic sequence (corresponding to a (σ, k)-periodic time support set) $\{t_{j_i}, t_{j_i+p}, \cdots, t_{j_i+(\sigma-1)\times p}\}$ in $\widetilde{\mathsf{PT}}_u$, we insert an element j into $\widetilde{\mathsf{IPT}}_u(t_{j_i+h\times p})$ for each $0 \leq h \leq \sigma - 1$, i.e., $\widetilde{\mathsf{IPT}}_u(t_{j_i+h\times p}).push(j)$. Based on $\widetilde{\mathsf{PT}}_u$, we can easily construct the inverted index $\widetilde{\mathsf{IPT}}_u$ by invoking the InvertIndex procedure (lines 20-28). Note that by our construction, $\widetilde{\mathsf{IPT}}_u(t)$ keeps all arithmetic sequences that contain the timestamp t. Therefore, once we have an invert index $\widetilde{\mathsf{IPT}}_u$, we can quickly retrieve the arithmetic sequences containing t.

The UpdatePeriod procedure explores all the temporal edges (w, v, t) to update $\widetilde{\mathsf{PT}}_w$ after deleting v (line 30). For each (w, v, t), the algorithm identifies all the arithmetic sequences (the elements in $\widetilde{\mathsf{PT}}_w$) that contain the timestamp t based on the inverted index structure (lines 31-33). For each arithmetic sequence, the algorithm decreases the degree of w at timestamp t by 1 (line 34). If the updated degree is smaller than k, the algorithm deletes the arithmetic sequence from $\widetilde{\mathsf{PT}}_w$ (lines 35-36), because it is no longer a valid (σ, k)-periodic time support set. Since our algorithm correctly computes and maintains $\widetilde{\mathsf{PT}}_w$ for every node w, the correctness of Algorithm 3 can be guaranteed. Below, we analyze the time and space complexity of Algorithm 3.

Theorem 3.3: The time and space complexity of Algorithm 3 is $O(\alpha m + n(\alpha\sigma + \mathcal{T}^2\sigma^{-1})$ and $O(m + n\alpha\sigma)$ respectively, where $\alpha = \max_{u \in V_c}\{|\widetilde{\mathsf{PT}}_u|\}$.

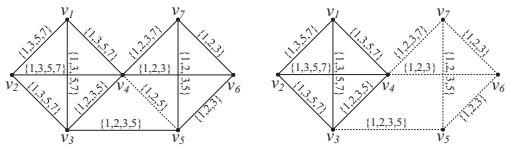
Note that the time complexity of Algorithm 3 is lower than that of Algorithm 1, as α is smaller than $\mathcal{T}^2\sigma^{-1}$. In practice, the space usage of Algorithm 3 is much smaller than the worst-case bound, because our algorithm only work on the k-core subgraph which is typically significantly smaller than the original temporal graph.

B. The SPCore *pruning rule*

Although WPCore can prune many unpromising nodes, it is not very effective for pruning unpromising edges. For example, in Fig. 2, the edge (v_4, v_5) is clearly not a *σ-periodic edge* with $\sigma = 3$, as the timestamps associated with this edge cannot form an 3-term arithmetic sequence. As a result, such an edge cannot be contained in any σ-periodic k-clique. To overcome this defect, we propose a novel σ-periodic strong k-core (abbreviated as SPCore) pruning technique which combines both σ-periodic nodes and edges for pruning. Below, we give a definition of σ-periodic edge.

Definition 10 (σ-periodic edge): Given a temporal graph $\mathcal{G}$, its de-temporal graph G and parameter σ, an edge $(u, v) \in G$ is called a σ-periodic edge if there exists a σ-periodic time support set for the subgraph $\{(u, v)\}$.

It is easy to see that a σ-periodic edge is also a special σ-periodic subgraph, because we can treat an edge (u, v) as a special subgraph. Therefore, every σ-periodic edge also have a set of σ-periodic time support sets. For convenience, we let



(a) σ-periodic-link graph (b) Subgraph reduced by SPCore

Fig. 4. Illustration of the SPCore pruning ($\sigma = 3, k = 3$)

$\mathsf{EPT}(u, v)$ be the set of all those σ-periodic time support sets for a σ-periodic edge (u, v). Clearly, an edge is a σ-periodic edge if and only if $\mathsf{EPT}(u, v) \neq \emptyset$. Note that to determine whether an edge (u, v) is a σ-periodic edge, we can make use of a similar algorithm as shown in Algorithm 2 to compute $\mathsf{EPT}(u, v)$, which takes $O(|\mathcal{T}|^2\sigma^{-1})$ time in the worst case. Based on Definition 10, we define the σ-periodic-link graph in the following.

Definition 11: A subgraph $\tilde{G}_c = (\tilde{V}_c, \tilde{E}_c)$ of the de-temporal graph G is called a σ-periodic-link graph if every edge $(u, v) \in \tilde{E}_c$ is a σ-periodic edge.

By Definition 11, we can obtain the maximum σ-periodic-link graph by removing all the *non-periodic* edges from G (i.e., only retain all the σ-periodic edges in G). The following example illustrates the above definitions.

Example 4: Reconsider the temporal graph $\mathcal{G}$ shown in Fig. 2. Suppose that $\sigma = 3, k = 3$. Then, we can see that the edge (v_4, v_5) has three timestamps $\{1, 2, 5\}$ which clearly cannot form a 3-term arithmetic sequence. Therefore, we have $\mathsf{EPT}(v_4, v_5) = \emptyset$, indicating that (v_4, v_5) is not a σ-periodic edge. We can easily derive that all the other edges in the de-temporal graph G (except (v_4, v_5)) are σ-periodic edges. Hence, the maximum σ-periodic-link graph is a subgraph by removing edge (v_4, v_5) in G which is shown in Fig. 4(a). ■

Based on the maximum σ-periodic-link graph, we define the σ-periodic strong k-core (SPCore) as follows.

Definition 12 (σ-periodic strong k-core): A subgraph S of the maximum σ-periodic-link graph $\hat{G}_c = (\hat{V}_c, \hat{E}_c)$ is called a σ-periodic strong k-core if it satisfies the following constraints.
(1) ***Periodic edge constraint:*** for any edge (u, v) in S, $\mathsf{PT}(u) \cap \mathsf{PT}(v) \cap \mathsf{EPT}(u, v) \neq \emptyset$;
(2) ***Maximal constraint:*** there does not exist a subgraph S' of $\hat{G}_c$ that satisfies (1) and $S \subset S'$.

Based on Definition 12, we are able to derive several useful properties for the σ-periodic strong k-core.

Lemma 3.4: Any edge in the MPClique must be contained in the SPCore.

Lemma 3.5: Given a temporal graph $\mathcal{G}$, its de-temporal graph G, parameters σ and k, the SPCore is unique in G if it exists.

Lemma 3.6: Let G be the de-temporal graph, G_w be the subgraph induced by the WPCore, and G_s is the SPCore. Then, we have $G_s \subseteq G_w$.

Based on Lemmas 3.4 and 3.5, we know that every MPClique are contained in the unique SPCore of G. Therefore, we can first compute the SPCore to prune unpromising nodes and edges, and then enumerate all MPCliques on the reduced graph. As shown in Lemma 3.6, such a SPCore pruning technique is more powerful than the WPCore pruning technique, since it may prune more edges and nodes of the original temporal graph. Below, we develop an algorithm to efficiently compute the SPCore.

The basic idea of our SPCore algorithm is that we first compute the subgraph induced by the WPCore, denoted by G_w. Then, we identify all the edges in G_w that do not satisfy the periodic edge constraint in Definition 12 (i.e., find the edge (u, v) meeting $\mathsf{PT}(u) \cap \mathsf{PT}(v) \cap \mathsf{EPT}(u, v) = \emptyset$). After that, we delete all those unpromising edges from G_w. Note that the deletion of an edge (u, v) may trigger u and v's outgoing edges that violate the periodic edge constraint. Therefore, we need to iteratively perform this edge peeling procedure, until no edge can be removed. The detailed description of our algorithm is shown in Algorithm 4.

Algorithm 4: SPCore $(\mathcal{G}, \sigma, k)$

Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parameters σ, and k
Output: The SPCore $G_s = (V_s, E_s)$

```
1  V_w ← WPCore+ (G, σ, k);
2  PT_u and IPT_u have already been computed in WPCore+ for each u ∈ V_w;
3  Let G = (V, E) be the de-temporal graph of G;
4  Let G_w = (V_w, E_w) be the subgraph induced by V_w in G;
5  EQ ← ∅; ED ← ∅;
6  for each (u, v) ∈ E_w do
7      Compute EPT_uv;
8      if PT_u ∩ PT_v ∩ EPT_uv = ∅ then
9          EQ.push((u, v));
10 while EQ ≠ ∅ do
11     (u, v) ← EQ.pop(); ED ← ED ∪ {(u, v)};
12     UpdatePeriod (PT_u, IPT_u, v, k);
13     for x ∈ N_u(G_w) do
14         if (u, x) ∉ EQ and (u, x) ∉ ED then
15             if PT_u ∩ PT_x ∩ EPT_ux = ∅ then EQ.push((u, x));
16     UpdatePeriod (PT_v, IPT_v, u, k);
17     for x ∈ N_v(G_w) do
18         if (v, x) ∉ EQ and (v, x) ∉ ED then
19             if PT_v ∩ PT_x ∩ EPT_vx = ∅ then EQ.push((v, x));
20 return G_s ← the subgraph comprises all edges in E_w \ ED ;
```

In line 1, the algorithm first invokes Algorithm 3 to calculate the WPCore V_w. Note that by Algorithm 3, we are able to obtain $\widetilde{\mathsf{PT}}_u$ and $\widetilde{\mathsf{IPT}}_u$ for each $u \in V_w$ (line 2). Also, we can easily get the subgraph $G_w = (V_w, E_w)$ induced by V_w. The algorithm uses a queue EQ and a set ED to maintain all the unpromising edges (line 5). For each $(u, v) \in E_w$, the algorithm computes the set $\mathsf{EPT}(u, v)$ (lines 6-7), which is denoted by EPT_{uv} in Algorithm 4. Then, if (u, v) violates the periodic edge constraint ($\mathsf{PT}(u) \cap \mathsf{PT}(v) \cap \mathsf{EPT}(u, v) = \emptyset$)), the algorithm pushes it into the queue EQ (lines 8-9). Subsequently, the algorithm iteratively deletes the element in EQ (lines 10-19). For each $(u, v) \in EQ$, the algorithm needs to update $\widetilde{\mathsf{PT}}_u$ and $\widetilde{\mathsf{PT}}_v$ by invoking the UpdatePeriod procedure (lines 12 and 16). This is because the deletion of an edge (u, v) decreases the degrees of both u and v by 1 which may further result in the updating of $\widetilde{\mathsf{PT}}_u$ and $\widetilde{\mathsf{PT}}_v$. Since $\widetilde{\mathsf{PT}}_u$ (or $\widetilde{\mathsf{PT}}_v$) may update, the algorithm has to verify each edge (u, x) (or edge (v, x)) for $x \in N_u(G_w)$ whether it satisfies the periodic edge constraint or not (lines 13-15 and lines 17-19). If the edge (u, x) (or edge (v, x)) does not satisfy the periodic edge constraint, the algorithm pushes it into EQ (lines 15 and 19). The algorithm terminates when $EQ = \emptyset$. At this moment, the subgraph comprises all the remaining edges is a SPCore. Since all the edges that violate the periodic edge constraint are deleted and every remaining edge meets the periodic edge constraint, Algorithm 4 can correctly compute the SPCore. The following example illustrates how Algorithm 4 works.

Example 5: Reconsider the temporal graph shown in Fig. 2. Suppose that $\sigma = 3, k = 3$. First, by computing the WPCore, the algorithm can obtain an induced subgraph $G_w = (V_w, E_w)$

where $V_w = \{v_1, \cdots, v_7\}$. Then, the algorithm calculates $\mathsf{EPT}(u,v)$ for each $(u,v) \in E_w$ (lines 6-7). Clearly, we have $\mathsf{EPT}(v_4,v_5) = \emptyset$, thus the algorithm pushes (v_4,v_5) into EQ (lines 8-9). Also, the algorithm pushes (v_3,v_5) into EQ. The reason is that $\widetilde{\mathsf{PT}}_{v_3} = \{[1,3,5]\}$, $\widetilde{\mathsf{PT}}_{v_5} = \{[1,2,3]\}$, and $\mathsf{EPT}(v_3,v_5) = \{[1,2,3]\}$, and thus $\widetilde{\mathsf{PT}}_{v_3} \cap \widetilde{\mathsf{PT}}_{v_5} \cap \mathsf{EPT}(v_3,v_5) = \emptyset$ (lines 8-9). Subsequently, the algorithm pops (v_4,v_5) from EQ and updates $\widetilde{\mathsf{PT}}_{v_4}$ and $\widetilde{\mathsf{PT}}_{v_5}$. Since $\widetilde{\mathsf{PT}}_{v_4}$ and $\widetilde{\mathsf{PT}}_{v_5}$ do not change after deleting (v_4,v_5), the algorithm continues to pop (v_3,v_5) from EQ. After removing (v_3,v_5), $\widetilde{\mathsf{PT}}_{v_5}$ is updated to be an empty set. Thus, the algorithm will pushes (v_5,v_6) and (v_5,v_7) into EQ, and then iteratively processes these two edges. When the algorithm terminates, we can obtain the SPCore as shown in Fig. 4(b) (the subgraph induced by the nodes $\{v_1,\cdots,v_4\}$). Compared to the WPCore pruning, the SPCore pruning can prune many additional nodes and edges, indicating that the SPCore pruning is indeed much more powerful than the WPCore pruning. ■

Below, we analyze the complexity of Algorithm 4.

Theorem 3.4: The time and space complexity of Algorithm 4 is $O(m|\mathcal{T}|^2\sigma^{-1})$ and $O(m+n+|\mathcal{T}|^2)$ respectively.

Note that since our algorithm only works on the WPCore (not the original temporal graph), the time cost of Algorithm 4 is much less than the worst case bound in practice, which is also confirmed in our experiments.

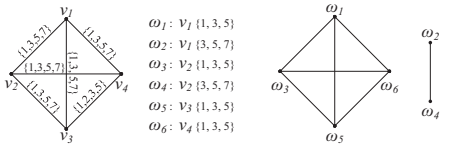
IV. Enumeration of MPCliques

Recall that the MPClique enumeration problem is NP-hard. Thus, there does not exist a polynomial-time algorithm to solve our problem unless P=NP. Moreover, most existing maximal clique enumeration algorithms (e.g., the classic Bron-Kerbosch algorithm [10]) can only work on static graphs, it is not clear how to apply them to identify periodic cliques in temporal graphs. To circumvent this problem, we propose a new Bron-Kerbosch style enumeration algorithm, called MPC, which can efficiently compute the complete set of all MPCliques. The MPC algorithm first invokes Algorithm 4 to significantly reduce the size of the original temporal graph. Then, the MPC algorithm transforms the reduced temporal graph into a special static graph $\tilde{G}$, and performs a Bron-Kerbosch style enumeration algorithm to find all maximal cliques on $\tilde{G}$. We show that the complete set of all maximal cliques in $\tilde{G}$ is a complete set of all MPCliques in the original temporal graph $\mathcal{G}$. The detail of the MPC algorithm is shown as follows.

A. The MPC algorithm

Note that the key step in MPC is to construct the transformed graph $\tilde{G}$. Below, we present our graph transformation approach.

Constructing the transformed graph. Recall that in the reduced graph $G_s = (V_s, E_s)$, each node u has a set of (σ,k)-periodic time support sets, i.e., $\mathsf{PT}(u)$, and each edge (u,v) also has a set of σ-periodic time support sets, i.e., $\mathsf{EPT}(u,v)$. Since every node u and every edge (u,v) in a MPClique C shares at least one periodic time support set, we can decompose a MPClique into a set of nodes and edges which are associated with the same periodic time support sets. This motivate us to construct a graph $\tilde{G} = (\tilde{V}, \tilde{E})$ as follows. For each node $v \in V_s$ and an element PT^s_v in $\mathsf{PT}(v)$, we construct a vertex (v, PT^s_v) for $\tilde{V}$. As a result, for each node $v \in V_s$, we can obtain $|\mathsf{PT}(v)|$ vertices in $\tilde{V}$. For any



(a) Transformed nodes (b) Transformed graph $\tilde{G}$

Fig. 5. Illustration of the graph transformation method

two vertices (u, PT^s_u) and (v, PT^s_v) in $\tilde{V}$, we create an edge $(u, v, \mathsf{EPT}^s_{uv})$ if and only if $\mathsf{EPT}^s_{uv} = \mathsf{PT}^s_u = \mathsf{PT}^s_v$ (i.e., the same arithmetic sequence), where EPT^s_{uv} is an element in $\mathsf{EPT}(u,v)$. This is because for any edge (u,v) in a MPClique, the nodes u, v and the edge (u,v) shares the same periodic time support set. Clearly, by this construction, each vertex in the transformed graph is a two-tuple (a node and a periodic time support set), and each edge is a three-tuple (an edge and a periodic time support set). The following example illustrates our graph transformation method.

Example 6: Consider the temporal graph shown in Fig. 2. Suppose that $\sigma = 3, k = 3$. Then, the reduced graph by SPCore is shown in Fig. 4(b). Based on the reduced graph, we can obtain the transformed graph $\tilde{G}$ shown in Fig. 5. Specifically, Fig. 5(a) depicts the reduced graph and the transformed vertices. For example, for the node v_1, we have $\mathsf{PT}(v_1) = \{[1,3,5],[3,5,7]\}$. Therefore, we construct two vertices $\omega_1 = (v_1,[1,3,5])$ and $\omega_2 = (3,5,7)$ in $\tilde{G}$. Similarly, we can obtain four other vertices in $\tilde{G}$ which are $\omega_3, \cdots, \omega_6$ as shown in Fig. 5(a). Since the nodes v_1, v_2, and edge (v_1,v_2) are associated with the same periodic time support sets $[1,3,5]$ and $[3,5,7]$, we can obtain two edges (ω_1,ω_3) and (ω_2,ω_4) in the transformed graph $\tilde{G}$. Likewise, we can get all the other edges in $\tilde{G}$. The final transformed graph $\tilde{G}$ is shown in Fig. 5(b) which contains 6 nodes and 7 edges. ■

Let $\tilde{C} = (\tilde{V}_c, \tilde{E}_c)$ be a maximal clique in the transformed graph $\tilde{G}$. By our graph transformation method, the first term of a vertex $\omega = (v, \mathsf{PT}^s_v) \in \tilde{V}_c$ is a node v in the reduced graph G_s. For a maximal clique $\tilde{C}$ in the transformed graph $\tilde{G}$, we let V_c be the set of nodes of $\tilde{C}$ in the reduced graph G_s, i.e., $V_c \triangleq \{v | (v, \mathsf{PT}^s_v) \in \tilde{V}_c\}$. Note that there may exist two maximal cliques $\tilde{C}_1$ and $\tilde{C}_2$ in $\tilde{G}$ having the same node set V_c. For example, suppose that we have a maximal periodic clique induced by the nodes $\{v_1,v_2,v_3\}$ in $\mathcal{G}$ with two periodic time support sets $[1,3,5]$ and $[3,5,7]$. Then, such a periodic clique will be transformed to two maximal cliques $\{(v_1,[1,3,5]),(v_2,[1,3,5]),(v_3,[1,3,5])\}$ and $\{(v_1,[3,5,7]),(v_2,[3,5,7]),(v_3,[3,5,7])\}$ in $\tilde{G}$. By the graph transformation approach, we can derive the following result.

Lemma 4.1: For any MPClique $C' = (V'_c, E'_c)$ in the reduced graph G_s, there exists a maximal clique $\tilde{C} = (\tilde{V}_c, \tilde{E}_c)$ in the transformed graph $\tilde{G}$ such that the node set V_c of $\tilde{C}$ is equal to V'_c. For any maximal clique $\tilde{C} = (\tilde{V}_c, \tilde{E}_c)$ with node set V_c in $\tilde{G}$, the subgraph induced by V_c is a MPClique C in G_s.

Based on Lemma 4.1, we are able to obtain the complete set of MPCliques by enumerating all maximal cliques in $\tilde{G}$. Since $\tilde{G}$ is a static graph, we make use of a Bron-Kerbosch style algorithm to identify all maximal cliques in $\tilde{G}$. The detailed description of our algorithm is shown in Algorithm 5.

In line 2, the algorithm invokes the SPCore pruning technique (Algorithm 4) to prune the temporal graph. Note that in this pruning procedure, we can also obtain $\widetilde{\mathsf{PT}}_u$ for each

Algorithm 5: MPC $(\mathcal{G}, \sigma, k)$

Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parameters σ and k
Output: the set of MPCliques $\mathcal{C}$

1 $\mathcal{C} \leftarrow \emptyset$;
2 $G_s = (V_s, E_s) \leftarrow$ SPCore$(\mathcal{G}, \sigma, k)$;
3 $\widetilde{\mathsf{PT}}_u$ has been computed in SPCore for each $u \in V_s$;
4 EPT_{uv} has been calculated in SPCore for each $(u, v) \in E_s$;
5 $\tilde{G} = (\tilde{V}, \tilde{E}) \leftarrow$ construct the transformed graph based on $\widetilde{\mathsf{PT}}_u$ and EPT_{uv};
6 EnumMPClique $(\tilde{V}, \emptyset, \emptyset, k)$;
7 **Procedure** EnumMPClique $(\tilde{P}, \tilde{R}, \tilde{X}, k)$
8 **if** $|\tilde{P}| + |\tilde{R}| < k$ **then return**;
9 **if** $\tilde{P} \cup \tilde{X} = \emptyset$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{\tilde{R}\}$;
10 $(v', \widetilde{\mathsf{PT}}^s_{v'}) \leftarrow \arg \max_{(v, \widetilde{\mathsf{PT}}^s_v) \in \tilde{P} \cup \tilde{X}} |\tilde{P} \cap N_{(v, \widetilde{\mathsf{PT}}^s_v)}(\tilde{G})|$;
11 **for** $(v, \widetilde{\mathsf{PT}}^s_v) \in \tilde{P} \setminus N_{(v', \widetilde{\mathsf{PT}}^s_{v'})}(\tilde{G})$ **do**
12 $\quad \tilde{R}' \leftarrow \tilde{R}' \cup (v, \widetilde{\mathsf{PT}}^s_v)$;
13 $\quad \tilde{P}' \leftarrow \tilde{P} \cap N_{(v, \widetilde{\mathsf{PT}}^s_v)}(\tilde{G})$; $\tilde{X}' \leftarrow \tilde{X} \cap N_{(v, \widetilde{\mathsf{PT}}^s_v)}(\tilde{G})$;
14 $\quad$ EnumMPClique $(\tilde{P}', \tilde{R}', \tilde{X}', k)$;
15 $\quad \tilde{P} \leftarrow \tilde{P} \setminus (v, \widetilde{\mathsf{PT}}^s_v)$; $\tilde{X} \leftarrow \tilde{X} \cup (v, \widetilde{\mathsf{PT}}^s_v)$;

$u \in V_s$ and EPT_{uv} for each $(u, v) \in E_s$ (lines 3-4). Based on $\widetilde{\mathsf{PT}}_u$ and EPT_{uv}, the algorithm can construct the transformed graph $\tilde{G}$ (line 5). Then, the algorithm performs a Bron-Kerbosch algorithm with pivoting technique to identify all maximal cliques in $\tilde{G}$ (line 6). Specifically, the set $\tilde{R}$ denotes the current clique, $\tilde{P}$ denotes the set of candidate vertices, and $\tilde{X}$ denotes the set of vertices that have already been processed. Note that each vertex in $\tilde{P}$, $\tilde{R}$, and $\tilde{X}$ is a two-tuple $(v, \widetilde{\mathsf{PT}}^s_v)$. In line 10, the algorithm adopts a similar pivoting technique developed in [12] to speed up the enumeration procedure. Note that the operator $N_{(v, \widetilde{\mathsf{PT}}^s_v)}(\tilde{G})$ is to take the neighbors of the vertex $(v, \widetilde{\mathsf{PT}}^s_v)$ in the transformed graph $\tilde{G}$. The correctness of Algorithm 5 can be guaranteed by [12] and Lemma 4.1.

B. Number of MPCliques

In this subsection, we analyze the number of MPCliques in the temporal graph $\mathcal{G}$ based on a novel concept of σ-periodic degeneracy. The classic degeneracy is a well-known metric for measuring the sparsity of a static graph [8]. Many real-life networks are often very sparse, thus having a small degeneracy [8]. Below, we give the definition of degeneracy.

Definition 13 (Degeneracy): The degeneracy of a static graph G is the minimum integer δ such that each subgraph S of G contains a node v with degree no larger than δ.

Eppstein et al. [8] proved that the number of maximal cliques in a static graph is bounded by $(|V| - \delta)3^{\delta/3}$. They also developed an efficient maximal clique enumeration algorithm with time complexity $O(\delta|V|3^{\delta/3})$ based on the degeneracy ordering. The classic degeneracy, however, cannot be directly used to bound the number of MPCliques in temporal graphs. Below, we introduce a novel concept, called σ-periodic degeneracy, which will be applied to bound the number of MPCliques.

Definition 14 (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 $\hat{\delta}$.

Since the degeneracy-based bound for the number of maximal cliques is tailored for static graph [8], it is not clear how to use the σ-periodic degeneracy to bound the number of MPCliques in temporal graph. To circumvent this problem, we resort to bound the number of maximal cliques in the transformed graph $\tilde{G}$. The rationale is that the number of maximal cliques in $\tilde{G}$ is no less than the number of MPCliques in the temporal graph $\mathcal{G}$ by Lemma 4.1. Since the transformed graph $\tilde{G}$ is a static graph, we are capable of applying the results developed by Eppstein et al. [8] to bound the number of maximal cliques in $\tilde{G}$. Let $\tilde{\delta}$ be the degeneracy of the transformed graph $\tilde{G}$. Then, the following lemma shows that $\tilde{\delta}$ is bounded by $\hat{\delta}$.

Lemma 4.2: For any temporal graph $\mathcal{G}$ and the transformed graph $\tilde{G}$ of the SPCore of $\mathcal{G}$, we have $\tilde{\delta} \leq \hat{\delta}$.

Based on Lemma 4.2, we can leverage $\hat{\delta}$ to bound the number of MPCliques in $\mathcal{G}$ as shown in the following theorem.

Theorem 4.1: Given a temporal graph $\mathcal{G}$, parameters σ and k, the number of maximal σ-periodic k-cliques (MPCliques) in $\mathcal{G}$ is less than $(4m^2k^{-2}\sigma^{-1} - \hat{\delta})3^{\hat{\delta}/3}$.

Based on the results developed by Eppstein et al. [8], we can also bound the worst-case time complexity of the MPClique enumeration problem by the σ-periodic degeneracy of $\mathcal{G}$, i.e., $\hat{\delta}$. Specifically, we have the following results.

Theorem 4.2: Given a temporal graph $\mathcal{G}$, parameters σ and k, there exists an algorithm to enumerate all MPCliques in $\mathcal{G}$ in $O(\hat{\delta}m^2k^{-2}\sigma^{-1}3^{\hat{\delta}/3})$ time, where $\hat{\delta}$ is the σ-periodic degeneracy of $\mathcal{G}$ and m is the number of temporal edges in $\mathcal{G}$.

Not that Theorem 4.2 indicates that enumerating all MPCliques in a temporal graph $\mathcal{G}$ is *fixed-parameter tractable* with respect to the parameter σ-periodic degeneracy $\hat{\delta}$ of $\mathcal{G}$. Since the σ-periodic degeneracy of $\mathcal{G}$ is typically very small in real-life temporal graphs, the proposed algorithm can be very efficient in practice.

V. Experiments

In our experiments, we implement four various algorithms to identify maximal σ-periodic k-cliques: MPC-B, MPC-KC, MPC-WC+, MPC-SC. Specifically, MPC-B is a baseline algorithm which is not integrated with any graph reduction techniques. MPC-B first computes $\mathsf{PT}(u)$ (for each node u) and $\mathsf{EPT}(u, v)$ (for each edge (u, v)) using Algorithm 2, and then constructs a transformed graph $\tilde{G}$. After that, MPC-B uses the Bron-Kerbosch algorithm with a pivoting technique to enumerate all maximal σ-periodic k-cliques on $\tilde{G}$. MPC-KC is the MPC-B algorithm combined with the KCore pruning rule. MPC-WC+ denotes the MPC-B algorithm integrated with the WPCore pruning rule. Note that we also implement two algorithms which are WC (Algorithm 1) and WC+ to compute the WPCore. The MPC-WC+ algorithm is integrated with a more efficient WC+ algorithm. MPC-SC denotes the MPC-B algorithm with the SPCore pruning rule, i.e., Algorithm 5. To evaluate the effectiveness of the proposed maximal σ-periodic k-clique model, we use WPCore and SPCore as two intuitive baseline models. The reasons are as follows. First, to the best of our knowledge, there is no existing community model that can be used to model periodic communities in temporal networks. Second, by Definitions 9 and 12, both WPCore and SPCore can capture periodic and cohesive properties of a community in temporal graphs, thus WPCore and SPCore can serve as two baselines for modeling periodic communities. All algorithms are implemented in Python. All the experiments are conducted on a server of Linux kernel 4.4 with Intel Core(TM) i5-8400@3.20GHz and 32 GB main memory.

Datasets. We use five different types of real-life temporal networks in the experiments. The detailed statistics of our

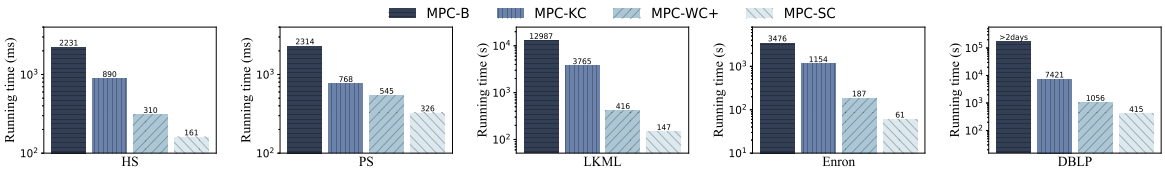


Fig. 7. Running time of different algorithms on various datasets ($\sigma = 4, k = 4$)

TABLE II
RUNNING TIME OF DIFFERENT ALGORITHMS WITH VARYING PARAMETERS (DBLP)

	$\sigma = 3$			$\sigma = 4$			$\sigma = 5$			$\sigma = 6$			$\sigma = 7$		
	MPC-KC	MPC-WC+	MPC-SC	MPC-KC	MPC-WC+	MPC-SC	MPC-KC	MPC-WC+	MPC-SC	MPC-KC	MPC-WC+	MPC-SC	MPC-KC	MPC-WC+	MPC-SC
$k = 3$	INF	32,213	**4,339**	24,517	12,313	**1,237**	8,321	4,567	**936**	4,235	3,456	**804**	2,145	1,023	**144**
$k = 4$	23,100	3,574	**580**	7,421	1,056	**415**	3,441	774	**326**	1,960	114	**71**	1,467	48	**38**
$k = 5$	9,770	736	**280**	3,428	801	**75**	1,771	417	**45**	1,220	63	**35**	1,023	**33**	37
$k = 6$	4,464	621	**112**	2,035	585	**45**	1,643	142	**32**	980	32	**27**	576	**14**	16
$k = 7$	2,382	534	**24**	1,292	51	**23**	1,201	44	**27**	620	21	**20**	231	**10**	11

TABLE I
DATASETS

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert\mathcal{E}\rvert$	$d_{\max}$	$\lvert\mathcal{T}\rvert$	Time scale
HS	327	5,818	20,448	322	101	hour
PS	242	8,317	26,351	393	34	hour
LKML	26,885	159,996	328,092	14,172	96	month
Enron	86,978	297,456	499,983	4,311	48	month
DBLP	1,729,816	8,546,306	12,007,380	5,980	59	year

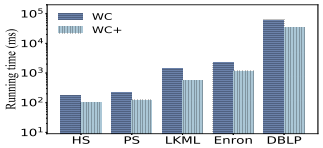


Fig. 6. Running time of WC and WC+

datasets are summarized in Table I. In Table I, the first two datasets are human contact temporal networks which are download from (http://www.sociopatterns.org/datasets/). Specifically, HS is a temporal network of face-to-face contacts between students in a French high school [2], and PS is a temporal network of contacts between the children and teachers in a French primary school [2]. Each snapshot of these temporal networks is extracted in a hour. Both LKML and Enron are temporal communication networks downloaded from (http://konect.uni-koblenz.de), where each temporal edge (u, v, t) represents an email communication from a user u to v at time t. Each snapshot of these temporal networks is extracted in a month. DBLP is a temporal collaboration network of authors in DBLP downloaded from (http://dblp.uni-trier.de/xml/), where each temporal edge (u, v, t) denotes that two authors u and v co-authored one paper at time t. Each snapshot of DBLP is extracted in a year. In Table I, $d_{\max}$ is the maximum number of temporal edges associated with a node, and $|\mathcal{T}|$ denotes the number of snapshots.

Parameter settings. There are two parameters k, σ in our algorithm. For the parameter k, we vary it from 3 to 7 with a default value of 3. We also vary σ from 3 to 7 with a default value of 3. Unless otherwise specified, the value of the other parameter are set to its default value when varying a parameter.

A. *Efficiency Testing*

Exp-1: Comparison between WC **and** WC+**.** Fig. 6 evaluates the running time of WC (Algorithm 1) and WC+ (Algorithm 3) for computing WPCore. As can be seen, WC+ is much faster than WC on all datasets. The running time of WC+ is around a half of the running time of WC. For example, on Enron, WC+ takes 1.1 seconds and WC consumes 2.3 seconds to identify all MPCliques. The reason is that WC+ is based on an on-demand computing paradigm which can substantially reduce redundant computations. These results are consistent with our theoretical analysis presented in Section III-A. In the following experiments, we will use WC+ to compute WPCore.

TABLE III
NUMBER OF NODES OF THE REDUCED GRAPH

	KCore		WPCore		SPCore	
HS	326	99%	280	86%	165	51%
PS	242	100%	233	96%	211	87%
LKML	9,773	36%	1,785	6.6%	926	3.4%
Enron	18,591	21%	3,314	3.8%	2,315	2.7%
DBLP	1,258,540	73%	126,357	7.3%	73,109	4.2%

Exp-2: Efficiency of various MPClique **mining algorithms.** Fig. 7 shows the running time of MPC-B, MPC-KC, MPC-WC+ and MPC-SC on different datasets with parameters $\sigma = 4$ and $k = 4$. Similar results can also be observed under the other parameter settings. From Fig. 7, we can see that MPC-SC is much faster than all the other competitors on all datasets. For example, on DBLP, MPC-SC takes around 7 minutes to enumerate all MPCliques which cuts the running time over MPC-WC+ and MPC-KC by 154% and 1,688% respectively. Note that MPC-B cannot get results on DBLP in 2 days. These results indicate that the SPCore pruning rule is indeed very powerful in practice which are consistent with our analysis in Section III-B.

Exp-3: Effect of parameters. Table. II reports the running time of different algorithms with varying parameters on DBLP. Similar results can also be observed on the other datasets. Since the parameters do not significantly affect MPC-B, we focus mainly on MPC-KC, MPC-WC+ and MPC-SC. As can be seen, MPC-SC is faster than all the other algorithms under almost all parameter settings. In general, the running time of MPC-KC, MPC-WC+ and MPC-SC decrease with increasing k and σ, because the size of the transformed graph decreases as k or σ increases. Note that when $\sigma = 7$ and $k \geq 5$, MPC-WC+ is slightly faster than MPC-SC. The reason could be that for a large σ and k, the original temporal graph can be reduced to a very small graph by WPCore, thus the benefit of SPCore may be not significant.

Exp-4: Number of nodes of the reduced graph. Table III shows the number of remaining nodes obtained by KCore, WPCore and SPCore on all datasets under the default parameter setting. In columns 2-4 of Table III, the left integer is the number of remaining nodes and the right value is the percentage of the remaining nodes over all nodes in the graph. As can be seen, both WPCore and SPCore can prune a large number of unpromising nodes on large datasets. For example, on DBLP, the number of remaining nodes obtained by WPCore and SPCore is only 7.3% and 4.2% of the original graph respectively. These results confirm that our graph reduction techniques are indeed very effective on large real-life temporal networks.

Exp-5: Size of the transformed graph. Table IV reports the size of the transformed graph $\tilde{G} = (\tilde{V}, \tilde{E})$ generated by MPC-SC. We can observe that the size of $\tilde{G}$ scales linearly

TABLE IV
THE SIZE OF THE TRANSFORMED GRAPH (MPC-SC)

	$\|\tilde{V}\|$	$\|\tilde{E}\|$	#. MPCliques	$\tilde{\delta}$
HS	1,946	3,388	57	12
PS	3,508	12,174	474	10
LKML	149,385	505,514	17,382	8
Enron	37,869	173,914	10,203	7
DBLP	353,557	1,028,598	45,442	12

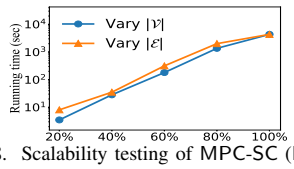
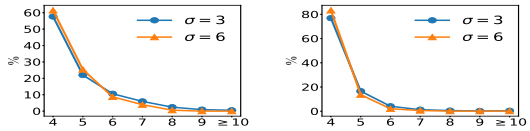


Fig. 8. Scalability testing of MPC-SC (DBLP)

TABLE V
MEMORY OVERHEAD OF MPC-SC

	Graph size	Memory (PT, EPT)	Memory (all)
HS	5.2MB	25.2MB	45MB
PS	2.8MB	15.8MB	35MB
LKML	20.1MB	35.4MB	101MB
Enron	53.3MB	98.6Mb	198MB
DBLP	678.5MB	2,398MB	3,234MB



(a) Enron (b) DBLP

Fig. 9. Distribution of the size of MPCliques ($k = 3$)

w.r.t. the original graph size. Moreover, the degeneracy $\tilde{\delta}$ of $\tilde{G}$ is very small in all datasets. The number of MPCliques is clearly less than $(4m^2k^{-2}\sigma^{-1} - \tilde{\delta})3^{\tilde{\delta}/3}$, which confirms our theoretical analysis in Section IV-B.

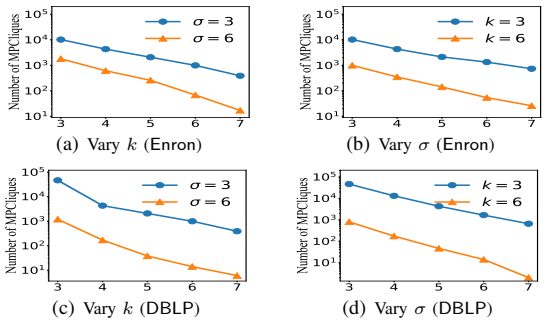
Exp-6: Scalability testing. Fig. 8 shows the scalability of MPC-SC on DBLP. Similar results can also be observed on the other datasets. We generate four temporal subgraphs by randomly picking 20%-80% of the nodes (temporal edges), and evaluate the running time of MPC-SC on those subgraphs. As can be seen, the running time increases smoothly with increasing $|\mathcal{V}|$ and $|\mathcal{E}|$. These results suggest that the MPC-SC algorithm is scalable when handling large temporal networks.

Exp-7: Memory overhead. Table V shows the memory usage of MPC-SC on different datasets. We can see that the memory usage of MPC-SC is higher than the size of the temporal graph, because MPC-SC needs to store $\mathsf{PT}(u)$ (for each node u) and EPT_{uv} (for each edge (u, v)). However, on large datasets, it is typically lower than five times of the size of the temporal graph. For instance, MPC-SC consumes 3,234MB memory on DBLP while the temporal graph uses 678.5MB memory. These results indicate that MPC-SC achieves near linear space complexity which confirms our theoretical analysis in Sections III-B and IV.

B. Effectiveness Testing

Exp-8: Distribution of the size of MPCliques. Fig. 9 shows the distribution of the size of MPCliques on Enron and DBLP with parameters $\sigma = 3$ (or $\sigma = 6$) and $k = 3$. Similar trends can also be observed on the other datasets and using other parameter settings. We can see that most MPCliques has a small size on both Enron and DBLP, and very few MPCliques have a size no less than 10. This is because a MPClique must satisfy the periodic clique constraint which may rule out large cliques.

Exp-9: Number of MPCliques with varying parameters. Fig. 10 shows the number of MPCliques with varying k or σ on Enron and DBLP. The results on the other datasets are consistent. As shown in Fig. 10(a) and Fig. 10(c), the number



(a) Vary k (Enron) (b) Vary σ (Enron)

(c) Vary k (DBLP) (d) Vary σ (DBLP)

Fig. 10. Number of MPCliques with varying parameters

(a) WPCore (b) SPCore (c) MPClique

Fig. 11. Case study on DBLP

of MPCliques drops sharply with an increasing k. Likewise, we can observe from Fig. 10(b) and Fig. 10(d) that the number of MPCliques decreases with a growing σ. The reason is that with a large k or σ, the periodic clique constraint will be strong, thus the number of MPCliques decreases. These results confirm our theoretical analysis in Theorem 4.2.

Exp-10: Case study on DBLP. We conduct a case study using DBLP to further evaluate the effectiveness of various models. Fig. 11 shows three communities of Prof. Michael Stonebraker obtained by WPCore, SPCore and MPClique respectively, using default parameters. As can be seen in Fig. 11(c), the community obtained by MPClique contains two cliques, and each clique comprises the close and long-term collaborators of Prof. Michael Stonebraker. Moreover, we find that each clique appears in 2015, 2016, and 2017 year, suggesting that there are two periodic communities containing Prof. Michael Stonebraker in recent years. From Figs. 11(a-b), we can see that the communities obtained by WPCore and SPCore not only contain two MPCliques in Fig. 11(c), but they also include some short-term collaborators of Prof. Michael Stonebraker who did not collaborate with him periodically, which indicates that both WPCore and SPCore models cannot fully capture the periodic patterns of a community. These results further confirm that MPClique is more effective than the baselines to detect periodic communities in temporal graphs.

VI. RELATED WORK

Temporal graph analysis. Our work is related to the studies on temporal graph analysis. Yang et al. [13] proposed an algorithm to detect frequent changing components in the temporal graph. Huang et al. [14] investigated the minimum spanning tree problem in temporal graphs. Gurukar et al. [15] presented a model to identify the recurring subgraphs that have similar sequence of information flow in temporal graphs. Wu et al. [16] proposed an efficient algorithm to answer the reachability and time-based path queries in temporal graphs. Yang et al. [3] studied a problem of finding a set of diversified quasi-cliques from a temporal graph. Wu et al. [7] proposed a temporal k-core model based on the counts of temporal edges. Ma et al. [4] investigated a dense subgraph problem in

temporal graphs. Li et al. [5] developed an efficient algorithm to identify persistent communities in temporal graphs. To the best of our knowledge, our work is the first to study the problem of mining periodic communities in temporal graphs.

Community detection in dynamic graphs. There is a number of studies for mining communities on dynamic networks [17]. Most of them aim to identify and analyze the community structures that evolve over time. Lin et al. [18] proposed a probabilistic generative model to analyze evolving communities. Chen et al. [19] developed an algorithm for tracking community dynamics. Agarwal et al. [20] studied how to find dense clusters for dynamic microblog streams. Li et al. [21] devised an algorithm to maintain the k-core in large dynamic graphs. Rossetti et al. [22] proposed an online iterative algorithm for tracking the evolution of communities. Unlike these studies, our work focuses mainly on detecting periodic communities in temporal graphs.

Maximal cliques enumeration. Our work is also related to the maximal clique enumeration problem. Notable algorithms for enumerating maximal clique include the classic Bron-Kerbosch algorithm [10] and its variants [8], [12], [23]. Tomita et al. [12] proved that the Bron-Kerbosch algorithm with a pivoting technique is essentially optimal according to the worst-case bound. Eppstein et al. [8] developed an algorithm which is fixed-parameter tractable w.r.t. the degeneracy of the graph. Cheng et al. [23] proposed an external-memory algorithm for clique enumeration in massive graphs. More recently, Himmel [24] developed a Bron-Kerbosch style algorithm for enumerating maximal cliques in temporal graph. Their work, however, cannot be used to enumerate periodic cliques.

Periodic behavior mining. The studies of periodic behavior mining are also related to our work. Notable examples are summarized below. Li et al. [25] addressed the problem of mining periodic behaviors for moving objects. Kurashima et al. [26] modeled the periodic actions in real-world (e.g., eating, sleep, and exercise) to make predictions for future actions. Radinsky et al. [27] also developed a temporal model to predict the periodic actions. Lahiri et al. [28] investigated a problem of mining periodic subgraphs in dynamic social networks. Their work, however, does not consider the communities in the periodic subgraphs, thus cannot be used for mining periodic communities.

VII. Conclusion

In this work, we study a problem of mining periodic communities in temporal graph. We propose a novel model, called maximal σ-periodic k-clique, to characterize the periodic communities in a temporal graph. To find all maximal σ-periodic k-cliques, we first develop several new pruning techniques to substantially reduce the size of the temporal graph. Then, on the reduced temporal graph, we propose an enumeration algorithm based on a carefully-designed graph transformation technique to efficiently identify all maximal σ-periodic k-cliques. Comprehensive experiments on five real-life temporal networks demonstrate the efficiency, scalability and effectiveness of our algorithms.

Acknowledgement. This work was partially supported by (i) NSFC Grants 61772346, 61732003, U1809206, 61572119, 61622202, U1401256, 61729201; (ii) National Key R&D Program of China 2018YFB1004402; (iii) Beijing Institute of Technology Research Fund Program for Young Scholars; (iv) ARC Discovery Project Grant DP160101513; (v) Fundamental Research Funds for the Central Universities N150402005. Guoren Wang is the corresponding author of this paper.

References

[1] P. Vanhems, A. Barrat, C. Cattuto, J.-F. Pinton, N. Khanafer, C. Regis, B. a Kim, B. Comte, and N. Voirin, "Estimating potential infection transmission routes in hospital wards using wearable proximity sensors," *PLoS ONE*, vol. 8, p. e73970, 2013.
[2] J. Fournet and A. Barrat, "Contact patterns among high school students," *PLOS ONE*, vol. 9, p. e107878, 2014.
[3] Y. Yang, D. Yan, H. Wu, J. Cheng, S. Zhou, and J. C. S. Lui, "Diversified temporal subgraph pattern mining," in *KDD*, 2016.
[4] S. Ma, R. Hu, L. Wang, X. Lin, and J. Huai, "Fast computation of dense temporal subgraphs," in *ICDE*, 2017.
[5] R.-H. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai, "Persistent community search in temporal networks," in *ICDE*, 2018.
[6] I. R.Fischhoff, S. R.Sundaresan, J. Cordingley, H. M.Larkin, and M.-J. Sellier, "Social relationships and reproductive state influence leadership roles in movements of plains zebra, equus burchellii," *Animal Behaviour*, vol. 73, no. 5, pp. 825–831, 2007.
[7] H. Wu, J. Cheng, Y. Lu, Y. Ke, Y. Huang, D. Yan, and H. Wu, "Core decomposition in large temporal graphs," in *IEEE International Conference on Big Data*, 2015.
[8] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," *ACM Journal of Experimental Algorithmics*, vol. 18, 2013.
[9] V. Batagelj and M. Zaversnik, "An O(m) algorithm for cores decomposition of networks," *CoRR cs.DS/0310049*, 2003.
[10] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.
[11] S. B. Seidman, "Network structure and minimum degree," *Social networks*, vol. 5, no. 3, pp. 269–287, 1983.
[12] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical Computer Science*, vol. 363, no. 1, pp. 28–42, 2006.
[13] 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.
[14] S. Huang, A. W. Fu, and R. Liu, "Minimum spanning trees in temporal graphs," in *SIGMOD*, 2015.
[15] S. Gurukar, S. Ranu, and B. Ravindran, "COMMIT: A scalable approach to mining communication motifs from dynamic networks," in *SIGMOD*, 2015.
[16] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke, "Reachability and time-based path queries in temporal graphs," in *ICDE*, 2016.
[17] G. Rossetti and R. Cazabet, "Community discovery in dynamic networks: A survey," *ACM Comput. Surv.*, vol. 51, no. 2, pp. 35:1–35:37, 2018.
[18] Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng, "Facetnet: A framework for analyzing communities and their evolutions in dynamic networks," in *WWW*, 2008.
[19] Z. Chen, K. A. Wilson, Y. Jin, W. Hendrix, and N. F. Samatova, "Detecting and tracking community dynamics in evolutionary networks," in *ICDMW*, 2010.
[20] M. K. Agarwal, K. Ramamritham, and M. Bhide, "Real time discovery of dense clusters in highly dynamic graphs: Identifying real world events in highly dynamic environments," *PVLDB*, vol. 5, no. 10, 2012.
[21] R. H. Li, J. X. Yu, and R. Mao, "Efficient core maintenance in large dynamic graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 10, pp. 2453–2465, 2014.
[22] G. Rossetti, L. Pappalardo, D. Pedreschi, and F. Giannotti, "Tiles: an online algorithm for community discovery in dynamic social networks," *Machine Learning*, vol. 106, no. 8, pp. 1213–1241, 2017.
[23] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks," *ACM Trans. Database Syst.*, vol. 36, no. 4, pp. 21:1–21:34, 2011.
[24] A.-S. Himmel, H. Molter, R. Niedermeier, and M. Sorge, "Adapting the bron–kerbosch algorithm for enumerating maximal cliques in temporal graphs," *Social Network Analysis and Mining*, vol. 7, no. 1, pp. 7–35, 2017.
[25] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye, "Mining periodic behaviors for moving objects," in *KDD*, 2010.
[26] T. Kurashima, T. Althoff, and J. Leskovec, "Modeling Interdependent and Periodic Real-World Action Sequences," in *WWW*, 2018.
[27] K. Radinsky, K. Svore, S. Dumais, J. Teevan, A. Bocharov, and E. Horvitz, "Modeling and predicting behavioral dynamics on the web," in *WWW*, 2012.
[28] M. Lahiri and T. Y. Berger-Wolf, "Periodic subgraph mining in dynamic networks," *Knowledge and Information Systems*, vol. 24, no. 3, pp. 467–497, 2010.