Finding Time-Proximity Communities in Temporal Heterogeneous Information Networks

Yifu Tang
Swinburne University of Technology
Melbourne, Australia
craigtang@swin.edu.au

Chengfei Liu
Swinburne University of Technology
Melbourne, Australia
cliu@swin.edu.au

Lu Chen
Swinburne University of Technology
Melbourne, Australia
luchen@swin.edu.au

Rui Zhou
Swinburne University of Technology
Melbourne, Australia
rzhou@swin.edu.au

Jianxin Li
Edith Cowan University
Perth, Australia
jianxin.li@ecu.edu.au

ABSTRACT

Community search in heterogeneous information networks (HINs) often neglects temporal dynamics, yielding structures that poorly reflect real-world interactions. We introduce the Temporal HIN Community Search (THCS) problem and propose a novel $(k, T_q, \mathcal{P}_\delta)$-core model that captures both structural cohesiveness and temporal relevance. Our model uses a time span constraint δ to ensure interaction recency and a query interval T_q for flexible temporal exploration, filtering irrelevant connections while preserving structural density. We develop two efficient online algorithms—Center-based Sliding Window search and Incremental Center Expansion—that exploit meta-path symmetry and dynamic connectivity tracking. For frequent queries, we design a Temporal HIN Core Interval-Index (TCI-Index), organising minimal core intervals hierarchically with innovative compression techniques. Experiments on real-world datasets show our methods significantly outperform baselines, finding temporally meaningful communities with high efficiency.

PVLDB Reference Format:
Yifu Tang, Chengfei Liu, Lu Chen, Rui Zhou, and Jianxin Li. Finding Time-Proximity Communities in Temporal Heterogeneous Information Networks. PVLDB, 18(13): 5740 - 5752, 2025.
doi:10.14778/3773731.3773747

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Craig-Tang/THCS.

1 INTRODUCTION

Community search (CS) in networks aims to find cohesively connected groups. While extensively studied in homogeneous networks, the problem becomes more complex in heterogeneous information networks (HINs) where multiple entity and relationship types exist. In HINs, meta-path based approaches [29] establish connections between same-type vertices that are semantically related, enabling classical community models (e.g., k-core, k-truss) to be extended for HIN community search [5, 6, 9, 36]. These models have been successfully applied in various domains such as patent networks, e-commerce platforms, and bibliographic databases [2, 11, 26–28, 38, 42].

Although many HIN community models exist, they operate primarily in a *static* setting, disregarding the temporal evolution of interactions. In real-world applications, relationships evolve over time, and failing to incorporate temporal constraints can lead to the inclusion of outdated or irrelevant connections in detected communities. While *temporal community search* has been studied in homogeneous graphs [15, 17, 19], enforcing both structural density and temporal proximity, these approaches focus on direct edge connections and fail to address the multi-relational nature of HINs.

Extending Temporal Constraints to HINs. In this work, we focus on the popular $(k, \mathcal{P})$-core [6] approach. It uses a meta-path $\mathcal{P}$ to establish $\mathcal{P}$-neighbors over same-type nodes, forming a derived homogeneous graph $G_{\mathcal{P}}$ where each node must have at least k $\mathcal{P}$-neighbors for structural cohesiveness. For temporal HINs (THINs) with timestamped edges, simply enforcing a time window on $(k, \mathcal{P})$-core creates challenges. Since $\mathcal{P}$-neighbor spans vary dramatically—from short durations to periods as long as the overall community window—it fails to capture connection-level temporal proximity. Alternatively, enforcing short spans on each $\mathcal{P}$-neighbor pair may result in communities spanning excessively long periods, compromising overall temporal proximity.

To secure $\mathcal{P}$-neighbor proximity, community proximity, apart from community cohesiveness, we adapt $(k, \mathcal{P})$-core to $(k, T_q, \mathcal{P}_\delta)$-core as follows:

Temporal Proximity aware $\mathcal{P}$-neighbor ($\mathcal{P}_\delta$-neighbor): We introduce a new parameter δ to limit the maximum length of the span of the $\mathcal{P}$-neighbor. This ensures the temporal proximity at the neighbor-level.

Community Cohesiveness: Similar to $(k, \mathcal{P})$-core, we adopt the minimum degree constraint k for structural cohesiveness, but apply it to valid $\mathcal{P}_\delta$-neighbors.

Community-Wide Temporal Proximity: We introduce another temporal parameter, the query time window T_q, to limit the overall temporal span of the community and ensure it reflects users' specific periods of interest.

$(k, T_q, \mathcal{P}_\delta)$-core has many real-world applications, below are two examples:

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 18, No. 13 ISSN 2150-8097.
doi:10.14778/3773731.3773747

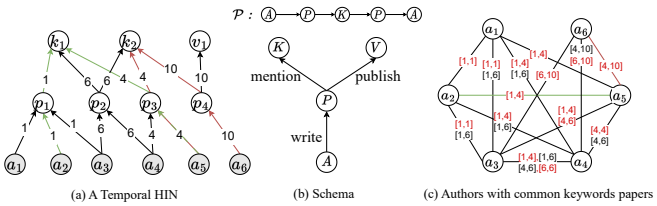
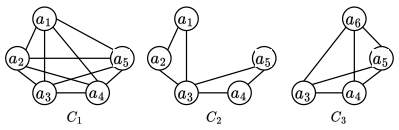


Figure 1: A THIN of DBLP (a), its schema (b), and the derived graph with connection intervals (c).



Temporal Communities	T_q	δ	k_{max}
C_1	[1,10]	3	4
C_2	[1,10]	2	2
C_3	[4,10]	4	3

Figure 2: Illustration of $(k, T_q, \mathcal{P}_\delta)$-cores with varying settings.

Example 1.1. *Figure 1 illustrates an example THIN from DBLP, with four node types: author (A), paper (P), venue (V), and keyword (K), and their relationships as shown in the schema (Figure 1(b)). Edges represent relationships, with timestamps indicating the interaction times (for simplicity, we index calendar years to integers). Consider the meta-path APKPA, which identifies authors publishing with common keywords. In THINs, such connections require time to form—the connection between a_2 and a_5 requires three years, while a_5 and a_6 requires six years. Figure 1(c) shows the derived graph $G_{\mathcal{P}}$, where connections may consist of multiple instances with different intervals.*

For $(k, T_q, \mathcal{P}_\delta)$-core search, T_q defines the period of interest, δ ensures that research interest convergence occurs within a narrow timeframe, and k guarantees connection intensity. Figure 2 demonstrates this flexibility: with $T_q = [1, 10]$ and $\delta = 3$, we find a 4-core community C_1 where all meta-path instances satisfy the δ constraint. When δ is reduced to 2, some instances become invalid, yielding a maximum 2-core community C_2.

Example 1.2. *A Real-World Application. Our model is well-suited for identifying high-risk groups in disease spread modeling. Consider a trajectory network where the meta-path $\mathcal{P}$ = Person-Location-Person connects individuals who visited the same location. A $(k, T_q, \mathcal{P}_\delta)$-core community represents a group with significant transmission potential. Here, δ captures the virus's viability window (e.g., within 2 hours), ensuring connections are temporally relevant for transmission. The query interval T_q allows epidemiologists to focus on a specific outbreak phase, such as the critical first week. The parameter k quantifies the transmission risk, requiring each person to have at least k potential exposure links to others within the same community. This identifies not just isolated pairs, but dense communities where the higher number of contacts elevates the overall risk and potential for rapid disease propagation.*

Technical Challenges and Our Approaches. These modeling requirements introduce substantial computational challenges. Unlike static approaches that confirm meta-path existence, temporal constraints demand a much finer-grained evaluation as multiple meta-path instances can connect the same target node pairs, each with a unique formation interval. Consequently, the search space for valid connections becomes vast, rendering a naive enumeration of all meta-path instances computationally infeasible.

To address this challenge, we develop efficient online algorithms. Our Center-based Sliding Window search leverages meta-path symmetry. Building on this, our Incremental Center Expansion method incorporates an Incremental Connectivity Tracker that efficiently maintains and updates neighborhood relationships as the temporal window slides, offering significant performance gains. These are evaluated against a Baseline Sliding Window algorithm.

To support frequent user searches efficiently, we also develop a novel index-based solution, the THIN Core Interval-Index (TCI-Index). While prior work like the PHC-Index by Yu et al. [37] effectively handles historical k-core queries in temporal *homogeneous* graphs by recording valid time intervals for node coreness, extending this to THINs presents significant hurdles. The primary challenge lies in the complexity introduced by meta-paths: instead of direct timestamped edges, we must evaluate δ-constrained multi-edge meta-path instances. This makes determining valid time intervals for core membership under our $(k, T_q, \mathcal{P}_\delta)$-core definition far more intricate. Our TCI-Index tackles this by first efficiently identifying time intervals where meta-path based neighborhoods are valid under the δ constraint, and then determining the minimal intervals where nodes satisfy the k-core property with respect to these temporally valid HIN connections. By organizing these findings into an interval-based framework and employing compression techniques, the TCI-Index enables rapid retrieval of $(k, T_q, \mathcal{P}_\delta)$-cores without redundant computations.

Contributions. Our key contributions are summarized as follows:

- We formally define the $(k, T_q, \mathcal{P}_\delta)$-*core* model for THINs, introducing a novel framework that integrates structural cohesiveness with temporal proximity constraints, addressing the community search problem in THINs. (**Section 2**)
- We develop efficient online search algorithms, that exploit meta-path symmetry and Incremental Connectivity Tracker to dynamically maintain temporal neighborhoods, demonstrating superior performance. (**Section 3**)
- We introduce a compact TCI-Index that organizes minimal temporal intervals in a hierarchical structure, enabling fast community retrieval for varying temporal and structural parameters. (**Sections 4 & 5**)
- We conduct extensive experiments on real-world datasets to evaluate the efficiency of searching methods and effectiveness of our proposed model. (**Section 6**)

2 PRELIMINARIES AND PROBLEM DEFINITION

We list frequently used notations in Table 1.

2.1 Preliminaries

Firstly, we introduce the concept of heterogeneous information networks (HIN), which serves as the foundation for our temporal model.

Definition 2.1 (HIN). A Heterogeneous Information Network (HIN) is defined as a directed graph $H = (V, E, \phi, \psi)$, where V and E denote the sets of nodes and edges, respectively. Besides, two type

Table 1: Notation

Symbol	Description
$\mathcal{H}$, $\mathcal{H}_T$	THIN and its subgraph in interval T
$\mathcal{V}$, $\mathcal{E}$, $\mathcal{T}$	Sets of nodes, edges and timestamps
ϕ, ψ	Node/edge type mapping functions
$\mathcal{A}$, $\mathcal{R}$	Sets of node/edge types
$\mathcal{P}$, $\mathcal{P}_{half}$, p	Meta-path, half meta-path, meta-path instance
T_q=$[q_s, q_e]$, T_p	Query/Meta-path instance time interval
δ, k	Time span and core number constraints
$G_{\mathcal{P}_\delta}$	Derived target node graph under constraints
mni, mci_k	Minimal neighbor/core intervals

mapping functions $\phi : V \to \mathcal{A}$ and $\psi : E \to \mathcal{R}$ indicate the sets of types of nodes and edges, satisfying $|\mathcal{A}| + |\mathcal{R}| > 2$.

An HIN follows a meta template called a network schema, which is a directed graph defined over node types $\mathcal{A}$ and edge types $\mathcal{R}$. As shown in Figure 1(b) is the network schema for the academic network, which defines all the allowable edge types between node types. Note that in the network schema, if an edge type R exists between node type A and B (i.e. ARB), the inverse relation $BR^{-1}A$ naturally holds.

Definition 2.2 (Meta Path)**.** A meta path $\mathcal{P}$ is a path defined on the network schema, denoted as $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} ... \xrightarrow{R_l} A_{l+1}$, where l is the length of $\mathcal{P}$, $A_i \in \mathcal{A}$ and $R_i \in \mathcal{R} (1 \le i \le l)$.

For simplicity, we also denote $\mathcal{P}$ as an ordered set in forms of $(A_1, A_2, ..., A_{l+1})$. We call a meta-path $\mathcal{P}$ a symmetric path if it is identical to its reverse path $\mathcal{P}^{-1}$. For example, in Figure 1(c), a meta-path $APVPA$ is symmetric. In this paper, we focus on symmetric $\mathcal{P}$. Below, when referring to a meta-path $\mathcal{P}$, $\mathcal{P}$ is symmetric.

We call a path $p = (a_1 \to a_2 ... \to a_{l+1})$ an instance of $\mathcal{P}$, if $\forall i \in [1, l+1]$, node a_i and edge $e_i = (a_i, a_{i+1})$, satisfy $\phi(a_i) = A_i$ and $\psi(e_i) = R_i$. If two nodes u, v are endpoints of any instance of $\mathcal{P}$, we say nodes u and v are $\mathcal{P}$-neighbours, and we call the type of u (and v) the *target type*. Two nodes are called $\mathcal{P}$-connected if there is a chain of target type nodes between them so that adjacent nodes are $\mathcal{P}$-neighbours. Then, we present the widely used HIN community model:

Definition 2.3 ($(k, \mathcal{P})$-core)**.** Given an HIN H, an integer k, and a symmetric meta-path $\mathcal{P}$, the $(k, \mathcal{P})$-core is a maximal set S of target type nodes that are $\mathcal{P}$-connected with each other, such that each node in S has at least k $\mathcal{P}$-neighbors also in S.

2.2 Problem Definition

We now extend the concept of $(k, \mathcal{P})$-core to temporal settings.

Definition 2.4 (Temporal HIN $\mathcal{H}$)**.** A temporal HIN (THIN) extends a HIN by associating each edge $e \in E$ with a timestamp $t \in \mathcal{T}$, where $\mathcal{T}$ is the set of all possible timestamps. A THIN is denoted by $\mathcal{H}$.

Given $\mathcal{H}$ and a time interval $T = [t_s, t_e]$, $\mathcal{H}_T$ denotes the subgraph induced by the edges whose timestamps lie within T, along with the corresponding incident nodes. $\mathcal{H}_T$ is considered a static HIN obtained by discarding temporal information on edges.

Given $\mathcal{H}$ and a meta-path $\mathcal{P}$, each instance p of $\mathcal{P}$ is associated with a set of timestamped edges, forming a time interval $T_p = [t_{\min}(p), t_{\max}(p)]$, where the interval length is defined as $|T_p| = t_{\max}(p) - t_{\min}(p)$. In order to explore the connections over target vertices that are both structurally and temporally close, we introduce the definition of $\mathcal{P}_\delta$-instance and $\mathcal{P}_\delta$-connected as below:

Definition 2.5 ($\mathcal{P}_\delta$-instance)**.** Given a THIN $\mathcal{H}$ and a meta-path $\mathcal{P}$, an instance p of $\mathcal{P}$ is called a $\mathcal{P}_\delta$-instance, if $|T_p| \le \delta$. Any pair of u, v ($u \ne v$) of target type connected by at least one $\mathcal{P}_\delta$-instance is called $\mathcal{P}_\delta$-neighbors.

Definition 2.6 ($\mathcal{P}_\delta$-connected)**.** Given a THIN $\mathcal{H}$, a meta-path $\mathcal{P}$, and a time constraint δ, two nodes u and v of the target type are said to be $\mathcal{P}_\delta$-connected if there exists a sequence of nodes $(n_1, n_2, ..., n_k)$ of the target type, where $n_1 = u$ and $n_k = v$, and a corresponding sequence of $\mathcal{P}_\delta$-instances $(p_{1,2}, p_{2,3}, ..., p_{k-1,k})$ such that $p_{i,i+1}$ $(1 \le i < k)$ connects (n_i, n_{i+1}).

Intuitively, $\mathcal{P}_\delta$-connection describes that u and v are connected through a series of intermediate nodes via meta-path instances, where each step maintains a time span constraint δ. We denote by $G_{\mathcal{P}_\delta}$ the derived homogeneous graph where vertices are nodes of the target type from $\mathcal{H}$, and an edge exists between two vertices if and only if they are $\mathcal{P}_\delta$-neighbors in $\mathcal{H}$. We use $G_{\mathcal{P}}$ when δ is unspecified. For example, Figure 1(c) shows the derived $G_{\mathcal{P}}$ based on the THIN from Figure 1(a). In this example, a_1 and a_6 are $(4, \mathcal{P})$-connected via a_3, where a_1 and a_3 are $(0, \mathcal{P})$-neighbors, and a_3 and a_6 are $(4, \mathcal{P})$-neighbors.

Definition 2.7 ($(k, T_q, \mathcal{P}_\delta)$-core)**.** Given a THIN $\mathcal{H}$, an integer k, a meta-path $\mathcal{P}$, a time span δ, and a time interval $T_q = [q_s, q_e]$, a $(k, T_q, \mathcal{P}_\delta)$-core is a *maximal* set S of nodes with the target type, satisfying:

(1) $\forall v \in S$, v has at least k $\mathcal{P}_\delta$-neighbors in S.
(2) $\forall$ $\mathcal{P}_\delta$-neighbors $(u, v) \in S$, $\exists p$ connecting (u, v), s.t. $T_p \subseteq T_q$;
(3) $\forall u, v \in S$, u and v are $\mathcal{P}_\delta$-connected.

This definition captures the dual objectives of temporal closeness and structural connectivity, ensuring that nodes within a $(k, T_q, \mathcal{P}_\delta)$-core are not only densely connected but also interact within a constrained time span. Now we formally define the Temporal HIN Community Search (THCS) problem as follows.

Problem 1 (**THIN Community Search**). *Given a THIN $\mathcal{H}$, an integer k, a symmetric meta-path $\mathcal{P}$, a time span constraint δ, and a query time interval T_q, the goal is to identify all $(k, T_q, \mathcal{P}_\delta)$-cores.*

Building on Figure 2, our $(k, T_q, \mathcal{P}_\delta)$-core model enables systematic community discovery. For instance, with the derived author connection graph in Figure 1(c), different parameter combinations yield different communities as demonstrated in Figure 2. The flexibility in adjusting δ, T_q, and k allows analysts to explore various community patterns based on their specific temporal and structural requirements.

3 ONLINE QUERY

3.1 Center-based Sliding Window

A naive THCS solution enumerates all meta-path instances, verifying each against the δ constraint. This exhaustive check of instances,

which can have varying time spans, leads to exponential complexity. To overcome this, we handle the query interval $T_q = [q_s, q_e]$ based on its length relative to δ. If $|T_q| \leq \delta$, all meta-path instances in $\mathcal{H}_{T_q}$ are δ-valid, simplifying to standard $(k, \mathcal{P})$-core search. Otherwise ($|T_q| > \delta$), we use a sliding window approach, processing subgraphs $\mathcal{H}_\Delta$ induced by windows $\Delta = [t, t + \delta]$ that slide from q_s to $q_e - \delta + 1$. In each $\mathcal{H}_\Delta$, instances inherently satisfy δ. The challenge is then efficiently finding $\mathcal{P}_\delta$-neighbors per window, as naive searches from all target nodes cause redundant computations over shared paths.

In HINs, symmetric meta-paths are characterized by a structure that is symmetric around a central node type. This structural symmetry allows us to reformulate $\mathcal{P}_\delta$-neighbor search as a problem of connectivity via center nodes.

Observation 1 (Center Node Connectivity)**.** Two target nodes are $\mathcal{P}_\delta$-neighbors in a symmetric meta-path if and only if they share at least one common center node.

For example, in *APVPA*, the venue type V is the center, with symmetric author-paper patterns on both sides. Instead of searching from each target node, we group target nodes by shared center nodes. This approach halves the search depth and avoids redundant traversal, as any $\mathcal{P}_\delta$-neighbor relation must pass through a center node. Once centers are identified, forming neighbor pairs reduces to combining target nodes connected to the same center.

A symmetric meta-path $\mathcal{P}$ of length l, with target type A_1 and center type A_c, comprises two symmetric half meta-paths of length $l/2$. The center-based expansion strategy (Algorithm 1) searches along the half meta-path from A_c outwards to A_1. For each window Δ, Algorithm 1 identifies all center nodes, expands outwards to find reachable target nodes, and stores these connections in *cnt*. The *DeltaPNbrs* procedure (Algorithm 2) then constructs $G_{\mathcal{P}_\delta}$ by adding edges between target nodes sharing common centers. After all windows, k-core decomposition yields the final $(k, T_q, \mathcal{P}_\delta)$-core.

The time complexity of Algorithm 1 is bounded by $O(|T_q| \cdot |V| \cdot |E|)$, where $|V|$ and $|E|$ are the total number of nodes and edges in the temporal HIN $\mathcal{H}$, respectively. For each of up to $|T_q|$ windows, the algorithm performs an expansion from each center node, which takes up to $O(|E|)$ time. Since the number of center nodes n_c is bounded by $|V|$, the cost per window is up to $O(|V| \cdot |E|)$. The costs of DeltaPNbrs and k-core decomposition are dominated by this expansion complexity.

3.2 Incremental Center Expansion Algorithm

Instead of recomputing full connectivity from scratch per window, we perform incremental updates as the window slides. Specifically, as edges expire and new ones appear, we adjust only the local A_1-A_c connectivity directly affected by each insertion or deletion, thereby avoiding redundant computations across overlapping windows. Consider the half meta-path $\mathcal{P}_{half} = (A_1, ..., A_c)$. When an edge (u, v) is inserted or removed from $\mathcal{H}$ (assuming u is closer to A_1 and v to A_c), the key question is: *how does this local modification affect connectivity between A_1 and A_c?* We have the following observations:

Let A_1^u denote the set of A_1-type nodes reachable from u, and A_c^v denote the set of A_c-type nodes reachable from v. When inserting an edge (u, v), for any pair $(a_1, a_c) \in A_1^u \times A_c^v$, we can establish

Algorithm 1: Center-based Sliding Window (CSW)

Input: $\mathcal{H}$, k, $T_q = [q_s, q_e]$, $\mathcal{P}$, δ
Output: The $(k, T_q, \mathcal{P}_\delta)$-core $\mathcal{R}$

$G_{\mathcal{P}_\delta} \leftarrow \emptyset$;
for $t \in [q_s, q_e - \delta + 1]$ **do**
  $\Delta \leftarrow [t, t + \delta]$; Induce $\mathcal{H}_\Delta$; $cnt \leftarrow \emptyset$;
  for *each* *center node* $c \in \mathcal{H}_\Delta$ **do**
    $S \leftarrow \{c\}$;
    for $j \leftarrow l/2$ ***down to*** 1 **do**
      $L \leftarrow \emptyset$;
      for *each* *node* $u \in S$ **do**
        for *each* *neighbor* $v \in N_{\mathcal{H}_\Delta}(u)$ **do**
          if (v, u) *matches the j-th edge type of* $\mathcal{P}$ **then** $L.add(v)$;
      $S \leftarrow L$;
    $cnt[c] \leftarrow S$;
  DeltaPNbrs($cnt, G_{\mathcal{P}_\delta}$) ;
$\mathcal{R} \leftarrow$ k-core decomposition of $G_{\mathcal{P}_\delta}$;
return $\mathcal{R}$;

Algorithm 2: DeltaPNbrs($cnt, G_{\mathcal{P}_\delta}$)

for *each* *center node* $c \in cnt$ **do**
  for *each* *pair* (a_1, a_1') *where* $a_1, a_1' \in cnt[c]$ $(a_1 \neq a_1')$ **do**
    if $(a_1, a_1') \notin G_{\mathcal{P}_\delta}$ **then**
      Add edge (a_1, a_1') to graph $G_{\mathcal{P}_\delta}$;

a connection. If these nodes were not previously connected, this creates a new path; if they were already connected, adding another path is idempotent and does not affect existing connectivity. However, when removing an edge (u, v), we cannot simply remove all connections in $A_1^u \times A_c^v$, as alternative paths may still connect them. Each potentially affected pair (a_1, a_c) requires verification to determine if other valid paths remain. This asymmetry between insertion and deletion operations necessitates careful tracking of path counts rather than just binary connectivity.

Incremental Connectivity Tracker (ICT). To maintain the $\mathcal{P}_\delta$-connectivity efficiently, we introduce the *Incremental Connectivity Tracker* (ICT), which tracks path propagation between A_1 and A_c. Specifically, for each node a_i ($1 \leq i \leq l/2$) along the half meta-path $\mathcal{P}_{half}$, ICT maintains:

$$F_C : A_i \times A_c \to \mathbb{Z}^+,$$

where $F_C(a_i, a_c)$ records the number of valid paths from a_i to a_c. The path count $F_C(a_i, a_c)$ at an intermediate node a_i (where A_i is not the center type A_c) accumulates from its A_{i+1}-type neighbors $\mathcal{N}(a_i)$ as follows:

$$F_C(a_i, a_c) = \sum_{a_{i+1} \in \mathcal{N}(a_i)} F_C(a_{i+1}, a_c). \tag{1}$$

This is because paths from a_i to a_c must traverse an $a_{i+1} \in \mathcal{N}(a_i)$, each contributing its $F_C(a_{i+1}, a_c)$ independently. This property

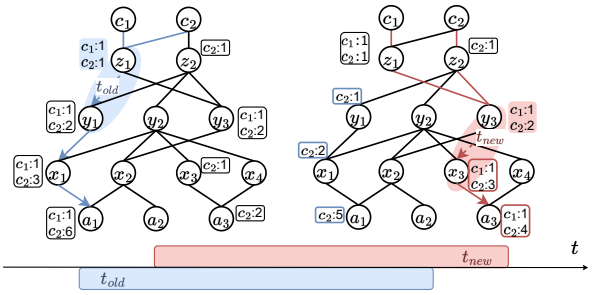


Figure 3: An example of ICT

holds for any $1 \le i < l/2$, forming a recursive propagation mechanism from A_c back to A_1. It ensures that (a_1, a_c) connectivity is maintained as long as at least one intermediate path exists. Removing an edge (a_i, a_{i+1}) does not necessarily disconnect the pair if alternative paths contribute to $F_C(a_1, a_c)$. Therefore, we track $F_C(a_i, a_c)$ and propagate updates along $\mathcal{P}_{half}$.

Incremental Update Mechanism. For an edge (a_i, a_{i+1}) $(1 \le i \le l/2 - 1)$ along $\mathcal{P}_{half}$, ICT updates connectivity by propagating changes from a_i toward A_1 via depth-first traversal along $\mathcal{P}^{-1}_{half}$. For each intermediate node a_j reachable from a_i where $1 \le j \le i$, ICT cumulatively updates its connectivity: when inserting (deleting) edge (a_i, a_{i+1}), each visited a_j increases (decreases) its path count to centers by $F_C(a_{i+1}, a_c)$ for all $a_c \in A_c^{a_{i+1}}$.

As updates propagate backward toward A_1, connectivity between A_1 and A_c is determined by examining $F_C(a_1, a_c)$: transitions from 0 to positive indicate new connections; transitions to 0 indicate removals. For example in Figure 3, removing edge (y_1, z_1) causes $F_C(a_1, c_1)$ to become 0, while inserting (x_3, y_3) establishes $F_C(a_3, c_1) = 1$.

ICT only stores active connections, discarding zero-valued entries. The worst-case storage complexity is $\sum_{i=1}^{l/2} n_i \cdot n_c$, but in practice, sparse connectivity makes ICT space-efficient.

We present our advanced online algorithm, Incremental Center Expansion (ICE), in Algorithm 3, which integrates center-based expansion with incremental connectivity tracking to efficiently maintain $\mathcal{P}_\delta$-neighbors over sliding windows. The algorithm begins with an **initialization phase**, constructing complete connectivity information (F_C and cnt) for the first window using full center-based expansion (lines 1-7). In the **incremental update phase**, as the window slides forward, ICE identifies expiring and entering edges (lines 8-15). Instead of recomputing from scratch, it propagates connectivity changes for each edge insertion and deletion using the ICT mechanism described above. After each update, 'DeltaPNbrs' (Algorithm 2) updates $G_{\mathcal{P}_\delta}$. Finally, the $(k, T_q, \mathcal{P}_\delta)$-core is extracted using standard k-core decomposition.

The time complexity of the ICE algorithm is bounded by $O(|E|^2)$. The initialization phase performs expansion from n_c center nodes, taking up to $O(n_c \cdot |E|)$ time. In the incremental phase, each edge is processed at most twice (insertion and deletion). The propagation for each edge update involves a search from the edge's endpoint towards the target nodes and ICT updates, which takes up to $O(|E|)$ time. Since n_c is bounded by $|V|$ and the number of processed edges is bounded by $|E|$, the total complexity is bounded by $O(|E|^2)$ (usually $|V|$ is smaller than $|E|$).

Algorithm 3: Incremental Center Expansion (ICE)

Input: $\mathcal{H}$, k, $T_q = [q_s, q_e]$, $\mathcal{P}$, δ
Output: The $(k, T_q, \mathcal{P}_\delta)$-core $\mathcal{R}$

1 $G_{\mathcal{P}_\delta} \leftarrow \emptyset$; $F_C \leftarrow \emptyset$; $cnt \leftarrow \emptyset$;
2 $\Delta \leftarrow [q_s, q_s + \delta]$; induce $\mathcal{H}_\Delta$ from $\mathcal{H}$;
// Initial expansion on the first window
3 **foreach** *center node* c *in* $\mathcal{H}_\Delta$ **do**
// half-path expansion from center node to A_1
4 Perform BFS along $\mathcal{P}_{half}$ from c towards A_1-type nodes;
5 Update $F_C(\cdot, c)$ by Eq. (1);
6 $cnt[c] \leftarrow$ all A_1-type nodes u with $F_C(u, c) > 0$;
7 DeltaPNbrs($cnt, G_{\mathcal{P}_\delta}$);
8 **for** $t \leftarrow q_s$ **to** $(q_e - \delta - 1)$ **do**
9 $E_{\text{del}} \leftarrow \{ e \in \mathcal{E} \mid \text{timestamp}(e) = t \}$;
10 **foreach** *edge* $e = (u, v)$ *in* E_{del} **do**
11 Propagate path count changes for deletion via ICT;
12 $E_{\text{ins}} \leftarrow \{ e \in \mathcal{E} \mid \text{timestamp}(e) = t + \delta + 1 \}$;
13 **foreach** *edge* $e = (u, v)$ *in* E_{ins} **do**
14 Propagate path count changes for insertion via ICT;
15 DeltaPNbrs($cnt, G_{\mathcal{P}_\delta}$);
16 $\mathcal{R} \leftarrow$ k-core decomposition of $G_{\mathcal{P}_\delta}$;
17 **return** $\mathcal{R}$;

4 THIN CORE INDEX

While online algorithms offer speed, frequent user searches can lead to significant computational overhead, especially when users repeatedly query the same time periods. To enhance query efficiency, with a given meta-path $\mathcal{P}$, we construct the THIN Core Interval-Index (TCI-Index), designed for storing and efficiently retrieving $(k, T_q, \mathcal{P}_\delta)$-core information. Our indexing approach addresses the complex interplay between three parameters: the temporal constraint δ, the query time interval $T_q = [q_s, q_e]$, and the core number k. By capturing the minimal necessary temporal information, our index efficiently integrates these parameters to support fast querying. By leveraging the hierarchical nature of k-core and containment relationships between temporal intervals, our index organizes information in a highly compact manner, enabling direct access to relevant intervals without computationally intensive calculations.

4.1 Building on Minimal Intervals

As illustrated in Figure 1(c), connections between authors through meta-paths can generate multiple time intervals on each edge, representing distinct interaction periods. However, not all intervals are necessary for determining community candidature.

Observation 2 (Interval Dominance)**.** Given two intervals $I_1 = [s_1, e_1]$ and $I_2 = [s_2, e_2]$, if $I_2 \subseteq I_1$ (i.e., $s_1 \le s_2$ and $e_1 \ge e_2$), then for any time constraint δ, if nodes are $\mathcal{P}_\delta$-neighbors in I_2, they are also $\mathcal{P}_\delta$-neighbors in I_1.

This observation allows us to focus on the smallest intervals that fully characterize temporal relationships (shown as red intervals in Figure 1(c)), leading to our first key concept:

Definition 4.1 (Minimal $\mathcal{P}$-Neighbor Interval (MNI))**.** For two nodes u and v connected via meta-path $\mathcal{P}$, an interval I is their minimal $\mathcal{P}$-neighbor interval (*mni*) if: (1) u and v are $\mathcal{P}$-neighbors throughout I, and (2) no smaller interval $I' \subset I$ satisfies condition (1).

The key contribution of the *MNI* concept is its role in bridging the fundamental gap between direct edge connectivity in traditional temporal graphs and meta-path-based connectivity in THINs. Unlike existing approaches where core times can be derived directly from edges, our method first requires identifying these essential meta-path connections as a critical preprocessing step.

Given an *mni* between nodes u and v:

- The span of *mni* determines the minimum δ required: u and v can be $\mathcal{P}_\delta$-neighbors only if $\delta \geq |mni|$
- The interval must fit within T_q: the *mni* contributes to connectivity only if $mni \subseteq T_q$.

By identifying these minimal intervals, we transform complex meta-path connectivity into a tractable interval-based record, enabling us to efficiently build higher-level core intervals rather than compute directly from the raw THIN structure. With *mni*s serving as the foundation for temporal relationships between target node pairs, we now integrate the third parameter—core number k—by examining how multiple *mni*s collectively contribute to forming $(k, T_q, \mathcal{P}_\delta)$-cores. This leads us to define intervals where nodes maintain reachable core membership:

Definition 4.2 (Minimal $(k, \mathcal{P})$-Core Interval (MCI))**.** For a node v, an interval I is its minimal $(k, \mathcal{P})$-core interval of k (mci_k) if: (1) v is in a $(k, \mathcal{P})$-core in I, and (2) no smaller interval $I' \subset I$ satisfies condition (1). We denote the set of all *mci*s for v of k as $C_k(v)$.

Crucially, the length of an mci_k must not exceed δ to preserve a $(k, T_q, \mathcal{P}_\delta)$-core property. While *mni*s establish valid neighborhood relations within I, the span of mci_k determines whether the node maintains core membership under the temporal constraint δ. Hence, the *mci* concept naturally unifies k, δ, and T_q in a single framework, which we leverage to construct our index structure.

Lemma 4.1 (Core Membership via MCI). *A node v belongs to a $(k, T_q, \mathcal{P}_\delta)$-core if and only if there exists an $mci_k \in C_k(v)$ such that: (i) $mci_k \subseteq T_q$ and (ii) the span of mci_k is at most δ.*

The strategy to efficiently identify these MCIs and ensure they satisfy the minimality condition is detailed in Section 5. This is achieved implicitly during our incremental index construction, avoiding explicit sub-interval comparisons.

4.2 A Compact Index Structure

Building on the node information of *mci*s in C, The THIN Core Interval-Index (TCI-Index) $\mathbb{T}$ is an inverted index layered by core number k. Each layer corresponds to a distinct k value and contains multiple Core Interval Blocks (CI-Blocks) CI keyed by $mci = [t_s, t_e]$ sorted by the start time t_s, then by the end time t_e. Each Core Interval Block owns:

- $CI.T = [t_s, t_e]$: The minimal core interval (*mci*).
- $CI.V_I$: The set of nodes S for which this interval is one of their *mci*s, i.e., $\forall v \in S, mci \in C_k(v)$

However, storing each node for all its *mci*s can lead to large index sizes in practice. Two properties of *mci*s make this challenging:

Property 4.2 (Multiplicity). *Multiple mcis may exist for the same node v and core number k, representing different temporal intervals of core membership. These intervals may overlap but cannot strictly contain one another.*

Property 4.3 (Completeness). *mcis fully capture all $(k, \mathcal{P})$-core intervals: a node v is in a $(k, \mathcal{P})$-core during interval I if and only if there exists at least one $mci_k \in C_k(v)$ such that $mci_k \subseteq I$.*

The multiplicity property implies that we need to handle multiple *mci*s for each node and core number, while the completeness property requires capturing all such intervals. For example, in Figure 4(a), node a_3 appears in three CI-Blocks with $k = 2$, one CI-Block with $k = 3$, and one CI-Block with $k = 4$. Storing the same nodes in each CI-Block is both redundant and inefficient as the network grows. To reduce storage overhead, we leverage the nested nature of time-expanded cores:

Lemma 4.4 (Time Nesting). *If $I \subseteq I'$, then the core number of I' is no less than that of I for the same node v.*

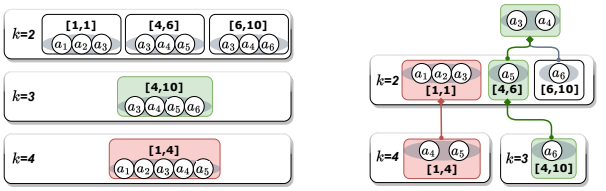
As interval I' grows, it can encompass more minimal neighbor intervals (*mni*s), establishing additional valid meta-path connections that potentially raise v's core membership. Hence, a node v that achieves a higher core number k' in a larger interval I' typically maintains a lower core number k in some smaller interval $I \subset I'$. For example, in Figure 4(a), nodes a_1, a_2, a_3 are in a 2-core at $[1, 1]$ and a 4-core at $[1, 4]$.

The multiplicity property also implies that the same connected component may appear in multiple CI-Blocks with non-overlapping *mci*s at the same core number k. Rather than repeatedly storing these node sets, we maintain a single copy and have each relevant CI-Block point to it. We propose two types of compression in our index:

(1) **Vertical:** In $\mathbb{T}$, we link CI-Blocks with higher k values to the lower k CI-Blocks that store a subgraph of their components, retaining only incremental nodes.
(2) **Horizontal:** At $\mathbb{T}[k_{min}]$, we extract frequently appearing connected components and store them in a separate block, with those CI-Blocks pointing to this shared representation.

These compression techniques create a space-efficient index structure that preserves both connectivity information and temporal evolution of core numbers. Without compression, the TCI-Index would require $O(|V| \cdot |C| \cdot k_{max})$ space, where $|V|$ is the number of nodes, $|C|$ is the average number of *mci*s per node, and k_{max} is the maximum core number. With vertical compression, higher-k CI-Blocks store only incremental nodes and pointers to lower-k blocks, exploiting the hierarchical nature where k-core components typically grow or merge as windows expand. This reduces index size by a factor proportional to the overlap ratio between k-core components, which is typically significant in real-world networks.

Figure 4 illustrates our TCI-Index, where CI-Blocks are organized by k, then t_s, t_e. For example, retrieving the 4-core for $[1, 4]$



(a) TCI-Index (b) Compressed TCI-Index

Figure 4: TCI-Index structure with vertical compression.

(red) gets $\{a_4, a_5\}$ from the $k = 4$, $[1, 4]$ CI-Block, and $\{a_1, a_2, a_3\}$ via a pointer from the $k = 2$, $[1, 1]$ CI-Block. This shows vertical compression ensuring efficient storage and complete retrieval via shared components.

4.3 Query Algorithm

Given a meta-path $\mathcal{P}$ and query parameters (k, T_q, δ), our task is to identify all nodes belonging to $(k, T_q, \mathcal{P}_\delta)$-cores within the query window. The TCI-Index's hierarchical structure enables efficient query processing through three key observations:

Observation 3 (Core Hierarchy)**.** If nodes form a connected k'-core ($k' > k$) in interval I, they must also form a connected k-core in I. By following our index's pointer structure from higher to lower core layers, we avoid redundant processing of the same connected components.

Observation 4 (Temporal Filtration)**.** For any CI-Block CI to contribute to a $(k, T_q, \mathcal{P}_\delta)$-core, it must satisfy both containment ($CI.T \subseteq T_q$) and span ($|CI.T| \leq \delta$) constraints simultaneously.

These observations lead to an efficient two-phase query strategy:

(1) **Core-Layer Processing:** Starting from the highest core number k_{max}, we examine each core layer $k' \geq k$ in descending order. For each layer, we identify CI-Blocks whose intervals satisfy both temporal constraints.

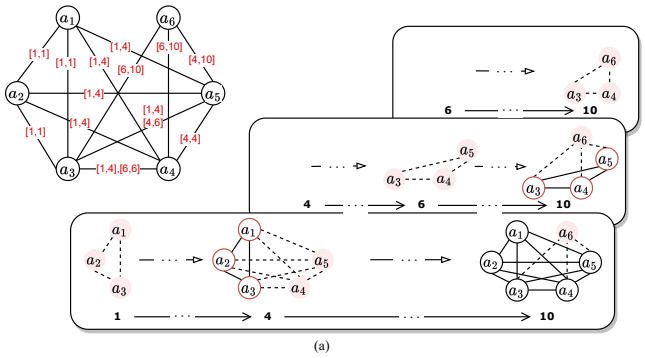
(2) **Component Assembly:** For each qualifying CI-Block, we follow the pointer structure to retrieve its complete node set by collecting linked subgraphs from lower core layers, effectively reconstructing the full temporal community without redundant storage access.

Algorithm 4: TCI-Query

Input: $\mathbb{T}_\mathcal{P}$, k, T_q, $\mathcal{P}$, δ
Output: Set of nodes in $(k, T_q, \mathcal{P}_\delta)$-cores

```
result ← ∅;
for each core number k' from k_max down to k do
    for each CI ∈ T_P[k'] do
        if CI.T ⊆ T_q and |CI.T| ≤ δ then
            result ← result ∪ all nodes in CI and its
              referenced CI-Blocks;
return result;
```

For each qualifying CI-Block, all nodes are assembled by recursively following compression pointers to lower-k CI-Blocks, ensuring the complete result set is retrieved without redundancy. The recursive collection of nodes via compression pointers ensures that all relevant nodes are included, even when components are shared across multiple CI-Blocks due to vertical compression.



(a)

Figure 5: Index Construction Process

Algorithm 4 outlines the query process. We iterate through each core layer k', identifying all qualifying CI-Blocks within the query window. For each CI-Block, we collect the connected components in the current and lower core layers, assembling the complete $(k, T_q, \mathcal{P}_\delta)$-core structure. The time complexity of the query algorithm is at most $O(k_{max} \cdot (T_q - \delta + 1))$, where k_{max} is the maximum core number in the network, and $(T_q - \delta + 1)$ is the number of possible query intervals.

5 INDEX CONSTRUCTION

To construct the TCI-Index, we must first identify all *mnis* between node pairs, forming the basis for computing $\mathcal{P}_\delta$-neighborhoods and core numbers. Like *mcis*, *mnis* exhibit both Multiplicity and Completeness, requiring systematic exploration of all possible temporal intervals, which makes TCI-Index construction considerably more complex than PHC-Index.

5.1 Incremental Window Expansion

While our online algorithm employs a sliding window approach handling both edge additions and deletions, for index construction we adopt a more efficient incremental window expansion strategy. This choice is motivated by a key observation: when expanding temporal windows, adding new $\mathcal{P}$-neighbors is an *unharmful* operation requiring only simple meta-path verification. In contrast, when shrinking windows, determining if a $\mathcal{P}$-neighbor relationship breaks requires maintaining path counts, which incurs significant storage overhead for large graphs. Although prior work has shown efficient core maintenance for edge deletions in simple graphs [16, 24, 39], the meta-path search in THINs is computationally more expensive than core decomposition [6, 7]. Therefore, window expansion avoids the complexity of tracking meta-path counts during edge deletions, making it more suitable for large-scale index construction.

Our approach leverages the monotonic nature of core numbers with respect to temporal expansion:

Lemma 5.1 (Core Monotonicity). *For a fixed starting time t_s, as we expand the window from $[t_s, t_e]$ to $[t_s, t_e + 1]$, core number of a node can only increase or remain unchanged as new edges enter the window.*

For example, in Figure 5, when expanding the window from $[1, 1]$ to $[1, 4]$, the core number of nodes $\{a_1, a_2, a_3\}$ and $\{a_4, a_5\}$ increases from 2 to 4 and 0 to 4, respectively, as new edges are added. When the window expands to $[1, 10]$, the core number of node a_6 rises from 0 to 3, while the core number of $\{a_1, ..., a_5\}$ remains unchanged at 4.

For each starting timestamp t_s, we incrementally expand t_e while tracking new *mnis* and resulting core number changes. When updating core number at each step, we identify connected components within each core layer. When the core number of a node v first reaches or changes to k' precisely at time t_e, we record the interval and component identifier:

$$C_{k'}(v)[t_e] = (t_s, comp\_id) \tag{2}$$

The recorded interval $[t_s, t_e]$ is a Minimal $(k', \mathcal{P})$-Core Interval (MCI) per Definition 4.2. Minimality is ensured through our construction strategy: when processing subsequent starting times, if a later $t_s' > t_s$ also produces core number k' at the same t_e, we overwrite t_s with t_s'. Since $[t_s', t_e] \subset [t_s, t_e]$, this automatic replacement guarantees we maintain only minimal intervals ending at t_e. For example, in Figure 5, if processing with $t_s = 1$ yields an MCI $C_{k'}(a_6)[10] = (1, \mathrm{id}_1)$, and later processing with $t_s = 4$ also results in a_6 being a k'-core in $[4, 10]$, the entry updates to $C_{k'}(a_6)[10] = (4, \mathrm{id}_2)$, reflecting the minimal interval. This direct relationship between MNI discovery and MCI updates makes our incremental approach both efficient and theoretically sound.

Bidirectional Search and Matrix Updates. When a new temporal edge (u, v, t_e) is inserted into $\mathcal{H}$ during window expansion, it may establish new $\mathcal{P}_\delta$-connections between target nodes. Unlike our online algorithm that maintains detailed path counters for both additions and deletions, for index construction we only need to identify new connections. We leverage the symmetric structure of meta-paths through a bidirectional search strategy:

$$\text{Target nodes} \xleftarrow{\text{Backward search}} (u, v) \xrightarrow{\text{Forward search}} \text{Center nodes}$$

This approach splits the half meta-path search: backward from u to find target-type nodes (A_1), and forward from v to find center-type nodes (A_c). When new target-center pairs (a_1, a_c) are found, we can immediately determine that all target nodes previously connected to c are now neighbors of a, and vice versa. Forwards and backwards searches are performed independently, and the resulting target-center pairs are combined to form the new connections.

To efficiently manage these connections, we maintain a sparse Boolean matrix $M_{tc} \in \{0, 1\}^{|V_t| \times |V_c|}$ where entry $M_{tc}[i, j] = 1$ indicates a valid meta-path between target node i and center node j. The target node adjacency matrix is then derived as:

$$A_{tt} = M_{tc} \cdot M_{tc}^T \tag{3}$$

For computational efficiency when new connections ΔM_{tc} are discovered, we compute only the incremental updates:

$$\Delta A_{tt} = (\Delta M_{tc} \cdot M_{tc}^T) + (M_{tc} \cdot \Delta M_{tc}^T) \tag{4}$$

Algorithm 5: TCI-Index Construction

Input: $\mathcal{H}$,$\mathcal{P}$, time range $[0, T]$
Output: TCI-Index $\mathbb{T}$

$C, \mathbb{T} \leftarrow \emptyset, \emptyset$;
for *each starting time t_s from 0 to T* **do**
  $M_{tc} \leftarrow$ initialize empty target-center connection matrix;
  $G_{\mathcal{P}} \leftarrow \emptyset$;
  for *each end time t_e from t_s to T* **do**
    `// Process new edges at time` t_e
    for *each edge (u, v, t_e) in $\mathcal{H}$* **do**
      $\Delta M_{tc} \leftarrow$ BidirectionalSearch((u, v, t_e), $\mathcal{P}$);
    Update M_{tc} and compute ΔA_{tt};
    $G_{\mathcal{P}}$ add edges $\{(i, j) \mid \Delta A_{tt}[i, j] \neq 0\}$;
    Update core number on $G_{\mathcal{P}}$;
    for *each node v whose core number changed to k'* **do**
      $comp\_id \leftarrow$ component ID of v in k'-core;
      $C_{k'}(v)[t_e] \leftarrow (t_s, comp\_id)$;
`// Build hierarchical index structure`
for *each core number k from k_{max} down to 1* **do**
  for *each $CI \in \mathbb{T}[k]$* **do**
    $CI.T \leftarrow mci$ for distinct *mcis*;
    $CI.V_I \leftarrow$ group v by comp_id;
return TCI-Index $\mathbb{T}$;

Using compressed sparse row storage and limiting computation to affected submatrices, this approach enables efficient parallel processing on sparse matrices. With the updated adjacency information, we perform incremental core number updates as the temporal window expands, recording *mcis* whenever the core number of a node increases.

5.2 MCI Organization and Index Finalization

Algorithm 5 summarizes our complete index construction process. For each starting time t_s, we incrementally extend the window to all possible end times t_e, updating target-center connections and core numbers. When the core number of a node v changes, we record current t_s, t_e as $mci_k(v)$ along with its component ID. After processing all temporal windows, we organize these *mcis* into our hierarchical structure by grouping nodes with the same core number k and component $comp\_id$ for each minimal interval $[t_s, t_e]$, creating entries of the form $\text{Index}[k][t_s, t_e] = \{v \mid C_k(v)[t_e] = (t_s, comp\_id)\}$. These CI-Blocks are sorted primarily by start time t_s and secondarily by end time t_e to support efficient temporal queries. We then apply the vertical and horizontal compression techniques described in Section 4 to reduce storage requirements while preserving the connectivity information across core layers.

The time complexity of our index construction is dominated by three operations: bidirectional meta-path search ($O(T \cdot |E| \cdot d^{\lceil l/2 \rceil})$), incremental core maintenance ($O(T^2 \cdot |\Delta V| \cdot d)$ where $|\Delta V|$ is the average number of affected nodes per update), and index organization ($O(|V| \cdot T \cdot k_{max})$). Overall, the complexity is $O(T \cdot |E| \cdot d^{\lceil l/2 \rceil} + T^2 \cdot |\Delta V| \cdot d + |V| \cdot T \cdot k_{max})$, typically dominated

by meta-path search due to the exponential factor $d^{\lceil l/2 \rceil}$ and the sparsity of real-world networks where $|\Delta V| \ll |V|$.

5.3 Incremental Index Updates

Our index construction strategy naturally supports incremental updates when new edges arrive, which follows the incremental window expansion paradigm. For sparse insertions involving a limited number of edges, systematic reconstruction becomes computationally inefficient, as the impact of new edges is typically localized to specific temporal windows and graph regions.

The update process involves identifying new $\mathcal{P}$-neighbor relationships triggered by the insertion and then performing localized core updates on the affected temporal intervals. Specifically, when a new edge (u, v, t_{new}) is inserted into $\mathcal{H}$, we first identify any new target-center connections. These connections may establish new $\mathcal{P}$-neighbor relationships and their corresponding valid temporal intervals. For each affected interval, we update its derived graph by inserting the newly formed $\mathcal{P}$-neighbor edges and then perform a localized core update. If a node's core number increases to k', this suggests a new minimal core interval (MCI) has formed at level k'. We then check the TCI-Index for an existing MCI at level k' that contains this new interval. If such a containing interval is found, we add the affected node to its corresponding CI-Block. Otherwise, we create a new MCI entry for the new interval and its associated node set. This ensures that updates are confined to the affected graph regions and temporal windows as much as we can.

6 EXPERIMENTS

We conduct extensive experiments to evaluate the performance of our THIN community search methods. Our evaluation encompasses three online algorithms and an index-based query method. The online algorithms include: (1) a baseline (Base) approach that applies standard meta-path traversal within each temporal slice of a sliding window; (2) our Center-based Sliding Window search (CSW); and (3) our Incremental Center-based Expansion method (ICE). We also assess our index-based query TCI-Q (Algorithm 4). We adopt Base as a fundamental reference and ablation point to quantify the gains of center-based expansion (CSW) and incremental maintenance (ICE). Base adapts the FastBCore algorithm [6] within a sliding-window framework for temporal HIN community search. FastBCore efficiently computes $(k, \mathcal{P})$-cores by performing depth-first search along the entire $\mathcal{P}$ with labelling to avoid redundant traversals. The complexity of Base is $O(|T_q| \cdot n_t \cdot |E|)$, where n_t is the number of target nodes. We also evaluate our index construction approach and assess the effectiveness of our model against traditional $(k, \mathcal{P})$-core. All experiments run on a Windows machine with an Intel E5-2680, 2.70 GHz and 192GB RAM.

6.1 Setup

Datasets. We employ four real-world networks from diverse domains: *IMDB* (movie database with user ratings and reviews), *Trip* (trajectory network of for-hire vehicles in New York City), *DBLP* (bibliographic network with authors, papers, venues, and keywords), and *Yelp* (business reviews with user interactions). Table 2 summarizes these datasets, including the number of nodes ($|\mathcal{V}|$), edges ($|\mathcal{E}|$), node types ($|\mathcal{A}|$), relation types ($|\mathcal{R}|$), distinct timestamps ($|T|$), and meta-paths ($|\mathcal{P}|$). The temporal granularity varies across datasets: IMDB and Yelp contain 277 months and 850 weeks respectively, DBLP uses yearly granularity (65 years), and Trip records 9337 hours. These granularities are chosen to reflect the natural recording frequency of events in each domain (e.g., yearly publications in DBLP, hourly vehicle movements in Trip) and to evaluate our methods across diverse temporal scales. Following prior work, we use meta-paths of length at most 4 on IMDB and DBLP, and for Trip and Yelp, we generate 20 meta-paths due to their complex schema with length at most 6 [6, 7, 12]. For instance, these include paths like Author-Paper-Topic-Paper-Author in DBLP to find authors working on similar topics, User-Review-Movie-Review-User in IMDB for users reviewing the same movies, or Taxi-PickupLocation-DropoffLocation-Taxi in Trip to connect taxis serving similar routes.

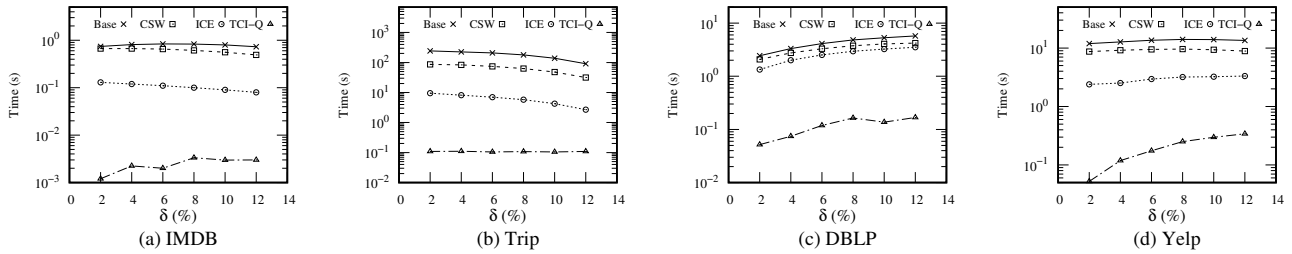
Table 2: Dataset Statistics

Dataset	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	$\|\mathcal{A}\|$	$\|\mathcal{R}\|$	$\|\mathrm{T}\|$	$\|\mathcal{P}\|$
IMDB	17,492	234,615	5	4	277	12
DBLP	1,223,635	2,607,149	4	3	65	11
Trip	7,834	1,689,218	17	72	9337	20
Yelp	1,624,358	4,195,932	7	14	850	20

Parameters. The default setting for k is 8. The default values for δ and T_q are 5% and 15% of the entire timeline span, respectively. Query results are averaged over 20 random query intervals T_q. In our experiments varying k, we explore the range [4, 20], which is a common setting in related HIN community search and core decomposition literature [6, 12, 13, 36].

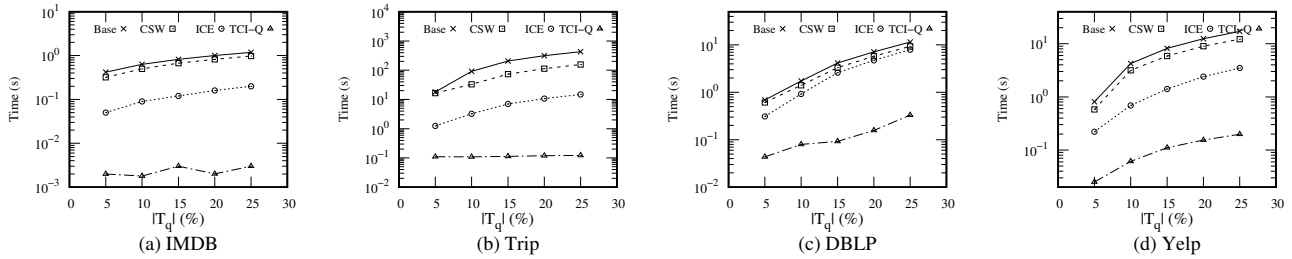
6.2 Query Performance

Efficiency with varying parameters. We evaluate the efficiency of our online methods Base, CSW, ICE and index-based query TCI-Q by measuring the average query processing time. We vary δ from 2% to 12%, query interval length T_q from 5% to 25% of the timeline span, and k from 4 to 20. Results are presented in Figures 6, 7, and 8. CSW consistently outperforms Base due to optimized meta-path search, while ICE achieves substantial further speedups, often by one to two orders of magnitude. The index-based TCI-Q demonstrates superior efficiency over all online methods. With increasing δ, processing times for DBLP and Yelp generally increase as the relaxed temporal constraint broadens the meta-path search space. For TCI-Q, larger δ means more intervals satisfy query conditions, increasing lookup time. Interestingly, for IMDB and Trip—datasets characterized by lower temporal density (i.e., fewer edges per timestamp)—online methods become faster as δ increases. This is because the benefit of reducing the total number of window shifts outweighs the moderately increased cost per window, highlighting that the number of shifts is the dominant performance factor for these temporally fine-grained datasets. As T_q expands, processing times grow proportionally with the search space; this is evident in DBLP where execution times for Base and CSW increase over 15 times as $|T_q|$ goes from 5% to 25%, partially due to early publication data sparsity. When k increases, online algorithms maintain stable performance since core decomposition complexity remains constant,



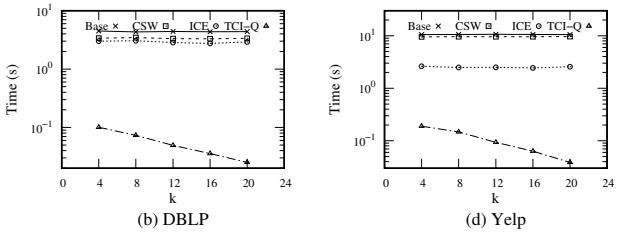
(a) IMDB (b) Trip (c) DBLP (d) Yelp

Figure 6: Effect of varying δ on query runtime



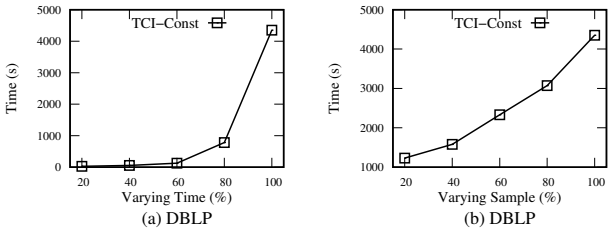
(a) IMDB (b) Trip (c) DBLP (d) Yelp

Figure 7: Effect of varying $|T_q|$ on query runtime



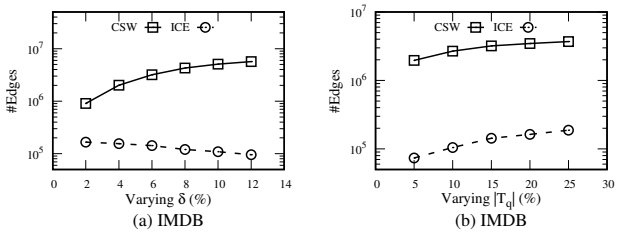
(b) DBLP (d) Yelp

Figure 8: Effect of varying k on query runtime



(a) DBLP (b) DBLP

Figure 10: Index construction time



(a) IMDB (b) IMDB

Figure 9: Number of edges visited

while index-based query time decreases as higher core constraints filter out more intervals.

Edge Traversal Analysis. Figure 9 compares the number of edges visited by CSW and ICE on the IMDB dataset as δ and T_q vary. ICE consistently visits significantly fewer edges than CSW, demonstrating the effectiveness of incremental maintenance. As δ increases from 2% to 12%, CSW's edge visits increase dramatically due to larger window sizes, while ICE's visits actually decrease as fewer windows need processing. When T_q increases from 5% to 25%, both algorithms visit more edges, but ICE maintains its significant advantage. This confirms that ICE's incremental approach effectively avoids redundant computation between overlapping windows.

6.3 Index Evaluation

Index Construction. We evaluate index construction time on the DBLP dataset using two scaling approaches, as shown in Figure 10. In Figure 10(a), we vary temporal coverage from 20% to 100% of the timeline. The construction time increases quadratically, confirming the $O(T^2)$ complexity of our window expansion approach. In Figure 10(b), we fix the full timeline but vary the data sample size from 20% to 100%, observing a more gradual increase. This difference in scaling behavior occurs because: (1) recent DBLP data is denser, requiring more computation per window; and (2) the quadratic relationship between construction time and temporal range makes the algorithm sensitive to the number of timestamps rather than the data volume.

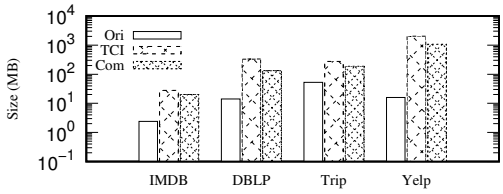


Figure 11: Index size comparison across datasets

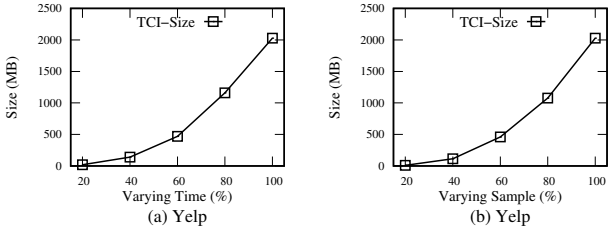


Figure 12: Effect of varying index size on query performance

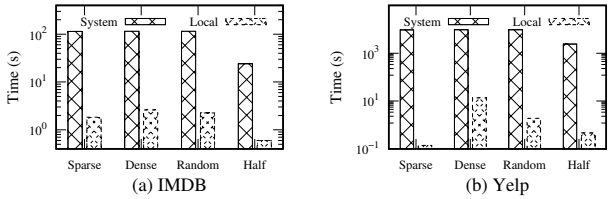


Figure 13: Index maintenance cost upon new edge insertion.

Index Size. Figure 11 compares the index sizes across all datasets, showing both original (Ori) data size and index size before (TCI) and after compression (Com). The impact of temporal granularity is evident: DBLP with its coarse yearly granularity has an index-to-data ratio of approximately 23:1, while Trip with fine-grained hourly timestamps has only about 1.5:1. Our compression techniques demonstrate significant effectiveness, reducing index size by approximately 27% to 59% across datasets. The compression is most effective on DBLP due to its higher density of overlapping k-cores across different time intervals, while achieving moderate but meaningful reductions on datasets with finer temporal granularity.

Index Updates and Scalability. We evaluate incremental index maintenance by simulating edge insertions on IMDB and Yelp datasets. We compare two update strategies: (i) *System*, following the original window-expansion procedure used during full index construction; and (ii) *Local*, which performs targeted updates on only the affected temporal windows and graph regions as described in Section 5.3. To comprehensively evaluate the update performance, we consider four edge selection strategies: Sparse (both edge endpoints in lower 50% degree range), Dense (both in upper 50%), Random (no constraints), and Half (insertions on an index built with half the timeline). Results in Figure 13 demonstrate that Local is consistently faster across all settings, while speedup gaps

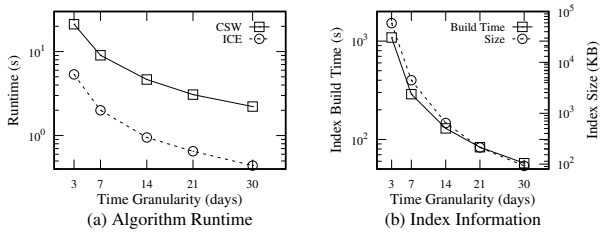


Figure 14: Effect of temporal granularity

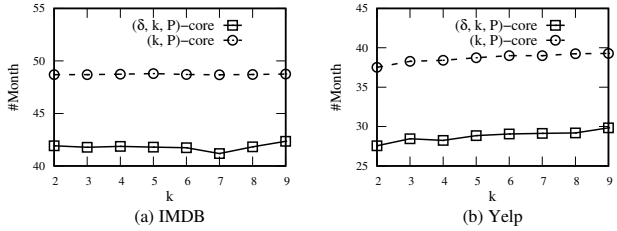


Figure 15: Average temporal distance between community members

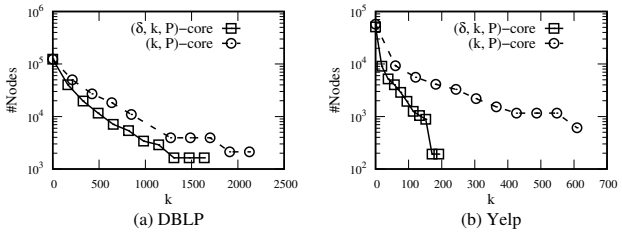
are more pronounced on sparse datasets like Yelp, where Local updates affect fewer windows. Higher-degree edge insertions incur higher costs due to more affected $\mathcal{P}$-neighbors and larger update regions. Half performs best due to shorter temporal coverage.

In terms of scalability, we evaluate index size on the Yelp dataset using two sampling approaches: temporal sampling and data sampling, with results shown in Figure 12. With 20% temporal coverage, the index is 19.21MB, expanding to 2026.82MB at full coverage. Similarly, with 20% data sampling, the index starts at 9.43MB and reaches 1077.01MB at 80% sampling.

Timestamp Granularity Analysis. Figure 14 examines varying temporal granularity on the Yelp dataset from 3-day to 30-day periods. Figure 14(a) demonstrates that finer granularities increase runtimes for online algorithms due to more discrete timestamps, with ICE consistently outperforming CSW. Figure 14(b) shows finer granularities significantly increase TCI-Index construction time and size due to more temporal intervals and k-core states. These results confirm that performance and storage requirements are directly linked to temporal resolution.

6.4 Effectiveness Evaluation

Temporal Cohesiveness. Figure 15 illustrates the temporal cohesiveness of communities discovered by our $(k, T_q, \mathcal{P}_\delta)$-core compared to traditional $(k, \mathcal{P})$-core. We measure the average temporal distance between community members with varying k, where lower values indicate stronger temporal connections. We randomly sample 10,000 pairs of community members and calculate their average time gap in community candidature. Using default values for δ and T_q in our approach (with $(k, \mathcal{P})$-core using $\delta = T_q$), our experiments show that δ-constrained communities maintain significantly tighter temporal bonds than the structure-only approach. As we pioneer the $(k, T_q, \mathcal{P}_\delta)$-core model for temporal HIN communities,



(a) DBLP (b) Yelp

Figure 16: K-core distribution

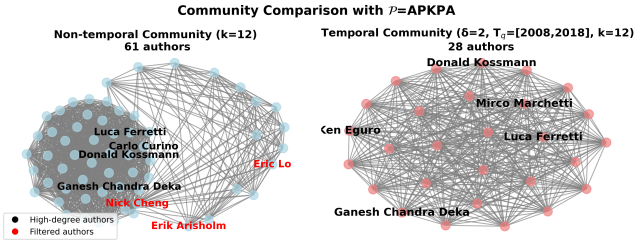


Figure 17: Temporal vs. non-temporal community

our comparison with the non-temporal $(k, \mathcal{P})$-core foundationally validates its temporal effectiveness. Future work can build on this by exploring nuanced temporal metrics and comparisons with other emergent temporal HIN community models.

Core Distribution. Figure 16 compares the distribution of core numbers in communities discovered by our $(k, T_q, \mathcal{P}_\delta)$-core approach versus traditional $(k, \mathcal{P})$-core. Our method produces a significantly more concentrated core distribution, filtering out loosely connected nodes. This filtering effect is particularly pronounced in fine-grained temporal datasets like Yelp, where long-range temporal interactions that don't represent meaningful communities are successfully eliminated.

Case Study. To demonstrate our model's effectiveness over $(k, \mathcal{P})$-core, we present a case study on DBLP with $k = 2$ and $\mathcal{P} = APKPA$ over 10 database-related keywords. With parameters $T_q = [2008, 2018]$ and $\delta = 2$, our temporal model filters the community from 61 to 28 authors. As shown in Figure 17, the resulting temporal community (right) is significantly more concentrated than its non-temporal counterpart (left). This illustrates how temporal constraints effectively prune temporally-distant connections, revealing a core of authors with more cohesive and temporally-aligned research interests.

7 RELATED WORKS

Community Search in HINs. Community search has been extensively studied using various cohesive models, including k-core [4, 14], k-truss [10, 21], k-clique [3, 25], conductance [1, 18, 35], connectivity [23, 32] and so on. When extending to HINs, researchers have leveraged meta-paths [29] to search for communities among same-type nodes. Fang et al. [6] introduced the $(k, \mathcal{P})$-core model, requiring each node to have at least k neighbors via a meta-path $\mathcal{P}$, later refining this with additional constraints [5]. Guo et al. [7] enhanced scalability through sparse matrix multiplication techniques for $(k, \mathcal{P})$-core decomposition. Other cohesive models include Yang et al.'s [36] $(k, \mathcal{P})$-truss model, requiring each edge to be supported by $k-2$ meta-triangles, and Hu et al.'s [9] $(k, \mathcal{P})$-clique model for higher-order cohesiveness. Recent specialized approaches include relational communities utilizing multiple paths [12], star-schema communities centered around key entities [13], and influential communities maximizing propagation impact [41]. Despite these advances, all these models operate in static environments, overlooking the crucial temporal dynamics of real-world interactions.

Temporal Community Search. Temporal community search got its prevalence in recent years, with many works exploring the communities in temporal environments. A prominent direction focuses on querying historical cores in temporal networks. Early works [17, 31, 33] focused on fundamental techniques for core decomposition and maintenance in temporal networks, addressing computational challenges as graph structures evolve over time. Yang et al.[34] extended this with the TCQ problem, finding distinct k-cores in time subintervals using their Tightest Time Interval concept. Zhong et al. [40] further generalized the approach with Temporal (k, X)-Core Queries supporting user-defined metrics like community size or interaction frequency. Beyond core-based models, researchers have explored span-constrained triangles [8], periodic community patterns [22], and query-centered temporal search [20]. While some works have examined community evolution in dynamic heterogeneous networks [30], they typically treat temporality as discrete snapshots rather than continuous constraints. Most temporal community models focus on homogeneous networks with direct edge-based connections. While their flexible temporal query designs were inspirational for our approach, they fail to address the unique challenges of heterogeneous networks where communities form through meta-paths spanning multiple edges. This gap highlights the need for our model that bridges these two streams by developing a unified framework that addresses both heterogeneous structure and temporal dynamics.

8 CONCLUSION

In this work, we introduced the Temporal Heterogeneous Information Network Community Search (THCS) problem, which extends traditional approaches by incorporating temporal constraints. Our $(k, T_q, \mathcal{P}_\delta)$-core model ensures communities are both structurally cohesive and temporally meaningful. We proposed two solutions: online algorithms (our Center-based Sliding Window and Incremental Center Expansion using meta-path symmetry and dynamic tracking) and the TCI-Index with novel compression for frequent queries. Extensive experiments demonstrated that our methods effectively reduce index size while maintaining query performance, identifying communities with tighter temporal bonds than traditional approaches. Future work includes incremental index maintenance, parallel processing techniques, and supporting more complex temporal patterns for domain-specific applications.

ACKNOWLEDGMENTS

This research was jointly supported by ARC Discovery and DECRA Projects under Grant No. DP250100536, DP220102191, DP240101591 and DE240100200.

REFERENCES

[1] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*. IEEE, 475–486.

[2] Deng Cai, Zheng Shao, Xiaofei He, Xifeng Yan, and Jiawei Han. 2005. Mining hidden community in heterogeneous social networks. In *Proceedings of the 3rd international workshop on Link discovery*. 58–65.

[3] 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.

[4] 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.

[5] Yixiang Fang, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2021. Cohesive subgraph search over big heterogeneous information networks: Applications, challenges, and solutions. In *Proceedings of the 2021 International Conference on Management of Data*. 2829–2838.

[6] 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.

[7] Yucan Guo, Chenhao Ma, and Yixiang Fang. 2024. Efficient Core Decomposition Over Large Heterogeneous Information Networks. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2393–2406.

[8] Chuhan Hu, Ming Zhong, Yuanyuan Zhu, Tieyun Qian, Ting Yu, Hongyang Chen, Mengchi Liu, and X Yu Jeffrey. 2024. Querying Cohesive Subgraph Regarding Span-Constrained Triangles on Temporal Graphs. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 3338–3350.

[9] Jiafeng Hu, Reynold Cheng, Kevin Chen-Chuan Chang, Aravind Sankar, Yixiang Fang, and Brian YH Lam. 2019. Discovering maximal motif cliques in large heterogeneous information networks. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 746–757.

[10] 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.

[11] Mubashir Imran, Hongzhi Yin, Tong Chen, Zi Huang, and Kai Zheng. 2022. Dehin: a decentralized framework for embedding large-scale heterogeneous information networks. *IEEE Transactions on Knowledge and Data Engineering* 35, 4 (2022), 3645–3657.

[12] 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.

[13] Yangqin Jiang, Yixiang Fang, Chenhao Ma, Xin Cao, and Chunshan Li. 2022. Effective community search over large star-schema heterogeneous information networks. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2307–2320.

[14] 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.

[15] 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.

[16] Rong-Hua Li, Jeffrey Xu Yu, and Rui Mao. 2013. Efficient core maintenance in large dynamic graphs. *IEEE Transactions on Knowledge and Data Engineering* 26, 10 (2013), 2453–2465.

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

[18] Longlong Lin, Ronghua Li, and Tao Jia. 2023. Scalable and effective conductance-based graph clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 4471–4478.

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

[20] 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.

[21] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. Truss-based community search over large directed graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2183–2197.

[22] 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.

[23] Natali Ruchansky, Francesco Bonchi, David García-Soriano, Francesco Gullo, and Nicolas Kourtellis. 2015. The minimum wiener connector problem. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 1587–1602.

[24] Ahmet Erdem Saríyüce, Buğra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, and Ümit V Çatalyürek. 2013. Streaming algorithms for k-core decomposition. *Proceedings of the VLDB Endowment* 6, 6 (2013), 433–444.

[25] Jing Shan, Derong Shen, Tiezheng Nie, Yue Kou, and Ge Yu. 2016. Searching overlapping communities for group query. *World Wide Web* 19, 6 (2016), 1179–1202.

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

[27] Chuan Shi, Chong Zhou, Xiangnan Kong, Philip S Yu, Gang Liu, and Bai Wang. 2012. Heterecom: a semantic-based recommendation system in heterogeneous networks. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1552–1555.

[28] Yixin Song, Lihua Zhou, Peizhong Yang, Jialong Wang, and Lizhen Wang. 2024. CS-DAHIN: Community Search Over Dynamic Attribute Heterogeneous Network. *IEEE Transactions on Knowledge and Data Engineering* (2024).

[29] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment* 4, 11 (2011), 992–1003.

[30] Yizhou Sun, Jie Tang, Jiawei Han, Manish Gupta, and Bo Zhao. 2010. Community evolution detection in dynamic heterogeneous information networks. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*. 137–146.

[31] Yifu Tang, Jianxin Li, Nur Al Hasan Haldar, Ziyu Guan, Jiajie Xu, and Chengfei Liu. 2022. Reliable community search in dynamic networks. *arXiv preprint arXiv:2202.01525* (2022).

[32] Hanghang Tong and Christos Faloutsos. 2006. Center-piece subgraphs: problem definition and fast solutions. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 404–413.

[33] 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.

[34] Junyong Yang, Ming Zhong, Yuanyuan Zhu, Tieyun Qian, Mengchi Liu, and Jeffrey Xu Yu. 2023. Scalable time-range k-core query on temporal graphs (full version). *arXiv preprint arXiv:2301.03770* (2023).

[35] Renchi Yang, Xiaokui Xiao, Zhewei Wei, Sourav S Bhowmick, Jun Zhao, and Rong-Hua Li. 2019. Efficient estimation of heat kernel pagerank for local clustering. In *Proceedings of the 2019 International Conference on Management of Data*. 1339–1356.

[36] 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.

[37] 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).

[38] Xiao Yu, Xiang Ren, Yizhou Sun, Bradley Sturt, Urvashi Khandelwal, Quanquan Gu, Brandon Norick, and Jiawei Han. 2013. Recommendation in heterogeneous information networks with implicit user feedback. In *Proceedings of the 7th ACM conference on Recommender systems*. 347–350.

[39] Yikai Zhang, Jeffrey Xu Yu, Ying Zhang, and Lu Qin. 2017. A fast order-based approach for core maintenance. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 337–348.

[40] Ming Zhong, Junyong Yang, Yuanyuan Zhu, Tieyun Qian, Mengchi Liu, and Jeffrey Xu Yu. 2024. A Unified and Scalable Algorithm Framework of User-Defined Temporal $(k, \mathcal{X})$(k,X)-Core Query. *IEEE Transactions on Knowledge and Data Engineering* 36, 7 (2024), 2831–2845. https://doi.org/10.1109/TKDE.2023.3349310

[41] 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.

[42] Honglei Zhuang, Jing Zhang, George Brova, Jie Tang, Hasan Cam, Xifeng Yan, and Jiawei Han. 2014. Mining query-based subnetwork outliers in heterogeneous information networks. In *2014 IEEE International Conference on Data Mining*. IEEE, 1127–1132.