# Effective Durable Community Search in Large Temporal Graph

Yingli Zhou*
yinglizhou@link.cuhk.edu.cn
The Chinese University of Hong Kong, Shenzhen

Yige Jiang*
yigejiang@link.cuhk.edu.cn
The Chinese University of Hong Kong, Shenzhen

Yixiang Fang
fangyixiang@cuhk.edu.cn
The Chinese University of Hong Kong, Shenzhen

Wensheng Luo
luowensheng@hnu.edu.cn
Hunan University

Yongmin Hu
huyongmin@bytedance.com
ByteDance Inc

Yingqian Hu
huyingqian@bytedance.com
ByteDance Inc

Cheng Chen
chencheng.sg@bytedance.com
ByteDance Inc

## ABSTRACT

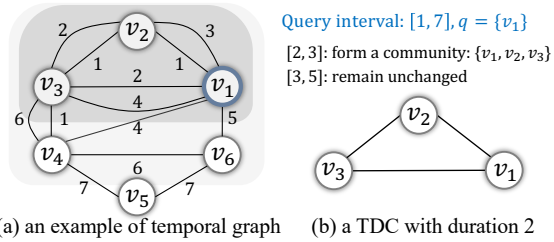
A temporal graph is an undirected graph where each edge is associated with a timestamp indicating when it occurs. As a fundamental topic in graph analysis, community search (CS) in temporal graphs has received much attention. Existing CS works on temporal graphs typically identify sets of vertices that form a $k$-core within a specific time window (temporal $k$-core). However, they overlook the duration of a temporal community, which is the continues time period that its members remain unchanged. Intuitively, the longer the duration of a temporal community, the higher its stability. Long-duration communities are useful in many areas, such as event detection and network analysis. In this paper, we introduce a novel community model, called temporal durable community (TDC), which is the temporal $k$-core with the longest duration in the temporal graph, and aim to efficiently find the TDC containing a query vertex. To solve this problem, we first propose a novel online algorithm based on binary search. We further develop two index structures that can quickly determine the duration of a given temporal $k$-core, followed by query algorithms. Experiments on ten real large temporal graphs show that our TDC model is effective for finding stable communities, and our index-based query algorithms are up to five orders of magnitude faster than the online algorithm.

**PVLDB Reference Format:**
Yingli Zhou, Yige Jiang, Yixiang Fang, Wensheng Luo, Yongmin Hu, Yingqian Hu, and Cheng Chen. Effective Durable Community Search in Large Temporal Graph. PVLDB, 19(1): XXX-XXX, 2025.
doi:XX.XX/XXX.XX

**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://github.com/kiwiHM/Temporal-Stable-Community-Search.git.

## 1 INTRODUCTION

In many real-world applications, the relationships between entities can be modeled as temporal graphs, where each edge has a timestamp representing the interaction time. For example, in a transaction network, the transactions between accounts are timestamped; in social networks, users' posting and sharing messages have timestamps; and in email networks, the sending and receiving of emails include time information. Figure 1 gives a toy example, where the numbers on the edges are the interaction timestamps.




(a) an example of temporal graph (b) a TDC with duration 2

**Figure 1: Illustrating the TDC in a temporal graph.**

In this paper, we aim to find communities in temporal graphs containing a query vertex, which play a vital role in many real-world applications and have garnered much research interest [27, 29, 31, 47, 50, 83, 86, 90]. For example, in the bibliographic network, by revealing the temporal communities of a researcher in different time periods, we can keep track of the evolution of her communities and analyze the changes in her collaborators and research interests.

**Prior works.** Earlier community search (CS) studies mainly focus on static graphs [21, 22, 36, 69, 95], often neglecting temporal interactions. To fill this gap, a few recent temporal CS models have emerged [27, 29, 31, 47, 50, 83, 85, 86, 90], and most of them are based on the well-known $k$-core model, which can be further classified into two groups. The first group [27, 29, 47] imposes a strong constraint on temporal cohesiveness, i.e., the community should form a $k$-core for every individual timestamp, which makes the problems NP-hard. To relax the constraint above, the second group [31, 50, 83, 86, 90] models a community by a set of vertices that form a $k$-core in the static graph projected from the temporal graph over a time window. For example, the temporal $k$-core [86, 90] identifies $k$-cores within graph snapshots within a given time window for a specified $k$; QTC [50] also models the community by using the temporal $k$-core but considers the proximity between the query vertex and other vertices. Although the models above are effective in revealing meaningful temporal communities, they overlook the duration of a temporal community, which is the continuous period in which its members remain unchanged.

Intuitively, the longer the duration of a temporal community, the higher its stability. We observe that in many real-world applications, the stability of a community plays a vital role, since it indicates the duration of common interests of community members. Motivated by these, in this paper, we introduce a novel community model, called temporal durable community (TDC), which is the temporal $k$-core with the longest community duration in the temporal graph. Specifically, given a query time window $[t_s, t_e]$, if a set of vertices

*The first two authors contributed equally to this research.


Proceedings of the VLDB Endowment, Vol. 19, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

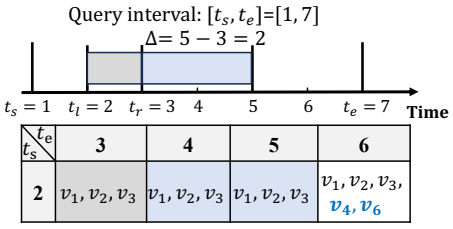



**Figure 2: Illustrating the duration of TDC.**

forms a community in the time window $[t_l, t_r]$ and its members remain unchanged in $[t_r, t_r + \Delta]$ where $\Delta \geq 0$ and $t_r, t_r + \Delta \leq t_e$, then we say that its community duration is $\Delta$. In Figure 1, let the query interval be $[t_s, t_e] = [1, 7]$, the query vertex $q = v_1$, and $k = 2$. We see that $\{v_1, v_2, v_3\}$ forms a community in the time interval $[t_l, t_r]$=[2, 3], and remains unchanged in [3, 5], resulting in a duration of $\Delta = 5 - 3 = 2$. Figure 2 illustrates this process: across the time windows [2, 3], [2, 4], and [2, 5], the community remains the same. In particular, during the time interval [3, 5], no new members join the community. Note that at the timestamp 6, $v_4$ and $v_6$ join the community, leading to the community change. This community is the TDC, as it achieves the longest duration among all communities that can be formed within $[t_s, t_e]$.

**Applications.** TDCs have found various applications in many real-world scenarios, to name a few:

*(1) Dynamic network analysis.* TDCs provide a quantitative lens to track the evolution of research teams in co-authorship networks. We consider a co-authorship network where two researchers are connected if they have co-authored at least two papers [97], and the timestamp of each edge corresponds to the publication time. Stable research teams emerge as TDCs with long durations, while shifts in research focus disrupt this continuity. For instance, when a researcher transitions to a new domain (e.g., Jim Gray's shift from databases to astronomy), the duration of their existing community drops sharply due to an influx of short-term collaborations. As the researchers establish themselves in the new field, the duration rebounds, reflecting the formation of a new stable collaboration community. Our TDC model captures these dynamics effectively. In Figure 3, we analyze Jim Gray's co-authorship community from 1957 to 1997 using TDC queries with $k$=5 over five-year time windows. We observe that: (1) during his transition from databases to astronomy (e.g., 1977–1987), his community duration dropped significantly due to collaborations with many new co-authors, and (2) after establishing himself in astronomy (e.g., 1992–1997), the duration increased again, indicating the formation of a new stable research group. In contrast, the duration values of the other temporal communities remain largely unchanged throughout this period, failing to capture these important structural changes. We report his two corresponding research communities in Figure 4.

*(2) Event tracking.* On social media platforms like Reddit, user interactions (e.g., comments on posts) create temporal graphs where activity patterns mirror real-world events. TDCs with long durations often correspond to major events [64], particularly long-term topics like the Russia-Ukraine War. Besides, by keeping track of TDCs, we may identify some bursting events (e.g., FIFA World Cup or the Olympic Games) and monitor the trends of users' opinions or interests. We demonstrate these applications through a case study using Reddit data in Section 6.

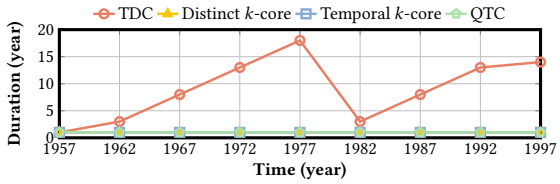
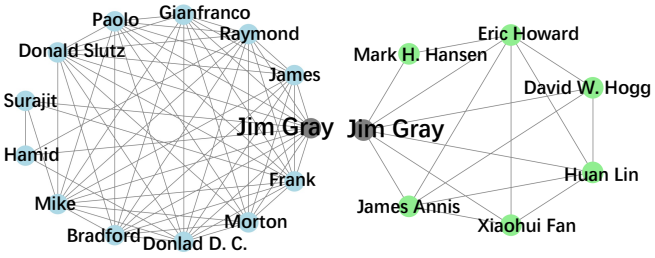



**Figure 3: Duration of community for Jim Gray.**




(a) Jim Gray's Database Community (b) Jim Gray's Astronomy Community

**Figure 4: The community of Jim Gary.**

*(3) Filter bubbles detection.* On platforms like X (formerly known as Twitter) or Facebook, if a group of users consistently interacts only with each other and shares similar content without any new users joining over a long period, it indicates a filter bubble [42], where the absence of new members shows the group's isolation, keeping interactions and information within the existing members. By searching TDCs, we may identify the hidden filter bubbles, and break filter bubbles by recommending some new community members to join the community, reducing the echo chamber effect.

**Challenges.** Given a query time window $[t_s, t_e]$, to find the TDC containing a query vertex, a naive algorithm needs to enumerate all the possible sub-windows in $[t_s, t_e]$ to identify the temporal $k$-core with the longest duration, which consists of two major steps: *(1) compute temporal $k$-core*, and *(2) identify the duration of this temporal core*. For step (1), we can directly use the core decomposition method [3] to compute it, while step (2) is more challenging because for a temporal $k$-core over $[t_l, t_r] \subseteq [t_s, t_e]$, we have to examine all the sub-windows in $[t_r, t_e]$ to find the longest duration when the temporal $k$-core remains unchanged, which is impractical since it takes $O(t_{max}^3 \cdot (n+m))$ time in the worst case, where $t_{max}$ represents the number of distinct timestamps in the graph.

**Our technical contributions.** We tackle the issues above by proposing both efficient online and index-based solutions.

*(1) Online solution.* We propose a non-trivial online algorithm based on the binary search. Specifically, we observe that given a start time $t_l$ and an integer $k$, the temporal $k$-core will never be split and can only be merged as the end time of the window increases. Based on this observation, for a time window $[t_l, t_r]$, the community duration of the temporal $k$-core can be computed by performing a binary search to find the largest end time at which this core changes, with the start time anchored at $t_l$.

*(2) Index-based solutions.* To enable efficient frequent TDC queries with various parameters, we propose time- and space-efficient index-based solutions that can quickly determine the community duration of a given temporal $k$-core. For each start time $t_l$ and integer $k$, we store the active time $\mathcal{L}_{t_l}(e)_k$ for edge $e$, which represents the smallest end time $t_r$ that $e$ can be included in the temporal

$k$-core over $[t_l, t_r]$, and the active times form a minimum spanning forest (MSF). The MSF of these active times forms the Active Spanning Forest index (ASF-index). The ASF-index can be constructed using the Kruskal algorithm [15] for each $k$ and start time $t_s$, but it is time-consuming and memory-intensive. We further observe that the MSF at end time $t_l$ is obtained by inserting the edges with timestamps $t_l$ and those with decreased active times into the MSF at $t_l + 1$. As a result, we only need to store the different parts of MSFs, allowing the representation of $t_{\max}$ distinct MSFs as a single graph, termed the Active Time Graph index, or ATG-index.

Given a time window $[t_l, t_r]$, based on the ATG-index, we develop a TDC search algorithm, which computes the temporal $k$-core by considering edges with active times that are at least the start time $t_l$. The algorithm then identifies the shortest active time among all edges that connect a vertex in the $k$-core to one outside the core. We call this algorithm Basic Index-based TDC search algorithm (BIT). While BIT is faster than our online algorithm, it still needs to recompute the temporal core and its duration for each time window, leading to redundant computations. To reduce redundant computation, we further propose an Advanced Index-based TDC search algorithm (AIT), which uses the temporal core from the previous time window to compute the core and its duration for the current window, achieving higher efficiency.

In addition, we have conducted extensive experiments on ten real-world large temporal graphs, and the reuslts show that our solutions are effective in finding TDCs. Besides, AIT is up to five and three orders of magnitude faster than the online and BIT algorithms respectively, and the ATG-index consumes much less space cost than the directly calculated MSFs, and its construction is also much faster. In summary, our main contributions are as follows.

- We introduce the problem of searching TDCs on temporal graphs, which have not been studied in the literature yet.
- We propose both efficient online and index-based solutions to search TDCs in large temporal graphs.
- We conduct experiments on 10 real-world large temporal graphs to demonstrate the efficiency and effectiveness of our algorithm.

**Outline.** We formally present the TDC search problem in Section 2. Section 3 proposes the online-based TDC search algorithm. In Section 4 and 5, we present our index-based TDC search algorithms. We report the experimental results in Section 6. We review the related works in Section 7 and conclude in Section 8. For lack of space, all proofs in this paper are included in our technical report [98].

## 2 PROBLEM FORMULATION

We consider an unweighted and undirected temporal graph $G = (V, E)$, where $V$ and $E$ denote the sets of vertices and edges, respectively. Let $|V| = n$ and $|E| = m$. Each edge $e \in E$ is a triplet $(u, v, t)$ with $u \in V, v \in V$ and $t \in \mathbb{N}$ representing the interaction timestamp between $u$ and $v$, denoted by $w(e)$. W.l.o.g., we assume that the timestamps of edges are consecutive integer values in the range $[1, t_{max}]$. Note that if timestamps are represented in other formats, we can first apply string hashing to map each edge's timestamp to a value in the range $[1, t_{\max}]$.

*Definition 2.1 (Projected graph [9, 83, 86, 90]).* Given a temporal graph $G = (V, E)$ and a time window $[t_l, t_r]$, the projected graph of $G$ over $[t_l, t_r]$ is denoted as $G[t_l, t_r]=(V, E[t_l, t_r])$, where $E[t_l, t_r] = \{(u, v)|(u, v, t) \in E \wedge t \in [t_l, t_r]\}$

**Table 1: Notations and meanings.**

| Notation | Meaning |
|---|---|
| $G = (V, E)$ | A temporal graph with vertex set $V$, and edge set $E$ |
| $n, m$ | The number of vertices and edges of $G$ respectively |
| $t_{max}$ | The maximum timestamp in $G$ |
| $w(e)$ | The interaction timestamp of edge $e$. |
| $\mathcal{L}_{t_s}(e)_k$ | The active time of edge $e$ over start time $t_s$ and $k$. |
| $S_k[t_s, t_e]$ | The temporal $k$-core over time window $[t_s, t_e]$. |
| $\mathcal{F}[t_s, k]$ | The ASF-index of $G$ over start time $t_s$ and $k$. |
| $\mathcal{M}[k][e]$ | The set of active windows of $e$ over $k$. |
| $\mathcal{G}[k]$ | The ATG-index of $G$ over $k$. |
| $\mathcal{T}$ | The stale community of $G$. |

*Definition 2.2 (Temporal k-core [86, 90]).* Given a temporal graph $G = (V, E)$, a time window $[t_l, t_r]$, and an integer $k$, the temporal $k$-core is a maximal vertex set $S_k[t_l, t_r] \subseteq V$, such that each vertex in the induced subgraph of $G[t_l, t_r]$ has a degree of at least $k$.

*Definition 2.3 (Temporal connectivity [77]).* Given a temporal graph $G = (V, E)$, a time window $[t_l, t_r]$, and two vertices $u$ and $v$, we say $u$ is temporally connected to $v$ in $[t_l, t_r]$, if there exists a path between $u$ to $v$ in $G[t_l, t_r]$.

Following the previous works [47, 86, 90], we model a temporal community by a temporal $k$-core $S_k[t_l, t_r]$ satisfying the temporal connectivity property, i.e., $\forall u, v \in S_k[t_l, t_r]$, they are temporally connected in $[t_l, t_r]$. For simplicity, throughout this paper, we focus on temporal $k$-cores that satisfy the temporal connectivity property.

*Definition 2.4 (**Community duration**).* Given a temporal graph $G$, the community duration of a temporal $k$-core $S_k[t_l, t_r]$ is a value $f(S_k[t_l, t_r])$, such that for any time window $[t_l, t_r + \Delta]$ with $\Delta < f(S_k[t_l, t_r])$, $S_k[t_l, t_r]$ remains unchanged.

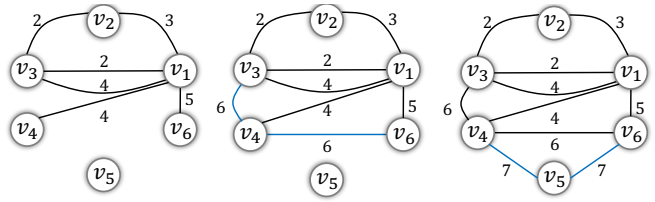
In other words, for a community that is generated in a time interval $[t_l, t_r]$, we characterize its community duration by how long it will remain unchanged after the timestamp $t_r$. For an empty set, its duration is defined as 0, i.e., $f(\emptyset) = 0$. In Figure 1, for example, the three users $\{v_1, v_2, v_3\}$, form a temporal 2-core during $[2, 3]$, i.e., $t_l$=2, and $t_r$=3. They are still a temporal 2-core during $[2, 4]$ and $[2, 5]$, but they are in a larger temporal 2-core during $[2, 6]$, so they are unchanged during $[3, 5]$ and the community duration is 2.

*Definition 2.5 (**Temporal durable community (TDC)**).* Given a temporal graph $G$, a positive integer $k$, and a time window $[t_s, t_e]$, the temporal durable community (TDC) is a vertex set $\mathcal{T}$ satisfying:

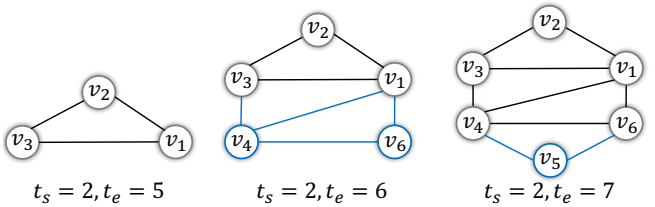
- **Cohesiveness.** Vertices of $\mathcal{T}$ form a temporal $k$-core over $[t_l, t_r]$ where $[t_l, t_r] \subseteq [t_s, t_e]$;
- **Durability**. There is no other temporal $k$-core $\mathcal{T}'$ with community duration $f(\mathcal{T}') > f(\mathcal{T})$.

Clearly, a TDC is the temporal $k$-core that has the longest community duration within $[t_s, t_e]$, after they are generated in $[t_l, t_r]$. Note that although $[t_l, t_r]$ is in $[t_s, t_e]$, it is unknown in advance.

Problem 1 (TDC search). *Given a temporal graph $G$=$(V, E)$, an integer $k$, a query time window $[t_s, t_e]$, and a query vertex $q$, find the TDC containing $q$.*




(a) Projected Graph over each time interval with $t_s = 2$




(b) Temporal 2-core over each time interval with $t_s = 2$

**Figure 5: Illustrating the temporal $k$-core.**

For example, in Figure 1, let $k = 2$, $[t_s, t_e] = [1, 7]$, and $q = v_2$. The TDC should be $\{v_1, v_2, v_3\}$ since they form a temporal 2-core having the longest duration among all the temporal 2-cores.

In practice, temporal graphs are often modeled with edges associated with time intervals (lifespans) [100, 101]. Since a single timestamp can be considered a time interval whose start and end timestamps are the same, the interval-based temporal graph can be regarded as an extension of the timestamp-based temporal graph, implying that the timestamp-based temporal graph is more fundamental. Actually, our proposed solutions on timestamp-based temporal graph can also be easily extended to interval-based temporal graph. Details are provided in our technical report [98].

## 3 ONLINE TDC SEARCH ALGORITHM

In this section, we introduce an online TDC search (ONCE) algorithm based on binary search. Given a time window $[t_s, t_e]$, an integer $k$, and a query vertex $q$, ONCE needs to enumerate all possible subwindows in $[t_s, t_e]$ to compute TDC. An interesting fact is that for an anchored start time $t_s$, the temporal $k$-core would only be merged but never split, when $t_e$ increases from $t_s$ to $t_{max}$. We illustrate this via the Example 1.

Example 1. *Consider the temporal graph shown in Figure 1. We present a chain of projected graphs and their corresponding temporal $k$-cores for $t_s = 2$ and $t_e$ varying from 5 to 7 in Figure 5. It can be observed that as $t_e$ increases, new edges are incrementally added into the projected graph, and thus the corresponding temporal $k$-core either remains the same or grows larger by merging new vertices or edges (marked in blue color). No edges or vertices are ever deleted from the core during this process. This behavior directly follows from the definition of the temporal $k$-core on the projected graph.*

This observation motivates us to use binary search to find the longest end time to enlarge this core, when start time fixed.

Algorithm 1 presents the details. Specifically, for each time window $[t_l, t_r]$, ONCE first computes $S_k[t_l, t_r]$ containing the query vertex $q$ using the classical $k$-core algorithm [3], and then determines its community duration via binary search (lines 4–10). If the community duration of $S_k[t_l, t_r]$ is larger than $f(\mathcal{T})$, we update

**Algorithm 1:** ONCE

```
input : A graph G, two integers k, q, and a time window [t_s, t_e]
output: a TDC T containing q
1  T ← ∅ ;                                   // store the TDC containing q
2  foreach t_l ← t_s, ··· , t_e do
3      foreach t_r ← t_l, ··· , t_e do
4          S_k[t_l, t_r] ← compute a temporal k-core containing q;
5          f(S_k[t_l, t_r]) ← 0; Δ_l ← 0; Δ_r ← t_e − t_r;
           // binary search for computing f(S_k[t_l, t_r])
6          while l < r do
7              Δ_mid ← (Δ_l + Δ_r + 1)/2;
8              if S_k[t_l, t_r]) is same as S_k[t_l, t_r + Δ] for any
                Δ < Δ_mid then
9                  f(S_k[t_l, t_r]) ← Δ_mid; Δ_l ← Δ_mid;
10             else Δ_r ← Δ_mid − 1 ;
11         if f(S_k[t_l, t_r]) > f(T) then T ← S_k[t_l, t_r];
12 return T ;
```

$S_k[t_l, t_r]$ as TDC (line 11). After traversing all time windows, $\mathcal{T}$ is returned as TDC (line 12).

Lemma 3.1. *The total time cost of* ONCE *is $O(t_{max}^2 \cdot (n + m) \cdot \log t_{max})$, where $n$, $m$ and $t_{max}$ denotes the number of vertices, edges and the number of distinct time slots in a temporal graph $G$, respectively.*

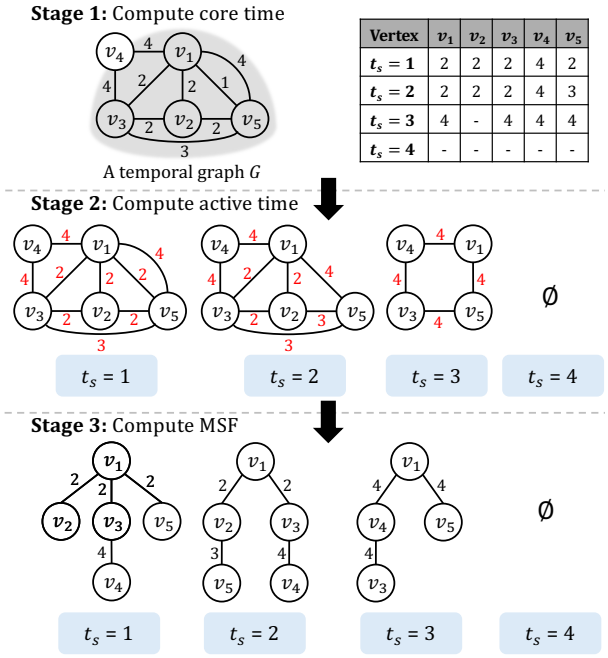
There are two major drawbacks of ONCE: (1) it must enumerate $t_{\max}^2$ time windows in the worst case, and (2) compute the community duration of temporal core for each window takes $O((n + m) \cdot \log t_{\max})$ time. Thus, ONCE is inefficient and un-scalable for large temporal graphs. For example, on the CM dataset which has 1,899 vertices and 59,835 edges, the above algorithm takes more than 20 minutes to answer a single TDC search. To address this, we introduce index-based solutions in the following sections. We first develop the ASF-index, which offers a straightforward design but incurs high construction overhead and substantial space consumption. To overcome these drawbacks, we propose the advanced ATG-index, which achieves significant improvements over the ASF-index by reducing both construction time and index size (see Table 2).

## 4 THE INDEX STRUCTURES FOR TDC SEARCH

A straightforward method to design index-based solutions is to pre-compute and store the community durations of all the possible temporal $k$-cores offline. This, however, is impractical for large temporal graphs, especially when $t_{max}$ is very large. In this section, we first introduce a novel definition, called *active time*, which serves as the cornerstone for building our indices. Building on this, we design a basic index, ASF-index, by precomputing active times. We further introduce optimization techniques to refine the size of ASF-index, leading to our advanced index, ATG-index.

### 4.1 Overview of ASF-index

In this subsection, we present the overview of ASF-index. In the temporal graph, once we have identified the temporal $k$-core $S_k[t_s, t_e]$, computing its duration involves tracking its expansion as $t_e$ increases (i.e., as new vertices join). Specifically, we monitor the temporal edges that connect $S_k[t_s, t_e]$ to vertices that are not in




| Vertex | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ |
|---|---|---|---|---|---|
| $t_s = 1$ | 2 | 2 | 2 | 4 | 2 |
| $t_s = 2$ | 2 | 2 | 2 | 4 | 3 |
| $t_s = 3$ | 4 | - | 4 | 4 | 4 |
| $t_s = 4$ | - | - | - | - | - |

**Figure 6: An example for ASF-index of $G$ with $k = 2$.**

it. The first such temporal edge appearing in the core signals a community change, allowing us to determine the duration. As a result, given a specific $t_s$ and a fixed $k$, by treating the edges' appearance time in the graph as their weights, the above computing process is identical to the Kruskal algorithm [15], since both of them incrementally select the smallest-weight edge to connect new vertices and expand the component. Fortunately, as edge weights encode their temporal $k$-core appearance time, the corresponding temporal $k$-core can be directly obtained from this MSF. The above key insight motivates to develop the ASF-index.

To achieve this, we first introduce the following definitions:

*Definition 4.1 (Core time [90]).* Given a temporal graph $G$, and a vertex $u$, the core time of $u$ for an integer $k$ and a start time $t_s$, denoted as $\mathcal{CT}_{t_s}(u)_k$, is the smallest time $t_e$ such that $u$ is in a temporal $k$-core of the projected graph $G[t_s, t_e]$.

Based on the definition of core time, we propose the following novel definitions for designing our index.

*Definition 4.2 (Active time).* Given a temporal graph $G$, and an edge $e = (u, v)$, the active time of $e$ for an integer $k$ and a start time $t_s$ , denoted as $\mathcal{L}_{t_s}(e)_k$, is the smallest time $t_e$ such that $e$ is in a temporal $k$-core of the projected graph $G[t_s, t_e]$, where $\mathcal{L}_{t_s}(e)_k$ is calculated by the below equation:

$$\mathcal{L}_{t_s}(e)_k = \max\{\mathcal{CT}_{t_s}(u)_k, \mathcal{CT}_{t_s}(v)_k, w(e)\} \quad (1)$$

Example 2. *In the temporal graph $G$ shown in Figure 6 Stage 1, let $k = 2$ and $t_s = 1$. The active time of the edge $e = (v_1, v_2)$ equals to 2, since for all time windows $[1, 2] \subseteq [1, 3] \subseteq [1, 4]$, $e$ is always included in the corresponding temporal 2-cores.*

*Definition 4.3 (Key edge).* Given a temporal $k$-core $S_k[t_l, t_r]$, the key edge $e^*$ of $S_k[t_l, t_r]$ is the edge with the smallest active time such that $S_k[t_l, t_r] \subset S_k[t_l, \mathcal{L}_{t_l}(e^*)_k]$.

Example 3. *As shown in the Figure 6, consider the temporal 2-core $S_2[1, 2] = \{v_1, v_2, v_3, v_5\}$ of temporal graph $G$ in Stage 1 of Figure 6, which is highlighted as shaded . The edge $(v_3, v_4)$ serves as its key edge, with its active time $\mathcal{L}_1((v_3, v_4))_2 = 4$, this is because, $S_2[1, 2] = S_2[1, 3] = \{v_1, v_2, v_3, v_5\} \subset S_2[1, 4] = \{v_1, v_2, v_3, v_4, v_5\}$.*

Lemma 4.4. *The community duration of the temporal $k$-core $S_k[t_l, t_r]$ can be calculated by using the active time of its key edge subtraction $(t_r + 1)$.*

In light of this, when $k$ and $t_s$ are fixed, we can pre-compute the active time for each edge to build an index. Specifically, we construct MSFs using the active time of each edge as its weight. The key edge can be efficiently identified in the MSF, as the Kruskal algorithm incrementally adds such edges to connect new vertices and expand the component.

Here, we can build a naive index, called the active spanning forest (ASF) index, or ASF-index.

*Definition 4.5 (*ASF-index*).* Given a temporal graph $G$, a start time $t_s$, and an integer $k$, the ASF-index is a spanning forest over $t_s$ and $k$, where each tree is a minimum spanning tree (MST), and the active time of each edge is used as its weight.

Example 4. *Figure 6 shows a temporal graph and its ASF-index construction process for $k = 2$. We first compute the core time for each vertex in $G$, defined as the earliest end time $t_e$ such that $u$ is included in the temporal $k$-core of the projected graph $G[t_s, t_x]$ for all $t_x \geq t_e$. The core time is then used to determine the active time for each edge, as illustrated in the accompanying table and Stage 2 of Figure 6. Stage 2 presents four graphs, where each corresponds to a different start time, with edge weights indicating active time. For each graph, we compute the MSF, and the resulting MSFs are shown in Stage 3. These four MSFs constitute the ASF-index of $G$ for $k = 2$.*

We denote $\mathcal{F}[t_s, k]$ as the MSF over $t_s$, and $k$. The neighbors of $u$ in the MSF denoted by $N(u, \mathcal{F}[t_s, k])$. The ASF-index takes $O(k_{max} \cdot t_{max} \cdot n)$ space, since for each $k$, and start time $t_s$, it needs $O(n)$ space to store the MSF.

Lemma 4.6. *During the ASF-index construction process, the MSF is not necessarily unique. However, if an edge cannot be included in any MSF, it is also impossible to serve as a key edge.*

To illustrate this, we assume that there exists a key edge $e^* = (u, v)$ of $S_k[t_l, t_r]$, but it is not included in any $\mathcal{F}[t_l, k]$, where $u \in S_k[t_l, t_r]$ and $v \notin S_k[t_l, t_r]$. We denote $e'$ as the edge with the longest active time among the edges on the path from $u$ to $v$ in $\mathcal{F}[t_l, k]$ such that $\mathcal{L}_{t_l}(e')_k < \mathcal{L}_{t_l}(e^*)_k$. Hence, $v \in S_k[t_l, \mathcal{L}_{t_l}(e')_k]$, which contradicts the definition of key edge.

**Index construction algorithm.** The ASF-index is constructed by first computing the active times of all edges for each $k$ and start time $t_s$, and then using these active times as weights to compute the MST via the Kruskal algorithm [15]. We denote this algorithm as ASFIC and present its details in Algorithm 2. The time complexity of ASFIC is $O(k_{\max} \cdot t_{\max} \cdot m \log n)$.

**Algorithm 2:** ASFIC

**input** : A graph temporal graph $G$
**output**: The ASF-index $\mathcal{F}$

```
foreach k ← 1, ··· , k_max do
    foreach t_s ← 1, ··· , t_max do
        foreach e ∈ G[t_s, t_max] do compute L_{t_s}(e)_k;
        F[t_s, k] ← build MST based on the active times;
return F;
```

**How to utilize the index.** Given a query window $[t_s, t_e]$, the temporal $k$-core $S_k[t_s, t_e]$ containing vertex $q$ can be obtained by performing breadth-first search (BFS) [15] from $q$ and limiting edge traversal to those with active times not exceeding $t_e$. By doing this, we can identify a connected $k$-core containing $q$ in the time window $[t_s, t_e]$. Subsequently, we determine its duration by computing the key edge of $S_k[t_s, t_e]$. That is, ASF-index addresses the two key challenges underlying index design for the TDC search problem.

**Limitations of** ASF-index. A major limitation of ASF-index is that given a fixed $k$, it has to rebuild and store the MSF for each start time in $[1, t_{max}]$, which is very costly in terms of building and storing. Actually, for two consecutive start times, the MSFs are very similar. This inspires us to construct the index incrementally rather than rebuilding it entirely.

## 4.2 Our advanced index ATG-index

In this subsection, we introduce our advanced index, ATG-index from two perspectives: (1) index size, and (2) construction time to optimize the ASF-index. Particularly, our advanced index is based on the following two key observations:

**Two key observations.** For a fixed start time $t_s$ and $k$: (1) *as the start time decreases, the active times of all edges decrease or stay the same*, and (2) *when updating the graph by adding edges or decreasing active times, the* ASF-index *of this new graph is equivalent to directly inserting these edges into the original graph's* ASF-index.

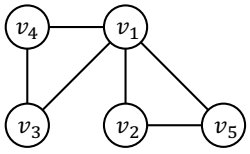
Lemma 4.7. *Given a temporal graph $G$, an integer $k$, an edge $e$=$(u, v)$ and two start times $t_s$, $t'_s$, if $t_s < t'_s$, then $\mathcal{L}_{t_s}(e)_k \leq \mathcal{L}_{t'_s}(e)_k$.*

The first observation is based on the lemma above, and the second observation is based on the relationship of $\mathcal{F}[t, k]$ and $\mathcal{F}[t+1, k]$, as shown in theorem 4.8.

Theorem 4.8. *Given a temporal graph $G$, an integer $k$, and two start times $t$, $t+1$, the $\mathcal{F}[t, k]$ can be constructed by inserting the edges from $\tau[t]$ into $\mathcal{F}[t+1, k]$, where $\tau[t]$ denotes the union of the edges whose active times have changed from $t+1$ to $t$ and the edges with timestamps equal to $t$.*

Note that inserting an edge into $\mathcal{F}[t, k]$ is not always successful, as this edge may not contribute to any MST in $\mathcal{F}[t, k]$. There are two cases when an edge $e = (u, v)$ can be successfully inserted into $\mathcal{F}[t, k]$: (1) $u$ cannot reach $v$. In this case, we directly connect $u$ and $v$, and (2) $u$ can reach $v$. We need to first identify the edge with the longest active time in the path from $u$ to $v$ and replace it with $e$. Here, we can introduce two lemmas about our second observation, based on the properties of MST.

Lemma 4.9. *Given a temporal graph $G$, a start time $t_s$, and an integer $k$, the* ASF-index *of the updated graph $G'$, by adding some edges not exiting in $G$ is equivalent to directly inserting these edges into the* ASF-index *of $G$.*




(a) the ATG-index

| Edge | $\mathcal{M}[2][\cdot]$ |
|---|---|
| $(v_1, v_2)$ | {[1,2]} |
| $(v_1, v_3)$ | {[1,2]} |
| $(v_1, v_4)$ | {[3,3]} |
| $(v_1, v_5)$ | {[1,1], [3,3]} |
| $(v_2, v_5)$ | {[2,2]} |
| $(v_3, v_4)$ | {[1,3]} |

(b) the edge labels of ATG-index

**Figure 7: Illustrating of the ATG-index with $k$ = 2.**

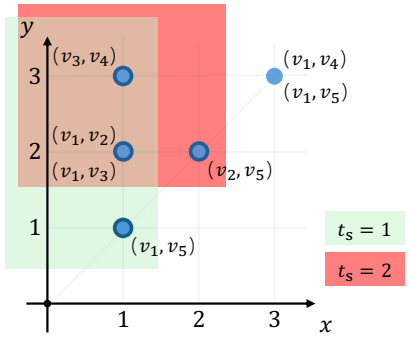



**Figure 8: Illustrating of 2D listing problem.**

This lemma is directly derived from the criteria of MST [33, 67].

Lemma 4.10. *Given a temporal graph $G$, a start time $t_s$, and an integer $k$, the* ASF-index *of the updated graph $G'$, by decreasing the active times of some existing edges in $G$, is equivalent to directly inserting these edges into the* ASF-index *of $G$.*

Here, we are ready to present the proof for Theorem 4.8.

Proof. From the start time $t+1$ to $t$, for the graph $G[t, t_{max}]$, there are two types of change compared to $G[t+1, t_{max}]$: $E_1$: all edges with timestamps equal $t$, and $E_2$: all edges that their active times are decreased. Based on the lemmas 4.9 and 4.10, we can conclude that the ASF-index of $G[t, t_{max}]$ is equals to insert all edges in $E_1 \cup E_2$ (i.e., $\tau[t]$) into the ASF-index of $G[t+1, t_{max}]$. □

**Key idea of** ATG-index. According to the above discussions, we can use an incremental paradigm to construct our index, which not only reduces the building time but also enables a more space-friendly index structure. Specifically, for a fixed start time $t_s$ ($t_s > 1$), and an integer $k$, we can build $\mathcal{F}[t_s - 1, k]$ from $\mathcal{F}[t_s, k]$, instead of rebuilding it entirely. For each anchored $k$, we build the index for all start times from $t_{max}$ to 1. The $t_{max}$ MSFs can be represented by a labeled graph, where each edge $e$ in the graph is labeled with a set of active windows ($\mathcal{M}[k][e]$), which is used to determine whether, given a start time $t_s$, the edge $e$ appears in $\mathcal{F}[t_s, k]$. This index is denoted by the active graph time index (ATG-index).

Formally, the active window is defined as:

*Definition 4.11 (Active window).* Given an ASF-index $\mathcal{F}$, an integer $k$, and an edge $e$, we call a time window $[t_x, t_y]$ an active window of $e$ if all the following conditions are satisfied:

- $\forall t \in [t_x, t_y], e \in \mathcal{F}[t, k]$;
- $e \notin \mathcal{F}[t_x - 1, k]$ if $t_x > 1$;
- $e \notin \mathcal{F}[t_y + 1, k]$ if $t_y < t_{max}$

Given an integer $k$ and an edge $e$, let $\mathcal{M}[k][e]$ denote the set of active windows for $e$. Based on this, we can easily determine whether edge $e$ is included in the ASF-index $\mathcal{F}[t, k]$ by checking whether $t$ falls within any active window in $\mathcal{M}[k][e]$. Formally, we have $e \in \mathcal{F}[t, k]$, iff $\exists [t_x, t_y] \in \mathcal{M}[k][e]$, such that $t \in [t_x, t_y]$.

We denote $\mathcal{G}[k]$ as the ATG-index of $k$. Intuitively, ATG-index can be considered as a "compact" version of ASF-index, by merging all forests into a graph and only keeping the necessary edges to recover the TDCs. Based on the ATG-index, given an integer $k$ and a start time $t_s$, we can quickly identify all edges in the $\mathcal{F}[t_s, k]$, by transferring it into a 2D axis rectangular range reporting problem.

**Problem Transformation.** In ATG-index, each edge in $\mathcal{G}[k]$ is associated with a timestamp pair $(l, r)$, where $l$ represents the starting time of the edge and $r$ represents the ending time. By mapping each edge to a 2D point $(l, r)$ in a two-dimensional space, the original indexing problem is transformed into a 2D axis-parallel rectangular range reporting problem.

Given a query time $t_s$, our task is to find all edges that satisfy the condition: $x \le t_s$ and $y \ge t_s$. In the 2D space, this is equivalent to listing all points $(l, r)$ that fall within the query range: $(-\infty, t_s] \times [t_s, +\infty)$. To efficiently retrieve these points, we employ a Priority Search Tree (PST), a data structure that supports orthogonal range queries efficiently. A PST is constructed as a binary search tree on the $x$-coordinates, while each subtree maintains a min-heap structure on the $y$-coordinates. This allows us to efficiently enumerate all qualifying points with average time complexity $O(\log n + k)$, where $k$ is the number of matching edges.

Example 5. *Figure 7 shows the* ATG-index *($k$ = 2) for the temporal graph in Figure 6 Stage 1. Within this* ATG-index $\mathcal{G}[2]$*, each edge is associated with a label, as shown in Figure 7(b). We also present an example to illustrate how to derive the MSF from* ATG-index *based on the 2D listing in Figure 8. For instance, given a start time $t_s = 1$ and $k = 2$, all points shaded in the green region satisfy $x \le 1$ and $y \ge 1$. The corresponding edges, including $(v_1, v_2)$, $(v_1, v_3)$, $(v_1, v_5)$, and $(v_3, v_4)$, would be included in $\mathcal{F}[1, 2]$. In addition, for a start time $t_s = 2$ and $k = 2$, all points shaded in the red region are valid, so $(v_1, v_2)$, $(v_1, v_3)$, $(v_2, v_5)$, and $(v_3, v_4)$ are included in $\mathcal{F}[4, 2]$.*

Based on this novel transformation, we can use an incremental-based algorithm to build the ATG-index, instead of rebuilding it for each time window. The key idea is that for each $k$, it first computes $\mathcal{F}[t_{max}, k]$, then iteratively uses $\mathcal{F}[t, k]$ to construct $\mathcal{F}[t-1, k]$, and finally only stores the differing parts. We denote this algorithm as ATGIC, the details are shown in Algorithm 3.

For each $k$, we first initialize the $G[k]$ and $\mathcal{M}[k]$ as empty sets (line 2). Then, for each start time $t_s$ from $t_{max} - 1$ to 1, we first identify the edge set $\tau[t_s]$), where each edge needs to be inserted into the index $G[k]$ at $t_s$ timestamp (line 7). Next, for each edge $e = (u, v) \in \tau[t_s]$, we try to insert it into $\mathcal{G}[k]$. Specifically, when $e$ is successfully inserted into the index at $t_s$ (lines 8-9), and if a path

**Algorithm 3:** ATGIC

```
input  : A graph temporal graph G
output : The ATG-index G
1  foreach k ← 1, ··· , k_max do
2      M[k] ← ∅; G[k] ← ∅;
3      foreach t_s ← t_max − 1, ··· , 1 do
4          H ← ∅;
5          τ[t_s] ← {e | L_{t_s+1}(e)_k > L_{t_s}(e)_k ∪ w(e) = t_s} ;
6          foreach e = (u, v) ∈ τ[t_s] do
7              insert {e} into the index G[k];
8              if {e} can be inserted into G[k] then
9                  H[e] ← t_s;
10                 if there exists a path from u to v then
11                     e' ← the edge needs to remove from G[k];
12                     M[k][e'] ← M[k][e'] ∪ {(H[e'], t_s)};
13 return G ;
```

**Table 2: Summary of two indices for TDC search.**

| | Index space | Building time |
|---|---|---|
| ASF-index | $O(k_{\max} \cdot t_{\max} \cdot n)$ | $(O(k_{\max} \cdot t_{\max} \cdot m \log n)$ |
| ATG-index | $O(k_{\max} \cdot \overline{m})$ | $O(k_{\max} \cdot \bar{c} \cdot m \log n)$ |

★ Note: $n$ and $m$ denote the number of vertices and edges in $G$.
★ Note: Typically, $\bar{c} \ll t_{max}$, and $\overline{m} \ll t_{max} \cdot n$.

from $u$ to $v$ already exists, another edge $e'$ will be removed from $\mathcal{G}[k]$ (lines 10-11). When we identify this edge $e'$, we append this timestamp pair $\{(H[e'], t_s)\}$ into $\mathcal{M}[k][e']$ (lines 12). Finally, the ATG-index $\mathcal{G}$ is returned, where for each $k$ and edge $e$, $\mathcal{M}[k][e]$ stores the set of active windows assigned to $e$.

Lemma 4.12. *The total time cost of* ATGIC *is $O(k_{max} \cdot \bar{c} \cdot m \log n)$, where $\bar{c}$ denotes the average number of changes for the active time of an edge. Typically, $\bar{c} \ll t_{max}$.*

Lemma 4.13. *The* ATG-index *requires $O(k_{max} \cdot \overline{m})$ space, where $\overline{m}$ represents the number of all edges in the index, which be calculated by multiplying the number of edges by the number of times each edge appears in the index. Note that $\overline{m}$ is bounded by $m \cdot \bar{c}$ and $\overline{m} \ll n \cdot t_{max}$, as shown in our experimental results.*

**Remark 1.** *We summarize the index size and building time of* ASF-index *and* ATG-index *in Table 2. We can see that our advanced* ATG-index *can be constructed very efficiently. Note that $\bar{c}$ is even smaller than 10, as shown in our latter experiments, indicating that our algorithms can effectively handle large real-world temporal graphs.*

## 5 INDEX-BASED TDC SEARCH ALGORITHMS

In this section, we propose two index-based TDC search algorithms, which employ ATG-index to quickly compute temporal $k$-core and its community duration. We would like to highlight that both the ASF-index and ATG-index are identical for the TDC search when $k$ and $t_s$ are specified, as both represent the same ASF. Since ATG-index is more space-efficient and faster to construct than ASF-index, we primarily investigate the TDC search algorithm based on ATG-index.

## 5.1 A basic index-based TDC search algorithm

Recall that for a fixed $k$, and a start time $t_s$, we need to enumerate all possible end times to identify the durable community. In our index, there are at most $\min\{n, t_{max}\}$ end times for a given $k$ and $t_s$. Here, we first introduce our basic index-based TDC search algorithm, BIT.

**Algorithm 4:** BIT

**input** : A graph $G$, two integers $k$, $q$, a time window $[t_s, t_e]$, and the ATG-index $\mathcal{G}$ of $G$
**output**: a TDC $\mathcal{T}$ containing $q$

1 $\mathcal{T} \leftarrow \emptyset$;
2 **foreach** $t_l \leftarrow t_s, \cdots, t_e$ **do**
3  $\mathcal{F}[t_l, k] \leftarrow$ derive the MSF from $\mathcal{G}[k]$;
4  **foreach** $t_r \in$ *the timestamps of the edges in* $\mathcal{F}[t_l, k]$ **do**
5  $S_k[t_l, t_r] \leftarrow$ computeCore$(q, \mathcal{F}[t_l, k], t_r)$;
6  $e^* \leftarrow$ calculate the key edge of $S_k[t_l, t_r]$;
  // compute the community duration time
7  **if** $\mathcal{L}_{t_l}(e^*)_k - t_l - 1 > f(\mathcal{T})$ **then** $\mathcal{T} \leftarrow S_k[t_l, t_r]$;
8 **return** $\mathcal{T}$;

9 **Function** computeCore$(q, \mathcal{F}[t_l, k], t_r)$:
10  $S \leftarrow \emptyset$; $Q \leftarrow \emptyset$; $vis \leftarrow \emptyset$;
11  $Q$.add($q$); $vis[q] \leftarrow$ True;
12  **while** $Q \neq \emptyset$ **do**
13  $u \leftarrow Q$.poll(); $S$.add(q);
14  **foreach** $v \in N(u, \mathcal{F}[t_l, k])$ ***and*** $vis[v]$ *is not* True **do**
15  **if** $\mathcal{L}_{t_l}(u, v)_k \leq t_r$ **then**
16  $Q$.add($v$); $vis[v] \leftarrow$ True;
17  **return** $S$;

As shown in Algorithm 4, BIT works as follows: Given a fixed start time $t_l$, it first derives the MSF $\mathcal{F}[t_l, k]$ from $\mathcal{G}[k]$ (line 3), and for each time window $[t_l, t_r] \subseteq [t_s, t_e]$, BIT utilizes computeCore to obtain $S_k[t_l, t_r]$ containing $q$ by only visiting the edges in $\mathcal{F}[t_l, k]$ that have active times not larger than $t_r$ (lines 9-17). The key edge of $S_k[t_l, t_r]$ can be calculated based on its definition (line 6). When a temporal $k$-core with a longer community duration is obtained, it is updated as the TDC $\mathcal{T}$ (line 7). After all time windows have been enumerated, $\mathcal{T}$ is returned as the TDC.

Lemma 5.1. *The total time cost of* BIT *is* $O(t_{max} \cdot \min\{n, t_{max}\} \cdot n)$, *where* $n$ *denotes the number of vertices in a temporal graph* $G$.

## 5.2 An advanced index-based TDC search algorithm

In BIT, it recomputes the temporal $k$-core and its community duration for each time window, which is inefficient and unnecessary. To improve the efficiency, we propose an advanced index-based TDC search algorithm (AIT) based on the nested relationship of temporal $k$-core.

Lemma 5.2 ([90]). *Given a temporal graph* $G$, *a fixed start time* $t_l$, *and an integer* $k$, *there exists a chain of temporal* $k$*-core such that*

$$S_k[t_l, t_l] \subseteq S_k[t_l, t_l + 1] \subseteq \cdots \subseteq S_k[t_l, t_{max}]. \quad (2)$$

The above lemma is also illustrated in Example 1. Hence, we can leverage $S_k[t_l, t_r]$ to compute $S_k[t_l, t_r + 1]$ without recomputation. However, there are two key challenges for designing AIT: (1) *how to identify which vertices need to be added into* $S_k[t_l, t_r]$ *to construct* $S_k[t_l, t_r + 1]$. (2) *once* $S_k[t_l, t_r + 1]$ *is determined, how to quickly calculate its community duration.*

To solve these two challenges, we first introduce the following novel definitions.

*Definition 5.3 (Outer set).* Given a temporal $k$-core $S_k[t_l, t_r]$, and MSF $\mathcal{F}[t_l, k]$, the outer set $\mathcal{E}(S_k[t_l, t_r])$ of $S_k[t_l, t_r]$ is defined as the edge set such that $\mathcal{E}(S_k[t_l, t_r]) = \{e = (u, v) \mid e \in \mathcal{F}[t_l, k] \wedge u \in S_k[t_l, t_r] \wedge v \notin S_k[t_l, t_r] \wedge \mathcal{L}_{t_l}(e)_k > t_r\}$.

Clearly, for a temporal $k$-core $S_k[t_l, t_r]$, the edge in its outer set with the shortest active time is referred to as the key edge of $S_k[t_l, t_r]$. Here, we classify the relationship among a temporal $k$-core, its key edge, outer set, and duration. Specifically, given a temporal $k$-core $S_k[t_l, t_r]$, we first identify its outer set. Among these edges, the one with the shortest active time is defined as the key edge of $S_k[t_l, t_r]$. The timestamp of this key edge corresponds exactly to the duration of $S_k[t_l, t_r]$, denoted as $f(S_k[t_l, t_r])$.

Lemma 5.4. *Given a temporal* $k$*-core* $S_k[t_l, t_r]$, *and its key edge* $e^* = (u, v)$, *assuming* $u \in S_k[t_l, t_r]$ *and* $v \notin S_k[t_l, t_r]$, $v$ *must be included in* $S_k[t_l, t'_r]$, *if* $t'_r \geq \mathcal{L}_{t_l}(e^*)_k$.

The lemmas above inspire us to employ an iterative strategy to search TDC. Specifically, to compute $S_k[t_l, t_r+1]$ and its community duration, we repeatedly add vertices to $S_k[t_l, t_r]$ by selecting the key edge from its outer set. The outer set is also updated as new vertices join $S_k[t_l, t_r]$. This process stops until the selected key edge's active time exceeds $t_r + 1$.

**Algorithm 5:** AIT

**input** : A graph $G$, two integers $k$, $q$, a time window $[t_s, t_e]$, and the ATG-index $\mathcal{G}$ of $G$
**output**: a TDC $\mathcal{T}$ containing $q$

1 **foreach** $t_l \leftarrow t_s, \cdots, t_e$ **do**
2  $\mathcal{E}(S_k) \leftarrow \emptyset$; $\mathcal{F}[t_l, k] \leftarrow$ derive the MSF from $\mathcal{G}[k]$;
3  **foreach** $t_r \in$ *the active times of the edges in* $\mathcal{F}[t_l, k]$ **do**
4  **if** $S_k = \emptyset$ **then**
5  $S_k \leftarrow$ computeCore$(q, \mathcal{F}[t_l, k], t_r)$;
6  $e^* = (u, v) \leftarrow \text{argmin}_e\{\mathcal{L}_{t_l}(e)_k \mid e \in \mathcal{E}(S_k)\}$;
7  **foreach** $e = (u, v) \in \mathcal{F}[t_l, k]$ **do**
8  **if** $u \in S_k[t_l, t_r]$ ***and*** $v \notin S_k$ **then**
9  $\mathcal{E}(S_k) \leftarrow \mathcal{E}(S_k) \cup \{e\}$;
10  **else**
11  **repeat**
12  $S_k \leftarrow S_k \cup \{v\}$, assuming $v \notin S_k \wedge u \in S_k$;
13  $e^* = (u, v) \leftarrow \text{argmin}_e\{\mathcal{L}_{t_l}(e)_k \mid e \in \mathcal{E}(S_k)\}$;
14  remove $e^*$ from $\mathcal{E}(S_k)$;
15  $\mathcal{E}(S_k) \leftarrow \mathcal{E}(S_k) \cup \{(v, x) \in \mathcal{F}[t_l, k] \wedge x \notin S_k\}$;
16  **until** $\mathcal{L}_{t_l}(e^*) > t_r$;
  // $e^*$ is the key edge
17  **if** $\mathcal{L}_{t_l}(e^*)_k - t_r - 1 > f(\mathcal{T})$ **then** $\mathcal{T} \leftarrow S_k$;
18 **return** $\mathcal{T}$;

Algorithm 5 presents the details of AIT. Given a fixed $k$, for an anchored start time $t_l$, the $\mathcal{E}(S_k)$ is set $\emptyset$, and $\mathcal{F}[t_l, k]$ is derived from $\mathcal{G}[k]$ (line 2). Then, for each time window $[t_l, t_r]$, the temporal $k$-core $S_k$ and its community duration can be calculated (lines 5 - 17). Specifically, if $S_k = \emptyset$, we need to use computeCore to compute this temporal $k$-core, and the $\mathcal{E}(S_k)$ (lines 4 - 9). Otherwise, $S_k$

denotes $S_k[t_l, t_r - 1]$, and then we iteratively selects the key edge $e^*$ from $\mathcal{E}(S_k)$. If $\mathcal{L}_{t_l}(e^*)_k \leq t_r$ and $u \in S_k \wedge v \notin S_k$, $v$ is added to $S_k$ without additional computations. Next, $e^*$ is removed from $\mathcal{E}(S_k)$, and all edges starting from $v$ to vertices not in $S_k$ are added to $\mathcal{E}(S_k)$. This process continues until the active time of the key edge selected from $\mathcal{E}(S_k)$ exceeds $t_r$ (lines 12 - 17) Then, the $S_k[t_l, t_r]$ and its key edge can be correctly calculated.

Theorem 5.5. *The algorithm 5 can correctly return the TDC.*

Example 6. *We continue to consider the temporal graph G in Stage 1 of Figure 6. Given a time window $[1, 4]$, $k = 2$, and a query vertex $v_1$, taking start time $t_l = 1$ as an example: First,* AIT *computes $S_2[1, 2] = \{v_1, v_2, v_3, v_5\}$ and its outer set $\mathcal{E}(S_2[1, 2]) = \{v_4\}$, selecting edge $e^* = (v_3, v_4)$ as the key edge of $S_2[1, 2]$. Then $f(S_2[1, 2]) = 1$ due to $\mathcal{L}_1(e^*)_2 = 4$. For other start times,* AIT *employs a similar process to search for TDC, however, it does not find any community with a community duration greater than 1. Thus, $S_2[1, 2]$ is the TDC.*

Lemma 5.6. *The time cost of* AIT *is $O(t_{max} \cdot n + t_{max} \cdot \min\{t_{max}, n\} \cdot \log n)$, where all variables are defined as in Lemma 3.1.*

Here, we summarize the online and two index-based TDC search algorithms. As we can see, the AIT algorithm achieves significantly improved performance compared to ONCE and BIT. In addition, AIT is more efficient than BIT.

# 6 EXPERIMENTS

We now present the experimental results. Section 6.1 discusses the setup. We discuss the experimental results in Sections 6.2 and 6.3.

## 6.1 Setup

**Datasets.** We use 13 real-world temporal graphs from different domains, which are downloaded from the Stanford Network Analysis Platform [1] and Konect [40]. Table 3 provides the statistics of each graph, where $n$ and $m$ are the numbers of vertices and edges, respectively, and $t_{max}$ is the maximum timestamp (counting from 1). Their detailed descriptions can also be found on these websites. All algorithms are implemented in C++, compiled with the g++ compiler at -Ofast optimization level, and run on a Linux machine with an Intel Xeon 2.40GHz CPU and 384GB RAM. If an algorithm cannot finish in 72 hours, we mark its running time as **INF**. If the index construction does not complete within three days, we record its index size as "TLE".

**TDC queries.** To measure the query efficiency, for each dataset, we generate 1,000 queries with different time window sizes (i.e., $t_e - t_s$) and $k$, where $k$ ranges from 5 to 13, the size of time window varies from $0.2 \cdot t_{max}$ to $t_{max}$, and $t_s$ is selected randomly. Then we execute the 1000 queries sequentially and compute the average time cost of the queries. Besides, the index construction time and the memory usage for each dataset are also measured, respectively.

**Competitors.** We test the following algorithms:

- ONCE: Our online TDC search algorithm.
- TECE: Integrate the algorithm from [90] into ONCE, replacing its original temporal $k$-core computation component.
- DICE: Enumerate all distinct $k$-cores containing $q$ using the algorithm in [86], and select the one with the longest duration.
- BIT: A basic index-based TDC search algorithm.
- AIT: An advanced index-based TDC search algorithm.
- ASFIC:the ASF-index construction algorithm (Algorithm 2).
- ATGIC: the ATG-index construction algorithm (Algorithm 3).

## 6.2 Efficiency evaluation

In this section, we evaluate the efficiency of our proposed methods.

**1. Effect of $k$.** Figures 9 and 11(a)-(c) compare the average running time of these three algorithms on all datasets by varying the $k$ from 5 to 13. Clearly, AIT is up to three orders and five orders of magnitude faster than BIT and ONCE, respectively, since it does not require recomputing the temporal $k$-core and its duration in each time window. For example, on the AU dataset with $k$=5, AIT only takes 10s to compute the TDS, while ONCE and BIT cannot finish in three days. In addition, our index-based algorithms are significantly faster than the online method. The online method ONCE cannot finish in three days for all datasets, this is because the online method takes $O(t_{\max}^2 \cdot (n + m) \cdot \log t_{\max})$ time to find the TCD, while our index-based solutions, BIT and AIT, require $O(t_{\max} \cdot \min\{n, t_{\max}\} \cdot n)$ and $O(t_{\max} \cdot n + t_{\max} \cdot \min\{t_{\max}, n\} \cdot \log n)$ time, respectively. In addition, our index-based algorithms are significantly faster than the TECE and DECE algorithms, as both are based on ONCE and inherit its key disadvantages.

**2. Effect of the size of time windows.** In this experiment, we compare the average running time of our three algorithms on five datasets, by varying the size of time windows (we fixed $k$ as 5), for each graph, we fix the start time of the query window as $t_s = 1$ and consider five query time window sizes, i.e., 20%, 40%, 60%, 80% and 100% of $t_{max}$ respectively, as shown in Figures 10 and and 11(d)-(f). We can obtain the following observations: (1) the running time of all algorithms increases with the growth of the size of time windows, and (2) when the time window is larger, the BIT algorithm takes more time, while the time costs of AIT do not change much. (3) for the DP dataset, we observe that two index-based search algorithms achieve comparable performance. Since $t_{\max}$ is quite small for DP, the $t_{\max}^2$ term is also negligible, making the total running time of both algorithms theoretically comparable on this dataset.

**3. Index space cost.** We report the space cost of each index on all graphs in Table 4. We can see that: (1) The space cost of ATG-index is much less than that of ASF-index. This is because their space costs are bounded by $O(k_{max} \cdot n \cdot t_{max})$ and $O(k_{max} \cdot \overline{m})$, respectively, where $\overline{m} \ll n \cdot t_{max}$. Besides, on the EN dataset, ASF-index needs more than 5 GB, while ATG-index only takes 291.3 MB. Meanwhile, on the SL dataset, ASF-index needs 422.3 MB, while ATG-index only takes 21.8 MB. (2) Our ATG-index typically consumes around 10 times the memory required to store the original graph. This suggests that ATG-index is capable of scaling to extremely large graphs on modern servers (e.g., with 512 GB of memory).

**4. Index construction time.** As shown in Figure 12, ATGIC is up to two orders of magnitude faster than ASFIC. For the larger datasets, ASFIC cannot finish the index construction process in three days, since it needs to calculate the MSFs for all different $k$ values and start time. We also report the actual values of $\overline{c}$ on all datasets in table 3, which are very small, and often less than 10.

**5. Scalability test.** We test the scalability of our two indexes on the first 10 graphs. Specifically, for each graph, we first randomly select 25%, 50%, 75%, and 100% of its edges and then obtain four sub-temporal graphs induced by these edges, respectively. Table 5 reports the time and space costs of our two index-based solutions for different graphs. We can observe that the ATGIC and ATG-index scales better than ASFIC and ASF-index respectively because of their optimized time and space complexities. If the index exceeds

**Table 3: Dataset Statistics.**

| Dataset | Type | $n$ | $m$ | $t_{max}$ | $\bar{c}$ | $\overline{m}$ | $n \cdot t_{max}$ | $k_{max}$ |
|---|---|---|---|---|---|---|---|---|
| CollegeMsg (CM) | Communication | 1,899 | 59,835 | 6,856 | 6.060 | 31,870 | 13,019,544 | 20 |
| Slashdot (SL) | Communication | 51,083 | 140,778 | 620 | 3.819 | 51,082 | 31,671,460 | 14 |
| Topology (TO) | Computer | 34,761 | 171,403 | 733 | 1.437 | 10,262 | 25,479,813 | 63 |
| Email (ET) | Communication | 986 | 332,334 | 3,702 | 2.892 | 71,769 | 3,650,172 | 34 |
| AskUbuntu (AU) | Online Contact | 159,316 | 964,437 | 6,997 | 3.414 | 147,518 | 1,114,734,052 | 48 |
| Enron (EN) | Communication | 87,273 | 1,148,072 | 38,568 | 3.651 | 247,523 | 3,365,945,064 | 53 |
| SuperUser (SU) | Online Contact | 194,085 | 1,143,339 | 1,149 | 3.510 | 162,510 | 223,003,665 | 61 |
| YouTube (YT) | Online Social | 3,223,589 | 9,375,374 | 203 | 1.168 | 485,468 | 654,388,567 | 88 |
| DBLP (DP) | Collaboration | 1,824,701 | 29,487,744 | 77 | 0.055 | 71,769 | 140,501,977 | 286 |
| Flickr (FL) | Online Social | 2,302,926 | 33,140,017 | 103 | 1.340 | 428,499 | 308,591,950 | 600 |
| Patent (PT) | Communication | 12,214 | 41,916 | 891 | 3.098 | 8,774 | 10,882,674 | 26 |
| S-DBLP (SD) | Collaboration | 28,085 | 236,894 | 27 | 0.401 | 10,301 | 758,295 | 36 |
| School (SC) | Communication | 327 | 188,508 | 7,375 | 2.936 | 54,720 | 2,411,625 | 24 |

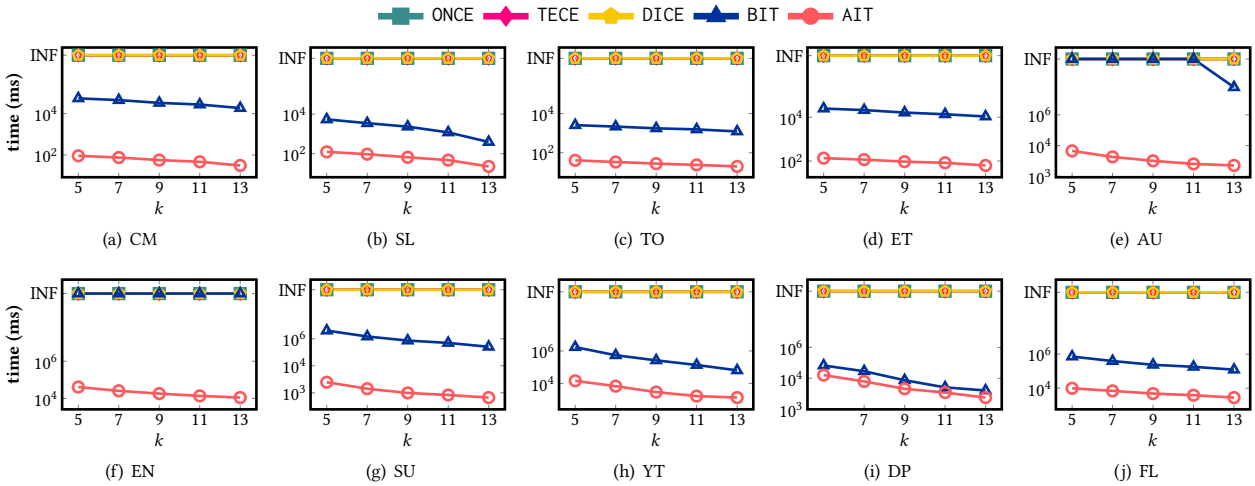



**Figure 9: Effect of $k$ on the efficiency of TDC search algorithms.**

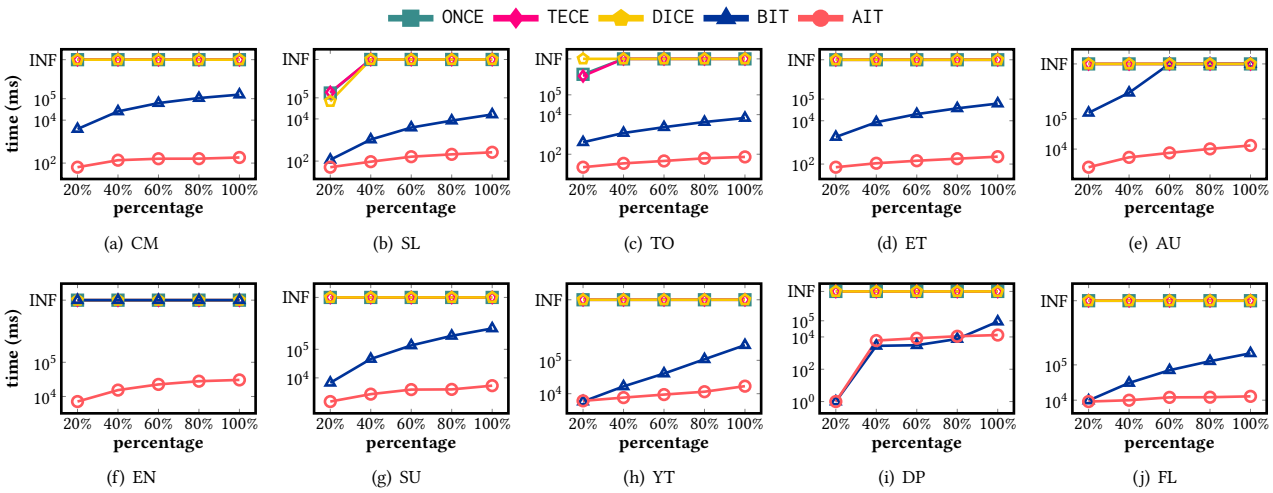



**Figure 10: Effect of the size of time windows on the efficiency of TDC search algorithms.**

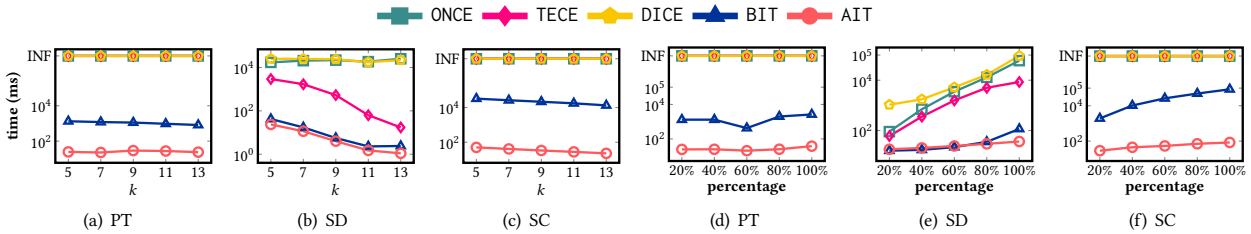



**Figure 11: Efficiency results of TDC search algorithms on PT, SD, and SC graphs.**

**Table 4: Dataset and index sizes (MB).**

| Dataset | graph size | ASF-index | ATG-index |
|---|---|---|---|
| CM | 1.1 | 374.9 | 17.8 |
| SL | 1.6 | 416.1 | 16.9 |
| TO | 2.0 | 95.8 | 19.2 |
| ET | 5.3 | 502.2 | 69.0 |
| AU | 22.8 | TLE | 134.4 |
| EN | 24.7 | TLE | 380.1 |
| SU | 34.2 | 5534.1 | 291.3 |
| YT | 245.2 | TLE | 1276.9 |
| DP | 749.2 | TLE | 624.5 |
| FL | 859.8 | TLE | 7177.0 |
| PT | 0.6 | 389.1 | 5.1 |
| SD | 3.2 | 45.0 | 6.8 |
| SC | 2.2 | 350.1 | 31.3 |

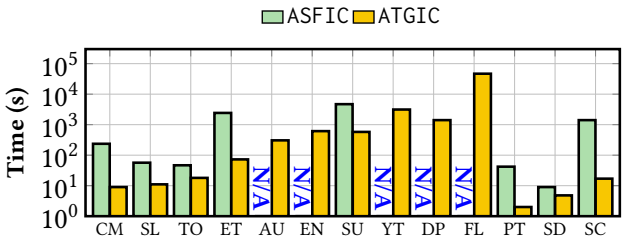



**Figure 12: Index building time across all datasets.**

the memory limit, we denote memory usage as **OOM** and running time as **N/A**, respectively. For large datasets, ASF-index exceeds memory limits when handling only 25% of the graph's edges, while ATG-index consumes significantly less memory. Moreover, ASF-index encounters out-of-memory (OOM) issues in approximately 50% of the cases, limiting its applicability to real-world temporal graphs. In contrast, the largest ATG-index instance requires only 7.1GB of memory, which is considerably smaller. This demonstrates that our advanced ATG-index is highly scalable and can be applied to even larger temporal graphs in the future.

## 6.3 Effectiveness evaluation

We analyze the effectiveness of TDCs on real-world temporal graphs.

**1. Community duration.** We compare the community duration of the communities found by our TDC search solution and time-range $k$-core search solution [86], which is the most relevant work and finds the communities by identifying all distinct temporal $k$-core, but does not consider the community's duration. Specifically, we consider five datasets, and for each of them, we randomly generate 200 queries and then run TDC queries and time-range $k$-core queries. Next, we compute the average community duration value of communities obtained by them. We report the results in Table 6. Clearly, communities of TDC queries always have higher community duration values than communities of time-range $k$-core queries, indicating that our TDC search solution is able to find communities with high community duration.

**2. Event detection.** We consider the Reddit temporal graph [35] from 2006, a well-known online social media platform. Each vertex represents a user, and each interaction corresponds to a reply from one user to another. We run 100 queries for temporal $k$-core, distinct $k$-core, QTC, and TDC with $k = 12$, varying the time windows from Feb 24 to Nov 16, each of length 600 hours. The average community durations for the four methods are shown in Figure 13. In addition, we collect the "major" events from the Wikipedia website [80] and summarize them into Table 7. We observe that among all models, only TDC clearly reflects the impact of major events: we observe significant drops in TDC community duration that align closely with these events, indicating periods of rapid community change and increased new user engagement. In contrast, the durations produced by the other three models remain largely unchanged, failing to capture both these important structural fluctuations and their strong correlation with real-world events.

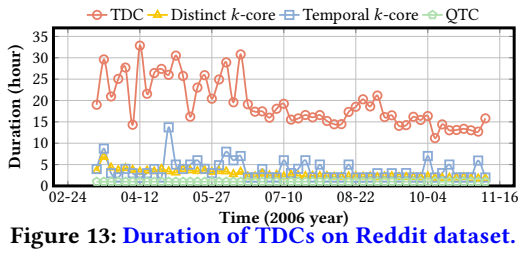



**Figure 13: Duration of TDCs on Reddit dataset.**

**3. Community quality.** In this experiment, we evaluate the quality of communities obtained by different temporal community search methods. Specifically, we compare temporal $k$-core [90], distinct $k$-core [86], and the state-of-the-art method QTCS [50] on three real-world temporal graphs: Patent, S-DBLP, and School, all with ground-truth communities [52]. Those statistics are also shown in Table 3. As shown in Table 10, our TDC model typically produces higher-quality communities, as indicated by higher F1-scores compared to the competitors. This is mainly because TDC explicitly captures the temporal duration of communities. Additionally, on the School dataset, QTCS achieves slightly higher community quality. However, it incurs significantly higher computational costs than the other three methods. In contrast, TDC offers a strong trade-off between efficiency and accuracy, making it a suitable choice when real-time response is required.

**Table 5: Scalability test of indexing time and size.**

| Metrics | Construction time ($ms$) | | | | | | | | Index space cost (bytes) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Indices | ASFIC | | | | ATGIC | | | | ASF-index | | | | ATG-index | | | |
| | 25% | 50% | 75% | 100% | 25% | 50% | 75% | 100% | 25% | 50% | 75% | 100% | 25% | 50% | 75% | 100% |
| CM | 26K | 84K | 153K | 236K | 1.1K | 3.6K | 6.2K | 9K | 148M | 256M | 325M | 379M | 5.2M | 11M | 16M | 19M |
| SL | 4.5K | 15K | 30K | 56K | 0.5K | 2.5K | 6.6K | 11K | 116M | 225M | 325M | 422M | 2.9M | 8.4M | 14M | 21M |
| TO | 7K | 19K | 32K | 46K | 1.8K | 5.9K | 11K | 18K | 27M | 53M | 75M | 95M | 4M | 9.4M | 14M | 19M |
| ET | 333K | 887K | 1.5M | 2.4M | 12K | 30K | 51K | 72K | 285M | 397M | 465M | 502M | 25M | 44M | 58M | 69M |
| AU | 1M | N/A | N/A | N/A | 15K | 79K | 171K | 306K | 7.2G | OOM | OOM | OOM | 34M | 91M | 151M | 209M |
| EN | N/A | N/A | N/A | N/A | 52K | 167K | 304K | 614K | OOM | OOM | OOM | OOM | 93M | 188M | 271M | 380M |
| SU | 310K | 982K | 1.9M | 4.6M | 37K | 168K | 332K | 578K | 1.8G | 3.2G | 4.5G | 5.5G | 54M | 135M | 215M | 291M |
| YT | 470K | N/A | N/A | N/A | 154K | 680K | 1.7M | 3.1M | 2.6G | OOM | OOM | OOM | 157M | 458M | 839M | 1.2G |
| DP | 2.5M | N/A | N/A | N/A | 383K | 783K | 1.1M | 1.4M | 3.7G | OOM | OOM | OOM | 292M | 483M | 587M | 624M |
| FL | N/A | N/A | N/A | N/A | 5M | 24M | 47M | 63M | OOM | OOM | OOM | OOM | 1.6G | 3.6G | 5.5G | 7.1G |

**Table 6: Duration values of communities.**

| | CM | SL | TO | ET | AU |
|---|---|---|---|---|---|
| Time-range $k$-core | 16.66 | 1.29 | 2.44 | 4.53 | 1.12 |
| Durable community | **171.86** | **7.16** | **19.40** | **91.62** | **10.49** |

**Table 7: Major global events of 2006.**

| Month | Date | Description |
|---|---|---|
| March | March 10 | Mars Orbiter achieves Mars orbit |
| March | March 10 | Michelle Bachelet elected in Chile |
| March | March 15 | UN creates Human Rights Council |
| March | March 21 | Twitter (later rebranded to X) was launched |
| April | April 4 | Faddoul Brothers tragedy in Venezuela |
| April | April 11 | Venus Express enters Venus orbit |
| April | April 11 | Iran starts uranium enrichment |
| April | April 20 | Iran-Russia uranium deal fails |
| May | May 17 | Genome Project publishes final sequence |
| May | May 18-20 | Lordi wins Eurovision Song Contest |
| May | May 27 | Yogyakarta earthquake kills thousands |
| June | June 3 | Montenegro and Serbia declare independence |
| June | June 9 | 2006 FIFA World Cup |
| June | June 28 | Israel responds to Gaza rocket fire |
| June | June 28 | US forces withdraw from Iceland |

**Table 8: Comparison of running time and community quality across three datasets; The best and second-best results are marked in bold and underlined, respectively.**

| Method | Patent | | S-DBLP | | School | |
|---|---|---|---|---|---|---|
| | Time | F1-score | Time | F1-score | Time | F1-score |
| Temporal $k$-core [90] | <u>0.023</u> | 0.102 | **0.001** | **0.015** | **0.010** | 0.288 |
| Distinct $k$-core [86] | **0.020** | 0.101 | **0.001** | <u>0.011</u> | **0.010** | 0.780 |
| QTCS [49] | 1.399 | <u>0.107</u> | 3.971 | 0.010 | 554.3 | **0.859** |
| TDC | 0.057 | **0.118** | <u>0.002</u> | **0.015** | <u>0.012</u> | <u>0.811</u> |

## 7 RELATED WORKS

In this section, we first review the existing works of community search (CS), including CS over static and temporal graphs, and then briefly review the related works of community detection.

**CS over static graphs.** CS aims to query densely connected subgraphs containing a specific vertex or a set of vertices [17, 22, 23, 69, 87]. To measure the structure cohesiveness of a community, people often use the cohesive subgraph models [22], like $k$-core [3, 5], $k$-truss [14, 93], $k$-clique [16, 91] and $k$-edge connected component [7, 34]. A representative group of CS works is based on the $k$-core model. For example, in [17, 69], the metric of minimum degree used in $k$-core is used for CS. Another group of CS works uses the $k$-truss [14, 93]. For example, in [36, 38], the $k$-truss-based model is used for CS. Besides, many CS works have considered vertices' attributes (e.g., [10, 22, 37]). In addition, some works have considered vertices' importance values and studied the problem of influential CS [4, 11, 37, 43–46, 54, 82]. Some works have studied CS over heterogeneous information networks [19, 23, 30, 39, 59, 76, 87, 94]. For example, Fang et al and Zhou et al studied CS over heterogeneous information networks by using the $(k, \mathcal{P})$-core model [23, 96]; Jian et al. [39] searched communities with vertices of multiple types by using relational constraints.

**CS over temporal graphs.** Recently, many works have studied CS over temporal graphs. For example, Galimberti et al. [29] introduced temporal core decomposition with span-cores and proposed algorithms for maximal span-cores. Yu et al. [90] studied historical $k$-cores queries, while Yang et al. [86] identified all distinct $k$-cores within sub-windows. Historically connected component [68, 83] and structural diversity [9] queries were explored. Wu et al. [81] introduced the $(k, h)$-core, with Bai et al. [2] focusing on its maintenance. Li et al. [47] proposed the $(\theta, \tau)$-persistent $k$-core, and Li et al. [48] discussed continual cohesive subgraphs. Chu et al. [12] examined subgraphs with bursting density, while Qin et al. [60] introduced maximal bursting cores. Qin et al. [61] studied periodic $k$-cores. The temporal $k$-truss-based communities also have been studied [53, 85]. Ma et al. [55] worked on finding temporal community on weighted temporal graphs.

**Community detection.** In the literature, various community detection methods have been proposed [24, 26, 58], such as spectral clustering [18, 79, 88], consensus clustering [32, 74], modularity-based approaches [13, 20, 57, 58], statistical inference-based approaches [41, 62], and structural similarity-based approaches [6, 8, 51, 63, 75, 78, 84, 92]. In addition, the link-based analysis methods [25, 56] are proposed for detecting network communities. Some recent works [65, 66, 70–73, 99] have studied community detection in heterogeneous information networks. Besides, detecting community in temporal graphs has also been mainly studied [28, 52, 89].

## 8 CONCLUSIONS

In this paper, we investigate the problem of temporal durable community (TDC) search in large temporal graphs, aiming to identify communities with the longest duration. To tackle this problem, we first introduce a novel online algorithm that utilizes binary search. We then propose a novel index, called `ASF-index`, to support frequent TDC queries with various parameters. Furthermore, we propose some optimization techniques to improve the efficiency of index construction and reduce its space cost, thus obtaining our advanced index, `ATG-index`. Based on `ATG-index`, we develop two query algorithms that reduce redundant computation. Extensive experiments validate the efficiency and effectiveness of our proposed solutions. In the future, we will explore how to find the TDC on the interval-based (lifespan) temporal graph.

# REFERENCES


[1] 2006. Stanford Network Analysis. https://networkrepository.com/network-data.php.

[2] Wen Bai, Yadi Chen, and Di Wu. 2020. Efficient temporal core maintenance of massive graphs. *Information Sciences* 513 (2020), 324–340.

[3] Vladimir Batagelj and Matjaz Zaversnik. 2003. An o (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049* (2003).

[4] Fei Bi, Lijun Chang, Xuemin Lin, and Wenjie Zhang. 2018. An Optimal and Progressive Approach to Online Search of Top-K Influential Communities. *Proc. VLDB Endow.* 11, 9 (2018), 1056–1068.

[5] Francesco Bonchi, Arijit Khan, and Lorenzo Severini. 2019. Distance-generalized core decomposition. In *proceedings of the 2019 international conference on management of data*. 1006–1023.

[6] Lijun Chang, Wei Li, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2016. pSCAN: Fast and Exact Structural Graph Clustering. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, Helsinki, Finland, 253–264. https://doi.org/10.1109/ICDE.2016.7498245

[7] Lijun Chang, Xuemin Lin, Lu Qin, Jeffrey Xu Yu, and Wenjie Zhang. 2015. Index-based optimal algorithms for computing steiner components with maximum connectivity. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 459–474.

[8] Yulin Che, Shixuan Sun, and Qiong Luo. 2018. Parallelizing Pruning-based Graph Structural Clustering. In *Proceedings of the 47th International Conference on Parallel Processing (ICPP '18)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3225058.3225063

[9] Kaiyu Chen, Dong Wen, Wenjie Zhang, Ying Zhang, Xiaoyang Wang, and Xuemin Lin. 2024. Querying Structural Diversity in Streaming Graphs. *Proceedings of the VLDB Endowment* 17, 5 (2024), 1034–1046.

[10] Lu Chen, Chengfei Liu, Rui Zhou, Jianxin Li, Xiaochun Yang, and Bin Wang. 2018. Maximum co-located community search in large scale social networks. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1233–1246.

[11] Shu Chen, Ran Wei, Diana Popova, and Alex Thomo. 2016. Efficient computation of importance based communities in web-scale networks using a single machine. In *CIKM*. 1553–1562.

[12] Lingyang Chu, Yanyan Zhang, Yu Yang, Lanjun Wang, and Jian Pei. 2019. Online density bursting subgraph detection from temporal graphs. *Proceedings of the VLDB Endowment* 12, 13 (2019), 2353–2365.

[13] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. 2004. Finding Community Structure in Very Large Networks. *Physical Review E* 70, 6 (Dec. 2004), 066111. https://doi.org/10.1103/PhysRevE.70.066111 arXiv:cond-mat/0408187

[14] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report* 16, 3.1 (2008), 1–29.

[15] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.

[16] 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.

[17] 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.

[18] W. E. Donath and A. J. Hoffman. 1973. Lower Bounds for the Partitioning of Graphs. *IBM Journal of Research and Development* 17, 5 (Sept. 1973), 420–425. https://doi.org/10.1147/rd.175.0420

[19] Zheng Dong, Xin Huang, Guorui Yuan, Hengshu Zhu, and Hui Xiong. 2021. Butterfly-core community search over labeled graphs. *Proceedings of the VLDB Endowment* (2021).

[20] J. Duch and A. Arenas. 2005. Community Detection in Complex Networks Using Extremal Optimization. *Physical Review E* 72, 2 (Aug. 2005), 027104. https://doi.org/10.1103/PhysRevE.72.027104 arXiv:cond-mat/0501368

[21] Yixiang Fang, CK Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective community search for large attributed graphs. *Proceedings of the VLDB Endowment* (2016).

[22] 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 (2020), 353–392.

[23] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and efficient community search over large heterogeneous information networks. *Proceedings of the VLDB Endowment* 13, 6 (2020), 854–867.

[24] Santo Fortunato. 2010. Community Detection in Graphs. *Physics Reports* 486, 3 (Feb. 2010), 75–174. https://doi.org/10.1016/j.physrep.2009.11.002

[25] Santo Fortunato. 2010. Community detection in graphs. *Physics reports* 486, 3-5 (2010), 75–174.

[26] Santo Fortunato and Darko Hric. 2016. Community Detection in Networks: A User Guide. *Physics Reports* 659 (Nov. 2016), 1–44. https://doi.org/10.1016/j.physrep.2016.09.002

[27] Dongqi Fu, Dawei Zhou, and Jingrui He. 2020. Local motif clustering on time-evolving graphs. In *Proceedings of the 26th ACM SIGKDD International conference on knowledge discovery & data mining*. 390–400.

[28] Dongqi Fu, Dawei Zhou, Ross Maciejewski, Arie Croitoru, Marcus Boyd, and Jingrui He. 2023. Fairness-aware clique-preserving spectral clustering of temporal graphs. In *Proceedings of the ACM Web Conference 2023*. 3755–3765.

[29] Edoardo Galimberti, Alain Barrat, Francesco Bonchi, Ciro Cattuto, and Francesco Gullo. 2018. Mining (maximal) span-cores from temporal networks. In *Proceedings of the 27th ACM international Conference on Information and Knowledge Management*. 107–116.

[30] Edoardo Galimberti, Francesco Bonchi, and Francesco Gullo. 2017. Core decomposition and densest subgraph in multilayer networks. In *CIKM*. 1807–1816.

[31] Edoardo Galimberti, Martino Ciaperoni, Alain Barrat, Francesco Bonchi, Ciro Cattuto, and Francesco Gullo. 2020. Span-core decomposition for temporal networks: Algorithms and applications. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15, 1 (2020), 1–44.

[32] Andrey Goder and Vladimir Filkov. 2008. Consensus Clustering Algorithms: Comparison and Refinement. In *2008 Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*. Society for Industrial and Applied Mathematics, 109–117. https://doi.org/10.1137/1.9781611972887.11

[33] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. 47–57.

[34] Jiafeng Hu, Xiaowei Wu, Reynold Cheng, Siqiang Luo, and Yixiang Fang. 2017. On minimal steiner maximum-connected subgraph queries. *IEEE Transactions on Knowledge and Data Engineering* 29, 11 (2017), 2455–2469.

[35] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. 2024. Temporal graph benchmark for machine learning on temporal graphs. *Advances in Neural Information Processing Systems* 36 (2024).

[36] 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.

[37] Xin Huang and Laks VS Lakshmanan. 2017. Attribute-driven community search. *Proceedings of the VLDB Endowment* 10, 9 (2017), 949–960.

[38] Xin Huang, Laks VS Lakshmanan, Jeffrey Xu Yu, and Hong Cheng. 2015. Approximate Closest Community Search in Networks. *Proceedings of the VLDB Endowment* 9, 4 (2015).

[39] Xun Jian, Yue Wang, and Lei Chen. 2020. Effective and efficient relational community detection and search in large dynamic heterogeneous information networks. *Proceedings of the VLDB Endowment* 13, 10 (2020), 1723–1736.

[40] Konect. 2006. Konect. http://konect.cc/networks/.

[41] Johan H. Koskinen and Tom A.B. Snijders. 2007. Bayesian Inference for Dynamic Social Network Data. *Journal of Statistical Planning and Inference* 137, 12 (Dec. 2007), 3930–3938. https://doi.org/10.1016/j.jspi.2007.04.011

[42] Laks VS Lakshmanan. 2022. On a Quest for Combating Filter Bubbles and Misinformation. In *SIGMOD*. 2–2.

[43] Rong-Hua Li, Lu Qin, Fanghua Ye, Jeffrey Xu Yu, Xiaokui Xiao, Nong Xiao, and Zibin Zheng. 2018. Skyline Community Search in Multi-valued Networks. In *SIGMOD*. ACM, 457–472.

[44] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2015. Influential Community Search in Large Networks. *Proc. VLDB Endow.* 8, 5 (2015), 509–520.

[45] Rong-Hua Li, Lu Qin, Fanghua Ye, Guoren Wang, Jeffrey Xu Yu, Xiaokui Xiao, Nong Xiao, and Zibin Zheng. 2020. Finding skyline communities in multi-valued networks. *The VLDB Journal* 29, 6 (2020), 1407–1432.

[46] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2017. Finding influential communities in massive networks. *The VLDB Journal* 26, 6 (2017), 751–776.

[47] 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)*. IEEE, 797–808.

[48] Yuan Li, Jinsheng Liu, Huiqun Zhao, Jing Sun, Yuhai Zhao, and Guoren Wang. 2021. Efficient continual cohesive subgraph search in large temporal graphs. *World Wide Web* 24 (2021), 1483–1509.

[49] Longlong Lin, Pingpeng Yuan, Rong-Hua Li, Jifei Wang, Ling Liu, and Hai Jin. 2021. Mining stable quasi-cliques on temporal networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52, 6 (2021), 3731–3745.

[50] 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.

[51] Kaixin Liu, Sibo Wang, Yong Zhang, and Chunxiao Xing. 2023. An efficient algorithm for distance-based structural graph clustering. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–25.

[52] Meng Liu, Yue Liu, Ke Liang, Wenxuan Tu, Siwei Wang, Sihang Zhou, and Xinwang Liu. 2023. Deep temporal graph clustering. *arXiv preprint arXiv:2305.10738* (2023).

[53] Quintino Francesco Lotito and Alberto Montresor. 2020. Efficient algorithms to mine maximal span-trusses from temporal graphs. *arXiv preprint arXiv:2009.01928* (2020).

[54] Wensheng Luo, Xu Zhou, Jianye Yang, Peng Peng, Guoqing Xiao, and Yunjun Gao. 2020. Efficient approaches to top-r influential community search. *IEEE Internet of Things Journal* 8, 16 (2020), 12650–12657.

[55] Shuai Ma, Renjun Hu, Luoshu Wang, Xuelian Lin, and Jinpeng Huai. 2019. An efficient approach to finding dense temporal subgraphs. *IEEE Transactions on Knowledge and Data Engineering* 32, 4 (2019), 645–658.

[56] Mark EJ Newman and Michelle Girvan. 2004. Finding and evaluating community structure in networks. *Physical review E* 69, 2 (2004), 026113.

[57] M. E. J. Newman. 2004. Fast Algorithm for Detecting Community Structure in Networks. *Physical Review E* 69, 6 (June 2004), 066133. https://doi.org/10.1103/PhysRevE.69.066133 arXiv:cond-mat/0309508

[58] M. E. J. Newman and M. Girvan. 2004. Finding and Evaluating Community Structure in Networks. *Physical Review E* 69, 2 (Feb. 2004), 026113. https://doi.org/10.1103/PhysRevE.69.026113

[59] Lianpeng Qiao, Zhiwei Zhang, Ye Yuan, Chen Chen, and Guoren Wang. 2021. Keyword-centric community search over large heterogeneous information networks. In *Database Systems for Advanced Applications: 26th International Conference, DASFAA 2021, Taipei, Taiwan, April 11–14, 2021, Proceedings, Part I*. Springer, 158–173.

[60] 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.

[61] 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.

[62] Joerg Reichardt and Stefan Bornholdt. 2006. Statistical Mechanics of Community Detection. *Physical Review E* 74, 1 (July 2006), 016110. https://doi.org/10.1103/PhysRevE.74.016110 arXiv:cond-mat/0603718

[63] Boyu Ruan, Junhao Gan, Hao Wu, and Anthony Wirth. 2021. Dynamic Structural Clustering on Graphs. In *Proceedings of the 2021 International Conference on Management of Data*. ACM, Virtual Event China, 1491–1503. https://doi.org/10.1145/3448016.3452828

[64] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. 2010. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*. 851–860.

[65] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. 2016. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (2016), 17–37.

[66] Chuan Shi, Ran Wang, Yitong Li, Philip S Yu, and Bin Wu. 2014. Ranking-based clustering on general heterogeneous information networks by network projection. In *CIKM*. 699–708.

[67] Daniel Dominic Sleator and Robert Endre Tarjan. 1985. Self-adjusting binary search trees. *Journal of the ACM (JACM)* 32, 3 (1985), 652–686.

[68] Jingyi Song, Dong Wen, Lantian Xu, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2024. On Querying Historical Connectivity in Temporal Graphs. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–25.

[69] 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.

[70] Yizhou Sun, Charu C Aggarwal, and Jiawei Han. 2012. Relation strength-aware clustering of heterogeneous information networks with incomplete attributes. *arXiv preprint arXiv:1201.6563* (2012).

[71] Yizhou Sun, Jiawei Han, Peixiang Zhao, Zhijun Yin, Hong Cheng, and Tianyi Wu. 2009. Rankclus: integrating clustering with ranking for heterogeneous information network analysis. In *EDBT*. 565–576.

[72] Yizhou Sun, Brandon Norick, Jiawei Han, Xifeng Yan, Philip S Yu, and Xiao Yu. 2013. Pathselclus: Integrating meta-path selection with user-guided object clustering in heterogeneous information networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 7, 3 (2013), 1–23.

[73] Yizhou Sun, Yintao Yu, and Jiawei Han. 2009. Ranking-based clustering of heterogeneous information networks with star network schema. In *KDD*. 797–806.

[74] A. Topchy, A.K. Jain, and W. Punch. 2005. Clustering Ensembles: Models of Consensus and Weak Partitions. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27, 12 (Dec. 2005), 1866–1881. https://doi.org/10.1109/TPAMI.2005.237

[75] Tom Tseng, Laxman Dhulipala, and Julian Shun. 2021. Parallel Index-Based Structural Graph Clustering and Its Approximation. In *Proceedings of the 2021 International Conference on Management of Data*. 1851–1864. https://doi.org/10.1145/3448016.3457278 arXiv:2012.11188 [cs]

[76] Ruby W Wang and Y Ye Fred. 2019. Simplifying Weighted Heterogeneous networks by extracting h-Structure via s-Degree. *Scientific reports* 9, 1 (2019), 1–8.

[77] Dong Wen, Yilun Huang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2020. spanning span-reachability queries in large temporal graphs. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1153–1164.

[78] Dong Wen, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2019. Efficient Structural Graph Clustering: An Index-Based Approach. *The VLDB Journal* 28, 3 (June 2019), 377–399. https://doi.org/10.1007/s00778-019-00541-4

[79] Scott White and Padhraic Smyth. 2005. A Spectral Clustering Approach To Finding Communities in Graphs. In *Proceedings of the 2005 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, 274–285. https://doi.org/10.1137/1.9781611972757.25

[80] Wikipedia. 2006. Wikipedia event. https://en.wikipedia.org/wiki/2006.

[81] Huanhuan Wu, James Cheng, Yi Lu, Yiping Ke, Yuzhen Huang, Da Yan, and Hejun Wu. 2015. Core decomposition in large temporal graphs. In *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 649–658.

[82] Yanping Wu, Jun Zhao, Renjie Sun, Chen Chen, and Xiaoyang Wang. 2021. Efficient personalized influential community search in large networks. *Data Science and Engineering* 6, 3 (2021), 310–322.

[83] Haoxuan Xie, Yixiang Fang, Yuyang Xia, Wensheng Luo, and Chenhao Ma. 2023. On querying connected components in large temporal graphs. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27.

[84] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas A. J. Schweiger. 2007. SCAN: A Structural Clustering Algorithm for Networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, San Jose California USA, 824–833. https://doi.org/10.1145/1281192.1281280

[85] Huihui Yang, Chunxue Zhu, Longlong Lin, and Pingpeng Yuan. 2024. Towards Truss-Based Temporal Community Search. *arXiv preprint arXiv:2410.15046* (2024).

[86] Junyong Yang, Ming Zhong, Yuanyuan Zhu, Tieyun Qian, Mengchi Liu, and Jeffrey Xu Yu. 2023. Scalable time-range k-core query on temporal graphs. *Proceedings of the VLDB Endowment* 16, 5 (2023), 1168–1180.

[87] Yixing Yang, Yixiang Fang, Xuemin Lin, and Wenjie Zhang. 2020. Effective and efficient truss computation over large heterogeneous information networks. In *ICDE*. IEEE, 901–912.

[88] Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. 2017. Local Higher-Order Graph Clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. Association for Computing Machinery, New York, NY, USA, 555–564. https://doi.org/10.1145/3097983.3098069

[89] Jingyi You, Chenlong Hu, Hidetaka Kamigaito, Kotaro Funakoshi, and Manabu Okumura. 2021. Robust dynamic clustering for temporal networks. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2424–2433.

[90] Michael Yu, Dong Wen, Lu Qin, Ying Zhang, Wenjie Zhang, and Xuemin Lin. 2021. On querying historical k-cores. *Proceedings of the VLDB Endowment* (2021).

[91] 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.

[92] Fangyuan Zhang and Sibo Wang. 2022. Effective Indexing for Dynamic Structural Graph Clustering. *Proceedings of the VLDB Endowment* 15, 11 (July 2022), 2908–2920. https://doi.org/10.14778/3551793.3551840

[93] Yikai Zhang and Jeffrey Xu Yu. 2019. Unboundedness and efficiency of truss maintenance in evolving graphs. In *SIGMOD*. 1024–1041.

[94] Alexander Zhou, Yue Wang, and Lei Chen. 2020. Finding large diverse communities on networks: the edge maximum k*-partite clique. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2576–2589.

[95] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2009. Graph clustering based on structural/attribute similarities. *Proceedings of the VLDB Endowment* 2, 1 (2009), 718–729.

[96] Yingli Zhou, Yixiang Fang, Wensheng Luo, and Yunming Ye. 2023. Influential community search over large heterogeneous information networks. *Proceedings of the VLDB Endowment* 16, 8 (2023), 2047–2060.

[97] Yingli Zhou, Qingshuo Guo, Yixiang Fang, and Chenhao Ma. 2024. A Counting-based Approach for Efficient k-Clique Densest Subgraph Discovery. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.

[98] Yingli Zhou and etc. Jiang, Yige. 2025. Effective Durable Community Search in Large Temporal Graph (technical report). https://github.com/kiwiHM/VLDB2025_Durable_Temporal_CS_Appendix/blob/main/VLDB2025_Durable_Temporal_CS_Appendix.pdf.

[99] Yang Zhou and Ling Liu. 2013. Social influence based clustering of heterogeneous information networks. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 338–346.

[100] Kaijie Zhu, George Fletcher, and Nikolay Yakovets. 2021. Leveraging temporal and topological selectivities in temporal-clique subgraph query processing. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 672–683.

[101] Kaijie Zhu, George Fletcher, Nikolay Yakovets, Odysseas Papapetrou, and Yuqing Wu. 2019. Scalable temporal clique enumeration. In *Proceedings of the 16th International Symposium on Spatial and Temporal Databases*. 120–129.

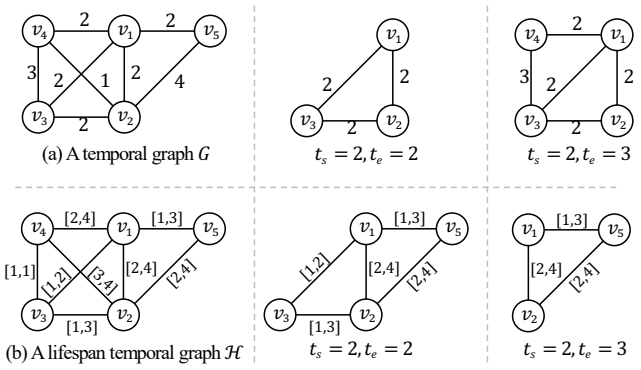



**Figure 15: Illustrating the key properties of temporal $k$-cores on timestamp-based temporal graph and interval-based temporal graph.**

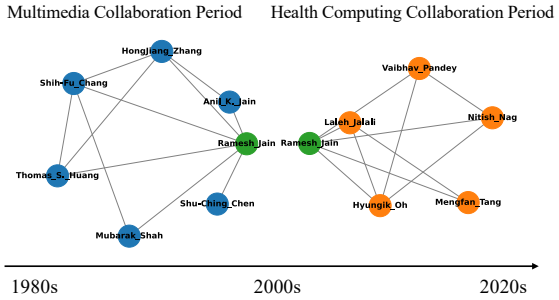



**Figure 14: A case study on DBLP co-authorship network.**

**Table 9: Statistics of datasets: number of nodes ($n$), edges ($m$), maximum timestamp ($t_{\max}$), and number of communities.**

| Dataset | $n$ | $m$ | $t_{\max}$ | # of comminutes |
|---|---|---|---|---|
| Patent | 12,214 | 41,916 | 891 | 6 |
| S-DBLP | 28,085 | 236,894 | 27 | 10 |
| School | 327 | 188,508 | 7,375 | 9 |

**Table 10: Comparison of running time and community quality across three datasets; The best and second-best results are marked in bold and underlined, respectively.**

| | Patent | | S-DBLP | | School | |
|---|---|---|---|---|---|---|
| Method | Time | F1-score | Time | F1-score | Time | F1-score |
| Temporal $k$-core [90] | <u>0.023</u> | 0.102 | **0.001** | **0.015** | **0.010** | 0.288 |
| Distinct $k$-core [86] | **0.020** | 0.101 | **0.001** | <u>0.011</u> | **0.010** | 0.780 |
| QTC [49] | 1.399 | <u>0.107</u> | 3.971 | 0.010 | 554.3 | **0.859** |
| TDC | 0.057 | **0.118** | <u>0.002</u> | **0.015** | <u>0.012</u> | <u>0.811</u> |

# A MORE DISCUSSIONS

## A.1 Case study

We analyze the DBLP co-authorship network by using AIT to identify research groups of "Ramesh Jain", a renowned computer scientist in the multimedia area. Specifically, we set $k = 5$, the query vertex $q$ to "Ramesh Jain", and the query time interval to $[1980, 2020]$. As shown in Figure 14, our TDC model captures Jain's research evolution by identifying two distinct communities: one focused on multimedia from 1980-2000, and another interested in health computing from 2000-2020.

## A.2 Extended to the lifespan temporal graph

In the literature, there are two temporal graph models, which are timestamp-based and interval-based temporal graphs, and they associate each edge with a timestamp [9, 49] and a time interval (lifespan) [100, 101], respectively. Since a single timestamp can be considered a time interval whose start and end timestamps are the same, the interval-based temporal graph can be regarded as an extension of the timestamp-based temporal graph, implying that the timestamp-based temporal graph is more fundamental. Actually, our proposed solutions on timestamp-based temporal graph can also be easily extended to interval-based temporal graph.

Let us briefly explain the key idea of extension: First, we extend the $k$-core model on the timestamp-based temporal graph for the interval-based temporal graph. Given an interval-based temporal graph, its temporal $k$-core can be a subgraph, where each vertex has at least $k$ neighbors and all the temporal edges have a non-empty intersected interval, denoting the valid time interval for this $k$-core. Afterwards, we can easily extend our proposed algorithms for identifying the communities from the interval-based temporal graph. Specifically, given a query time window, to find the TDC on an interval-based temporal graph, an online solution is to first enumerate all sub-windows within the given window, treating them as intersection intervals for constructing the corresponding $k$-core, then compute each duration, and finally select the temporal $k$-core with the longest duration as the TDC.

Regarding the index-based solution, recall that our original indexes rely on a key property of the temporal $k$-core on timestamp-based temporal graph: *Given a fixed start time $t_s$ and an integer $k$, for all the temporal $k$-cores for the intervals $[t_s, t_e]$, when we increase $t_e$ from $t_s$ to $t_{max}$, they can only add new edges and no edge deletes as $t_e$ increases.* However, for interval-based temporal graphs, when considering the intersection interval $[t_s, t_e]$ of all the edges in the temporal $k$-core, the set of edges can only shrink as $t_e$ increases from $t_s$ to $t_{max}$. This is because only edges whose intervals fully cover the interval $[t_s, t_e]$ are included, so some edges may drop out as the interval grows. To tackle this issue, we only need to slightly revise our original indexing algorithm to build the index. Specifically, we first enumerate $t_e$ from $t_{max}$ to $t_1$, and then for each specific $t_e$, we decrease $t_s$ from $t_e$ to $t_1$, during which we progressively derive the $k$-cores on all the possible intervals. The above index can also efficiently support the TDC search.

For example, we illustrate the key properties of temporal $k$-cores on two types of temporal graphs in Figure 15. Consider the timestamp-based and interval-based temporal graphs shown in Figures 15(a) and (b), respectively. Fixing $k = 2$ and $t_s = 2$, we increase $t_e$ from 2 to 3. In the timestamp-based temporal graph, the temporal $k$-core in $[2, 3]$ adds two edges and one vertex compared to $[2, 2]$. In contrast, in the interval-based temporal graph, the temporal $k$-core in $[2, 3]$ removes two edges and one vertex compared to $[2, 2]$.

## A.3 An example of temporal $k$-core

We would like to clarify that, for a temporal $k$-core with a fixed start time $t_s$, the temporal $k$-core can only be merged and never split as the end time $t_e$ increases, unlike the general $k$-core in static graphs. This property arises from the nature of the projected graph: when $t_s$ is fixed, the projected graph over the time window $[t_s, t_e]$ can only gain edges as $t_e$ increases, but never loses any existing edges. Consequently, both vertices and edges are only ever added to

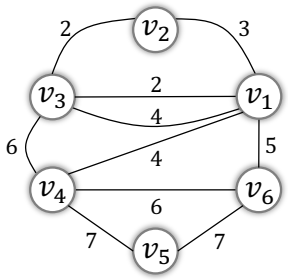



**Figure 16: An example of the temporal graph.**

the temporal $k$-core as $t_e$ increases; there is no removal of vertices or edges from the core. To illustrate this, we provide a detailed example as follows:

Example 7. *Consider the temporal graph shown in Figure 16. We present a chain of projected graphs and their corresponding temporal $k$-cores for $t_s = 2$ and $t_e$ varying from 2 to 7 (i.e., $t_{max}$) in Figure 17. It can be observed that as $t_e$ increases, new edges are incrementally added into the projected graph, and thus the corresponding temporal $k$-core either remains the same or grows larger by merging new vertices or edges (marked in blue color). No edges or vertices are ever deleted from the core during this process. This behavior directly follows from the definition of the temporal $k$-core on the projected graph.*

# B ADDITIONAL PROOFS OF LEMMAS AND THEOREMS

Lemma 3.1 The total time cost of ONCE is $O(t_{max}^2 \cdot \log t_{max} \cdot (n+m))$, where $n$, $m$ and $t_{max}$ denotes the number of vertices, edges and the number of distinct time slots in a temporal graph $G$, respectively.

Proof. The proof of Lemma 3.1 is straightforward. There are at most $t_{max}^2$ time windows to check. For each time window, ONCE requires $O(m)$ time to compute the temporal $k$-core containing $q$ and at most $O(\log t_{max} \cdot (n+m))$ time to determine the duration. Therefore, the lemma holds. □

Lemma 4.4 The community duration of the temporal $k$-core $S_k[t_l, t_r]$ can be calculated by using the active time of its key edge subtraction $(t_r + 1)$.

Proof. We prove this by contradiction. Denote $e^*$ as the key edge of $S_k[t_s, t_e]$. Assume there exists a time window $[t_s, t'_e]$ such that $\exists u \in S_k[t_s, t'_e] \wedge u \notin S_k[t_s, t_e]$, where $t'_e < t(e^*)$. For an edge $e' = (u, v)$ with $v \in S_k[t_s, t_e]$, we have $\mathcal{L}_{t_s}(e')_k < t'_e < \mathcal{L}_{t_s}(e^*)_k$, which contradicts the definition of the key edge. Hence, this lemma can be proved. □

Lemma 4.7 Given a temporal graph $G$, an integer $k$, an edge $e$=$(u, v)$ and two start times $t_s$, $t'_s$, if $t_s < t'_s$, then $\mathcal{L}_{t_s}(e)_k \leq \mathcal{L}_{t'_s}(e)_k$.

Proof. Given that $t_s < t'_s$, all edges in the projected graph of $[t'_s, \mathcal{L}_{t_s}(e)_k]$ must exist in that of $[t_s, \mathcal{L}_{t_s}(e)_k]$. The core number of $u$ and $v$ over $[t'_s, \mathcal{L}_{t_s}(e)_k - 1]$ is smaller than $k$ based on Definition 4.1. Therefore, the lemma holds. □

Lemma 4.10 Given a temporal graph $G$, a start time $t_s$, and an integer $k$, the ASF-index of the updated graph $G'$, by decreasing the active times of some existing edges in $G$, is equivalent to directly inserting these edges into the ASF-index of $G$.

Proof. Clearly, decreasing the weight of an edge is equivalent to first adding the edge with the new, lower weight and then removing the edge with the old, higher weight. Here, we denote the edges in $E$ with lower weights as $E'$. Hence, the ASF-index after inserting $E'$ into $G$ is equivalent to inserting all edges from $E'$ into $\mathcal{F}[k, t]$, as shown in lemma 4.9. When we delete all edges $E$ from $G$, there are two types of edges in $E$: those not in the index and those in the index. For the former, deleting them does not affect the index. For the latter, during the insertion process, we have already deleted them by inserting the same edges with smaller weights. □

Lemma 4.12 The total time cost of ATGIC is $O(k_{\max} \cdot m \cdot \bar{c} \log n)$, where $\bar{c}$ denotes the average number of changes for the active time of an edge. Typically, $\bar{c} \ll t_{max}$.

Proof. For each $k$, and start time $t_s$ ($t_s > 1$), AGIC needs to insert all the edges that active times decreased and edges with the timestamps $t_s$ into the index of the previous start time (i.e., $t_s$ - 1). We use $\bar{c}$ to denote the average number of changes for the active time of one edge, so there are at most $((\bar{c}+1) \cdot m)$ edges that would be inserted into the index. In addition, inserting an edge into the index takes $O(\log n)$ time [33, 67]. Hence, the total construction time is $O(k_{\max} \cdot m \cdot \bar{c} \log n)$. □

Lemma 4.13 The ATG-index requires $O(k_{\max} \cdot \overline{m})$ space, where $\overline{m}$ represents the number of all edges in the index, which be calculated by multiplying the number of edges by the number of times each edge appears in the index. Note that $\overline{m}$ is bounded by $m \cdot \bar{c}$ and $\overline{m} \ll n \cdot t_{max}$, as shown in our experimental results.

Proof. For each $k$, our ATG-index is a graph with $n$ vertices and $\overline{m}$ edges ($n < \overline{m}$), so the index costs $O(k_{\max} \cdot \overline{m})$ memory. Note that when an edge's active time changes, it appears multiple times in the index. This indicates that the number of times each edge appears in the index is always less than or equal to $\bar{c}$. □

Lemma 5.1 The total time cost of BIT is $O(t_{max} \cdot \min\{n, t_{max}\} \cdot n)$, where $n$ denotes the number of vertices in a temporal graph $G$.

Proof. There are at most $t_{max} \cdot \{n, t_{max}\}$ time windows, since for each start time $t_s$ only the timestamps of the edges in $\mathcal{F}[t_s, k]$ need to be considered as the end time. For each time window, BIT needs $O(n)$ time to calculate the temporal $k$-core containing $q$ and its corresponding community duration in the index $\mathcal{F}[t_l, k]$. Hence, the above lemma holds. □

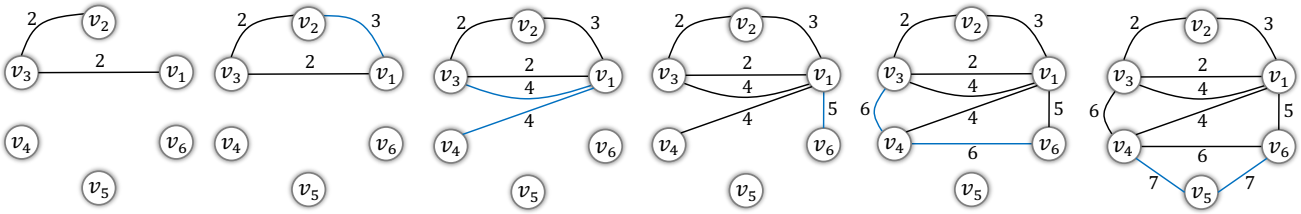
Lemma 5.4 Given a temporal $k$-core $S_k[t_l, t_r]$, and its key edge $e^* = (u, v)$, assuming $u \in S_k[t_l, t_r]$ and $v \notin S_k[t_l, t_r]$, $v$ must be included in $S_k[t_l, t'_r]$, if $t'_r \geq \mathcal{L}_{t_l}(e^*)_k$.

Proof. We prove this lemma by contradiction. Suppose there exists an edge $e^* = (u, v)$ and the end time $t'_e$ satisfies the conditions, but $v$ is not in $S_k[t_s, t'_e]$. This implies that the smallest active time required to include $v$ in $S_k[t_s, t'_e]$ is greater than $t'_e$, contradicting the initial assumption. Therefore, the lemma holds. □

Theorem 5.5 The algorithm 5 can correctly return the TDC.

,

➢ **Projected Graph over each time interval with $t_s = 2$**




➢ **Temporal 2-core over each time interval with $t_s = 2$**

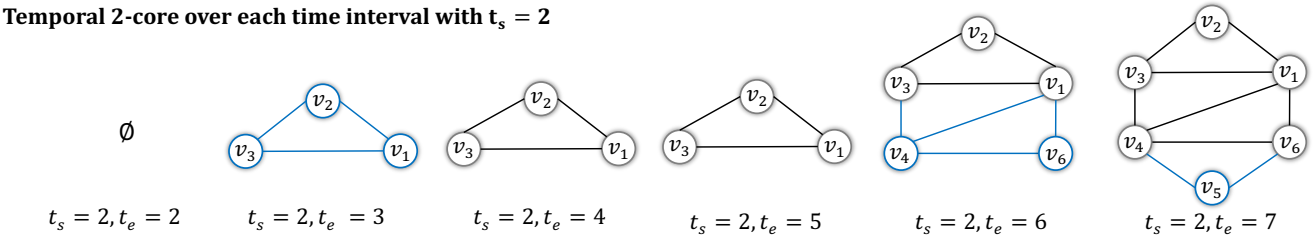



**Figure 17: Projected graphs and their corresponding temporal 2-cores for each time window ($t_s = 2$).**

Proof. To prove the correctness of algorithm 5, we only need to prove that for a fixed $k$, the $S_k[t_s, t_e + 1]$ can be correctly calculated by $S_k[t_s, t_e]$. Supposing that there exists a vertex $v \in S_k[t_s, t_e + 1]$ but not be included in it by our algorithm. That is to say, for an edge $e^* = (u, v)$, $\mathcal{L}_{t_s}(e^*)_k > t_e + 1$, which contradicts the definition of active time. Hence, the above theorem holds. □

Lemma 5.6 The total time cost of AIT is $O(t_{max} \cdot n + t_{max} \cdot \min\{t_{max}, n\} \cdot \log n)$, where all variables are defined as in Lemma 3.1.

Proof. First, computing MSF takes $O(t_{max} \cdot n)$ time. And then, for each start time $t_s$, AIT takes $O(\min\{t_{max}, n\} \cdot \log n)$ time to compute the temporal core with the largest duration for all time windows which using $t_s$ as the start time. Specifically, for the start time $t_s$, there are at most $\min\{t_{max}, n\}$ end times in our index, and for each window, AIT can retrieve $e^*$ from $\mathcal{E}(k)$ in $O(\log n)$ time. Hence, the above lemma holds. □