

Effective Structurally Similar Community Search in Large Temporal Graphs

YUE ZHANG, The Chinese University of Hong Kong, Shenzhen, China

YINGLI ZHOU, The Chinese University of Hong Kong, Shenzhen, China

FANGYUAN ZHANG, Harbin Institute of Technology, Shenzhen, China

XIAOLIN HAN, Northwestern Polytechnical University, China

CHENHAO MA*, The Chinese University of Hong Kong, Shenzhen, China

A temporal graph is an undirected graph where each edge is associated with a timestamp representing the interaction time. As a fundamental problem in graph analysis, community search (CS) in temporal graphs has received tremendous research attention. Existing CS works on temporal graphs typically focus on identifying cohesive subgraphs (e.g., k -cores) within a specific time window. However, they overlook the specific roles of vertices (e.g., core, hub) and fail to capture their dynamic evolution over time. In this paper, we introduce a novel community model, called *temporal structurally similar community (TSC)*, which models the communities in temporal graphs by explicitly considering the evolution of vertex roles across different time windows. Based on the TSC, we develop fast online and index-based algorithms that support efficient TSC queries over arbitrary time windows. Extensive experiments on real-world datasets demonstrate the effectiveness of the TSC model in revealing meaningful communities and the high efficiency of our proposed algorithms.

CCS Concepts: • **Theory of computation** → **Graph algorithms analysis**.

Additional Key Words and Phrases: temporal graph, community search, structural similarity, graph indexing

ACM Reference Format:

Yue Zhang, Yingli Zhou, Fangyuan Zhang, Xiaolin Han, and Chenhao Ma. 2026. Effective Structurally Similar Community Search in Large Temporal Graphs. *Proc. ACM Manag. Data* 4, 4 (SIGMOD), Article 271 (September 2026), 28 pages. <https://doi.org/10.1145/3837109>

1 Introduction

In many real-world applications, such as transaction, social, and email networks, the relationships between entities can be modeled as temporal graphs, where each edge is associated with a timestamp representing the time of interaction [15, 35, 44, 55, 74]. 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. Fig. 1(a) illustrates a temporal graph comprising 16 nodes and 27 edges, where edge labels denote interaction timestamps. For instance, the edge connecting v_1 and v_2 signifies an interaction (e.g., a transaction confirmation) occurring at $t = 1$.

*Corresponding author.

Authors' Contact Information: Yue Zhang, The Chinese University of Hong Kong, Shenzhen, Shenzhen, China, yuezhang5@link.cuhk.edu.cn; Yingli Zhou, The Chinese University of Hong Kong, Shenzhen, Shenzhen, China, yinglizhou@link.cuhk.edu.cn; Fangyuan Zhang, Harbin Institute of Technology, Shenzhen, Shenzhen, China, zhangfangyuan@hit.edu.cn; Xiaolin Han, Northwestern Polytechnical University, Xi'an, China, xiaolinh@nwpu.edu.cn; Chenhao Ma, The Chinese University of Hong Kong, Shenzhen, Shenzhen, China, machenhao@cuhk.edu.cn.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2836-6573/2026/9-ART271

<https://doi.org/10.1145/3837109>

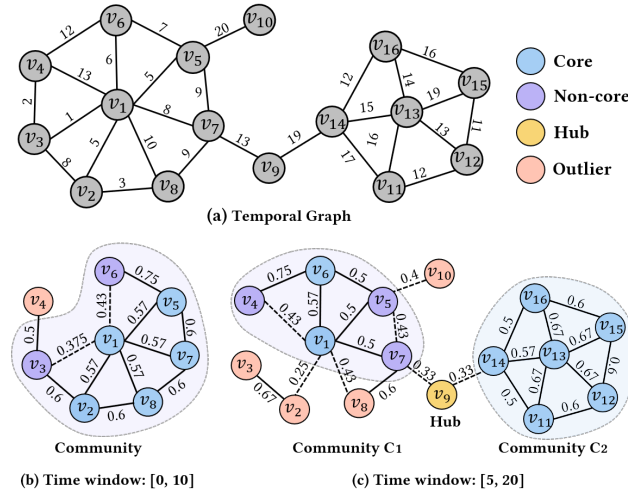


Fig. 1. Illustration of TSCs ($\epsilon = 0.5, \mu = 3$) over different time windows. (a) Edge labels denote interaction timestamps. (b)-(c) Edge labels denote structural similarity.

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 [30, 32, 34, 44, 45, 79, 82, 86]. 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. In the literature, earlier community search (CS) studies mainly focus on static graphs [25, 26, 36, 63, 92, 93], often neglecting temporal interactions. To fill this gap, a few recent temporal CS models have emerged [30, 32, 34, 44, 45, 79, 81, 82, 86], and most of them are based on the well-known k -core model [4], which can be further classified into two groups. The first group [30, 32, 44] 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 mitigate this rigidity, the second line relaxes the constraint by defining communities over a time window, where a community is modeled as a set of vertices that form a k -core in the static graph projected from the temporal graph within that window [34, 45, 79, 82, 86]. For example, the temporal k -core [82, 86] identifies k -cores in graph snapshots over a given time window for a specific k .

Although the aforementioned models are effective in revealing meaningful temporal communities, they overlook the structural similarity between vertices, a crucial factor for mining high-quality communities [80]. More importantly, they neglect the specific roles of members, such as cores, non-cores, hubs, and outliers. As demonstrated in static graph community mining, structural similarity has proven effective for modeling such roles. The classic SCAN method [80] explicitly models structural similarity using neighborhood sets. It requires each core vertex to have at least μ neighbors with a structural similarity exceeding ϵ . Notably, SCAN identifies four distinct vertex roles: *cores* and *non-cores*, *hubs*, and *outliers*, where the first two roles are used to distinguish the importance of vertices in a community, hubs are vertices connecting different communities, and outliers do not belong to any community.

In temporal graphs, these roles are not static; rather, they naturally evolve dynamically. For instance, a junior researcher may initially appear as a peripheral outlier but gradually evolve into a core contributor as their reputation grows. Similarly, a hub researcher connecting multiple groups

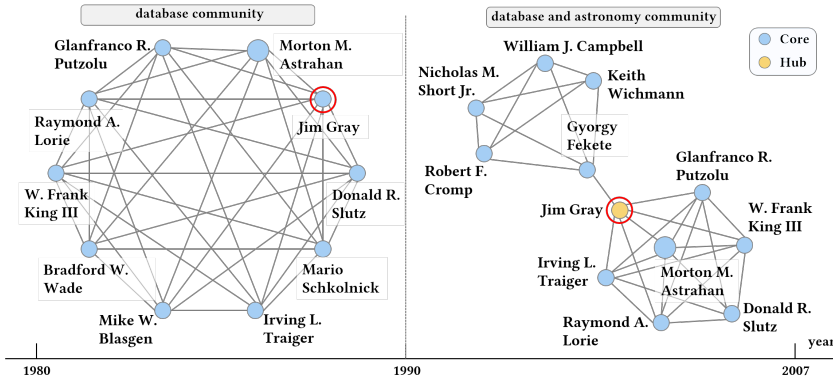


Fig. 2. The community of Jim Gray.

may eventually specialize in a specific field and transition into a core role. Tracking such role transitions provides deeper insights into community evolution. Motivated by this, we propose the *temporal structurally similar community* (TSC) search problem, which extends structural similarity to the temporal graph, enabling the discovery of structurally cohesive communities while explicitly capturing the dynamic evolution of vertex roles. Formally, given a time window $[t_s, t_e]$, a query vertex q , and two parameters ϵ and μ , the TSC search problem aims to identify a community containing q such that, within the graph projected over $[t_s, t_e]$, each core vertex has at least μ neighbors whose structural similarity values exceed the threshold ϵ . For example, Fig. 1(c) shows two TSCs containing vertices v_1 and v_{13} with $\epsilon = 0.5$ and $\mu = 3$. This means that every core vertex in these communities has at least three neighbors whose structural similarity exceeds 0.5.

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

- **Dynamic network analysis.** TSCs provide a role-aware temporal view for tracking 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 [95], and the timestamp of each edge corresponds to the publication time. Indeed, when a senior researcher transitions from an established domain to a new one (e.g., Jim Gray’s shift from databases to astronomy), the evolution involves more than a simple reconstruction of collaborations. Before the transition, the researcher typically occupies a *core member* within a community in the original field. Upon entering the new domain, they initially appear as a peripheral member, forming new collaborations with unfamiliar co-authors. As the researcher becomes established in the new field, they often evolve into a *bridging hub*, connecting the original community with the emerging one and facilitating cross-domain knowledge transfer. Our TSC model captures these dynamics effectively. In Fig. 2, we analyze Jim Gray’s co-authorship communities from 1980 to 2007 using TSC queries with parameters set to $\mu = 2$ and $\epsilon = 0.3$. We observe that during his transition from databases to astronomy (e.g., 1977–1987), his structural role shifts from a core member in the database community to a connective hub linking the database and astronomy communities.

- **Event detection.** TSCs enable event detection by capturing the temporal evolution of network structure. Specifically, by modeling changes in member roles and the internal similarity among community members, TSCs can reveal significant structural shifts that often correspond to major events. In our case study (Section 6.2), we demonstrate that TSCs can be effectively used to detect major events in transaction networks.

Challenges. Given a query time window $[t_s, t_e]$, to find the TSC containing a query vertex, a naive online algorithm needs to first obtain the projected graph over $[t_s, t_e]$, and then invoke the state-of-the-art online SCAN algorithm on the static graph (i.e., pSCAN [9]) to compute TSCs. In practice, users may need to examine TSCs over different time windows to analyze the dynamics of a network. In this scenario, the above online algorithm is not suitable, as it is designed for a single, fixed window and does not support efficient multi-window queries. Therefore, an index-based solution is required, and the existing index structures designed for SCAN on static graphs can be naturally extended to temporal graphs. In particular, prior works [18, 22, 89] commonly pre-compute bottom- k sketches for each vertex, which support efficient Jaccard similarity computation while requiring only linear space with respect to the number of vertices. However, a naive approach that constructs such indices for all possible time windows is clearly both time- and space-consuming, rendering it impractical for large temporal graphs.

Our technical contributions. We tackle the issues above by proposing novel index-based solutions. To support efficient and frequent TSC queries with various parameters, we propose time- and space-efficient index-based solutions that enable fast computation of Jaccard similarity between any pair of vertices over arbitrary time windows, thereby enabling efficient discovery of TSCs. To this end, we adopt the bottom- k sketch to estimate Jaccard similarities between vertices. In fact, we observe that the bottom- k sketch often remains unchanged across many overlapping time windows. This allows us to aggregate such windows into two-dimensional regions, within which all windows share an identical bottom- k sketch. We refer to each such region as a *sketch equivalence block* (SEB). For example, the four time windows that share the same bottom- k sketch: $[1, 4]$, $[2, 4]$, $[1, 3]$, and $[2, 3]$, can be jointly represented as a 2×2 rectangular region defined by start times $\{1, 2\}$ and end times $\{3, 4\}$. That is, we can construct an index by storing all SEBs. However, this raises two key questions: (1) *How to efficiently compute the SEB?* and (2) *How to organize these blocks to enable efficient TSC search?*

For the first question, we design a series of pruning criteria to avoid enumerating all possible time windows and computing a bottom- k sketch for each of them. Our design is based on the key observation that the top- k elements remain nearly identical across adjacent time windows [50]. Specifically, we consider two-level pruning: 1) For time windows with the same start time, consider a fixed start time t_s . As the end time t_e increases, once a valid bottom- k sketch is formed at an end time t_y , we can directly determine the end time t_d ($t_d > t_y$) at which the top- k elements first change. Consequently, the bottom- k sketches of all windows $[t_s, t_y] \subset [t_s, t_y + 1] \subset \dots \subset [t_s, t_d - 1]$ remain identical. Therefore, all end times in the interval $[t_y + 1, t_d - 1]$ can be safely skipped during sketch computation. 2) For time windows with different start times, we observe the following property. Given two start times t_x and t_y with $t_x < t_y$, if their bottom- k sketches are identical at some end time t_d , i.e., the bottom- k sketch of $[t_x, t_d]$ is the same as that of $[t_y, t_d]$, then for any end time $t_e \in [t_d, t_{\max}]$, the bottom- k sketches of $[t_x, t_e]$ and $[t_y, t_e]$ remain identical. Consequently, for the start time t_y , we can directly reuse the sketch computations from t_x , thereby avoiding redundant computation.

For the second question, if all SEBs are stored in a flat manner, then given a query time window $[t_s, t_e]$, retrieving the corresponding bottom- k sketch requires examining all SEBs whose start times are earlier than t_s . This leads to substantial unnecessary computation, since many irrelevant SEBs must be checked. To further improve query efficiency, we propose a novel index structure that partitions the start-time domain of SEBs into disjoint *base intervals*. Using this index, SEBs are organized according to their start times and can be accessed more selectively. As a result, for a query time window $[t_s, t_e]$, retrieving the corresponding bottom- k sketch only requires examining the SEBs within the base interval associated with the start time t_s . This significantly reduces the number of examined SEBs and greatly improves query efficiency.

In addition, we conducted extensive experiments on six large real-world temporal graphs. The results show that TSCs achieve higher community quality than other temporal community models. Moreover, our index-based TSC search algorithm is up to two orders of magnitude faster than the online-based method. We have released the source code and datasets used in this work at <https://github.com/zyxxx2/TSC>.

In summary, our main contributions are:

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

Outline. We formally present the TSC search problem in Section 2. Section 3 proposes the online-based TSC search algorithm. In Section 4 and 5, we present our index-based TSC search algorithms. We report the experimental results in Section 6. We review the related works in Section 7 and conclude in Section 8.

2 PRELIMINARIES

2.1 Structural Clustering in Static Graphs

Notations in graphs. We consider an unweighted and undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges in the graph. We use n to indicate the number of vertices and m to indicate the number of edges. An edge e connecting vertex u and v is denoted as an unordered pair of vertices, (u, v) or (v, u) , and u and v are called a neighbor of each other or simply neighboring vertices. For a vertex $u \in V$, the *neighborhood* $N[u]$ consists of (i) the set of vertices that are neighbors of vertex u and (ii) u itself. The *degree* d_u of vertex u is defined as the number of neighbors of u .

Terminologies in SCAN. To measure the structural similarity $\sigma(u, v)$ between two neighboring vertices u and v , we assess the overlap of their neighborhoods, $N[u]$ and $N[v]$. Consistent with prior work [9, 59, 69, 88], we use the Jaccard similarity. In other words, given two neighboring vertices u and v , their similarity $\sigma(u, v) = |N[u] \cap N[v]| / |N[u] \cup N[v]|$. If u and v are not neighbors, their similarity is 0.

In the Structural Clustering Algorithm for Networks (SCAN), the first input parameter is a threshold ϵ , which identifies the ϵ -neighbors, defined as follows.

Definition 1 (ϵ -neighbor [89]). Given a threshold $\epsilon \in (0, 1]$ of the similarity, v is called an ϵ -neighbor of u if $v \in N[u]$ and the similarity $\sigma(u, v)$ between u and v is at least ϵ . The ϵ -neighbor set $N_\epsilon[u]$ is defined as the set of ϵ -neighbors of vertex u , i.e., $N_\epsilon[u] = \{v \in N[u] \mid \sigma(u, v) \geq \epsilon\}$.

Given the definition of ϵ -neighbors, the second input parameter is an integer threshold $\mu \geq 2$, which is used to define *core vertices*.

Definition 2 (Core Vertex [89]). A vertex $u \in V$ is a core vertex if u has at least a number μ of ϵ -neighbors, i.e., $|N_\epsilon[u]| \geq \mu$.

Definition 3 (Structural Reachability [75]). Given two vertices u and v , v is structurally reachable from u if there is a sequence of vertices $v_1, v_2, \dots, v_l$ ($l \geq 2$) such that: (i) $v_1 = u, v_l = v$; (ii) for all $1 \leq i \leq l - 1$, v_i is a core vertex, and $v_{i+1} \in N_\epsilon[v_i]$.

Based on the above definitions, we provide the definition of a (ϵ, μ) -community, i.e., a structural cluster in [9, 75, 80, 89].

Definition 4 ((ϵ, μ) -community). Given a graph G , an integer μ , and a threshold vector ϵ , a (ϵ, μ) -community C is a subset of V with at least two vertices (i.e., $|C| \geq 2$) such that:

- **Connectivity:** For any two vertices $v_1, v_2 \in C$, there is a vertex $u \in C$ such that both v_1 and v_2 are structurally reachable from u .
- **Maximality:** If a core vertex $u \in C$, then all vertices that are structurally reachable from u also belong to C .

If a vertex of a (ϵ, μ) -community is not a core vertex, then it is called a non-core vertex. For a vertex v that is not in any (ϵ, μ) -communities, if it has neighbors belonging to multiple communities, then it is a *hub vertex*. Otherwise, it is an *outlier*.

2.2 Problem Definition in Temporal Graphs

Based on the concepts defined above, we formally define the (ϵ, μ) -community search problem in temporal graphs.

Notations in temporal graphs. We consider an unweighted and undirected temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges in the graph. Each edge $e \in \mathcal{E}$ is represented by a triplet (u, v, t) , where u and v are vertices from the set $\mathcal{V}$, and $t \in \mathbb{N}$ is the timestamp of the interaction between them.

Definition 5 (Projected Graph [79]). Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a time window $[t_s, t_e]$, the *projected graph* of $\mathcal{G}$ over $[t_s, t_e]$ is denoted as $G_{[t_s, t_e]} = (V, E_{[t_s, t_e]})$, where $E_{[t_s, t_e]} = \{(u, v) \mid \exists (u, v, t) \in \mathcal{E} \wedge t \in [t_s, t_e]\}$, and V consists of all vertices incident to the edges in $E_{[t_s, t_e]}$.

As $G_{[t_s, t_e]}$ is a static graph, we follow the notational conventions used for G . The *windowed neighborhood* of a vertex u is defined as $N[u, [t_s, t_e]] = \{v : (u, v) \in E_{[t_s, t_e]}\} \cup \{u\}$. We also define the *instantaneous neighborhood* at time t : $N(u, t) = \{v : (u, v, t) \in \mathcal{E}\}$.

Next, we introduce the temporal structurally similar community search problem, which aims to find the (ϵ, μ) -community in temporal graphs. In the following, without ambiguity, we refer to the (ϵ, μ) -community as the temporal structurally similarity community (or TSC). Formally,

PROBLEM 1. Give a temporal graph $\mathcal{G}$, a time window $[t_s, t_e]$, two parameters $\epsilon \in (0, 1]$ and $\mu \geq 2$, and a query vertex q , the temporal structurally similar community search problem aims to find the (ϵ, μ) -community containing q in the corresponding projected graph $G_{[t_s, t_e]}$ of $\mathcal{G}$.

EXAMPLE 1. Consider the temporal graph $\mathcal{G}$ in Fig. 1(a). Given the parameters $\epsilon = 0.5$ and $\mu = 3$, for a query vertex $q = v_1$ and time window $[0, 10]$, the identified TSC is $\{v_1, v_2, v_3, v_5, v_6, v_7, v_8\}$.

3 ONLINE SEARCH ALGORITHM

To search for the TSC of a query vertex, a straightforward idea is to perform an online search over the projected graph $G_{[t_s, t_e]}$. We term this naive approach OTCS (Online search algorithm for TSC Search). Specifically, OTCS materializes the projected graph for the query time window and then applies a core-based expansion procedure. Although simple, it serves as a natural baseline before introducing our index-based acceleration.

Core-Based Expansion. Following the SCAN community model, a vertex is a core if it has at least μ neighbors whose similarity with it is no less than ϵ . Starting from a core query vertex q , we expand its community using BFS over the projected graph by invoking Algorithm 1. If q is not a core vertex, we expand from its ϵ -similar core neighbors via Algorithm 1; if none exist, $\{q\}$ is returned.

To speed up similarity verification, we adopt the similar-degree (*sd*) and effective-degree (*ed*) bounds from [9]. For a vertex u , $sd(u)$ counts neighbors already verified as ϵ -neighbors, while

Algorithm 1: OTCS

Input: A projected graph $G_{[t_s, t_e]}$, a core vertex q , parameters (e.g., ϵ and μ)
Output: A community C containing q

```

1  $C \leftarrow \{q\}, Q.enqueue(q);$ 
2 while  $Q \neq \emptyset$  do
3    $u \leftarrow Q.dequeue();$ 
4   if  $CheckCore(u)$  is true then
5     for each vertex  $v \in N[u]$  do
6       compute  $\sigma(u, v)$  under the similarity measure;
7       if  $v \notin C$  and  $\sigma(u, v) \geq \epsilon$  then
8          $C \leftarrow C \cup \{v\}, Q.enqueue(v);$ 
9 return  $C;$ 
```

Algorithm 2: CheckCore(u)

```

1  $sd(u) \leftarrow 0, ed(u) \leftarrow d_u;$ 
2 if  $ed(u) \geq \mu$  then
3   for each vertex  $v \in N[u]$  do
4     compute  $\sigma(u, v)$  under the similarity measure;
5     if  $\sigma(u, v) \geq \epsilon$  then  $sd(u) \leftarrow sd(u) + 1;$ 
6     else  $ed(u) \leftarrow ed(u) - 1;$ 
7     if  $sd(u) \geq \mu$  or  $ed(u) < \mu$  then break;
8 if  $sd(v) \geq \mu$  then return true;
9 return false;
```

$ed(u)$ tracks the maximum remaining number of possible ϵ -neighbors. These bounds allow early termination during core checking.

Algorithm Overview. Algorithm 1 performs community expansion from a core vertex q . Vertices are explored in BFS order; a vertex u is used to expand the community if it is a core vertex (line 4), and a neighbor v of u is added to the community if $\sigma(u, v) \geq \epsilon$ (lines 7-8). Algorithm 2 verifies whether a vertex is a core. The algorithm repeatedly evaluates similarities with neighbors (lines 3-6) until either $sd(v) \geq \mu$ (core confirmed) or $ed(v) < \mu$ (core impossible) (line 7).

4 INDEX-BASED SEARCH ALGORITHM

The online algorithm requires reconstructing the projected graph and repeatedly computing structural similarities for each query window. When users issue queries over diverse or arbitrary time windows, these repeated computations lead to significant overhead. To avoid such redundancy, we seek an index that can support efficient similarity evaluation for any time window.

A naive strategy is to precompute similarity values for every adjacent vertex pair under all possible windows. However, this requires $O(\bar{m} \cdot t_{\max}^2)$ space, where $\bar{m}$ denotes the average edge count and $t_{\max}$ is the number of distinct timestamps, which is prohibitive for real temporal graphs with large $t_{\max}$. Therefore, a more compact indexing mechanism is needed. We next introduce our index-based solution. We first outline the index structure, then describe how queries are processed using the index, and finally present the index construction algorithm.

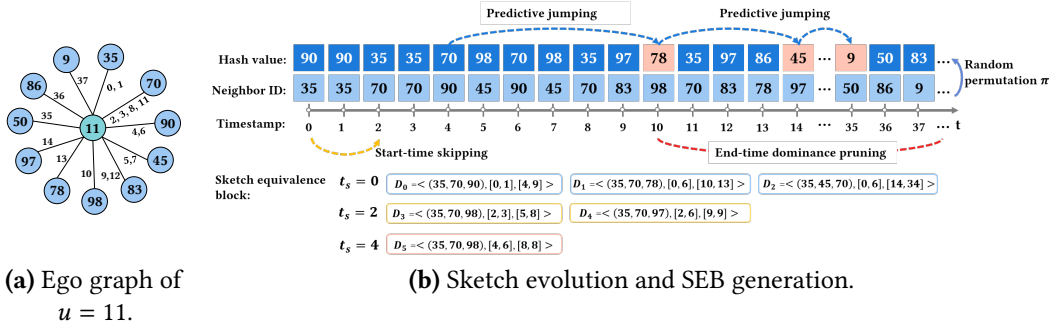


Fig. 3. Running example of sketch equivalence block generation.

4.1 Overview of Index

To address this challenge, we propose a novel indexing framework based on bottom- k sketches. Our core strategy avoids explicit similarity score materialization by maintaining only a compact, time-compressed sketch for each vertex's neighborhood. This approach is highly efficient for two key reasons: First, the index size is decoupled from the $t_{\max}^2$ due to observed *temporal coherence* in vertex sketches. Second, the sketch structure inherently provides an unbiased estimator for Jaccard similarity of neighborhoods, enabling high-fidelity queries without full computation.

Temporal bottom- k sketch. To formally define our compact index and the underlying similarity estimation, we first introduce the core mechanism: the bottom- k sketch.

Definition 6 (Bottom- k Sketch [89]). Let U be the universe of the elements that may appear in a set A . Let $h : U \rightarrow [1 \cdots |U|]$ be a hash function that is a random permutation of U . The bottom- k sketch of A is the set of k smallest hash values among elements in A . If $k > |A|$, it consists of the set of hash values of all elements in A .

We adapt the bottom- k sketch to temporal graphs by using a global, time-invariant random permutation $\pi : \mathcal{V} \rightarrow \{1, \dots, |\mathcal{V}|\}$ as the hash function. For a vertex u and a time window $[t_s, t_e]$, the set to be sketched is its windowed neighborhood $N[u, [t_s, t_e]]$. Applying the bottom- k mechanism to this set yields the temporal bottom- k sketch $S_k(u, [t_s, t_e])$, which consists of the k smallest hash values derived from the neighborhood. For brevity, when the time window is clear, we write $S_k(u)$ and $N[u]$ in place of $S_k(u, [t_s, t_e])$ and $N[u, [t_s, t_e]]$.

An important benefit of the bottom- k sketch is its inherent space efficiency. For a vertex u with degree d_u in the window $[t_s, t_e]$, the sketch stores only $\min(k, d_u)$ vertex identifiers. This compact representation adapts to local density and provides an unbiased basis for estimating neighborhood similarities, as formalized in the following lemma.

LEMMA 1 (BOTTOM- k ESTIMATOR OF JACCARD SIMILARITY [89]). *Given two sets $X \subseteq U$ and $Y \subseteq U$, let $S_k(X)$ and $S_k(Y)$ denote their bottom- k sketches, respectively. If $|X| \geq k$ or $|Y| \geq k$, then the value:*

$$\hat{J}(X, Y) = \frac{|S_k(X) \cap S_k(Y) \cap S_k(X \cup Y)|}{k} \quad (1)$$

is an unbiased estimator of the Jaccard similarity $J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$.

We adopt a hybrid estimation strategy. When at least one of the two vertices has degree $\geq k$ (i.e., $d_u \geq k$ or $d_v \geq k$), we use the sketch-based estimator. If both degrees are below k , we compute the exact Jaccard similarity from the raw neighborhoods. This selective scheme guarantees correctness while keeping the similarity computation cost within $O(k)$.

Sketch equivalence block. While the bottom- k sketch compactly summarizes a neighborhood within a single time window, naively indexing sketches for all possible windows would still produce substantial redundancy. Our key observation is *sketch temporal coherence*: a bottom- k sketch for a vertex often remains valid across many adjacent or overlapping time windows.

Fig. 3(a) shows the running ego graph of $u = 11$, and Fig. 3(b) shows the corresponding sketch evolution with $k = 3$. Each timestamp column in Fig. 3(b) represents one incoming edge to u ; the “Neighbor ID” and “Hash value” rows give the arrived neighbor and its randomized ID under π , respectively.

This example shows that the same bottom- k sketch can persist across many query windows. For instance, the sketch $\{35, 70, 78\}$ is shared by all windows $[t_s, t_e]$ with $t_s \in [0, 6]$ and $t_e \in [10, 13]$. Motivated by this observation, we aggregate all windows that generate the same sketch into a single *sketch equivalence block*.

Definition 7 (Sketch Equivalence Block (SEB)). Let $\mathcal{H} = S_k(u, [t_s, t_e])$ be the temporal bottom- k sketch of a vertex u for window $[t_s, t_e]$. The corresponding *sketch equivalence block* is defined as $D(\mathcal{H}) = \langle \mathcal{H}, \mathcal{T}_s, \mathcal{T}_e \rangle$, where $\mathcal{T}_s = [T_{s,\min}, T_{s,\max}]$ and $\mathcal{T}_e = [T_{e,\min}, T_{e,\max}]$ denote the maximal valid start-time and end-time intervals. The sketch equivalence block must satisfy:

- (1) **Validity:** For any $t_1 \in \mathcal{T}_s$ and $t_2 \in \mathcal{T}_e$ with $t_1 \leq t_2$, the induced sketch remains unchanged: $S_k(u, [t_1, t_2]) = \mathcal{H}$.
- (2) **Maximality:** Intervals $\mathcal{T}_s$ and $\mathcal{T}_e$ cannot be extended without violating validity.

EXAMPLE 2. Consider the query window $[6, 10]$ for vertex u in Fig. 3, which yields the sketch $\mathcal{H} = \{35, 70, 78\}$. The corresponding SEB is $D_1 = \langle \{35, 70, 78\}, [0, 6], [10, 13] \rangle$.

Index organization. The set of all SEBs associated with a vertex u forms the SEB-index. Query processing leverages the SEB-index to facilitate efficient similarity estimation. For a given query vertex and time window $[t_s, t_e]$, the algorithm performs a lookup on the SEB-index to identify the unique SEB covering this window. The retrieved temporal bottom- k sketch is subsequently used to estimate the Jaccard similarity between the query vertex and its neighbors. Section 5 will subsequently detail the SEB-index construction framework and a structural optimization strategy to enable efficient retrieval.

Theoretical analysis. We now formalize the correctness properties underlying SEBs.

LEMMA 2. For any unique set of bottom- k sketch elements $\mathcal{H}$ associated with a vertex u , there exists a unique sketch equivalence block $D(\mathcal{H})$.

Proof sketch. For a fixed bottom- k sketch $\mathcal{H}$, the validity condition determines the set of admissible time windows, while maximality enforces a unique pair of intervals $(\mathcal{T}_s, \mathcal{T}_e)$. Hence, $D(\mathcal{H})$ is uniquely defined. $\square$

LEMMA 3. For any vertex u and any time window $[t_s, t_e]$, the window $[t_s, t_e]$ is contained in exactly one sketch equivalence block.

PROOF. (Existence) Let $\mathcal{H} = S_k(u, [t_s, t_e])$. By Lemma 2, $\mathcal{H}$ maps to a unique SEB $D(\mathcal{H})$. Since $[t_s, t_e]$ generates $\mathcal{H}$, it must lie within $D(\mathcal{H})$. **(Uniqueness)** Assume $[t_s, t_e]$ belongs to two SEBs $D_1(\mathcal{H}_1)$ and $D_2(\mathcal{H}_2)$. By validity, both windows must generate the same sketch, i.e., $\mathcal{H}_1 = \mathcal{H}_2$. By Lemma 2, the corresponding SEB is unique, implying $D_1 = D_2$, which is a contradiction. $\square$

4.2 Query Processing

Given the SEB-index introduced in the previous subsection, we now describe how it is used to process TSC queries efficiently.

Algorithm 3: CheckSim($u, v, [t_s, t_e]$)**Input:** Vertices u, v , time window $[t_s, t_e]$, ϵ, k **Output:** *true* if similar, *false* otherwise

```

1 Let  $d_u, d_v$  be degrees in  $G_{[t_s, t_e]}$  (assume  $d_u \geq d_v$ );
2 if  $d_v < d_u \cdot \epsilon$  then return false;
3 if  $d_u < k$  then return ExactJaccard( $u, v, G_{[t_s, t_e]}$ )  $\geq \epsilon$ ;
4 else
5   if  $d_v < k \cdot \epsilon$  then return false;
6    $\mathcal{H}_u \leftarrow \text{GetSketch}(u, [t_s, t_e])$ ;
7    $\mathcal{H}_v \leftarrow \text{GetSketch}(v, [t_s, t_e])$ ;
8   if  $\min(\mathcal{H}_u) > \max(\mathcal{H}_v)$  or  $\max(\mathcal{H}_u) < \min(\mathcal{H}_v)$  then
9     return false;
10  return CheckEstSim( $\mathcal{H}_u, \mathcal{H}_v, k$ );

```

We present our Index-based TSC Search algorithm (ITCS). Given a query vertex q and a time window $[t_s, t_e]$, ITCS performs a local traversal rooted at q , expanding core vertices and aggregating their valid neighbors. Unlike the baseline OTCS, structural similarity verification is performed via index lookups instead of on-the-fly neighborhood traversal, thereby avoiding redundant graph reconstruction and repeated similarity computation.

While ITCS follows the same expansion framework as OTCS (Algorithm 1), it replaces the costly similarity checks (line 6 of Algorithm 1 and line 4 of Algorithm 2) with calls to CheckSim, which leverages the SEB-index for fast pruning and estimation.

Similarity verification. Algorithm 3 details the CheckSim procedure, which verifies whether two vertices are structurally similar under a query window. The algorithm applies a hierarchical pruning strategy to eliminate dissimilar pairs before performing any sketch-based estimation.

First, a degree-based pruning is applied (lines 1-2). Assuming $d_u \geq d_v$, any pair with $d_v < d_u \cdot \epsilon$ is immediately discarded, as it cannot satisfy the Jaccard similarity threshold. If the larger neighborhood size satisfies $d_u < k$, the exact Jaccard similarity is computed directly (line 3).

For larger neighborhoods, an additional cardinality filter discards pairs with $d_v < k \cdot \epsilon$ (line 5), since insufficient overlap can be achieved even under the best-case sketch alignment. For the remaining candidates, sketches $\mathcal{H}_u$ and $\mathcal{H}_v$ are retrieved from the SEB-index (lines 6-7). A fast range check is then applied: if the hash ranges of the two sketches are disjoint, the pair is pruned (lines 8-9). Only the surviving candidates invoke the sketch-based estimator via CheckEstSim (line 10).

Estimated similarity checking. Algorithm 4 depicts CheckEstSim, which determines whether the estimated Jaccard similarity $\hat{J}$ meets the threshold ϵ . The algorithm performs a linear two-pointer scan over the sorted sketches $\mathcal{H}_u$ and $\mathcal{H}_v$, identifying valid overlapping elements (lines 2-10).

A matched element between $\mathcal{H}_u$ and $\mathcal{H}_v$ contributes to $\hat{J}$ only if it also lies within the bottom- k sketch of the union of the neighborhoods, according to Lemma 1. This condition is verified dynamically using the union-rank $x + y - \text{overlap} - 1$ of the current matched element. The overlap counter is incremented only when this rank does not exceed k (lines 4-5). If the rank exceeds k , further matching cannot be included in the sketch of the union, and the algorithm terminates early (line 7). Since satisfying $\hat{J} \geq \epsilon$ requires at least $\lceil k\epsilon \rceil$ valid overlaps, the algorithm returns *true* immediately once this threshold is reached (line 6).

Algorithm 4: CheckEstSim($\mathcal{H}_u, \mathcal{H}_v, k$)**Input:** Sorted sketches $\mathcal{H}_u, \mathcal{H}_v$, sketch size k **Output:** *true* if similar, *false* otherwise

```

1  $x \leftarrow 1, y \leftarrow 1, \text{overlap} \leftarrow 0;$ 
2 while  $x \leq |\mathcal{H}_u|$  and  $y \leq |\mathcal{H}_v|$  do
3   if  $\mathcal{H}_u[x-1] == \mathcal{H}_v[y-1]$  then
4     if  $x + y - \text{overlap} - 1 \leq k$  then
5        $\text{overlap} = \text{overlap} + 1;$ 
6       if  $\text{overlap} \geq \lceil k\epsilon \rceil$  then return true ;
7     else return false;
8      $x \leftarrow x + 1, y \leftarrow y + 1;$ 
9   else if  $\mathcal{H}_u[x] < \mathcal{H}_v[y]$  then  $x \leftarrow x + 1;$ 
10  else  $y \leftarrow y + 1;$ 
11 return false

```

EXAMPLE 3. This toy example illustrates the early stopping rule in CheckEstSim. Consider a candidate pair (u, v) with projected degrees $d_u = 100$ and $d_v = 80$ under $k = 10$ and $\epsilon = 0.6$. After passing degree-based pruning, sketches $\mathcal{H}_u = \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19\}$ and $\mathcal{H}_v = \{1, 3, 5, 7, 9, 11, 20, 22, 24, 26\}$ are retrieved. As their ranges overlap, CheckEstSim is invoked. The first six matched elements satisfy the union-rank condition, reaching the required overlap $\lceil k\epsilon \rceil = 6$, and the algorithm terminates early with a positive result.

Complexity Analysis. Each call to CheckSim involves only constant-time pruning, sketch retrieval, and a linear scan over sketches of size at most k , yielding a worst-case time of $O(k)$. Since similarity verification may be performed for each examined edge during community expansion, the worst-case query-time complexity of ITCS is $O(k \cdot \bar{m})$, where $\bar{m}$ represents the number of edges in the projected graph.

In contrast, the baseline online algorithm OTCS may require scanning the full neighborhoods to compute exact similarities, resulting in a worst-case cost of $O(d_{\max} \cdot \bar{m})$, where $d_{\max}$ is the maximum vertex degree in the projected graph. As typically $k \ll d_{\max}$, ITCS achieves a substantial reduction in query-time complexity, particularly on large and dense temporal graphs.

5 INDEX CONSTRUCTION

In this section, we present the framework for constructing the index structures used by ITCS. Algorithm 5 outlines a node-centric, two-step pipeline. For each vertex $u \in \mathcal{V}$, we first invoke ConstructSEB to build the raw SEB-index, denoted as $\mathcal{D}_u$, which contains all SEBs of u . We then apply PartitionSEB to transform $\mathcal{D}_u$ into the Partitioned Sketch equivalence block index (PISA), denoted as $\mathcal{I}_u$, which is optimized for query processing as described in Section 5.2. The final output is a global index $\mathcal{I}$ that maps each vertex u to its corresponding $\mathcal{I}_u$.

5.1 Construct SEB

A naive approach to constructing the SEBs is to compute the bottom- k sketch for every possible time window $[t_s, t_e]$. This incurs a prohibitive cost of $O(mt_{\max}^2)$, where $m = |\mathcal{E}|$ is the number of temporal edges. The quadratic dependence on $t_{\max}$ renders such an approach infeasible for real temporal graphs.

Algorithm 5: Index construction algorithm.**Input:** A temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, sketch size k **Output:** $\mathcal{I}$: a map from each vertex $u \in \mathcal{V}$ to its PISA

```

1  $\mathcal{I} \leftarrow$  new empty Map;
2 for each vertex  $u \in \mathcal{V}$  do
    // Step 1: Construct raw SEB-index
3    $(\mathcal{D}_u, \mathcal{B}_u) \leftarrow \text{ConstructSEB}(u, \mathcal{G}, k)$ ;
    // Step 2: Transform to PISA
4    $\mathcal{I}_u \leftarrow \text{PartitionSEB}(\mathcal{D}_u, \mathcal{B}_u)$ ;
5 return  $\mathcal{I}$ ;

```

We therefore propose an optimized algorithm, ConstructSEB, which exploits temporal regularities in neighborhood evolution. The algorithm is driven by two key observations: *fixed-start-time stability* and *cross-start-time coherence*, both illustrated in Fig. 3.

Observation 1: Fixed-start-time stability and predictive jumping. We first consider a fixed start time t_s (e.g., $t_s = 0$). As the end time t_e advances, the accumulated neighborhood grows monotonically. At a certain timestamp t_1 , the neighborhood becomes large enough to form the first valid bottom- k sketch $\mathcal{H}_0$.

In Fig. 3, when $t_s = 0$ and $k = 3$, the first sketch $\mathcal{H}_0 = \{35, 70, 90\}$ forms at $t_1 = 4$. This sketch defines a SEB D_0 with boundaries determined as follows:

(i) *Start-time boundaries.* The minimal start time is fixed at $T_{s,\min} = t_s = 0$. The maximal start time $T_{s,\max} = 1$ is determined by the minimum latest appearance times of the sketch elements $\{35, 70, 90\}$, ensuring that all elements remain present in the window.

(ii) *End-time boundaries.* The minimal end time is $T_{e,\min} = t_1 = 4$. Since the bottom- k sketch acts as a low-pass filter, it remains unchanged until a *disruptive event* occurs. A disruptive event at time t_d is triggered by the arrival of a neighbor v with $\pi(v) < \max(\mathcal{H}_0)$. This event definitely bounds the window at $T_{e,\max} = t_d - 1$.

In the example, a disruptive hash value $78 < 90$ appears at $t_d = 10$, yielding $T_{e,\max} = 9$. This enables a *predictive jumping* strategy: instead of verifying sketches at every intermediate timestamp, the algorithm skips the entire stable interval $[t_1 + 1, t_d - 1]$ and jumps directly to t_d . Moreover, the same disruptive event immediately triggers the formation of the next sketch $\mathcal{H}_1 = \{35, 70, 78\}$. As illustrated by the predictive jumping in Fig. 3, this transforms a dense linear scan into a sparse sequence of jumps (e.g., $4 \rightarrow 10 \rightarrow 14 \rightarrow 35$).

Observation 2: Cross-start-time coherence and dominance pruning. Beyond fixed-start-time optimization, SEBs exhibit strong coherence across different start times, enabling two complementary pruning mechanisms.

(i) *Start-time skipping.* When processing $t_s = 0$, all SEBs generated remain valid for any $t_s \in [T_{s,\min}^0, T_{s,\max}^0]$. Here, $[T_{s,\min}^0, T_{s,\max}^0]$ denotes the start time interval of the first SEB derived with t_s fixed at 0 (e.g., $[0, 1]$). Consequently, the entire construction process for these start times is identical to that of $t_s = 0$, and the algorithm can safely skip all intermediate start times and jump directly to $t_s = T_{s,\max}^0 + 1$ (as illustrated by start-time skipping in Fig. 3).

(ii) *End-time dominance pruning.* SEBs constructed for an earlier start time may also dominate those that would be generated for later start times. Specifically, a SEB D_1 constructed at $t_s = 0$ may remain valid for all $t_s \in [0, 6]$ and for a contiguous end-time interval. When processing a later start time (e.g., $t_s = 2$), once the search reaches the earliest end time covered by D_1 (i.e., 10), all subsequent windows are guaranteed to be dominated. The algorithm can therefore terminate

Algorithm 6: ConstructSEB**Input:** Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, vertex u , sketch size k **Output:** $\mathcal{D}_u$: list of sketch equivalence blocks; $\mathcal{B}_u$: set of boundary timestamps

```

1  $\mathcal{D}_u \leftarrow []$ ;  $\mathcal{B}_u \leftarrow \{0\}$ ;
2 initialize  $\mathcal{M}_u[t] \leftarrow t_{\max}$  for all  $t$ ;
3  $t_s \leftarrow 0$ ;
4 while  $t_s \leq t_{\max}$  do
5    $t_e \leftarrow$  first timestamp  $\geq t_s$  where a bottom- $k$  sketch is formed;
6   while  $t_e < \mathcal{M}_u[t_s]$  and  $t_e \leq t_{\max}$  do
7      $\mathcal{H} \leftarrow$  bottom- $k$  sketch over window  $[t_s, t_e]$ ;
8      $T_{s,\max}, T_{e,\max} \leftarrow \text{ComputeBounds}(\mathcal{H}, t_s, t_e)$ ;
9     append  $D = \langle \mathcal{H}, [t_s, T_{s,\max}], [t_e, T_{e,\max}] \rangle$  to  $\mathcal{D}_u$ ;
10    insert  $T_{s,\max}$  to  $\mathcal{B}_u$ ;
11    if  $D$  is the not first window for this  $t_s$  then
12       $\mathcal{M}_u[t] \leftarrow \min(\mathcal{M}_u[t], t_e), \forall t \in [t_s, T_{s,\max}]$ 
13     $t_e \leftarrow T_{e,\max} + 1$ ;
14    let  $[T_{s,\min}^0, T_{s,\max}^0]$  be the start interval of the first window;
15     $t_s \leftarrow T_{s,\max}^0 + 1$ ;
16 return  $(\mathcal{D}_u, \mathcal{B}_u)$ ;

```

immediately for this start time, eliminating redundant construction of the entire suffix (marked as end-time dominance pruning in Fig. 3).

Construct SEB. Algorithm 6 describes the ConstructSEB procedure, which constructs SEBs for a vertex u by combining predictive jumping, start-time skipping, and end-time dominance pruning.

The algorithm maintains two outputs: $\mathcal{D}_u$, the list of all sketch equivalence blocks of u , and $\mathcal{B}_u$, which records the boundary timestamps defining the start-time interval of the SEBs generated for each processed start time and is later used for index partitioning. To enable dominance-based pruning, a horizon memo $\mathcal{M}_u$ is initialized (line 2), where $\mathcal{M}_u[t]$ stores the minimum end time beyond which any window starting at t is already dominated.

The construction proceeds via a nested loop structure in which both loops advance by jumps rather than unit increments. For a given start time t_s , the algorithm first identifies the earliest end time $t_e \geq t_s$ at which a valid bottom- k sketch can be formed (line 5). The inner loop then iteratively generates SEBs by advancing t_e , subject to the horizon constraint $t_e \leq \mathcal{M}_u[t_s]$ (lines 6-13).

For each candidate window $[t_s, t_e]$, the bottom- k sketch $\mathcal{H}$ is computed (line 7) and passed to ComputeBounds, which jointly determines the maximal start-time boundary $T_{s,\max}$ and the maximal end-time boundary $T_{e,\max}$ (line 8). These bounds define a SEB $D = \langle \mathcal{H}, [t_s, T_{s,\max}], [t_e, T_{e,\max}] \rangle$, which is added to $\mathcal{D}_u$ (line 9).

The boundary timestamp $T_{s,\max}$ is added to $\mathcal{B}_u$ to support the index partitioning process (line 10). If D is not the first window generated for the current t_s , the horizon memo is updated by setting $\mathcal{M}_u[t] \leftarrow \min(\mathcal{M}_u[t], t_e)$ for all $t \in [t_s, T_{s,\max}]$ (lines 11-12), enforcing dominance-based early termination for future start times.

After processing a window, the inner loop jumps directly to $t_e = T_{e,\max} + 1$ (line 13), skipping the entire stable interval. Once all windows for the current t_s are generated, the outer loop advances to $t_s = T_{s,\max}^0 + 1$, where $[T_{s,\min}^0, T_{s,\max}^0]$ is the start-time interval of the first window (lines 14-15). Through these coordinated jumps, the algorithm avoids exhaustive enumeration and visits only sketch-altering temporal events.

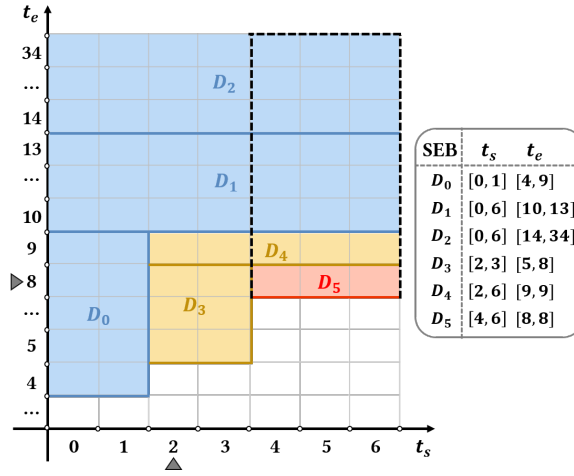


Fig. 4. Visualization of SEB validity regions in the t_s - t_e plane for the running example in Fig. 3.

Correctness. We prove the correctness of ConstructSEB through monotonic progression and index completeness.

LEMMA 4 (MONOTONICITY UNDER FIXED START TIME). *For any vertex u and fixed start time t_s , let $D_i = \langle \mathcal{H}_i, [t_s, T_{s,\max}^i], [T_{e,\min}^i, T_{e,\max}^i] \rangle$ be the sequence of sketch equivalence blocks generated by ConstructSEB. Then $T_{e,\min}^{i+1} > T_{e,\min}^i$ and $T_{s,\max}^{i+1} \geq T_{s,\max}^i$.*

Proof sketch. Each new window begins at $t_e = T_{e,\max}^i + 1$, implying $T_{e,\min}^{i+1} > T_{e,\min}^i$. The bound $T_{s,\max}^i$ is defined by the minimum latest-appearance time among elements in the bottom- k sketch. As t_e advances, retained sketch elements only gain later timestamps, and when a disruptive event updates the sketch, the newly inserted element has a timestamp no earlier than that of the displaced element. Thus the minimum latest-appearance time cannot decrease, yielding $T_{s,\max}^{i+1} \geq T_{s,\max}^i$. $\square$

LEMMA 5 (COMPLETENESS). *For any vertex u , the sketch equivalence blocks generated by ConstructSEB cover every time window $[t_s, t_e]$ with $|N[u, [t_s, t_e]]| \geq k$.*

Proof sketch. For a fixed t_s , the inner traversal advances t_e strictly forward and skips only intervals where the bottom- k sketch is invariant, ensuring that the unique SEB corresponding to $S_k(u, [t_s, t_e])$ is generated. Across start times, start-time skipping and end-time dominance pruning eliminate only windows already covered by previously generated SEBs. Hence no valid window is omitted, and every $[t_s, t_e]$ is contained in exactly one generated SEB. $\square$

5.2 Partitioned Index Construction

Although Algorithm 6 successfully constructs SEB, its flat organization is not suitable for efficient query processing. The difficulty stems from the inherently two-dimensional nature of SEBs: a query window $[t_s, t_e]$ must simultaneously satisfy both the start-time constraint $t_s \in [T_{s,\min}, T_{s,\max}]$ and the end-time constraint $t_e \in [T_{e,\min}, T_{e,\max}]$.

Fig. 4 visualizes the time-window ranges covered by the SEBs from Fig. 3. Each SEB covers the region $[T_{s,\min}, T_{s,\max}] \times [T_{e,\min}, T_{e,\max}]$ in the t_s - t_e plane, where each grid cell denotes a time window. For example, the window $[1, 4]$ is covered by D_0 , so it retrieves the sketch stored in D_0 . As shown in Fig. 4, a single query start time (e.g., $t_s = 2$) may fall within the valid start ranges of multiple sketch equivalence blocks (e.g., D_3 , D_4 , D_1 , and D_2), each associated with different and partially

overlapping start-time intervals. This overlap prevents the SEBs from being indexed effectively by either min or max start-time boundaries alone, leading to difficulty during lookup. Similar issues also exist for end-time intervals, e.g., querying with $t_e = 8$.

To resolve this issue, we reorganize the SEB-index into the PISA, which guarantees a unique and efficient mapping for every query window.

Base intervals. To enable efficient lookup, we first partition the start-time domain of vertex u using a set of boundary timestamps $\mathcal{B}_u$, which are recorded during ConstructSEB (Line 10 of Algorithm 6). Let $\mathcal{B}_u = \{b_0 < b_1 < \dots < b_m\}$ be the sorted boundary timestamps. We define the induced *base intervals* as the atomic segments between adjacent boundaries, i.e., $[b_i + 1, b_{i+1}]$, $0 \leq i < m$. By construction, these base intervals are pairwise disjoint and together cover the valid start-time range of u , so any query start time t_s maps to a unique base interval via binary search on $\mathcal{B}_u$.

Association induced by sketch equivalence blocks. Given a SEB $D = \langle \mathcal{H}, [T_{s,\min}, T_{s,\max}], [T_{e,\min}, T_{e,\max}] \rangle$, we associate D with a base interval B if and only if $B \subseteq [T_{s,\min}, T_{s,\max}]$. For each base interval B , we collect all associated pairs $([T_{e,\min}, T_{e,\max}], \mathcal{H})$ from the SEB-index. This partitioning reveals the following structural properties, which directly motivate our PISA design.

LEMMA 6 (DECOMPOSITION AND ORDERED CONTIGUITY). *Fix a vertex u and its SEB-index. Let $\mathcal{B}_u$ be the set of boundary timestamps and let B be any induced base interval. Then:*

- (1) **Start-interval decomposition.** *For any sketch equivalence block with start range $[T_{s,\min}, T_{s,\max}]$, this range can be expressed exactly as a union of one or more consecutive base intervals.*
- (2) **End-interval contiguity.** *For a fixed base interval B , the end-time intervals of all sketch equivalence blocks associated with B can be arranged to form a single contiguous sequence (i.e., they cover a continuous end-time range without gaps).*
- (3) **Reverse-start-time order.** *Among sketch equivalence blocks associated with the same base interval B , sorting them by decreasing $T_{s,\min}$ yields their end-time intervals in increasing order of $T_{e,\min}$.*

Proof sketch. (1) The boundaries in $\mathcal{B}_u$ are exactly the jump targets produced by ConstructSEB; hence any valid start range returned by ConstructSEB is aligned with boundary timestamps and thus decomposes into whole base intervals.

(2–3) Consider SEBs that all span the same base interval B . By construction, SEBs generated from later effective start times correspond to earlier feasible end-time segments (due to end-time dominance pruning), while for a fixed start time, the generated SEBs progress monotonically in end time (Lemma 4). Together, these imply that, restricted to B , the associated end-time intervals appear in reverse-start-time order and abut without gaps, forming a contiguous sequence. $\square$

For instance, as indicated by the black dashed outline in Fig. 4, processing the SEBs in the order D_5, D_3, D_4, D_1, D_2 for the base interval $[4, 6]$ results in a contiguous sequence covering $[8, 34]$.

Reverse start-time partitioning strategy. Lemma 6 (3) implies that if we process SEBs in descending order of their $T_{s,\min}$, then for each base interval B the corresponding end-time intervals are encountered in increasing order. We therefore build PISA by appending entries to each base interval list in one pass, yielding $O(1)$ insertion per entry and eliminating post-hoc sorting.

Algorithm 7 takes the complete list of sketch equivalence blocks $\mathcal{D}_u$ and base intervals $\mathcal{B}_u$ as input. It first initializes the index structure $\mathcal{I}_u$ by assigning an empty list to each base interval (line 1). To enable batch processing, the algorithm groups SEBs in $\mathcal{D}_u$ by their identical start-time intervals $[T_{s,\min}, T_{s,\max}]$ (line 2). The key design choice lies in the traversal order (line 3): groups are processed in descending order of $T_{s,\min}$ and, for ties, in ascending order of $T_{s,\max}$. For each group P , the algorithm extracts its associated end-time interval and sketch pairs $R = \{([T_{e,\min}, T_{e,\max}], \mathcal{H})\}$ (line 5). It then identifies all base intervals fully contained in the group's start-time range (line 6)

Algorithm 7: ConstructPISA

Input: $\mathcal{D}_u$: list of sketch equivalence blocks; $\mathcal{B}_u$: set of boundary timestamps
Output: $\mathcal{I}_u$: PISA of vertex u

```

1 foreach  $B \in \mathcal{B}_u$  do  $\mathcal{I}_u[B] \leftarrow \emptyset$ ;
2  $\mathcal{P} \leftarrow$  group  $\mathcal{D}_u$  by identical start interval  $[T_{s,\min}, T_{s,\max}]$ ;
3 foreach group  $P \in \mathcal{P}$  in order of decreasing  $P.T_{s,\min}$ , breaking ties by increasing  $P.T_{s,\max}$  do
4    $[T_{s,\min}, T_{s,\max}] \leftarrow P.startInterval$ ;
5    $R \leftarrow P.entries$ ;                                     // each entry is  $(\mathcal{T}_e, \mathcal{H})$ 
6    $\mathcal{B}_{span} \leftarrow \{B \in \mathcal{B}_u \mid B \subseteq [T_{s,\min}, T_{s,\max}]\}$ ;
7   foreach  $B \in \mathcal{B}_{span}$  do
8      $\mathcal{I}_u[B] \leftarrow \mathcal{I}_u[B] \cup R$ ;
9 return  $\mathcal{I}_u$ ;

```

and appends the entire block R to the corresponding lists in $\mathcal{I}_u$ (lines 7–8). Due to the reverse start-time traversal, entries are appended in sorted order, allowing the PISA to be constructed in a single linear pass without additional sorting.

With PISA constructed, given a query $(u, [t_s, t_e])$, we first locate the base interval via binary search on t_s . We then perform a second binary search on the associated candidate end-time intervals for t_e to retrieve the target sketch.

Complexity Analysis. The construction phase consists of two steps: (1) SEB Generation: This process is output-sensitive, performing exactly $|\mathcal{D}_u|$ iterations for vertex u . Within each iteration, rigorous optimizations, such as skipping non-disruptive events and bypassing invariant boundary computations, ensure that the amortized cost c_u is bounded by m_u , i.e., the number of edges associated with vertex u in the projected graph. (2) PISA Construction: Leveraging the reverse-order strategy, this step is strictly linear with respect to the final index size $|\mathcal{I}_u|$. Summing over all nodes, the total time complexity is $O(\sum_{u \in \mathcal{V}} (|\mathcal{D}_u| \cdot c_u + |\mathcal{I}_u|))$.

The index storage comprises two optimized components: (1) Sketch Storage: for each vertex u , shared pointers are used to deduplicate repeated sketches, so the sketch storage only depends on the number of unique sketches $|\mathcal{H}_u|$, requiring $O(|\mathcal{H}_u| \cdot k)$ space. (2) Index Structure: the validity intervals and sketch pointers in PISA are linear in $|\mathcal{I}_u|$. Thus, the total space complexity is $\sum_{u \in \mathcal{V}} O(|\mathcal{H}_u| \cdot k + |\mathcal{I}_u|)$.

In the worst case, each of these quantities can be bounded by the number of possible time windows, i.e., $|\mathcal{D}_u|, |\mathcal{I}_u|, |\mathcal{H}_u| = O(t_{\max}^2)$. Moreover, since $c_u \leq m_u$ and $\sum_{u \in \mathcal{V}} m_u = O(m)$, where m is the number of temporal edges, substituting these bounds into the parameter-sensitive time complexity gives $O((m + n)t_{\max}^2)$. Similarly, the space complexity becomes $O(nt_{\max}^2(k + 1))$. In practice, temporal coherence and predictive jumping usually make $|\mathcal{D}_u| \ll t_{\max}^2$, as supported by Table 5.

6 EXPERIMENTS

6.1 Experimental Settings

Datasets. In our experiments, we use six real-world temporal graph datasets from SNAP [1]. Table 1 summarizes the key statistics for each dataset, including the number of vertices (n), temporal edges (m), the maximum timestamp ($t_{\max}$, indexed from 0), and its memory cost (*size(bytes)*). The temporal graph datasets are presented in ascending order based on m . All datasets, along with their corresponding detailed descriptions, are publicly available and sourced from SNAP. All timestamps in the datasets are recorded in Unix format.

Table 1. Dataset statistics.

Dataset	n	m	$t_{\max}$	size (bytes)
bitcoin (BTC)	5,881	35,592	35,591	284,736
dblp-cite (DC)	12,590	49,759	29	396,632
CollegeMsg (CM)	1,899	59,835	58,910	478,680
email-eu (EE)	986	332,334	207,879	2,658,672
wikipedia (WK)	1,140,149	7,833,140	2,320	173,541,969
flickr (FK)	2,302,926	33,140,017	133	265,120,136

Main Competitor. We test the following algorithms:

- OTCS: our online TSC search algorithm.
- MPC [55]: a maximal σ -periodic k -clique model to characterize periodic communities in a temporal graph.
- QTCS [45]: a query-centered temporal community search model based on a novel concept of Time-Constrained Personalized PageRank.
- PHC [86]: an index-based solution for historical k -core queries.
- pSCAN [9]: the state-of-the-art online SCAN algorithm.
- ITCS: our index-based TSC search algorithm.

OTCS is the direct *same-objective* baseline: it materializes the projected graph and performs the same TSC search without the index. MPC, QTCS, and PHC are *representative temporal community search baselines* from prior work. They are based on different temporal community models. pSCAN provides an objective-aligned SCAN baseline. We adapt it to temporal graphs by applying it to the projected graph of each query window and returning the community containing the query vertex.

Parameter settings. We adopt parameter settings consistent with prior studies on SCAN [9, 75, 89] and CS [26]. We vary the core threshold μ ($\mu \geq 2$) within $\{2, 3, 4, 5, 6\}$ (default $\mu = 2$) and the similarity threshold ϵ ($0 < \epsilon \leq 1$) within $\{0.2, 0.3, 0.4, 0.5, 0.6\}$ (default $\epsilon = 0.6$). Regarding the bottom- k sketch size, a trade-off exists between index efficiency (space and construction time) and query quality. Based on our sensitivity analysis (see Fig. 10), we set $k = 250$ for smaller datasets (BTC, DC, CM, EE) and $k = 2000$ for larger ones (WK, FK) to achieve an optimal balance. For each dataset, we randomly generate 15 time windows with different temporal spans, where the window lengths are set to $f \cdot t_{\max}$ for $f \in \{0.3, 0.5, 0.7\}$. For each time window, we randomly sample 100 query vertices. As a result, we obtain a total of 1,500 queries per dataset. All reported results are averaged over these 1,500 queries, and unless otherwise specified, default parameter settings are used. All algorithms are implemented in C++ (compiled with -O3) and executed on a server with Ubuntu 24.04 LTS, equipped with Intel Xeon Platinum 8358 CPUs @ 2.60GHz, and 2TB RAM.

6.2 Effectiveness Evaluation

In this subsection, we evaluate the effectiveness of our TSC model by comparing the quality of communities identified by different CS methods on temporal graphs. We also provide a case study to show that TSC can be used in event detection.

Exp. 1: The quality of community. Table 2 evaluates the retrieved communities using four metrics: density (Den.), Jaccard similarity (Jac.), diameter (Dia.), and clustering coefficient (CC). Den. and Jac. measure internal density and structural cohesion, while Dia. and CC further characterize community compactness and local connectivity. As shown in Table 2, ITCS consistently achieves competitive performance across all metrics. MPC always returns invalid results. QTCS performs well

Table 2. Effectiveness evaluation of retrieved communities. “-” denotes that the result is unavailable or that the algorithm fails to complete within 48 hours. Den., Jac., Dia., and CC denote density, Jaccard similarity, diameter, and clustering coefficient.

Methods f	BTC				DC				CM				EE				WK				FK				
	Den.	Jac.	Dia.	CC	Den.	Jac.	Dia.	CC	Den.	Jac.	Dia.	CC	Den.	Jac.	Dia.	CC	Den.	Jac.	Dia.	CC	Den.	Jac.	Dia.	CC	
MPC	0.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
	0.5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
	0.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
QTCS	0.3	0.36	0.13	2.14	0.63	0.11	0.09	5.08	0.49	0.32	0.07	2.28	0.55	0.60	0.19	2.19	0.75	0.14	0.22	2.17	0.66	-	-	-	-
	0.5	0.34	0.12	2.19	0.62	0.16	0.08	4.20	0.54	0.32	0.07	2.37	0.55	0.62	0.20	2.14	0.75	0.14	0.23	2.20	0.64	-	-	-	-
	0.7	0.36	0.11	2.11	0.65	0.17	0.07	4.30	0.54	0.33	0.07	2.39	0.54	0.62	0.21	2.15	0.75	-	-	-	-	-	-	-	-
PHC	0.3	0.18	0.13	3.79	0.34	0.01	0.01	6.29	0.19	0.04	0.03	4.30	0.09	0.18	0.12	3.53	0.44	0.14	0.10	2.88	0.30	0.08	0.05	3.50	0.24
	0.5	0.16	0.12	3.78	0.33	0.01	0.01	6.15	0.20	0.05	0.04	3.80	0.12	0.09	0.06	3.67	0.42	0.07	0.05	3.00	0.24	0.05	0.04	3.67	0.25
	0.7	0.10	0.07	4.00	0.32	0.01	0.01	6.35	0.20	0.07	0.05	3.75	0.13	0.15	0.10	3.50	0.45	0.05	0.04	3.11	0.20	0.05	0.04	3.83	0.23
pSCAN	0.3	0.11	0.10	6.21	0.33	0.06	0.07	8.39	0.25	0.11	0.08	4.81	0.19	0.13	0.07	5.00	0.57	0.05	0.07	6.21	0.30	0.17	0.24	11.55	0.27
	0.5	0.12	0.10	5.97	0.24	0.04	0.09	9.68	0.34	0.13	0.11	4.13	0.26	0.11	0.06	5.15	0.51	0.05	0.10	5.49	0.33	0.18	0.27	12.19	0.34
	0.7	0.11	0.08	6.27	0.29	0.03	0.05	9.59	0.38	0.15	0.08	4.63	0.21	0.14	0.08	4.37	0.59	0.03	0.13	4.31	0.29	0.18	0.27	11.07	0.37
ITCS	0.3	0.55	0.32	2.89	0.56	0.48	0.25	2.81	0.48	0.66	0.19	2.23	0.35	0.58	0.20	3.17	0.68	0.62	0.18	1.98	0.45	0.55	0.38	2.35	0.45
	0.5	0.63	0.38	2.50	0.59	0.37	0.20	3.86	0.55	0.71	0.16	2.28	0.53	0.54	0.21	2.94	0.67	0.76	0.20	2.01	0.35	0.60	0.38	2.31	0.50
	0.7	0.63	0.39	2.74	0.63	0.42	0.21	3.73	0.52	0.69	0.16	2.18	0.65	0.56	0.22	2.89	0.68	0.46	0.32	2.07	0.32	0.57	0.38	2.30	0.57

Table 3. Comparison with community detection methods.

Alg.	f	BTC		DC		CM		EE	
		Den.	Jac.	Den.	Jac.	Den.	Jac.	Den.	Jac.
Leiden	0.3	0.03	0.11	0.02	0.04	0.04	0.04	0.13	0.07
	0.5	0.02	0.10	0.01	0.03	0.04	0.04	0.14	0.08
	0.7	0.01	0.09	0.01	0.02	0.03	0.04	0.14	0.08
Louvain	0.3	0.03	0.10	0.02	0.04	0.04	0.04	0.13	0.07
	0.5	0.02	0.09	0.01	0.03	0.04	0.04	0.13	0.07
	0.7	0.01	0.08	0.01	0.02	0.03	0.03	0.14	0.08
LPA	0.3	0.09	0.15	0.10	0.18	0.03	0.02	0.04	0.03
	0.5	0.07	0.11	0.08	0.13	0.03	0.02	0.03	0.03
	0.7	0.03	0.07	0.05	0.09	0.02	0.01	0.04	0.03
ITCS	0.3	0.55	0.32	0.48	0.25	0.66	0.19	0.58	0.20
	0.5	0.63	0.38	0.37	0.20	0.71	0.16	0.54	0.21
	0.7	0.63	0.39	0.42	0.21	0.69	0.16	0.56	0.22

on some small datasets but fails to scale to FK and some WK settings within the 48-hour cutoff, pSCAN is a strong baseline on large datasets but still underperforms ITCS. These results show that ITCS remains competitive while preserving scalability on large temporal graphs.

Exp. 2: Comparison with community detection methods. We further compare with representative global community detection methods: Leiden [68], Louvain [6], and LPA [57]. For each query window, we run them on the projected graph and return the community containing the query vertex. Table 3 reports Den. and Jac. on the small datasets. ITCS achieves significantly higher quality in all settings. On WK and FK, these baselines fail to finish within the 48-hour cutoff, so their results are omitted. This indicates that global community detection methods are less suitable for large-scale arbitrary-window TSC queries.

Exp. 3: Case study. We conducted a case study on the tgbl-coin stablecoin dataset [35] to analyze structural evolution during the market turbulence of 2022. We partition the data into 15 weekly snapshots from April 2 to July 9, 2022, and run ITCS on each window to obtain TSCs. These communities are then compared across windows using membership overlap or Jaccard similarity, which enables tracking community evolution over time. Thanks to PISA, each window query can

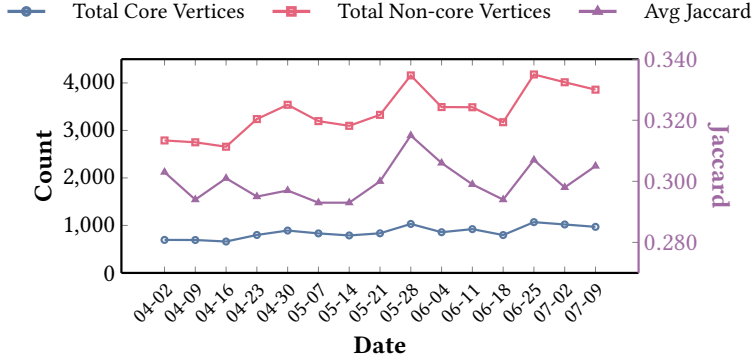
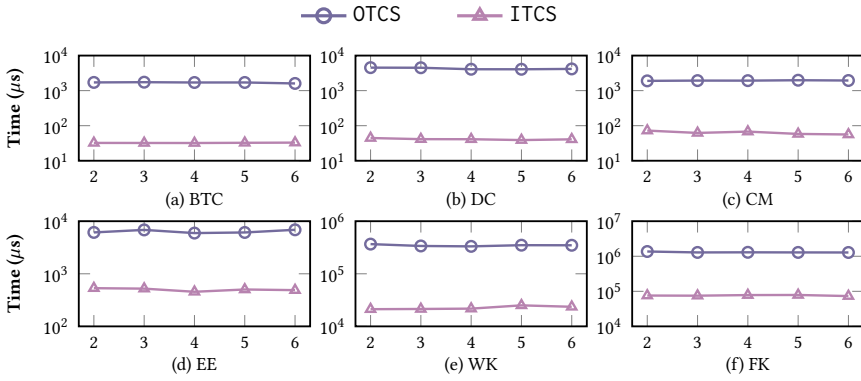


Fig. 5. Evolution of vertex roles and Jaccard similarity.

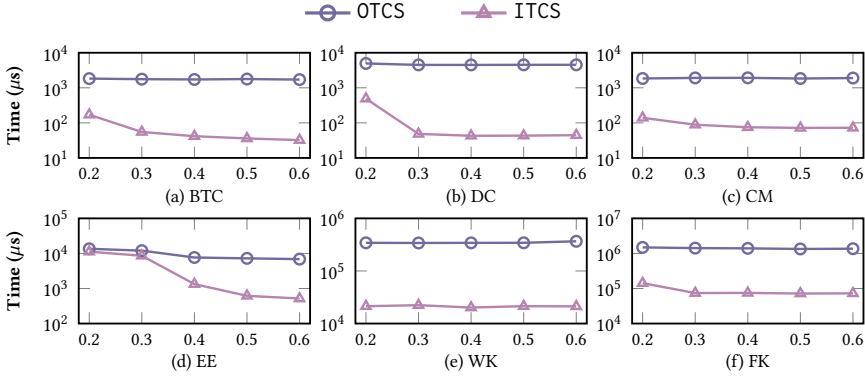
Fig. 6. Query time for different μ values ($\epsilon = 0.6$).

be answered efficiently; For each snapshot, we aggregate the numbers of core and non-core vertices and compute the average pairwise Jaccard similarity within the returned TSCs to measure cohesion. As shown in Fig. 5, the core/non-core counts and average Jaccard similarity change consistently over time. Their sharp drops align with two major market events: the Terra (LUNA) crash during May 7–13 and the liquidity crisis during June 12–18. These results show that running ITCS over consecutive windows can track community development and reveal structural responses to major events.

6.3 Efficiency Evaluation

In this subsection, we evaluate the efficiency of our method by analyzing its scalability across datasets, parameter sensitivity, and the comparison with the exact solutions.

Exp. 1: Query performance on different datasets. Fig. 8 reports the average query time of different methods. ITCS achieves consistently low query latency and substantially reduces query time compared with OTCS and pSCAN, because PISA avoids online projected-graph reconstruction and sketch computation. The speedup is particularly clear on dense datasets. On EE, ITCS answers a query in $517 \mu s$, while QTCS takes about seven minutes and fails to finish on WK and FK within the 48-hour cutoff. Although PHC is efficient on large datasets, its lower quality in Table 2 reflects

Fig. 7. Query time for different ϵ values ($\mu = 2$).

an efficiency–quality trade-off. Overall, ITCS provides fast query processing while preserving high-quality communities.

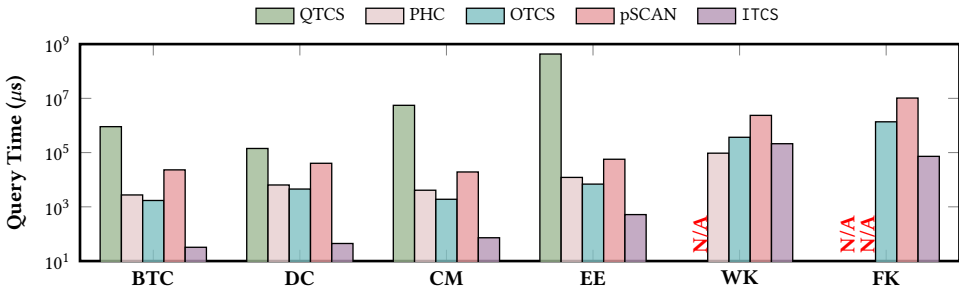


Fig. 8. Query time on different datasets.

Exp. 2: Query performance under different parameters. We evaluate the query efficiency of ITCS against OTCS. Due to space constraints, we report results for a representative window duration of $0.7 \cdot t_{\max}$ under two settings: varying μ with fixed $\epsilon = 0.6$ (Fig. 6), and varying ϵ with fixed $\mu = 2$ (Fig. 7). Overall, ITCS consistently outperforms OTCS, achieving a speedup of up to two orders of magnitude across most configurations. While query time for both methods naturally decreases as ϵ increases due to smaller resulting communities (Fig. 7), the dense EE dataset presents an exception at low ϵ (0.2, 0.3). Here, communities expand to cover nearly half the graph, causing ITCS’s overhead to approach that of OTCS. However, in typical scenarios where communities are distinct subgraphs, ITCS maintains a significant performance lead. Furthermore, when varying μ , both methods exhibit stable performance because the strict similarity threshold ($\epsilon = 0.6$) already effectively limits the search space; crucially, ITCS retains its significant efficiency advantage throughout this range, confirming the robustness of our indexing and pruning strategies.

Exp. 3: Query time under practical fixed windows. We further evaluate query performance on WK and FK using Δ values of 7, 14, 30, 60, and 90 days. Fig. 9 reports the results. Under narrow windows, OTCS becomes more competitive because reconstructing and searching a small projected graph is inexpensive, explaining the narrowed gap on WK. On FK, whose short-window projected

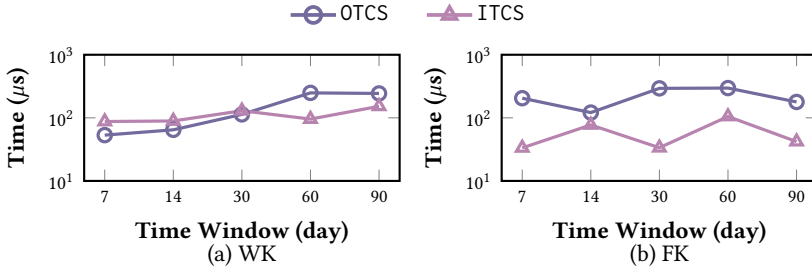


Fig. 9. Query time under different time windows.

graphs remain relatively dense, ITCS maintains a consistent advantage. Overall, ITCS applies to realistic short-window queries, with greater speedups on larger or denser projected graphs.

Exp. 4: Approximate vs. exact solutions. We evaluate our approximate algorithm against exact OTCS, treating its output as the ground truth. Following [26, 89], we use F1-score and Adjusted Rand Index (ARI). Table 4 reports their mean and standard deviation for window-length ratios $f \in \{0.3, 0.5, 0.7\}$, where each query is defined by a sampled window and query vertex. The consistently high scores and small deviations show that our approximate results closely match exact OTCS and remain stable across windows and query nodes.

Table 4. Community quality metrics under different f .

Data	$f = 0.3$		$f = 0.5$		$f = 0.7$	
	F1	ARI	F1	ARI	F1	ARI
BTC	99.47±0.45	99.41±0.49	99.61±0.32	99.56±0.36	99.46±0.47	99.39±0.51
DC	98.57±1.17	98.49±1.23	98.47±1.14	98.40±1.20	98.24±1.52	98.15±1.60
CM	99.46±0.42	99.40±0.46	99.50±0.38	99.45±0.41	99.31±0.55	99.26±0.59
EE	94.68±3.19	92.77±2.86	95.23±3.05	93.41±2.79	95.33±2.90	93.55±2.61
FK	99.68±0.28	99.63±0.32	99.54±0.40	99.48±0.45	99.65±0.30	99.60±0.34
WK	99.86±0.12	99.82±0.15	99.95±0.04	99.91±0.07	99.93±0.06	99.89±0.09

6.4 Detailed Analysis of Index Properties

In this subsection, we first report the index size and construction time of PISA on each dataset. We then analyze how different time-window settings affect both index size and construction cost. Finally, we study the effectiveness of PISA by reporting the number of SEBs generated on each dataset.

Exp. 1: Index size and construction time on different datasets. In this experiment, we report the index size and construction time of PISA on each dataset under the different k values. As shown in Fig. 10, we make the following observations: 1) the index size depends not only on k , but also on the number of SEBs. In general, it decreases as k increases because a larger k typically produces fewer SEBs. However, on BTC and WK, k dominates the index size, which therefore grows approximately proportionally to k ; 2) index construction time follows a similar trend: it generally decreases with k as fewer SEBs are produced, but increases on BTC and WK where k dominates; and 3) both costs remain practical for real-world applications.

Exp. 2: Compressed index size and peak RAM. To further compress the index size, we apply lossless checkpointed differential encoding, which stores only sketch differences between adjacent checkpoints. Fig. 12 reports compressed size and construction-time peak RAM, normalized by raw

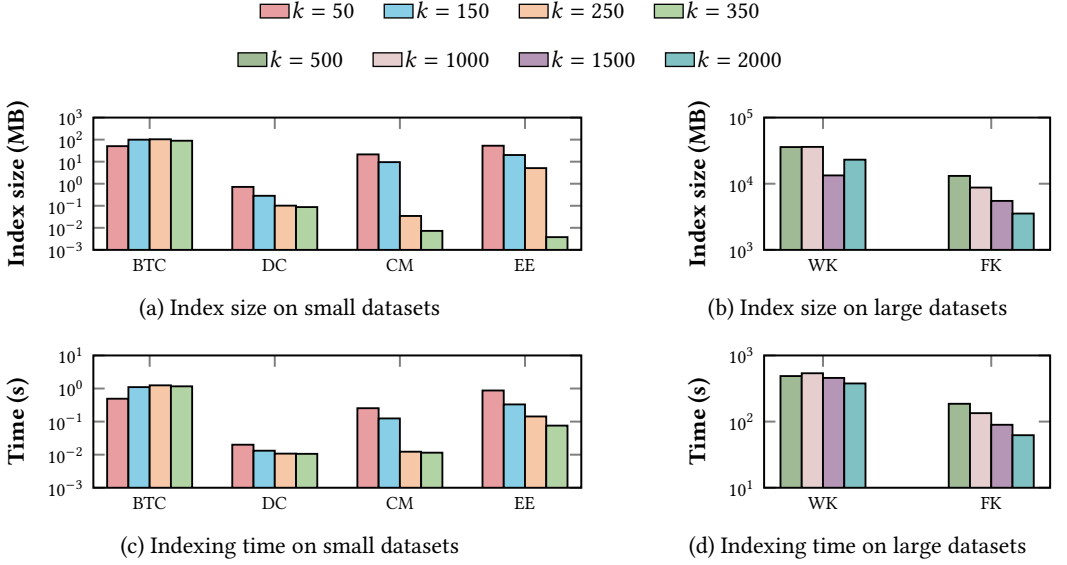
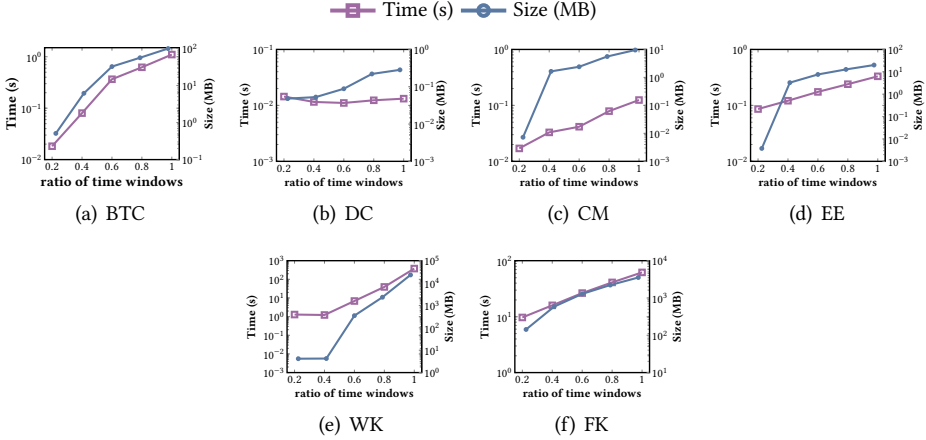
Fig. 10. Varying k on different datasets.

Fig. 11. Scalability test of indexing time and size.

data size. Compressed size stays below $10\times$ on WK and FK for every tested k , and below $5\times$ on average. Peak RAM is higher because ConstructSEB temporarily materializes raw sketches. Size is not proportional to k : larger sketches may produce fewer SEBs, and the default $k = 2000$ gives the lowest ratio on WK and FK.

Exp. 3: Scalability testing. We evaluate Algorithm 5 by progressively increasing the temporal span. For each dataset, we generate five subgraphs containing edges in $[0, f \cdot t_{\max}]$, where $f \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. Fig. 11 shows that construction time and index size increase monotonically with f , as larger windows contain more temporal edges.

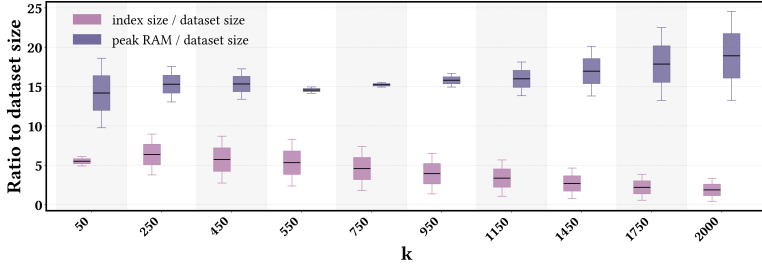


Fig. 12. Relative memory and index size w.r.t. dataset size.

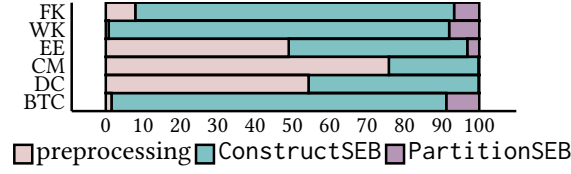


Fig. 13. Proportion of time cost of each step in PISA.

Exp. 4: The number of sketch equivalence blocks. To assess the effectiveness of PISA, Table 5 reports the average number of SEBs and the compression ratio $\rho = t_{\max}^2 / \text{Avg. SEBs}$. The number of SEBs is significantly smaller than $t_{\max}^2$, indicating that each SEB covers multiple valid windows. Thus, PISA decouples storage cost from the quadratic growth of $t_{\max}^2$.

Table 5. Index compression statistics.

Dataset	Avg. Num. of SEBs	$t_{\max}^2$	ρ
BTC	13,313.00	1.27×10^9	9.51×10^4
DC	7.86	8.41×10^2	1.07×10^2
CM	28.00	3.47×10^9	1.24×10^8
EE	5,241.00	4.32×10^{10}	8.25×10^6
WK	15,485.35	5.38×10^6	3.48×10^2
FK	210.30	1.77×10^4	8.41×10^1

Exp. 5: Breakdown of index construction time. Fig. 13 breaks construction time into preprocessing, ConstructSEB, and PartitionSEB. ConstructSEB dominates on most datasets, while PartitionSEB is consistently lightweight. On small datasets such as CM, preprocessing dominates because few vertices require SEB construction, exposing its fixed overhead.

7 RELATED WORK

We review community search (CS), community detection, and graph sketching.

• **Community search on static graphs.** CS problem aims to find structurally cohesive connected subgraphs containing a specific vertex or a set of vertices [21, 26, 27, 63]. To measure the structural cohesiveness of a community, people often use the cohesive subgraph models [26], like k -core [4, 7], k -truss [19, 90], k -clique [20, 87, 95] and k -edge connected component (k -ECC)[10]. Besides, many CS works have considered vertices' attributes [13, 26, 37], and vertices' importance values [5, 14, 37, 40–43, 48, 78]. Some works have studied CS over heterogeneous information networks

[23, 27, 33, 38, 54, 71, 83, 91]. For example, Fang et al. and Zhou et al. studied CS over heterogeneous information networks by using the $(k, \mathcal{P})$ -core model [27, 93]; Jian et al. searched communities with vertices of multiple types by using relational constraints [38]. Recently, we noticed that Wang et al. [72] addressed structural similarity community search in HINs. In contrast to our work, they focus on static networks and do not consider temporal dynamics.

• **Community search on temporal graphs.** Galimberti et al. [34] proposed the span-core model, which persistently maintains a k -core in every timestamp of a small time window. Afterwards, the existing works studied on historical k -cores queries [73, 82, 86]. Wu et al. [77] introduced the (k, h) -core, with Bai et al. [3] focusing on its maintenance. Li et al. [44] proposed the (θ, τ) -persistent k -core. Qin et al. [56] studied periodic k -cores and k -clique [55]. Recent work also considered time-constrained personalized PageRank scores to identify communities containing nodes with higher relevance scores [45]. However, none of these works considered structural similarity between pairs of nodes or the roles of members within temporal communities. In this paper, we aim to fill this gap.

• **Community detection.** In the literature, various community detection methods have been proposed [28, 29, 52]. Unlike community search, these methods aim to identify all communities in a network, rather than focusing on communities centered around a given query vertex. Representative methods include spectral clustering [3, 76, 84], modularity-based approaches [16, 24, 51, 52], such as Louvain and Leiden [6, 68], label-propagation approaches, such as LPA [57], statistical inference-based approaches [39, 58], and structural similarity-based approaches [9, 12, 46, 60, 70, 75, 80, 89]. Some recent works [61, 62, 64–67, 94, 96] have studied community detection in heterogeneous information networks, and temporal graphs [31, 47, 85].

• **Graph sketching and Jaccard similarity estimation.** Graph sketching summarizes graphs and streams for sparsification, spanners, subgraph estimation, and distance estimation [2, 17, 49]. Set-similarity estimators such as MinHash [8], SimHash [11], and DotHash [53] approximate pairwise similarities such as Jaccard. In contrast, our SEB and PISA exploit temporal locality to reuse window-specific neighborhood sketches and answer arbitrary window queries without reconstructing projected graphs.

8 CONCLUSION

In this paper, we propose a novel community model, Temporal Structurally Similar Community (TSC), which explicitly captures the evolution of structural roles across different time windows. To support efficient TSC queries, we developed a fast online algorithm and an optimized index-based approach capable of handling queries over arbitrary time windows. Extensive experiments on real-world datasets demonstrated that our algorithms significantly outperform baselines in terms of query efficiency. Furthermore, qualitative case studies confirmed the effectiveness of the TSC model in revealing meaningful, evolving structural patterns in temporal graphs.

Acknowledgments

Yue Zhang and Chenhao Ma were partially supported by the National Natural Science Foundation of China (Grant 62302421), the Basic and Applied Basic Research Fund in Guangdong Province (Grant 2025A1515010439), the CCF-Ant Research Fund, the 1+1+1 CUHK-CUHK(SZ)-GDSTC Joint Collaboration Fund (Grant 2025A0505000079), and the Guangdong Provincial Key Laboratory of Big Data Computing, The Chinese University of Hong Kong, Shenzhen. Xiaolin Han was supported by the National Natural Science Foundation of China (Grant 62302397), the Young Talent Fund of Xi'an Association for Science and Technology (Grant 0959202613046), the China Postdoctoral Science Foundation (Grant 2025M774364), and the Shaanxi Post-doctoral Research Project (Grant 2025BSHSDZZ106).

References

- [1] 2014. SNAP Datasets. <http://snap.stanford.edu/data>.
- [2] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2012. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*. 5–14.
- [3] Wen Bai, Yadi Chen, and Di Wu. 2020. Efficient temporal core maintenance of massive graphs. *Information Sciences* 513 (2020), 324–340.
- [4] Vladimir Batagelj and Matjaz Zaversnik. 2003. An $o(m)$ algorithm for cores decomposition of networks. *arXiv preprint cs/0310049* (2003).
- [5] Fei Bi, Lijun Chang, Xuemin Lin, and Wenjie Zhang. 2017. An optimal and progressive approach to online search of top-k influential communities. *arXiv preprint arXiv:1711.05857* (2017).
- [6] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.
- [7] 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.
- [8] Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*. IEEE, 21–29.
- [9] Lijun Chang, Wei Li, Lu Qin, Wenjie Zhang, and Shiyu Yang. 2017. pSCAN: Fast and Exact Structural Graph Clustering. *IEEE Trans. Knowl. Data Eng.* 29, 2 (2017), 387–401.
- [10] 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.
- [11] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*. 380–388.
- [12] Yulin Che, Shixuan Sun, and Qiong Luo. 2018. Parallelizing pruning-based graph structural clustering. In *Proceedings of the 47th International Conference on Parallel Processing*. 1–10.
- [13] 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.
- [14] 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 *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 1553–1562.
- [15] 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.
- [16] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 70, 6 (2004), 066111.
- [17] Edith Cohen. 2014. All-distances sketches, revisited: HIP estimators for massive graphs analysis. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 88–99.
- [18] Edith Cohen and Alex Arenas. 2007. Summarizing data using bottom-k sketches. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. 225–234.
- [19] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report* 16, 3.1 (2008), 1–29.
- [20] 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.
- [21] 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.
- [22] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Mikkel Thorup. 2017. Fast similarity sketching. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 663–671.
- [23] Zheng Dong, Xin Huang, Guorui Yuan, Hengshu Zhu, and Hui Xiong. 2021. Butterfly-core community search over labeled graphs. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2006–2018.
- [24] Jordi Duch and Alex Arenas. 2005. Community detection in complex networks using extremal optimization. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 72, 2 (2005), 027104.
- [25] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective community search for large attributed graphs. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1233–1244.
- [26] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *The VLDB Journal* 29, 1 (2020), 353–392.
- [27] 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.
- [28] Santo Fortunato. 2010. Community detection in graphs. *Physics reports* 486, 3-5 (2010), 75–174.

- [29] Santo Fortunato and Darko Hric. 2016. Community detection in networks: A user guide. *Physics reports* 659 (2016), 1–44.
- [30] 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.
- [31] 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.
- [32] 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.
- [33] Edoardo Galimberti, Francesco Bonchi, and Francesco Gullo. 2017. Core decomposition and densest subgraph in multilayer networks. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1807–1816.
- [34] 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.
- [35] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. 2023. Temporal graph benchmark for machine learning on temporal graphs. *Advances in Neural Information Processing Systems* (2023).
- [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] 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.
- [39] Johan H Koskinen and Tom AB Snijders. 2007. Bayesian inference for dynamic social network data. *Journal of statistical planning and inference* 137, 12 (2007), 3930–3938.
- [40] 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.
- [41] 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 *Proceedings of the 2018 international conference on management of data*. 457–472.
- [42] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2015. Influential community search in large networks. *Proceedings of the VLDB Endowment* 8, 5 (2015), 509–520.
- [43] 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.
- [44] 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.
- [45] 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.
- [46] Kaixin Liu, Sibao 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.
- [47] 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).
- [48] 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.
- [49] Andrew McGregor. 2014. Graph stream algorithms: a survey. *ACM SIGMOD Record* 43, 1 (2014), 9–20.
- [50] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient computation of frequent and top-k elements in data streams. In *International conference on database theory*. Springer, 398–412.
- [51] Mark EJ Newman. 2004. Fast algorithm for detecting community structure in networks. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 69, 6 (2004), 066133.
- [52] Mark EJ Newman and Michelle Girvan. 2004. Finding and evaluating community structure in networks. *Physical review E* 69, 2 (2004), 026113.
- [53] Igor Nunes, Mike Heddes, Pere Vergés, Danny Abraham, Alex Veidenbaum, Alex Nicolau, and Tony Givargis. 2023. Dothash: estimating set similarity metrics for link prediction and document deduplication. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*. 1758–1769.
- [54] Lianpeng Qiao, Zhiwei Zhang, Ye Yuan, Chen Chen, and Guoren Wang. 2021. Keyword-centric community search over large heterogeneous information networks. In *International Conference on Database Systems for Advanced Applications*. Springer, 158–173.

- [55] Hongchao Qin, Rong-Hua Li, Guoren Wang, Lu Qin, Yurong Cheng, and Ye Yuan. 2019. Mining periodic cliques in temporal networks. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1130–1141.
- [56] 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.
- [57] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 76, 3 (2007), 036106.
- [58] Jörg Reichardt and Stefan Bornholdt. 2006. Statistical mechanics of community detection. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 74, 1 (2006), 016110.
- [59] Boyu Ruan, Junhao Gan, Hao Wu, and Anthony Wirth. 2021. Dynamic Structural Clustering on Graphs. In *SIGMOD*. 1491–1503.
- [60] 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*. 1491–1503.
- [61] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and Philip S Yu. 2016. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (2016), 17–37.
- [62] 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 *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. 699–708.
- [63] 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.
- [64] 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).
- [65] 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 *Proceedings of the 12th international conference on extending database technology: advances in database technology*. 565–576.
- [66] 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.
- [67] Yizhou Sun, Yintao Yu, and Jiawei Han. 2009. Ranking-based clustering of heterogeneous information networks with star network schema. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 797–806.
- [68] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. 2019. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific reports* 9, 1 (2019), 5233.
- [69] Tom Tseng, Laxman Dhulipala, and Julian Shun. 2021. Parallel Index-Based Structural Graph Clustering and Its Approximation. In *SIGMOD*. 1851–1864.
- [70] 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.
- [71] Ruby W Wang and Fred Y Ye. 2019. Simplifying weighted heterogeneous networks by extracting h-structure via s-degree. *Scientific reports* 9, 1 (2019), 18819.
- [72] Shu Wang, Yixiang Fang, and Wensheng Luo. 2025. Searching and Detecting Structurally Similar Communities in Large Heterogeneous Information Networks. *Proceedings of the VLDB Endowment* 18, 5 (2025), 1425–1438.
- [73] Zhi Wang, Ming Zhong, Yuanyuan Zhu, Tiejun Qian, Mengchi Liu, and Jeffrey Xu Yu. 2025. On more efficiently and versatily querying historical k-cores. *Proceedings of the VLDB Endowment* 18, 5 (2025), 1335–1347.
- [74] Dong Wen, Yilun Huang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2020. Efficiently answering span-reachability queries in large temporal graphs. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1153–1164.
- [75] Dong Wen, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2019. Efficient structural graph clustering: an index-based approach. *VLDB J.* 28, 3 (2019), 377–399.
- [76] 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*. SIAM, 274–285.
- [77] 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.
- [78] 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.
- [79] 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.

- [80] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas AJ Schweiger. 2007. Scan: a structural clustering algorithm for networks. In *SIGKDD*. 824–833.
- [81] Huihui Yang, Chunxue Zhu, Longlong Lin, and Pingpeng Yuan. 2024. Towards Truss-Based Temporal Community Search. *arXiv preprint arXiv:2410.15046* (2024).
- [82] Junyong Yang, Ming Zhong, Yuanyuan Zhu, Tiejun 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.
- [83] Yixing Yang, Yixiang Fang, Xuemin Lin, and Wenjie Zhang. 2020. Effective and efficient truss computation over large heterogeneous information networks. In *2020 IEEE 36th international conference on data engineering (ICDE)*. IEEE, 901–912.
- [84] 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*. 555–564.
- [85] 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.
- [86] 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).
- [87] 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.
- [88] Nurcan Yuruk, Xiaowei Xu, and Thomas A. J. Schweiger. 2008. On Structural Analysis of Large Networks. In *HICSS*. 143.
- [89] Fangyuan Zhang and Sibo Wang. 2022. Effective indexing for dynamic structural graph clustering. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2908–2920.
- [90] Yikai Zhang and Jeffrey Xu Yu. 2019. Unboundedness and efficiency of truss maintenance in evolving graphs. In *Proceedings of the 2019 International Conference on Management of Data*. 1024–1041.
- [91] 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.
- [92] 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.
- [93] 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.
- [94] Yingli Zhou, Yixiang Fang, Chenhao Ma, Tianci Hou, and Xin Huang. 2024. Efficient Maximal Motif-Clique Enumeration over Large Heterogeneous Information Networks. *Proceedings of the VLDB Endowment* 17, 11 (2024), 2946–2959.
- [95] 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.
- [96] 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.

Received 17 January 2026; revised 19 May 2026; accepted 12 June 2026