Periodic Community Search in Temporal Graphs: Time Series-based Methods

YU CHEN, Zhejiang University, China
QING LIU, Zhejiang University, China
CHENGYANG LUO, Zhejiang University, China
YUNJUN GAO, Zhejiang University, China

In this paper, we study periodic community search in temporal graphs. First, motivated by the limitations of existing arithmetic sequence-based periodicity determination method, we propose a novel *k-core time series-based approach* for periodicity determination, which leverages time series similarity to discover periodic patterns. Based on it, we formally define the local periodic community (LPC) search problem, which aims to identify a periodic community with the highest occurrence frequency of its periodic pattern. A naive solution to LPC search is to enumerate all vertex subsets for examination, which is computationally infeasible. To this end, we develop efficient *time-deletion-based* (TD) algorithm and *vertex-deletion-based* (VD) algorithm. Specifically, the TD algorithm leverages novel temporal monotonicity and incremental generation strategies to produce high quality candidate vertex subsets to reduce the search space and devises TP-Core index to facilitate the candidate generation. The VD algorithm avoids the vertex subset enumeration by iteratively deleting vertices that cannot contribute to the final results, thereby enhancing efficiency. Extensive experiments validate the effectiveness, efficiency, and scalability of our proposed algorithms.

CCS Concepts: • **Theory of computation → Dynamic graph algorithms**.

Additional Key Words and Phrases: Periodic Community Search; Time Series-based Periodicity Determination; Temporal Graphs

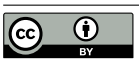
ACM Reference Format:
Yu Chen, Qing Liu, Chengyang Luo, and Yunjun Gao. 2026. Periodic Community Search in Temporal Graphs: Time Series-based Methods. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 222 (June 2026), 26 pages. https://doi.org/10.1145/3802099

1 Introduction

Community search, as a fundamental task in graph data mining, aims to identify highly cohesive and personalized communities containing the given query vertices [9, 23, 25, 44, 45]. In real-world scenarios, graph structures evolve dynamically over time. Notably, periodicity is a common pattern in such dynamic networks. Representative examples include animal mobility [22], neuron activation in the brain [3] and gene expression during the cell cycle [38], as well as recurring behaviors in social [11, 32] and infrastructure networks [15]. Modeling and analyzing such periodic communities in temporal graphs not only provides in-depth insights into the formation mechanisms of complex patterns but also establishes theoretical foundations for critical downstream applications.

Research on query-driven periodic community search in temporal graphs has been notably overlooked in comparison to the extensive focus on *periodic subgraph mining*. The seminal work

Authors' Contact Information: Yu Chen, Zhejiang University, Hangzhou, Zhejiang, China, cy1098@zju.edu.cn; Qing Liu, Zhejiang University, Hangzhou, Zhejiang, China, qingliucs@zju.edu.cn; Chengyang Luo, Zhejiang University, Hangzhou, Zhejiang, China, luocy1017@zju.edu.cn; Yunjun Gao, Zhejiang University, Hangzhou, Zhejiang, China, gaoyj@zju.edu.cn.



ACM 2836-6573/2026/6-ART222
https://doi.org/10.1145/3802099

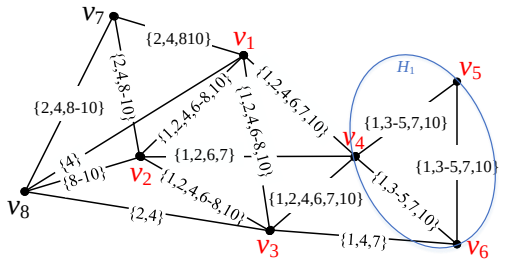


Fig. 1. An Illustrative Example of Temporal Graph $\mathcal{G}$

by Lahiri et al. [16, 17] formalized periodicity using *arithmetic sequences*, though their exhaustive enumeration approach suffered from computational inefficiency. Subsequent work improved efficiency by concentrating on periodic k-core [35] and k-clique [32] mining, as well as densest periodic subgraph mining [33]. Recently, Zeng et al. [51] introduced the problem of quasi-periodic subgraph mining to capture quasi-periodic events.

Existing arithmetic sequence-based periodicity determination method. Let T_H denote the time support set of a subgraph H, i.e., the set of timestamps at which H appears. If T_H includes an arithmetic sequence of length $\sigma \geq 3$, H is deemed a periodic subgraph. **Illustrative Example.** Figure 1 depicts a temporal academic collaboration network $\mathcal{G}$, where vertices represent researchers and edges describe their collaborations over ten years. For example, researchers v_3 and v_8 collaborated in years t_2 and t_4. The subgraph H_1 induced by the vertex set $V_1 = \{v_4, v_5, v_6\}$ (encircled in blue lines) has time support set $T_{H_1} = \{t_1, t_3, t_4, t_5, t_7, t_{10}\}$. Since T_{H_1} contains an arithmetic sequence $AS_1 = \{t_1, t_4, t_7, t_{10}\}$ with $\sigma = 4$, H_1 is identified as periodic. *Nevertheless, this arithmetic sequence–based method suffers from three key limitations:*

L1: Irrational periodicity determination. Existing works judge the periodicity of a subgraph H by identifying an arithmetic sequence of timestamps. This approach only focuses on isolated timestamps, however, and overlooks the full time interval, leading to irrational periodicity determinations. For instance, in above example, considering only timestamps t_1, t_4, t_7, and t_{10} would incorrectly classify H_1 as periodic. When evaluating the complete time interval $[t_1, t_{10}]$, by contrast, H_1 is not actually periodic.

L2: Overreliance on single-timestamp occurrences. Existing works identify periodicity under the assumption that a subgraph H manifests at discrete, individual timestamps, a premise that lacks reasonableness in real-world applications. To illustrate this limitation, consider a subgraph with a time support set $\{t_1, t_2, t_6, t_7, t_{11}, t_{12}\}$. Using existing approaches, two "periodic" sequences are identified, i.e., $\{t_1, t_6, t_{11}\}$ and $\{t_2, t_7, t_{12}\}$. However, these sequences are clearly not truly periodic in practical terms.

L3: Inability to capture periodicity across multiple intervals. Existing works are limited to detecting periodic patterns within a single time interval, failing to capture periodic behaviors that span multiple disjoint intervals. For example, a subgraph H_3 with $T_{H_3} = \{t_2, t_4, t_6, t_8, t_{12}, t_{14}, t_{16}\}$ exhibits two periodic behaviors $\{t_2, t_4, t_6, t_8\}$ and $\{t_{12}, t_{14}, t_{16}\}$. Existing methods can identify at most one of these, missing the full scope of periodic events.

To address these limitations, we investigate periodic community search in temporal graphs, a problem not previously explored. First, we introduce a new concept of k-core time series k-$TS(V)$ for a vertex set V, represented as a binary series. For a timestamp t, the t-th bit of k-$TS(V)$ is 1 if the induced subgraph of V at snapshot G_t is a k-core, and 0 otherwise, where G_t denotes the graph consisting of all vertices and edges present at t. Building upon this concept, we propose a k-core time series-based periodicity determination approach. Specifically, a maximal periodic k-core time series $S \subseteq k$-$TS(V)$ contains at least two successive ε-approximate subseries of equal

length. In this context, if the Jaccard similarity between two k-core time series equals or exceeds a threshold ε, they are considered ε-approximate. A vertex set V can form a periodic k-core if its k-$TS(V)$ includes at least one such maximal periodic k-core time series. We then formally define our local periodic community (LPC) search problem. Given a query vertex q, an integer k, and a threshold ε, the LPC aims to identify the maximal periodic k-core containing q with the highest occurrence frequency of its periodic pattern, i.e., the maximal number of ε-approximate subseries within a single maximal periodic k-core time series.

Example 1.1. We utilize Figure 1 to illustrate our k-core time series-based periodicity determination approach and the LPC search problem. Taking $q = v_2$, $k = 2$, and $\varepsilon = 0.8$, we examine the vertex set $V' = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ marked in red, whose induced subgraph forms a 2-core solely in snapshots G_1, G_4, G_7 and G_{10}. Thus, its 2-core time series is 2-$TS(V')$ = 1001001001, showcasing a distinct periodic pattern '100' repeated three times. The Jaccard similarity between any two periodic subseries is 1 (as they are identical), surpassing 0.8. Then, the subseries $S' = 100100100 \subseteq 2\text{-}TS(V')$ is a maximal periodic 2-core time series with a periodic pattern frequency of 3. Since V' is a maximal vertex set containing v_2 that achieves the highest occurrence frequency of its periodic pattern within S', it is identified as a LPC.

Compared to the arithmetic sequence-based periodicity model, our k-core time series-based approach provides a more flexible and realistic characterization of periodic communities, enabling practical applications: (1) **Biological Behavior Analysis [22]**. In animal social networks, LPC can be employed to detect collective behaviors such as seasonal migration, foraging patterns, or mating gatherings that recur in response to environmental cues with varying intervals. This aids in ecosystem monitoring, behavioral ecology, and species conservation planning. (2) **Social Network Prediction [55]**. In human interaction networks, LPC can uncover complex recurring activities, such as multi-day book club retreats, study groups with evolving collaboration patterns, and irregular project sprints, facilitating accurate trend forecasting, dynamic community recommendation, and targeted content or event promotion.

A straightforward approach to address the LPC search is to enumerate all $2^{|\mathcal{V}|-1}$ vertex subsets containing q, where $\mathcal{V}$ denotes the vertex set of temporal graph. Then, for each subset V, we compute the k-core time series k-$TS(V)$, and use signal processing techniques [42] to determine its period set $\mathcal{P}$. For each period $p \in \mathcal{P}$, the maximal periodic k-core time series and their periodic pattern frequencies are extracted by checking Jaccard similarity among length-p subseries of k-$TS(V)$. After evaluating all subsets, LPC is identified. However, due to the exponential search space, this approach is computationally infeasible for large-scale temporal graphs, thereby posing a significant challenge for efficient periodic community search.

To tackle this challenge, we firstly develop a time-deletion-based (TD) algorithm that iteratively processes and deletes each timestamp. Specifically, for each timestamp, the TD algorithm leverages a newly designed *time indicator series* to generate the corresponding *maximal temporal core set*, which serves as a candidate vertex set for LPC. This avoids enumerating all vertex subsets and effectively narrows the search space. To efficiently generate all maximal temporal core sets, we propose an incremental generation method based on the monotonicity of maximal temporal core sets. After processing all timestamps, the TD algorithm evaluates all candidate vertex subsets as the baseline to get the final result. To further accelerate the TD algorithm, we present a novel TP-Core index Φ_c, which stores the *time-pair coreness*. For a vertex u and time pair $\mathbb{T} = \{t_s, t_e\}$, the time-pair coreness $c(u, \mathbb{T})$ is defined as the maximum k such that u is contained in a common non-empty k-core at both snapshots G_{t_s} and G_{t_e}. By recording time-pair coreness for each vertex in a compressed manner, the TD algorithm eliminates substantial redundant computations involved in generating maximal temporal core sets, leading to a significant efficiency improvement.

However, in cases such as dense graphs, the TD algorithm may still need to evaluate a large number of vertex subsets. To avoid vertex subsets enumeration, we propose a vertex-deletion-based (VD) algorithm. Specifically, the VD algorithm utilizes a new concept called *time list*. For a vertex u, u's time list $TL(u)$ is the set of all timestamps where u and the query vertex q co-occur in a k-core within the corresponding snapshots. For each vertex u, the VD algorithm enumerates all sublists $sl \subseteq TL(u)$, each of which induces a k-core time series. If none of these k-core time series contains a maximal periodic k-core time series, u cannot belong to any LPC and can thus be safely deleted. Otherwise, a candidate LPC containing u is identified. By iteratively pruning unpromising vertices, the VD algorithm significantly reduces the search space and improves overall efficiency. Additionally, the VD algorithm leverages the proposed TP-Core index and incorporates a batch deletion strategy to further enhance performance.

Our key contributions are summarized as follows:

- We introduce a k-core time series-based method for periodicity determination and formally define the problem of periodic community search.
- We propose a TD algorithm to address the problem and design a compact TP-Core index to accelerate it.
- We develop a more efficient VD algorithm, enhanced by a batch deletion strategy and the TP-Core index.
- We conduct extensive experiments on real-world graphs to demonstrate the effectiveness and efficiency of our proposed model and algorithms.

Roadmap. Section 2 reviews related work. Section 3 formalizes the problem and presents a baseline. Sections 4 and 5 introduce the TD and VD algorithms, respectively, along with the TP-Core index and optimizations. Experimental results are reported in Section 6. Finally, Section 7 concludes the paper.

2 Related Work

Community search. Community search aims to find a densely connected subgraph containing a given query vertex [10, 13]. In static graphs, existing community models include *dense subgraph-based models* like k-core [8, 39], k-truss [12, 26], k-clique [7, 49], and k-plex [43], as well as *predefined function-based models* such as density [47], connectivity [36, 37, 41], and conductance [1, 24]. This exploration has also been extended to diverse graph types such as directed [23, 26], attributed [27, 44], multilayer [29, 40], and heterogeneous graphs [6, 21].

In the context of temporal graphs, existing community search methods can be broadly classified into two paradigms: *time-aware community search* and *temporal feature-aware community search*. The former identifies communities satisfying structural criteria within a specific time interval [14, 48], while the latter uncovers communities exhibiting notable temporal features such as persistence [19, 20], high activity [2, 18], and significant engagement patterns [54]. *Despite these advances, periodic community search in temporal graphs has not yet been explored.*

Periodic subgraph mining. Periodic community search is closely related to periodic subgraph mining in temporal graphs, which enumerates all subgraphs satisfying both structural cohesiveness and a predefined periodicity model. Early studies [11, 16, 17] focused solely on subgraphs appearing at consistent time intervals, overlooking structural density. Zhang et al. [52] extended this to seasonal-periodic subgraphs with distinct periods over multiple consecutive time intervals; however, they only set an upper limit for those periods, neglecting lower limits. Recent studies have integrated cohesive structures. Qin et al. [32, 33, 35] mined periodic subgraphs based on dense models such as k-core and k-clique, focusing on strictly arithmetic periodicity while disregarding quasi-periodicity,

e.g., near-arithmetic repetitions with bounded deviations. In contrast, Zeng et al. [51] formally addressed quasi-periodic community mining, relaxing strict periodicity assumptions.

Despite these advances, all aforementioned methods except [52] rely on an *arithmetic sequence-based periodicity determination model*. Specifically, a subgraph H is considered (quasi-)periodic if its appearance timestamps form a (quasi-)arithmetic sequence of length $\sigma \geq 3$. As analyzed in the Introduction, this model suffers from three key limitations that hinder the identification of complex, truly periodic patterns. *Consequently, a more robust and flexible framework for periodicity determination is needed to uncover real-world, meaningful periodic communities.*

Discussion. Our periodic community search problem differs from existing periodic subgraph mining problem in three key aspects. (i) *Objective and scope.* We target query-driven discovery of a periodic community containing a given query vertex, whereas prior works perform global enumeration of all periodic subgraphs without a query constraint. (ii) *Periodicity modeling.* Existing methods employ arithmetic sequences to define periodicity, while we detect periodicity through time series analysis, facilitating rational periodicity determination and capturing more flexible and robust periodic patterns. (iii) *Efficiency.* Global enumeration is computationally expensive. By considering the query vertex, our proposed methods achieve significantly higher efficiency and are better suited for interactive or real-time applications, such as biological behavior analysis and dynamic social network prediction. Thus, our work diverges from prior arts in both problem formulation and algorithmic approach, with enhanced practical applicability.

3 Preliminaries

In this section, we formally define the problem of periodic community search and present the baseline algorithm.

3.1 Problem Definition

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}) = \{G_1, G_2, ..., G_T\}$ be a temporal graph consisting of a sequence of snapshot graphs, where $\mathcal{V}$ and $\mathcal{E}$ represent the set of vertices and temporal edges, respectively. The snapshot of $\mathcal{G}$ at timestamp $t \in [1, T]$ is defined as $G_t = (V_t, E_t)$. We assume that all snapshots share the same vertex set, i.e., $\forall t \in [1, T]$, $V_t = \mathcal{V}$. In a simple static graph $G = (V, E)$, for a vertex set $U \subseteq V$, the induced subgraph of G on U is denoted by $G[U]$. The degree of a vertex $u \in V$ is defined as $deg(u, G) = |\{v \in V | (u, v) \in E\}|$.

Definition 3.1. ***(k-core)*** Given a simple graph G and a vertex set $U \subseteq V$, the induced subgraph $G[U]$ of G on U is a k-core if each vertex in U has a degree of at least k within $G[U]$, i.e., $\forall u \in U$, $deg(u, G[U]) \geq k$.

Definition 3.2. ***(k-core Time Series)*** Given a temporal graph $\mathcal{G} = \{G_1, G_2, ..., G_T\}$, a vertex set $U \subseteq \mathcal{V}$, and a parameter k, the k-core time series of U, denoted by k-$TS(U)$, is a binary series of length T. Specifically, for a timestamp $t \in [1, T]$, the t-th bit of k-$TS(U)$ is defined as:

$$k\text{-}TS(U)_t = \begin{cases} 1, & G_t[U] \text{ is a } k\text{-core;} \\ 0, & \text{otherwise.} \end{cases}$$

In a word, the k-core time series of U indicates the specific timestamps at which U forms a k-core. For clarity, Figure 2 presents the ten snapshots of the temporal graph $\mathcal{G}$ from Figure 1. Given $k = 2$, the vertex set $V_1 = \{v_1, v_2, v_3, v_7, v_8\}$ forms 2-cores in snapshots G_2, G_4, G_8, and G_{10}. Accordingly, 2-$TS(V_1)$ = 0101000101. Notably, such time series may contain repeatedly occurring periodic patterns [22, 53]. For instance, in 2-$TS(V_1)$, the periodic pattern '01' appears four times. However, in real-world scenarios, strict periodicity is often disrupted by noise such as schedule

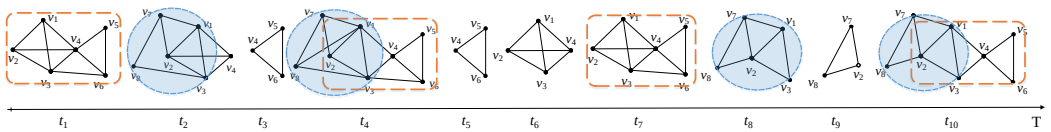


Fig. 2. Ten Snapshots of $\mathcal{G}$ (with Highlighted Vertex Sets $V_1 = \{v_1, v_2, v_3, v_7, v_8\}$ and $V_2 = \{v_1, v_2, v_3, v_4, v_5, v_6\}$)

changes or emergency event [51]. To improve the detection of recurring periodic patterns amid such disruptions, we focus on quasi-periodic patterns. To achieve this, we first introduce the concept of k-core time series similarity.

Definition 3.3. ***(ε-Approximate k-core Time Series)*** Given two k-core time series S_1 and S_2 of identical length, and a threshold $\varepsilon \in [0, 1]$, S_1 and S_2 are ε-approximate, denoted by $S_1 \overset{\varepsilon}{\sim} S_2$, if their Jaccard similarity is no less than ε, i.e.,

$$\mathcal{J}(S_1, S_2) = \frac{|\mathcal{T}_{S_1} \cap \mathcal{T}_{S_2}|}{|\mathcal{T}_{S_1} \cup \mathcal{T}_{S_2}|} \geq \varepsilon,$$

where $\mathcal{T}_S$ denotes the set of timestamps at which the time series S takes the value 1.

For example, given two k-core time series $S_1 = 0101000101$ and $S_2 = 0101010101$, and a threshold $\varepsilon = 0.8$. Since $\mathcal{J}(S_1, S_2) = 4 \div 5 = 0.8$, $S_1 \overset{0.8}{\sim} S_2$. Note that if two k-core time series are identical, their Jaccard similarity is 1. In practice, the value of ε is usually not very small, e.g., $\varepsilon < 0.1$. It is because real-world periodic patterns typically exhibit strong inherent regularity, with k-core time series similarities well above 0.1, as confirmed in the experiment (Exp-5). Based on it, we define the *maximal periodic k-core time series*.

Definition 3.4. ***(Maximal Periodic k-core Time Series)*** Given a k-core time series S, a subseries $S' \subseteq S$, and a threshold $\varepsilon \in [0, 1]$, if S' is a *maximal periodic k-core time series*, it should satisfy the following two conditions.

(1) ***Periodicity***. S' can be partitioned into at least two disjoint subseries of equal length, i.e., $S' = S'_1 S'_2 ... S'_m (m \geq 2)$. Every pair of these subseries is ε-approximate, i.e., $\forall S'_i,\ S'_j \in \{S'_1, S'_2, ..., S'_m\}$, $S'_i \overset{\varepsilon}{\sim} S'_j$. Here, each S'_i is referred to as the periodic pattern of S'.

(2) ***Maximality***. There does not exist another subseries $S'' \subseteq S$ such that $S' \subset S''$ and S'' shares the same periodic pattern as S'.

For instance, consider the k-core time series 010101000100: the subseries 010101 is the maximal periodic k-core time series with its periodic pattern being 01. It is worth mentioning that a single k-core time series may contain multiple maximal periodic k-core time series. For example, the k-core time series 10101000001010 contains two maximal periodic k-core time series, i.e., 101010 within the first six timestamps and 1010 within the last four timestamps. To formalize this, we use $\mathcal{PS}(S)$ to denote all maximal periodic k-core time series contained in a k-core time series S. Furthermore, in a maximal periodic k-core time series $S' \in \mathcal{PS}(S)$, the periodic pattern occurs repeatedly. We define $\mathcal{F}(S')$ as the occurrence frequency of this periodic pattern in S'. For instance, for the maximal periodic k-core time series $S' = 010101$, since the periodic pattern 01 appears 3 times, $\mathcal{F}(S') = 3$.

Given a temporal graph $\mathcal{G} = \{G_1, G_2, ..., G_T\}$, an integer k, and a vertex set $U \subseteq \mathcal{V}$, if U's k-core time series contains at least one maximal periodic k-core time series, i.e., $\mathcal{PS}(k\text{-}TS(U)) \neq \emptyset$, we say that U can form a *periodic k-core*. In $\mathcal{G}$, multiple periodic k-cores may exist, each characterized by distinct periodicities. In this paper, we focus specifically on periodic k-cores with the highest occurrence frequency of their respective periodic patterns. Although there are other alternative objectives such as total cycle length, maximizing the frequency of periodic patterns can help

to retrieve more reliable communities. Specifically, in temporal data analysis, the frequency of a periodic pattern is a key indicator of its stability and reliability. A higher frequency signifies more regular recurrences, reducing the likelihood that the pattern is a product of random noise or sporadic coincidence. Consequently, a community that emerges frequently at regular intervals is more substantively meaningful. Guided by this rationale, we define our problem as follows.

Problem 1. **(Local Periodic Community (LPC) Search)** *Given a temporal graph $\mathcal{G}$, a parameter k, a threshold $\varepsilon \in [0, 1]$, and a query vertex q, the goal of LPC search is to find a vertex set $U \subseteq \mathcal{V}$ that satisfies the following three conditions.*

(1) ***Containment****. The query vertex is included in U: $q \in U$.*

(2) ***Local Periodicity****. The k-core formed by U exhibits periodic behavior with the highest occurrence frequency of its periodic pattern within a single maximal periodic k-core time series i.e., $U = \arg\max_{V \subseteq \mathcal{V}} \max_{S \in \mathcal{PS}(k\text{-}TS(V))} \mathcal{F}(S)$.*

(3) ***Maximality****. There does not exist a vertex set $U' \subseteq \mathcal{V}$ such that $U \subset U'$ and U' satisfies conditions (1) and (2).*

In essence, Problem 1 identifies the maximal periodic k-core containing q with the most frequently recurring pattern within a single contiguous time interval, revealing short-term, event-driven periodic interactions like ad hoc project team collaborations. For example, in Figure 2, let $k = 2$, $\varepsilon = 0.8$, and $q = v_2$. For a vertex set $V_2 = \{v_1, v_2, v_3, v_4, v_5, v_6\}$, its k-core time series is 2-$TS(V_2)$ = 1001001001. The corresponding maximal periodic k-core time series is 100100100, with a periodic pattern frequency of 3, the highest frequency observed across any single maximal periodic 2-core time series. Thus, Problem 1 returns V_2 as the LPC.

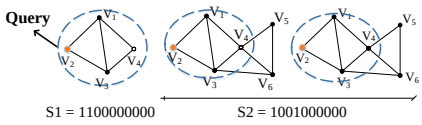
Discussion. The LPC search can be extended naturally by altering condition (2) in Problem 1, particularly by modifying the approach used to compute the periodic pattern frequency.

- Global Periodic Community (GPC) Search. GPC finds the maximal periodic k-core whose periodic pattern achieves the highest cumulative occurrence frequency across all maximal periodic k-core time series: $U = \arg\max_{V \subseteq \mathcal{V}} \sum_{S \in \mathcal{PS}(k\text{-}TS(V))} \mathcal{F}(S)$. Unlike LPC, GPC focuses on capturing long-term periodic behavior persisting across multiple time intervals, such as annual conference attendee interactions.
- Emergent Periodic Community (EPC) Search. EPC prioritizes recently active periodicity by exponentially discounting older occurrences [5] using a discount factor $\gamma \in (0, 1)$: $U = \arg\max_{V \subseteq \mathcal{V}} \max_{S \in \mathcal{PS}(k\text{-}TS(V))} \gamma^{T - \mathcal{T}_S^m} \cdot \mathcal{F}(S)$, where $\mathcal{T}_S^m = \min\{\mathcal{T}_S\}$ denotes the earliest occurrence timestamp of the maximal periodic k-core time series S. Differing from LPC, EPC emphasizes emergent periodic behavior whose recurrence is more strongly correlated with the present, offering a reliable indicator of ongoing structural patterns.

Notably, the primary distinction between Problem 1 and its extensions lies solely in the way the periodic pattern frequency is calculated. Since our algorithms (presented in Sections 4 and 5) are designed around flexible periodic pattern frequency evaluation, they can be readily extended to address these variants.

3.2 Baseline

We first introduce a straightforward approach to address the LPC search. Initially, we enumerate all possible vertex subsets $U \subseteq \mathcal{V}$ containing the query vertex q. Then, for each vertex subset U, we compute the frequency of periodic pattern within a single maximal periodic k-core time series of U. Finally, the vertex set with the highest periodic pattern frequency is returned as result.



(a) Initialization of *Candi* (t_1 vs. $\{t_2, t_3, t_4\}$)

MV$_1$: {v$_1$, v$_2$, v$_3$, v$_4$} → 1101011001; MV$_2$: {v$_1$, v$_2$, v$_3$, v$_4$, v$_5$, v$_6$} → 1001001001; MV$_3$: {v$_1$, v$_2$, v$_3$, v$_4$, v$_5$, v$_6$, v$_7$, v$_8$} → 0001000001; MV$_4$: {v$_1$, v$_2$, v$_3$, v$_4$, v$_7$, v$_8$} → 0101000001; MV$_5$: {v$_1$, v$_2$, v$_3$, v$_7$, v$_8$} → 0101000101; MV$_6$: {v$_1$, v$_2$, v$_3$} → 1101011101.

(b) Final *Record*

Fig. 3. Examples of TD Algorithm. Here, $q = v_2$, $k = 2$ and $\varepsilon = 0.8$.

An essential stage in this process involves detecting periods in the k-core time series k-$TS(U)$ for a vertex set U. In discrete signal processing, Fourier transform (FT) and Autocorrelation (AC) are prominent methods for periodicity analysis [4, 22, 28, 42]. The integration of FT and AC, pioneered by [42], has notably enhanced performance in period detection. Thus, we adopt this combined approach, which consists of three steps.

Step-1: Fourier Transform and Periodogram Construction. Given a k-core time series $S = s_1 s_2 \dots s_T$, FT maps S into the frequency domain, yielding a sequence of complex coefficients: $\mathbf{X} = \{\mathbf{x}_i \mid i \in [1, T]\}$, denoted by $\mathbf{X} = \mathsf{FT}(S)$. The periodogram is then defined as $\mathcal{D} = \{d_i = |\mathbf{x}_i|^2 | i \in [1, T]\}$, where each d_i represents the spectral power at frequency i.

Step-2: Candidate Period Identification via Thresholding. To distinguish meaningful periodic signals from noise, we establish a spectral power threshold $\Theta_{99}(S)$ at the 99% confidence level, following existing work [22]. This involves generating 100 random permutations S' of S, computing the maximum spectral power $d'_m = \max\{d'_i = |\mathbf{x}'_i|^2 | \mathbf{x}'_i \in \mathsf{FT}(S')\}$ for each permutation, and determining $\Theta_{99}(S)$ as the 99th largest value among the set $\{d^1_m, d^2_m, \dots, d^{100}_m\}$. A spectral frequency f[1], is deemed significant if its power d_f exceeds $\Theta_{99}(S)$, corresponding to a candidate period interval $I^f_p = [T/f, T/(f-1))$ in time domain.

Step-3: Period Refinement via Autocorrelation. AC measures the similarity between the k-core time series S and its lagged versions. Specifically, for a lag $\alpha \in [1, T)$, $\mathsf{AC}(S, \alpha) = \sum_{i=1}^{T} s_i s_{(i+\alpha)\%T}$. For each candidate period interval $[st, ed)$ obtained from Step-2, if $\mathsf{AC}(S, p)$ is a local peak within the range $\{\mathsf{AC}(S, st), \dots, \mathsf{AC}(S, ed-1)\}$, then $p \in [st, ed)$ is identified as a period of S.

This three-step period detection approach determines the period set $\mathcal{P}$ for the k-core time series k-$TS(U)$. For each period $p \in \mathcal{P}$, we segment k-$TS(U)$ to extract all maximal periodic k-core time series $\mathcal{PS}(k\text{-}TS(U))$, and compute their periodic pattern frequencies.

Summary. The primary limitation of this baseline is its high computational complexity. There are $2^{|\mathcal{V}|-1}$ vertex subsets containing q. For each vertex subset U, computing its k-core time series k-$TS(U)$ takes $O(T \cdot (|\mathcal{V}| + E_m))$ time, where $E_m = \max_{i \in [1,T]} |E_i|$. In the period detection phase, FT and AC entail $O(T \cdot \log T)$ time, while identifying all maximal periodic k-core time series incurs additional $O(|\mathcal{P}| \cdot T)$ time. Since $|\mathcal{P}| \ll \log T$, this phase is dominated by $O(T \cdot \log T)$. Thus, the total time complexity is $O(2^{|\mathcal{V}|-1} \cdot T \cdot (|\mathcal{V}| + E_m + \log T))$, which is infeasible for large-scale temporal graphs. To address this, we present two optimized approaches based on different design principles: time-deletion-based algorithms (Section 4) and vertex-deletion-based algorithms (Section 5).

4 Time-Deletion-Based Algorithms

This section introduces the time-deletion-based (TD) algorithm for solving the LPC search. Furthermore, a novel TP-Core index is designed to enhance algorithm's efficiency.

[1]Terminology clarification. The spectral frequency f in Step-2 refers to a signal frequency (Hz), determined solely by the temporal structure of S. In contrast, the occurrence frequency $\mathcal{F}(\cdot)$ in Problem 1 counts how many times the periodic pattern repeats in a maximal periodic k-core time series. Hence, f guides period detection and is independent of problem parameters such as k or $\mathcal{F}(\cdot)$.

4.1 Basic TD Algorithm

To avoid exhaustively enumerating all $2^{|\mathcal{V}|-1}$ vertex subsets containing the query vertex q, the TD algorithm leverages monotonicity to effectively reduce the search space. However, existing monotonicity-based methods are not directly applicable to the LPC search due to varying forms of monotonicity across different problems, leading to distinct vertex/edge removal criteria. For example, TFSM [50] exploits subgraph-overlap fractional monotonicity to efficiently enumerate all frequent subgraphs, a method not suited for identifying periodic communities.

To address this, we introduce a *novel temporal monotonicity* based on inclusion among time indicator series. Formally, we define the time indicator series S as a binary series of length T, where $S_t = 1$ indicates that a community must form a k-core at timestamp t. Subsequently, TD iteratively processes timestamps, derives *maximal temporal core sets* from the corresponding time indicator series as LPC candidates, and leverages their inclusion-based monotonicity for efficient incremental candidate generation.

Definition 4.1. ***(Maximal Temporal Core Set, MC)*** Given a temporal graph $\mathcal{G}$, a parameter k, a query vertex q, and a time indicator series S, the maximal temporal core set of S, denoted by $MC_q^k(S)$, is the vertex set $U \subseteq \mathcal{V}$ satisfying:

(1) U contains the query vertex: $q \in U$.

(2) For every timestamp t where $S_t = 1$, the induced subgraph $G_t[U]$ is a k-core.

(3) No superset $U' \supset U$ satisfies conditions (1) and (2).

Intuitively, $MC_q^k(S)$ represents the largest community that forms a k-core at timestamps t where $S_t = 1$, without any structural constraints at timestamps where $S_t = 0$. Take Figure 2 as an example. Given $k = 2$, $q = v_2$, and a time indicator series $S = 0001000000$, $MC_{v_2}^2(S) = \{v_1, v_2, \ldots, v_8\}$, since no larger vertex set forms 2-core at snapshot G_4. Building on this, we introduce *series inclusion* to formalize the monotonicity of maximum temporal core sets.

Definition 4.2. ***(Series Inclusion)*** Given two time indicator series S and S' of equal length, if $S \vee S' = S'$, we say S' includes S, denoted by $S \leq S'$.

For example, given $S1 = 01100$ and $S2 = 01101$, since $S1 \vee S2 = 01101 = S2$, $S1 \leq S2$.

Theorem 4.3. *Given a temporal graph $\mathcal{G}$, a parameter k, a query vertex q, and two time indicator series S and S', if $S \leq S'$, then $MC_q^k(S') \subseteq MC_q^k(S)$.*

Proof. Suppose $MC_q^k(S') \supset MC_q^k(S)$. Given $S \leq S'$, it is evident that $MC_q^k(S')$ can form a connected k-core at snapshot G_t where $S_t = 1$. This contradicts the maximality of $MC_q^k(S)$. □

For instance, in Figure 2 with $k = 2$ and $q = v_2$, given a time indicator series $S' = 1001000000$, $MC_{v_2}^2(S') = \{v_1, \ldots, v_6\}$. Since $S \leq S'$, it follows that $MC_{v_2}^2(S') \subseteq MC_{v_2}^2(S)$. When the context is clear, we use $MC(S)$ instead of $MC_q^k(S)$.

Theorem 4.3 illustrates the monotonicity of maximal temporal core sets with respect to series inclusion. Specifically, if $MC(S) = \emptyset$, then $MC(S') = \emptyset$ for all $S \leq S'$. Leveraging this property, we design *an incremental generation theorem* to compute all non-empty maximal temporal core sets as candidate LPCs.

To formalize this process, we present two key concepts: (i) Given a time indicator series S, the number of '1's in S is denoted as $n(S)$. (ii) For an integer $l \in [1, n(S)]$, $S[:l]$ represents the *prefix series* of S up to and including its l-th '1', following [30]. For example, given $S1 = 01100$, then $n(S1) = 2$, $S1[:1] = 01$ and $S1[:2] = 011$.

Theorem 4.4. *Given a temporal graph $\mathcal{G}$, a parameter k, a query vertex q, and two time indicator series $S1$ and $S2$ of length T, let $S = S1 \vee S2$ and $\mathcal{M} = MC(S1) \cap MC(S2)$. If $n(S1) = n(S2) = i (i \geq 2)$,*

$S1[:i-1] = S2[:i-1]$, and there exists $U \subseteq \mathcal{M}$ containing q such that $G_t[U]$ is a k-core for all t with $S_t = 1$, then $MC(S) \neq \emptyset$.

Proof. By assumption, $S1$ and $S2$ share the same first $i-1$ active timestamps and differ only in their i-th '1', so $S = S1 \vee S2$ activates the union of their timestamps. If there exists a vertex set U containing q such that $G_t[U]$ forms a k-core for each t with $S_t = 1$, then U is a valid temporal core set of S, implying $MC(S) \neq \emptyset$. □

Algorithm 1: Time-Deletion-Based Algorithm (TD)

Input: A temporal graph $\mathcal{G}$, a query vertex q, parameters k, ε, and thresholds Θ_{99}
Output: Local periodic community (LPC)

$\mathcal{R} \leftarrow \emptyset$; $\mathcal{F}_{max} \leftarrow 0$; $Record \leftarrow \emptyset$;
for *each timestamp* $t \in [1, T]$ **do**
  $Candi \leftarrow \emptyset$;
  for *each timestamp* $t' \in [1, T]$ **do**
    $S \leftarrow$ time indicator series of length T with $S_t = S_{t'} = 1$;
    $M \leftarrow Compute(V_t \cap V_{t'}, S, q, k)$; // Algorithm 2
    if $M \neq \emptyset$ **then**
      $Candi[S] \leftarrow M$;
  while $Candi \neq \emptyset$ **do**
    $C \leftarrow \emptyset$;
    for $\forall (Si, Mi) \in Candi$ **do**
      $l \leftarrow n(Si)$;
      for $\forall (Sj, Mj) \in Candi$ **do**
        if $n(Sj) = l$ ***and*** $Si[:l\text{-}1] = Sj[:l\text{-}1]$ **then**
          $S \leftarrow Si \vee Sj$; $M \leftarrow Compute(Mi \cap Mj, S, q, k)$;
          if $M \neq \emptyset$ **then**
            $C[S] \leftarrow M$;
      if $Mi \notin Record$ **then**
        $Record[Mi] \leftarrow Si$;
      else
        $Record[Mi] \leftarrow Record[Mi] \vee Si$;
      delete Si from $Candi$;
    $Candi \leftarrow C$;
  delete G_t from $\mathcal{G}$;
for $\forall (U, S = k\text{-}TS(U)) \in Record$ **do**
  identify the periods of S and compute $\mathcal{PS}(S)$ (subSection 3.2);
  determine U's periodic pattern frequency $\mathcal{F}$;
  if $\mathcal{F} > \mathcal{F}_{max}$ **then**
    $\mathcal{R} \leftarrow U$; $\mathcal{F}_{max} \leftarrow \mathcal{F}$;
return $\mathcal{R}$.

Theorem 4.4 enables the incremental generation of all non-empty maximal temporal core sets. Specifically, for each timestamp $t \in [1, T]$, we initialize time indicator series S of length T such that $S_t = 1$ and $n(S) = 2$. Then, new time indicator series S' with $n(S') = i > 2$ are generated by merging pairs of existing time indicator series $S1$ and $S2$ satisfying $S1[:i\text{-}1] = S2[:i\text{-}1]$ and $MC(S') \neq \emptyset$. This process iterates until no further merges produce a non-empty maximal temporal core set. All distinct time indicator series S with $S_t = 1$ and $MC(S) \neq \emptyset$ are thus enumerated, after which t is deleted. This approach is suitable for handling all timestamps. Finally, by merging identical maximal temporal core sets, all candidate LPCs and their k-core time series can be correctly computed.

Next, for each maximal temporal core set U, we apply the period detection approach outlined in Section 3.2 to find the periods of its k-core time series $k\text{-}TS(U)$. Following this, we thus identify

all the maximal periodic k-core time series, leading to the determination of the LPC from these candidates.

Based on the above discussion, we propose the basic time-deletion-based algorithm TD for LPC search. Algorithm 1 shows the pseudo-code of TD. Specifically, it sequentially processes each timestamp $t \in [1, T]$. For each t, TD initializes time indicator series (lines 2-8), and uses the incremental generation method in Theorem 4.4 to derive all maximal temporal core sets (lines 9-17). The function Compute is utilized to obtain the maximal temporal core set M of a given time indicator series S (lines 6 and 15). Upon processing all timestamps, *Record* contains all maximal temporal core sets and their k-core time series (lines 18-21). Next, for each candidate U in *Record*, TD computes its maximal periodic k-core time series $\mathcal{PS}(S)$, and determines the periodic pattern frequency $\mathcal{F}$. If $\mathcal{F}$ exceeds the current highest frequency $\mathcal{F}_{max}$, U is recorded, and $\mathcal{F}_{max}$ is updated (lines 26-29). Finally, TD returns the identified LPC.

Algorithm 2: Compute($\mathcal{V}_c$, S, q, k)

Input: A candidate vertex set $\mathcal{V}_c$, a time indicator series S, and parameters q, k
Output: The maximal temporal core set $MC(S)$

```
1  if q ∉ V_c then
2      return ∅.
3  D ← ∅;
4  for ∀t ∈ [1, T] and S_t = 1 do
5      for ∀u ∈ V_c do
6          if deg(u, G_t[V_c]) < k then
7              D ← D ∪ {u};
8  while D ≠ ∅ do
9      v ← D.front;
10     if v = q then
11         V_c ← ∅; break;
12     D ← D − {v};    V_c ← V_c − {v};
13     for ∀t ∈ [1, T] and S_t = 1 do
14         for each neighbor u of v in V_c do
15             if deg(u, G_t[V_c]) < k then
16                 D ← D ∪ {u};
17 return V_c.
```

Algorithm 2 shows the pseudo-code of the function Compute. If the vertex set $\mathcal{V}_c$ does not contain q, it directly returns $\emptyset$. Otherwise, a queue $\mathcal{D}$ of vertices to be deleted is initialized (lines 1-3). For each timestamp t with $S_t = 1$, if a vertex u in $G_t[\mathcal{V}_c]$ has degree less than k, it is added to $\mathcal{D}$ (lines 4-7). Here, $G_t[\mathcal{V}_c]$ is the induced subgraph of $\mathcal{V}_c$ at snapshot G_t. Then, Compute removes vertices from $\mathcal{D}$, updating $\mathcal{V}_c$, and adding neighbors of deleted vertices to $\mathcal{D}$ if their degrees fall below k (lines 8-16). This continues until $\mathcal{D}$ is empty. Finally, Compute returns the maximal temporal core set $MC(S)$ of a time indicator series S.

Example 4.5. Figures 2 and 3 illustrate Algorithm 1 with query parameters $q = v_2$, $k = 2$, and $\varepsilon = 0.8$. TD handles timestamps sequentially from 1 to 10. For $t = 1$, the time indicator series $S1 = 1100000000, S2 = 1010000000, S3 = 1001000000$, etc., are initialized. Since $MC(S1)$ and $MC(S3)$ are non-empty, they are added to the set *Candi* (in Figure 3(a)). The algorithm then processes each candidate in *Candi*. Since $n(S1) = n(S3) = 2$ and $S1[:1] = S3[:1]$, a new time indicator series $S' = S1 \vee S3$ is generated, and its non-empty maximal temporal core set $MC(S') = \{v_1, v_2, v_3, v_4\}$ (in blue dotted line) is added to *Candi*. This process terminates when no new time indicator series can

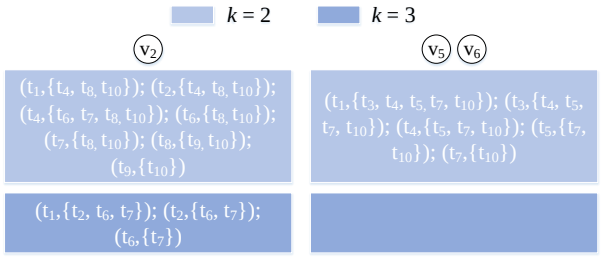


Fig. 4. The Compressed TP-Core Index.

be generated. Next, $t = 1$ is deleted. After processing all timestamps, the maximal temporal core sets and their k-core time series are recorded, as shown in Figure 3(b). Finally, TD evaluates those sets. For MC_1, no period can be found in 2-$TS(MC_1)$. Conversely, 2-$TS(MC_2)$ exhibits a period of 3, yielding a maximal periodic k-core time series 100100100 with a pattern frequency of 3. As no other maximal periodic k-core time series with a higher frequency is found, MC_2 is identified as a LPC and returned.

TD Complexity Analysis. Let N be the number of maximal temporal core sets (MC), with k-core time series of MC_i denoted by Si. Define $V_m = \max_i |MC_i|, E_m = \frac{1}{2}(\max_{i,j} \sum_{u \in MC_i} deg(u, G_j[MC_i]))$, and $L_s = \max_i n(Si)$, where $i \in [1, N]$ and $j \in [1, T]$. For each Si, Algorithm 1 calls Algorithm 2 $(n(Si) - 2)$ times to compute the MC_i, with each call costing $O(n(Si) \cdot (V_m + E_m))$ time. The total time cost is $O(n^2(Si) \cdot (V_m + E_m))$. Similarly, during the LPC identification phase, it takes $O(T \cdot \log T)$ time to compute the maximal periodic k-core time series. Considering each $Si(i \in [1, N])$, the total time complexity of Algorithm 1 is $O(N \cdot [L_s^2 \cdot (V_m + E_m) + T \cdot \log T])$.

4.2 Index-based Optimization

The TD algorithm reduces the search space by leveraging the monotonicity of maximal temporal core sets, but still incurs high computational cost due to repeated calls to the Compute function for each time indicator series. To improve query efficiency, we adopt indexing, a well-established graph mining technique. However, existing index structures are often problem-specific and not directly transferable. For example, tree-based indexes [46] are designed for similar subgraph queries, which do not align with our problem context. To this end, we propose a novel *compressed TP-Core index*, which precomputes and compactly stores common k-core structures across all time pairs, thereby reducing space overhead while maintaining efficiency.

TP-Core Index. In a temporal graph $\mathcal{G}$, the time-pair coreness of u for each time pair $\mathbb{T} = \{i, j\}$ is denoted by $c(u, \mathbb{T})$, referring to the maximum k where u belongs to a common non-empty k-core $H_{\mathbb{T}}^k$ in snapshots G_i and G_j. For each vertex $u \in \mathcal{V}$ and integer k, we define $\Phi(u, k)$ as the set of all time pair $\mathbb{T}$ for which u has time-pair coreness k, i.e., $\Phi(u, k) = \{\mathbb{T} \subseteq \{1, 2, \ldots, T\} | c(u, \mathbb{T}) = k\}$. Based on this, we define the TP-Core index as follows.

Definition 4.6. ***(TP-Core Index)*** Given a temporal graph $\mathcal{G}$, the TP-Core index of $\mathcal{G}$, denoted by Φ, is the set of $\Phi(u, k)$ for $\forall u \in \mathcal{V}$ w.r.t. $\forall k \in [2, c'_u]$. Here, $c'_u = max\{c(u, \mathbb{T}) | \mathbb{T} \subseteq \{1, 2, \ldots, T\}\}$.

Take the vertex v_2 in Figure 2 as an example. Since there is no $H_{\mathbb{T}}^k$ with $k > 3$ containing v_2, $c'_{v_2} = 3$. When $k = 3$, $\Phi(v_2, 3) = \{\{t_1, t_2\}, \{t_1, t_6\}, \{t_1, t_7\}, \{t_2, t_6\}, \{t_2, t_7\}, \{t_6, t_7\}\}$.

Index Compression. A direct representation of $\Phi(u, k)$ as a list of time pairs can lead to significant space redundancy. To enhance space efficiency, we introduce a compressed index representation.

Let O be a set of ordered time pairs $\mathbb{T}_o = \{t_s, t_e\}$, where $t_s < t_e$. Define $F_O = \{t_s | \{t_s, t_e\} \in O\}$ as the set of all start timestamps in O. For each $t \in F_O$, its aggregate set is $Agg(t, O) = \{t_e | \{t, t_e\} \in O\}$.

Algorithm 3: TP-Core Index Construction

Input: A temporal graph $\mathcal{G}$
Output: The compressed TP-Core index Φ_c

```
1  initialize Φ_c(u) for all u ∈ V;
2  for ∀t_s ∈ [1, T] do
3      k ← 2;
4      for ∀t_e ∈ [1, T] and t_e > t_s do
5          T_o = {t_s, t_e};    C ← H^k_{T_o};
6          while C ≠ ∅ do
7              C' ← H^{k+1}_{T_o};
8              for ∀u ∈ (C − C') do
9                  if k ∉ Φ_c(u) or t_s ∉ Φ_c(u, k) then
10                     initialize Φ_c(u, k)[t_s];
11                 Φ_c(u, k)[t_s] ← Φ_c(u, k)[t_s] ∪ {t_e};
12             C ← C';    k ← k + 1;
13 return Φ_c.
```

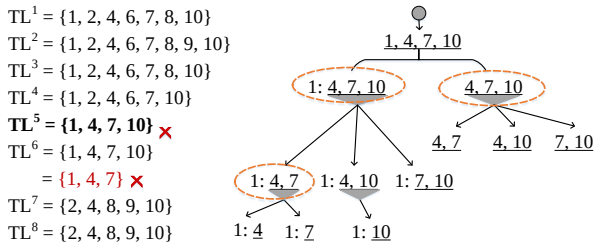
Definition 4.7. ***(Compressed TP-Core Index)*** Assume the time pairs in set $\Phi(u,k)$ are ordered. The compressed TP-Core Index $\Phi_c(u,k)$ is obtained by computing $Agg(t, \Phi(u,k))$ for $\forall t \in F_{\Phi(u,k)}$.

For $\Phi(v_2, 3)$, $F_{\Phi(v_2,3)} = \{t_1, t_2, t_6\}$. For timestamp t_1, we have $Agg(t_1, \Phi(v_2,3)) = \{t_2, t_6, t_7\}$. Thus, $\Phi(v_2,3)$ can be compressed to $\Phi_c(v_2,3) = \{(t_1, \{t_2, t_6, t_7\}), (t_2, \{t_6, t_7\}), (t_6, \{t_7\})\}$. All $\Phi(u,k)$ should undergo similar compression. This process generates the compressed index Φ_c. For example, Figure 4 shows the compressed TP-Core index for some vertices of $\mathcal{G}$ (in Figure 2).

Index Construction. We build the compressed TP-Core index by computing the time-pair coreness $c(u, \mathbb{T}_o)$ for each vertex u at all sorted time pair $\mathbb{T}_o$. The construction procedure is outlined in Algorithm 3. It iterates over each start timestamp $t_s \in [1, T]$ and initializes k to 2 (lines 2-3). Algorithm 3 then traverses end timestamp $t_e > t_s$ to generate the ordered time pair $\mathbb{T}_o$ (lines 4-5). At the current k, it computes the common connected k-core and $(k+1)$-core at $\mathbb{T}_o$, denoted as $H^k_{\mathbb{T}_o}$ and $H^{K+1}_{\mathbb{T}_o}$ (lines 5 and 7), respectively. For each vertex u in $H^k_{\mathbb{T}_o}$ but not in $H^{k+1}_{\mathbb{T}_o}$, its time-pair coreness $c(u, \mathbb{T}_o)$ is identified as k, and $\mathbb{T}_o$ is added to Φ_c (lines 8-11). Subsequently, k is increased by 1 until $H^k_{\mathbb{T}_o}$ becomes empty.

Complexity Analysis of TP-Core Index. **(i)** Time Complexity. Let $E_m = max_{i \in [1,T]} |E_i|$. For each time pair $\mathbb{T}_o = \{t_s, t_e\}$, computing the time-pair coreness $c(u, \mathbb{T}_o)$ for each vertex u is akin to performing core decomposition at G_{t_s} and G_{t_e}, which takes $O(2 \cdot (|\mathcal{V}| + E_m))$ time. As there are $(T^2 + T)/2$ time pairs, the total time complexity of Algorithm 3 is $O(T^2 \cdot (|\mathcal{V}| + E_m))$. **(ii)** Space Complexity. The uncompressed index $\Phi(u,k)$ stores all time pairs $\{i, j\}$ with time-pair coreness k for each vertex u, incurring $O(T^2 \cdot |\mathcal{V}|)$ space, prohibitive for large graphs. Our compressed index $Agg(t, \Phi(u,k))$ aggregates time pairs in $\Phi(u,k)$ by start timestamp t. Let $\alpha = \max_{u \in \mathcal{V}} |Agg(t, \Phi(u,k))|$. The space complexity reduces to $O(\alpha \cdot T \cdot |\mathcal{V}|)$. In practice, $\alpha \ll T$ due to temporal locality, yielding significant storage efficiency.

Index-based Optimization Algorithm. The TP-Core index accelerates Algorithm 1 as follows. For each timestamp $t \in [1, T]$, TD initializes a time indicator series S of length T, where $S_t = S_{t'} = 1$ and $t' \in (t, T]$. Here, S corresponds to a sorted time pair $\mathbb{T}_o = \{t, t'\}$. During a query involving vertex q and parameter k, we first check whether $c(q, \mathbb{T}_o) \geq k$ before invoking Compute. If not, indicating no common k-core containing q at $\mathbb{T}_o$, $MC(S) = \emptyset$. Thus, lines 6-8 in Algorithm 1 are bypassed. This approach reduces calls to the costly Compute function, particularly for large k or vertices with low time-pair coreness, leading to enhanced efficiency.



(a) Time List (b) Recursive generation tree of v_5

Fig. 5. Examples of VD Algorithm.

5 Vertex-Deletion-Based Algorithms

Despite its strategy of enumerating maximal temporal core sets, the TD algorithm struggles with efficiency in dense graphs due to a large number of unpromising candidates. To address this limitation, we propose the vertex-deletion-based (VD) algorithm for LPC search and two additional optimizations to improve efficiency.

5.1 Basic VD Algorithm

The VD algorithm efficiently prunes unpromising vertices by utilizing a new concept called *time list*. The time list of a vertex u, denoted as $TL(u)$, comprises all timestamps at which u co-occurs with the query vertex in a k-core. For each u, VD employs a *time list-based determination* criterion: it enumerates all sublists $sl \subseteq TL(u)$, each of which induces a k-core time series. If none of these k-core time series contains a maximal periodic k-core time series, u is safely removed. Otherwise, a *parent-list-based validation theorem* is utilized to identify a candidate LPC containing u.

Definition 5.1. ***(Time List, TL)*** In a temporal graph $\mathcal{G}$, given a query vertex q and a parameter k, the time list of a vertex $u \in \mathcal{V}$, denoted by $TL_q^k(u)$, is the set of timestamps $t \in [1, T]$ for which there exists a vertex subset $U_t \subseteq \mathcal{V}$ such that:

(1) U_t includes vertices q and u, i.e., $q, u \in U_t$.

(2) its induced subgraph $G_t[U_t]$ is a k-core.

Definition 5.1 allows the vertex set U_t to vary across timestamps. For each $t \in TL_q^k(u)$, it suffices that a k-core containing both q and u exists at timestamp t. For example, in Figure 2 with $q = v_2$ and $k = 2$, vertices v_5 and v_6 are contained in a 2-core induced by $U_t = \{v_1, \ldots, v_6\}$ at $t = 1, 4, 7, 10$. Thus, $TL_{v_2}^2(v_5) = TL_{v_2}^2(v_6) = \{1, 4, 7, 10\}$. We write $TL(u)$ when k and q are clear from context. Time lists for all other vertices are provided in Figure 5(a).

Based on this, the VD algorithm iteratively processes vertices in non-decreasing order of their time list lengths. Specifically, for the vertex u with the shortest time list, it proceeds as follows:

Time List-based Determination. A key observation is that, if u is part of a LPC, all timestamps when the community forms a k-core must be included in $TL(u)$. This leads to the following theorem.

Theorem 5.2. *Given a temporal graph $\mathcal{G}$, a query vertex q, and parameters k, ε, let $U \subseteq \mathcal{V}$ be a local periodic community in $\mathcal{G}$. For any vertex $u \in U$, it holds that $\mathcal{T}_{k\text{-}TS(U)} \subseteq TL(u)$, where $\mathcal{T}_{k\text{-}TS(U)}$ denotes the set of timestamps at which U is a k-core.*

Proof. Assume $S = \mathcal{T}_{k\text{-}TS(U)} \supset TL(u)$. For any timestamp $t \in S - TL(u)$, the induced subgraph of U at G_t must be a k-core as per Definition 3.2. However, according to Definition 5.1, $TL(u)$ must contain t, leading to a contradiction. □

Revisiting Example 4.5, given $q = v_2$ and $k = 2$, the vertex set $U' = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ is identified as a LPC with k-core time series $2\text{-}TS(U') = 1001001001$. For a vertex $v_5 \in U'$, we have

$\mathcal{T}_{2\text{-}TS(U')} = \{1, 4, 7, 10\} \subseteq TL(v_5)$. By Theorem 5.2, to ascertain u's inclusion in a LPC, it is adequate to consider only sublists of $TL(u)$ of length at least 2, denoted by $\binom{TL(u)}{\geq 2} = \{sl \subseteq TL(u) || sl| \geq 2\}$. Each sublist $sl \in \binom{TL(u)}{\geq 2}$ corresponds to a time indicator series $\mathcal{S}_{sl}$ with $\mathcal{T}_{\mathcal{S}_{sl}} = sl$. If no such series $\mathcal{S}_{sl}$ contains a maximal periodic k-core time series (one satisfying periodicity constraint), then $\mathcal{T}_{k\text{-}TS(U)} \nsubseteq TL(u)$, indicating that u cannot belong to a LPC. Thus, u can be safely deleted.

Sublist Enumeration. However, directly enumerating all sublists of $TL(u)$ incurs a time complexity of $O(2^{|TL(u)|})$, which is computationally prohibitive for long time lists. To address this, Theorem 5.3 introduces a *structure-aware divide-and-conquer strategy.*

Theorem 5.3. *Given a temporal graph $\mathcal{G}$, a query vertex q, and a parameter k, let τ be a length threshold and L be u's time list with $|L| > \tau$. For any $t \in L$, define: $L^t_{\in} = \{t' \in L \mid \exists U \subseteq \mathcal{V}$ with $q, u \in U$ such that $G_t[U]$ and $G_{t'}[U]$ are k-cores$\}$, and $L^t_{\notin} = L \setminus \{t\}$. Then, the set of sublists of L with size at least l satisfies: $\binom{L}{\geq l} = (\binom{L^t_{\in}}{\geq l-1} \oplus \{t\}) \cup \binom{L^t_{\notin}}{\geq l}$, where $* \oplus \{t\}$ denotes adding t to each sublist in $*$.*

Proof. Sublists of L with size $\geq l$ either contain t or not. If a sublist contains t, the remaining $l-1$ timestamps must be chosen from $L^t_{\in}$, i.e., those t' for which there exists a k-core induced by a common set U at G_t and $G_{t'}$. Otherwise, all l timestamps are selected from $L^t_{\notin}$. The identity follows from this partition. □

Through additional structural constraints, that is, only timestamps t' that share a k-core induced by the same vertex set U with t can be contained in $L^t_{\in}$, Theorem 5.3 prunes unpromising timestamp combinations, significantly reducing enumeration complexity. Moreover, a length threshold τ bounds the recursion depth: if $|L| \leq \tau$, all sublists are enumerated directly; otherwise, Theorem 5.3 is applied recursively to decompose the problem. Each recursion splits L into two sublists, i.e., $L^t_{\in}$ and $L^t_{\notin}$, each of length at most $|L| - 1$, until all sublists adhere to the length constraint.

For instance, in Figure 5(b) with $\tau = 3$ and $l = 2$, consider the time list $L = TL(v_5) = \{1, 4, 7, 10\}$. Since $|L| > \tau$, Theorem 5.3 is employed recursively. Given $t = 1$, we compute $L^1_{\in} = \{4, 7, 10\}$ and $L^1_{\notin} = L \setminus \{1\} = \{4, 7, 10\}$. Both $L^1_{\in}$ and $L^1_{\notin}$ have a size of 3, not exceeding τ. Thus, all sublists of size at least $l - 1 = 1$ within $L^1_{\in}$ and of size at least $l = 2$ within $L^1_{\notin}$ are enumerated directly. This process ensures that all potential sublists of L are generated.

LPC Verification. We now consider another case where a time indicator series $\mathcal{S}_{sl}(sl \in \binom{TL(u)}{\geq 2})$ includes a maximal periodic k-core time series. This suggests that u could be part of a LPC. To identify such a community, we seek a vertex set satisfying Problem 1. To ensure maximality, we calculate the maximal temporal core set $MC(\mathcal{S}_{sl})$ as defined in Definition 4.1. If $MC(\mathcal{S}_{sl})$ is not empty, it is a candidate. However, it is essential to verify that the k-core time series of $MC(\mathcal{S}_{sl})$ exactly equals $\mathcal{S}_{sl}$, rather than including it (as per Definition 4.2), since the latter may not be periodic. To this end, we introduce the concept of parent list set.

Definition 5.4. ***(Parent List Set)*** Given a time list L and an integer l, the parent list set of a sublist $sl \in \binom{L}{\geq l}$ in L, denoted by $\mathcal{P}(sl, L)$, is the set of all sublists $sl' \in \binom{L}{\geq l}$ such that $sl \subset sl'$ and $|sl'| = |sl| + 1$.

For example, in Figure 5(b), for the time list $L = \{1, 4, 7, 10\}$, the parent list set of $sl = \{1, 4, 7\}$ is $\mathcal{P}(sl, L) = \{L\}$. Based on the monotonicity of maximal temporal core sets in Theorem 4.3, we propose a *parent-list-based validation theorem.*

Theorem 5.5. *Given a temporal graph $\mathcal{G}$, a query vertex q, and parameters k, ε, l, let u be a vertex with the shortest time list. For a sublist $sl \in \binom{TL(u)}{\geq l}$, the maximal temporal core set $MC(\mathcal{S}_{sl})$ is a candidate LPC if and only if: (1) $u \in MC(\mathcal{S}_{sl})$, and (2) $\forall pl \in \mathcal{P}(sl, TL(u))$, $MC(\mathcal{S}_{pl}) \subset MC(\mathcal{S}_{sl})$.*

Algorithm 4: Vertex-Deletion-Based Algorithm (VD)

Input: A temporal graph $\mathcal{G}$, a query vertex q, parameters k and ε, thresholds Θ_{99} and τ
Output: Local periodic community (LPC)

$\mathcal{R} \leftarrow \emptyset$; $\mathcal{F}_{max} \leftarrow 1$; $Candi \leftarrow \emptyset$;
initialize time list $TL(u)$ for all vertices $u \in \mathcal{V}$;
while $\mathcal{V} \neq \varnothing$ **do**
  $u \leftarrow \arg\min_{v \in \mathcal{V}} |TL(v)|$; $\mathcal{C} \leftarrow \emptyset$;
  recursive apply Theorem 5.3 to generate $\binom{TL(u)}{\geq(\mathcal{F}_{max}+1)}$;
  for $\forall sl \in \binom{TL(u)}{\geq(\mathcal{F}_{max}+1)}$ **do**
    compute $\mathcal{PS}(\mathcal{S}_{sl})$ and determine the periodic pattern frequency $\mathcal{F}$ of $\mathcal{S}_{sl}$;
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{(sl, \mathcal{F})\}$;
  for $\forall (sl, \mathcal{F}) \in \mathcal{C}$ **do**
    if $\mathcal{F} \leq \mathcal{F}_{max}$ **then**
      remove sl from $Candi$ **if** $sl \in Candi$;
      continue;
    $Com \leftarrow Compute(\mathcal{V}, \mathcal{S}_{sl}, q, k)$; // Algorithm 2
    if $Com = \emptyset$ **then**
      continue;
    $isflag \leftarrow$ **true**;
    for $\forall pl \in \mathcal{P}(sl, TL(u))$ ***and*** $pl \notin Candi[sl]$ **do**
      $Pcom \leftarrow Compute(Com, \mathcal{S}_{pl}, q, k)$;
      if $Com = Pcom$ **then**
        $isflag \leftarrow$ **false**;
        break;
    if $isflag$ **then**
      if $u \in Com$ **then**
        $\mathcal{R} \leftarrow Com$; $\mathcal{F}_{max} \leftarrow \mathcal{F}$;
      else
        $Candi[sl] \leftarrow \mathcal{P}(sl, TL(u))$;
    else
      remove sl from $Candi$ **if** $sl \in Candi$;
  delete u from $\mathcal{V}$;
  update $TL(v)$ for each remaining vertex $v \in \mathcal{V}$;
return $\mathcal{R}$.

Proof. Condition (1) ensures u's inclusion in $MC(\mathcal{S}_{sl})$. For Condition (2), $MC(\mathcal{S}_{pl}) \subseteq MC(\mathcal{S}_{sl})$ (Theorem 4.3). If equality holds, the k-core time series S of $MC(\mathcal{S}_{sl})$ must satisfy $sl \subset \mathcal{T}_S$; however, the latter may lack periodicity. Together, these conditions guarantee that the k-core time series of $MC(\mathcal{S}_{sl})$ is exactly $\mathcal{S}_{sl}$. Given the periodicity of $\mathcal{S}_{sl}$, $MC(\mathcal{S}_{sl})$ qualifies as a candidate LPC. □

Theorem 5.5 is to confirm that the k-core time series of the maximal temporal core set $MC(\mathcal{S}_{sl})$ equals $\mathcal{S}_{sl}$. Once validated, $MC(\mathcal{S}_{sl})$ is a candidate LPC containing u. For example, in Figure 5, consider the vertex v_5 with query parameters $q = v_2$, $k = 2$, and $l = 2$. For the sublist $sl = \{1, 4, 7, 10\} \in \binom{TL(v_5)}{\geq 2}$, its 2-core time series is $\mathcal{S}_{sl} = 1001001001$, which contains a maximal periodic 2-core time series. The maximal temporal core set is $MC(\mathcal{S}_{sl}) = \{v_1, \cdots, v_5, v_6\}$. Since $v_5 \in MC(\mathcal{S}_{sl})$ and $\mathcal{P}(sl, TL(v_5)) = \emptyset$, Theorem 5.5 holds, confirming $MC(\mathcal{S}_{sl})$ as a candidate LPC.

After all sublists of u's time list that include a maximal periodic k-core time series are processed, u is removed, and the time lists of the remaining vertices are updated. By iterative processing of the vertex with the shortest time list, the algorithm ultimately identifies the LPC with the highest periodic pattern frequency.

Building on the prior analysis, we introduce the basic vertex-deletion-based algorithm VD for LPC search. Algorithm 4 outlines the pseudo-code of VD. Firstly, it initializes the time list $TL(u)$ for each vertex $u \in \mathcal{V}$ (line 2). Then, the algorithm selects the vertex u with shortest $|TL(u)|$, and recursively employs Theorem 5.3 to enumerate all sublists $sl \in \binom{TL(u)}{\geq(\mathcal{F}_{max}+1)}$ along with their periodic pattern frequency $\mathcal{F}$ (lines 4-8). For each sl, if $\mathcal{F} \leq \mathcal{F}_{max}$, it is skipped (lines 9-12); otherwise, Theorem 5.5 is used to identify a candidate LPC containing u (lines 13-28). Specifically, if $MC(\mathcal{S}_{sl}) = \emptyset$, Theorem 5.5 is not satisfied (lines 13-15); otherwise, it validates all parent lists pl of sl to ensure $MC(\mathcal{S}_{pl}) \subset MC(\mathcal{S}_{sl})$ (lines 16-21). If validated and $u \in MC(\mathcal{S}_{sl})$, a candidate LPC is confirmed and the highest frequency $\mathcal{F}_{max}$ is updated to $\mathcal{F}$ (lines 22-24). If validated but $u \notin MC(\mathcal{S}_{sl})$, sl is marked as a possible list, and all processed parent lists are added to *Candi*, reducing computational overhead in subsequent steps (lines 17 and 26). If validation fails, it indicates that the k-core time series of $MC(\mathcal{S}_{sl})$ includes $\mathcal{S}_{sl}$ but may not be periodic. (lines 27-28). Finally, u is removed and the remaining time lists are updated (lines 29-30). VD terminates when all vertices are processed, returning a LPC.

Example 5.6. Using Figures 2 and 5, we demonstrate Algorithm 4 with parameters $q = v_2$, $k = 2$, and $\varepsilon = 0.8$. Initial time lists are shown in Figure 5(a) (in black font), with $\mathcal{F}_{max}$ set to 1. The vertex v_5 with the shortest time list $TL(v_5)$ is selected, and all its sublists of length at least 2 are generated. In Figure 5(b), sublists with periodic pattern frequency greater than $\mathcal{F}_{max}$ are enclosed by orange dotted lines. For $sl = \{1, 4, 7, 10\}$, $\mathcal{S}_{sl}$ has a maximal periodic k-core time series 100100100 with frequency 3. According to Theorem 5.5, $MC(\mathcal{S}_{sl}) = \{v_1, \ldots, v_6\}$ is a candidate LPC. Thus, $\mathcal{F}_{max}$ is updated to 3. Remaining sublists are skipped as their periodic pattern frequency does not exceed 3. Then, v_5 is removed and $TL(v_6)$ is updated (in red font). This process iterates over all vertices. As no maximal periodic k-core time series with higher frequency is found, the set $\{v_1, \ldots, v_6\}$ is identified as the final LPC and returned.

VD Complexity Analysis. To efficiently initialize the time list $TL(u)$ for each vertex $u \in \mathcal{V}$, we precompute the coreness of u at each timestamp t as $c_t(u)$. Let $\mathcal{T}_q = \{t \in [1, T] \mid c_t(q) \geq k\}$ denote the set of timestamps at which the query vertex q resides in a k-core. Define $L_t = max_{u \in \mathcal{V}} |TL(u)|$ and $E_m = max_{t \in [1,T]} |E_t|$. For each $t \in \mathcal{T}_q$, we perform a BFS traversal starting from q to identify a connected k-core, requiring $O(|\mathcal{V}| + E_m)$ time per snapshot. The total initialization cost is $O(|\mathcal{T}_q| \cdot (|\mathcal{V}| + E_m))$. Then, a minimum priority queue is utilized to maintain all vertices, with the length of their time lists as the key. It takes $O(|\mathcal{V}| \cdot \log |\mathcal{V}|)$ time. Next, the vertex u with the shortest $|TL(u)|$ is processed in sequence, which involves three steps. (1) Using Theorem 5.3, all sublists $sl \in \binom{TL(u)}{\geq 2}$ are enumerated, which takes at most $O(2^{|TL(u)|})$ time. (2) For each sublist sl, computing the maximal periodic k-core time series takes $O(T \cdot \log T)$ time. (3) Theorem 5.5 is used to verify a candidate LPC containing u, which takes $O((|TL(u)| - |sl|) \cdot |TL(u)| \cdot (|\mathcal{V}| + E_m))$ time. The total processing time for a single vertex u is $O(2^{|TL(u)|} \cdot (T \cdot \log T + |TL(u)|^2 \cdot (|\mathcal{V}| + E_m)))$. Thus, the overall time complexity of VD is $O((|\mathcal{T}_q| + 2^{L_t} \cdot L_t^2) \cdot (|\mathcal{V}| + E_m) + |\mathcal{V}| \cdot \log |\mathcal{V}| + 2^{L_t} \cdot T \cdot \log T)$.

TD VS. VD. We now provide a detailed comparison of the time complexities of TD and VD. As analyzed in Section 4.1, the runtime of TD is dominated by $N \cdot L_s^2$, where N is the number of maximal temporal core sets and L_s is the maximum number of active timestamps in any k-core time series. Thus, TD is efficient on sparse graphs where N and L_s are small but scales poorly on dense graphs as N grows rapidly and $L_s \approx T$. In contrast, VD addresses this by dynamically pruning short time lists, ensuring that the maximum processed time list length L_t remains much smaller than T, i.e., $L_t \ll T$. The time complexity of VD primarily depends on $2^{L_t} \cdot L_t^2$. Despite including an exponential term 2^{L_t}, it is effectively mitigated in practice by the structure-aware pruning strategy in Theorem 5.3. As a result, VD achieves optimal overall performance.

5.2 Optimizations

To enhance the efficiency of Algorithm 4, we propose two optimizations: index-based pruning and batch deletion.

OPT-1: TP-Core Index-based Pruning. Theorem 5.3 utilizes a divide-and-conquer strategy to partition each time list TL into sublists of length τ or less, thereby reducing enumeration complexity. Despite this, it may still generate substantial unpromising sublists sl, i.e., $MC(\mathcal{S}_{sl}) = \emptyset$. To prune such candidates early, we leverage the proposed TP-Core index Φ_c, which records the time-pair coreness $c(u, \mathbb{T})$ of each vertex $u \in \mathcal{V}$ at all time pairs $\mathbb{T} = \{t, t'\}$ (Definition 4.6). Given a query vertex q and a parameter k, if $c(q, \mathbb{T}) < k$, then no common k-core containing q exists simultaneously at t and t'; thus, any sublist containing both timestamps is invalid and can be safely discarded. Based on this, Theorem 5.7 enables more efficient, structure-aware enumeration using Φ_c.

Theorem 5.7. *Given a temporal graph $\mathcal{G}$, a query vertex q, and parameters k, l, let τ be a length threshold and L be a time list with $|L| \leq \tau$. For each $t \in L$, its candidate set is defined as $L^t_\in = \{t' \in L | c(q, \{t, t'\}) \geq k\}$. Then, the set of all valid sublists of L with size at least l is given by: $\binom{L}{\geq l} = \bigcup_{t \in L} (\binom{L^t_\in}{\geq l-1} \oplus \{t\})$, where $* \oplus \{t\}$ denotes adding t to each sublist in $*$.*

Proof. For any timestamp $t' \notin L^t_\in$, since $c(q, \{t, t'\}) < k$, it holds that $MV(\mathcal{S}_{\{t,t'\}}) = \emptyset$. Thus, among all combinations containing t, only sublists of $L^t_\in$ of size at least $l-1$, augmented with t, can form valid sublists, i.e., $\{sl \cup \{t\} | sl \in \binom{L^t_\in}{\geq l-1}\}$. By aggregating over all timestamps $t \in L$, the identity holds. □

Example 5.8. Let $\tau = 5$. In Example 5.6, after deleting v_5 and v_6, $\mathcal{F}_{max}$ is updated to 3. The vertex v_7, with the shortest time list $TL = \{2, 4, 8, 9, 10\}$, is selected. In the basic VD algorithm, all sublists of TL with length greater than $\mathcal{F}_{max}$ should be enumerated and verified. In contrast, according to Theorem 5.7, we can compute the candidate set $TL^2_\in = \{4, 8, 10\}$ for $t = 2$ using Φ_c, and generate sublists containing $t = 2$ via $(\binom{TL^2_\in}{\geq 3} \oplus \{2\})$. The sublists involving $\{2, 9\}$ can be pruned effectively.

OPT-2: Batch Deletion. The basic VD Algorithm sequentially handles vertices based on their time list lengths in ascending order. When multiple vertices share identical shortest time list (i.e., $TL(u) = TL(v)$), redundant computations occur due to duplicate sublist enumeration (Theorem 5.3) and repeated parent-list-based verification (Theorem 5.5). To address this issue, a batch deletion strategy is introduced into VD. Specifically, vertices with the same shortest TL are grouped into a batch B and processed collectively. This approach ensures that candidate sublists $sl \in \binom{TL}{\geq 2}$ are enumerated once. For each sl, Theorem 5.5 verifies if there exists a vertex $u \in B$ such that $u \in MC(\mathcal{S}_{sl})$ and $k\text{-}TS(MC(\mathcal{S}_{sl})) = \mathcal{S}_{sl}$. Then, all vertices in B are deleted simultaneously.

For example, in Figure 5, vertices $B_1 = \{v_5, v_6\}$ are processed together, followed by $B_2 = \{v_7, v_8\}$, $B_3 = \{v_4\}$, and $B_4 = \{v_1, v_2, v_3\}$. These eight vertices are divided into four batches for processing, thereby improving the algorithm's efficiency.

6 Experimental Evaluation

This section evaluates the performance of our proposed algorithms. All algorithms are implemented in C++, and compiled by G++ with -O3 optimization. The experiments are conducted on CentOS Linux 7 Core with Intel Xeon 2.40GHz and 125G main memory.

6.1 Experimental Settings

Datasets: We employ five real-world temporal graphs in experiments, which are summarized in Table 1. Specifically, LKML and Enron are email interaction networks, Flickr is a social network, and

Table 1. Detailed Descriptions of Temporal Graphs

Dataset (Abbr.)	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	$\|T\|$	Time scale
LKML (LK)	27,926	585,506	2,922	day
Enron (EN)	87,273	925,171	1,288	day
IMDB (IM)	815,028	18,964,173	135	Year
Flickr (FL)	2,302,925	33,140,017	30	week
Wikipedia (WK)	6,935,516	129,885,939	198	month

Wikipedia is a collaborative network. These datasets are obtained from http://konect.cc/. Moreover, IMDB[2] is constructed from the Internet Movie Database downloaded in March 2025.

Algorithms: We evaluate the following algorithms in experiments.

- **PSubG-0 and PSubG-10**: The baselines for periodic subgraph mining utilize periodicity determination models based on arithmetic sequences [32] and quasi-arithmetic sequences [51], respectively, with a minimum sequence length $\sigma \geq 3$. To efficiently identify the periodic subgraph containing the query vertex q with the maximum valid σ following each model, we perform a binary search over σ, invoking their corresponding mining algorithms at each iteration. The search range is $[3, |TL(q)|]$, where $TL(q)$ denotes the initial time list of q (Definition 5.1). For brevity, these two methods are collectively labeled as PSubG-$*$.
- **LTD**: Our time-deletion-based algorithm for Problem 1.
- **LVD**: Our vertex-deletion-based algorithm for Problem 1.
- **Optimizations**. **i-**: optimization based on TP-Core index. **b-**: batch deletion optimization. **ib-**: combination of both optimizations.

Metrics: In the evaluation of *community quality*, we adopt established metrics from prior research [31, 33, 34], including Temporal Separability (TS) and Temporal Conductance (TC). For a community C, let $\mathcal{T}_C$ denote the set of timestamps when C occurs.

$$\mathrm{TS}(C) = \frac{1}{|\mathcal{T}_C|} \sum_{t \in \mathcal{T}_C} \frac{|\{(u,v,t) \in \mathcal{E} | u, v \in C\}| / |C|}{|S_t = \{(u,v,t) \in \mathcal{E} : u \in C, v \notin C\}| / |S_t|} \tag{1}$$

$$\mathrm{TC}(C) = \frac{|\{(u,v,t) \in \mathcal{E} | u \in C, v \notin C, t \in \mathcal{T}_C\}|}{min\{|\{(u,v,t) \in \mathcal{E} | u \in C, t \in \mathcal{T}_C\}|, |\{(u,v,t) \in \mathcal{E} | u, v \notin C, t \in \mathcal{T}_C\}|\}} \tag{2}$$

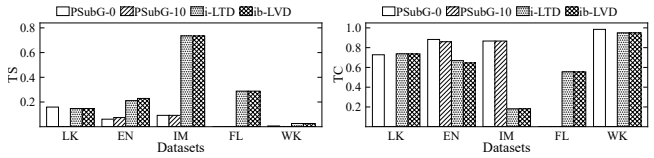
Specifically, TS measures the ratio between the average internal density and average external density. Higher TS indicates stronger internal cohesion and weaker external connectivity. TC evaluates the ratio of cross-community edges to the smaller of: (1) edges involving C, and (2) edges outside C. Lower TC indicates better structural separation. Therefore, a high-quality community exhibits high TS and low TC.

In the evaluation of *query efficiency*, we report the average time (in seconds) over 500 queries per dataset. If an algorithm cannot complete within two hours, its running time is recorded as INF. In our periodic community search problem, key parameters are k and ε, with default values $k = 4$ and $\varepsilon = 20\%$ for most datasets. For vertex-deletion-based algorithms (e.g., LVD), the default value of τ is 15. Due to space limitations and similar trends over different temporal graphs, we report experimental results of partial datasets in some experiments.

6.2 Effectiveness Evaluation

Exp-1: PSubG-$*$ VS. Our proposed algorithms. For Problem 1, we compare the baselines PSubG-0 and PSubG-10 against our algorithms i-LTD and ib-LVD. Figure 6 presents community qualities

[2]https://developer.imdb.com/non-commercial-datasets/



(a) Temporal Separability (b) Temporal Conductance

Fig. 6. Quality Evaluation of LPC

Table 2. Time (sec) of Different Algorithms for LPC Search

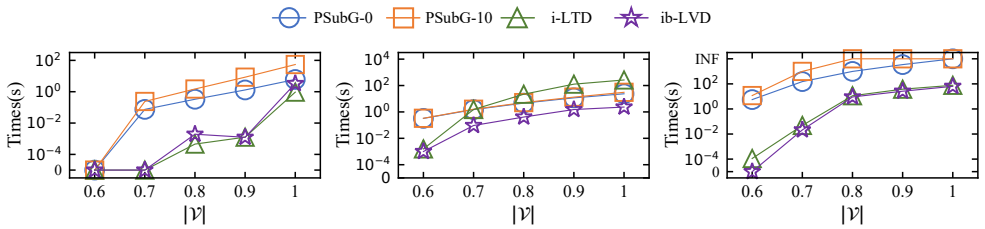
Dataset / Algorithm	LK	EN	IM	FL	WK
PSubG-0	5.352	5.712	814.846	23.168	INF
PSubG-10	INF	56.202	2,066.2	27.720	INF
i-LTD	**0.00041**	**0.940**	103.544	263.781	77.124
ib-LVD	0.00094	3.332	**7.696**	**2.341**	**44.008**

under default parameters, showing consistent trends across settings. A temporal separability (TS) or temporal conductance (TC) value of 0 indicates that PSubG-$*$ fails to discover any periodic community, e.g., on LK, FL, and WK. As seen in Figures 6(a) and 6(b), i-LTD and ib-LVD consistently achieve higher TS and lower TC on all datasets except LK. This advantage stems from a fundamental modeling difference: PSubG-$*$ requires identical subgraph structures at all active timestamps, leading to numerous cross-community edges and high external density. In contrast, our local periodic community (LPC) allows structurally varying k-cores at different timestamps, yielding stronger internal connectivity and clearer boundaries. For example, on IM, the identified LPC corresponds to a closely interconnected director group that collaborates periodically with stable membership but flexible role assignments. This yields high-quality communities (TS = 0.735 and TC = 0.181), whereas PSubG-$*$ produces markedly inferior results (TS = 0.092 and TC = 0.867).

Notably, i-LTD and ib-LVD may have different TS and TC values, as observed on EN. This discrepancy arises from the existence of multiple local periodic communities with distinct maximal periodic k-core time series, leading to non-identical outputs.

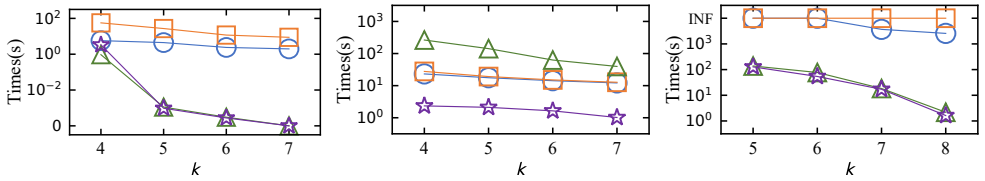
6.3 Efficiency Evaluation

Exp-2: PSubG-$*$ VS. Our proposed algorithms. Firstly, we compare four algorithms for LPC search: PSubG-0, PSubG-10, i-LTD and ib-LVD. As shown in Table 2, i-LTD outperforms PSubG-$*$ on all datasets except FL, achieving speedups of up to four orders of magnitude. ib-LVD demonstrates significant improvements, with speedup ratios ranging from 1.714 to 5,693.6 over PSubG-$*$. This performance gap stems from fundamental differences in algorithmic design. PSubG-$*$ identifies (quasi-)periodic communities by enumerating all candidate (quasi-)arithmetic sequences. When T is large or the graph is dense, the query vertex's time list becomes long, leading to increased search space and degraded performance, particularly on datasets like IM and WK. In contrast, i-LTD avoids explicit enumeration by calculating maximal temporal core sets with varying k-core time series, with complexity dependent on graph density. Thus, it is efficient on sparse graphs (e.g., LK and EN), but scales poorly on dense graphs such as FL. ib-LVD addresses this limitation by prioritizing vertices with the shortest time lists, which are dynamically shortened through pruning. Therefore, ib-LVD achieves superior overall performance, excelling on dense graphs while maintaining efficiency on sparse graphs.



(b) EN (c) FL (d) WK

Fig. 7. Scalability Evaluation



(a) EN (b) FL (c) WK

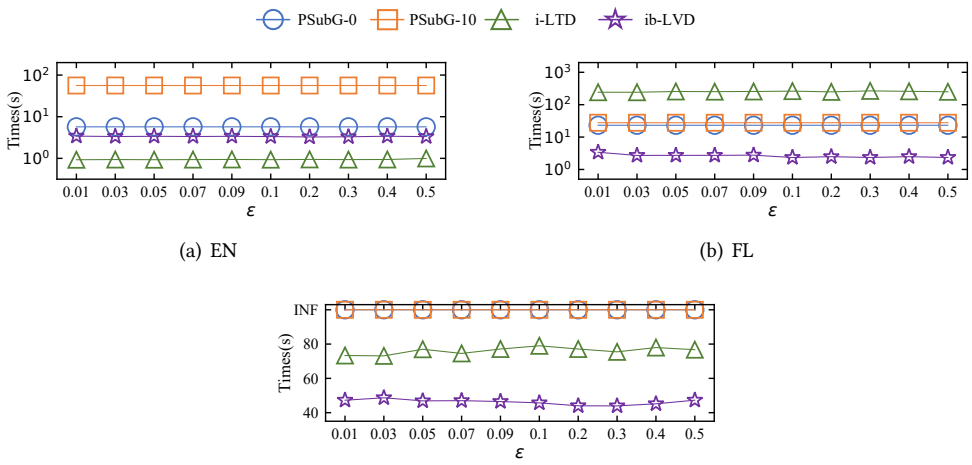
Fig. 8. Effect of k

Exp-3: Scalability Evaluation. To evaluate scalability, we sample 20%, 40%, 60%, 80%, and 100% of the vertices from the original graphs, and extract the corresponding induced subgraphs. The results in Figure 7 reveal that as the graph size increases, the running time of all algorithms tends to grow. This trend arises from larger graphs entail more maximum temporal core sets, leading to a linear rise in i-LTD's runtime, while ib-LVD incurs higher costs due to processing more vertices. Despite this, i-LTD and ib-LVD exhibit performance superiority over PSubG-$*$ at most datasets.

Exp-4: Effect of k. We investigate the effect of the parameter k by varying it from 4 to 7 for most datasets. Figure 8 plots the running time of the four algorithms. As k increases, the runtime diminishes for all algorithms. The reason behind this is that higher k values impose stricter structural constraints, reducing the number of qualifying vertices and shrinking the search space. However, the performance of i-LTD and ib-LVD is still much better than that of other algorithms.

Exp-5: Effect of ε. We next examine the effect of the threshold ε, which specifies the minimum subseries similarity required for a maximal periodic k-core time series as per Definition 3.4, and thereby determines the resulting periodic community. To evaluate robustness under diverse similarity criteria, we vary ε from 0.01 (lenient) to 0.5 (strict). The results are shown in Figure 9. When ε increases, i-LTD and ib-LVD exhibit very small time fluctuations, while PsubG-$*$ remains unchanged, as it does not involve ε. This stability stems from two factors. (i) For $\varepsilon \in [0.01, 0.1]$, the quality metrics (TS and TC) of their identified communities remain unchanged, confirming that the measured subseries similarity usually surpasses 0.1. (ii) Their core computational steps are independent of ε, such as maximal temporal core set generation, sublist enumeration, and k-core computation. The threshold is applied only during final LPC validation. Thus, our approaches remain both effective and efficient across all ε values, including very small ones.

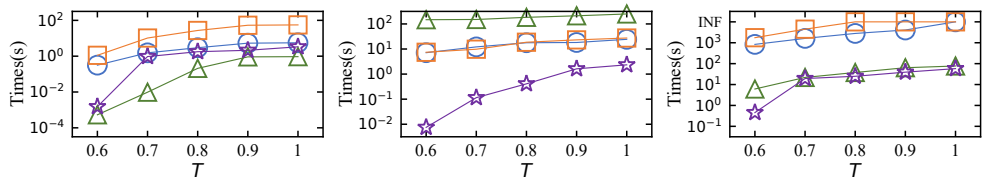
Exp-6: Effect of T. To assess the impact of the number of timestamps, we retain varying percentages of timestamps (i.e., 60%, 70%, 80%, 90%, and 100%) from the original graphs. Figure 10 illustrates that running time increases for all algorithms as T grows. A larger T leads to more timestamps processed in i-LTD, resulting in a linear rise in its execution time. ib-LVD faces a higher volume of candidate vertices, and the average length of the shortest time lists also lengthens, raising computational cost. Notably, i-LTD and ib-LVD consistently outperform PSubG-$*$ on the majority of datasets.



(a) EN (b) FL

(c) WK

Fig. 9. Effect of ε

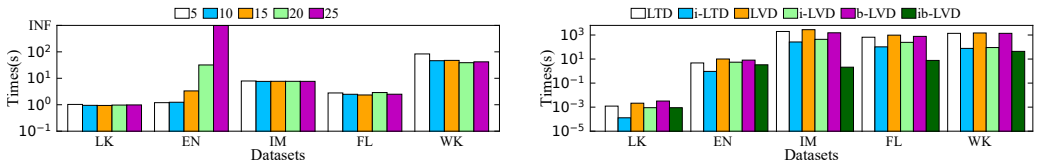


(a) EN (b) FL (c) WK

Fig. 10. Effect of T

Exp-7: Effect of τ. Recall that our vertex-deletion-based algorithm introduces a threshold τ to bound sublist enumeration complexity. We study its impact on ib-LVD by setting $\tau \in \{5, 10, 15, 20, 25\}$. Results are shown in Figure 11(a). With increasing τ, the execution time of ib-LVD exhibits varying trends across datasets. On EN, the exponential growth in runtime stems from the increasing number of sublists to enumerate. Conversely, on datasets characterized by shorter time lists, enumeration complexity is primarily determined by the list length rather than τ, resulting in relatively stable runtimes. These results indicate that $\tau = 15$ achieves a favorable trade-off between efficiency and scalability.

Exp-8: Our proposed Optimizations. In this experiment, we evaluate the effectiveness of two optimizations: i- and b-. The i- optimization is added into the basic LTD algorithm, labeled as i-LTD; while both optimizations are integrated into the basic LVD algorithm, denoted by ib-LVD. The results are summarized in Figure 11(b). We can observe that the two optimizations have different performance improvements. Specifically, i- significantly enhances both LTD and LVD across all datasets. For example, i-LTD and i-LVD are 8.783× and 6.173× faster than their respective baselines. This gain is attributed to the TP-Core index Φ_c, which efficiently prunes time series lacking a common k-core, reducing unnecessary computations. Furthermore, b- also improves LVD, particularly on dense graphs, achieving a speedup of 1.815 on FL, though it shows limited benefit on LK. When combined, the two optimizations together make the basic LVD algorithm the most efficient. For example, ib-LVD achieves a maximum speedup of 1,317.4 on FL, with an average speedup of 296.7.



(a) Effect of τ (b) Evaluation of Optimizations

Fig. 11. Additional Evaluations

Table 3. TP-Core Index Size and Construction Time

Dataset	LK	EN	IM	FL	WK
Index Size (MB)	314.7	434.9	467.3	327.7	5,408.5
Time (hour)	0.393	0.058	4.394	0.591	1.773

Exp-9: TP-Core index construction. Finally, we report the size and construction time of the TP-Core index Φ_c in Table 3. It is evident that the index size of Φ_c increases with the temporal graph size, depending on the number of vertices and timestamps. For example, Φ_c has a minimum size of 314.7 MB on LK, and a maximum of 5,408.5 MB on WK. However, the construction time of Φ_c does not strictly correlate with graph size. For instance, IM requires the longest construction time (4.3 hours) due to its high average edge density per time pair.

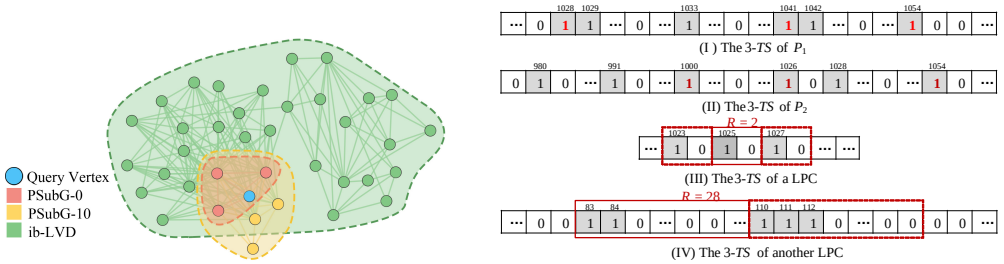
Discussion. Our experimental results reveal three key insights: (1) i-LTD is efficient on sparse temporal graphs but slows down on larger or denser networks due to the exhaustive enumeration of maximal temporal core sets. (2) ib-LVD maintains consistent performance across various graphs, attributed to its dynamic pruning of time lists. (3) Therefore, ib-LVD stands out as the optimal choice for real-world large-scale temporal graphs, particularly when responsiveness and scalability are critical.

6.4 Case Study

We conduct a case study on EN under identical parameters ($k = 3$, $\varepsilon = 0.2$, and $q = 2607$) to compare different algorithms for LPC search (Problem 1). Specifically, we evaluate PSubG-0, PSubG-10 and ib-LVD. Figure 12 presents the identified communities and their corresponding k-core time series (k-TS), highlighting only subseries containing the value 1 for clarity. Figures 12(b-I) and 12(b-II) depict the 3-TS of the periodic subgraphs P_1 and P_2, detected by PSubG-0 and PSubG-10 (highlighted in pink and yellow dashed lines in Figure 12(a)). While certain timestamps form (quasi-)arithmetic sequences of length $\sigma \geq 3$ (in red), the presence of many non-periodic timestamps suggests irrational periodicity determination. In contrast, ib-LVD identifies a LPC (in green dashed line) with a period $R = 2$ and a maximal periodic 3-core time series $S = 101010$ in Figure 12(b-III), achieving the highest periodic pattern frequency $\mathcal{F}(S) = 3$. Clearly, this result demonstrates superior temporal regularity and interpretability compared to the baselines.

Furthermore, Figure 12(b-IV) presents another LPC found by ib-LVD with a long period $R = 28$, where each occurrence spans multiple consecutive timestamps (e.g., persisting at $t = 83$ for two timestamps and at $t = 110$ for three timestamps). Such complex periodic patterns, undetectable by arithmetic sequence-based methods, are effectively modeled and identified by our k-core time series-based periodicity determination approach.

Practical implications. In email interaction networks like EN, the returned periodic communities reveal stable communication patterns within research teams or departments. These patterns facilitate cross-departmental collaboration and aid in detecting anomalies, e.g., stalled projects, in research and development workflows.



(a) Identified Periodic Communities (b) The k-core Time Series (k-TS) of Periodic Communities

Fig. 12. Case Study

7 Conclusion

This paper investigates periodic community search in temporal graphs. We propose a k-core time series-based method for period determination and define the local periodic community problem. To solve it, we develop time-deletion-based algorithms and vertex-deletion-based algorithms. Extensive experiments demonstrate the effectiveness and efficiency of our proposed model and algorithms.

Acknowledgments

This work was supported in part by the NSFC under Grants No. (62025206, U23A20296, and 62302444), and Zhejiang Province's "Lingyan" R&D Project under Grant No. 2024C01259. Yunjun Gao is the corresponding author of the work.

References

[1] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science*. 475–486.
[2] Md Musfique Anwar. 2020. Query-oriented temporal active intimate community search. In *Australasian Database Conference*. Springer, 206–215.
[3] Jennifer K Bizley, Kerry MM Walker, Andrew J King, and Jan WH Schnupp. 2010. Neural ensemble codes for stimulus periodicity in auditory cortex. *Journal of Neuroscience* 30, 14 (2010), 5078–5091.
[4] Simon R Blackburn. 2002. A generalisation of the discrete Fourier transform: Determining the minimal polynomial of a periodic sequence. *IEEE Transactions on Information Theory* 40, 5 (2002), 1702–1704.
[5] Udi Boker. 2024. Discounted-sum automata with real-valued discount factors. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science*. 15:1–15:14.
[6] Weibin Cai, Fanwei Zhu, Zemin Liu, and Minghui Wu. 2023. HeteroCS: A heterogeneous community search system with semantic explanation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 3155–3159.
[7] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yiqi Lu, and Wei Wang. 2013. Online search of overlapping communities. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 277–288.
[8] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. 991–1002.
[9] Yizhou Dai, Miao Qiao, and Rong-Hua Li. 2024. On density-based local community search. *Proceedings of the ACM on Management of Data* 2, 2 (2024), 1–25.
[10] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *The VLDB Journal* 29, 1 (2020), 353–392.
[11] Sajal Halder, Md Samiullah, and Young-Koo Lee. 2017. Supergraph based periodic pattern mining in dynamic social networks. *Expert Systems with Applications* 72 (2017), 430–442.
[12] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. 1311–1322.
[13] Xin Huang, Laks VS Lakshmanan, and Jianliang Xu. 2017. Community search over big graphs: Models, algorithms, and opportunities. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. 1451–1454.

[14] Zongli Jiang, Yirui Tan, Guoxin Chen, Fangda Guo, Jinli Zhang, and Xiaolu Bai. 2024. GNN-based persistent k-core community search in temporal graphs. In *2024 IEEE International Conference on Big Data (BigData)*. 569–578.
[15] Tanvi Jindal, Prasanna Giridhar, Lu-An Tang, Jun Li, and Jiawei Han. 2013. Spatiotemporal periodical pattern mining in traffic data. In *Proceedings of the 2nd ACM SIGKDD International Workshop on Urban Computing*. 1–8.
[16] Mayank Lahiri and Tanya Y Berger-Wolf. 2008. Mining periodic behavior in dynamic social networks. In *2008 Eighth IEEE International Conference on Data Mining*. 373–382.
[17] Mayank Lahiri and Tanya Y Berger-Wolf. 2010. Periodic subgraph mining in dynamic networks. *Knowledge and Information Systems* 24, 3 (2010), 467–497.
[18] Ling Li, Yuhai Zhao, Yuan Li, Fazal Wahab, and Zhengkui Wang. 2022. The most active community search in large temporal graphs. *Knowledge-Based Systems* 250 (2022), 109101.
[19] Mo Li, Zhiran Xie, and Linlin Ding. 2023. Persistent community search over temporal bipartite graphs. In *International Conference on Advanced Data Mining and Applications*. Springer, 324–339.
[20] Rong-Hua Li, Jiao Su, Lu Qin, Jeffrey Xu Yu, and Qiangqiang Dai. 2018. Persistent community search in temporal networks. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 797–808.
[21] Yuan Li, Xiuxu Chen, Yuhai Zhao, Wen Shan, Zhengkui Wang, Guoli Yang, and Guoren Wang. 2024. Self-training gnn-based community search in large attributed heterogeneous information networks. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 2765–2778.
[22] Zhenhui Li, Bolin Ding, Jiawei Han, Roland Kays, and Peter Nye. 2010. Mining periodic behaviors for moving objects. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1099–1108.
[23] Xuankun Liao, Qing Liu, Xin Huang, and Jianliang Xu. 2024. Truss-based community search over streaming directed graphs. *Proceedings of the VLDB Endowment* 17, 8 (2024), 1816–1829.
[24] Longlong Lin, Ronghua Li, and Tao Jia. 2023. Scalable and effective conductance-based graph clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 4471–4478.
[25] Longlong Lin, Pingpeng Yuan, Rong-Hua Li, Chunxue Zhu, Hongchao Qin, Hai Jin, and Tao Jia. 2024. QTCS: efficient query-centered temporal community search. *Proceedings of the VLDB Endowment* 17, 6 (2024), 1187–1199.
[26] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. Truss-based community search over large directed graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2183–2197.
[27] Qing Liu, Yifan Zhu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. VAC: vertex-centric attributed community search. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 937–948.
[28] Hans Dieter Luke, Hans D Schotten, and Hafez Hadinejad-Mahram. 2004. Binary and quadriphase sequences with optimal autocorrelation properties: A survey. *IEEE Transactions on Information Theory* 49, 12 (2004), 3271–3282.
[29] Chengyang Luo, Qing Liu, Yunjun Gao, and Jianliang Xu. 2025. Synergetic community search over large multilayer graphs. *Proceedings of the VLDB Endowment* 18, 5 (2025), 1412–1424.
[30] Sanghyun Park, Sang-Wook Kim, June-Suh Cho, and Sriram Padmanabhan. 2001. Prefix-querying: An approach for effective subsequence matching under time warping in sequence databases. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*. 255–262.
[31] Hongchao Qin, Rong-Hua Li, Guoren Wang, Xin Huang, Ye Yuan, and Jeffrey Xu Yu. 2020. Mining stable communities in temporal networks by density-based clustering. *IEEE Transactions on Big Data* 8, 3 (2020), 671–684.
[32] Hongchao Qin, Rong-Hua Li, Guoren Wang, Lu Qin, Yurong Cheng, and Ye Yuan. 2019. Mining periodic cliques in temporal networks. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 1130–1141.
[33] Hongchao Qin, Rong-Hua Li, Ye Yuan, Yongheng Dai, and Guoren Wang. 2023. Densest periodic subgraph mining on large temporal graphs. *IEEE Transactions on Knowledge and Data Engineering* 35, 11 (2023), 11259–11273.
[34] Hongchao Qin, Rong-Hua Li, Ye Yuan, Guoren Wang, Lu Qin, and Zhiwei Zhang. 2022. Mining bursting core in large temporal graphs. *Proceedings of the VLDB Endowment* 15, 13 (2022), 3911–3923.
[35] Hongchao Qin, Rong-Hua Li, Ye Yuan, Guoren Wang, Weihua Yang, and Lu Qin. 2020. Periodic communities mining in temporal networks: Concepts and algorithms. *IEEE Transactions on Knowledge and Data Engineering* 34, 8 (2020), 3927–3945.
[36] Natali Ruchansky, Francesco Bonchi, David García-Soriano, Francesco Gullo, and Nicolas Kourtellis. 2015. The minimum wiener connector problem. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 1587–1602.
[37] Natali Ruchansky, Francesco Bonchi, David García-Soriano, Francesco Gullo, and Nicolas Kourtellis. 2017. To be connected, or not to be connected: That is the minimum inefficiency subgraph problem. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 879–888.
[38] Gabriella Rustici, Juan Mata, Katja Kivinen, Pietro Lió, Christopher J Penkett, Gavin Burns, Jacqueline Hayles, Alvis Brazma, Paul Nurse, and Jürg Bähler. 2004. Periodic gene expression program of the fission yeast cell cycle. *Nature Genetics* 36, 8 (2004), 809–817.

[39] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 939–948.
[40] Longxu Sun, Xin Huang, Zheng Wu, and Jianliang Xu. 2024. Efficient cross-layer community search in large multilayer graphs. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 2959–2971.
[41] Hanghang Tong and Christos Faloutsos. 2006. Center-piece subgraphs: problem definition and fast solutions. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 404–413.
[42] Michail Vlachos, Philip Yu, and Vittorio Castelli. 2005. On periodicity detection and structural periodic similarity. In *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, 449–460.
[43] Yue Wang, Xun Jian, Zhenhua Yang, and Jia Li. 2017. Query optimal k-plex based community in graphs. *Data Science and Engineering* 2, 4 (2017), 257–273.
[44] Yuxiang Wang, Zhangyang Peng, Xiangyu Ke, Xiaoliang Xu, Tianxing Wu, and Yuan Gao. 2025. Cohesiveness-aware hierarchical compressed index for community search on attributed graphs. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–27.
[45] Yuxiang Wang, Shuzhan Ye, Xiaoliang Xu, Yuxia Geng, Zhenghe Zhao, Xiangyu Ke, and Tianxing Wu. 2024. Scalable community search with accuracy guarantee on attributed graphs. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 2737–2750.
[46] Qi Wen, Yutong Ye, Xiang Lian, and Mingsong Chen. 2025. S3AND: Efficient subgraph similarity search under aggregated neighbor difference semantics. *Proceedings of the VLDB Endowment* 18, 11 (2025), 3708–3720.
[47] Yubao Wu, Ruoming Jin, Jing Li, and Xiang Zhang. 2015. Robust local community detection: On free rider effect and its elimination. *Proceedings of the VLDB Endowment* 8, 7 (2015), 798–809.
[48] Junyong Yang, Ming Zhong, Yuanyuan Zhu, Tieyun Qian, Mengchi Liu, and Jeffrey Xu Yu. 2024. Evolution forest index: Towards optimal temporal k-core component search via time-topology isomorphic computation. *Proceedings of the VLDB Endowment* 17, 11 (2024), 2840–2853.
[49] Long Yuan, Lu Qin, Wenjie Zhang, Lijun Chang, and Jianye Yang. 2017. Index-based densest clique percolation community search in networks. *IEEE Transactions on Knowledge and Data Engineering* 30, 5 (2017), 922–935.
[50] Lyuheng Yuan, Da Yan, Wenwen Qu, Saugat Adhikari, Jalal Khalil, Cheng Long, and Xiaoling Wang. 2023. T-FSM: A task-based system for massively parallel frequent subgraph pattern mining from a big graph. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.
[51] Yue Zeng, Hongchao Qin, Rong-Hua Li, Kai Wang, Guoren Wang, and Xuemin Lin. 2024. Mining quasi-periodic communities in temporal network. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 2476–2488.
[52] Qianzhen Zhang, Deke Guo, Xiang Zhao, Xinyi Li, and Xi Wang. 2020. Seasonal-periodic subgraph mining in temporal networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2309–2312.
[53] Yuxin Zhang, Yiqiang Chen, Jindong Wang, and Zhiwen Pan. 2021. Unsupervised deep anomaly detection for multi-sensor time-series signals. *IEEE Transactions on Knowledge and Data Engineering* 35, 2 (2021), 2118–2132.
[54] Yifei Zhang, Longlong Lin, Pingpeng Yuan, and Hai Jin. 2022. Significant engagement community search on temporal networks. In *International Conference on Database Systems for Advanced Applications*. Springer, 250–258.
[55] Xiaokang Zhou, Wei Liang, Zijia Luo, and Yi Pan. 2021. Periodic-aware intelligent prediction model for information diffusion in social networks. *IEEE Transactions on Network Science and Engineering* 8, 2 (2021), 894–904.

Received October 2025; revised January 2026; accepted February 2026