2018 IEEE 34th International Conference on Data Engineering

Persistent Community Search in Temporal Networks

Rong-Hua Li[†], Jiao Su[*], Lu Qin[‡], Jeffrey Xu Yu[*], and Qiangqiang Dai[§]

[†]*Beijing Institute of Technology, China;* [*]*The Chinese University of Hong Kong, Hong Kong*

[‡]*Centre for Artificial Intelligence, University of Technology, Sydney, Australia;* [§]*Shenzhen University, China*

lironghuascut@gmail.com; {jiaosu,yu}@se.cuhk.edu.hk; Lu.Qin@uts.edu.au; qiang56734@163.com

***Abstract*—Community search is a fundamental graph mining task. In applications such as analysis of communication networks, collaboration networks, and social networks, edges are typically associated with timestamps. Unfortunately, most previous studies focus mainly on identifying communities in a network without temporal information. In this paper, we study the problem of finding persistent communities in a temporal network, in which every edge is associated with a timestamp. Our goal is to identify the communities that are persistent over time. To this end, we propose a novel persistent community model called (θ, τ)-persistent k-core to capture the persistence of a community. We prove that the problem of identifying the maximum (θ, τ)-persistent k-core is NP-hard. To solve this problem, we first propose a near-linear temporal graph reduction algorithm to prune the original temporal graph substantially, without loss of accuracy. Then, in the reduced temporal graph, we present a novel branch and bound algorithm with several carefully-designed pruning rules to find the maximum (θ, τ)-persistent k-cores efficiently. We conduct extensive experiments in several real-world temporal networks. The results demonstrate the efficiency, scalability, and effectiveness of the proposed solutions.**

I. Introduction

Many real-world networks such as social networks, biological networks, and communication networks contain community structures. Finding communities in a network is a fundamental task in network analysis which has attracted much attention in recent years due to a large number of real-life applications [1], [2], [3], [4].

In applications such as analysis of communication networks, collaboration networks of scientific papers, and social networks, the edges are often associated with temporal information. For example, in the mobile-phone calling network, each phone call contains a sender and a receiver, as well as the time when the phone call was made. In the face-to-face contact network [5], [6], each edge (u, v, t) represents a contact between two individuals u and v at time t. Another example is the collaboration network, where each edge represents two authors, u and v, coauthored a paper at time t. Unfortunately, most previous community search algorithms ignore the temporal information of the graph, thus may fail to identify important temporal patterns such as the persistent and evolving community structures.

In this paper, we investigate the problem of finding persistent community structures in the temporal network, where each edge is associated with a timestamp. We aim to discover the communities in a temporal network that are persistent over time. To this end, an intuitive approach is to identify all the *persistent edges* in the temporal network (e.g., the edge with persistence no less than a given threshold), and then applies traditional community search algorithms in the temporal sub-network induced by those *persistent edges* to find the communities. This method, however, cannot completely reveal the persistent communities. This is because various *persistent edges* may be persistent over diverse time intervals, and thus the whole community is not necessary persistent over time. Indeed, we empirically show that such an intuitive solution fails to identify persistent communities (see Section V-B for details). Another potential approach is the temporal clique enumeration [7], where a temporal clique involves a set of nodes and a time interval I. It persistently maintains the clique structure in any θ-length subinterval of I. However, in this model, the *clique constraint* could be too strong, which may miss many useful persistent communities violating such a *clique constraint*. Moreover, the temporal clique model only considers one continue time interval. The real-world persistent communities, however, may be persistent over different time intervals. Therefore, the temporal clique model cannot fully capture the persistence of the communities.

To overcome the limitations of the above methods, we propose a new persistent community model, called (θ, τ)-persistent k-core, based on the concept of k-core [8]. For a time interval I, our model persistently preserves the k-core structure in any θ-length subinterval of I. We refer to such an interval I as a persistent interval for convenience. A persistent interval is maximal if none of its super-intervals is a persistent interval. A community may have many different maximal persistent intervals. We thus aggregate the duration of all such maximal persistent intervals to measure the *core persistence* of a community. We define a community as a (θ, τ)-persistent k-core if it is a maximal induced temporal subgraph such that its *core persistence* is no smaller than a given parameter τ (see Section II for details). In many temporal network applications, (θ, τ)-persistent k-core can help to discover interesting groups that are persistent over time. Such persistent groups can be useful for finding a stable team of experts in collaboration networks, and revealing different contact patterns in face-to-face contact networks [9]. Below, we detail the applications.

Consider an application of finding a team of experts in a scientific collaboration network (e.g., DBLP). We may wish to find a *stable* and cohesive team to complete a task in which the members continuously coauthored papers with each other over a prolonged period. By identifying (θ, τ)-persistent k-cores, we can obtain the communities that persistently preserve a k-core structure for a long time, indicating that they are persistent and cohesive teams. Another interesting application of our model arises in a face-to-face contact network. Consider a temporal network of face-to-face contacts between patients and health-care workers (including nurses, doctors, and administrative staffs) in a hospital [5]. Finding (θ, τ)-persistent k-cores in this temporal network can help to reveal the nature of interactions between patients and health-care workers, and show interesting semantics of contact. For example, let $\theta = 60$ seconds, $k = 3$, and $\tau = 300$ seconds. Suppose that we find a $(60, 300)$-persistent 3-core community involving a patient, two nurses and a doctor. This community is likely to be an event that the doctor treats the patient with the help of two nurses, as a typical treatment process may last around 5 minutes. Note that traditional community models (e.g., k-core) cannot reveal such a persistent contact pattern. Another example is that a $(300, 3600)$-persistent 10-core community, including an administrative staff and 10 nurses, probably corresponds to a one-hour meeting among these individuals. Further, assume that we find a $(3600, 28800)$-persistent 3-core community including four nurses during more than 8 hours. This community is likely to be a group of nurses that work in the same room, because they persistently contact each other in

2375-026X/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDE.2018.00077

a prolonged period. Note that revealing such important contact patterns in a hospital can be useful for preventing the spread of hospital-acquired infections [5]. For instance, consider again the $(3600, 28800)$-persistent 3-core community involving four nurses. If one of them get an infectious disease, then the other three nurses are at a high risk of getting the same infectious disease.

Contributions. In this paper, we formalize and provide solutions to find the persistent communities in temporal networks. In particular, we make the following principal contributions.

New model and hardness result. We propose a new persistent community model called (θ, τ)-persistent k-core to fully characterize the persistence of a community. We prove that computing the maximum (θ, τ)-persistent k-core (or enumerating all (θ, τ)-persistent k-cores) is NP-hard.

Novel algorithms. To find the maximum (θ, τ)-persistent k-core, we first develop a novel temporal graph reduction (TGR) algorithm with near-linear time complexity to prune the input temporal graph. The TGR algorithm is based on an elegant and newly-developed meta-interval decomposition technique. We show that computing the meta-interval decomposition in a temporal graph can be done in linear time and uses linear space. Then, in the reduced temporal graph, we propose a novel branch and bound algorithm with several powerful pruning rules to find the maximum (θ, τ)-persistent k-core efficiently. Moreover, our algorithm can also be used to enumerate all (θ, τ)-persistent k-cores.

Experimental evaluation. We conduct extensive experiments to evaluate the proposed model and algorithms using several real-world temporal networks. The results show that in a temporal graph with more than one million nodes and ten millions edges, our algorithm consumes less than 100 seconds to identify the maximum (θ, τ)-persistent k-core in most parameter settings. We also perform comprehensive case studies to evaluate the effectiveness of the proposed model. The results demonstrate that our model can identify many meaningful and interesting persistent communities that cannot be found by the other models.

II. Preliminaries

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected temporal graph, where $\mathcal{V}$ and $\mathcal{E}$ denote the set of vertices 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 vertices in V, and t is the interaction time between u and v. Let st and ed be the time of the first and the last occurred temporal edges respectively. The interaction time t of any temporal edge (u, v, t) in $\mathcal{G}$ must fall into the interval $[st, ed]$. The length of the interval $[st, ed]$ is equal to $|ed - st|$. In this paper, we assume that t is an integer, because the timestamp is an integer in practice. Note that the temporal edges (u, v, t_1) and (u, v, t_2) with $t_2 \neq t_1$ are considered as two different temporal edges. Clearly, the temporal graph can be represented as edge streams [7], [10]. Fig. 1(a) illustrates an example of a temporal graph with 4 nodes and 7 temporal edges.

For a subset of vertices $\mathcal{V}_S$, the induced temporal subgraph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$ is a subgraph such that $\mathcal{E}_S = \{(u, v, t) | u, v \in \mathcal{V}_S, (u, v, t) \in \mathcal{E}\}$. The temporal neighborhood of a node u in $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by $\mathcal{N}_u(\mathcal{G}) \triangleq \{(u, v, t) | (u, v, t) \in \mathcal{E}\}$. For a temporal graph $\mathcal{G}$, the projected graph denoted by G over the time interval $[t_s, t_e]$ is defined as $G = (V, E, [t_s, t_e])$, where $V = \mathcal{V}$ and $E = \{(u, v) | (u, v, t) \in \mathcal{E}, t \in [t_s, t_e]\}$. Fig. 1(b) illustrates the projected graph of the temporal graph in Fig. 1(a) over the interval $[1, 8]$. The degree of a node u in G, denoted by $d_u(G)$, is the number of neighbors in G. A k-core in a graph G is an induced subgraph $C = (V_C, E_C)$,

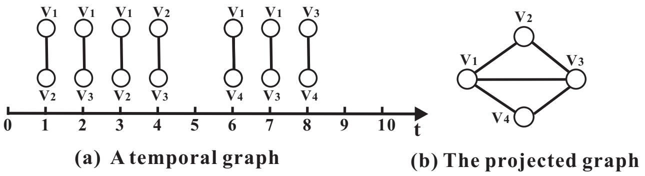


Fig. 1. Running example

where each node in V_C has a degree no smaller than k [8]. A *connected* k-core is a k-core and it is also connected.

The persistent community model. Intuitively, a persistent community should continue to occur over a prolonged period in the temporal graph. To model a community, we choose the well-known k-core model [8], as it is widely used in community search applications [11], [12], [13]. Based on this, we propose a novel persistent community model, called (θ, τ)-persistent k-core, which can capture the persistence of the k-core structure.

For convenience, we assume that both the parameters θ and τ in our model are integers, because the timestamp is an integer. Note that the proposed model and algorithms can also be extended to handle the case when θ and τ are non-integers. Below, we first give a definition called maximal (θ, k)-persistent-core interval.

*Definition 1: (*Maximal (θ, k)-persistent-core interval*)* Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and parameters θ and k, an interval $[t_s, t_e]$ with $t_e - t_s \geq \theta$ is called a maximal (θ, k)-persistent-core interval for $\mathcal{G}$ if and only if the following two conditions hold. (1) For any $t \in [t_s, t_e - \theta]$, the projected graph of $\mathcal{G}$ over the interval $[t, t + \theta]$ is a *connected* k-core. (2) There is no super-interval of $[t_s, t_e]$ such that (1) holds.

By Definition 1, we can see that the temporal graph $\mathcal{G}$ in the maximal (θ, k)-persistent-core interval persistently maintains the k-core structure in any θ-length subinterval.

Example 1: Consider the temporal graph $\mathcal{G}$ in Fig. 1(a). Assume that $\theta = 3$ and $k = 2$. We can see that there is no maximal $(3, 2)$-persistent-core interval for the entire graph $\mathcal{G}$. As shown in Fig. 2, there is a maximal $(3, 2)$-persistent-core interval $[1, 5]$ for the subgraph $\mathcal{C}$ induced by nodes $\{v_1, v_2, v_3\}$. This is because $[1, 5]$ is the maximal interval such that in any 3-length subinterval of $[1, 5]$, the nodes $\{v_1, v_2, v_3\}$ form a connected 2-core.

There may exist several maximal (θ, k)-persistent-core intervals for a temporal graph $\mathcal{G}$, and these intervals may overlap each other. For example, if $\theta = 3$ and $k = 1$, the 1-core $\{v_1, v_3\}$ has two overlapped maximal (θ, k)-persistent-core intervals $[-1, 5]$ and $[4, 10]$. Lemma 1 shows that the length of the overlapped subintervals of any two maximal (θ, k)-persistent-core intervals must be smaller than θ. In the rest of this paper, all proofs are omitted due to the space limit.

Lemma 1: Let $[t_{s_1}, t_{e_1}]$ and $[t_{s_2}, t_{e_2}]$ be two overlapped maximal (θ, k)-persistent-core intervals of a temporal graph $\mathcal{G}$, and $t_{e_1} < t_{e_2}$. Then, we have $t_{e_1} - t_{s_2} \leq \theta$.

Based on the maximal (θ, k)-persistent-core interval, we define the core persistence of a temporal graph $\mathcal{G}$ as follows.

*Definition 2: (*Core persistence*)* Let $\mathcal{T} = \{[t_{s_1}, t_{e_1}], \cdots, [t_{s_r}, t_{e_r}]\}$ be the set of all maximal (θ, k)-persistent-core intervals of $\mathcal{G}$. Then, the core persistence of $\mathcal{G}$ with parameters θ and k, denoted by $F(\theta, k, \mathcal{G})$, is defined as

$$F(\theta, k, \mathcal{G}) \triangleq \begin{cases} \sum_{i=1}^{r} (t_{e_i} - t_{s_i}) - (r-1)\theta, & if \mathcal{T} \neq \emptyset \\ 0, & otherwise \end{cases} \quad (1)$$

By Definition 2, the core persistence of a temporal graph $\mathcal{G}$ is equal to the total length of all maximal (θ, k)-persistent-core intervals of $\mathcal{G}$ minus $(r - 1)\theta$. The rationale behind this definition is twofold. First, since two maximal (θ, k)-persistent-core intervals may overlap and the length of the overlapped interval is bounded by θ (see Lemma 1), we

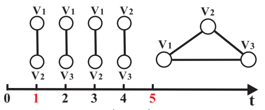


Fig. 2. Illustration of the maximal (θ, k)-persistent-core interval ($\theta = 3, k = 2$). The 2-core $\{v_1, v_2, v_3\}$ appears in any 3-length subinterval of $[1, 5]$.

discount θ for every two *adjacent* intervals. For r maximal (θ, k)-persistent-core intervals, the total length thus decreases by $(r-1)\theta$. Second, using Eq. (1) can reduce the effect of the θ-length interval. Consider the following two cases: (1) there are r θ-length maximal (θ, k)-persistent-core intervals for $\mathcal{G}$, and (2) there is only one maximal (θ, k)-persistent-core interval for $\mathcal{G}$ with length $r \times \theta$. Intuitively, $\mathcal{G}$ under the case (2) should be considered as more persistent than $\mathcal{G}$ under the case (1). This is because in case (2), $\mathcal{G}$ maintains the k-core structure in a r times longer interval than each interval in case (1). By using the total length of all intervals, we cannot distinguish these two different cases. By Definition 2, the core persistence of $\mathcal{G}$ in the case (1) is θ, whereas in the case (2) it equals $r \times \theta$. Based on these reasons, we use Definition 2 to define the core persistence. It is worth mentioning that the proposed algorithms also work when the core persistence is defined as the total length of all maximal (θ, k)-persistent-core intervals.

Definition 3: ((θ, τ)-persistent k-core) Given a temporal graph $\mathcal{G}$, parameters θ, τ, and k, a (θ, τ)-persistent k-core is an induced temporal subgraph $\mathcal{C} = (\mathcal{V}_C, \mathcal{E}_C)$ that meets the following properties.

- Persistent core property: $F(\theta, k, \mathcal{C}) \geq \tau$;
- Maximal property: there does not exist an induced temporal subgraph $\mathcal{C}'$ that contains $\mathcal{C}$ and also satisfies the persistent core property.

Intuitively, the (θ, τ)-persistent k-core is capable of capturing the persistence of a community, because it preserves a k-core structure in every θ time in the maximal (θ, k)-persistent-core intervals with total length no less than τ.

Example 2: Reconsider the temporal graph $\mathcal{G}$ in Fig. 1(a). Let $\theta = 3$, $\tau = 4$ and $k = 2$. We claim that the subgraph $\mathcal{C}$ induced by nodes $\{v_1, v_2, v_3\}$ is a $(3, 4)$-persistent 2-core. The reason is as follows. First, the subgraph $\mathcal{C}$ has a maximal $(3, 2)$-persistent-core interval $[1, 5]$ (see Example 1). Therefore, the core persistence of $\mathcal{C}$ is equal to 4 (i.e., $F(3, 2, \mathcal{C}) = 4$), which is no less than τ. On the other hand, $\mathcal{C}$ is the maximal subgraph that meets such a persistent core property. As a consequence, the subgraph $\mathcal{C}$ is a $(3, 4)$-persistent 2-core.

Note that the (θ, τ)-persistent k-cores of a temporal graph may overlap each other. As a result, the number of (θ, τ)-persistent k-cores may be exponentially large, and thus it is very costly to enumerate all (θ, τ)-persistent k-cores. In practice, we are often interested in finding the maximum (θ, τ)-persistent k-core. Therefore, in this paper, we focus mainly on finding the largest (θ, τ)-persistent k-core. We emphasize that the proposed solutions can also be used to enumerate all (θ, τ)-persistent k-cores as discussed in Section IV.

The persistent community (PC) search problem. Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parameters θ, τ and k, we aim to find the largest (θ, τ)-persistent k-core in $\mathcal{G}$.

Hardness result. Since the number of (θ, τ)-persistent k-cores in a temporal graph can be exponentially large, it is very hard to identify the largest one. We show that the PC search problem is NP-hard.

Theorem 1: The problem of finding the largest (θ, τ)-persistent k-core is NP-hard.

Proof: We consider the decision version of Problem 2. Given a temporal graph $\mathcal{G}$, parameters θ, τ, and k, test whether $\mathcal{G}$ contains an induced temporal subgraph with $\eta + 1$ nodes that satisfies the persistent core property. Clearly, the maximum subgraph that meets the persistent core property is the largest (θ, τ)-persistent k-core. To prove the theorem, we construct a reduction from the balanced bipartite subgraph (BBS) problem, which is known to be NP-hard [14]. Given a connected bipartite graph $\mathcal{B} = (\mathcal{X}, \mathcal{Y}, \mathcal{E}_B)$ and an integer η, the BBS problem is to verify whether $\mathcal{B}$ contains a complete bipartite subgraph with η nodes on each side. Based on this, we construct an instance of our problem which consists of a temporal graph $\mathcal{G}$, parameters $\theta = 1$, $\tau = (\eta + 1)$, and $k = 1$. For convenience, we assume that the nodes in $\mathcal{Y}$ are sorted by their identities, denoted by $\mathcal{Y} = \{u_1, u_2, \cdots, u_n\}$. First, we create a node s for the temporal graph, which corresponds to every node $u_i \in \mathcal{Y}$ $(i = 1, \cdots, n)$. For each edge $(v, u_i) \in \mathcal{E}_B$, we create a temporal edge $(s, v, 2 \times i - 1)$. Based on this construction, we obtain a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{X} \cup \{s\}$ and $\mathcal{E} = \{(s, v, 2 \times i - 1) | (v, u_i) \in \mathcal{E}_B\}$. Clearly, for each node $u_i \in \mathcal{Y}$, we can obtain a star-shaped subgraph in $\mathcal{G}$ at time $2i - 1$, which is denoted by $\mathcal{S}_i$. Below, we show that the instance of the BBS problem is a Yes-instance if and only if the corresponding instance of our problem is a Yes-instance.

First, suppose that $\mathcal{B}$ contains a complete bipartite subgraph $\mathcal{H} = (\mathcal{X}_H, \mathcal{Y}_H, \mathcal{E}_H)$ such that $|\mathcal{X}_H| = |\mathcal{Y}_H| = \eta$. Then, by our construction, for each node $u_j \in \mathcal{Y}_H$, we have a star-shaped subgraph in $\mathcal{G}$ which includes all the nodes in $\mathcal{X}_H$ and a node s. Let $\mathcal{C}$ be the star-shaped subgraph induced by $\mathcal{X}_H \cup \{s\}$. Clearly, $\mathcal{C}$ is a 1-core. For each $u_j \in \mathcal{Y}_H$, the maximal (θ, k)-persistent-core interval of $\mathcal{C}$ is $[2j - 2, 2j]$. Since $|\mathcal{Y}_H| = \eta$, we have η different and non-overlapped maximal (θ, k)-persistent-core intervals with the same length 2. Therefore, the core persistence of $\mathcal{C}$ is $\eta + 1$. As a consequence, we obtain an Yes-instance of our problem, because $\mathcal{C}$ has $\eta + 1$ nodes and its core persistence is no smaller than $\tau = \eta + 1$.

Second, suppose that $\mathcal{G}$ consists of a temporal subgraph $\mathcal{C}$ with $\eta + 1$ nodes and its core persistence (core parameter $k = 1$) is no less than $\eta + 1$. Recall that $\mathcal{G}$ is constructed by a set of star-shaped subgraphs at different timestamps. For a star-shaped subgraph $\mathcal{S}$, we call an interval $[t_s, t_e]$ the maximal θ-persistent interval of $\mathcal{S}$ if and only if the following two conditions hold: (1) $\mathcal{S}$ appears in every θ-length subintervals and (2) there does not exist a super-interval that satisfies (1). Clearly, for each star-shaped subgraph $\mathcal{S}_i$, the maximal θ-persistent interval is $[2i - 2, 2i]$ $(\theta = 1)$. Since all the maximal 1-persistent intervals of the star-shaped subgraphs are non-overlapped and the core persistence of $\mathcal{C}$ is no less than $\eta + 1$, $\mathcal{C}$ must be contained in at least η different star-shaped subgraphs. On the other hand, since the core persistence of $\mathcal{C}$ is no less than $\eta + 1$, the projected graph of $\mathcal{C}$ is connected. By our construction, we can conclude that $\mathcal{C}$ must contain the node s. By picking η star-shaped subgraphs that contains $\mathcal{C}$, we can recover a balanced complete bipartite subgraph that consists of η nodes on each side. Thus, we obtain an Yes-instance of the BBS problem. ■

III. Temporal graph reduction

As shown in Theorem 1, the problem of finding the largest (θ, τ)-persistent k-core is NP-hard. Therefore, straightforwardly computing the largest (θ, τ)-persistent k-core is intractable for large temporal graphs. To solve our problem efficiently, we propose a novel temporal graph reduction (TGR) technique, which can substantially reduce the size of the temporal graph without missing any (θ, τ)-persistent k-core. We show that our TGR technique can be implemented in near-linear time (with a factor θ), thus it is very efficient in practice. After reducing the temporal graph size, we will propose an efficient branch and bound algorithm to solve our problem in Section IV.

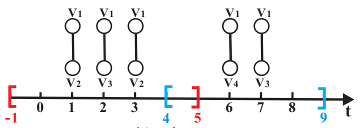


Fig. 3. Illustration of maximal (θ, k)-persistent-degree intervals for v_1 ($\theta = 3, k = 2$). $d_{v_1} \geq 2$ holds in any 3-length subinterval of $[-1, 5]$ (or $[4, 9]$).

A. The key idea

By Definition 3, we can easily derive that every node in a (θ, τ)-persistent k-core must persistently preserve the property of "degree no less than k" in the maximal (θ, k)-persistent-core intervals with total length no smaller than τ. For convenience, we refer to such a property as a *persistent-degree* property. Clearly, we can safely prune the nodes in the temporal graph $\mathcal{G}$ that do not meet the *persistent-degree* property. Furthermore, pruning the unpromising nodes may result in their neighbor nodes that do not satisfy the *persistent-degree* property. Therefore, we can iteratively remove the unpromising nodes from $\mathcal{G}$ until all nodes in the reduced graph satisfy the persistent-degree property. Below, we introduce two new definitions, which are important to devise the temporal graph reduction (TGR) algorithm.

*Definition 4: (*Maximal (θ, k)-persistent-degree interval*)* Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a node $u \in \mathcal{V}$, and two integer parameters θ and k, an interval $[t_s^u, t_e^u]$ with $t_e^u - t_s^u \geq \theta$ is called a maximal (θ, k)-persistent-degree interval for u if and only if the following two conditions hold. (1) For any $t \in [t_s^u, t_e^u - \theta]$, the degree of u in the projected graph of $\mathcal{G}$ over the interval $[t, t + \theta]$ is no less than k. (2) There does not exist a super-interval of $[t_s^u, t_e^u]$ such that (1) holds.

In the maximal (θ, k)-persistent-degree interval of u, the degree of u is no smaller than k in any θ-length subinterval by Definition 4. In this sense, u persistently maintains the property of "degree no less than k" in the maximal (θ, k)-persistent-degree interval.

Example 3: Consider the temporal graph in Fig. 1(a). Let $\theta = 3$ and $k = 2$. Then, as illustrated in Fig. 3, there are two maximal (θ, k)-persistent-degree intervals for node v_1 which are $[-1, 5]$[1] and $[4, 9]$. This is because in $[-1, 5]$, v_1 has a degree no less than 2 in each subinterval with length 3. Moreover, there is no super-interval of $[-1, 5]$ such that v_1 persistently maintains the property of degree no less than 2. Note that the interval $[-1, 6]$ is not a maximal (θ, k)-persistent-degree interval, as there exist θ-length subintervals (e.g., $[2.5, 5.5]$) such that v_1's degree is smaller than 2. Similar results also hold for the interval $[4, 9]$. Clearly, we can see that these two maximal (θ, k)-persistent-degree intervals overlap. Likewise, for node v_2, we can obtain a maximal (θ, k)-persistent-degree interval $[1, 6]$. For node v_3, we obtain two maximal (θ, k)-persistent-degree intervals which are $[1, 5]$ and $[5, 10]$. For v_4, we have a maximal (θ, k)-persistent-degree interval $[5, 9]$.

Similar to Definition 1, there may exist several maximal (θ, k)-persistent-degree intervals for a node u, and these intervals may overlap each other (as shown in Example 3). The following lemma shows that for a node u, the length of the overlapped subinterval of two maximal (θ, k)-persistent-degree intervals is smaller than θ.

Lemma 2: Let $[t_{s_1}^u, t_{e_1}^u]$ and $[t_{s_2}^u, t_{e_2}^u]$ be two overlapped maximal (θ, k)-persistent-degree intervals of a node u, and $t_{e_1}^u < t_{e_2}^u$. Then, we have $t_{e_1}^u - t_{s_2}^u \leq \theta$.

Similar to Definition 2, we define the degree persistence for a node u as follows.

[1]Note that "-1" as a *timestamp* does not affect the correctness of all the examples given in this paper.

*Definition 5: (*Degree persistence*)* Let $\mathcal{T} = \{[t_{s_1}^u, t_{e_1}^u], \cdots, [t_{s_r}^u, t_{e_r}^u]\}$ be the set of all maximal (θ, k)-persistent-degree intervals of u. Then, the degree persistence of u with parameters θ and k, denoted by $f(u, \theta, k, \mathcal{G})$, is defined as

$$f(u, \theta, k, \mathcal{G}) \triangleq \begin{cases} \sum_{i=1}^{r} (t_{e_i}^u - t_{s_i}^u) - (r-1)\theta, & if \mathcal{T} \neq \emptyset \\ 0, & otherwise. \end{cases} \quad (2)$$

By Definition 5, the degree persistence of each node satisfies *monotonic* property. That is to say, the degree persistence of any node u in a temporal graph $\mathcal{G}$ is no less than the degree persistence of u in any temporal subgraph of $\mathcal{G}$.

Lemma 3: Let $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ be a temporal subgraph of $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$. Then, for any node $u \in \mathcal{V}_1$, we have $f(u, \theta, k, \mathcal{G}_1) \leq f(u, \theta, k, \mathcal{G}_2)$.

Example 4: Reconsider the temporal graph $\mathcal{G}$ in Fig. 1(a). Let $\theta = 3$ and $k = 2$. Then, based on the results in Example 3, we can easily derive that $f(v_1, 3, 2, \mathcal{G}) = 8$, $f(v_2, 3, 2, \mathcal{G}) = 5$, $f(v_3, 3, 2, \mathcal{G}) = 6$, and $f(v_4, 3, 2, \mathcal{G}) = 4$. However, if we only consider the temporal subgraph $\mathcal{G}'$ induced by the nodes $\{v_1, v_2, v_3\}$, we can obtain that $f(v_1, 3, 2, \mathcal{G}') = 6$, $f(v_2, 3, 2, \mathcal{G}') = 5$, $f(v_3, 3, 2, \mathcal{G}') = 4$. Clearly, we have $f(v_1, 3, 2, \mathcal{G}') < f(v_1, 3, 2, \mathcal{G})$, $f(v_2, 3, 2, \mathcal{G}') = f(v_2, 3, 2, \mathcal{G})$, and $f(v_3, 3, 2, \mathcal{G}') < f(v_3, 3, 2, \mathcal{G})$. This example confirms that the degree persistence of each node indeed satisfies the monotonic property (Lemma 3).

By Definitions 3 and 5, we can easily prove that the degree persistence of each node in a (θ, τ)-persistent k-core must be no less than τ.

Lemma 4: Suppose that there is a (θ, τ)-persistent k-core $\mathcal{C}$ in the temporal graph $\mathcal{G}$. Then, for every node $u \in \mathcal{C}$, we has $f(u, \theta, k, \mathcal{C}) \geq \tau$.

Based on Lemmas 3 and 4, we can safely prune the node v in $\mathcal{G}$ if $f(u, \theta, k, \mathcal{G}) < \tau$, as v is definitely not contained in any (θ, τ)-persistent k-core. Moreover, after pruning all the unpromising nodes in $\mathcal{G}$, the degree persistence of the residual nodes may decrease (by Lemma 3). Thus, this pruning procedure may trigger a *domino effect*, i.e., the pruning of unpromising nodes may result in new unpromising nodes. As a consequence, we can iteratively remove the unpromising nodes until the degree persistence of every node in the residual graph is no less than τ. We refer to such an iterative pruning procedure as a TGR procedure. The following example illustrates the TGR procedure.

Example 5: Reconsider the temporal graph $\mathcal{G}$ in Fig. 1(a). Suppose that $\theta = 3$, $k = 2$ and $\tau = 5$. As shown in Example 4, v_4 is an unpromising node, as $f(v_4, 3, 2, \mathcal{G}) = 4 < \tau$. Thus, we can remove v_4. After pruning v_4, we obtain a temporal subgraph $\mathcal{G}'$ induced by $\{v_1, v_2, v_3\}$. The deletion of v_4 leads to $f(v_3, 3, 2, \mathcal{G}') = 4 < \tau$ (Example 4), and thus we can further delete v_3. Iteratively, we can get that all nodes will be deleted, indicating that $\mathcal{G}$ does not contain a $(3, 5)$-persistent 2-core.

Challenges. There are two challenges to implement the TGR procedure. First, we need to compute the degree persistence for each node u in $\mathcal{G}$ efficiently. By Definition 5, we has to find all the maximal (θ, k)-persistent-degree intervals of u to compute the degree persistence of u. It is nontrivial to devise an efficient algorithm to find all such intervals, because the intervals may overlap each other. Second, the degree persistence of a node u may frequently update in the TGR procedure. It is challenging to devise an efficient algorithm to maintain the degree persistence for the remaining nodes after pruning an unpromising node.

Below, we propose an elegant meta-interval decomposition technique to tackle these challenges. Based on the meta-interval decomposition technique, we can implement the TGR procedure in near-linear time, using linear space.

B. Meta-interval decomposition

We assume in this paper that all the temporal edges in $\mathcal{G}$ are sorted in an increasing order by their timestamps. Note that if this assumption does not hold, we can sort the temporal edges in linear time, because each timestamp is an integer.

By Definitions 4 and 5, we can compute the degree persistence for a node u based on the temporal neighborhood of u. Let $\mathcal{N}_u(\mathcal{G}) = \{(u, v_1, t_1), \cdots, (u, v_h, t_h)\}$ with $t_i < t_{i+1}$ $(i = 1, \cdots, h-1)$ be the temporal neighborhood of u. Note that all the $v_i \in \mathcal{N}_u(G)$ are not necessarily distinct, i.e., there may exist repeated neighbors. For example, in the temporal graph shown in Fig. 1(a), $\mathcal{N}_{v_1} = \{(v_1, v_2, 1), (v_1, v_3, 2), (v_1, v_2, 3), (v_1, v_4, 6), (v_1, v_3, 7)\}$ in which v_2 and v_3 are repeated neighbors of v_1. The degree of u w.r.t. $\mathcal{N}_u(\mathcal{G})$ denotes the number of distinct neighbors in $\mathcal{N}_u(\mathcal{G})$. To easily describe the meta-interval decomposition technique, we first assume that all the $v_i \in \mathcal{N}_u(G)$ are distinct, and then we will show how to extend our technique to handle the case when there exist repeated neighbors in $\mathcal{N}_u(G)$.

For each node u, the key idea of our technique is to decompose the entire time interval of $\mathcal{N}_u(G)$ into a set of meta-intervals, and then compute and maintain the degree persistence of u only using the meta-intervals. We introduce the definition of the meta-interval as follows.

*Definition 6: (*Meta-Interval*)* Given a node u and its temporal neighborhood $\mathcal{N}_u(\mathcal{G})$, an open interval (t_s, t_e) is a meta-interval of u if and only if the following conditions hold: (1) $t_e - t_s > \theta$, and (2) in every θ-length subinterval of (t_s, t_e), u has the same degree.

Definition 6 indicates that a node u in the meta-interval (t_s, t_e) has a *uniform* degree. Below, we give a useful definition called θ-persistent degree for a node u in the interval $[t_l, t_r]$ (or (t_l, t_r)).

Definition 7: (θ-persistent degree) Given a node u and a temporal graph $\mathcal{G}$. The θ-persistent degree of a node u in the interval $I = [t_l, t_r]$ (or (t_l, t_r)) is the maximal integer d such that u has a degree no less than d in every θ-length subinterval of I.

Clearly, in the meta-interval of u, the so-called *uniform* degree of u is equal to the θ-persistent degree by Definitions 6 and 7. However, it should be noted that the θ-persistent degree of a node u is also well-defined for any general interval (not only for meta-intervals). The following example illustrates the definitions of meta-interval and θ-persistent degree.

Example 6: Consider the node v_1 in the temporal graph in Fig. 1(a). Suppose that $\theta = 3$. Then, we can check that $(0, 5)$ is a meta-interval for v_1, because in any 3-length subinterval of $(0, 5)$, v_1 has the same degree 2. Similarly, the open interval $(2, 6)$ is also a meta-interval for v_1. This is because in the open interval $(2, 6)$, there are only two temporal edges ($(v_1, v_2, 3)$ and $(v_2, v_3, 4)$), and v_1 has the same degree 1 in any 3-length subinterval of $(2, 6)$. Clearly, by Definition 7, the 3-persistent degree of v_1 in the meta-interval $(0, 5)$ is 2, because 2 is the maximal integer such that v_1 has a degree no less than 2 in every 3-length subinterval of $(0, 5)$. Also, we can see that in the closed interval $[5, 9]$, the 3-persistent degree of v_1 is 2.

To compute the meta-intervals of a node u and the corresponding θ-persistent degree in each meta-interval, we define the lifespan of a temporal edge (u, v, t) as $[t, t+\theta]$, because the edge (u, v, t) contributes 1 to the degree of u in this interval.

Algorithm 1 outlines our meta-interval decomposition technique, which can compute all the meta-intervals as well as the corresponding θ-persistent degrees for a node u. For each temporal edge $(u, v_i, t_i) \in \mathcal{N}_u(G)$, the algorithm generates

Algorithm 1 Meta-interval-Decomposition (u, $\mathcal{N}_u$, θ)

```
1: Let $\mathcal{N}_u = \{(u, v_1, t_1), \cdots, (u, v_h, t_h)\}$ with $v_i \neq v_j$ for $i \neq j$;
2: for $i = 1$ to $h$ do
3:     Let $[t_i, t_i + \theta]$ be the lifespan of $(u, v_i, t_i)$;
4:     $\mathcal{T}_1(i) \leftarrow \{t_i, +1\}$; $\mathcal{T}_2(i) \leftarrow \{t_i + \theta, -1\}$;
5: $\mathcal{T}' \leftarrow$ Sort($\{\mathcal{T}_1, \mathcal{T}_2\}$); $i \leftarrow 1$; $j \leftarrow 1$;
6: while $i < |\mathcal{T}'| - 1$ do
7:     $\mathcal{T}(j).first \leftarrow \mathcal{T}'(i).first$; $d \leftarrow \mathcal{T}'(i).second$;
8:     while $i < |\mathcal{T}'| - 1$ and $\mathcal{T}'(i).first = \mathcal{T}'(i+1).first$ do
9:         $d \leftarrow d + \mathcal{T}'(i+1).second$; $i \leftarrow i + 1$;
10:    $\mathcal{T}(j).second \leftarrow d$; $j \leftarrow j + 1$;
11: $d \leftarrow 0$;
12: for $i = 1$ to $|\mathcal{T}| - 1$ do
13:    $d \leftarrow d + \mathcal{T}(i).second$;    {// compute the prefix sum for $\mathcal{T}$}
14:    $\mathcal{D}_u(i) \leftarrow d$; $\mathcal{MI}_u(i) \leftarrow (\mathcal{T}(i).first, \mathcal{T}(i+1).first)$;
15: return $\{\mathcal{D}_u, \mathcal{MI}_u\}$;
```

two pairs $\{t_i, +1\}$ and $\{t_i + \theta, -1\}$ (lines 2-4), denoting that the edge (u, v_i) contributes 1 to the degree of u at time t_i and decreases the degree by 1 at time $t_i + \theta$. This is because the lifespan of each temporal edge (u, v_i, t_i) is $[t_i, t_i + \theta]$ by our definition. Then, in line 5, Algorithm 1 sorts all these pairs in an increasing order by their timestamps, i.e., the first item of each pair. Let $\mathcal{T}'$ be the sorted list of those pairs. For each $\mathcal{T}'(i)$, the first term $\mathcal{T}'(i).first$ is a timestamp, and the second term $\mathcal{T}'(i).second$ is an integer either 1 or -1. Note that the first terms of $\mathcal{T}'$ (i.e., all $\mathcal{T}'(i).first$) are not necessarily distinct. Algorithm 1 filters those repeated timestamps in $\mathcal{T}'$, and takes the sum of the second terms of $\mathcal{T}'$ at each repeated timestamp (lines 6-10). After this processing, the algorithm creates an array $\mathcal{T}$ to record the results, where all the first terms in $\mathcal{T}$ are distinct. Subsequently, the algorithm computes the prefix sum based on the second terms of $\mathcal{T}$ (line 13), and builds two arrays $\mathcal{D}_u$ and $\mathcal{MI}_u$ to record the prefix sums and the intervals $(\mathcal{T}(i).first, \mathcal{T}(i+1).first)$ for all $i = 1, \cdots, |\mathcal{T}|-1$, respectively (line 14). By our construction, we can easily derive that the prefix sum computed from $\mathcal{T}(1)$ to $\mathcal{T}(i)$ is equal to the number of *surviving* edges at timestamp $\mathcal{T}(i).first$. Finally, the algorithm returns the arrays $\mathcal{D}_u$ and $\mathcal{MI}_u$. The following theorem shows that for each interval $\mathcal{MI}_u(i) = (t_s, t_e)$, the θ-shifted interval $(t_s - \theta, t_e)$ is a meta-interval, and the corresponding θ-persistent degree of u in $(t_s - \theta, t_e)$ is $\mathcal{D}_u(i)$.

Theorem 2: For each $\mathcal{MI}_u(i) = (t_s, t_e)$ and $\mathcal{D}_u(i)$ computed by Algorithm 1, the interval $(t_s - \theta, t_e)$ is a meta-interval. Moreover, in any θ-length subinterval of $(t_s - \theta, t_e)$, u has the same degree $\mathcal{D}_u(i)$.

Since each interval in $\mathcal{MI}_u$ corresponds to a meta-interval, we call $\mathcal{MI}_u$ the meta-interval array. We also refer to $\mathcal{D}_u$ as the θ-persistent degree array, because $\mathcal{D}_u(i)$ is equal to the θ-persistent degree of u in the meta-interval $\mathcal{MI}_u(i)$. The following example illustrates the meta-interval decomposition technique.

Example 7: Reconsider the temporal graph in Fig. 1(a). Let us consider the node v_4 with parameter $\theta = 3$. By Algorithm 1, we have $\mathcal{T}' = \mathcal{T} = \{(6, +1), (8, +1), (9, -1), (11, -1)\}$. Then, we can easily derive that the θ-persistent degree array (i.e., the prefix sum array) $\mathcal{D}_{v_4} = \{1, 2, 1\}$ and the meta-interval array $\mathcal{MI}_{v_4} = \{(6, 8), (8, 9), (9, 11)\}$. We can see that for any $\mathcal{MI}_{v_4}(i) = (t_s, t_e)$, the θ-persistent degree of v_4 in the meta-interval $(t_s - \theta, t_e)$ is $\mathcal{D}_{v_4}(i)$. For instance, in the meta-interval $(5, 9)$ (corresponding to $\mathcal{MI}_{v_4}(2)$), v_4 has a degree 2 in every subinterval with length 3.

Handling repeated neighbors. Here we extend the meta-interval decomposition technique to the case when the node has repeated neighbors. First, we find that in $\mathcal{N}_u(G)$, if the time gap between any two repeated neighbors is no less than θ, we can deem them as different neighbors and use Algorithm 1 to compute the meta-intervals. The reason is as follows. Let

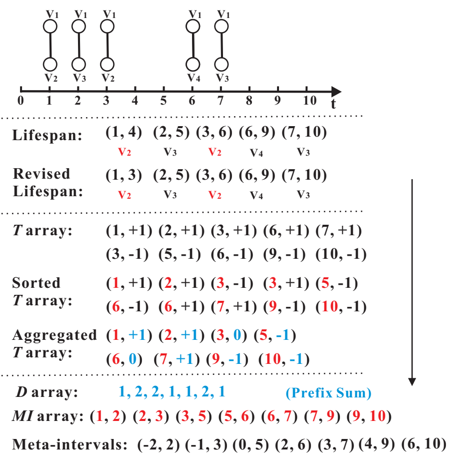

v1 v1 v1 v1 v1
v2 v3 v2 v4 v3
0 1 2 3 4 5 6 7 8 9 10 t
Lifespan: (1, 4) (2, 5) (3, 6) (6, 9) (7, 10)
v2 v3 v2 v4 v3
Revised Lifespan: (1, 3) (2, 5) (3, 6) (6, 9) (7, 10)
v2 v3 v2 v4 v3
T array: (1, +1) (2, +1) (3, +1) (6, +1) (7, +1) (3, -1) (5, -1) (6, -1) (9, -1) (10, -1)
Sorted T array: (1, +1) (2, +1) (3, -1) (3, +1) (5, -1) (6, -1) (6, +1) (7, +1) (9, -1) (10, -1)
Aggregated T array: (1, +1) (2, +1) (3, 0) (5, -1) (6, 0) (7, +1) (9, -1) (10, -1)
D array: 1, 2, 2, 1, 1, 2, 1 (Prefix Sum)
MI array: (1, 2) (2, 3) (3, 5) (5, 6) (6, 7) (7, 9) (9, 10)
Meta-intervals: (-2, 2) (-1, 3) (0, 5) (2, 6) (3, 7) (4, 9) (6, 10)

Fig. 4. Illustration of the meta-interval decomposition for v_1 ($\theta = 3$)

(u, v, t_1) and (u, v, t_2) be two temporal edges, and v is the repeated neighbor of u in $\mathcal{N}_u(G)$. If the gap $t_2 - t_1 \geq \theta$, the next repeated neighbor (v, t_2) appears after (v, t_1) vanishing, thus they do not repeatedly contribute 1 to the degree in any θ-length interval.

Second, for two temporal edges (u, v, t_1) and (u, v, t_2) with $0 < t_2 - t_1 < \theta$, the lifespan of (u, v, t_1) and (u, v, t_2) is $[t_1, t_1 + \theta]$ and $[t_2, t_2 + \theta]$ respectively. Clearly, by our previous algorithm (Algorithm 1), the edges (u, v, t_1) and (u, v, t_2) will contribute 2 to the degree of u in the interval $[t_2, t_1 + \theta]$, which is incorrect. To remedy this issue, we revise the lifespan of (u, v, t_1) by $[t_1, t_2]$, and keep the lifespan of (u, v, t_2) unchanged. Note that this processing can be done in linear time. After this processing, we can use Algorithm 1 to compute the meta-intervals of u. Specifically, we only need to change the line 3 of Algorithm 1 using the revised lifespan for the repeated neighbors. The following example illustrates how the algorithm works.

Example 8: Assume that $\theta = 3$. Fig. 4 illustrates the meta-interval decomposition procedure for v_1 in our running example. For node v_1, the time gap between the two repeated neighbors $(v_2, 1)$ and $(v_2, 3)$ is smaller than θ. Thus, we need to revise the lifespan of the temporal edge $(v_1, v_2, 1)$ by $[1, 3]$. Note that for the two repeated neighbors $(v_3, 2)$ and $(v_3, 7)$, since the time gap between them is larger than θ, it is no need to update the lifespan of the temporal edge $(v_1, v_3, 2)$. After that, we can obtain that $\mathcal{T}' = \{(1, +1), (2, +1), (3, -1), (3, +1), (5, -1), (6, -1), (6, +1), (7, +1), (9, -1), (10, -1)\}$ (i.e., the sorted $\mathcal{T}$ array in Fig. 4), and $\mathcal{T} = \{(1, +1), (2, +1), (3, 0), (5, -1), (6, 0), (7, +1), (9, -1), (10, -1)\}$ (i.e., the aggregated $\mathcal{T}$ array in Fig. 4). Then, we have $\mathcal{D}_{v_1} = \{1, 2, 2, 1, 1, 2, 1\}$ and $\mathcal{MI}_{v_1} = \{(1, 2), (2, 3), (3, 5), (5, 6), (6, 7), (7, 9), (9, 10)\}$. The meta-intervals of v_1 are the θ-shifted intervals as shown in Fig. 4.

Complexity analysis. We analyze the time and space complexity to compute the meta-interval decomposition for all nodes in $\mathcal{G}$ using Algorithm 1 as follows.

Theorem 3: The total time and space costs of Algorithm 1 to compute the meta-interval decomposition for all nodes $u \in \mathcal{G}$ are $O(m)$, where m is the number of temporal edges.

C. Computing degree persistence

Recall that to compute the degree persistence for a node u, we need to find all the maximal (θ, k)-persistent-degree intervals of u. Here we show that all the maximal (θ, k)-persistent-degree intervals of u can be constructed by merging adjacent meta-intervals, and the degree persistence of u can also be computed by the meta-interval decomposition technique.

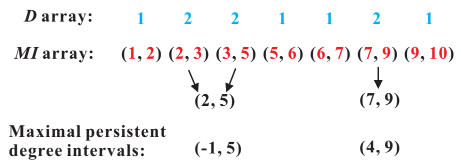

D array: 1 2 2 1 1 2 1
MI array: (1, 2) (2, 3) (3, 5) (5, 6) (6, 7) (7, 9) (9, 10)
(2, 5)
(7, 9)
Maximal persistent degree intervals: (-1, 5) (4, 9)

Fig. 5. Illustration of constructing maximal (θ, k)-persistent-degree intervals for v_1 based on meta-interval decomposition ($\theta = 3, k = 2$)

Below, we consider two different cases.

Case I: $f(u, \theta, k, \mathcal{G}) > \theta$. If the degree persistence $f(u, \theta, k, \mathcal{G}) > \theta$, there must be a maximal (θ, k)-persistent-degree interval of u with length larger than θ. In this case, we can ignore all θ-length maximal (θ, k)-persistent-degree intervals of u, because these intervals contribute nothing to the degree persistence by Definition 5. Therefore, we focus on finding all maximal (θ, k)-persistent-degree intervals of u with length larger than θ.

Let $\mathcal{MI}_u = \{(t_{s_1}, t_{e_1}), \cdots, (t_{s_h}, t_{e_h})\}$ be the meta-interval array of a node u calculated by Algorithm 1 with parameter θ, and $\mathcal{D}_u = \{\mathcal{D}_u(1), \cdots, \mathcal{D}_u(h)\}$ be the θ-persistent degree array of u. For convenience, we define $\mathcal{D}_u(0) = \mathcal{D}_u(h+1) = -\infty$. Then, we have the following result.

Lemma 5: For a parameter k and $i \leq h-1$, if both $\mathcal{D}_u(i) \geq k$ and $\mathcal{D}_u(i+1) \geq k$ hold, u has a degree no less than k in any θ-length subinterval of $[t_{s_i} - \theta, t_{e_{i+1}}]$.

Lemma 5 indicates that we can merge two adjacent meta-intervals in $\mathcal{MI}_u$ if the θ-persistent degree of u in these intervals is no less than k. In the merged interval, u persistently maintains the property of "degree no less k". Based on Lemma 5, we can obtain the maximal (θ, k)-persistent-degree intervals of u by merging the adjacent meta-intervals in $\mathcal{MI}_u$.

Theorem 4: If $\mathcal{D}_u(i) \geq k$ for all $i = l, \cdots, r$ $(l \leq r)$, $\mathcal{D}_u(l-1) < k$ and $\mathcal{D}_u(r+1) < k$ hold, then $[t_{s_l} - \theta, t_{e_r}]$ is a maximal (θ, k)-persistent-degree interval of u.

Example 9: Let $k = 2$ and $\theta = 3$. Fig. 5 illustrates the procedure of constructing maximal (θ, k)-persistent-degree intervals for the node v_1 based on meta-interval decomposition. By Example 8 and Fig. 4, we have $\mathcal{D}_{v_1} = \{1, 2, 2, 1, 1, 2, 1\}$ and $\mathcal{MI}_{v_1} = \{(1, 2), (2, 3), (3, 5), (5, 6), (6, 7), (7, 9), (9, 10)\}$. In $\mathcal{MI}_{v_1}$, we have two adjacent meta-intervals $(2, 3)$ and $(3, 5)$ such that v_1's θ-persistent degree in the intervals $(-1, 3)$ and $(0, 5)$ is no less than k $(k = 2)$. Also, we have a meta-interval $(7, 9)$ such that v_1's θ-persistent degree in the θ-shifted interval $(4, 9)$ is no less than 2. By Theorem 4, we can obtain two maximal (θ, k)-persistent-degree intervals $[-1, 5]$ and $[4, 9]$, which are consistent with the results shown in Example 3.

Theorem 4 shows that all maximal (θ, k)-persistent-degree intervals of u with length larger than θ can be obtained by meta-interval decomposition. Based on this, we can compute the degree persistence for every node u if $f(u, \theta, k, \mathcal{G}) > \theta$ by Theorem 5.

Theorem 5: For a node u, if $f(u, \theta, k, \mathcal{G}) > \theta$, we have $f(u, \theta, k, \mathcal{G}) = \sum_{\mathcal{D}_u(i) \geqslant k} (t_{e_i} - t_{s_i}) + \theta$, where $(t_{s_i}, t_{e_i}) \in \mathcal{MI}_u$.

Case II: $f(u, \theta, k, \mathcal{G}) \leq \theta$. By Definition 5, if $f(u, \theta, k, \mathcal{G}) \leq \theta$, there is no maximal (θ, k)-persistent-degree interval of u with length larger than θ. We consider the following two cases. First, if $\tau > \theta$, we can immediately prune node u, because u's degree persistence $f(u, \theta, k, \mathcal{G}) \leq \theta < \tau$.

Second, if $\tau \leq \theta$, we can slightly modify Algorithm 1 to compute all the maximal (θ, k)-persistent-degree intervals of u. Note that in this case, the length of each maximal (θ, k)-persistent-degree interval of u must be equal to θ (because $f(u, \theta, k, \mathcal{G}) \leq \theta$). To compute the maximal (θ, k)-persistent-degree intervals of u, we create two additional arrays $\tilde{\mathcal{MI}}_u$ and $\tilde{\mathcal{D}}_u$. For each $\mathcal{T}(i)$ in the line 14 of Algorithm 1, we generate an additional closed *point-interval*

$\tilde{\mathcal{MI}}_u(i) = [\mathcal{T}(i).first, \mathcal{T}(i).first]$ and calculate the degree $\tilde{\mathcal{D}}_u(i)$ which equals the number of surviving temporal edges at the timestamp $\mathcal{T}(i).first$. Note that this process can be done in linear time. Clearly, in the θ-shifted interval $[\mathcal{T}(i).first - \theta, \mathcal{T}(i).first]$, u has a degree $\tilde{\mathcal{D}}_u(i)$. Any maximal (θ, k)-persistent-degree interval must be a θ-shifted interval of a *point-interval* contained in $\tilde{\mathcal{MI}}_u$. Theorem 6 shows how to recognize the θ-length maximal (θ, k)-persistent-degree intervals from $\tilde{\mathcal{MI}}_u$ based on the arrays $\mathcal{D}_u$ and $\tilde{\mathcal{D}}_u$.

Theorem 6: For an interval $\tilde{\mathcal{MI}}_u(i) = [t_i, t_i]$, if $\tilde{\mathcal{D}}_u(i) = k$, $\mathcal{D}_u(i-1) < k$ and $\mathcal{D}_u(i) < k$, then $[t_i - \theta, t_i]$ is a θ-length maximal (θ, k)-persistent-degree interval of u.

Based on the above results, we can find all maximal (θ, k)-persistent-degree intervals of u and compute $f(u, \theta, k, \mathcal{G})$ if $f(u, \theta, k, \mathcal{G}) \le \theta$.

Complexity. Theorems 5 and 6 demonstrate that computing the degree persistence for all nodes in $\mathcal{G}$ can be done in linear time based on the meta-interval decomposition technique.

D. Updating degree persistence

For each node u, we can update the degree persistence of u when a temporal edge in $\mathcal{N}_u(\mathcal{G})$ is deleted or inserted based on the meta-interval decomposition. In the rest of this paper, we focus mainly on the case of $f(u, \theta, k, \mathcal{G}) > \theta$, and the proposed algorithms can easily extend to handle the $f(u, \theta, k, \mathcal{G}) \le \theta$ case. Let $\mathcal{MI}_u = \{(t_{s_1}, t_{e_1}), \cdots, (t_{s_h}, t_{e_h})\}$ and $\mathcal{D}_u$ be the arrays computed by Algorithm 1 in $\mathcal{N}_u(\mathcal{G})$ with parameter θ. Let $e = (u, v_i, t_i)$ be a temporal edge in $\mathcal{N}_u(\mathcal{G})$, and $[t_i, t_e]$ be its lifespan ($t_e \le t_i + \theta$). Note that t_e is not necessarily equal to $t_i + \theta$, because v_i may be a repeated neighbor of u (see Section III-B). Recall that by the meta-interval decomposition technique, there must exist an index l such that $\mathcal{MI}_u(l) = (t_{s_l}, t_{e_l})$ and $t_{s_l} = t_i$. If $t_{e_l} < t_e$, there is an index r ($l < r \le h$) such that $\mathcal{MI}_u(r) = (t_{s_r}, t_{e_r})$ and $t_{s_r} < t_e \le t_{e_r}$. Otherwise, we let $r = l$.

By the above definition, the lifespan of the temporal edge $e = (u, v_i, t_i)$ (i.e., $[t_i, t_e]$) *hits* every meta-interval in $\mathcal{I} = \{(t_{s_l} - \theta, t_{e_l}), \cdots, (t_{s_r} - \theta, t_{e_r})\}$. Clearly, for node u, the deletion (or insertion) of a temporal edge $e = (u, v_i, t_i)$ only affects the θ-persistent degree of u in each meta-interval in $\mathcal{I}$. Specifically, we have the following results.

Lemma 6: After deleting a temporal edge $e = (u, v_i, t_i)$ from $\mathcal{N}_u(\mathcal{G})$, we decrease $\mathcal{D}_u(j)$ by 1 for all $j = l, \cdots, r$, and keep $\mathcal{D}_u(j)$, for all $j \notin [l, r]$, unchanged. Based on the updated $\mathcal{D}_u$ array, we have $f(u, \theta, k, \mathcal{G}\backslash\{e\}) = \sum_{\mathcal{D}_u(i) \geqslant k} (t_{e_i} - t_{s_i}) + \theta$.

Lemma 7: After inserting a temporal edge $e = (u, v_i, t_i)$ back into $\mathcal{N}_u(\mathcal{G})\backslash\{e\}$, we increase $\mathcal{D}_u(j)$ by 1 for all $j = l, \cdots, r$ and keep $\mathcal{D}_u(j)$, for all $j \notin [l, r]$, unchanged. Based on the updated $\mathcal{D}_u$ array, we have $f(u, \theta, k, \mathcal{G}) = \sum_{\mathcal{D}_u(i) \geqslant k} (t_{e_i} - t_{s_i}) + \theta$.

Lemmas 6 and 7 show that we only need to update the array $\mathcal{D}_u$ when we delete an edge from $\mathcal{N}_u(\mathcal{G})$ or add back an edge e into $\mathcal{N}_u(\mathcal{G})\backslash\{e\}$. It is important to note that we do not destroy the original meta-interval structure of u computed in $\mathcal{N}_u(\mathcal{G})$ when updating u's degree persistence. Moreover, since the lifespan of each temporal edge e (i.e., $[t_i, t_e]$) is no larger than θ by our definition, the number of elements in $\mathcal{D}_u$ that are needed to be updated is bounded by θ (because $[t_i, t_e]$ hits at most θ meta-intervals). As a consequence, we can update the degree persistence in $O(\theta)$ time after obtaining the meta-interval decomposition.

Example 10: Reconsider the node v_1 in Fig. 1(a). Let $k = 2$ and $\theta = 3$. By the meta-interval decomposition, we obtain $\mathcal{D}_{v_1} = \{1, 2, 2, 1, 1, 2, 1\}$ and $\mathcal{MI}_{v_1} = \{(1, 2), (2, 3), (3, 5), (5, 6), (6, 7), (7, 9), (9, 10)\}$.

Algorithm 2 TGR ($\mathcal{G}$, θ, k, τ)

```
Input:   $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\theta$, $k$, and $\tau$
Output:  The reduced temporal graph $\tilde{\mathcal{C}}$
1: $\mathcal{Q} \leftarrow \emptyset$; $flag(u) \leftarrow 1$ for all $u \in \mathcal{V}$;
2: for all $u \in \mathcal{V}$ do
3:     $\{\mathcal{D}_u, \mathcal{MI}_u\} \leftarrow$ Meta-interval-Decomposition ($u$, $\mathcal{N}_u$, $\theta$);
4:     Compute $f(u, \theta, k, \mathcal{G})$ (by Theorems 5 and 6); $\tilde{d}_u \leftarrow f(u, \theta, k, \mathcal{G})$;
5:     if $\tilde{d}_u < \tau$ then
6:         $\mathcal{Q}.push(u)$; $flag(u) \leftarrow 0$;
7: while $\mathcal{Q} \neq \emptyset$ do
8:     $u \leftarrow \mathcal{Q}.pop()$;
9:     for all $(u, v, t) \in \mathcal{N}_u(\mathcal{G})$ do
10:        Let $[t, t_e]$ be the lifespan of the temporal edge $(u, v, t)$;
11:        Let $\mathcal{MI}_v = \{(t_{s_1}, t_{e_1}), \cdots, (t_{s_h}, t_{e_h})\}$;
12:        Suppose $t_{s_i} = t$; $j \leftarrow i$;
13:        while $j \le h$ and $t_{s_j} < t_e$ do
14:            $\mathcal{D}_v(j) \leftarrow \mathcal{D}_v(j) - 1$;
15:            if $\mathcal{D}_v(j) + 1 \ge k$ and $\mathcal{D}_v(j) < k$ then
16:                $\tilde{d}_v \leftarrow \tilde{d}_v - (t_{e_j} - t_{s_j})$; {// Update the degree persistence}
17:                if $\tilde{d}_v + (t_{e_j} - t_{s_j}) \ge \tau$ and $\tilde{d}_v < \tau$ then
18:                    $\mathcal{Q}.push(v)$; $flag(v) \leftarrow 0$;
19:            $j \leftarrow j + 1$;
20: return $\tilde{\mathcal{C}} \leftarrow$ Subgraphs induced by the nodes with $flag(u) = 1$;
```

Suppose that we delete the temporal edge $e = (v_1, v_2, 1)$. Since the lifespan of e is $[1, 3]$ and $(t_{s_1}, t_{e_1}) = (1, 2)$, we have $l = 1$. Since $t_{e_l} = 2 < t_e$ and $t_{s_2} < t_e \le t_{e_2}$, we have $r = 2$. Therefore, by Lemma 6, we decrease both $\mathcal{D}_{v_1}(1)$ and $\mathcal{D}_{v_1}(2)$ by 1. The degree persistence is updated by 7, because there are two meta-intervals with θ-persistent degree no less than k (intervals $(3, 5)$ and $(7, 9)$). Note that after deleing e, the maximal (θ, k)-persistent-degree intervals of v_1 are $[0, 5]$ and $[4, 9]$ by Definition 4. Thus, our result is consistent with the result obtained by definition. On the other hand, if we add back e into $\mathcal{N}_{v_1}(\mathcal{G}\backslash\{e\})$, we increase both $\mathcal{D}_{v_1}(1)$ and $\mathcal{D}_{v_1}(2)$ by 1 based on Lemma 7. We can verify that the degree persistence of v_1 is updated by 8, which is consistent with the result obtained by Definition 4.

E. The TGR *algorithm*

Armed with the results shown in the previous sections, we are now ready to present the TGR algorithm. The TGR algorithm is outlined in Algorithm 2. Specifically, Algorithm 2 first invokes Algorithm 1 to compute the meta-interval decomposition for every node $u \in \mathcal{V}$ (line 3). Then, the algorithm calculates the degree persistence based on the meta-interval decomposition, and pushes all the nodes with degree persistence less than τ into the queue $\mathcal{Q}$ (lines 4-6). Subsequently, the algorithm iteratively removes the nodes with degree persistence less than τ (lines 7-19). In each iteration, the algorithm pops a node from $\mathcal{Q}$ (line 8), and traverses its temporal neighborhood $\mathcal{N}_u(\mathcal{G})$. For each temporal edge $e = (u, v, t) \in \mathcal{N}_u(\mathcal{G})$, the algorithm updates the degree persistence of v after deleing e based on Lemma 6 (lines 10-19). If the updated degree persistence of a node is smaller than τ, the algorithm pushes it into the queue $\mathcal{Q}$ (line 18). The algorithm terminates when $\mathcal{Q}$ is empty. When the algorithm terminates, all the remaining nodes in $\mathcal{G}$ have degree persistence no less than τ. The complexity of Algorithm 2 is analyzed in Theorem 7.

Theorem 7: The time and space complexity of Algorithm 2 is $O(\theta m)$ and $O(m)$ respectively.

IV. Algorithm for PC search

In this section, we propose a tractable algorithm for the PC search problem. Our algorithm includes two stages. First, we invoke Algorithm 2 to reduce the temporal graph. Then, in the reduced temporal graph, we propose an efficient branch and bound algorithm with several powerful pruning rules to compute the largest (θ, τ)-persistent k-core. Below, we first

Algorithm 3 Meta-Interval-Intersection ($\mathcal{R}$, k)

```
1: Let $\mathcal{MIR} \leftarrow \{(-\infty, +\infty)\}$; $len \leftarrow +\infty$;
2: for all $u \in \mathcal{R}$ do
3:     Let $h \leftarrow |\mathcal{MI}_u|$, and $\mathcal{MI}_u = \{(t_{s_1}, t_{e_1}), \cdots, (t_{s_h}, t_{e_h})\}$;
4:     Let $p \leftarrow |\mathcal{MIR}|$, and $\mathcal{MIR} = \{(t_{l_1}, t_{r_1}), \cdots, (t_{l_p}, t_{r_p})\}$;
5:     $\mathcal{MIT} \leftarrow \emptyset$; $tlen \leftarrow 0$; $j \leftarrow 1$;
6:     for $i = 1$ to $h$ do
7:         if $\mathcal{D}_u(i) \geq k$ then
8:             while $j \leq p$ and $t_{r_j} \leq t_{s_i}$ do
9:                 $j \leftarrow j + 1$;
10:            while $j \leq p$ and $t_{r_j} \leq t_{e_i}$ do
11:                $l \leftarrow \max\{t_{l_j}, t_{s_i}\}$; $r \leftarrow t_{r_j}$;
12:                if $l < r$ then
13:                    $\mathcal{MIT} \leftarrow \mathcal{MIT} \cup \{(l, r)\}$; $tlen \leftarrow tlen + (r - l)$;
14:                $j \leftarrow j + 1$
15:            if $j \leq p$ and $t_{l_j} \leq t_{e_i}$ then
16:                $l \leftarrow \max\{t_{l_j}, t_{s_i}\}$; $r \leftarrow t_{e_i}$;
17:                if $l < r$ then
18:                    $\mathcal{MIT} \leftarrow \mathcal{MIT} \cup \{(l, r)\}$; $tlen \leftarrow tlen + (r - l)$;
19:    $\mathcal{MIR} \leftarrow \mathcal{MIT}$; $len \leftarrow tlen$;
20: return $\{\mathcal{MIR}, len + \theta\}$;
```

propose three key sub-algorithms which will be frequently invoked in the branch and bound algorithm.

A. Three key sub-algorithms

Compute common meta-intervals. Here we propose an algorithm to compute the common meta-intervals for a set of nodes which will be utilized to evaluate the core persistence. The algorithm is outlined in Algorithm 3. Given a set of nodes $\mathcal{R}$, Algorithm 3 outputs the common meta-intervals such that in each of those meta-intervals every node in $\mathcal{R}$ has a θ-persistent degree no less than k. Algorithm 3 first initializes a common meta-interval array $\mathcal{MIR} = \{(-\infty, +\infty)\}$ for an empty node set (line 1). Then, the algorithm iteratively traverses a node $u \in \mathcal{R}$, and takes the interval-intersection between the common meta-interval array $\mathcal{MIR}$ and the meta-interval array of u (lines 2-19). In each iteration, the algorithm updates $\mathcal{MIR}$ by the resulting meta-intervals (line 19). Note that for each node $u \in \mathcal{R}$, we only need to consider the meta-intervals $\mathcal{MI}_u(i)$ for all $i = 1, \cdots, h$ such that $\mathcal{D}_u(i) \geq k$ (line 7). For convenience, we refer to those meta-intervals as *active* meta-intervals. This is because only the active meta-intervals will be used to compute the core persistence. Algorithm 3 also outputs len which is equal to the total length of the common meta-intervals plus θ (line 20). Note that if $\mathcal{R}$ forms a connected k-core and all $\mathcal{D}_u$ for $u \in \mathcal{R}$ have been updated w.r.t. $\mathcal{R}$, we can easily obtain that $F(\theta, k, \mathcal{R}) = len + \theta$ (similar to the results of Theorem 5).

Let N_u be the number of meta-intervals of u with $\mathcal{D}_u(i) \geq k$ for $i = 1, \cdots, h$ (i.e., N_u denotes the number of active meta-intervals.). Then, we analyze the time complexity of Algorithm 3 in Theorem 8.

Theorem 8: Let $N_R = \sum_{u \in \{\mathcal{R}\}} N_u$. The time complexity of Algorithm 3 is $O(|\mathcal{R}| N_R)$.

Remove nodes. We present an algorithm to maintain the degree persistence of the nodes after deleting a node u, because our branch and bound algorithm frequently invokes the node-removal procedure to prune the search space. The algorithm is described in Algorithm 4. In Algorithm 4, the parameter u is the node that we want to delete, and the parameter $\mathcal{S}$ denotes a set of nodes that cannot be deleted. After removing u, we only need to maintain the degree persistence of the neighbors of u. Thus, the algorithm iteratively traverses a temporal edge in $\mathcal{N}_u(\mathcal{G})$. In each iteration, the algorithm update the degree persistence of u's neighbors based on the result shown in Lemma 6 (lines 3-14). The algorithm makes use of a set $\mathcal{R}$ to record the neighbors whose updated degree persistence is smaller than τ (lines 10-11). Since the nodes in $\mathcal{R}$ cannot be contained in the (θ, τ)-persistent k-core, all nodes in $\mathcal{R}$ can

Algorithm 4 Remove-Node (u, $\mathcal{S}$, k, τ)

```
1: $flag \leftarrow 1$; $\mathcal{R} \leftarrow \emptyset$;
2: for all $(u, v, t) \in \mathcal{N}_u(\mathcal{G})$ do
3:     Let $[t, t_e]$ be the lifespan of $(u, v, t)$;
4:     Let $\mathcal{MI}_v = \{(t_{s_1}, t_{e_1}), \cdots, (t_{s_h}, t_{e_h})\}$;
5:     Suppose $t_{s_i} = t$; $j \leftarrow i$;
6:     while $j \leq h$ and $t_{s_j} < t_e$ do
7:         $\mathcal{D}_v(j) \leftarrow \mathcal{D}_v(j) - 1$;
8:         if $\mathcal{D}_v(j) + 1 \geq k$ and $\mathcal{D}_v(j) < k$ then
9:             $\tilde{d}_v \leftarrow \tilde{d}_v - (t_{e_j} - t_{s_j})$; {// Update the degree persistence}
10:            if $\tilde{d}_v + (t_{e_j} - t_{s_j}) \geq \tau$ and $\tilde{d}_v < \tau$ then
11:                $\mathcal{R} \leftarrow \mathcal{R} \cup \{v\}$;
12:                if $v \in \mathcal{S}$ then
13:                    $flag \leftarrow 0$;
14:        $j \leftarrow j + 1$;
15: return $\{\mathcal{R}, flag\}$;
```

Algorithm 5 Add-Node (u, k)

```
1: for all $(u, v, t) \in \mathcal{N}_u(\mathcal{G})$ do
2:     Let $[t, t_e]$ be the lifespan of $(u, v, t)$;
3:     Let $\mathcal{MI}_v = \{(t_{s_1}, t_{e_1}), \cdots, (t_{s_h}, t_{e_h})\}$;
4:     Suppose $t_{s_i} = t$; $j \leftarrow i$;
5:     while $j \leq h$ and $t_{s_j} < t_e$ do
6:         $\mathcal{D}_v(j) \leftarrow \mathcal{D}_v(j) + 1$;
7:         if $\mathcal{D}_v(j) \geq k$ and $\mathcal{D}_v(j) - 1 < k$ then
8:             $\tilde{d}_v \leftarrow \tilde{d}_v + (t_{e_j} - t_{s_j})$; {// Update the degree persistence}
9:         $j \leftarrow j + 1$;
```

be deleted. If $\mathcal{R}$ consists of a node in $\mathcal{S}$, the algorithm returns false, indicating that we cannot delete node u (lines 12-13). The time complexity of Algorithm 4 is shown Theorem 9.

Theorem 9: The time complexity of Algorithm 4 is $O(\theta|\mathcal{N}_u(\mathcal{G})|)$.

Add nodes. Similarly, we present Algorithm 5 to maintain the degree persistence of the nodes after adding a node u. This is because after deleting the nodes by the branch and bound algorithm, we need to add back the removed nodes to recover the search state (i.e., the backtracking procedure). Algorithm 5 is based on the results shown in Lemma 7. Since the general procedure of the algorithm is very similar to Algorithm 4, we omit the details. The time complexity of Algorithm 5 is $O(\theta|\mathcal{N}_u(\mathcal{G})|)$.

B. The branch and bound framework

Equipped with the three sub-algorithms, we propose the branch and bound algorithm and the PC search algorithm in Algorithm 6 and Algorithm 7 respectively. In Algorithm 7, we first call Algorithm 2 to reduce the temporal graph (line 2), and then invokes Algorithm 6 to compute the maximum (θ, τ)-persistent k-core. Below, we details Algorithm 6.

The key idea of Algorithm 6. Let $\tilde{\mathcal{C}}$ be the input graph (i.e., the reduced temporal graph) of the algorithm. First, we compute all the maximal connected components of $\tilde{\mathcal{C}}$ (line 2). For each component $\mathcal{C}$, we invoke Algorithm 3 to compute the total length of the common meta-intervals of the nodes in $\mathcal{C}$, denoted by len (line 6). If $len + \theta \geq \tau$, $\mathcal{C}$ must be a (θ, τ)-persistent k-core by Definition 3 (line 7). Otherwise, we divide the search space into two disjoint subspaces by randomly picking a node u in $\mathcal{C}$ (line 10): (1) the subspace of excluding u, and (2) the subspace of including u. Clearly, any (θ, τ)-persistent k-core must be contained in one of these two subspaces. We make use of a set $\mathcal{S}$ to maintain all the included nodes.

Subspace-I: excluding u. In this subspace, the resulting (θ, τ)-persistent k-cores cannot contain u. Thus, we invoke Algorithm 4 to delete u (line 15). Since removing u may trigger the deletions of the other nodes, Algorithm 4 returns all these deleted nodes $\mathcal{R}$ and an indicator variable $flag$. If

Algorithm 6 Branch-Bound ($\tilde{\mathcal{C}}$, $\mathcal{S}$, k, τ)

```
1: if $|\tilde{\mathcal{C}}| < size$ then return;
2: $\mathcal{CM} \leftarrow$ All-Components ($\tilde{\mathcal{C}}$); {// compute the connected components}
3: for all $\mathcal{C} \in \mathcal{CM}$ do
4:     if $\mathcal{S} \subseteq \mathcal{C}$ then
5:         if $|\mathcal{C}| < size$ then return;
6:         $\{\mathcal{MIC}, lc\} \leftarrow$ Meta-Interval-Intersection($\mathcal{C}, k$);
7:         if $lc \geq \tau$ then $\mathcal{R}^* \leftarrow \mathcal{C}$; $size \leftarrow |\mathcal{C}|$; return;
8:         $\{\mathcal{MIS}, ls\} \leftarrow$ Meta-Interval-Intersection($\mathcal{S}, k$);
9:         if $ls \geq \tau$ and $|\mathcal{S}| < |\mathcal{C}|$ then
10:            Randomly pick a node $u \in \mathcal{C} \backslash \mathcal{S}$;
11:            $f_u \leftarrow \tilde{d}_u$; $\tilde{d}_u \leftarrow 0$; {// $f_u$ records the degree persistence}
12:            $\mathcal{Q} \leftarrow \emptyset$; $\mathcal{Q}.push(u)$; $\mathcal{A} \leftarrow \emptyset$;
13:            while $\mathcal{Q} \neq \emptyset$ do
14:                $u \leftarrow \mathcal{Q}.pop()$; $\mathcal{A} \leftarrow \mathcal{A} \cup \{u\}$;
15:                $\{\mathcal{H}, flag\} \leftarrow$ Remove-Node ($u$, $\mathcal{S}$, $k$, $\tau$);$\mathcal{Q} \leftarrow \mathcal{Q} \cup \mathcal{H}$;
16:                if $flag = 0$ then break;
17:            if $flag = 1$ then Branch-Bound ($\mathcal{C} \backslash \mathcal{A}$, $S$, $k$, $\tau$);
18:            $\tilde{d}_u \leftarrow f_u$;
19:            for all $v \in \mathcal{A}$ do
20:                Add-Node ($v$, $k$);
21:            $\mathcal{S} \leftarrow \mathcal{S} \cup \{u\}$; {// the case of including $u$}
22:            $\{\mathcal{MIS}, ls\} \leftarrow$ Meta-Interval-Intersection($\mathcal{S}, k$);
23:            if $ls \geq \tau$ then
24:                $\mathcal{B} \leftarrow \emptyset$; $\mathcal{Q} \leftarrow \emptyset$;
25:                for all $v \in \mathcal{C} \backslash \mathcal{S}$ do
26:                    $\{\mathcal{MIV}, lv\} \leftarrow$ Meta-Interval-Intersection($\mathcal{S} \cup \{v\}, k$);
27:                    if $lv < \tau$ then
28:                        $f_v \leftarrow \tilde{d}_v$; $\tilde{d}_v \leftarrow 0$;
29:                        $\mathcal{B} \leftarrow \mathcal{B} \cup \{v\}$;
30:                $\mathcal{Q} \leftarrow \mathcal{B}$; $\mathcal{A} \leftarrow \emptyset$;
31:                while $\mathcal{Q} \neq \emptyset$ do
32:                    $v \leftarrow \mathcal{Q}.pop()$; $\mathcal{A} \leftarrow \mathcal{A} \cup \{v\}$;
33:                    $\{\mathcal{H}, flag\} \leftarrow$ Remove-Node ($u$, $\mathcal{S}$, $k$, $\tau$);$\mathcal{Q} \leftarrow \mathcal{Q} \cup \mathcal{H}$;
34:                    if $flag = 0$ then break;
35:                if $flag = 1$ then Branch-Bound ($\mathcal{C} \backslash \mathcal{A}$, $S$, $k$, $\tau$);
36:                for all $v \in \mathcal{B}$ do
37:                    $\tilde{d}_v \leftarrow f_v$;
38:                for all $v \in \mathcal{A}$ do
39:                    Add-Node ($v$, $k$);
40:                $\mathcal{S} \leftarrow \mathcal{S} \backslash \{u\}$;
```

Algorithm 7 PC ($\mathcal{G}$, θ, k, τ)

```
Input:  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\theta$, $k$, and $\tau$
Output: The maximum $(\theta, \tau)$-persistent $k$-core
1: $\mathcal{R}^* \leftarrow \emptyset$; $size \leftarrow 0$; {// global variables}
2: $\tilde{\mathcal{C}} \leftarrow$ TGR ($\mathcal{G}$, $\theta$, $k$, $\tau$);
3: Branch-Bound($\tilde{\mathcal{C}}, \emptyset, k, \tau$);
4: return $\mathcal{R}^*$;
```

$flag = 1$, we remove all the nodes in $\mathcal{R}$ from $\mathcal{C}$, and then recursively invoke the branch and bound algorithm in the reduced subgraph $\mathcal{C} \backslash \mathcal{R}$ (line 17). Otherwise, we can safely prune this subspace, because we fail to delete u, implying that there is no (θ, τ)-persistent k-core in this subspace (line 16). After this step, we backtrack to the search space before removing u. This can be done by invoking Algorithm 5 to add back the deleted nodes (lines 18-20).

Subspace-II: including u. In this subspace, we invoke the branch and bound algorithm subject to the constraint that the resulting (θ, τ)-persistent k-cores must contain u. After that, we backtrack to the search space before including u, which can be done by removing u from $\mathcal{S}$ (lines 21-40).

The algorithm uses a variable $size$ to record the largest size of the (θ, τ)-persistent k-cores computed so far. If $|\mathcal{C}| < size$, we can prune this search space, because the (θ, τ)-persistent k-cores contained in this space must be smaller than $size$ (line 5). Below, we propose several effective pruning rules to further reduce the search space.

Upper-bound pruning. Let ls be the total length of the common meta-intervals of the nodes in $\mathcal{S}$ calculated by Algorithm 3. Since the resulting (θ, τ)-persistent k-cores must contain $\mathcal{S}$, $ls + \theta$ is an upper bound of the core persistence of the (θ, τ)-persistent k-core. For a component $\mathcal{C}$ and the included-node set $\mathcal{S}$, we can first compute ls for $\mathcal{S}$. If $ls < \tau$, we can prune the current search space, because the core persistence of the (θ, τ)-persistent k-cores in this space must be smaller than τ (lines 8-9 in Algorithm 6).

Excluding-node pruning. Consider the case of excluding a node u. After invoking Algorithm 4, if $\mathcal{R} \neq \emptyset$ and $flag = 1$, we may further delete the node whose degree persistence is smaller than τ when deleting $\mathcal{R}$. This is because the deletion of nodes in $\mathcal{R}$ may trigger the degree persistence of the neighbors of the nodes in $\mathcal{R}$ smaller than τ. As a result, we can iteratively remove the nodes until all the nodes with degree persistence no less than τ (similar to the TGR procedure). To this end, we create a queue $\mathcal{Q}$ that initially includes all the nodes in $\mathcal{R}$. Then, we pop a node v from $\mathcal{Q}$, and invoke Algorithm 4 to delete v and also to update the degree persistence of v's neighbors. Suppose that Algorithm 4 returns $\mathcal{R}'$ and $flag'$ in an iteration. If $flag' = 0$, we are able to prune this subspace, because we cannot delete u in this case (line 16). Otherwise, we add all nodes in $\mathcal{R}'$ into $\mathcal{Q}$. Then, we iteratively perform the same procedure until $\mathcal{Q} = \emptyset$ (lines 12-16). This pruning rule can largely reduce the number of nodes of the input component, thus substantially reduce the search time as verified in our experiments.

Including-node pruning. Consider the case of including a node u. Assume that the current input component is $\mathcal{C}$ and the included-node set is $\mathcal{S}$. We have several pruning tricks in this case. First, we can apply the upper-bound pruning rule to prune the unpromising search space. Specifically, we compute the total length of the common meta-intervals of the nodes in $\mathcal{S} \cup \{u\}$, and then we make use of the upper-bound pruning rule to prune the search space (lines 21-23). Second, for each node $v \in \mathcal{C}$, we compute the total length of the common meta-intervals of the nodes in $\mathcal{S} \cup \{u, v\}$, denoted by lv. If $lv + \theta < \tau$, we can conclude that v cannot be contained in any (θ, τ)-persistent k-core that already consists of $\mathcal{S} \cup \{u\}$. We use a set $\mathcal{B}$ to record all those nodes (lines 24-29). Clearly, we can delete all the nodes in $\mathcal{B}$. Note that the deletion of the nodes in $\mathcal{B}$ may result in the other nodes that must also be removed. As a consequence, we can use the same method as used in the excluding-node pruning to prune the unpromising search spaces (lines 30-35). As demonstrated in the experiments, this pruning rule can further reduce the search time significantly.

Since Algorithm 6 does not miss any search space, the results obtained by our algorithm are correct. Below, we analyze the complexity of our algorithm.

Complexity analysis. Since the problem of finding the largest (θ, τ)-persistent k-core is NP-hard, the worst-case time complexity of Algorithm 7 is exponential. More specifically, there are at most $O(2^{\tilde{n}})$ subspaces (because the recursion tree of our algorithm is clearly a binary tree), where $\tilde{n}$ denotes the size of the largest connected component of the reduced temporal graph. In each subspace, the algorithm takes at most $O(\theta m)$ time to delete and add nodes, and $O(\tilde{n}N)$ time to compute the common meta-intervals, where N denotes the maximum number of active meta-intervals in the reduced temporal graph. Therefore, the worst-case time complexity of Algorithm 7 is $O(2^{\tilde{n}}(\tilde{n}N + \theta m)n_c)$, where n_c denotes the number of connected component of the reduced temporal graph. Since $\tilde{n}$ is typically not very large and the proposed pruning rules are very effective, the proposed algorithm is tractable in many real-world large-scale temporal graphs. In the experiments, we will show that our algorithm is scalable to handle the temporal graph with more than 1 million nodes and 10 million edges.

Enumerating all persistent communities. The proposed branch and bound framework can be easily extended to enumerate all (θ, τ)-persistent k-cores. In particular, we remove the size constraint in Algorithm 6, and add a constraint of $|\mathcal{C}| = |\mathcal{S}|$ for maximal property testing. Specifically, in

TABLE I
DATASETS AND PARAMETERS (DEFAULT VALUE AND RANGE)

Datasets	n	m	θ	k	τ
Chess	7,301	65,053	7, $[5, 9]$	4, $[3, 6]$	16, $[12, 20]$
Linux	63,399	1,028,233	7, $[3, 11]$	10, $[6, 14]$	30, $[22, 38]$
Enron	87,273	1,134,990	6, $[4, 8]$	4, $[3, 7]$	22, $[18, 26]$
DBLP	1,570,522	11,596,310	4, $[2, 6]$	8, $[4, 12]$	18, $[14, 22]$

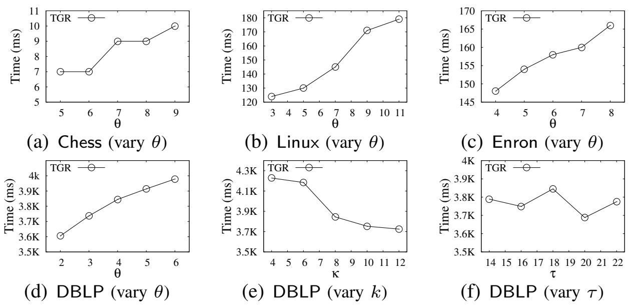
Algorithm 6, we delete lines 1 and 5. In line 7, we add a constraint $|\mathcal{C}| = |\mathcal{S}|$. If $|\mathcal{C}| = |\mathcal{S}|$ and $lc \geq \tau$, we output $\mathcal{C}$ as a (θ, τ)-persistent k-core. The reasons are as follows. First, $\mathcal{C}$ is connected and the core persistence of $\mathcal{C}$ is no less than τ. Second, since $|\mathcal{C}| = |\mathcal{S}|$, there is no candidate node that can be added into $\mathcal{S}$, thus $\mathcal{S} = \mathcal{C}$ satisfies the maximal property. As a result, $\mathcal{C}$ is a (θ, τ)-persistent k-core.

V. EXPERIMENTS

We conduct comprehensive experiments to evaluate the efficiency and effectiveness of the proposed algorithms. We implement the TGR algorithm (Algorithm 2) to prune the temporal graph. For the PC search problem, we implement four various algorithms: Basic, BB+, BB++, and BB-All. All these algorithms first invoke TGR to reduce the temporal graph, and then call Algorithm 6 to find the maximum persistent k-core. The differences among these algorithms are the pruning rules used in Algorithm 6. Specifically, Basic does not use any pruning rule presented in Section IV-B. We make use of this algorithm as the baseline for efficiency testings, because no existing algorithm can be used to find persistent k-cores. BB+ denotes the Basic algorithm with upper-bound pruning. BB++ is the BB+ algorithm with excluding-node pruning, and BB-All denotes BB++ with including-node pruning. All algorithms are implemented in C++. All experiments are conducted on a computer with two 3.46GHz Xeon CPUs and 64GB memory running Red Hat Enterprise Linux 6.4.

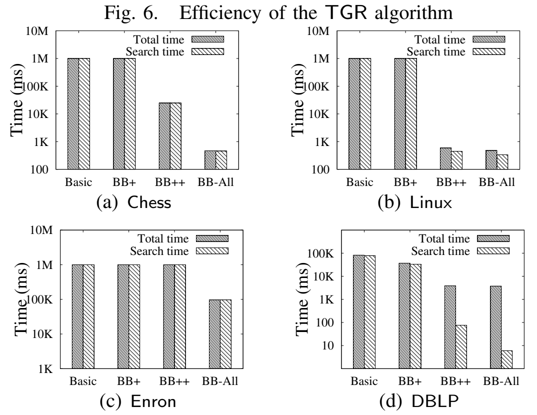
Datasets. We use four different types of real-world temporal networks in the experiments. The detailed statistics of our datasets are summarized in Table I. All the datasets are downloaded from (http://konect.uni-koblenz.de). Chess is a temporal network in which a temporal edge (u, v, t) represents a chess game between the players u and v at time t. Linux is a temporal communication network of the Linux kernel mailing list, where a temporal edge (u, v, t) denotes a reply from a user u to v at time t. Enron is a temporal email network between employees of Enron between 1999 and 2003. DBLP is a temporal collaboration network of authors.

Parameters. There are three parameters in our community models which are θ, k and τ. Note that if θ is too large, the persistent constraint of our models will be very loose, and thus the resulting communities are not the qualified persistent communities. On the other hand, if θ is too small, there will be no persistent community. Similarly, if k and/or τ are too large, there will be no answer, while if k and/or τ are too small, the obtained communities are not cohesive and are also not persistent. We will study how these parameters affect the community size in the following experiments. Table I shows the ranges and default values of the parameters used in the experiments. The default values of our parameters are estimated based on the following method. First, we compute the average gap between too consecutive temporal edges and use it as an estimation of θ. Then, we divide the temporal graph into several θ-length snapshots, and compute the maximum k-core number ($k_{\max}$) on each snapshot. The parameter k is estimated by the average $k_{\max}$ values over all snapshots. After determining θ and k, we invoke the meta-interval decomposition method to calculate the degree persistence for each node, and use the average degree persistence as an estimation of τ. Note that since the time scales of the datasets are diverse, the parameter settings are also different for various datasets. When varying a certain parameter, the values for all the other parameters are set to their default values.



(a) Chess (vary θ) (b) Linux (vary θ) (c) Enron (vary θ)
(d) DBLP (vary θ) (e) DBLP (vary k) (f) DBLP (vary τ)

Fig. 6. Efficiency of the TGR algorithm



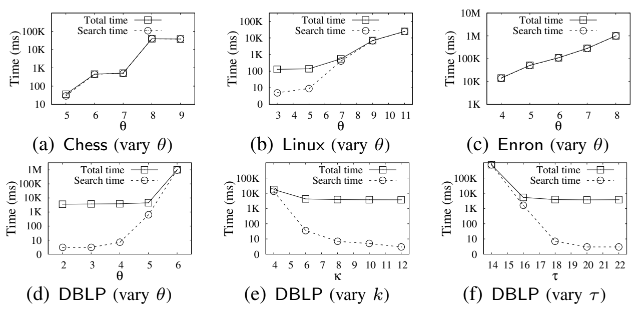
(a) Chess (b) Linux
(c) Enron (d) DBLP

Fig. 7. Comparison of different PC search algorithms

A. Efficiency testings

Efficiency of the TGR algorithm. In this experiment, we test the efficiency of the TGR algorithm (Algorithm 2) with varying θ, k, and τ. The results of varying θ in all the datasets are shown in Figs. 6(a-d). As can be seen, the running time of the algorithm increases with increasing θ in all datasets. This is because the time complexity of the TGR algorithm is $O(\theta m)$, which is linearly dependent on θ. For the results of varying k and τ, we only report the results in the DBLP dataset in Figs. 6(e-f), and similar results can also be observed in the other datasets. From Fig. 6(e), we can see that the running time of TGR decreases when k increases. This is because for a large k, the algorithm can quickly remove a large number of unpromising nodes, and thus significantly reduces the running time of the iterative-refinement procedure. As observed in Fig. 6(f), the efficiency of TGR seems not very sensitive w.r.t. τ. In DBLP, the fluctuation of the running time of our algorithm is around 0.1 seconds. Additionally, we can see that the TGR algorithm is very efficient in all testings. For example, in DBLP, TGR takes less than 4 seconds under the default parameters setting. These results confirm the complexity analysis shown in Section III-E.

Comparison of different PC search algorithms. In this experiment, we compare the efficiency of different PC search algorithms. The results are reported in Fig. 7. For all the algorithms, the search time is the time spent in the branch and bound procedure (excluding the time taken by the TGR procedure), while the total time denotes total running time of the algorithm. Since some of the PC search algorithms may not be tractable, we limit the maximal running time by 1000 seconds for all algorithms. As can be seen, both Basic and BB+ are very costly, which are intractable in the Chess, Linux, and Enron datasets. BB++ is very efficient in the Chess, Linux, and DBLP datasets. It is at least two orders of magnitude faster than both Basic and BB+. The best algorithm is the BB-All



(a) Chess (vary θ) (b) Linux (vary θ) (c) Enron (vary θ)
(d) DBLP (vary θ) (e) DBLP (vary k) (f) DBLP (vary τ)

Fig. 8. Efficiency of BB-All with varying parameters

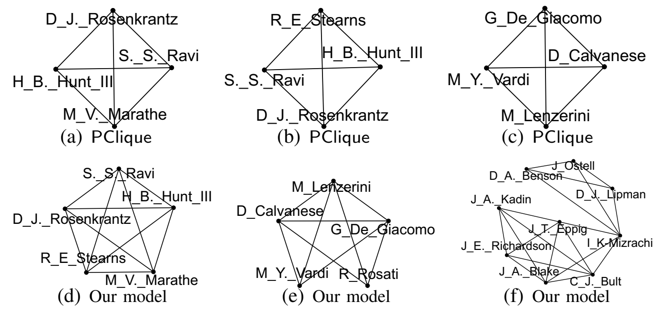
algorithm, which is very efficient in all datasets. Moreover, we can see that the search time of BB-All is generally one order of magnitude faster than BB++. For example, in DBLP, BB-All takes only 6ms, whereas BB++ consumes 76ms. These results indicate that both the excluding-node pruning and including-node pruning are very effective to prune the search space.

Efficiency of BB-All **with varying parameters.** Since BB-All is significantly more efficient than the other algorithms, in this experiment we study how the parameters affect the performance of the BB-All algorithm. Figs. 8(a-d) depict the results with varying θ in all the datasets. As shown in Figs. 8(a-d), both the total time and search time increase with increasing θ. This is because there could be a large number of (θ, τ)-persistent k-cores for a large θ, and thus the search space will significantly increase with increasing θ. The total time is dominated by the search time in the first three datasets. However, in DBLP, when θ is small, the total time is dominated by the TGR procedure. This is because when θ is small, the size of the reduced temporal graph in DBLP could be very small, and thus the branch and bound procedure will be much faster. Figs. 8(e-f) report the results in DBLP when varying k and τ respectively. Similar results can also be observed in the other datasets. We can observe that both the total time and search time decrease with increasing k or τ. This is because for a large k or τ, the size of the reduced temporal graph could be small, and therefore the branch and bound search algorithm will be very efficient. In most parameter testings, the running time of BB-All is no large than 15 minutes. These results indicate that BB-All is indeed tractable for real-world large temporal graphs.

B. Effectiveness testings

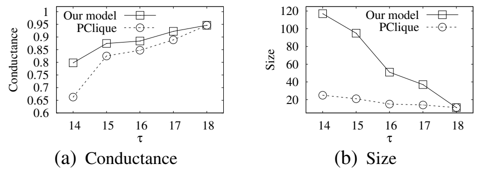
In this experiment, we implement two baselines for comparison: KCore and PClique. KCore is an intuitive baseline which first finds all the persistent edges (with persistence larger than the threshold τ), and then computes the k-core in the temporal subgraph induced by all persistent edges. Note that we can apply the same method as used for computing the core persistence of a temporal subgraph (see Definition 2) to calculate the persistence of an edge, because each edge can be deemed as a 1-core. PClique is a persistent clique model which slightly improves the state-of-the-art temporal clique model [7] to fully capture the persistence of a community. Specifically, PClique makes use of the same method as used in Eq. (1) to aggregate all persistent intervals of the temporal cliques [7]. We omit the detailed definition of PClique due to the space limit.

Case study. Fig. 9 shows the communities obtained by our model and PClique using parameters $\theta = 2$ (i.e., 2 years), $k = 3$, and $\tau = 17$ (i.e., 17 years). Note that under the same parameter setting, the community obtained by KCore contains 161 authors (not shown[2]) that come from different research



(a) PClique (b) PClique (c) PClique
(d) Our model (e) Our model (f) Our model

Fig. 9. Comparison of various models ($\theta = 2$, $k = 3$, $\tau = 17$)



(a) Conductance (b) Size

Fig. 10. Quality between our model and PClique (vary τ, $k = 8, \theta = 4$).

areas including theoretical computer sciences, bioinformatics, and so on. In fact, given that $\theta = 2$ and $\tau = 17$, the KCore algorithm can obtain a 2-core including 917 nodes, a 3-core containing 161 nodes, and there does not exist high-order cores (i.e., 3 is the maximum k). Clearly, in DBLP, all the 161 authors found by KCore cannot form a persistent and compact community, as these authors come from diverse research areas. This result indicate that KCore is ineffective to identify persistent communities. Instead, as shown in Fig. 9, both PClique and our model tend to find persistent and compact communities. For example, in Figs. 9(b) and (d), the communities found by both PClique and our model include several famous researchers in the theoretical computer science area who persistently coauthor a paper in the past few decades. Compared to PClique, our model is more effective to detect persistent communities. For instance, in Figs. 9(a-b), the results obtained by PClique are redundant, and both of them can only reveal a partial persistent community of the result obtained by our model (Fig. 9(d)). In effect, the five researchers in Fig. 9(d) continuously coauthor papers in the last few decades, thus they should be included in a persistent community. Similar results can also be observed in Figs. 9(c) and (e). In addition, our model can find a large persistent community in the bioinformatics area that cannot be found by PClique. The reason could be that the clique constraint in PClique may be too strong, thus ruling out many interesting persistent communities.

Quality of different models. Conductance is a well-known metric to measure the quality of a community [15]. Here we evaluate the conductance of the community obtained by our model and PClique. Note that we do not show the results of KCore, because it is ineffective to identify persistent communities. Fig. 10 shows the conductance and the size of the communities obtained by our model and PClique with varying τ. Similar results can also be observed using the other parameters. As can be seen, our model consistently outperforms PClique based on the conductance metric. Also, we can see that the community size of our model is no smaller than that of PClique as desired. These results further confirm the effectiveness of the proposed model.

VI. Related work

Temporal graph mining. Our work is related to the problem of mining temporal graphs, which has attracted much attention in the database community. For example, Bansal et al. [16] investigated a problem of finding stable keywords clusters in

[2]We do not visualize this community, as it is too large to show in a figure.

a collection of blog posts for specific temporal intervals. A cluster in [16] is modeled as a bi-connected component of the keyword graph, which could be less cohesive for natural graphs. Wu et al. [17] studied the shortest path problem in temporal graphs. Yang et al. [18] proposed an algorithm to detect frequent changing components in temporal graph. Huang et al. [19] investigated the minimum spanning tree (MST) problem in temporal graphs. Gurukar et al. [20] proposed a model called communication motifs to identify the recurring subgraphs that have similar sequence of information flow. More recently, Wu et al. [21] proposed an efficient algorithm to answer the reachability and time-based path queries in a temporal graph. Yang et al. [22] studied a problem of finding a set of diversified quasi-cliques from a temporal graph. Wu et al. [13] proposed a temporal k-core model based on the counts of temporal edges. Their temporal k-core model does not consider the persistence of the community, thus it cannot be used for detecting persistent communities in a temporal graph. Ma et al. [23] investigated a dense subgraph problem in temporal graphs, in which the temporal edges are associated positive and negative weights. Again, the dense subgraph proposed in their work did not consider the persistence of the community, thus cannot be applied to our problem. To (partially) capture the persistence, Viard et al. [7] proposed a temporal clique model as well as an algorithm using $O(2^n n^2 m^3 + 2^n n^3 m^2)$ time and $O(2^n nm^2)$ space to find all the temporal cliques. Himmel et al. [10] proposed an improved algorithm to enumerate all temporal cliques. Their algorithm, however, is still costly to handle large temporal graphs. Unlike the temporal clique model, our persistent community model is based on the concept of k-core, and the proposed algorithms can be scalable to handle large temporal graphs. In addition, temporal networks were also studied in the physics community and a comprehensive survey can be found in [24].

Cohesive subgraph mining. Our work is also related to cohesive subgraph mining. There are a number of cohesive subgraph models that have been proposed in the literature. Notable cohesive subgraph models include maximal clique [25], k-core [8], [26], k-truss [2], [3], maximal k-edge connected subgraph (MkCS) [27], [28], locally dense subgraph [4], influential community [12], and so on. To mining the maximal cliques from a disk-resident graph, Cheng et al. [25] proposed an I/O-efficient algorithm based on a concept of H-index graph [29]. The same group also proposed several I/O-efficient algorithms to compute the k-core and k-truss efficiently [1], [2]. Huang et al. presented a different k-truss model called k-truss community which can be used to find overlapped communities. To compute the MkCS efficiently, Chang et al. [28] proposed a linear-time algorithm which significantly improves an algorithm proposed by Zhou et al. [27]. More recently, Qin et al. [4] proposed a locally dense subgraph model as well as an efficient algorithm to identify all the disjoint dense subgraphs of a graph. Li et al. [12] introduced an influential community model which can capture both the influence and cohesiveness of the community. All the above-mentioned cohesive subgraph models do not consider the temporal information, thus they cannot be used to model persistent communities.

VII. Conclusion

In this paper, we propose a novel community model called (θ, τ)-persistent k-core to capture the persistence of a community in temporal networks. We prove that the problem of finding the maximum (θ, τ)-persistent k-core is NP-hard. A novel temporal graph reduction technique and a branch and bound algorithm are proposed to solve this problem efficiently. We show that the proposed algorithms can also be used to enumerate all (θ, τ)-persistent k-cores. Comprehensive experiments in real-life temporal networks demonstrate the efficiency and scalability of our algorithms, as well as the effectiveness of our model.

Acknowledgement. Rong-Hua Li was partially supported by the NSFC Grants 61772346 and 61732003, Guangdong Natural Science Foundation (2017B030314073), and Guangdong Provincial General University National Development Program (2014GKXM054). Lu Qin was supported by ARC DP 160101513. Jeffrey Xu Yu was supported by the Research Grants Council of the Hong Kong SAR, China No. 14221716.

References

[1] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu, "Efficient core decomposition in massive networks," in *ICDE*, 2011.
[2] J. Wang and J. Cheng, "Truss decomposition in massive networks," *PVLDB*, vol. 5, no. 9, pp. 812–823, 2012.
[3] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," *SIGMOD*, 2014.
[4] L. Qin, R. Li, L. Chang, and C. Zhang, "Locally densest subgraph discovery," in *KDD*, 2015.
[5] 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.
[6] J. Fournet and A. Barrat, "Contact patterns among high school students," *PLOS ONE*, vol. 9, p. e107878, 2014.
[7] J. Viard, M. Latapy, and C. Magnien, "Computing maximal cliques in link streams," *Theor. Comput. Sci.*, vol. 609, pp. 245–252, 2016.
[8] S. B. Seidman, "Network structure and minimum degree," *Social networks*, vol. 5, no. 3, pp. 269–287, 1983.
[9] J. Viard, M. Latapy, and C. Magnien, "Revealing contact patterns among high-school students using maximal cliques in link streams," in *ASONAM*, 2015.
[10] A. Himmel, H. Molter, R. Niedermeier, and M. Sorge, "Enumerating maximal cliques in temporal graphs," in *ASONAM*, 2016.
[11] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *KDD*, 2010.
[12] R. Li, L. Qin, J. X. Yu, and R. Mao, "Influential community search in large networks," *PVLDB*, vol. 8, no. 5, pp. 509–520, 2015.
[13] 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.
[14] D. S. Johnson, "The NP-completeness column: An ongoing guide," *J. Algorithms*, vol. 8, no. 3, pp. 438–448, 1987.
[15] S. Galhotra, A. Bagchi, S. Bedathur, M. Ramanath, and V. Jain, "Tracking the conductance of rapidly evolving topic-subgraphs," *PVLDB*, vol. 8, no. 13, pp. 2170–2181, 2015.
[16] N. Bansal, F. Chiang, N. Koudas, and F. W. Tompa, "Seeking stable clusters in the blogosphere," in *VLDB*, 2007, pp. 806–817.
[17] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *PVLDB*, vol. 7, no. 9, pp. 721–732, 2014.
[18] 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.
[19] S. Huang, A. W. Fu, and R. Liu, "Minimum spanning trees in temporal graphs," in *SIGMOD*, 2015.
[20] S. Gurukar, S. Ranu, and B. Ravindran, "COMMIT: A scalable approach to mining communication motifs from dynamic networks," in *SIGMOD*, 2015.
[21] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke, "Reachability and time-based path queries in temporal graphs," in *ICDE*, 2016.
[22] Y. Yang, D. Yan, H. Wu, J. Cheng, S. Zhou, and J. C. S. Lui, "Diversified temporal subgraph pattern mining," in *KDD*, 2016.
[23] S. Ma, R. Hu, L. Wang, X. Lin, and J. Huai, "Fast computation of dense temporal subgraphs," in *ICDE*, 2017.
[24] P. Holme and J. Saramaki, "Temporal networks," *Physics Reports*, vol. 519, pp. 97–125, 2012.
[25] 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, p. 21, 2011.
[26] R. Li, J. X. Yu, and R. Mao, "Efficient core maintenance in large dynamic graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 10, pp. 2453–2465, 2014.
[27] R. Zhou, C. Liu, J. X. Yu, W. Liang, B. Chen, and J. Li, "Finding maximal k-edge-connected subgraphs from a large graph," in *EDBT*, 2012.
[28] 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.
[29] D. Eppstein and E. S. Spiro, "The *h*-index of a graph and its application to dynamic subgraph statistics," in *WADS*, 2009.