Mining Quasi-Periodic Communities in Temporal Network

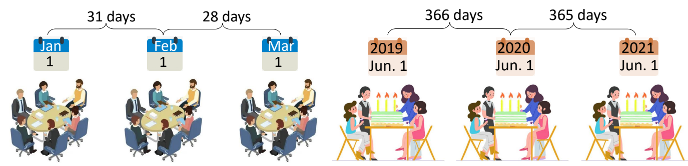
Yue Zeng*, Hongchao Qin*, Rong-Hua Li*, Kai Wang#, Guoren Wang*, Xuemin Lin#
*Beijing Institute of Technology; #Antai College of Economics and Management, Shanghai Jiao Tong University
bruceez@163.com; {qhc.neu, cskaelwang, wanggrbit}@gmail.com; lironghuabit@126.com; xuemin.lin@sjtu.edu.cn

***Abstract*—Periodic group behaviors often exist in temporal interaction networks, such as monthly group meetings, quarterly animal migrations, and yearly birthday parties. In real life, these events are usually quasi-periodic, meaning that the time intervals between two adjacent events are nearly constant but not exactly constant. Most existing studies mainly focus on identifying exact periodic group behaviors, which may result in an incomplete detection of periodic patterns in temporal networks. To fill this gap, we focus on a quasi-periodic community mining problem, which aims to find the most representative cohesive subgraphs, including the quasi-periodic k-core and quasi-periodic k-clique. The number of quasi-periodic communities is much larger than that of periodic communities, since the number of quasi-periodic sub-sequences is larger than that of periodic sub-sequences in a given time sequence. To efficiently compute the quasi-periodic communities, we propose a novel two-stage framework. In the first stage, the framework checks whether the time sequence of each vertex contains quasi-periodic sub-sequences. To this end, we develop a new structure, the DAG oracle, which comprises a set of concise DAGs that enables rapid extraction of all quasi-periodic sub-sequences. Based on the DAG oracle, we can easily compute all quasi-periodic sub-sequences for every vertex. In the second stage, the framework computes local quasi-periodic subgraphs that contain the vertex, which allows for the application of existing community mining algorithms. Given the large number of these subgraphs, we propose several carefully-designed pruning rules to further reduce redundant computations. Extensive experiments on 5 real-life datasets demonstrate the efficiency and effectiveness of our proposed solutions.**

I. Introduction

Temporal networks are networks in which each temporal edge between node u and v is associated with a created time t, denoted by (u, v, t). These networks often consist of periodic group behaviors, such as the activities of monthly group meetings, quarterly animal migrations, and yearly birthday parties, which often happen periodically in communities. However, the above events are quasi-periodic in real life, which means that the time intervals between two adjacent events are close to a constant but not precisely constant. For example, Figure 1 (a) shows an example of a monthly group meeting that is held periodically on the first day of each month. However, the meeting is actually quasi-periodic since the interval is not constant but ranges from 28 to 31 days. Figure 1 (b) shows an example of a yearly birthday party that is usually celebrated with an interval of 365 days, but there is also an interval of 366 days when considering leap years. Some other periodic communities exhibit quasi-periodicity because the schedule is delayed or moved up a bit due to unexpected events.

Quasi-periodic communities are useful in many potential applications, such as social network analysis and management. In contemporary social networks, a wealth of data concerning human interactions is documented. For instance, in datasets built upon facebook [1], users can establish a connection when they engage with the same post. Mining quasi-periodic communities in these datasets assists administrators in analyzing user gathering trends and forecasting potential social events, offering benefits in terms of targeted advertising and server maintenance. Quasi-periodic communities can also be applied in wildlife research and conservation. These patterns can be observed in natural animal herds, since many wild animals usually gather in groups and can be influenced by periodic factors, such as climates and human activities. Temporal networks can be built upon interactions of wild animals. For example, ecologists can tag the animals with wireless sensors to build proximity networks among animals [2]. By mining quasi-periodic communities in those networks, ecologists can gain insights into the migration and interaction patterns of wild animals, and implement essential conservation measures for animal herds that do not follow the above patterns (due to some unexpected events).



(a) Monthly group meeting (b) Yearly birthday party

Fig. 1. Example of quasi-periodic communities in real life

Therefore, it is significant to mine the quasi-periodic communities in the temporal network. In this paper, we study the quasi-periodic community mining problem, which aims to find the most representative cohesive subgraphs, including the quasi-periodic k-core and quasi-periodic k-clique.

There are some existing studies [3]–[5] that have already investigated the problem of periodic subgraph mining in temporal networks. For example, Lahiri et al. [3], [4] studied the problem of mining periodic subgraphs in temporal networks. However, in order to efficiently mining all periodic cohesive subgraphs, we should not enumerate all periodic subgraphs first, because many fringe nodes in the temporal graph can be safely pruned before enumerating periodic cohesive subgraphs. Therefore, directly using existing periodic subgraph mining techniques for mining periodic cohesive subgraphs are often inefficient. Recently, Qin et al. [6] studied the problem of mining periodic cliques in temporal networks; Zhang et al. [7] studied the problem of mining seasonal-periodic subgraphs. These methods can search the periodic cohesive subgraphs efficiently (e.g., the method proposed in [6] takes only 400 seconds to find all the results on a temporal graph with 12 million edges), but they do not consider the quasi-periodicity of the communities. The problem of mining quasi-periodic communities is often much harder than the problem of mining periodic communities, because quasi-periodicity can lead to a larger search space. To the best of our knowledge, we are the first to study the problem of quasi-periodic community mining in temporal networks.

To identify all quasi-periodic communities, one straightforward way is to find communities without considering temporal information first, and then check whether the communities are quasi-periodic. Clearly, such an approach is inefficient, as the number of possible communities can be very large in a temporal graph. Another potential method is to enumerate all maximal quasi-periodic subgraphs first and then find cohesive parts in those subgraphs to be the quasi-periodic communities. However, enumerating maximal quasi-periodic subgraphs is time- and space-consuming since the total number of quasi-periodic subgraphs is much larger than that of periodic subgraphs. This is because for a sequence T of maximum value T_{max}, the number of periodic sub-sequence of length σ is $O(T_{max}^2)$ [6] but the the number of quasi-periodic sub-sequence of length σ will raise to $O(T_{max}^2(T_{max}\epsilon + 1)^{\sigma-1})$ ($T_{max}\epsilon + 1 > 1$, as proved in Section III). As a consequence, the main challenge is how to enumerate quasi-periodic subgraphs properly and find the communities in those subgraphs with fewer redundant computations.

To address these challenges, we propose a novel two-stage framework for mining quasi-periodic communities in a temporal graph. In the first stage, the framework checks whether a vertex can be part of a quasi-periodic subgraph by mining quasi-periodic sub-sequences in time sequence associated to the vertex, and deletes vertex that can not be part of a quasi-periodic subgraph. To achieve this, we propose a quasi-periodic sub-sequence mining algorithm based on a novel structure: *DAG oracle*. The DAG oracle is a set of compact DAGs, which can fully characterize all quasi-periodic sub-sequences. Based on the *DAG oracle*, we can easily find all quasi-periodic sub-sequences and avoid enumerating the candidate sub-sequences for multiple times. Equipped with the DAG oracle, the time complexity of mining all quasi-periodic sub-sequences can be significantly reduced with a factor $O(T_{max})$. In the second stage, we utilize the reduced temporal graph to identify quasi-periodic communities. We compute local quasi-periodic subgraphs for each remaining vertex using the quasi-periodic sub-sequences obtained in the first stage. To reduce redundant computations, we further develop several non-trivial pruning techniques while computing quasi-periodic communities in those subgraphs.

Contributions. The main contributions are summarized as follows.

New models. We propose two new quasi-periodic community models based on the most representative cohesive subgraphs (k-core and k-clique). Specifically, we propose two models: ϵ-quasi σ-periodic k-core and ϵ-quasi σ-periodic k-clique. The ϵ-quasi σ-periodic k-core is a k-core which appears in the temporal network on a quasi-periodic time sequence of length σ, where the differences between adjacent timestamps fall within a range of $[d, d(\epsilon+1)]$. The ϵ-quasi σ-periodic k-clique model is defined similarly.

Novel algorithms. We develop several new algorithms to enumerate all quasi-periodic communities. First, we propose an algorithm based on DAG oracle to mine quasi-periodic sub-sequences in a time sequence, which is a basic task in quasi-periodic community mining. The DAG oracle is a set of compact DAGs and allows us to quickly search for all quasi-periodic sub-sequences. Second, we develop a two-stage framework for quasi-periodic community enumeration. The framework enumerates quasi-periodic communities for each vertex from a local perspective and can significantly reduce the cost of quasi-periodic community mining by integrating several carefully-designed pruning techniques.

Experimental evaluation. We conduct extensive experiments on five real-world temporal graphs to evaluate our proposed methods. First, we perform case studies on the real-world temporal graphs to evaluate the ability of quasi-periodic communities to reveal quasi-periodic group behaviors. The results show that our model can indeed find many interesting periodic patterns that cannot be detected by existing methods. Second, we evaluate the efficiency of the DAG oracle-based algorithm for quasi-periodic sub-sequence mining on two time sequences with different lengths. The results demonstrate that our approach is much more efficient than traditional methods in most cases. Finally, we evaluate the efficiency of our two-stage framework for quasi-periodic community mining, and the results show that our solution can be up to two orders of magnitude faster than the state-of-the-art method.

Reproducibility. The source code of this paper is released on Github: https://github.com/bruce1114/qpcommunity for reproducibility purposes.

Organization. Section II introduces the basic definitions and formulates the main problems. In Section III, we present our proposed algorithm for mining quasi-periodic sub-sequences. In Section IV, we introduce the framework to enumerate all quasi-periodic communities. We evaluate the effectiveness of our approach through experiments in Section V, and review related work in Section VI. Finally, we conclude our work in Section VII.

II. Preliminaries

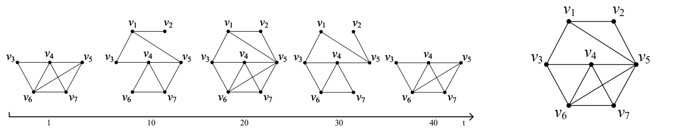
Let $\mathcal{G} = (V, \mathcal{E})$ be an undirected temporal graph, where V and $\mathcal{E}$ are the sets of vertices and temporal edges, respectively. The edges in $\mathcal{E}$ are of the form (u, v, t), where u and v are vertices in V and t is the timestamp of each edge. The snapshot of $\mathcal{G}$ at timestamp t is denoted as $SN_t = (V_t, E_t)$ where $V_t = \{u|(u, v, t) \in \mathcal{E}\}$ and $E_t = \{(u, v)|(u, v, t) \in \mathcal{E}\}$. The de-temporal graph of the temporal graph $\mathcal{G}$ is denoted as $G = (V, E)$, where $E = \{(u, v)|(u, v, t) \in \mathcal{E}\}$.

A graph $G_S = (V_S, E_S)$ is a subgraph of G if $V_S \subseteq V$ and $E_S \subseteq E$, which can be represented as $G_S \subseteq G$ ($G_S \subset G$ if $G_S \neq G$). For a vertex $u \in V$, $N_G(u) = \{v|(u, v) \in E\}$ is the set of neighbors of u in G, and $deg_G(u) = |N_G(u)|$ is the degree of u in G.

Definition 1 (ϵ-quasi σ-periodic time sequence ((ϵ, σ)-**QPT**)). *Given a time sequence $T = (t_1, t_2, ..., t_\sigma)$ and a real number $\epsilon \geq 0$ (T is in ascending order with length σ and contains no duplicate timestamps), T is an ϵ-quasi σ-periodic time sequence only if:*

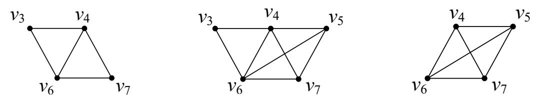
$$\exists d > 0, \forall i = 1, 2, ..., \sigma - 1, d \leq t_{i+1} - t_i \leq d(1 + \epsilon). \quad (1)$$

In other words, all differences between adjacent values of an (ϵ, σ)-QPT are in a specific range of $[d, d(1+\epsilon)]$. Definition 1 was first introduced in [8]. For an (ϵ, σ)-QPT T, let $d' = \min_{i=1,\ldots,\sigma-1}(t_{i+1} - t_i)$, it is clear that Equation 1 still holds if $d = d'$. For two time sequences T_1 and T_2, $T_1 \subseteq T_2$ indicates that T_1 is a sub-sequence of T_2 or T_1 is in T_2. If not specified, all time sequences in this paper are integer sequences and in ascending order without duplicate values. For a time sequence



(a) Five snapshots of the temporal graph (b) De-temporal graph

Fig. 2. A sample temporal graph



(a) $T = (20, 30, 40)$ (b) $T = (1, 20, 40)$ (c) $T = (1, 20, 40)$

Fig. 3. Sample MQPCore and MQPClique in Figure 2

T, $D^T = \{t_b - t_a | t_a, t_b \in T, t_a < t_b\}$, D^T_{min}, D^T_{max} are the smallest and biggest element in D^T respectively. T_{max} is the biggest timestamp in T.

Example 1. *$T = (1, 10, 20, 30, 40)$ is a $(12\%, 5)$-QPT, since $\exists 9 > 0, 9 \leq T[i+1] - T[i] \leq 9(1 + 12\%) = 10.08, i = 0, 1, 2, 3$.*

Definition 2 (Support time sequence (**SupT**))**.** *Given a temporal graph $\mathcal{G} = (V, \mathcal{E})$ and its de-temporal graph $G = (V, E)$, the support time sequence of a subgraph G_S of G is the longest sequence $SupT_{\mathcal{G}}(G_S) = (t_1, t_2, ...)$ satisfying $G_S \subseteq SN_{t_i}$ for all possible i. In particular, for an edge $(u, v) \in E$, $SupT_{\mathcal{G}}((u, v))$ is equivalent to $SupT_{\mathcal{G}}((\{u, v\}, \{(u, v)\}))$.*

Definition 3 (ϵ-quasi σ-periodic subgraph ((ϵ, σ)-**QPS**))**.** *Given a temporal graph $\mathcal{G}$, a two-tuple (G_S, T) represents an ϵ-quasi σ-periodic subgraph if T is an (ϵ, σ)-QPT and $T \subseteq SupT_{\mathcal{G}}(G_S)$.*

Definition 4 (ϵ-quasi σ-periodic k-core)**.** *Given a temporal graph $\mathcal{G}$, an (ϵ, σ)-QPS (G_S, T) represents an ϵ-quasi σ-periodic k-core in $\mathcal{G}$ if for each vertex u in G_S, $deg_{G_S}(u) \geq k$.*

It is important to note that in this paper, k-core is defined as a connected subgraph. An ϵ-quasi σ-periodic k-core (G_S, T) is maximal if there is no other ϵ-quasi σ-periodic k-core (G'_S, T) such that $G_S \subset G'_S$. We use (ϵ, σ, k)-**MQPCore** to represent maximal ϵ-quasi σ-periodic k-core, or simply **MQPCore** if ϵ, σ, k are not required.

Definition 5 (ϵ-quasi σ-periodic k-clique)**.** *Given a temporal graph $\mathcal{G}$, an (ϵ, σ)-QPS $(G_S = (V_S, E_S), T)$ represents an ϵ-quasi σ-periodic k-clique in $\mathcal{G}$ if G_S is a clique with $|V_S| \geq k$.*

An ϵ-quasi σ-periodic k-clique (G_S, T) is maximal if there is no other ϵ-quasi σ-periodic k-clique (G'_S, T) such that $G_S \subset G'_S$. We use (ϵ, σ, k)-**MQPClique** to represent maximal ϵ-quasi σ-periodic k-clique, or simply **MQPClique** if ϵ, σ, k are not required.

Example 2. *Figure 2 depicts a sample temporal graph with some periodic patterns. Firstly, if $\epsilon = 0$, a sample $(0, 3, 2)$-MQPCore (Figure 3 (a)) of the temporal graph in Figure 2 can be found. Then, by setting $\epsilon = 12\%$, a more complicated $(12\%, 3, 2)$-MQPCore can be obtained, as shown in Figure 3 (b). Additionally, the MQPCore of Figure 3 (b) contains a $(12\%, 3, 4)$-MQPClique induced by v_4, v_5, v_6, v_7 with $T = (1, 20, 40)$, as depicted in Figure 3 (c).*

Problem 1. Given a temporal graph $\mathcal{G}$ and parameters k, σ, ϵ, our goal is to enumerate all (ϵ, σ, k)-MQPCores.

Hardness analysis. To solve the above problem, the first solution is to enumerate all k-cores in the de-temporal graph and test whether these k-cores are maximal and appear in the temporal graph on a quasi-periodic time sequence. This solution is impracticable since it requires non-polynomial time to enumerate all possible k-cores. The second solution is to mine all quasi-periodic subgraphs and apply traditional algorithms to identify all MQPCores and MQPCliques. In detail, we mine quasi-periodic sub-sequences in the entire timeline of temporal graph, and extract the common subgraph of snapshots in timestamps of each obtained quasi-periodic sub-sequence. The above process can be completed in polynomial time, because the time complexity of quasi-periodic sub-sequences mining is polynomial (as shown in Theorem 5), as well as the time complexity of maximal k-cores mining in those common subgraphs. However, mining all quasi-periodic sub-sequences is still very costly because the number of quasi-periodic sub-sequences can be much larger than that of periodic sub-sequences ($O((T_{max}\epsilon + 1)^{\sigma - 1})$ times larger in the worst case, ϵ, σ are specified by users).

Problem 2. Given a temporal graph $\mathcal{G}$ and parameters k, σ, ϵ, our goal is to enumerate all (ϵ, σ, k)-MQPCliques.

Hardness analysis. In contrast to the MQPCore problem, the MQPClique enumeration problem is NP-hard. Consider a temporal graph $\mathcal{G}$ with identical snapshots at every timestamp, i.e., $SN_1 = SN_2 = \cdots = SN_t$, and $\sigma = t, \epsilon = 0$. It is clear that each maximal clique in SN_1 that is not smaller than k can form a MQPClique. In this case, the MQPClique enumeration problem is equivalent to the maximal clique enumeration problem (for cliques not smaller than k). Therefore, the MQPClique enumeration problem is NP-hard.

III. Quasi-Periodic Time Sequence Mining

In this section we introduce QPT mining, which is a basic task in solving all problems in this paper.

(ϵ, σ)-**QPT mining.** Given a time sequence $T = (t_1, t_2, ..., t_l)$, an integer σ ($\sigma \leq l$) and a real number ϵ ($\epsilon \geq 0$), the goal is to find all (ϵ, σ)-QPTs T' satisfying $T' \subseteq T$.

In this section, at first we do not make any assumptions about the properties of time sequence T. At the end of this section, we will discuss the properties of sequences that are typically encountered in mining quasi-periodic communities.

A. The Basic Method

Algorithm 1 is the basic method to solve the problem of QPT mining. The idea is simple. At the end of each iteration, $T[i]$ and each value of T in range $[0, i)$ will generate a new QPT candidate of length 2 (line 17-19), waiting for forming longer QPTs with the following values in T. Each QPT candidate is in form of $(T', maxD, minD)$ (line 3), where T' is the QPT and $maxD, minD$ are maximum and minimum differences respectively between adjacent values in T'. In each iteration, Algorithm 1 traverses all candidates in $candSet$ and checks whether they can form longer QPTs with $T[i]$. For a QPT candidate qpt, if $T[i]$ is appended to $qpt.T'$, then the upper limit of difference ($tmpLimit$) or the biggest difference ($tmpMaxD$) may be updated (line 7-8). If $tmpMaxD$ is still within the upper limit, then a new candidate or a new (ϵ, σ)-QPT is generated (line 9-15). We need to abandon candidates when they are hopeless to form new QPTs (line 16).

The size of $candSet$ is the key factor in analyzing the time complexity of Algorithm 1. We present the following theorem first.

Algorithm 1: QPT (T, σ, ϵ)

```
Input: A time sequence T. σ and ϵ
Output: QPT, the set of (ϵ, σ)-QPTs in T
QPT ← ∅; candSet ← ∅;
for i ← 1 to (|T| − 1) do
    // generating longer QPT using qpt and T[i]
    foreach qpt ← (T', maxD, minD) ∈ candSet do
        d ← T[i] − qpt.T'[|qpt.T'| − 1];
        tmpLimit ← (1 + ϵ) × qpt.minD;
        tmpMaxD ← qpt.maxD;
        if d < qpt.minD then tmpLimit ← d(1 + ϵ);
        if d > qpt.maxD then tmpMaxD ← d;
        if tmpMaxD ≤ tmpLimit then
            // qpt and T[i] can form a longer QPT
            cand ← qpt;
            cand.minD ← min(qpt.minD, d);
            cand.maxD ← max(qpt.maxD, d);
            cand.T'.append(T[i]);
            if |cand.T'| = σ then QPT ← QPT ∪ {cand};
            else candSet ← candSet ∪ {cand};
        // qpt can not form longer QPTs with all the
           following timestamps in T
        if d ≥ tmpLimit then candSet ← candSet − {qpt};
    for j ← 0 to (i − 1) do
        // generating new QPT candidates using T[j], T[i]
        cand ← ([T[j], T[i]], T[i] − T[j], T[i] − T[j]);
        candSet ← candSet ∪ {cand};
return QPT;
```

Theorem 1. *Given a time sequence T, the total number of (ϵ, σ)-QPTs in T is less than $T_{max}^2(T_{max}\epsilon + 1)^{\sigma-1}$.*

Due to the space limits, all missing proofs can be found in the full version of this paper [9].

Theorem 2. *The time complexity of Algorithm 1 is $O(T_{max}^3(T_{max}\epsilon+1)^{\sigma-2})$. The space complexity of Algorithm 1 is $O(T_{max}^2(T_{max}\epsilon + 1)^{\sigma-1})$.*

B. DAG Oracle Based Method

In the basic method, the size of candidate set increases when σ or ϵ increases, and it is traversed in each iteration, leading to low efficiency. In this subsection we introduce the DAG oracle based method. The idea is to represent each value in sequence T as a node in a DAG. For an (ϵ, σ)-QPT $T' \subseteq T$, it can be seen as a path in the DAG, which is composed of directed edges in the form of (u, v), satisfying $u, v \in T$ and $D_{min}^{T'} \leq v - u \leq D_{min}^{T'}(1 + \epsilon)$.

Definition 6 (DAG oracle)**.** *Given a sequence T and ϵ, the DAG oracle of T is a set of DAGs, $\mathcal{DAG} = \{DAG_d | d \in D^T\}$, and $DAG_d = \{(u, v)|u, v \in T, d \leq v - u \leq d(1 + \epsilon)\}$.*

Definition 7 (Key edge)**.** *Given a DAG oracle $\mathcal{DAG}$ and $DAG_d \in \mathcal{DAG}$, for any edge $(u, v) \in DAG_d$, if $(v - u) = d$, then (u, v) is a key edge in DAG_d.*

Example 3. *Let $T = (1, 10, 21, 35, 40, 49, 69, 75)$ and $\epsilon = 0.2$. The DAG oracle of T, denoted by $\mathcal{DAG}$, is illustrated in Figure 4. As depicted in the left-hand side of Figure 4, each value in T is treated as a node, resulting in at most $O(|T|^2)$ direct edges. Each direct edge can belong to more than one DAG. In the middle of Figure 4,* $(1, 40)$ *is distributed into DAG_{34}, DAG_{35} since* $(40 - 1)$ *is in both* $[34, 34 \times 1.2], [35, 35 \times 1.2]$. *Moreover,* $T' = (1, 35, 69)$ *is a* $(0.2, 3)$-*QPT in T, and it is a path of length 3 in DAG_{34}.*

Theorem 3. *Given a sequence T, $\epsilon \geq 0$ and the DAG oracle $\mathcal{DAG}$ of T, (1) for any (ϵ, σ)-QPT $T' \subseteq T$, there must be a $DAG_{d'} \in \mathcal{DAG}$ that T' is a path of length σ in $DAG_{d'}$,*

Algorithm 2: BuildDAG (T, ϵ)

```
   Input: A time sequence T. ϵ
   Output: DAG oracle for T, 𝒟𝒜𝒢
1  D^T ← {b − a|a, b ∈ T, a < b},
     𝒟𝒜𝒢 ← {DAG_d ← empty list|d ∈ D^T}; // initializing
     DAG oracle
2  for i ← 0 to |T| − 2 do
3      for j ← i + 1 to |T| − 1 do
4          d ← T[j] − T[i];
           // adding (T[i], T[j]) into 𝒟𝒜𝒢
5          for d' ∈ D^T ∧ d' in [d/(1+ϵ), d] do
6              if there is no item (i, *, *) ∈ DAG_d' then
                   DAG_d'.append((i, j, j));
7              Let (i, l, r) be the item (i, *, *) in DAG_d', r ← j;
8  return 𝒟𝒜𝒢;
```

and (2) for any $DAG_d \in \mathcal{DAG}$, if there exists a path $p = (u_1, u_2, ..., u_\sigma)$ of length σ in DAG_d, then p is also an (ϵ, σ)-QPT in T.

Based on Theorem 3, we can transform QPT mining into the problem of finding all fixed-length (the number of nodes) paths in the DAG oracle. In the following, we introduce the construction of the DAG oracle.

Construct the DAG oracle. Given ϵ, constructing DAG oracle for sequence T is very simple. It is clear that all possible direct edges in T are $\tilde{E} = \{(u, v)|u, v \in T, u < v\}$. For each $(u, v) \in \tilde{E}$, if $(u, v) \in DAG_d$, then $\frac{v-u}{1+\epsilon} \leq d \leq v - u$, which means we should add (u, v) into all DAG_d where d is in $[\frac{v-u}{1+\epsilon}, v - u]$. Algorithm 2 presents the procedure to construct DAG oracle.

Algorithm 2 first computes D^T and initiates $\mathcal{DAG}$ in line 1. Then, the algorithm traverses all possible edges in line 2-3. For each edge $(T[i], T[j])$, the algorithm adds it into $DAG_{d'}$ where $d' \leq T[j] - T[i] \leq d'(1 + \epsilon)$ (line 5-7). Note that we do not need to store DAGs in $\mathcal{DAG}$ in the regular way. Instead, for each node $T[i]$ in $DAG_{d'}$, since all its neighbors $T[j_l], T[j_{l+1}], ..., T[j_r]$ must be in a continuous interval of T, we only need to store all its adjacent edges in form of (i, j_l, j_r) as in line 6-7. DAGs in DAG oracle are compact because $T[j_l] - T[i]$ and $T[j_r] - T[i]$ are both in range of $[d', d'(1+\epsilon)]$ and ϵ is usually smaller than 1.

Theorem 4. *The time complexity of Algorithm 2 is $O(|T|^2(\frac{T_{max}\epsilon}{\epsilon+1} + 1))$. The space complexity of Algorithm 2 is $O(|T|T_{max})$.*

Mining QPT on DAG oracle. As we mentioned before, QPT mining is equivalent to finding all fixed-length paths in DAG oracle. In this paper we use Depth-First-Search (DFS) to find all such paths. There are some details to be considered.

Avoid redundant QPTs. The same path may be found in more than one DAGs of DAG oracle. For example, in Figure 4, $(1, 40, 75)$ is in both DAG_{34} and DAG_{35}. To avoid such redundant path, in each DAG_d of DAG oracle we only search fixed-length paths containing key edges (Definition 7) of DAG_d. For example, in DAG_{34} of Figure 4, we only search paths containing $(1, 35)$ and $(35, 69)$. The reason is that each QPT T' can be found in $DAG_{D_{min}^{T'}}$, as in the proof of Theorem 3, and paths containing no key edge will be found in other DAGs.

In a DAG, a path may contain 2 or more key edges, so it is necessary to delete any key edge after all valid paths

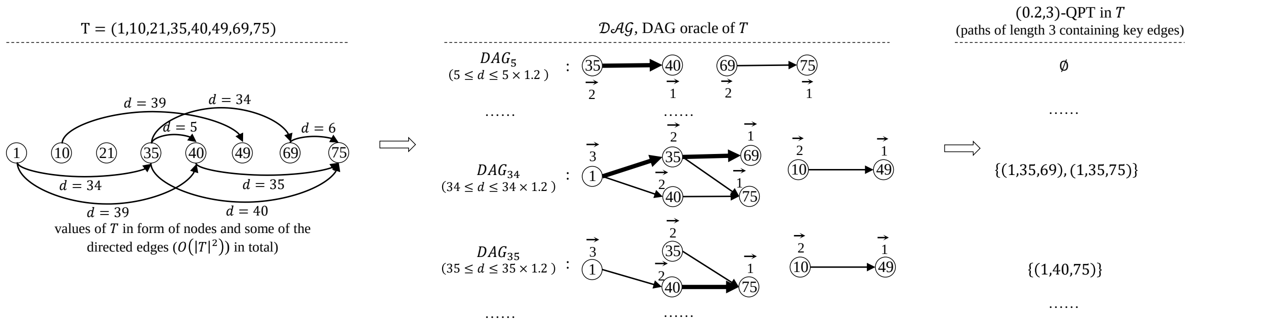


Fig. 4. Sample DAG oracle of $T = (1, 10, 21, 35, 40, 49, 69, 75)$. Bold edges in DAG oracle are key edges. Values in auxiliary array $maxLen$ are shown around nodes in DAG oracle

containing it are found. This avoids redundant paths when processing other key edges. For instance, in Figure 4, we delete key edge $(1, 35)$ in DAG_{34} after discovering the path $(1, 35, 69)$. This ensures that $(1, 35, 69)$ is not rediscovered when searching for all paths containing key edge $(35, 69)$. It is very simple to delete a key edge. For example, to delete $(1, 35)$ in DAG_{34} of Figure 4 we only need to modify $(0, 3, 4)$ stored in DAG_{34} to $(0, 4, 4)$.

Heuristic DFS. In the search for valid paths, we employ a heuristic depth-first search (DFS) with an auxiliary array called $maxLen$, which records the maximum length of paths from each node. For instance, the value in $maxLen$ for node 1 of DAG_{34} in Figure 4 is shown at the top of node 1. Utilizing $maxLen$, if there is no path of the required length starting from node u, then the successors of u need not be visited.

Update auxiliary array. After deleting the key edge (u, v) from DAG_d, we need to update the value of $maxLen[u]$ to reflect the new maximum length of paths starting from u. Specifically, we set $m = \max_{(u,v') \in DAG_d, v' \neq v} maxLen[v'] + 1$ ($m - 1$ is the maximum length of paths starting from direct successors of u) and update $maxLen[u]$ to m if $m < maxLen[u]$. We also need to check and update $maxLen[u']$ (u' is some of ancestors of u, not only parent nodes) in a similar way. Note that it is unnecessary to check all ancestors of u. Suppose u' is an ancestor of u, let $m = \max_{(u',v') \in DAG_d} maxLen[v'] + 1$ and σ be the required length of valid paths, if $m = \sigma - 1$, then all ancestors of u' do not need to be updated since the maximum length of paths starting from those nodes is at least σ, and it does not matter whether these values are σ or greater. To efficiently perform these updates, we use a Breadth-First-Search (BFS) method.

Algorithm implementation. Algorithm 3 presents the detailed process to mine QPT on DAG oracle. As in line 2, Algorithm 3 traverses each DAG (DAG_d) to search for valid paths. In line 3, Algorithm 3 creates $\overline{DAG}$, which is obtained by reversing all directed edges in DAG. Tuples in $\overline{DAG}$ are still in form of (i, l, r). $\overline{DAG}$ is essential because we also need to perform a reverse search starting from the starting node of key edges. In line 1, we create two auxiliary arrays $maxLen, \overline{maxLen}$ to record the maximum length of paths starting from each node in DAG and $\overline{DAG}$ respectively. In line 5-8, Algorithm 3 collects all key edges and initializes $maxLen, \overline{maxLen}$ (by traversing DAG and $\overline{DAG}$ only once).

Algorithm 3 traverses each key edge $edge$ and since $edge$ can appear at any position in a valid path, Algorithm 3 traverses each possible position in line 12 by enumerating pre, which is the length of the first part of valid paths separated by $edge[0]$ (not included). In line 16-17, Algorithm 3 searches both parts of valid paths in two directions. For example in line 16, we search the first part of valid paths of required length $pre + 1$ backwards with the help of auxiliary array $\overline{maxLen}$. In hDFS, recursive calls occur only if there exists a path of the required length starting from node i in line 27. In line 18, we simply put parts of valid paths in $preAns, postAns$ together in pairs to get the final set of results QPT.

After enumerating each key edge, we delete it in both DAG and $\overline{DAG}$ to avoid redundant results as in line 31-32 of Algorithm 3. After that, auxiliary arrays $maxLen, \overline{maxLen}$ should be updated. From Algorithm 2 we know that tuples (i, l, r) in DAG are ordered by i, consequently key edges $edge$ in $keyEdge$ are ordered by $edge[0]$. Deleting a key edge will not change the maximum length of paths in DAG starting from nodes of key edges which will be traversed later. As a result, $maxLen$ does not need to be updated. DeleteKeyEdge uses a BFS way to update $\overline{maxLen}$. The first node to be updated is the end node $edge[1]$ (line 33). The new value of $\overline{maxLen}[edge[1]]$ depends on all direct successors of $edge[1]$ in $\overline{DAG}$ (or parent nodes in DAG, line 36-37). If $\overline{maxLen}[edge[1]]$ should be updated (decreased, line 38), then all direct successors of $edge[1]$ in DAG should be added into Q, waiting for similar treatment to $edge[1]$ (line 41-42) unless $longest + 1 \geq \sigma - 1$ (line 40). If $longest + 1 \geq \sigma - 1$, then the $\overline{maxLen}$ values of all successors of $edge[1]$ (or $curNode$ in DeleteKeyEdge) in DAG are at least σ. In this case, it does not matter whether these values are σ or greater.

Theorem 5. *Both the time and space complexity of Algorithm 3 are $O(T_{max}^2 (T_{max}\epsilon + 1)^{\sigma-1})$.*

C. Discussion

From the analysis presented above, it is evident that the DAG oracle based method is more efficient compared to the basic method. This is due to the reduction of the T_{max}^3 factor to T_{max}^2. In the worst case, the time sequence T can closely resemble a natural number sequence, as interactions can occur at every time unit in real-world scenarios. In such cases, the actual running time of Algorithm 1 and Algorithm 3 may approach their theoretical time complexity. However, this work does not directly mine QPTs over the entire timeline of temporal networks. Instead, it focuses on time sequences associated with vertices. Moreover, at each timestamp of these time sequences, the degree of the corresponding vertices must exceed a specific threshold according to the definitions of MQPCore and MQPClique (see Definition 8). Consequently, these time sequences are unlikely to be as long and dense as

Algorithm 3: QPT+ $(T, \sigma, \epsilon, \mathcal{DAG})$

```
Input: A sequence T. σ and ε. DAG oracle of T, 𝒟𝒜𝒢
Output: QPT, the set of (ε, σ)-QPTs in T,
maxLen ← {}; \overline{maxLen} ← {}; QPT ← ∅;
foreach DAG_d ∈ 𝒟𝒜𝒢 do
    DAG ← DAG_d; \overline{DAG} ← DAG.reverse(); // a reversed DAG
    keyEdge ← [];
    for (i, l, r) ∈ DAG do
        if T[l] − T[i] = d then keyEdge.append((i, l));
    if |keyEdge| = 0 then continue;
    Initialize maxLen, \overline{maxLen};
    for edge ∈ keyEdge do
        tail ← maxLen[edge[1]] − 1; head ← 0;
        if tail < σ − 2 then head ← σ − 2 − tail;
        // enumerating the possible length of prefix separated by edge[0]
        for pre ← head to (σ − 2) do
            post ← σ − 2 − pre; // the length of postfix separated by edge[1]
            if \overline{maxLen}[edge[0]] − 1 < pre then break;
            preAns ← []; postAns ← []; curPath ← [];
            // searching for valid paths from two directions
            hDFS (edge[0], pre + 1, T, \overline{DAG}, \overline{maxLen}, curPath, preAns);
            hDFS (edge[1], post + 1, T, DAG, maxLen, curPath, postAns);
            CombineAns (preAns, postAns, QPT);
        DeleteKeyEdge (edge, DAG, \overline{DAG}, maxLen, \overline{maxLen});
return QPT;

// hDFS is used to mine paths of required length from start in DAG
Procedure hDFS (start, leftLen, T, DAG, maxLen, curPath, partAns)
curPath.append(T[start]); Let (start, l, r) be (start, *, *) in DAG;
if leftLen = 1 then partAns.append(curPath.copy());
else
    leftLen ← leftLen − 1;
    for i ← l to r do
        if maxLen[i] ≥ leftLen then
            hDFS (i, leftLen, T, DAG, maxLen, curPath, partAns);
curPath.pop();

Procedure DeleteKeyEdge (edge, DAG, \overline{DAG}, maxLen, \overline{maxLen})
Let (edge[0], l_i, r_i) be (edge[0], *, *) in DAG. Let (edge[1], l_j, r_j) be (edge[1], *, *) in \overline{DAG};
l_i ← l_i + 1, r_j ← r_j − 1; // delete the key edge
// update \overline{maxLen}
Q ← an empty queue; Q.push(edge[1]); // Q stores all nodes waiting to be updated
while Q.empty() = false do
    curNode ← Q.front(); Q.pop();
    Let (curNode, l_rev, r_rev) be (curNode, *, *) in \overline{DAG};
    longest ← max_{k=l_rev,...,r_rev} \overline{maxLen}[k];
    if \overline{maxLen}[curNode]! = longest + 1 then
        \overline{maxLen}[curNode] ← longest + 1;
        if longest + 1 < σ − 1 ∧ maxLen[curNode] > 1 then
            Let (curNode, l, r) be (curNode, *, *) in DAG;
            for i ← l to r do Q.push(i);
```

the overall timeline of the temporal networks. Further details will be provided in the next section.

IV. Mining Quasi-Periodic Communities

In this section, we present a framework for mining MQPCores and MQPCliques in a temporal graph. The overall process of this framework involves traversing all vertices, with two stages of tasks for each vertex. In the first stage, the framework determines whether the vertex has the potential to be included in a MQPCore or MQPClique. In the second stage, if the vertex is eligible for inclusion in a MQPCore or MQPClique, then the framework tries to compute all MQPCores or MQPCliques containing that vertex. The process then moves on to the next vertex. Two pruning rules are employed in the above process.

A. The First Stage

We first introduce how to determine whether a vertex has the potential to be included in a MQPCore or MQPClique. Considering that the degree of each vertex in MQPCore or MQPClique must exceed a threshold value, we present the following definition.

Definition 8 (ϵ-quasi (σ, k)-periodic vertex ((ϵ, σ, k)-**QPV**)). *Given a temporal graph $\mathcal{G} = (V, \mathcal{E})$ and its de-temporal graph $G = (V, E)$, for a vertex $u \in V$, $t_k(u)$ is the longest timestamp sequence where $\forall t \in t_k(u), deg_{SN_t}(u) \geq k$. u is an ϵ-quasi (σ, k)-periodic vertex if there exists an (ϵ, σ)-QPT in $t_k(u)$.*

All (ϵ, σ)-QPTs in $t_k(u)$ form an ϵ-quasi (σ, k)-periodic support time sequence set of u. For convenience, we let (ϵ, σ, k)-**QPTSET** (or **QPTSET** if no specified parameters) be the ϵ-quasi (σ, k)-periodic support time sequence set with no specified vertex, and (ϵ, σ, k)-$QPTSET_u$ be QPTSET of vertex u (or $QPTSET_u$).

Clearly, if a vertex u is not an (ϵ, σ, k)-QPV ($QPTSET_u = \emptyset$), then it is impossible to be contained in (ϵ, σ, k)-MQPCores or $(\epsilon, \sigma, k+1)$-MQPCliques. Such vertices can be directly deleted.

Example 4. *In temporal graph of Figure 2, let $\epsilon = 12\%, \sigma = 3, k = 2$, v_2 is not an (ϵ, σ, k)-QPV, since $t_k(v_2) = (20)$ and $QPTSET_{v_2} = \emptyset$. All other vertices are (ϵ, σ, k)-QPV, for example, $t_k(v_1) = (10, 20, 30)$ and $QPTSET_{v_1} = \{(10, 20, 30)\}$.*

B. The Second Stage (MQPCore Enumeration)

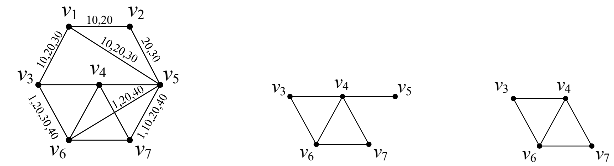
In this section, we present the computation method for MQPCores containing a specific QPV, which is one of the objectives of the second stage. Given a QPV u, it can be observed that any MQPCore that contains u must be a subgraph of a larger QPS, which is a local quasi-periodic subgraph and also contains u. Next we define quasi-periodic connected subgraph, the local quasi-periodic subgraph especially for MQPCore enumeration.

Definition 9 (ϵ-quasi σ-periodic connected subgraph ((ϵ, σ)-**QPCS**)). *Given a temporal graph $\mathcal{G} = (V, \mathcal{E})$ and ϵ, σ, an (ϵ, σ)-QPS $(G_S = (V_S, E_S), T)$ is an ϵ-quasi σ-periodic connected subgraph if G_S is a maximal connected subgraph such that no other (ϵ, σ)-QPS (G'_S, T) satisfies $G_S \subset G'_S$ and G'_S is connected.*

Given a temporal graph $\mathcal{G} = (V, \mathcal{E})$ and parameters ϵ, σ, k, an (ϵ, σ, k)-QPV u in $\mathcal{G}$ is contained by a QPCS $(G_S = (V_S, E_S), T)$ only if $u \in V_S$ and $T \in (\epsilon, \sigma, k)$-$QPTSET_u$. We call such QPCS as (ϵ, σ)-$QPCS_u$ or $QPCS_u$.

Example 5. *Consider the sample temporal graph in Figure 5 (a) and $\epsilon = 12\%, \sigma = 3, k = 2$. Figure 5 (b) shows an example of $QPCS_{v_3}$, since v_3 is an (ϵ, σ, k)-QPV, $(20, 30, 40) \in QPTSET_{v_3}$ and v_3 is in G_S, which is a maximal connected subgraph under the constraint of time sequence $(20, 30, 40)$.*

Theorem 6. *Given a temporal graph $\mathcal{G} = (V, \mathcal{E})$ and ϵ, σ, k, all MQPCores can be obtained in $QPCS_u, u \in V$.*



(a) $\mathcal{G}$, the sample temporal graph (b) $(G_S, (20, 30, 40))$, an example of $QPCS_{v_3}$ (c) k-core in G_S

Fig. 5. An example of (ϵ, σ)-$QPCS_{v_3}$. $\epsilon = 12\%, \sigma = 3, k = 2$, $(20, 30, 40) \in QPTSET_{v_3}$. Labels of edges with timestamps $(1, 10, 20, 30, 40)$ are omitted in (a)

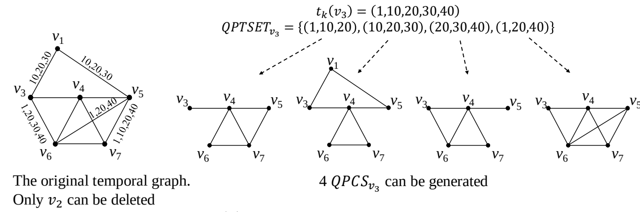
According to Theorem 6, the direct idea of MQPCore mining is to extract k-cores from all possible $QPCS_u$ in traversing each vertex u. In detail, with $QPTSET_u$ computed in the first stage (Section IV-A), we compute $QPCS_u$ and its k-cores for each $qpt \in QPTSET_u$. A $QPCS_u$ can be computed by Breadth-First-Search starting from u. For example, G_S in Example 5 can be obtained by Breadth-First-Search starting from v_3. In Example 5 we can see that by computing k-core in G_S, a MQPCore $(C, (20, 30, 40))$ can be obtained where C is shown in Figure 5 (c). Note that $(G_S, (20, 30, 40))$ can also be $QPCS_{v_5}$ but v_5 is not contained in $(C, (20, 30, 40))$ above. It does not affect the correctness of our framework to mine all MQPCores.

However, above solution involves numerous redundant computations. For example, for a vertex u, if $QPTSET_u \neq \emptyset$ and all possible $QPCS_u$ and MQPCores in those QPCS are computed, then all remaining MQPCores in the temporal network do not contain u any more. In this case, u can be deleted, and for remaining vertices the cost of computing QPTSET and QPCS can be reduced. As in Figure 5 (a), $deg_{SN_{10}}(v_3) \geq 2$, but $deg_{SN_{10}}(v_3) < 2$ after v_1 is deleted. So that $|t_k(v_3)|, QPTSET_{v_3}$ will be smaller and fewer $QPCS_{v_3}$ will be generated. As a result, we develop the following pruning rule.

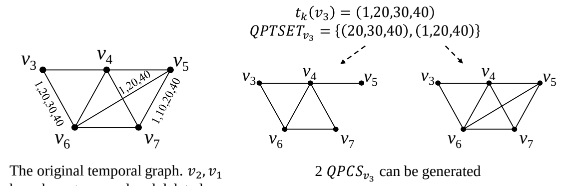
Pruning rule 1. Consider the temporal graph $\mathcal{G} = (V, \mathcal{E})$, we traverse vertices $u \in V$ with small degree first and delete them after both stages of computation have been completed. In detail, we sort all vertices by their degree in the de-temporal graph first, since vertex u with small degree may have shorter $t_k(u)$, smaller $QPTSET_u$ and fewer $QPCS_u$. Deleting each vertex that has been traversed may cause its neighbors v to have smaller degree, shorter $t_k(v)$, smaller $QPTSET_v$, and fewer $QPCS_v$.

Example 6. *Figure 6 shows two cases when v_3 is traversed with and without pruning rule 1. At first, v_2 is deleted directly in both Figure 6 (a) and (b) since $QPTSET_{v_2} = \emptyset$ as in Figure 5 (a). In Figure 6 (a), no more vertices can be deleted since every remaining vertex has nonempty QPTSET. As a result, in Figure 6 (a) $t_k(v_3) = (1, 10, 20, 30, 40)$, $|QPTSET_{v_3}| = 4$ and there are 4 $QPCS_{v_3}$. However, in Figure 6 (b) we delete vertices having been traversed, so that $t_k(v_3)$ is shorter and only 2 $QPCS_{v_3}$ are produced. MQPCore in the first $QPCS_{v_3}$ of Figure 6 (a) can still be computed in $QPCS_{v_7}$. MQPCore in the second $QPCS_{v_3}$ of Figure 6 (a) has already been computed after v_1 was traversed.*

Applying pruning rule 1 cannot eliminate all redundant computations. As in Example 6, when v_7 is being traversed, since v_3 has been deleted and $(20, 30, 40) \in QPTSET_{v_7}$, a $QPCS_{v_7}$ $(G'_S, (20, 30, 40))$ will be generated where G'_S is composed of only 4 edges: $(v_4, v_6), (v_4, v_7), (v_6, v_7), (v_4, v_5)$.



(a) Without pruning rule 1



(b) With pruning rule 1. Vertices are sorted as $v_2, v_1, v_3, v_7, v_6, v_4, v_5$ by their degree

Fig. 6. MQPCore enumeration when v_3 is traversed. $\epsilon = 12\%, \sigma = 3, k = 2, T = (1, 10, 20, 30, 40)$

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

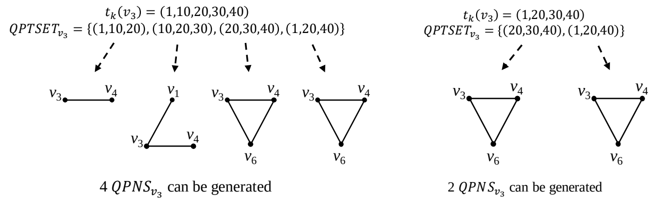
```
   Input: Temporal graph G, σ, k and ε
   Output: M, the set of all MQPCores in G
1  Let G be the de-temporal graph of G;
2  G_c = (V_c, E_c) ← k-core of G;
3  Initialize deg_{G_c}(u) for u ∈ V_c;
4  M ← ∅, X ← ∅, L ← ∅;
5  for u ∈ V_c in ascending order of deg_{G_c}(u) do
6      if u ∈ X then continue;
7      QPTSET_u ← ComputeQPTSET (u, k, σ, ε, G_c, G, X);
8      for qpt ∈ QPTSET_u do
           // pruning rule 2
9          if (qpt, u) ∈ L then continue;
10         G_u = (V_u, E_u) ← maximal connected subgraph of G_c
             containing u and ∀(u', v') ∈ E_u, qpt ⊆
             SupT_G((u', v')) ∧ ∀u' ∈ V_u, u' ∉ X ∧ (qpt, u') ∉ L;
11         L ← L ∪ {(qpt, u')|u' ∈ V_u};
12         SG ← sets of connected k-cores in G_u;
13         M ← M ∪ {(C', qpt)|C' ∈ SG};
       // pruning rule 1
14     X ← X ∪ {u};
       // update degree of vertices
15     Q ← an empty queue, Q.push(u);
16     while Q.empty() = false do
17         v ← Q.front(), Q.pop();
18         for w ∈ N_{G_c}(v) do
19             if w ∈ X then continue;
20             deg_{G_c}(w) ← deg_{G_c}(w) − 1;
21             if deg_{G_c}(w) < k then
22                 X ← X ∪ {w}, Q.push(w);
23 return M;
```

MQPCore obtained in $(G'_S, (20, 30, 40))$ is included in former MQPCores obtained in traversing v_3. To avoid the above redundant computations, our solution is to record vertices for all QPCS being computed, as described in the following pruning rule.

Pruning rule 2. For each QPCS $(G_S = (V_S, E_S), T)$ being computed, we record all tuples in form of $(T, u), u \in V_S$. In traversing each vertex v of the temporal graph, we skip all $T' \in QPTSET_v$ where (T', v) has been recorded.

Applying pruning rule 1 and 2 does not affect the integrity of the final results. This will be demonstrated in Theorem 7.

Algorithm 4 is the algorithm to mine all MQPCores based on our framework with the above two pruning rules. At first, Algorithm 4 prepares the de-temporal graph and its k-core to prune the search space (line 1-2). Algorithm 4 traverses all vertices in V_c with ascending order of their initial degree (line 5). In the first stage (line 7) of traversing each vertex,



(a) Without pruning rule 1 (b) With pruning rule 1

Fig. 7. Cases of traversing v_3 with and without pruning rule 1 in MQPClique enumeration. $\epsilon = 12\%, \sigma = 3, k = 3$

$QPTSET_u$ is computed in the pruned temporal graph (due to pruning rule 1) based on algorithms introduced in Section III. In the second stage, for each qpt (line 8), the algorithm computes $QPCS_u$ (line 10) unless (qpt, u) has been recorded (line 9, pruning rule 2). Then all vertices of $QPCS_u$ are recorded (line 11) and k-cores in $QPCS_u$ are computed (line 12). All new MQPCores are added into $\mathcal{M}$ (line 13). At last, the algorithm deletes u and vertices which can not retain degree not less than k (line 14-22, pruning rule 1).

Theorem 7. *Given a temporal graph $\mathcal{G}$ and ϵ, σ, k, Algorithm 4 correctly enumerates all MQPCores.*

Theorem 8. *Suppose* ComputeQPTSET *in Algorithm 4 is implemented by* QPT+*, the time and space complexity of Algorithm 4 are both $O(T_{max}^2(T_{max}\epsilon + 1)^{\sigma-1}(|V| + |E|))$. V, E are sets of vertices and edges in the de-temporal graph G. T is the set of timestamps in $\mathcal{G}$.*

C. The Second Stage (MQPClique Enumeration)

MQPClique enumeration is another objective of the second stage in traversing vertices. We first define a simpler local quasi-periodic subgraph for MQPClique enumeration: quasi-periodic neighbor subgraph.

Definition 10 (ϵ-quasi σ-periodic neighbor subgraph ((ϵ, σ)-**QPNS**)). *Given a temporal graph $\mathcal{G} = (V, \mathcal{E})$, its de-temporal graph G and ϵ, σ, k, for an (ϵ, σ, k)-QPV $u \in V$ and each $qpt \in (\epsilon, \sigma, k)$-$QPTSET_u$, an (ϵ, σ)-QPS $(G_S = (V_S, E_S), qpt)$ is an ϵ-quasi σ-periodic neighbor subgraph of u if $V_S = \{v | v \in N_G(u), qpt \subseteq SupT_{\mathcal{G}}((u, v))\} \cup \{u\}$ and $E_S = \{(u', v') | u', v' \in V_S, qpt \subseteq SupT_{\mathcal{G}}((u', v'))\}$.*

(ϵ, σ)-$QPNS_u$ or $QPNS_u$ represents a QPNS of (ϵ, σ, k)-QPV u. Examples of QPNS can be found in Figure 7.

Theorem 9. *Given a temporal graph $\mathcal{G} = (V, \mathcal{E})$ and ϵ, σ, k, all MQPCliques can be obtained in $QPNS_u$, $u \in V$.*

The proof of Theorem 9 is similar to the proof of Theorem 6. With Theorem 9, we can use the classical Bron-Kerbosch algorithm [10] with pivot technique to compute all MQPCliques in those QPNS. Similarly, **pruning rule 1** can be applied to MQPClique enumeration to achieve shorter $t_k(u)$, smaller QPTSET and fewer QPNS, which are shown in Figure 7. **pruning rule 2** is not used in MQPClique enumeration since there are already similar techniques to prevent redundant computations in Bron-Kerbosch algorithm with pivot technique.

Based on a similar framework in Algorithm 4, we directly present Algorithm 5, the algorithm that enumerates all MQPCliques in a temporal graph based on the classical Bron-Kerbosch algorithm with pivot technique. The main

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

Input: Temporal graph $\mathcal{G}$, σ, k and ϵ
Output: $\mathcal{M}$, the set of all MQPCliques in $\mathcal{G}$
1 Let G be the de-temporal graph of $\mathcal{G}$;
2 $G_c = (V_c, E_c) \leftarrow (k-1)$-core of G; // limit for degree is $k-1$
3 Initialize $deg_{G_c}(u)$ for $u \in V_c$;
4 $\mathcal{M} \leftarrow \emptyset$, $X \leftarrow \emptyset$;
5 **for** $u \in V_c$ *in ascending order of* $deg_{G_c}(u)$ **do**
6 　**if** $u \in X$ **then** continue;
7 　$QPTSET_u \leftarrow$ ComputeQPTSET $(u, k-1, \sigma, \epsilon, G_c, \mathcal{G}, X)$;
8 　**for** $qpt \in QPTSET_u$ **do**
　　// compute QPNS and MQPClique
9 　　$P \leftarrow \emptyset$, $R \leftarrow \emptyset$, $\tilde{X} \leftarrow \emptyset$;
10 　　$R \leftarrow R \cup \{u\}$;
11 　　**for** $v \in N_{G_c}(u)$ **do**
12 　　　**if** $qpt \subseteq SupT_{\mathcal{G}}((u, v))$ **then**
13 　　　　**if** $v \in X$ **then** $\tilde{X} \leftarrow \tilde{X} \cup \{v\}$;
14 　　　　**else** $P \leftarrow P \cup \{v\}$;
15 　　BKPivot $(P, R, \tilde{X}, \mathcal{M}, k, qpt, \mathcal{G}, G_c)$;
　// pruning rule 1
16 　$X \leftarrow X \cup \{u\}$;
17 　$Q \leftarrow$ an empty queue, $Q.push(u)$;
18 　**while** $Q.empty() = false$ **do**
19 　　$v \leftarrow Q.front()$, $Q.pop()$;
20 　　**for** $w \in N_{G_c}(v)$ **do**
21 　　　**if** $w \in X$ **then** continue;
22 　　　$deg_{G_c}(w) \leftarrow deg_{G_c}(w) - 1$;
23 　　　**if** $deg_{G_c}(w) < k-1$ **then**
24 　　　　$X \leftarrow X \cup \{w\}$, $Q.push(w)$;
25 **return** $\mathcal{M}$;

26 **Procedure** BKPivot $(P, R, X, \mathcal{M}, k, qpt, \mathcal{G}, G)$
27 **if** $|P| + |R| < k$ **then return**;
28 **if** $P = \emptyset \wedge X = \emptyset$ **then** $\mathcal{M} \leftarrow \mathcal{M} \cup \{(R, qpt)\}$, **return**;
29 $u' \leftarrow \arg \max_{u \in P \cup X} |P \cap \{v | v \in N_G(u), qpt \subseteq SupT_{\mathcal{G}}((u, v))\}|$;
30 **for** $u \in P - \{v | v \in N_G(u'), qpt \subseteq SupT_{\mathcal{G}}((u', v))\}$ **do**
31 　$P' \leftarrow P \cap \{v | v \in N_G(u), qpt \subseteq SupT_{\mathcal{G}}((u, v))\}$;
32 　$X' \leftarrow X \cap \{v | v \in N_G(u), qpt \subseteq SupT_{\mathcal{G}}((u, v))\}$;
33 　$R' \leftarrow R \cup \{u\}$;
34 　BKPivot $(P', R', X', \mathcal{M}, k, qpt, \mathcal{G}, G)$;
35 　$P \leftarrow P - \{u\}$, $X \leftarrow X \cup \{u\}$;

differences between Algorithm 5 and Algorithm 4 lie in line 9-15 and the BKPivot procedure. In line 9-15, the algorithm searches vertices of $QPNS_u$ among all neighbors of u (line 11-14) and computes maximal cliques in $QPNS_u$ (line 15). In BKPivot procedure, only edges in $QPNS_u$ will be visited, as shown in line 29-32 (e.g., $qpt \subseteq SupT_{\mathcal{G}}((u, v))$ in line 29).

Theorem 10. *Given a temporal graph $\mathcal{G}$ and ϵ, σ, k, Algorithm 5 correctly enumerates all MQPCliques.*

Theorem 11. *Suppose* ComputeQPTSET *in Algorithm 5 is implemented by* QPT+*, the time complexity of Algorithm 5 is $O(T_{max}^2(T_{max}\epsilon + 1)^{\sigma-1}|V|3^{|V|/3})$. V, E are sets of vertices and edges in the de-temporal graph G. T is the set of timestamps in $\mathcal{G}$.*

Based on Moon et al. [11], any n-vertex graph has at most $3^{n/3}$ maximal cliques, we can easily derive the following theorem.

Theorem 12. *Given a temporal graph $\mathcal{G}$, σ, k, ϵ and its de-temporal graph $G = (V, E)$, the number of MQPCliques in $\mathcal{G}$ is at most $T_{max}^2(T_{max}\epsilon + 1)^{\sigma-1}3^{|V|/3}$.*

V. Experiments

In this section we conduct extensive experiments to evaluate the effectiveness and efficiency of algorithms for QPT mining,

TABLE I
DATASETS-TEMPORAL GRAPHS

Datasets	$\|V\|$	$\mathcal{E}$	$\|T\|$	Time scale
Hospital	75	19,274	3,567	minute
LKML	26,885	547,814	2,922	day
Enron	86,978	912,762	1,235	day
DBLP	1,207,754	8,072,560	80	year
IMDB	3,497,300	37,096,631	144	year

MQPCore enumeration and MQPClique enumeration.

Datasets. In this work we use five real-world temporal graphs as the datasets. Table I shows the basic information of the five real-world temporal graphs (T represents the set of timestamps in all edges of each dataset). Hospital [12] is downloaded from http://www.sociopatterns.org/datasets/, LKML and Enron are downloaded from http://konect.cc/. DBLP [13] is extracted from dblp computer science bibliography data which is downloaded in September, 2021. IMDB [13] is extracted from the Internet Movie Database (IMDB) downloaded in November, 2021.

Algorithms. For QPT mining, we implement two algorithms: QPT and QPT+. QPT is the basic method (Algorithm 1) and QPT+ is the DAG oracle based method (Algorithm 3).

For MQPCore enumeration, we first present two baseline method, TsetCO and TsetCO+. In TsetCO, we first mine all QPT in the entire timestamp set of temporal networks using QPT. Then for each QPT, we extract the common snapshot in timestamps of the QPT (i.e., QPS) and perform core decomposition. TsetCO+ is similar to TsetCO but uses QPT+ to mine all QPT.

We also modify framework for (strict) periodic clique mining proposed in [6] as a comparison. In the modified framework, the first step is computing QPTSET using QPT or QPT+ for all vertices, then all QPS are computed based on QPT in those QPTSET. Then algorithm for community mining is invoked. MQPCO-B and MQPCO-B+ represent the modified framework with QPT and QPT+ respectively.

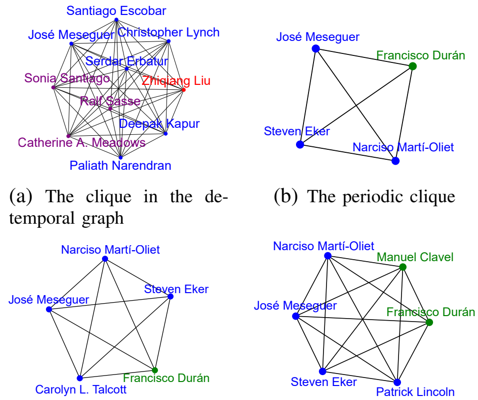
MQPCO-E and MQPCO-E+ represent our framework (Algorithm 4) where ComputeQPTSET is implemented by QPT and QPT+ respectively.

For MQPClique enumeration, we use the same frameworks above. Six similar algorithms are implemented: TsetCL and TsetCL+ based on the baseline method; MQPCL-B and MQPCL-B+ based on the modified framework; MQPCL-E and MQPCL-E+ based on our framework (Algorithm 5).

Parameters. There are three parameters in this work, σ, k, ϵ. The default value of σ is 6. For k, the default value is 3 (for MQPClique enumeration, we set the default value to 4 to keep the same degree limit). For ϵ, we set 4 levels: $5\%, 10\%, 15\%, 20\%$, with default value 5%.

A. Effectiveness Evaluation

Case study on DBLP. We compare the results of clique, periodic clique and quasi-periodic clique to evaluate the effectiveness of the community models. Prof. José Meseguer is with UIUC, and his research fields include computer theory, software engineering, computer architecture and so on. Figure 8 (a) shows the clique containing Prof. Meseguer in the de-temporal graph of DBLP. As can be seen, Figure 8 (a) contains many researchers who do not co-authored with Prof. Meseguer regularly and researchers who are in other research areas such as network security (Catherine A. Meadows, Ralf



(a) The clique in the de-temporal graph
(b) The periodic clique
(c) The MQPClique, $\epsilon = 20\%$
(d) The MQPClique, $\epsilon = 50\%$

Fig. 8. Case study in DBLP (Communities of Prof. José Meseguer). Researchers of different fields are in different colors: computer theory (blue), software engineering (green), computer architecture (red), network security (purple)

Sasse) and computer architecture (Zhiqiang Liu). This is because that the clique model do not consider the temporal information, so once a researcher has a short-term connection with Prof. Meseguer, he will be added into the community. Figure 8 (b) is the periodic clique, it shows a small close community with periodic co-authorship (years 2009-2016) of three researchers. Interestingly, according to dblp.org/pid/m/JoseMeseguer.html, the three researchers (Narciso Marti-Oliet, Francisco Duran, Steven Eker) are the 2nd-4th most collaborators with Prof. Meseguer. Figure 8 (c) shows the quasi-periodic clique with $k = 5, \sigma = 3, \epsilon = 20\%$. In this community, we can find a new member Carolyn L. Talcott, who is the 5th most collaborator with Prof. Meseguer. As seen in Figure 8 (d), the MQPClique with $\epsilon = 50\%$ contains a new researcher, Manuel Clavel, which is the 6th most collaborator with Prof. Meseguer. Therefore, we find that (i) the periodic clique and quasi-periodic clique can find more accurate periodic communities than the clique model; (ii) the quasi-periodic clique can find larger actual but not strictly periodical communities than the periodic clique model. In conclude, our proposed MQPClique model is more effective than the other models.

Quantification. In the preceding case study, we provide examples to demonstrate the effectiveness of MQPClique to find larger actual but not strictly periodical communities for a specific vertex (a researcher in DBLP). Here we quantitatively assess the aforementioned effectiveness by computing the proportion of those vertices (the vertices for which the size of the maximum community containing the vertex has increased with the growth of ϵ) in the initial periodic cliques. In detail, consider a graph $G = (V, E)$ and given σ, k, ϵ, let V_ϵ be the set of all vertices in MQPCliques in G, and $max_\epsilon(u), u \in V_\epsilon$ be the size of the maximum MQPClique containing u. Suppose V_ϵ^{inc} is the set of vertices for which the size of the maximum MQPClique containing the vertex has increased, i.e., $V_\epsilon^{inc} \subseteq V_0 \cap V_\epsilon, \forall u \in V_\epsilon^{inc}, max_\epsilon(u) > max_0(u)$, Table II shows the proportion of vertices in V_ϵ^{inc} relative to those in V_0 ($|V_\epsilon^{inc}|/|V_0|$) with different ϵ and $k = 4, \sigma = 3$. We can see that in DBLP, the value of $|V_\epsilon^{inc}|/|V_0|$ increases at a slow rate with the growth of ϵ. The reason is that DBLP is a dataset with coarse-grained timestamp unit (year), so that discovering quasi-periodic communities in DBLP is more

TABLE II
QUANTIFICATION OF EFFECTIVENESS-THE VALUE OF $|V_{\epsilon}^{inc}|/|V_0|$ WITH DIFFERENT ϵ. $k = 4, \sigma = 3$

	$\epsilon: 5\%$	$\epsilon: 10\%$	$\epsilon: 15\%$	$\epsilon: 20\%$	$\epsilon: 50\%$
DBLP	0%	0%	$< 0.01\%$	$< 0.1\%$	1.1%
Enron	1.7%	7.3%	10.6%	13.9%	18.4%

difficult especially under small ϵ (the specific reasons will be explained in **Exp-5**). However, V_0 in DBLP is sufficiently large, so we can still find examples for conducting a case study. In Enron, we can observe a noticeable increase in $|V_{\epsilon}^{inc}|/|V_0|$ with the growth of ϵ, indicating that by mining MQPCliques, an increasing number of vertices in the initial periodic cliques can be found in larger actual but not strictly periodical communities, which further demonstrates the effectiveness of the MQPClique model.

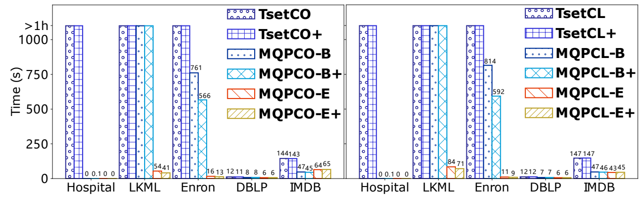
B. Efficiency Evaluation

In this subsection we evaluate the efficiency of algorithms for QPT mining, MQPCore and MQPClique enumeration.

Exp-1: Efficiency of algorithms for QPT mining. Table III shows the running time of QPT and QPT+ on two time sequences of different length selected from $t_k(u)$s in LKML with varying parameters. As we can see in Table III, QPT+ is more efficient than QPT especially when σ gets larger. The reason is that when σ gets larger, QPT has to maintain and traverse larger candidate set and QPT+ can still be accelerated by auxiliary arrays. More interestingly, the running time of QPT+ decreases when σ continues to increase, which is because the number of (ϵ, σ)-QPT has to decrease when σ continues to increase and the auxiliary arrays in QPT+ guide the algorithm to only access the correct path. From another perspective we can see that the running time increases when ϵ gets larger, and the reason is that larger ϵ causes more QPT in a sequence. When $|T| = 200$, both QPT and QPT+ can not end in an hour if $\epsilon = 20\%, \sigma \geq 8$, which is because the number of QPT become extremely large in these cases. As a result, it is important to keep a smaller $|T|$ (or $|t_k(u)|$) in following algorithms to enumerate MQPCores and MQPCliques, which is exactly what the pruning rule 1 does as we introduced in Section IV-B.

Table IV shows the time used in building DAG oracle (Algorithm 2) for QPT+. As we analyzed in Theorem 4, more time is needed for larger ϵ or $|T|$ in building DAG oracle. However, as ϵ or $|T|$ increases, the rate of increase in time used for building DAG oracle is much smaller than that of QPT+ in most cases. This proves that building DAG oracle is not a performance bottleneck in most cases.

Exp-2: Memory usage of the algotirhms for QPT mining. Table V shows the memory usage of QPT and QPT+. We extract data that $\sigma = 6, \sigma = 10$ and $\epsilon = 5\%, \epsilon = 15\%$. We can see that increasing ϵ both two algorithms take more memory, because larger ϵ leads to larger candidate set in QPT, larger DAG oracle in QPT+ and more QPT in T. However, the memory usage of the two algorithms does not correlate strongly with σ, which is mainly because the number of QPT in T does not always increase with σ. In most cases especially when $|T| = 200$, QPT+ takes less space because the DAGs in DAG oracle are compact structures and QPT+ does not need to maintain a large candidate set.



(a) MQPCore enumeration (b) MQPClique enumeration

Fig. 9. Running time of algorithms for MQPCore enumeration and MQPClique enumeration with default parameters

Exp-3: Efficiency of algorithms for MQPCore and MQPClique enumeration on all datasets with default parameters. In this experiment we first use the default parameters to evaluate the efficiency of algorithms for MQPCore and MQPClique enumeration on all datasets. Figure 9 (a) shows the running time of the 6 algorithms for MQPCore enumeration. We can see that TsetCO and TsetCO+ can not be applied to dataset like Hospital, LKML and Enron, because there are much more timestamps in these three temporal networks, and it is inefficient to mine QPT directly in timestamp set of these three datasets. In DBLP, IMDB and Hospital, MQPCO-E and MQPCO-E+ has no advantage over MQPCO-B and MQPCO-B+ respectively, but in LKML and Enron, MQPCO-E and MQPCO-E+ are more than 10 times faster than MQPCO-B and MQPCO-B+ respectively. The reason is that LKML and Enron have much more timestamps than DBLP and IMDB, as well as larger $t_k(u)$, QPTSET. In this case, our pruning rules especially the pruning rule 1 can help to avoid numerous computations. For example, the length of the top-5 longest $t_k(u)$s in Enron using MQPCO-B and MQPCO-B+ are 448, 435, 362, 339, 295, while such data using MQPCO-E and MQPCO-E+ are 219, 204, 150, 150, 149. The size of the top-5 largest QPTSET in Enron using MQPCO-B and MQPCO-B+ are 2139157, 2132108, 794811, 310340, 227683, while such data using MQPCO-E and MQPCO-E+ are 42355, 29345, 4921, 3731, 2561. Such differences in LKML is even more dramatic, so that MQPCO-B and MQPCO-B+ can not end in an hour. We can also find that algorithms integrated with QPT+ works better than algorithms integrated with QPT in LKML and Enron. This is because most $t_k(u)$s in DBLP and IMDB are much shorter than 100, and in these cases QPT works better than QPT+ slightly due to the preparation for DAG oracle. The results in MQPClique enuemration (Figure 9 (b)) and the reasons are similar.

Exp-4: Efficiency of algorithms for MQPCore and MQPClique enumeration with varying k, σ. We then evaluate the efficiency of algorithms for MQPCore and MQPClique enumeration with varying k, σ. Figure 10 (a) and (b) show the running time of algorithms for MQPCore enumeration on Enron with varying k and σ. We can see that TsetCO and TsetCO+ still can not handle dataset like Enron. For other algorithms, it is clear that increasing either k or σ strengthens the constraint on MQPCores, so the running time of the 4 remaining algorithms decreases. We can see that in most cases, algorithms implemented with our framework (MQPCO-E, MQPCO-E+) are faster than the other two algorithms. We can also find that QPT+ can do help to the enumeration progress especially when σ gets larger by comparing MQPCO-B and MQPCO-B+ (or MQPCO-E and MQPCO-E+). Such results confirm our analysis in **Exp-1**. In Figure 10 (c) and (d), as in the previous experiment, both DAG oracle based method (QPT +) and our framework have no significant effect due to fewer

TABLE III
RUNNING TIME (MS) OF ALGORITHMS FOR QPT MINING IN TIME SEQUENCE T WITH DIFFERENT LENGTH

	$\|T\| = 100$								$\|T\| = 200$							
	$\epsilon : 5\%$		$\epsilon : 10\%$		$\epsilon : 15\%$		$\epsilon : 20\%$		$\epsilon : 5\%$		$\epsilon : 10\%$		$\epsilon : 15\%$		$\epsilon : 20\%$	
	QPT	QPT+	QPT	QPT+	QPT	QPT+	QPT	QPT+	QPT	QPT+	QPT	QPT+	QPT	QPT+	QPT	QPT+
$\sigma : 4$	31	31	**55**	56	84	**82**	121	**115**	391	**269**	1,980	**1,710**	**6,911**	6,968	**17,500**	21,772
$\sigma : 6$	37	**31**	74	**49**	159	**86**	373	**201**	744	**130**	12,500	**4,992**	125,002	**109,830**	**797,769**	1,504,672
$\sigma : 8$	33	**27**	82	**44**	209	**62**	626	**106**	1019	**85**	41,034	**5,681**	**1,117,753**	1,340,381	INF	INF
$\sigma : 10$	32	**27**	82	**43**	224	**55**	788	**81**	1112	**77**	64,507	**1,826**	2,573,018	**346,324**	INF	INF

TABLE IV
TIME USED (MS) IN BUILDING DAG ORACLE FOR QPT+

	$\epsilon : 5\%$	$\epsilon : 10\%$	$\epsilon : 15\%$	$\epsilon : 20\%$
$\|T\| = 100$	23	39	49	57
$\|T\| = 200$	57	96	127	159

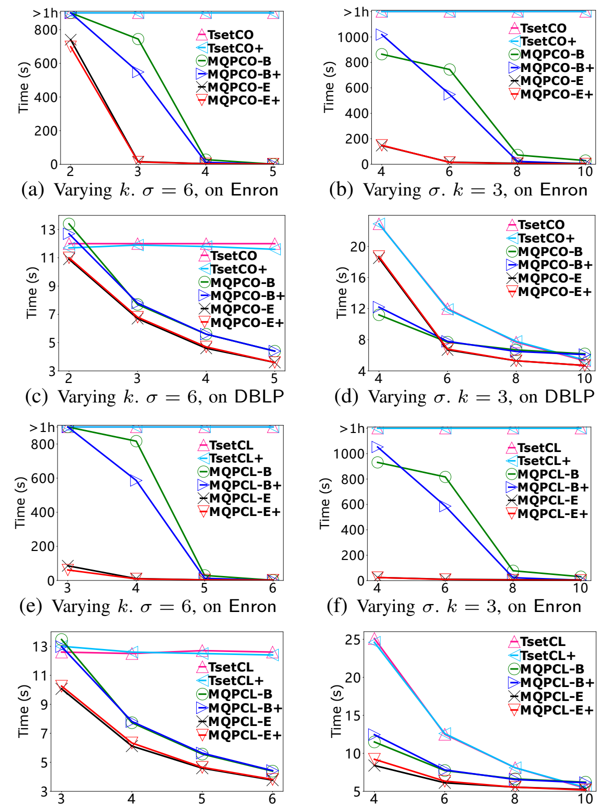
TABLE V
MEMORY USAGE (MB) OF QPT AND QPT+

	$\|T\| = 100$				$\|T\| = 200$			
	$\epsilon : 5\%$		$\epsilon : 15\%$		$\epsilon : 5\%$		$\epsilon : 15\%$	
	QPT	QPT+	QPT	QPT+	QPT	QPT+	QPT	QPT+
$\sigma : 6$	**2.14**	3.19	5.32	**4.24**	12.47	**4.24**	228.14	**137.22**
$\sigma : 10$	**1.55**	3.19	7.05	**4.24**	14.03	**3.83**	1,094.70	**276.51**

timestamps (or shorter $t_k(u)$) in DBLP. In Figure 10 (c), the performance of TsetCO and TsetCO+ is almost unaffected by varying k, because the bottleneck of these two algorithms lies in mining QPT in the first step. If σ is fixed, the number of QPT and QPS computed in the first step will not change. In MQPClique enumeration, Figure 10 (e) and (f) and Figure 10 (g) and (h) show similar results.

Exp-5: Efficiency of algorithms for MQPCore and MQPClique enumeration with varying ϵ. Next we evaluate the efficiency of algorithms for MQPCore and MQPClique enumeration with varying ϵ and default k, σ. Figure 11 (a) and (b) show the running time of algorithms for MQPCore enumeration with varying ϵ on Enron and DBLP. We can see that in Figure 11 (a), increasing ϵ puts heavy load on the six algorithms, but MQPCO-E and MQPCO-E+ still outperforms the other four algorithms significantly, which proves the effectiveness of our framework. We can also see that on DBLP (Figure 11 (b)), MQPCO-E and MQPCO-E+ are slightly faster than the other four algorithms. However, except TsetCO and TsetCO+, all these algorithms are not sensitive to ϵ. This is because the unit of timestamps in DBLP is year, and many different events are recorded to have occurred in the same year, although they did not happen at the same time. Therefore, many potential quasi-periodic events are inaccurately recorded as strict periodic events. TsetCO and TsetCO+ are sensitive to ϵ because they directly mine QPT in the whole timestamp set of DBLP. All results above can still be found in MQPClique enumeration (Figure 11 (c) and (d)). In conclusion, on datasets with fine-grained timestamp unit, such as Enron, ϵ can be set from a smaller starting value (5%). On datasets with coarse-grained timestamp unit, such as DBLP, ϵ can be set from a larger starting value, such as 15%.

Exp-6: Scalability. In this experiment we evaluate the scalability of MQPCL-E+ on IMDB and a larger dataset DeWiki. DeWiki is downloaded in konect (http://konect.cc/) and has 2 million vertices, 68 million temporal edges and 3,668 different timestamps. The results are shown in Figure 12. In each dataset, we sample vertices and timestamps respectively. In sampling timestamps, for example, if 20% of the timestamps are sampled, then only temporal edges with the smallest 20% of timestamps will be retained. We can see that with



(a) Varying k. $\sigma = 6$, on Enron
(b) Varying σ. $k = 3$, on Enron
(c) Varying k. $\sigma = 6$, on DBLP
(d) Varying σ. $k = 3$, on DBLP
(e) Varying k. $\sigma = 6$, on Enron
(f) Varying σ. $k = 3$, on Enron
(g) Varying k. $\sigma = 6$, on DBLP
(h) Varying σ. $k = 3$, on DBLP

Fig. 10. Running time of algorithms for MQPCore enumeration and MQPClique enumeration with varying k, σ. $\epsilon = 5\%$

increasing $|V|$ and $|T|$, the running time of MQPCL-E+ increases smoothly in both two datasets, which suggests that our algorithm is scalable.

Exp-7: Memory usage of the framework. In this experiment we evaluate the memory usage of our framework in MQPClique enumeration using TsetCL+, MQPCL-B+, and MQPCL-E+. Table VI shows the memory usage of these algorithms in all datasets with the default parameters. We can see that in the first three datasets, especially in LKML and Enron, although the temporal graph itself is smaller than DBLP and IMDB, with much more timestamps, TsetCL+ is unable to complete QPT mining in these datasets, and MQPCL-B+ have to store numerous QPTs and quasi-periodic subgraphs. On the contrary, MQPCL-E+ achieve smaller $t_k(u)$ and store only a few QPTs and local quasi-periodic subgraphs. As a result, MQPCL-E+ consume less than 10% of the memory that is consumed by MQPCL-B+. We can also see that in DBLP and IMDB, the memory usage of the three algorithms are close, which is because the temporal graph itself is pretty large but the number of timestamps is small, and only a small number of quasi-periodic subgraphs will be

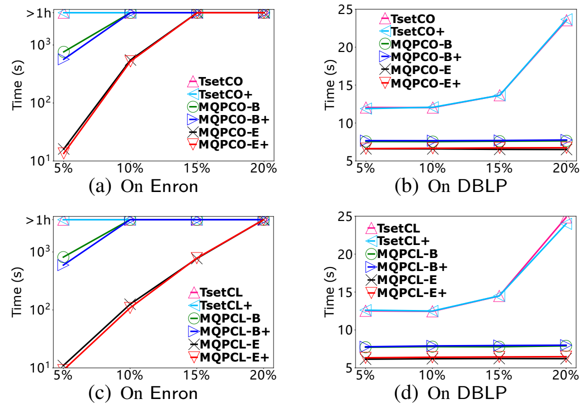


Fig. 11. Running time of algorithms for MQPCore ($k = 3, \sigma = 6$) and MQPClique ($k = 4, \sigma = 6$) enumeration with varying ϵ on Enron and DBLP

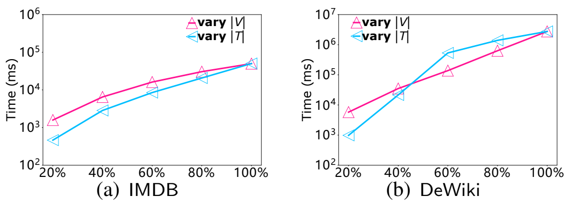


Fig. 12. Scalability testing of MQPCL-E+ on IMDB and DeWiki with default parameters

TABLE VI
MEMORY USAGE (MB) OF THE FRAMEWORK

	TsetCL+	MQPCL-B+	MQPCL-E+
Hospital	> 658.5	**2.3**	2.4
LKML	> 468.8	> 10,000	**66.5**
Enron	> 885.4	3,685.7	**105.2**
DBLP	1,974.9	**1,612.3**	1,613.4
IMDB	**7,156.9**	7,941.5	7,940.7

generated in these algorithms.

VI. Related Work

Periodic subgraph mining. Our work is related to periodic subgraphs mining in temporal networks. Previous research on this topic, such as [3]–[5], focused solely on identifying periodic subgraphs without considering cohesive structures such as communities or the quasi-periodicity of these structures. Zhang et al. [7] studied the problem of mining seasonal-periodic subgraphs, which exhibit periodicity for multiple particular periods (the differences between adjacent timestamps) over multiple consecutive time intervals in temporal networks. However, they only set an upper limit for those periods and did not consider the lower limits. Qin et al. [6] focused on mining periodic cliques in temporal networks but did not consider quasi-periodicity. Meanwhile, Zhang et al. [14] proposed PERCeIDs for periodic community detection using tensor factorization and period estimation. However, estimating a period for a periodic community over the entire time range of a temporal network may not be convincing since not all periodic behaviors persist throughout the entire duration of real-world datasets. Additionally, they did not address the quasi-periodicity of communities.

Quasi-periodic behavior mining in sequence data. Our work focuses on quasi-periodic behavior mining, which may also occur in time series or event sequences such as computer monitoring logs [15] or transaction histories. Previous research on this topic includes the work of Yang et al. [16], who first introduced the problem of asynchronous (or quasi) periodic pattern mining in time series data. Huang et al. [17] proposed a more general model of asynchronous periodic patterns where a time slot can contain multiple events. More recent works on asynchronous periodic pattern mining include [18]–[21], but all of these approaches use a user-specified value to limit the range of periods, which can not adapt to the period itself (e.g., the range of periods of yearly gathering should be larger than that of weekly meeting). Other works focus on mining quasi-periodic patterns in integer sequences, such as the work of Gfeller [8] and Amir et al. [22]. In their works, the range of periods is adaptive to the minimum period. However, their focus is on mining the longest quasi-periodic pattern, while our work aims to mine all fixed-length quasi-periodic patterns, which is more challenging. There are also researchers who have studied mining quasi-periodic behaviors in spatio-temporal event sequence, such as [23]–[28]. Their works did not consider quasi-periodic behavior in temporal networks.

Cohesive subgraph mining. Our work is also related to cohesive subgraph mining. There are many recent works that model cohesive subgraphs by k-core or k-clique. For example, Kim et al. [29] propose (p, n)-core that combines k-core and signed edges. Dai et al. [30] mine (k, η)-cores in uncertain graphs. Liao et al. [31] mine D-core in directed graphs, which can also be seen as a variant of k-core. For k-clique, the classical Bron-Kerbosch algorithm [10] and its variants [32]–[34] are widely used. Some recent works focus on fairness-aware maximal clique enumeration [35], maximal clique enumeration on uncertain graphs [36], maximum clique enumeration [37], and parallelizing maximal clique enumeration [38]. In temporal networks, Li et al. [39] developed an algorithm to detect persistent communities (modeled by k-core) over time in temporal networks. However, unlike these works, our work focuses on the problem of mining quasi-periodic cohesive subgraphs in temporal networks.

VII. Conclusion

In this paper, we address the problem of mining quasi-periodic communities in temporal networks. We propose two novel models for modeling quasi-periodic communities: maximal ϵ-quasi σ-periodic k-core and maximal ϵ-quasi σ-periodic k-clique. To efficiently mine these quasi-periodic communities, we develop a two-stage framework. In the first stage, we develop a novel DAG oracle based method to identify all quasi-periodic sub-sequences, which enables us to quickly determine whether a vertex can be part of a quasi-periodic community. In the second stage, our framework computes local quasi-periodic subgraphs containing each vertex and then detects quasi-periodic communities within these subgraphs. In addition, we also propose several non-trivial pruning rules to further speed up the proposed algorithms. Finally, we evaluate the effectiveness and efficiency of our algorithms using five real-world temporal graphs. Our experimental results demonstrate the superiority of the proposed methods.

Acknowledgement. This work was partially supported by (I) the National Key Research and Development Program of China 2020AAA0108503, and (II) NSFC Grants U2241211, 62072034. Rong-Hua Li is the corresponding author of this paper.

REFERENCES

[1] B. Viswanath, A. Mislove, M. Cha, and P. K. Gummadi, "On the evolution of user interactions in facebook," *Proceedings of the Acm Workshop on Online Social Networks*, vol. 39, no. 4, pp. 37–42, 2009.

[2] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet," in *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, 2002, pp. 96–107.

[3] M. Lahiri and T. Y. Berger-Wolf, "Mining periodic behavior in dynamic social networks," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 373–382.

[4] M. Lahiri and T. Y. Berger-Wolf, "Periodic subgraph mining in dynamic networks," *Knowledge and Information Systems*, vol. 24, no. 3, pp. 467–497, 2010.

[5] S. Halder, M. Samiullah, and Y.-K. Lee, "Supergraph based periodic pattern mining in dynamic social networks," *Expert Systems with Applications*, vol. 72, pp. 430–442, 2017.

[6] H. Qin, R.-H. Li, G. Wang, L. Qin, Y. Cheng, and Y. Yuan, "Mining periodic cliques in temporal networks," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1130–1141.

[7] Q. Zhang, D. Guo, X. Zhao, X. Li, and X. Wang, "Seasonal-periodic subgraph mining in temporal networks," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 2309–2312.

[8] B. Gfeller, "Finding longest approximate periodic patterns," in *Workshop on Algorithms and Data Structures*. Springer, 2011, pp. 463–474.

[9] Y. Zeng, H. Qin, R.-H. Li, K. Wang, G. Wang, and X. Lin, "Mining quasi-periodic communities in temporal network," 2023. [Online]. Available: https://github.com/bruce1114/qpcommunity/blob/master/supplementation/main-icde-fullversion.pdf

[10] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[11] J. W. Moon and L. Moser, "On cliques in graphs," *Israel journal of Mathematics*, vol. 3, no. 1, pp. 23–28, 1965.

[12] P. Vanhems, A. Barrat, C. Cattuto, J.-F. Pinton, N. Khanafer, C. Régis, B.-a. Kim, B. Comte, and N. Voirin, "Estimating potential infection transmission routes in hospital wards using wearable proximity sensors," *PloS one*, vol. 8, no. 9, p. e73970, 2013.

[13] G. Wang, Y. Zeng, R.-H. Li, H. Qin, X. Shi, Y. Xia, X. Shang, and L. Hong, "Temporal graph cube," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[14] L. Zhang, A. Gorovits, and P. Bogdanov, "Perceids: Periodic community detection," in *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2019, pp. 816–825.

[15] J. Koumar and T. Čejka, "Network traffic classification based on periodic behavior detection," in *2022 18th International Conference on Network and Service Management (CNSM)*. IEEE, 2022, pp. 359–363.

[16] J. Yang, W. Wang, and P. S. Yu, "Mining asynchronous periodic patterns in time series data," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 275–279.

[17] K.-Y. Huang and C.-H. Chang, "Smca: a general model for mining asynchronous periodic patterns in temporal databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 774–785, 2005.

[18] F. Rasheed, M. Alshalalfa, and R. Alhajj, "Efficient periodicity mining in time series databases using suffix trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 1, pp. 79–94, 2010.

[19] X. Yu and H. Yu, "An asynchronous periodic sequential patterns mining algorithm with multiple minimum item supports," in *2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. IEEE, 2014, pp. 274–281.

[20] G. Sirisha and M. Shashi, "A hash-based algorithm for mining non-redundant asynchronous periodic patterns," *International Journal of Business Intelligence and Data Mining*, vol. 11, no. 3, pp. 205–228, 2016.

[21] J.-W. Huang, B. P. Jaysawal, and C.-C. Wang, "Mining full, inner and tail periodic patterns with perfect, imperfect and asynchronous periodicity simultaneously," *Data Mining and Knowledge Discovery*, vol. 35, no. 4, pp. 1225–1257, 2021.

[22] A. Amir, A. Apostolico, E. Eisenberg, G. M. Landau, A. Levy, and N. Lewenstein, "Detecting approximate periodic patterns," *Theoretical Computer Science*, vol. 525, pp. 60–67, 2014.

[23] H. Cao, N. Mamoulis, and D. W. Cheung, "Discovery of periodic patterns in spatiotemporal sequences," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 4, pp. 453–467, 2007.

[24] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye, "Mining periodic behaviors for moving objects," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 1099–1108.

[25] F. Yi, L. Yin, H. Wen, H. Zhu, L. Sun, and G. Li, "Mining human periodic behaviors using mobility intention and relative entropy," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2018, pp. 488–499.

[26] D. Zhang, K. Lee, and I. Lee, "Hierarchical trajectory clustering for spatio-temporal periodic pattern mining," *Expert Systems with Applications*, vol. 92, pp. 1–11, 2018.

[27] D. Zhang, K. Lee, and I. Lee, "Semantic periodic pattern mining from spatio-temporal trajectories," *Information Sciences*, vol. 502, pp. 164–189, 2019.

[28] D. Zhang, K. Lee, and I. Lee, "Mining hierarchical semantic periodic patterns from gps-collected spatio-temporal trajectories," *Expert Systems with Applications*, vol. 122, pp. 85–101, 2019.

[29] J. Kim and S. Lim, "(p, n)-core: Core decomposition in signed networks," in *Database Systems for Advanced Applications: 27th International Conference, DASFAA 2022, Virtual Event, April 11–14, 2022, Proceedings, Part I*. Springer, 2022, pp. 543–551.

[30] Q. Dai, R.-H. Li, G. Wang, R. Mao, Z. Zhang, and Y. Yuan, "Core decomposition on uncertain graphs revisited," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 1, pp. 196–210, 2021.

[31] X. Liao, Q. Liu, J. Jiang, X. Huang, J. Xu, and B. Choi, "Distributed d-core decomposition over large directed graphs," *Proc. VLDB Endow.*, vol. 15, no. 8, pp. 1546–1558, 2022.

[32] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical computer science*, vol. 363, no. 1, pp. 28–42, 2006.

[33] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," *Journal of Experimental Algorithmics (JEA)*, vol. 18, pp. 3–1, 2013.

[34] K. A. Naudé, "Refined pivot selection for maximal clique enumeration in graphs," *Theoretical Computer Science*, vol. 613, pp. 28–37, 2016.

[35] M. Pan, R.-H. Li, Q. Zhang, Y. Dai, Q. Tian, and G. Wang, "Fairness-aware maximal clique enumeration," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 259–271.

[36] Q. Dai, R. Li, M. Liao, H. Chen, and G. Wang, "Fast maximal clique enumeration on uncertain graphs: A pivot-based approach," in *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 2022, pp. 2034–2047.

[37] C. Lu, J. X. Yu, H. Wei, and Y. Zhang, "Enumerating maximum cliques in massive graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 9, pp. 4215–4230, 2020.

[38] M. Almasri, Y.-H. Chang, I. E. Hajj, R. Nagi, J. Xiong, and W.-m. Hwu, "Parallelizing maximal clique enumeration on gpus," *arXiv preprint arXiv:2212.01473*, 2022.

[39] R.-H. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai, "Persistent community search in temporal networks," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 2018, pp. 797–808.