Mining Periodic k-Clique from Real-World Sparse Temporal Networks

Zebin Ren[1], Hongchao Qin[1], Rong-Hua Li[1(✉)], Yongheng Dai[2], Guoren Wang[1], and Yanhui Li[3]

[1] Beijing Institute of Technology, Beijing, China
lironghuabit@126.com
[2] Diankeyun Technologies Co., Ltd., Beijing, China
[3] Chongqing Jiaotong University, Chongqing, China

Abstract. In temporal networks, nodes and edges are associated with time series. To seeking the periodic pattern in temporal networks, an intuitive method is to searching periodic communities in them. However, most existing studies do not exploit the periodic pattern of communities. The only few works left do not take the sparse propriety of real-world temporal networks into consideration, such that (i) the answers searched for are few, (ii) the computation suffers from poor performance. In this paper, we propose a novel periodic community model in temporal networks, σ-periodic k-clique, and an efficient algorithm for enumerating all σ-periodic k-cliques in real-world sparse temporal networks. We first design a new data structure to store temporal networks in main memory, which can reduce the maintaining cost and support dynamic deletion of nodes and edges. Then, we propose several efficient pruning rules to eliminate unpromising nodes and edges that do not belong to any σ-period k-clique to reduce graph size. Next, we propose an algorithm that directly enumerates σ-periodic k-cliques on temporal graph to avoid redundant computation. Finally, extensive and comprehensive experiments show that our algorithm runs one to three orders of magnitudes faster and requires significantly less memory than the baseline algorithms.

Keywords: Temporal networks · Periodic community · k-clique

1 Introduction

Real-world networks are usually temporal networks, in which their edges are associated with timestamps, indicating the time periods in which they exist. For example, in an email communication network, edge (u, v, t) means that user u send a message to user v in time t. Many real-world networks are temporal networks such as phone-call networks, social-media interaction networks and research collaboration networks. Many studies have been done to mine significant patterns in temporal networks such as finding fast-changing components [29], detecting information flow [12], mining dense subgraphs [20,27,30] and reachability testing [28]. With

B. Li et al. (Eds.): APWeb-WAIM 2022, LNCS 13421, pp. 461–476, 2023.
https://doi.org/10.1007/978-3-031-25158-0_38

the existence of timestamps, there are many new community models for temporal networks [1,3,18,19,23,31]. Periodicity is a very important phenomenon in time series analysis. Communities that occur periodically often indicate strong patterns in real-world, such as weekly group discussion, monthly friends parties and yearly research collaboration. Therefore, finding the periodically occurring communities can be helpful for finding dense connections between nodes that persist over time or predicting future events.

Mining dense subgraphs in static networks has been widely studied during the past few decades and they are widely used in real world applications such as finance [8], biology [10,22], search engine [11] and queries on graphs [6,15]. As shown below, periodic clique is an important community pattern in temporal networks. However, existing studies on finding periodic communities in temporal networks don't dive in to the problem of how to represent temporal networks in main memory nor take the sparse propriety of real-world temporal networks into account [23,31]. As far as we know, there is no study on one of the most basic periodic community, periodic k-cliques.

In this paper, we propose a new periodic dense subgraph model called σ-period k-clique, which is a k-clique that occurs σ times with some period. The main contributions are summarized as follows:

1. We design a new data structure, *Augmented Adjacency Array*, based on adjacency array, to represent temporal networks in memory. The new data structure has a low maintaining cost and supports dynamic deletion of vertices and edges, which is used in the pruning algorithms.
2. We propose a new method to enumerate periodic k-clique efficiently on real-world sparse temporal networks. We propose several efficient pruning rules to prune the temporal network efficiently, which is much more efficient than the baseline algorithm and performs nearly as well as the complex pruning algorithms [23] in real-world sparse networks. Then, we propose a new algorithm which directly enumerates periodic k-cliques on temporal networks. This algorithm avoids the costly computation of most periods and it prunes vertices and edges as enumeration to reduce unnecessary computation. It also avoids redundant computation of overlapping periods.
3. We conduct extensive experiments on five real-world networks to show the efficiency and effectiveness of our algorithm. The experiments show that our algorithm is one to three orders of magnitudes faster than the baseline algorithm.

2 Preliminaries

An undirected temporal graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of temporal edges. Each temporal edge $e \in \mathcal{E}$ is a triplet (u, v, t), where $u, v \in \mathcal{V}$, t is the time when this edges exists. Without loss of generality, we assume t is an integer, because timestamps in real world are usually represented as integer. The de-temporal graph $G = (V, E)$ is a graph that ignores all the timestamps, where $V = \mathcal{V}$ and $E = \{(u, v) | (u, v, t) \in \mathcal{E}\}$. A snapshot of $\mathcal{G}$ at time i is a static graph defined as $G_i = (V_i, E_i)$ where $V_i = \mathcal{V}$

and $E_i = \{(u,v)|(u,v,t) \in \mathcal{E}, t = i\}$. $N_v(G)$ is the neighbor of v. $D_v(G)$ is the degree of v. Let $N_u(\mathcal{G})$ or $N_u(G)$ be the neighbors of u.

A graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ is called a subgraph of $\mathcal{G}$ if $\mathcal{V}' \in \mathcal{V}$ and $\mathcal{E}' \in \mathcal{E}$. Similarly, graph $G' = (V', E')$ is called a subgraph of G if $V' \in V$ and $E' \in E$. Given a set of vertex $\mathcal{V}_s$, a subgraph $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s)$ is called a vertex induced subgraph if $\mathcal{S}_s \in \mathcal{V}$ and $\mathcal{E}_s = \{(u,v,t) \ |u,v \in \mathcal{V}_s, (u,v,t) \in \mathcal{E}\}$. We call $G_i = (V_i, E_i)$ a snapshot of temporal graph $\mathcal{G}$ where $V_i = \{u|(u,v,t) \in \mathcal{E}, t = i\}$ and $E_i = \{(u,v)|(u,v,t) \in \mathcal{E}, t = i\}$. Fig-1 shows an example of a temporal graph.

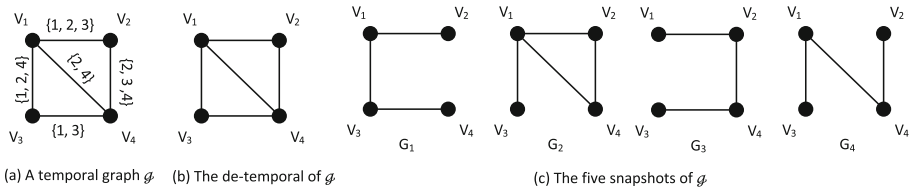

Fig. 1. An example of temporal graph

Definition 1 (time support set). *Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and its de-temporal graph G. The time support set of vertex $v \in V$ is $TS(v) = \{t|(v,u,t) \in \mathcal{E}, u \in \mathcal{V}\}$; the time support set of edge $(u,v) \in E$ is $TS(u,v) = \{t_i|(u,v,t) \in \mathcal{E}\}$, we will refer to the size of $TS(u,v)$, $|TS(u,v)|$ as an edge's weight for simplicity. And the time support set of a subgraph of G, $S = (V_s, E_s)$ is $TS(S) = \{t|\forall(u,v) \in E_s, t \in TS(u,v)\}$*

Definition 2 (periodic time support set). *Given a sorted time support set $P = \{ts_1, ts_2, \ldots, ts_k\}$, if the difference of any two adjacent timestamp $ts_i - (ts_i - 1)$ is a constant, then P is called a periodic time support. For convenience, we will call periodic time support as period and the difference between any two adjacent timestamps DIFF(P). If $|P| = \sigma$, P is called a σ-periodic time support set, also called σ-period.*

Definition 3 (σ-periodic k-clique). *Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a subgraph of $\mathcal{G}$, $\mathcal{C} = (\mathcal{V}_c, \mathcal{E}_c)$ and its de-temporal graph C is a σ-periodic k-clique with period P if (1) The de-temporal graph of $\mathcal{C}$ is a k-clique and (2) for any $u, v \in \mathcal{V}_c$, $(u,v,t) \in \mathcal{E}_c$ and (3) for any $(u,v,t) \in \mathcal{E}_c$, $t \in TS(C)$.*

Problem Statement. Given a temporal graph $\mathcal{G}$ and two parameters σ and k, our goal is to find all the σ-periodic k-cliques in $\mathcal{G}$. We refer this problem as the (σ, k)-PC problem.

We simply modify the algorithm of enumerating periodic maximal clique in [23] to find all the σ-periodic k-cliques. The algorithm is referred as PKC-B and it involves three parts: firstly, it prunes the graph using k-core and periodicity of vertices and edges; secondly, it translates the temporal graph into a static graph, the details of step 1 and 2 can be found in [23]; thirdly, it enumerates σ-periodic k-cliques on the translated static graph using the algorithm in [5].

Challenges. The challenges of searching the σ-periodic k-cliques by PKC-B are:

1. Existing studies don't dive into the memory representation of temporal graphs and use trivial methods to represent temporal graphs such as a combination of map and lists [23]. Maintaining such a data structure is costly. It is challenging to design a new data structure which supports fast access to vertices, edges and temporal edges and dynamic deletion of vertices and edges to satisfy the needs of the enumeration algorithm.

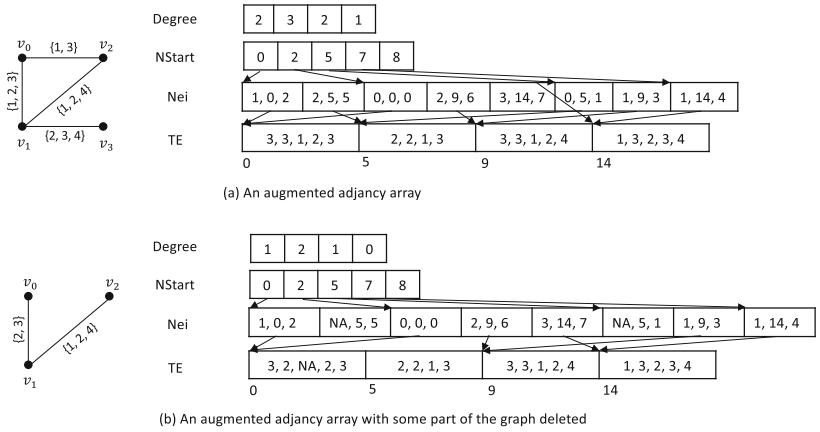

Fig. 2. The data structure of augmented adjacency array

2. In the pruning algorithm of the baseline algorithm, the periods of every vertices and edges in the graph are computed, which is costly. The real-world networks are always sparse. It is challenging to design a more efficient algorithm with the pruning power close to the complex pruning rules in the baseline algorithm [23].

3. The baseline algorithm first converts the pruned temporal graph to a static graph, then it applies clique enumeration algorithm to the static graph. The periods of all the vertices and edges is needed to do this conversion, which is costly to compute. Besides, there are overlapping between periods, which will lead to redundant computation. The problem is can we enumerate periodic k-cliques directly on the temporal graph and also avoid redundant computation?

3 The Proposed Algorithms

In this section, we present our algorithm in three parts. Firstly, we present a new data structure, called augmented adjacency array, an in-memory data structure for temporal networks which supports dynamic deletion of nodes and edges. Then, we propose an efficient pruning algorithm to prune unpromising vertices and edges that cannot be in any σ-periodic k-clique. Finally, we devise a σ-periodic k-clique enumeration algorithm which directly enumerates periodic cliques on temporal graphs.

3.1 Augmented Adjacency Array

To implement the pruning and enumerate algorithm efficiently, a data structure which can provide fast access of vertices and edges and support deletion of vertices and edges is needed. So we design an efficient in-memory data structure, called Augmented Adjacency Array, to store temporal networks in main memory that fits the access pattern of our algorithms. All the following algorithms in this section are based on this data structure since it provides fast access and deletion of the network. The data structure is efficient in four aspects (1) it has a low maintaining cost compared with previous map-based implementation [23], (2) when the temporal graph is scanned sequentially, the data structure preserves cache locality, (3) the deletion of edges and temporal edges can be done in constant time with a few operations and (4) our data structure is space efficient.

The augmented adjacency is composed of four arrays: *Degree*, *NStart*, *Nei*, *TE*. For each vertex v, *Degree*[v] stores the degree of vertex v. *NStart*[v] stores the start position of its neighbors in edges, so *NStart*[$v+1$] stores the position that just after the end of its neighbors in *Nei*. The elements between *NStart*[v] and *NStart*[$v+1$] in *Nei* are the neighbors of vertex v. Each element in *Nei* is composed by three parts: *Id*, *TsIdx* and *REIdx*. *Id* is the vertex id of the neighbor, *TsIdx* points to the start position of the current edge's timestamps in *TE*, and *REIdx* (Reverse Edge Index) points to location of the reverse direction edge in *Nei*, specially edge (v, u) in *Nei*. The *REIdx* is used to delete an edge in O(1) time. The timestamps of each non-temporal edge are stored in *TE* (Temporal Edge), each edge's timestamps are arranged in three parts, *L*, *TSize* and *TS*. *L* is the length of each edge's *TS* part in memory, counted *INVALID* timestamps, *TSize* is the size of time supports for the edge. *TS* is the sorted list of time supports of that edge.

Operations. This data structure supports dynamic deletion for the pruning procedure. We use special value, *INVALID*, to mark if an edge or a temporal edge is deleted. To delete a vertex, we should traverse its neighbor and delete all its edges, then set it's degree to 0. To delete an edge (u, v), mark the Id of $v's$ block in $u's$ neighbor and the Id of $u's$ block in $v's$ neighbor to be *INVALID*. To delete a temporal edge, we mark the timestamp in *TS* to be *INVALID* and decrement its *TSize*.

Figure 2 is an example of augmented adjacency array. (a) shows the in-memory representation of the left graph. (b) shows when vertex v_3, edge (v_0, v_2) and temporal edge $(v_0, v_1, 1)$ is deleted from the origin graph.

Complexity. The time complexity of constructing the data structure from unordered edges is $O(|\mathcal{E}| + |\mathcal{E}| ln \frac{|\mathcal{E}|}{|E|})$ and the time complexity of deleting edges or temporal edges is $O(1)$, the time complexity of deleting a vertex is $O(D_v(G))$. The space complexity of the data structure is $O(2|V| + 8|E| + |\mathcal{E}|)$. The can be easily computed, thus it is omitted due to space limitation.

3.2 The Proposed Pruning Techniques

Although [23] uses reverse index to avoid re-computation of periods caused by the deletion of a vertex, it is still inefficient because it needs to compute the periods of all vertices and edges and the cost of computing periods can not be omitted. We notice that real-world temporal networks are always sparse, so we proposed several pruning rules that can be efficiently implemented and perform nearly as well as the complex pruning rules.

We first devise several lemma and the pruning rules induced from these lemma. Then, we apply these pruning rules to our pruning algorithm.

Lemma 1. *Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, its de-temporal graph $G = (V, E)$ and edge $e_{u,v} = (u, v) \in E$, if $e_{u,v}$ belongs to some σ-periodic k-clique then the time support set of edge $e_{u,v}$, $TS(u, v)$ must contains a σ-period.*

With Lemma 1, given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and its de-temporal graph $G = (V, E)$. We can prune any edge $e_{u,v} = (u, v)$, we can have the flowing pruning rule:

Pruning Rule 1. If $|TS(u, v)| < \sigma$, we can delete edge e without reducing the number of σ-period k-cliques in graph $\mathcal{G}$.

Pruning Rule 2. If (1) $|TS(u, v)| = \sigma$, and (2) $TS(u, v)$ is not a σ-period, we can delete edge $e_{u,v}$ without reducing the number of σ-period k-cliques in graph $\mathcal{G}$. Note we only check the edge (u, v) where $|TS(u, v)| = \sigma$ for efficiency.

Lemma 2. *Given a temporal graph $\mathcal{G}$ and its de-temporal graph G. For any σ-periodic k-clique $\mathcal{C}$ in $\mathcal{G}$, it's de-temporal S must be a subgraph of the k-core of G.*

Pruning Rule 3. Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and its de-temporal graph $G = (V, E)$. We can prune any vertex v and it associated edges where v doesn't belong to the k-core of G without reducing the number of σ-period k-cliques in graph $\mathcal{G}$.

Lemma 3. *Given a temporal graph $\mathcal{G}$, for any temporal edge $e = (u, v, t)$, if edge e belongs to any σ-periodic k-clique, then in snapshot G_t, we have $D_u(G_t) \geq k$ and $D_v(G_t) \geq k$.*

Pruning Rule 4. Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we can prune all the temporal edges (u, v, t) where $D_u(G_t) < k$ or $D_v(G_t) < k$ without reducing the number of σ-period k-cliques in $\mathcal{G}$.

The Proof of Lemma 1, 2 and 3 is straightforward, so they are omitted due to space limitation.

Algorithm 1 prunes the input graph with a combination of the four rules with parameter $k = 3$ and $\sigma = 3$. It first computes the degrees of each vertex with pruning rule 1 and 2 (line 5–6). And it then counts the degree with invalid edge deleted (line 7). After that, it computes the k-core on the graph pruned by rule

1 and 2 (lines 11–15). Then it applies pruning rule 4 (lines 16–25). Finally it computes the degree of each vertex at each snapshot (line 19) and the edges do not following pruning rule 4 are deleted (line 23).

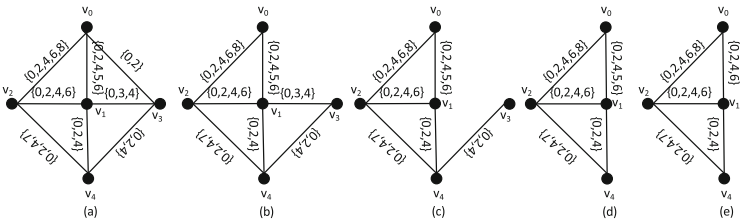

Fig. 3. Illustration of the Pruning algorithms

Figure 3 shows an example of our pruning rules, (a) is the origin graph. (b) is the graph pruned by rule 1, edge $(0, 3)$ is deleted. (c) shows the graph pruned by rule 2 based on (b), edge $(1, 3)$ is deleted. (d) shows the graph pruned by rule 3 based on (c), vertex v_3 and the edges associated with it are deleted. (e) shows the graph pruned by rule 4, temporal edges $(0, 1, 5)$, $(0, 2, 8)$ and $(2, 4, 7)$ are deleted.

Algorithm 1: AggregatedPrune

Data: temporal graph $\mathcal{G}$, its de-temporal graph G, parameter σ, k
Result: pruned temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, an array of the degree left *Degree*

```
1  Q = ∅;
2  Initialize Degree to an array of 0;
3  foreach u ∈ G do
4      foreach v ∈ u.neighbor do
5          if (u, v).ts.size() < σ then Delete edge (u, v) ;
6          else if |TS(u, v)| = σ AND |TS(u, v)| is not a σ-period then Delete edge (u, v) ;
7          else Degree[u] ← Degree[u]+1 ; // This edge is valid
8      end
9      if Degree[u] < k-1 then Q ← Q ∪ {u} ;
10 end
11 while Q ≠ ∅ do
12     v = Q.pop()
13     if Degree[v] == 0 then continue ;
14     Delete vertex v;
15 end
16 foreach u ∈ V do
17     SD ← ∅;
18     foreach v ∈ u.neighbor do
19         foreach t ∈ TS(u, v) do SD[t].append([u, v, t]) ;
20     end
21     foreach t ∈ SD.keys do
22         if |SD[t]| < k-1 then
23             foreach (u, v, t) ∈ SD[t] do Delete temporal edge (u, v, t) ;
24     end
25 end
```

Complexity. The worst case of our pruning algorithm occurs when the time support set of any two edges is a σ-period, then our pruning algorithm will iterate over all the timestamps. So the worst case time complexity is $O(|\mathcal{E}|)$.

3.3 Enumerating Periodic *k*-Cliques

It is very time-consuming to computing the periods of all the vertices and edges and constructing a new static graph using them. Besides, this procedure is also space inefficient since it needs to save the periods of all vertices and edges. The new graph is also need to be saved in memory. To address these problems, we propose an algorithm that enumerates periodic cliques directly on the temporal graph to reduce the computation of periods and induced memory cost.

Our enumeration algorithm improves the performance in several ways. Firstly, it directly enumerates periodic k-cliques on the temporal graph, reducing memory cost. Secondly, we adopt a new way to process the periods to avoid

Algorithm 2: ComputeMaximalPeriod(TS, σ)

Data: An array of timestamps TS, parameter σ
Output: An array of σ-maximal periods in TS

$P \leftarrow \emptyset$, $MP \leftarrow \emptyset$, $S \leftarrow \emptyset$;
foreach $t \in TS$ **do**
  $ED \leftarrow \emptyset$;
  foreach $p \in P$ **do**
    if $t - p.s = p.l * p.d$ **then**
      $p.l \leftarrow p.l + 1$, $ED \leftarrow ED \cup \{p.d\}$;
    else if $(t - p.s) > p.l * p.d$ **then**
      if $p.l \geq \sigma$ **then**
        $MP \leftarrow MP \cup \{[\mathrm{s} \leftarrow p.s, \mathrm{d} \leftarrow p.d, \mathrm{l} \leftarrow p.l]\}$;
      $P \leftarrow P \backslash \{p\}$;
  end
  foreach $s \in S$ **do**
    if $(t - s)(\sigma - 2) \leq max(TS) - t \ AND \ (t - s) \notin ED$ **then**
      $P \leftarrow P \cup \{[\mathrm{s} \leftarrow s, \mathrm{d} \leftarrow t - s, \mathrm{l} \leftarrow 1]\}$;
  end
  $S \leftarrow S \cup \{t\}$;
end
foreach $p \in P$ **do**
  if $p.l \geq \sigma$ **then**
    $MP \leftarrow P \cup \{[\mathrm{s} \leftarrow p.s, \mathrm{d} \leftarrow p.d, \mathrm{l} \leftarrow p.l]\}$;
end
return MP;

Procedure UnionPeriod(MP, TS)
$CMP \leftarrow \emptyset$;
foreach $p \in MP$ **do**
  $s \leftarrow p.s$, $l \leftarrow 0$, $next \leftarrow p.s$;
  foreach $i \in$ *[0, p.l-1]* **do**
    if $next \notin TS$ **then**
      if $l \geq \sigma$ **then** $CMP \leftarrow CMP \cup \{[\mathrm{s} \leftarrow s, \mathrm{d} \leftarrow p.d, \mathrm{l} \leftarrow l]\}$;
      $s \leftarrow next + p.d$, $l \leftarrow 0$;
    else $l \leftarrow l + 1$;
    $next \leftarrow next + p.d$;
  end
  if $l \geq \sigma$ **then** $CMP \leftarrow CMP \cup \{[\mathrm{s} \leftarrow s, \mathrm{d} \leftarrow p.d, \mathrm{l} \leftarrow l]\}$;
end
return CMP;

most computation of periods. Thirdly, we introduce a "prune while enumeration" strategy. During enumeration of periodic cliques, the processed vertices and edges are deleted to avoid re-enumeration. We push this deletion on step forward. The deletion of vertices and edges might cause the left graph violates the pruning rules presented in the previous subsection. So, we also delete the vertices that violate the pruning rules due to a deletion of a processed vertex or edge. At last, the overlaps between periods can cause redundant computation. We use "maximal period" in enumeration to prevent the redundant computation caused by overlapping periods.

Definition 4 (σ-maximal period). *Given a time support set TS, a σ-maximal period of TS, $\sigma - MP$, is a period of TS where (1) $|\sigma - MP| \geq \sigma$ and (2) there exist no period P such that $DIFF(P) = DIFF(\sigma - MP)$ and $P \subset \sigma - MP$.*

Definition 5 (σ-maximal support period). *Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a subgraph $\mathcal{S} = (\mathcal{V}_c, \mathcal{E}_c)$. The (σ, k)-maximal support period of $\mathcal{S}$, denoted as $MSP_{\sigma,k}(\mathcal{S})$, is a set of σ-maximal period of S where (1) for any $P \in MPS_{\sigma,k}(\mathcal{S})$, P is a σ-maximal period and (2) for any σ-maximal period, P, of subgraph $\mathcal{S}$, $P \in MPS_{\sigma,k}(\mathcal{S})$.*

Algorithm 2 computes the maximal periods of the given sorted timestamps. A period is represented by three parts, s, d, l. s is the smallest timestamp in the

Algorithm 3: KPC-A

```
   Data: temporal graph 𝒢, parameter k and σ
   Result: a vector of σ-periodic k-cliques
 1 Prune the graph using the algorithm in 3.2 ;
 2 cliques ← ∅;
 3 foreach u ∈ V do
 4 |   if Degree[u] < k then Delete vertex u, continue ;
 5 |   foreach v ∈ u.neighbors do
 6 |   |   PS = ComputeMaximalPeriod(TS(u, v));
 7 |   |   if |PS| ≠ 0 then
 8 |   |   |   PC ← {u, v};
 9 |   |   |   CV ← u.neighbor ∩ v.neighbor;
10 |   |   |   if |CV| ≥ k − 2 then kcliqueRec(PS, PC, CV) ;
11 |   |   |   Delete edge (u, v);
12 |   |   |   if Degree[u] < k then Delete vertex u; ;
13 |   |   |   if Degree[v] < k then Delete vertex v; ;
14 |   end
15 |   Delete vertex v;
16 end

17 Procedure KcliqueRec(PS, PC, CV)
18 if |PC| = k then cliques ← cliques ∪ PC, return ;
19 if |PC| + |CV| < k then return ;
20 for u ∈ CV do
21 |   CV ← CV\{u};
22 |   nCV ← CV ∩ u.neighbor;
23 |   nPS ← PS;
24 |   foreach v ∈ PC do
25 |   |   nPS ← UnionPeriod(nPS, TS(u, v));
26 |   |   if |nPS| = 0 then continue ;
27 |   end
28 |   KcliqueRec(nPS, CV ∪ u, nCV);
29 end
```

period, d is the difference between two adjacent timestamps and l is the size of the period. The algorithm first initializes some data structures (line 1): P is the set of periods that might be extended, MP is maximal periods that can not be extended and S is the set of timestamps that has been traveled, which used to add new periods. It then travels through the sorted timestamps (line 2). ED is used to avoid adding a non-maximal period (line 3). Then, for each candidate period that might be extended, extend it if the new timestamp t can be used to extend the candidate period (line 5–6). If the new timestamp t cannot be used to extend the candidate period and it is larger than the next expected timestamp for it, which means that the current period cannot be extended, so it is added to the result (line 8, 9) and the candidate is erased (line 10). After that, the algorithm will add new maximal period candidates where the current processing timestamp is the second timestamp in the period (line 12 to 14). Note the new candidates' difference cannot be in ED or it can't be the start of a maximal period. After all the timestamps are processed, the left candidates are checked, periods with size not smaller than σ are added to the final result (line 18–21).

Given a σ-maximal support period MP and a time support set TS, UnionPeriod will find the σ-maximal support period of $TS' = \{t|t \in p, p \in MP\} \cup \{t|t \in TS\}$. It travels through each maximal period (line 25). For each maximal period P, it checks the overlapping part of TS and P (line 27–32).

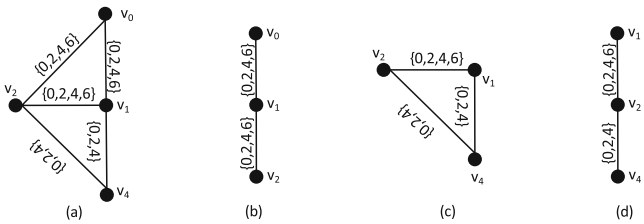

Fig. 4. Example of the Enumeration algorithm

We propose a σ-periodic k-clique enumeration algorithm based on Chiba and Nishizeki's algorithm for listing k-cliques [5]. For each vertex in u, it forms a 1-clique, called a partial clique, then the algorithm finds the vertex that can be added to the partial clique and compute the maximal support recursively. The support period set for the new partial clique can be computed incrementally with the algorithms in algorithm 4 efficiently.

The details of periodic clique enumeration is shown in Algorithm 3. The cliques are enumerated in a recursive manner. For each vertex at the root of the recursive tree, select a vertex in its neighbor (line 5). If the new vertex can be added to the partial clique (line 7), compute the new common neighbor of the partial clique (line 9) and try to add new vertex in the common neighbor to the partial clique recursively. After all the cliques start from a root vertex u and its neighbor v are checked, edge (u, v) is deleted to avoid replicate enumeration (line 11) and vertex u, v is checked if it can be deleted recursively. After all

the cliques start with v have been enumerated, vertex v is deleted. The recursive algorithm will first check if it has found a σ-periodic k-clique, if it has found one, return (line 18). And then it selects a vertex u from u's neighbors, compute the new common neighbor (line 22), the new maximal support period is computed when u is added (line 24–27) and then a new vertex is added from the common neighbor of the new partial clique recursively.

Figure 4 shows the enumeration of 3-periodic 3-clique on the pruned graph in Fig. 2. (a) is the pruned graph, (b) is the recursive tree with root v_0, after root v_0 and edge (v_0, v_1) is processed, edge (v_0, v_1) is deleted, causing v_0 deleted. The graph is shown in (c). Then we select v_1 as root, the recursive tree is shown in (d). After that, the enumeration is finished.

Complexity. The worst case complexity of ComputeMaximalPeriod is $O(l_t^3)$ for a time support of length l_t. And the complexity of UnionPeriod is $O(l_p)$ where l_p is the total length of periods in parameter MP. We assume that the degrees and temporal edges are distributed evenly in the graph, the time complexity of the enumeration procedure is $O(2|V|t_a C^{\sigma}_{d_a})$, where d_a is the average degree $|V|/|E|$ and t_a is the average time support $|\mathcal{E}/E|$.

4 Experiments

Algorithms. We implement four algorithms in our experiments. KPC-B is the baseline algorithm shown in Sect. 4.1. KPC-O1 is the implementation of our basic enumeration algorithm without any optimizations. KPC-O2 is the implementation of our enumeration algorithm which uses the pruning algorithm to prune the graph. KPC-A is the implementation of our enumeration algorithm with all the optimizations. All the algorithms are implemented with C++ and compiled

Table 1. Datasets

Dataset	$\|V\|$	$\|E\|$	$\|\mathcal{E}\|$	d_{max}	$d_{temp_{max}}$	Time scale
PS	242	8,317	32,079	134	503	Hour
LKML	26,885	159,996	328,092	2,989	14,172	Month
Enron	86,978	297,456	499,983	1,726	4,311	Month
DBLP	1,729,816	8,546,306	12,007,380	3,815	5,980	Year
WikiTalk	2,863,439	8,146,544	10,268,684	146,311	277,833	Month

Table 2. Running time(s) on different datasets

	σ, k = 3,3			σ, k = 4,4			σ, k = 5,5		
	KPC-B	KPC-A	Speedup	KPC-B	KPC-A	Speedup	KPC-B	KPC-A	Speedup
PS	0.36	0.04	9.41	0.32	0.02	15.43	0.24	0.01	19.97
LKML	34.32	1.02	31.91	18.27	0.07	27.48	10.31	0.34	30.08
Enron	16.120	0.360	44.21	6.54	0.20	33.43	3.01	0.09	32.20
DBLP	1071.14	2.82	379.53	151.77	0.77	197.90	53.17	0.31	172.00
WikiTalk	9609.68	12.46	771.32	5047.87	4.41	1145.20	2752.41	1.88	1465.06

using GCC version 11.1.0 with optimization level set to O3. All the experiments are carried out on a PC with Linux kernel 5.12.14 running on an Intel i5-9400 with 32 GB memory. The source code can be found in https://www.dropbox.com/s/1xcf8fh7srt0n2b/periodic_k_clique.zip?dl=0.

Datasets. We use 5 real-life temporal networks in our experiments, which are PS, LKML, ENRON, DBLP, WikiTalk. PS is a graph of face-to-face contacts between students and teachers in a French primary school. An temporal edge (u, v, t) in PS means there is face-to-face communication between u and v in time period t. Both LKML and Enron are email communication networks. A temporal edge (u, v, t) in LKML and Enron means u sent an email to v in time period t. Their timescale is rescaled to month. DBLP is a scientific collaboration graph generated from the DBLP dataset. An temporal edge (u, v, t) in DBLP means u and v published a paper together in year t. WikiTalk is a communication network between English wiki users. It's time is also rescaled to month. The time scale represents the time length of each timestamp. Their sizes are shown in Table 1.

Exp-1: Running Time of Different Datasets. Table 2 shows the running time of different datasets with three parameter settings, (σ, k) = (3, 3), (σ, k) = (4, 4) and (σ, k) =(5, 5). We can see that our algorithm outperforms the baseline algorithm in all five datasets and three parameter settings. The speedup grows as the graph size grows. Our algorithm can finish enumeration in DBLP and WikiTalk in a few seconds using a single thread. And it shows a maximum 2822 times speed up in DBLP where σ=3 and k = 3. For each dataset, the running time and speedup grow as parameter σ and k grow, mainly because the effectiveness of pruning algorithm grows as σ and k grows, we can see that in later experiments.

Exp-2: Running Time on Varies Parameters. Figure 5 shows the running time of DBLP with different parameters where both σ and k varies from 3 to

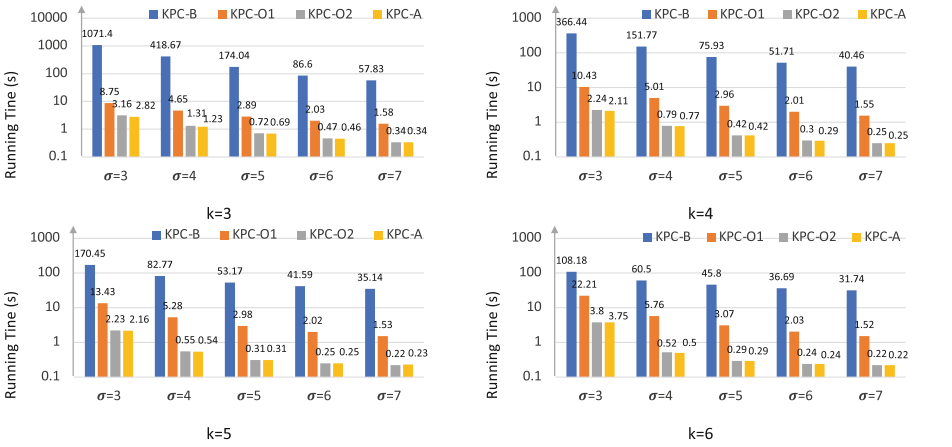

Fig. 5. Running time(s) on different parameters (on DBLP)

6. From the figure we can see that our algorithm outperforms the three baseline algorithms in all the parameter settings. The running time decreases with σ increases consistently, due to the effectiveness of our pruning algorithm. The running time of KPC-O1 and KPC-O2 are close since the enumeration procedure will also prune the graph as it enumerates. With the addition of elimination of overlapping periods, there is a significant improvement in KPC-A.

Exp-3: Pruning Power on Different Datasets. Table 3 shows the pruning power of our pruning algorithm with different datasets when $\sigma = 5$ and $k = 5$. It doesn't performs well in PS since PS is a small dense face-to-face communication graph and there tend to be no obvious large central vertex. The pruning algorithm is very effective in the four real-world large sparse networks and the pruning power grows with the size of graph grows. It can prune out over 97% vertices in four graphs and it even prune 99.79% vertices in DBLP. Despite the simplicity of our pruning algorithm, it performs pretty well in sparse networks.

Table 3. Pruning power with on different datasets ($\sigma = 5$ and k = 5)

	Vertices			Edges			Temporal edges		
	Origin	Left	Percent	Origin	Left	Percent	Origin	Left	Percent
PS	242	232	95.86%	8,317	1,822	21.91%	32,079	15988	49.8%
LKML	26,885	649	2.41%	159,996	7745	4.84%	328,092	77185	23.5%
Enron	86,978	1,003	1.51%	297,456	6601	2.21%	499,983	39639	7.9%
DBLP	1,729,816	3,678	0.21%	8,546,306	10,475	0.12	12,007,380	50438	0.4%
WikiTalk	2,863,439	7,015	0.24%	8,146,544	63,303	0.77%	10,268,684	394728	3.84%

Exp-4: Pruning Power on Different Parameters. Figure 6 shows the pruning power of our pruning algorithm on DBLP with different parameters where both σ and k varies from 3 to 7. We can see that our algorithm is very efficient even $\sigma = 3$ and $k = 3$. And the size of the pruned graph decreases dramatically as σ or k increases. With each increment σ or k, the size of left vertices, edges and temporal edges are nearly halved. And the increment of pruning power are directly reflected of total execution time presented in Experiment 1 and Experiment 2.

Exp-5: Memory Overhead. Figure 7 shows the memory usage of KPC-B and KPC-A. We can see that our algorithm costs significantly less memory than KPC-B. Our algorithm achieves high space-efficiency by using the augmented adjacency array and avoiding store the periods of all edges and vertices in memory.

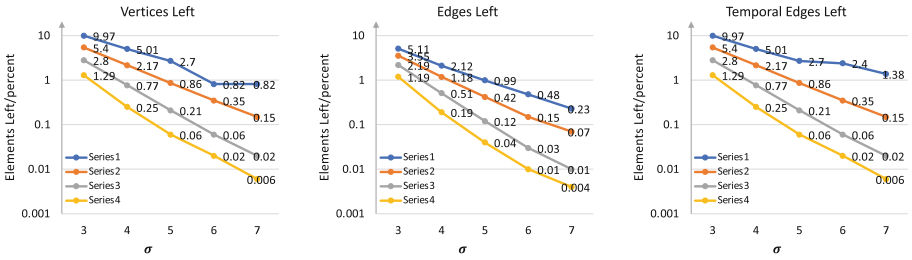

Fig. 6. Pruning power on different parameters on DBLP

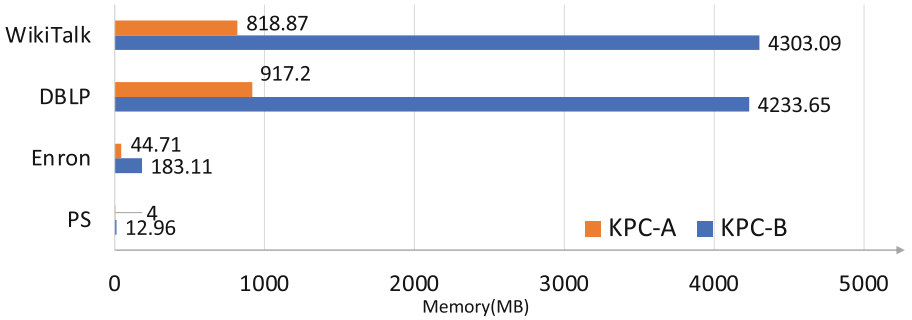

Fig. 7. Memory usage

5 Related Work

***k*-Clique Listing and Counting.** Enumerating k-clique in static graph has been studied for decades [26]. The first practical algorithm of enumerating k-cliques is the algorithm of Chiba and Nishizeki [5]. Danisch et al. [7] give an edge-parallel algorithm to enumerate k-clique and use DAG to avoid re-enumeration. [13] propose an approximately k-clique counting algorithm based on Turan Shadow. Li et al. [17] propose a k-clique listing algorithm based on graph coloring. Jain and Seshadhri [14] propose a k-clique counting algorithm based on the pivot technology and it is much faster than enumeration-based algorithms.

Maximal Clique Enumeration. Many techniques in enumerating maximal cliques can also be used in listing k-cliques with minor change [25]. The best known algorithm in listing maximal cliques is the Bron-Kerbosch algorithm [2]. There are many variants of the Bron-Kerbosch algorithm [4,9,21,25].

Temporal Graph Data Mining. Most previous studies on temporal graph data mining focus on finding dense subgraphs that persist over time or analyzing the evolve of communities. Li et al. [16] propose an algorithm to detect dense subgraphs which persist over a threshold. Ma et al. [20] give an algorithm to find dense subgraphs where edge weights vary with timestamps. Lin et al. [18] propose

an algorithm to analyze the evolution of communities through a unified process. Yang et al. [29] propose a method to evaluate the change speed of subgraphs and give an algorithm to detect the fast-changing subgraphs. Rossetti et al. [24] propose a method to extract and track the evolution of overlapping communities using an online iterative procedure. Qin et al. [23] address the problem of finding periodic maximal cliques in temporal cliques. And Zhang et al. [31] propose a new periodic subgraph model called seasonal-periodic subgraph.

6 Conclusion

In this paper, we study the problem of how to find dense subgraphs that periodically occurs in temporal networks, and then propose σ-periodic k-clique to model it. Firstly, we design a new data structure to store temporal networks in main memory as well as support dynamic deletion for the pruning algorithms. Secondly, we propose several effective and efficient rules to prune the graph to reduce search space. Thirdly, we design a search algorithm to enumerate all the σ-periodic k-clique based on the Bron-Kerbosch algorithm for static graphs. Finally, we conduct extensive experiments on five real-world temporal networks to show that our algorithm works efficiently in practice.

Acknowledgments. This work was partially supported by (i) National Key Research and Development Program of China 2020AAA0108503, (ii) NSFC Grants 62072034, 62002036, (iii)Natural Science Foundation of Chongqing CSTC cstc2021jcyj-msxm X0859.

References

1. Agarwal, M.K., Ramamritham, K., Bhide, M.: Real time discovery of dense clusters in highly dynamic graphs: identifying real world events in highly dynamic environments. arXiv preprint arXiv:1207.0138 (2012)
2. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Commun. ACM **16**(9), 575–577 (1973)
3. Chen, Z., Wilson, K.A., Jin, Y., Hendrix, W., Samatova, N.F.: Detecting and tracking community dynamics in evolutionary networks. In: ICDMW, pp. 318–327 (2010)
4. Cheng, J., Zhu, L., Ke, Y., Chu, S.: Fast algorithms for maximal clique enumeration with limited memory. In: SIGKDD, pp. 1240–1248 (2012)
5. Chiba, N., Nishizeki, T.: Arboricity and subgraph listing algorithms. SIAM J. Comput. **14**(1), 210–223 (1985)
6. Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and distance queries via 2-hop labels. SIAM J. Comput. **32**(5), 1338–1355 (2003)
7. Danisch, M., Balalau, O., Sozio, M.: Listing k-cliques in sparse real-world graphs. In: WWW, pp. 589–598 (2018)
8. Du, X., Jin, R., Ding, L., Lee, V.E., Thornton Jr., J.H.: Migration motif: a spatial-temporal pattern mining approach for financial markets. In: SIGKDD, pp. 1135–1144 (2009)

9. Eppstein, D., Strash, D.: Listing all maximal cliques in large sparse real-world graphs. In: International Symposium on Experimental Algorithms, pp. 364–375 (2011)
10. Fratkin, E., Naughton, B.T., Brutlag, D.L., Batzoglou, S.: Motifcut: regulatory motifs finding with maximum density subgraphs. Bioinformatics **22**(14), e150–e157 (2006)
11. Gibson, D., Kumar, R., Tomkins, A.: Discovering large dense subgraphs in massive graphs. Proc. VLDB Endow. 721–732 (2005)
12. Gurukar, S., Ranu, S., Ravindran, B.: Commit: a scalable approach to mining communication motifs from dynamic networks. In: SIGMOD, pp. 475–489 (2015)
13. Jain, S., Seshadhri, C.: A fast and provable method for estimating clique counts using turán's theorem. In: WWW, pp. 441–449 (2017)
14. Jain, S., Seshadhri, C.: The power of pivoting for exact clique counting. In: WSDM, pp. 268–276 (2020)
15. Jin, R., Xiang, Y., Ruan, N., Fuhry, D.: 3-hop: a high-compression indexing scheme for reachability query. In: SIGMOD, pp. 813–826 (2009)
16. Li, R.H., Su, J., Qin, L., Yu, J.X., Dai, Q.: Persistent community search in temporal networks. In: ICDE, pp. 797–808 (2018)
17. Li, R., Gao, S., Qin, L., Wang, G., Yang, W., Yu, J.X.: Ordering heuristics for k-clique listing. Proc. VLDB Endow. (2020)
18. Lin, Y.R., Chi, Y., Zhu, S., Sundaram, H., Tseng, B.L.: Facetnet: a framework for analyzing communities and their evolutions in dynamic networks. In: WWW, pp. 685–694 (2008)
19. Liu, P., Wang, M., Cui, J., Li, H.: Top-k competitive location selection over moving objects. Data Sci. Eng. **6**(4), 392–401 (2021)
20. Ma, S., Hu, R., Wang, L., Lin, X., Huai, J.: Fast computation of dense temporal subgraphs. In: ICDE, pp. 361–372 (2017)
21. Makino, K., Uno, T.: New algorithms for enumerating all maximal cliques. In: Scandinavian Workshop on Algorithm Theory, pp. 260–272 (2004)
22. Presson, A.P., et al.: Integrated weighted gene co-expression network analysis with an application to chronic fatigue syndrome. BMC Syst. Biol. **2**(1), 1–21 (2008)
23. Qin, H., Li, R., Yuan, Y., Wang, G., Yang, W., Qin, L.: Periodic communities mining in temporal networks: concepts and algorithms. IEEE TKDE (2020)
24. Rossetti, G., Pappalardo, L., Pedreschi, D., Giannotti, F.: Tiles: an online algorithm for community discovery in dynamic social networks. Mach. Learn. **106**(8), 1213–1241 (2017)
25. Takeaki, U.: Implementation issues of clique enumeration algorithm. Special issue: Theor. Comput. Sci. Discrete Math. Progress Inform. **9**, 25–30 (2012)
26. Tsourakakis, C.: The k-clique densest subgraph problem. In: WWW, pp. 1122–1132 (2015)
27. Wu, H., et al.: Core decomposition in large temporal graphs. In: IEEE Conference on Big Data, pp. 649–658 (2015)
28. Wu, H., Huang, Y., Cheng, J., Li, J., Ke, Y.: Reachability and time-based path queries in temporal graphs. In: ICDE, pp. 145–156 (2016)
29. Yang, Y., Yu, J.X., Gao, H., Pei, J., Li, J.: Mining most frequently changing component in evolving graphs. WWW, vol. 17, no. 3, pp. 351–376 (2014)
30. Yang, Y., Yan, D., Wu, H., Cheng, J., Zhou, S., Lui, J.C.: Diversified temporal subgraph pattern mining. In: SIGKDD, pp. 1965–1974 (2016)
31. Zhang, Q., Guo, D., Zhao, X., Li, X., Wang, X.: Seasonal-periodic subgraph mining in temporal networks. In: CIKM, pp. 2309–2312 (2020)