Evolution Forest Index: Towards Optimal Temporal k-Core Component Search via Time-Topology Isomorphic Computation

Junyong Yang
School of Computer Science
Wuhan University
Wuhan, China
thomasyang@whu.edu.cn

Ming Zhong*
School of Computer Science
Wuhan University
Wuhan, China
clock@whu.edu.cn

Yuanyuan Zhu
School of Computer Science
Wuhan University
Wuhan, China
yyzhu@whu.edu.cn

Tieyun Qian
Wuhan University
Wuhan, China
qty@whu.edu.cn

Mengchi Liu
South China Normal University
Guangzhou, China
liumengchi@scnu.edu.cn

Jeffrey Xu Yu
The Chinese University of Hong Kong
Hong Kong, China
yu@se.cuhk.edu.hk

ABSTRACT

For a temporal graph like transaction network, finding a densely connected subgraph that contains a vertex like a suspicious account during a period is valuable. Thus, we study the Temporal k-Core Component Search (TCCS) problem, which aims to find a connected component of temporal k-core for any given vertex and time interval. Towards this goal, we propose a novel Evolution Forest Index (EF-Index) that can address TCCS in optimal time. Essentially, EF-Index leverages the evolutionary order on temporal k-cores to both compress the connectivity between vertices in temporal k-cores of all time intervals into a minimum set of compactest Minimum Temporal Spanning Forests (MTSFs) and retrieve MTSF for a given time interval rapidly. Here, a crucial innovation is that, we extend the temporal k-core evolution theory by introducing a pair of time-topology isomorphic relations, on top of which the evolutionary order in topology domain can be simply computed by a "kernel function" in time domain. Moreover, we design an efficient mechanism to update EF-Index incrementally for dynamic edge streams. The experimental results on a variety of real-world temporal graphs demonstrate that, EF-Index outperforms the state-of-the-art approach by 1-3 orders of magnitude on processing TCCS, and its space overhead is reduced by 4-5 orders of magnitude compared with preserving connectivity uncompressedly.

PVLDB Reference Format:
Junyong Yang, Ming Zhong, Yuanyuan Zhu, Tieyun Qian, Mengchi Liu, and Jeffrey Xu Yu. Evolution Forest Index: Towards Optimal Temporal k-Core Component Search via Time-Topology Isomorphic Computation. PVLDB, 17(11): 2840 - 2853, 2024.
doi:10.14778/3681954.3681967

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

*The corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097.
doi:10.14778/3681954.3681967

1 INTRODUCTION

Plenty of evidence suggests that real-world graphs observed in a variety of applications are far from static, as indicated by [26]. For example, SNAP [20] and KONECT [18] projects have collected dozens of temporal graphs, such as social networks, transaction networks and user-item bipartite graphs, with diverse scales, time densities and time granularities (see our empirical study in Section 5.1). The temporality in graphs matters for revealing the correlation of edge activation, the order in which vertices join evolving communities, the rhythm of subgraph pattern appearance, etc. Meanwhile, due to the meet of complexities of time and topology, new research challenges arise from a wide range of traditional graph analysis objects, such as reachability [39], path [40], triangle [30], butterfly [2], centrality [29], motif [13], kernel [28], embedding [34], cohesive subgraph [44], community [22], subgraph isomorphism [33], etc.

Recently, querying k-cores that exist during a given time interval on temporal graphs becomes an emerging research topic. Wu et al [41] firstly propose the concept of temporal k-core, and design a particular core decomposition algorithm to deal with the frequency-constrained temporal k-core query. Galimberti et al [12] propose the concept of span-core, the most frequent temporal k-core in which two vertices have interaction at all time. Yu et al [45] propose the concept of historical k-core that can be seen as the de-temporal version of temporal k-core, and design a PHC-Index to address the historical k-core query or search by exploiting a threshold moment called core time insightfully. Yang et al [44] address the range query of temporal k-core (called TCQ) with an online OTCD algorithm, which is dramatically scalable for discovering a threshold period called tightest time interval beyond the core time. Zhong et al [47] extend TCQ to TXCQ that can filter temporal k-cores by an arbitrary user-defined metric, and address TXCQ with an improved OTCD* algorithm, which exploits another important threshold period called loosest time interval. In summary, the latest studies have built a solid theoretical foundation for temporal k-core query optimization.

However, the real-world k-cores are usually comprised of many separate components, especially when only considering a historical period because small components are usually merging into large ones gradually over time (namely, densification law [19]). Thus, querying the whole temporal k-core could return lots of irrelevant vertices. For that, many recent works [3, 9, 24, 38] focus on finding

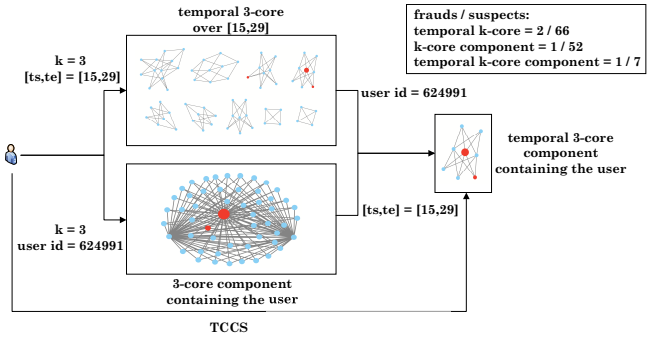


Figure 1: A case study of fraud detection on DGraph [16].

connected components of k-core. It motivates us to study Temporal k-Core Component Search (TCCS), which directly finds the connected component containing a given vertex in the temporal k-core over a given time interval. As shown in Table 1, TCCS is the first work of this category.

TCCS can facilitate a variety of real-world applications. For example, a typical financial anti-fraud pipeline is as follows: 1) detect user communities like k-cores that emerge during a specific period, 2) derive the candidate components containing blacklisted users from detected communities, and 3) investigate the other users in candidate components manually. Obviously, we can accelerate this pipeline with TCCS. As illustrated in Figure 1, the temporal k-core of a given time interval has a number of components, most of which actually contain none unknown fraud (small red vertex). While, if we find the k-core component that contains a known fraud (big red vertex) without specifying time interval, it still has too many suspects for investigation. In contrast, we can use TCCS to directly get the final temporal k-core component, in which only 7 suspects need to be investigated in order to detect the unknown fraud.

Although the querying of temporal k-core has been studied by [44, 45], the searching of temporal k-core component based on that will still rely on brute force. Either we use the TCD algorithm [44] to online compute the temporal k-core and then find a connected component from it, or we directly traverse the graph from the given vertex by BFS and meanwhile use the PHC-Index as a filter to prune vertices not in the temporal k-core, a possibly large number of vertices/edges not in the result will be inevitably traversed.

In this paper, we aim to propose a time-optimal solution for the valuable TCCS. For that, we try to preserve, maintain and retrieve the minimum spanning forests of all temporal k-cores in a temporal graph, which is a great challenge since the number of temporal k-cores is quadratic with that of distinct timestamps. To overcome the challenge, we leverage the evolutionary order of temporal k-core to index the forests with near minimum space. Moreover, to obtain the evolutionary order efficiently, we present a novel time-topology isomorphic computation paradigm.

Our contributions are summarized as follows.

- **Theory**. We extend the existing theory of temporal k-core evolution by introducing a pair of time-topology isomorphic lineage relations on distinct cores and time zones respectively. Thus, the transitive reduction of evolutionary order on distinct cores can be simply computed by a "kernel function" on time zones. Then, we propose an instance of Hasse diagram named lineage graph to model the global non-linear temporal k-core evolution for temporal graphs.
- **Index**. We design a sophisticated Evolution Forest Index (EF-Index) that compresses the minimum spanning forests of temporal k-cores losslessly into a set of compactest Minimum Temporal Spanning Forests (MTSFs), which preserve the evolving connectivity for each lineage chain in the minimum chain cover of lineage graph. Moreover, EF-Index leverages the lineage graph to retrieve MTSF in at most linear time to the layer number of lineage graph.
- **Search**. We develop a time-optimal algorithm to address TCCS. It neither decompresses the retrieved MTSF nor traverses any vertex not in the result, relying on the temporal labels of edges in MTSFs.
- **Maintenance**. We reveal the patterns of time zone appearance and expansion caused by dynamical graph updating, and present an incremental EF-Index maintaining mechanism. It follows the patterns to update the lineage graph without redundant core decomposition and then uses a heuristic strategy to update the lineage chain cover rapidly.
- **Experiment**. We study a set of temporal graphs from SNAP and KONECT projects empirically, and compose reasonable test graphs with respect to observations. We conduct experiments including index construction, index maintenance, and query processing on test graphs. The results demonstrate that, our approach both outperforms the state-of-the-art approach by a remarkable margin and reduces indexing and maintaining costs dramatically.

Table 1: Classification of k-core query works.

Core Type	Component	Whole
Temporal	**TCCS (This Work)**	[12, 41, 44, 45, 47]
Static	[3, 9, 24, 38]	Well Studied

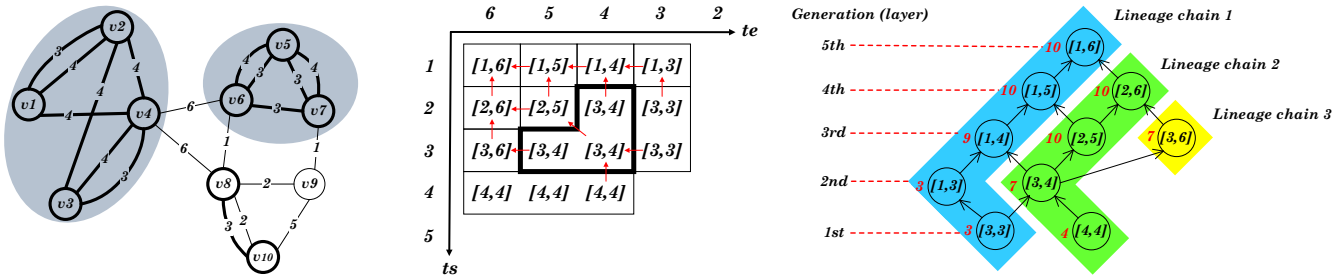
2 THEORETICAL FOUNDATION

2.1 Problem Formulation

We consider a **temporal graph** $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each edge $(u, v, t) \in \mathcal{E}$ is a triplet denoting $u, v \in \mathcal{V}$ have an interaction at time t. Without loss of generality, we use consecutive integers 1, 2, $\cdots$, n to denote all timestamps in $\mathcal{G}$. Figure 2a illustrates a temporal graph as our running example.

Given a **time interval** $[ts, te]$ with $1 \leqslant ts \leqslant te \leqslant n$, we can induce a **projected graph** $\mathcal{G}_{[ts,te]}$ from $\mathcal{G}$ by removing all edges that fall out of $[ts, te]$, and a **temporal** k**-core** $\mathcal{T}^k_{[ts,te]}(\mathcal{G})$ is defined as the maximal subgraph of $\mathcal{G}_{[ts,te]}$ such that each vertex has at least k neighbor vertices in the recent literatures [41, 44], where $k \geqslant 2$ is an integer. We denote by $[ts, te] \sqsubseteq [ts', te']$ that $[ts, te]$ is a subinterval of $[ts', te']$, namely, $ts \geqslant ts'$ and $te \leqslant te'$.

Note that, a temporal k-core is not guaranteed to be connected. Thus, we propose a novel community search problem for temporal graphs, namely, **Temporal k-Core Component Search (TCCS)**.



(a) Temporal graph and k-core $\mathcal{T}^2_{[3,4]}$. (b) TTIs, Time zones and lineage. (c) Lineage graph and chain cover.

Figure 2: The illustrative examples of fundamental concepts in temporal k-core evolution ($k = 2$).

Problem 1 (Temporal k-Core Component Search). *Given $\mathcal{G}$, k, $[ts, te]$ and a vertex u, retrieve the connected component of $\mathcal{T}^k_{[ts,te]}(\mathcal{G})$ that contains u.*

Example 1. *As illustrated in Figure 2a, for the time interval $[3, 4]$, the projected graph $\mathcal{G}_{[3,4]}$ contains the bold vertices and edges, and the temporal 2-core (the shaded subgraph) induced from $\mathcal{G}_{[3,4]}$ has two connected components $\{v_1, v_2, v_3, v_4\}$ and $\{v_5, v_6, v_7\}$. TCCS aims to find a community like $\{v_5, v_6, v_7\}$ that satisfies cohesiveness, time interval, and connectivity constraints for a given vertex like v_6.*

For brevity, we denote $\mathcal{T}^k_{[ts,te]}(\mathcal{G})$ by $\mathcal{T}^k_{[ts,te]}$ and the component of $\mathcal{T}^k_{[ts,te]}$ that contains u by $C^k_{[ts,te]}(u)$ from here.

2.2 Review of Temporal k-Core Evolution

The previous researches [44, 47] have revealed the important concepts, properties and theorems that facilitate the scalable online processing of temporal k-core/$(k, \mathcal{X})$-core queries. Our index and search algorithm for TCCS also rely on these theoretical findings to achieve the space and time optimality. Thus, we review the existing theory, but from a new systematic and insightful perspective called temporal k-core evolution, in this subsection.

A temporal k-core $\mathcal{T}^k_{[ts,te]}$ will evolve, namely, change its topological structure with the variation of ts or te, following the varied projected graph $\mathcal{G}_{[ts,te]}$. Intuitively, we say $[ts, te]$ is expanded to $[ts', te']$ if $[ts, te] \sqsubset [ts', te']$, or shrunk to $[ts', te']$ if $[ts, te] \sqsupset [ts', te']$. From the procedure of evolution, there is the following fundamental observation.

Theorem 1 (Monotonic and Discrete Evolution). *A temporal k-core $\mathcal{T}^k_{[ts,te]}$ can only but may not always have vertices and edges added (or deleted) with the expanding (or shrinking) of $[ts, te]$.*

Example 2. *Consider the temporal 2-core $\mathcal{T}^2_{[3,4]}$ in Figure 2a. Firstly, we expand the time interval to $[3, 5]$, and the topological structure of $\mathcal{T}^2_{[3,5]}$ is unchanged with respect to $\mathcal{T}^2_{[3,4]}$, so that $\mathcal{T}^2_{[3,4]}$ and $\mathcal{T}^2_{[3,5]}$ are actually identical subgraphs. Then, we expand the time interval to $[3, 6]$, and $\mathcal{T}^2_{[3,6]}$ has a new edge $(v_4, v_6, 6)$ added monotonically.*

Theorem 1 implies that, there will be threshold time intervals during the evolution of temporal k-core. We call them **Loosest Time Interval (LTI)** and **Tightest Time Interval (TTI)** respectively.

Definition 1 (Loosest/Tightest Time Interval). *For a temporal k-core $\mathcal{T}^k_{[ts,te]}$, a time interval $[ts', te']$ is the loosest/tightest, if and only if $\mathcal{T}^k_{[ts',te']}$ is topologically identical to $\mathcal{T}^k_{[ts,te]}$ and there does not exist an expanded/shrank time interval $[ts'', te''] \sqsupset/\sqsubset [ts', te']$ such that $\mathcal{T}^k_{[ts'',te'']}$ is also topologically identical to $\mathcal{T}^k_{[ts,te]}$.*

Intuitively, expanding an LTI or shrinking a TTI will make the corresponding temporal k-core evolve to a different subgraph. For a temporal k-core, its TTI is proved to be simply $[t_{min}, t_{max}]$, where t_{min} and t_{max} are the minimum and maximum timestamps of its edges respectively, which means the obtain of TTI is simple.

Example 3. *For ease of understanding, Figure 2b illustrates a time coordinate system in which the two axes represent start time ts and end time te respectively. Each grid cell coordinated by (ts, te) represents the time interval $[ts, te]$ that can induce a temporal k-core, and the TTI of $\mathcal{T}^k_{[ts,te]}$ is recorded in the cell. We can see both TTIs in the cells $[2, 4]$ and $[3, 5]$ are $[3, 4]$, which means $\mathcal{T}^2_{[2,4]}$ and $\mathcal{T}^2_{[3,5]}$ will not evolve when shrinking their time interval to $[3, 4]$. On the contrary, both $[2, 4]$ and $[3, 5]$ are the LTIs of $\mathcal{T}^2_{[3,4]}$.*

The significance of TTI is that, due to its three important properties [44], it connects the temporal and topological worlds, and makes two temporal k-cores topologically comparable with respect to their TTIs. Note that, different from TTI, a temporal k-core may have multiple LTIs. The LTIs of a temporal k-core surely contain its TTI as a subinterval, and meanwhile are partially overlapped with each other.

Theorem 1 also implies that, there are a number of "species" (namely, distinct temporal k-core structures) distributed in different sets of time intervals during the global procedure of temporal k-core evolution. Thus, we have the following definitions.

Definition 2 (Distinct Core). *Given $\mathcal{G}$ and k, a distinct core $\mathbb{T}^k$ is a distinct structure of temporal k-cores, whose vertex set and edge set are denoted by $\mathbb{V}^k$ and $\mathbb{E}^k$ respectively. The TTI and LTIs of $\mathbb{T}^k$ are the TTI and all LTIs of its identical temporal k-cores respectively.*

Definition 3 (Time Zone). *Given $\mathcal{G}$ and k, a time zone $\mathbb{Z}^k$ is a set of time intervals from which we can induce a complete set of identical temporal k-cores (represented by a distinct core $\mathbb{T}^k$). The TTI and LTIs of $\mathbb{T}^k$ are also called the TTI and LTIs of $\mathbb{Z}^k$ respectively.*

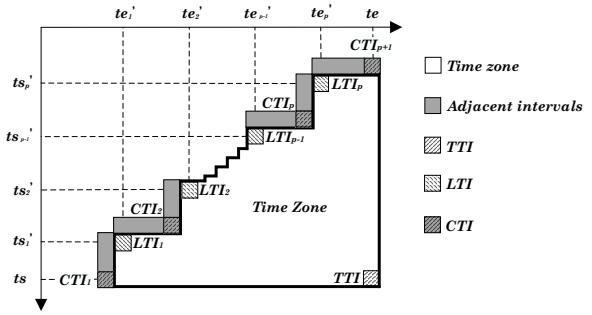


Figure 3: The adjacent and corner time intervals of time zone.

An important observation on time zone is that, the distribution of intervals in a time zone can be determined only by its TTI and LTIs, as described by the following property.

Property 1 (Boundness). *For a time zone $\mathbb{Z}^k$, a time interval $[ts, te] \in \mathbb{Z}^k$ if and only if $[ts', te'] \sqsubseteq [ts, te] \sqsubseteq [ts'', te'']$, where $[ts', te']$ and $[ts'', te'']$ are the TTI and an LTI of $\mathbb{Z}^k$ respectively.*

Example 4. *As illustrated in Figure 2b, the time intervals with a same TTI are merged into a time zone. If we view the coordinate system from a spatial perspective, each time zone is composed of a set of rectangles that have an overlapped bottom-right cell coordinated by its TTI. Moreover, the top-left cells of each rectangle are its LTIs. The time zone marked by bold border has two rectangles, in which the cell $[3, 4]$ is its TTI and the cells $[2, 4]$ and $[3, 5]$ are its LTIs.*

In a global view, the temporal k-core evolution for a temporal graph is decomposed to a set of non-linear stages, which can be seen as distinct cores $\mathbf{T}^k = \{\mathbb{T}_1^k, \mathbb{T}_2^k, \cdots, \mathbb{T}_m^k\}$ from the perspective of topology, or time zones $\mathbf{Z}^k = \{\mathbb{Z}_1^k, \mathbb{Z}_2^k, \cdots, \mathbb{Z}_m^k\}$ from the perspective of time, and there is a bijection between $\mathbf{T}^k$ and $\mathbf{Z}^k$.

Note that, the number m of distinct cores (time zones) is subquadratic to the number n of timestamps, and the exponent varies for different temporal graphs. The higher degree of discretization of evolution, the smaller value of m.

2.3 New Lineage Relation and Lineage Graph

The current temporal k-core evolution theory only considers the distinct cores (or bijective time zones) separately, and leaves the relations on them unexplored. Although the theory is sufficient for the optimization of processing TCQ [44] and TXCQ [47], there are inevitably algorithmic problems like TCCS concern particular relations on them like the evolutionary order. Therefore, we extend the existing theory by introducing a pair of isomorphic **lineage relations** on distinct cores and time zones respectively.

Firstly, let us consider an intuitive **core lineage** on distinct cores.

Definition 4 (Core Lineage). *Given the set of distinct cores $\mathbf{T}^k = \{\mathbb{T}_1^k, \mathbb{T}_2^k, \cdots, \mathbb{T}_m^k\}$ with respect to $\mathcal{G}$ and k, the core lineage $\mathbf{L}_t^k$ is a relation on $\mathbf{T}^k$, and $(\mathbb{T}_i^k, \mathbb{T}_j^k) \in \mathbf{L}_t^k$ if and only if $\mathbb{T}_i^k$ directly evolves to $\mathbb{T}_j^k$ with the expanding of time interval, which means there does not exists $\mathbb{T}_l^k$ such that both $(\mathbb{T}_i^k, \mathbb{T}_l^k) \in \mathbf{L}_t^k$ and $(\mathbb{T}_l^k, \mathbb{T}_j^k) \in \mathbf{L}_t^k$ hold, for $1 \leqslant i, j, l \leqslant m$.*

Algorithm 1: Lineage graph construction.

Input: A temporal graph $\mathcal{G}$ and an integer k
Output: The lineage graph $\mathbf{G}^k$

```
1 Generate all distinct cores {T_1^k, T_2^k, ..., T_m^k} and time zones
    {Z_1^k, Z_2^k, ..., Z_m^k} for G by OTCD* algorithm [47]
2 Add a node to G^k for each of T_1^k, T_2^k, ..., T_m^k with its TTI
3 for i ← 1 to m do
4     Compute the lineages to Z_i^k by Equation (1)
5     Add an edge to G^k for each lineage
```

Property 2 (Transitive Reduction of Containment). *The core lineage is a transitive reduction of a partial order "containment" on distinct cores $\mathbf{T}^k$, in which each ordered pair $(\mathbb{T}_i^k, \mathbb{T}_j^k)$ means $\mathbb{T}_i^k$ is contained by $\mathbb{T}_j^k$ as a subgraph.*

The core lineage brings an inherent evolutionary order to the separate distinct cores, so that they are interconnected by lineage. Thus, the global view of temporal k-core evolution can be extended as follows. Each minimal distinct core (1st generation) is evolving with the expanding of time interval. Its topological structure will remain unchanged as long as the expanded time interval is still in the same time zone. Otherwise, it will grow to a new larger distinct core with lineage to it, until it becomes the maximal distinct core $\mathcal{T}_{[1,n]}^k$. Note that, a distinct core may grow to multiple other distinct cores, and multiple distinct cores may grow to another same distinct core. We propose a Hasse diagram named **lineage graph** to represent the evolution procedure.

Definition 5 (Lineage Graph). *Given $\mathcal{G}$ and k, a lineage graph is a directed acyclic graph $\mathbf{G}^k = (\mathbf{T}^k, \mathbf{L}_t^k)$, where each node $\mathbb{T}_i^k \in \mathbf{T}^k$ represents a distinct core and each relation $(\mathbb{T}_i^k, \mathbb{T}_j^k) \in \mathbf{L}_t^k$ represents the lineage from $\mathbb{T}_j^k$ to $\mathbb{T}_i^k$, namely, $\mathbb{T}_i^k$ directly evolves to $\mathbb{T}_j^k$.*

Example 5. *The lineage graph of our example temporal graph is illustrated in Figure 2c. Since there are ten distinct cores (time zones) during the evolution for $k = 2$ (see Figure 2b), the lineage graph also have ten nodes labeled by the their TTIs, which can be seen as the unique ids of both distinct cores and time zones. Moreover, the arrow lines between nodes denote their lineage relations.*

Consequently, a crucial question is raised: how to infer the lineage between two specific distinct cores? To reveal the answer, let us consider the temporal relation between their time zones. As illustrated in Figure 3, for a time zone $\mathbb{Z}_i^k$, its time intervals can be expanded to the adjacent time intervals marked by grey color to trigger the evolution of $\mathbb{T}_i^k$, so that only the distinct cores of time zones containing the adjacent intervals could have lineage to $\mathbb{T}_i^k$ due to Definition 4. Moreover, for a time zone containing adjacent intervals but not any one in the corner marked by slashes additionally, its distinct core is surely not evolved directly from $\mathbb{T}_i^k$, because it can be evolved from the temporal k-core of the corner time interval in the same row or column. Thus, a distinct core has lineage to $\mathbb{T}_i^k$, if and only if its time zones contains a corner time interval of $\mathbb{Z}_i^k$. The formal definition of **Corner Time Interval (CTI)** is as follows.

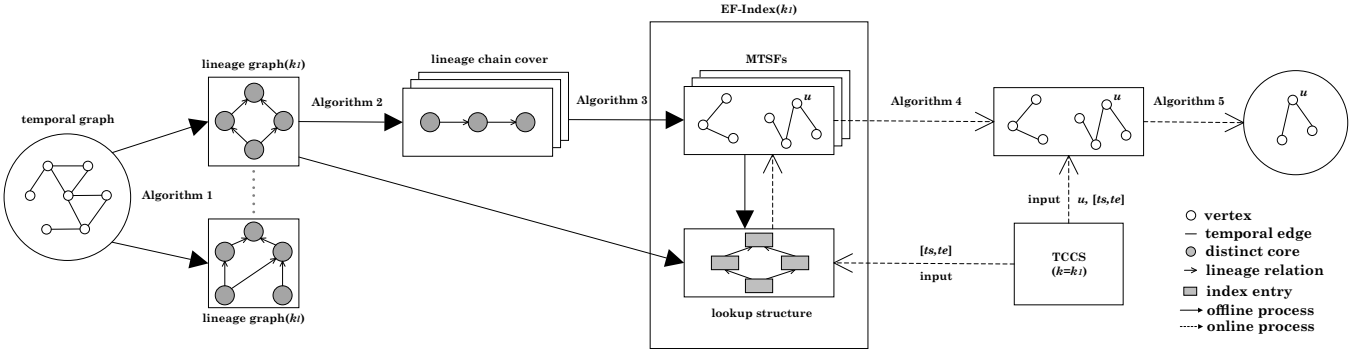


Figure 4: The pipeline of offline EF-Index construction and online TCCS processing.

Definition 6 (Corner Time Interval). *For a time zone with TTI $[ts, te]$ and LTIs $[ts'_1, te'_1], [ts'_2, te'_2], \cdots, [ts'_p, te'_p]$, the $p+1$ corner time intervals are $[ts, te'_1 + 1]$, $[ts'_1 - 1, te'_2 + 1]$, $\cdots$, $[ts'_p - 1, te]$, where p denotes the number of LTIs.*

Property 3 (Tightest). *A CTI of a time zone is a TTI of another time zone, and vice versa, if both time zones exist.*

Interestingly, a CTI identifies the time zone containing it uniquely with respect to Property 3. It means, given a distinct core, we can find the distinct cores with lineage to it only through the CTIs of its time zone. Thus, we can define the following **zone lineage** on time zones as the isomorphic relation of core lineage.

Definition 7 (Zone Lineage). *Given the set of time zones $\mathbf{Z}^k = \{\mathbb{Z}^k_1, \mathbb{Z}^k_2, \cdots, \mathbb{Z}^k_m\}$ with respect to $\mathcal{G}$ and k, the zone lineage $\mathbf{L}^k_z$ is a relation on $\mathbf{Z}^k$, and $(\mathbb{Z}^k_i, \mathbb{Z}^k_j) \in \mathbf{L}^k_z$ if and only if the TTI of $\mathbb{Z}^k_j$ is a CTI of $\mathbb{Z}^k_i$.*

Theorem 2 (Time-Topology Isomorphism). *For the bijective sets $\mathbf{T}^k$ and $\mathbf{Z}^k$, $(\mathbb{T}^k_i, \mathbb{T}^k_j) \in \mathbf{L}^k_t$ if and only if $(\mathbb{Z}^k_i, \mathbb{Z}^k_j) \in \mathbf{L}^k_z$.*

The time-topology isomorphism of two lineage relations enables a "kernel function" that infers the lineage relation on distinct cores in a $(|\mathcal{V}| + |\mathcal{E}|)$-dimensional space by inferring the lineage relation on their time zones in a $(2 + 2p)$-dimensional space as

$$\begin{aligned}\mathbb{I}((\mathbb{T}^k_i, \mathbb{T}^k_j) \in \mathbf{L}^k_t) &= \mathbb{I}((\mathbb{Z}^k_i, \mathbb{Z}^k_j) \in \mathbf{L}^k_z) \\ &= \mathbb{I}(\exists [ts, te] \in \mathbb{Z}^k_i.\text{CTI s.t.} [ts, te] = \mathbb{Z}^k_j.\text{TTI})\end{aligned} \tag{1}$$

where $\mathbb{I}()$ is an indicator function. Note that, p is only a little higher than 1 on average in our empirical experiments, so the kernel function is in a domain with extremely less dimensions. Moreover, the kernel function also reduces the computational complexity significantly, since $\mathbf{L}^k_t$ that represents transitively reduced containment cannot be trivially obtained like $\mathbf{L}^k_z$.

Example 6. *Compare the zone lineage and core lineage illustrated in Figure 2b and 2c respectively. We can see the one-to-one correspondence between the red and black arrow lines. Obviously, the obtain of zone lineage is simple and straightforward.*

Lastly, Algorithm 1 presents the pseudo code of building a lineage graph, which is the foundation of optimal TCCS processing. By using the OTCD* algorithm [47], we can compute all distinct cores and time zones for a given temporal graph efficiently. Then, the computational complexity of generating edges for a lineage graph is $O(\sum_{i=1}^{m} |\mathbb{Z}^k_i.\text{CTI}|)$, which is approximately $O(m)$ in practice.

Algorithm 2: Minimum lineage chain cover generation.

Input: A lineage graph $\mathbf{G}^k$ (layered as in Figure 2c)
Output: The minimum lineage chain cover of $\mathbf{G}^k$

```
Build a bipartite graph G = (U, V, E), where U, V = T^k and
  E = {(u, v) | u ∈ U, v ∈ V, (u, v) ∈ L_t^k}
Get the maximum match of G as M by Hopcroft-Karp [14]
while there exists untraversed node in G^k do
    T_i^k ← next untraversed node in descending layer order
    Generate a chain inversely from T_i^k with respect to M
```

3 INDEX CONSTRUCTION AND USAGE

3.1 Overview

To retrieve a specific temporal k-core component from an index in the optimal time, the index needs to 1) preserve the sets of vertices for temporal k-cores of each legal time interval, 2) preserve the connectivity between those vertices in each temporal k-core, and 3) map a TCCS instance (namely, specific k, $[ts, te]$ and u) to an index entry, so that the result connected component can be derived directly without traversing vertices not belong to it.

A straightforward design of such an index is to store a Minimum Spanning Forest (MSF, the union of minimum spanning trees of each component) for each temporal k-core. However, its space complexity is dramatically high. For each reasonable value of k, it can be estimated as $O(n^2|\mathcal{V}^k|)$, where n is the number of timestamps in $\mathcal{G}$, and $|\mathcal{V}^k|$ is the average number of vertices for all temporal k-cores, which is bounded by $|\mathcal{V}|$ in the worst case.

Therefore, we propose a sophisticated **Evolution Forest Index (EF-Index)** that guarantees the same size of search space as indexing all MSFs while being many orders of magnitude smaller. The pipeline of constructing EF-Index is illustrated in Figure 4. Firstly, we leverage the lineage graph to compute a **lineage chain cover** and only preserve the largest distinct cores on tails of each chain, thereby compressing the space of preserving vertex sets significantly. Secondly, for each chain, we generate a novel **Minimum**

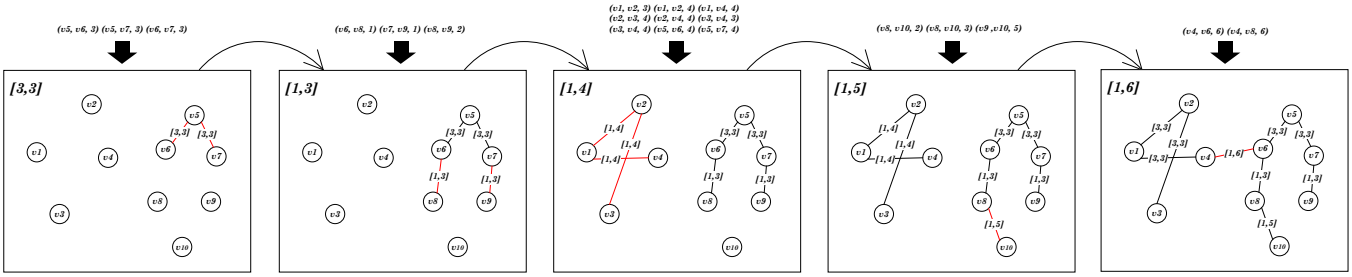


Figure 5: A step-by-step demonstration of building an MTSF for a lineage chain with five distinct cores.

Temporal Spanning Forest (MTSF) to preserve the evolving connectivity in all temporal k-cores on the chain within the minimum space. Lastly, an efficient **lookup structure** is designed to retrieve the MTSF for a given time interval. As a result, to find the result temporal k-core component, we only need to search the retrieved MTSF from the given vertex with a particular time constraint.

3.2 Vertex Set Compression via Lineage Chain

In this subsection, we propose a storage scheme of preserving vertex sets for temporal k-cores, which compresses them losslessly by two steps. Firstly, we only preserve the distinct cores but not all temporal k-cores, since each temporal k-core is identical to a distinct core. Secondly, we further compress the distinct cores by differential encoding. If a distinct core contains another that has been preserved, only the differential vertices need to be stored.

However, computing the containment relation on distinct cores is time-consuming. The brute force method needs to compare each pair of distinct cores. Although TTI can help to fulfill each comparison in constant time, the complexity is still quadratic to the number of distinct cores. Thus, we consider the lineage relation on distinct cores, which not only is a transitive reduction of containment (Property 2), and also can be computed in linear time to the number of distinct cores (Algorithm 1) for the isomorphism to zone lineage (Theorem 2). Specifically, we partition the non-linear lineage graph into a set of lineage chains first.

Definition 8 (Lineage Chain). *A lineage chain* $\mathbb{C}^k = [\mathbb{T}^k_1, \mathbb{T}^k_2, \cdots, \mathbb{T}^k_l]$ *is a linearly ordered list of distinct cores by the core lineage* $\mathbf{L}^k_t$*, where* $(\mathbb{T}^k_i, \mathbb{T}^k_{i+1}) \in \mathbf{L}^k_t$ *for* $1 \leqslant i < l$.

Definition 9 (Lineage Chain Cover). *Given a lineage graph* $\mathbf{G}^k$ *with respect to* $\mathcal{G}$ *and* k*, a lineage chain cover is a set of disjoint lineage chains* $\mathbf{C}^k = \{\mathbb{C}^k_1, \mathbb{C}^k_2, \cdots, \mathbb{C}^k_h\}$ *such that each distinct core in* $\mathbf{G}^k$ *appears exactly once in one of them.*

Then, the total space cost of preserving vertex sets is the sum of costs of preserving lineage chains in the cover, which are determined only by the number of vertices of the last and certainly the largest distinct cores on each chain with differential encoding. Moreover, the speed of compressing all chains depends on the total number of edges of their last distinct cores (see Section 3.3).

Thus, for a lineage graph, we obtain a lineage chain cover with the objective to minimize the number of chains. The rationale is twofold. Firstly, this objective is consistent with minimizing the total number of vertices or edges of tail distinct cores, thereby achieving high compression ratio and fast compression speed. Secondly, the computational complexity of finding the minimum-size set of chains is lower than the minimum-weight set (approximately $O(m^{1.5})$ vs. $O(m^3)$) that will achieve the compactest index space. Note that, although the parameterized linear time algorithm [5] is proposed, the parameter is too large in our experiments. The pseudo code of generating minimum chain cover is presented in Algorithm 2.

Algorithm 3: MTSF construction.

Input: A temporal graph $\mathcal{G}$, an integer k, and the lineage chain cover $\mathbf{C}^k$
Output: The MTSFs of each chain in $\mathbf{C}^k$

```
Start the OTCD* algorithm for G and k
while OTCD* does not stop do
    T^k ← the next distinct core returned by OTCD*
    C^k ← the lineage chain in C^k with T^k on tail
    Traverse C^k from T^k, and use TCD operation [44] to get
      the differential edge sets between each pair of adjacent
      distinct cores on C^k
    Call SingleChain(C^k) and collect the returned MTSF
Function SingleChain(C^k = [T^k_1, T^k_2, ..., T^k_l]):
    Initialize an MTSF M with respect to T^k_1
    for i ← 2 to l do
        for each (u, v, t) in the differential edge set between
          T^k_{i-1} and T^k_i do
            if u and v is not connected in M then
                Add an edge between u and v with the TTI
                  of T^k_i as a label in M
    return M
```

Example 7. *As illustrated in Figure 2c, the lineage graph can be partitioned to a minimum set of three chains marked by stripes in different colors. The red number asides each node is its vertex number. Thus, the total cost of this example cover is 10+10+7=27. There could be different minimum chain covers with different total costs (e.g., 29 or 30). Thus, we claim our storage scheme is near compactest.*

Lastly, each preserved vertex is supposed to have a particular label to hint which temporal k-cores it belongs to. While, the label is actually unnecessary because we preserve the temporal information within MTSF and lookup structure in the next subsections.

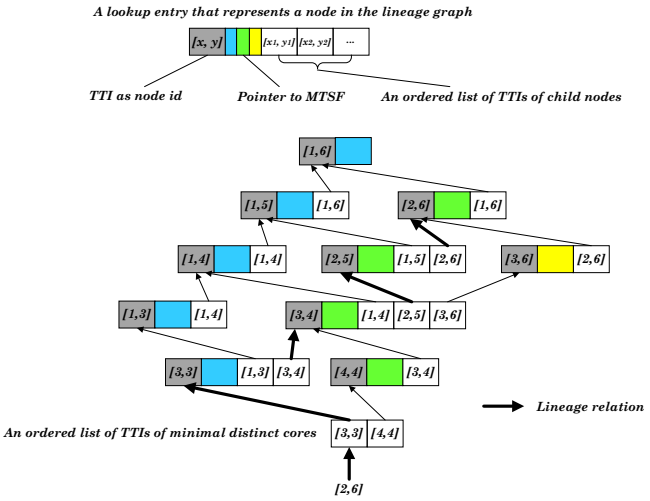


Figure 6: Example of MTSF retrieval by lookup structure.

3.3 Edge Set Compression via MTSF

After compressing the vertex sets of distinct cores via lineage chain, the next challenge becomes how to preserve the connectivity between vertices of distinct cores on each lineage chain. A straightforward solution is to preserve a minimal set of edges for each distinct core, namely, Minimum Spanning Forest (MSF). However, such a solution is not elegant due to the large number of duplicated edges in MSFs of different distinct cores. Inspired by [42], we propose a novel space-optimal structure called Minimum Temporal Spanning Forest (MTSF) that merges and compresses the MSFs of distinct cores on a lineage chain with respect to the following observation.

Theorem 3 (Lasting Connectivity). *Given a lineage chain* $[\mathbb{T}_1^k, \mathbb{T}_2^k, \cdots, \mathbb{T}_l^k]$, *for any two connected vertices in* $\mathbb{T}_i^k$ $(1 \leqslant i < l)$, *they are still connected in the subsequent distinct cores* $\mathbb{T}_{i+1}^k, \cdots, \mathbb{T}_l^k$.

Based on Theorem 3, MTSF incrementally maintains the connectivity between vertices in evolutionary order. Specifically, an MTSF considers the edges of each distinct core on a lineage chain gradually from $\mathbb{T}_1^k$ until $\mathbb{T}_l^k$. Each edge will be added into the MTSF only if it can connect two vertices that have not been connected. Obviously, there is only one single path between each pair of connected vertices in the final MTSF, so that the space cost of MTSF is $O(|\mathbb{V}_l^k|)$, which is the minimum because only preserving the MSF of $\mathbb{T}_l^k$ requires the same space.

Moreover, to determine in which distinct cores two vertices are connected, MTSF assigns a temporal label to each edge, which is the TTI of the distinct core from which the edge is added. Thus, we can decompress an MTSF to the MSF of a specific distinct core $\mathbb{T}_i^k$ by removing the edges whose labels are TTIs of $\mathbb{T}_{i+1}^k, \cdots, \mathbb{T}_l^k$.

Example 8. *Figure 5 demonstrates building an MTSF step-by-step for the lineage chain 1 marked by blue color in Figure 2c. For each distinct core on the chain, a box with its TTI shows the intermediate MTSF that has merged its edges above the box. The newly added edges in each box are highlighted by red color. Initially, we build an MSF of* $\mathbb{T}_1^2$ *and assign a label* $[3, 3]$ *to the edges. Then, for each edge in* $\mathbb{T}_2^2$ *but not in* $\mathbb{T}_1^2$ *like* $(v_6, v_8, 1)$, *we check whether the two vertices* v_6 *and* v_8 *have been connected, and add an edge with a label* $[1, 3]$ *between them into the MTSF if not. In contrast, the edge* $(v_8, v_9, 2)$ *will not be added because there is already a path between* v_8 *and* v_9.

Algorithm 4: EF-Index lookup.

Input: The query time interval $[ts, te]$
Output: The MTSF of lineage chain that contains the distinct core whose time zone contains $[ts, te]$

```
[ts', te'] ← a minimal TTI contained by [ts, te] in entrance
while [ts', te'] is not null do
    Jump to the index entry whose id is [ts', te']
    M ← the link to MTSF
    [ts', te'] ← a TTI contained by [ts, te] in entry's list
return M
```

The pseudo code of building MTSF is presented in Algorithm 3. The result MTSF could be different with respect to the order of input edges, while not hindering the space-optimality of MTSF and the time-optimality of searching MTSF.

When the differential edge sets between distinct cores are available, the computational complexity of `SingleChain()` function is $O(|\mathbb{E}_l^k| \cdot \log^* |\mathbb{V}_l^k|)$ by using an optimized Union-Find structure [11] to maintain MTSF. Note that, a tricky part of building MTSF is the computation of differential edge sets, whose trivial implementation is inefficient. Algorithm 3 exploits the TCD algorithm [44], which deletes the differential edges between distinct cores decrementally, to address it with limited resource. When the resource is unlimited, a simple parallel execution of `SingleChain()` function also works.

3.4 Lookup Structure

Since the vertices and the connectivity between them in all temporal k-cores can be compressed into a set of MTSFs, the design of EF-Index comes to the last mile. That is, for a TCCS instance with a query time interval $[ts, te]$, mapping it to the distinct core whose time zone contains $[ts, te]$ and then to the MTSF of the lineage chain that contains the distinct core, so that the result component can be obtained from the MTSF.

However, the efficient implementation of mapping is non-trivial. Although determining whether the query time interval belongs to a time zone is not complex due to Property 1, the number of time zones is normally sub-quadratic to the number of timestamps for a temporal graph. Thus, it is inefficient to consider each time zone individually for long-standing temporal graphs.

In this subsection, we propose a structure that leverages the lineage graph to lookup the MTSF for a given time interval efficiently with respect to the following observation on lineage graph.

Theorem 4 (All Roads Lead to Rome). *In a lineage graph, there exists at least one directed path from one node with TTI* $[ts, te]$ *to another node with TTI* $[ts', te']$ *as long as* $[ts', te'] \supseteq [ts, te]$, *and meanwhile the TTIs of other nodes on each path are subintervals of* $[ts', te']$.

Based on Theorem 4, we can choose an arbitrary minimal distinct core whose TTI is a subinterval of $[ts, te]$, traverse the distinct

Algorithm 5: Temporal k-core component search in MTSF.

Input: An MTSF $\mathcal{M}$, a vertex u, and a time interval $[ts, te]$
Output: A temporal k-core component $C^k_{[ts,te]}(u)$

```
C^k_[ts,te](u) ← {u}
Initialize the search frontier with the edges of u in M
while the search frontier is not empty do
    (v, v', [ts', te']) ← the next edge in the search frontier
    if [ts', te'] ⊑ [ts, te] then
        Add v' to C^k_[ts,te](u)
        Update the search frontier with the edges of v' in M
return C^k_[ts,te](u)
```

cores whose TTIs are also subintervals of $[ts, te]$ along the lineage relation until such a distinct core does not exist, and the time zone of the last distinct core certainly contains $[ts, te]$.

As illustrated in Figure 6, the lookup structure is logically equivalent to the lineage graph. Each node is an index entry comprised of 1) the TTI of the corresponding distinct core as id, 2) a pointer to the MTSF of the lineage chain that contains the distinct core, and 3) an ordered list of TTIs of distinct cores with lineage to the distinct core, each of which is linked to another node has the same id. Moreover, for the minimal distinct cores without incoming edge in the lineage graph, we use an additional node that only has the ordered TTI list to link them, so that this special node is actually an entrance of the lookup structure. The pseudo code of lookup operation is presented in Algorithm 4.

Example 9. *As illustrated in Figure 6, the lookup path for a given time interval* $[2, 6]$ *is highlighted by bold lines. It starts at the entrance, which contains two minimal TTIs* $[3, 3]$ *and* $[4, 4]$ *in its ordered list. By a binary search, the TTI* $[3, 3] \sqsubset [2, 6]$ *is chosen. Note that, the TTI* $[4, 4] \sqsubset [2, 6]$ *is also qualified, and the lookup paths from* $[4, 4]$ *will reach the same destination with respect to Theorem 4. Then, in the entry with the id* $[3, 3]$*, the TTI* $[3, 4] \sqsubset [2, 6]$ *is chosen. Such a process stops when the entry with the id* $[2, 6]$ *is reached, because none of the TTIs in its ordered list is a subinterval of* $[2, 6]$*. Lastly, the MTSF pointer (marked by green color) in the last entry is returned.*

Algorithm 4 converges within at most d iterations, where d denotes the depth (namely, number of layers) of lineage graph. In each iteration, the binary search takes $O(\log p)$ time, where p denotes the number of LTIs in a time zone (so that the length of TTI list in an entry is $p + 1$). Thus, the time complexity of Algorithm 4 is upper bounded by $O(d \log \max p)$. Since p tends to be small with respect to the size of lineage graph, Algorithm 4 retrieves the matched MTSF in at most linear time to d.

3.5 Time-Optimal TCCS Processing

For a specific k, the complete EF-Index is composed of a lookup structure pointing to a set of MTSFs. Accordingly, the online processing of TCCS mainly has two steps, namely, retrieving an MTSF from the index and searching for the component in the MTSF, as illustrated in Figure 4. In this subsection, we present the concrete principle and algorithm for searching MTSF.

Different from a static MTF, the connected vertices of an MTSF are not connected for all time, and some vertices in an MTSF may even not exist in a number of distinct cores from the head of chain. For example, as illustrated in Figure 5, although all vertices are connected in the final MTSF, v_{10} does not exist and the other vertices are divided into two connected components in the distinct core with the TTI $[1, 4]$. Thus, returning the connected component that contains the given vertex in a retrieved MTSF could be incorrect.

Instead, we can find the result component from a retrieved MTSF with respect to the following observation on MTSF.

Theorem 5 (Constrained Connectivity). *For a distinct core* $\mathbb{T}^k$ *with the TTI* $[ts, te]$ *on a lineage chain* $\mathbb{C}^k$*, two vertices are connected in* $\mathbb{T}^k$*, if and only if there exists a path between them in the MTSF of* $\mathbb{C}^k$ *and each edge label on the path is a subinterval of* $[ts, te]$*.*

Theorem 5 implies that, the connectivity between two vertices in any $\mathbb{T}^k$ can still be determined directly by the MTSF of lineage chain containing $\mathbb{T}^k$, without spending extra time on decompressing the MTSF to the MSF of $\mathbb{T}^k$. Specifically, we conduct a label-constrained depth-first search from the given vertex in the retrieved MTSF. The pseudo code is given in Algorithm 5. The time complexity is $O(\sum_{v \in C^k_{[ts,te]}(u)} \deg_{\mathcal{M}}(v))$, where $\deg_{\mathcal{M}}(v)$ is the degree of vertex v in the MTSF $\mathcal{M}$. Obviously, the search is guaranteed to be optimal, since the size of search space is the same as MSF.

Example 10. *To find the component containing the vertex* v_8 *in a distinct core with the TTI* $[1, 3]$*, we search the MTSF illustrated in Figure 5. For the two neighbors of* v_8*,* v_{10} *is excluded because* $[1, 5] \not\sqsubseteq [1, 3]$*, and* v_6 *is included because* $[1, 3] \sqsubseteq [1, 3]$*. Then, for the two neighbors of* v_6*,* v_4 *is excluded because* $[1, 6] \not\sqsubseteq [1, 3]$*, and* v_5 *is included because* $[3, 3] \sqsubseteq [1, 3]$*. Lastly, the component* $\{v_5, v_6, v_7, v_8, v_9\}$ *will be found, which can be verified in Figure 2a.*

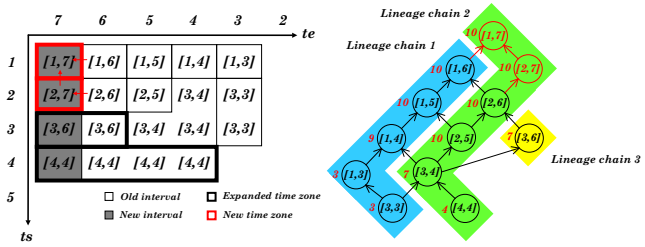
4 INDEX MAINTENANCE

4.1 Dynamic Graph and Maintenance Task

In real-world applications, a temporal graph tends to be updated dynamically over time. A basic assumption of dynamic graph is that, a set of edges with an incremental timestamp (which represents the current time) are added into the graph for each update. We do not consider deleting edges or inserting edges with elapsed timestamps because the history should not be changed.

Therefore, for a temporal graph $\mathcal{G}$, we represent an update to it as a finite set of new edges with the timestamp $n + 1$, where n is the latest timestamp in $\mathcal{G}$ and the edges could connect the vertices in $\mathcal{V}$ or new vertices. The updated temporal graph is denoted by $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$, where $\mathcal{V}'$ is the union of $\mathcal{V}$ and new vertices and $\mathcal{E}'$ is the union of $\mathcal{E}$ and new edges. Then, for an EF-Index of $\mathcal{G}$ with respect to a specific k and a given update, we address the problem of maintaining the up-to-date EF-Index of $\mathcal{G}'$ incrementally.

As illustrated in Figure 4, the pipeline of EF-Index construction mainly involves the generation of four objects, namely, lineage graph, lineage chain cover, MTSF and lookup structure. In particular, the lookup structure can be trivially maintained with respect to the updated lineage graph and MTSF. Thus, we present the methods of updating lineage graph, lineage chain cover and MTSF in the rest subsections respectively. Note that, this maintenance mechanism



(a) Updated time zones. (b) Updated lineage chain cover.

Figure 7: An example of updating the lineage graph of $\mathcal{G}$ for adding edges $\{(v_9, v_{10}, 7), (v_{10}, v_{11}, 7)\}$ to $\mathcal{G}$.

can be easily extended to handle a batch of new edges with multiple timestamps like $n+1, n+2, \cdots$ together.

4.2 Maintaining Lineage Graph

From the perspective of topology, a set of new edges may lead to the evolution of some distinct cores in $\mathcal{G}$, which further changes the lineage graph composed of distinct cores and their lineage relation. However, it is complicated to trace which distinct cores will evolve and how they will evolve. Thus, we still address the maintenance of lineage graph from the perspective of time.

The new edges added at time $n+1$ give birth to $n+1$ new time intervals like $[ts, n+1]$ $(1 \leqslant ts \leqslant n+1)$. For example, as illustrated in Figure 7a, the new time intervals marked by grey color will appear, if new edges like $(v_9, v_{10}, 7)$ and $(v_{10}, v_{11}, 7)$ are added to the graph in Figure 2a. Note that, not each new time interval can induce a temporal k-core, such as $[5,7]$, $[6,7]$ and $[7,7]$, thereby being omitted. Thus, the problem turns to identifying new time zones and getting their distinct cores, and then capturing the new lineage relation.

4.2.1 Time Zone Re-identification. Running the OTCD* algorithm (line 1 in Algorithm 1) on the updated temporal graph is a straightforward but not efficient way of re-identifying time zones. To accelerate the re-identification, we propose an incremental algorithm with respect to the following observation.

Theorem 6 (Zone Isolation). *For an existing time zone of $\mathcal{G}$, it can only be expanded for updating $\mathcal{G}$ to $\mathcal{G}'$ if it is adjacent to new time intervals, which means the maximum end time of its LTIs is n. The time zones isolated from new time intervals will never change.*

Intuitively, Theorem 6 implies that, the new time interval $[ts, n+1]$ is either in a new time zone or merged into an existing adjacent time zone. Specifically, if the end time of its TTI is $n+1$, it belongs to a new time zone, which means adding new edges results in the evolution of distinct cores of adjacent time zones. Otherwise, the distinct core of adjacent time zone cannot evolve, and thereby it is merged into the adjacent time zone.

Example 11. *As illustrated in Figure 7a, there are four valid new time intervals $[1,7]$, $[2,7]$, $[3,7]$ and $[4,7]$ marked by grey color. The first two with TTIs $[1,7]$ and $[2,7]$ induce new temporal k-cores, since none TTI of old temporal k-cores can have the end time 7. Thus, they form new time zones marked by red frame. The last two with TTIs $[3,6]$ and $[4,4]$ definitely induce the identical temporal k-cores to that of old time interval, and thus are merged into the adjacent old time zones marked by black frame according to Theorem 6, respectively. As a result, there are two new nodes marked by red color in the updated lineage graph illustrated in Figure 7b.*

Algorithm 6: Lineage graph maintenance.

Input: The updated graph $\mathcal{G}'$ and the lineage graph $\mathbf{G}^k$
Output: The updated lineage graph of $\mathcal{G}'$

```
1  ts ← 1
2  T^k ← TCD ([ts, n+1], k, G')
3  while ts ⩽ n+1 do // Identify new time zones
4      [ts', te'] ← TTI of T^k
5      if te' < n+1 then // Global Pruning
6          merge rest new intervals to adjacent time zones
7          break
8      if ts < ts' then // Local Pruning
9          ts ← ts'
10     Add a distinct core T^k to G^k
11     if T' is not empty then
12         Add a lineage (T^k, T') in G^k
13     [ts'', te''] ← TTI of T^k_[ts,n] got from lookup structure
14     Add a lineage from the distinct core whose TTI is
         [ts'', te''] to T^k in G^k
15     ts ← ts+1, T' ← T^k
16     T^k_[ts,n+1] ← TCD([ts, n+1], k, T^k)
17 return G^k
```

Thus, we can run a revised OTCD* algorithm only for the new time intervals. Specifically, we enumerate the new time intervals in ascending order of ts, and induce their temporal k-cores by TCD operation decrementally. Moreover, two pruning rules are proposed to optimize the algorithm. The first one called **local pruning** is essentially equivalent to the "rectangle pruning" of OTCD*, and the second one called **global pruning** is even more aggressive for stopping the algorithm immediately with respect to Theorem 6.

Local pruning. During the enumeration, if the TTI $[ts', te']$ of the current interval $[ts, n+1]$ satisfies $ts < ts'$ and $te' = n+1$, the following intervals $[ts+1, n+1], \cdots, [ts', n+1]$ are skipped, because they all have the same TTI $[ts', te']$.

Global pruning. During the enumeration, if the TTI $[ts', te']$ of the current interval $[ts, n+1]$ satisfies $te' < n+1$, all following intervals are skipped, since each of them can either be merged into the adjacent old time zone or induce none temporal k-core.

The above pruning rules guarantee that each new distinct core is induced exactly once and the existing distinct cores are not needed to be induced again except the one that triggers global pruning.

4.2.2 Lineage Regeneration. According to Definition 7, the lineage to a time zone is determined by its CTIs. Thus, we categorize the new lineage into the following three types, and derive the CTIs for each type respectively. 1) The new lineage to new time zones. A new time zone has at most one LTI $[ts, n+1]$, and thus can have

one valid CTI, namely, $[ts-1, n+1]$ if $ts > 1$ or none if $ts = 1$. 2) The new lineage to expanded time zones. Interestingly, this kind of lineage does not exist at all, because the expanded time zones have no valid new CTI. 3) The new lineage to other time zones. Obviously, only the distinct cores of time zones that have adjacent new time zones will evolve to new distinct cores. Thereby, for each new time zone $\mathbb{Z}_j^k$ with a TTI $[ts', te']$, we only need to figure out which time zone $\mathbb{Z}_i^k$ the adjacent interval $[ts', te'-1]$ belongs to due to Property 3, and generate a new lineage $(\mathbb{Z}_i^k, \mathbb{Z}_j^k)$. In summary, we have the following formal inference.

Theorem 7 (Lineage Backtracking). *For each new distinct core $\mathbb{T}_i^k$ with the TTI $[ts, n+1]$ and LTI $[ts', n+1]$, there exists the new lineage $(\mathbb{T}_l^k, \mathbb{T}_i^k)$ always and $(\mathbb{T}_i^k, \mathbb{T}_j^k)$ if $ts' > 1$, where the TTI of $\mathbb{T}_l^k$ is the same as $\mathcal{T}_{[ts,n]}^k$ and the TTI of $\mathbb{T}_j^k$ is $[ts'-1, n+1]$.*

Example 12. *As illustrated in Figure 7a, the new time zone with the TTI $[1, 7]$ has two new lineage to the adjacent old time zone with the TTI $[1, 6]$ and the adjacent new time zone with the TTI $[2, 7]$ respectively. While, the new time zone with the TTI $[2, 7]$ have only one new lineage to the adjacent old time zone with the TTI $[2, 6]$. Moreover, the CTIs of the expanded time zones with the TTIs $[3, 6]$ and $[4, 4]$ do not change ($[2, 6]$ and $[3, 4]$ respectively), so that there is no new lineage to them. Finally, there are three new edges marked by red color in the updated lineage graph illustrated in Figure 7b.*

Lastly, the complete procedure of updating the lineage graph is presented in Algorithm 6. In fact, we combine the time zone re-identification and the lineage regeneration in one pass according to Theorem 7, thereby reducing redundant computation mostly.

4.3 Maintaining Lineage Chain Cover and MTSF

After updating the lineage graph, we propose a heuristic method to update the lineage chain cover incrementally. It not only incurs few computation, and also can reuse the intermediate results (namely, differential edge sets) of Algorithm 6 to facilitate the construction of new MTSF. Although the updated cover is not guaranteed to be the minimum, we can use a daemon process to re-partition the chains when there is few search request in practice.

Specifically, we create a new chain with all new distinct cores, since they are naturally chained according to Theorem 7. Moreover, since the head of new chain certainly has lineage to an old distinct core, we check whether the old one is the tail of a chain in the cover, and if so merge these two chains to reduce the size of updated chain cover.

Example 13. *As illustrated in Figure 7b, the new distinct cores marked by red color can form a new chain through the new lineage also marked by red color between them. The cost of new chain is 10. Moreover, since the new distinct core with the TTI $[2, 7]$ has lineage to the old distinct core with the TTI $[2, 6]$, which is the tail of old chain marked by green stripe, we merge the new chain to the old chain. As a result, the final increased cost is reduced to $10 - 10 = 0$.*

Then, we only need to create or update one MTSF, since the updated lineage chain cover only contains one new or updated chain. Thus, we build a new MTSF of the new chain or incrementally extend the MTSF of an old chain by only using the `SingleChain()` function but not the full Algorithm 3. As mentioned above, the computation of differential edge sets (line 5 in Algorithm 3) between adjacent distinct cores in the new chain can be obtained by line 16 in Algorithm 6 without notable extra cost. Thus, the time cost of maintaining MTSF is only bounded by the number of edges of the largest new distinct core.

Lastly, to update the lookup structure, we simply add entries pointing to the new or updated MTSF for the new distinct cores, and then add links between entries with respect to the new lineage.

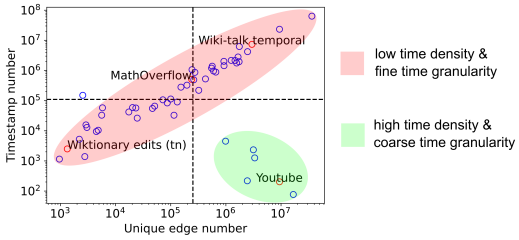


Figure 8: The distribution of temporal graphs from SNAP and KONECT projects. The dashed lines remark medians.

Table 2: The statistics of test temporal graphs.

$\mathcal{G}$	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	n	days	$\lvert\mathcal{E}\rvert/n$	days/n
Wiktionary	882	2.7K	2.5K	4.9K	1	2
Math-day	24.7K	390.0K	2.4K	2.4K	166	1
Wikitalk-day	1.1M	6.1M	2.2K	2.3K	2.7K	1
Youtube	3.2M	9.4M	203	225	46.2K	1
Math-day-2K	21.4K	343.3K	2.0K	2.0K	172	1
Math-5K	612	5.0K	5.0K	36	1	0.007

5 EXPERIMENT

In this section, we conduct experiments to evaluate EF-Index on a Windows machine with 2.20GHz CPU and 64GB RAM. The algorithms are implemented with C++ Standard Template Library.

5.1 Dataset

Firstly, we conduct an empirical study on 52 temporal graph datasets from the widely-used SNAP [20] and KONECT [18] projects, and show the distribution with respect to both unique edge number and timestamp number in Figure 8. Most temporal graphs in the red area have **time densities** almost equal to 1, namely, the average number of unique edges for each timestamp. However, due to the coarser **time granularity** (day or year but not second), a few of temporal graphs in the green area have extraordinary time densities. Based on the observations, we choose representative datasets (red cycles) from each quadrant, and compose diverse temporal graphs for experiments, the statistics of which are given in Table 2.

Note that, the temporal graphs like Math and Wikitalk have millions of timestamps in seconds, and thus building EF-Index or PHC-Index on the original graphs is not only too costly and also unreasonable in practice. If the user expects communities that emerge during several months, the time granularity would better

Table 3: The space cost (in MByte) and statistics of EF-Index.

$\mathcal{G}$, k_{max}	EF-Index	$k = \lceil$% of $k_{max}\rceil$				
		90%	80%	70%	60%	50%
Wiktionary 8	$\|\mathbf{T}^k\|$	3.3K	29.3K	83.9K	183.6K	410.4K
	$\|\mathbf{C}^k\|$	48	173	289	455	714
	space	0.4	2.9	8.1	17.5	39
Math-day 78	$\|\mathbf{T}^k\|$	115.7K	357.9K	616.1K	975.6K	1.38M
	$\|\mathbf{C}^k\|$	252	399	699	1017	1306
	space	13.5	38.3	66.7	106.8	154.2
Wikitalk-day 124	$\|\mathbf{T}^k\|$	0.2M	0.4M	0.6M	0.9M	1.1M
	$\|\mathbf{C}^k\|$	192	372	545	676	800
	space	27.1	62.7	105.2	154.5	216.1
Youtube 88	$\|\mathbf{T}^k\|$	0.2K	1.6K	3.7K	6.3K	9.2K
	$\|\mathbf{C}^k\|$	9	29	47	66	93
	space	0.3	1.7	5.5	14.5	33.4

be day, week or month, since the indexing overhead increases sub-quadratically to the number of timestamps. For example, we compose Math-day and Wikitalk-day by grouping timestamps by day. In contrast, if the user expects communities that emerge during a few of seconds or minutes, we can divide the history of temporal graphs into a sequence of short periods with hundreds or thousands of original timestamps in seconds, so that the index of each period can be built efficiently. For example, Math-5K is a projected graph of Math containing only the first five thousands timestamps.

5.2 Evaluation of Index Construction

We report the space and time costs of building EF-Index in this subsection. Since each graph has a different maximum value of k (denoted by k_{max}), we choose the ceilings of 90%, 80%, 70%, 60% and 50% of k_{max} as the values of k to build indexes for each graph (the first four) respectively.

Firstly, the space costs and related statistics are shown in Table 3. We have the following major observations. 1) The compression ratio of EF-Index, which can be roughly evaluated by $2|\mathbf{C}^k|/n(n+1)$, is mostly on the order of 10^{-4} or 10^{-5} for vertex and even better for edge. Especially for the graphs with thousands of timestamps, there are millions of temporal k-cores that are finally compressed to hundreds of MTSFs. Even the the compression ratio of distinct cores, namely, $|\mathbf{C}^k|/|\mathbf{T}^k|$ is averagely on the order of 10^{-3}. 2) Due to the significant semantic compression, the raw space costs we report are between 0.3MB and 216.1MB for the index of a specific k. The fewer chains and the smaller graphs, the less space. Moreover, the existing syntactic compression algorithms can still be applied to further reduce the space cost. 3) With the decrease of cohesiveness k, the degree of discretization of temporal k-core evolution described by Theorem 1 decreases, and thus the number of distinct cores $|\mathbf{T}^k|$ increases without exceeding $n(n+1)/2$, which further results in the increase of lineage chain cover size and index space.

Moreover, the time costs are shown in Figure 9. In particular, we compare two strategies of MTSF construction named All-in-One (Algorithm 3) and One-by-One (which computes the differential edge sets for each chain independently). We have the following observations. 1) Due to time-topology isomorphic computation, building the lineage graph (Algorithm 1) is feasible even for large-scale graphs like Youtube, though it is still the most time-consuming part of indexing. 2) Computing the minimum lineage chain cover (Algorithm 2) takes only a little indexing time. 3) Like space cost, the time cost of Algorithm 3 is correlated to the size of chain cover and the scale of graph. Wiktionary is the smallest graph and costs only 10^0-10^2 secs. Math-day is larger, and Youtube is even larger but has less chains, so that they cost 10^2-10^3 secs. Wikitalk-day is large and also has a lot of chains, so that it costs 10^3-10^4 secs. 4) As expected, All-in-One outperforms One-by-One remarkably.

Table 4: Basic information of forty test instances of TCCS.

$\mathcal{G}$, u, ts	Varied cohesiveness k				Varied time span $te - ts$			
	te	k	$\|C^k_{[ts,te]}(u)\|$	id	k	te	$\|C^k_{[ts,te]}(u)\|$	id
Wiktionary 1 1	2000	2	245	1	2	500	60	6
		3	116	2		1000	148	7
		4	81	3		1500	192	8
		5	65	4		2000	245	9
		6	39	5		2500	291	10
Math-day 1 1	500	27	653	11	25	100	76	16
		32	534	12		200	261	17
		37	439	13		300	456	18
		42	359	14		400	584	19
		47	220	15		500	715	20
Wikitalk-day 45 1	500	10	162	21	5	100	23	26
		12	140	22		200	45	27
		14	112	23		300	138	28
		16	96	24		400	235	29
		18	77	25		500	369	30
Youtube 4 1	20	30	8045	31	30	10	7783	36
		35	5442	32		15	7873	37
		40	3295	33		20	8045	38
		45	2183	34		25	8245	39
		50	855	35		30	8437	40

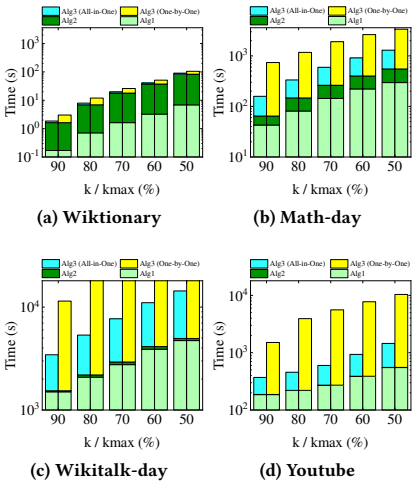
5.3 Evaluation of Index Maintenance

To simulate the real-world graph updates, we set up experiments on Math-5K and Math-day-2K with different graph scales and time densities respectively. Specifically, for each of the next 100 timestamps in the original Math (or Math-day), we derive the set of edges with this timestamp as an update of Math-5K (or Math-day-2K). Then, we keep maintaining the EF-Index incrementally for the updates.

The time cost of each index maintenance and the total number of time zones in each updated graph are shown in Figure 11. We have the following observations. 1) The maximum time cost of maintaining the EF-Index is less than 0.4 sec for tiny Math-5K and 13 sec for large Math-day-2K, while their updating frequencies are 1 sec and 1 day respectively. Thus, EF-Index can indeed be maintained in real-time if a balance between the graph scale and the update frequency is achieved. 2) Each index maintenance has a basic cost of updating lineage graph (Algorithm 6). For Math-5K, the cost is about 0.05 sec. 3) There is no new time zone after most updates of Math-5K and thereby no more time cost other than updating lineage graph, because the time density is nearly one edge per timestamp in Math. In contrast, since the time granularity is day but not sec in Math-day-2K, the high time density leads to constant changes of time zone, so that the deviation of maintenance time costs is small.

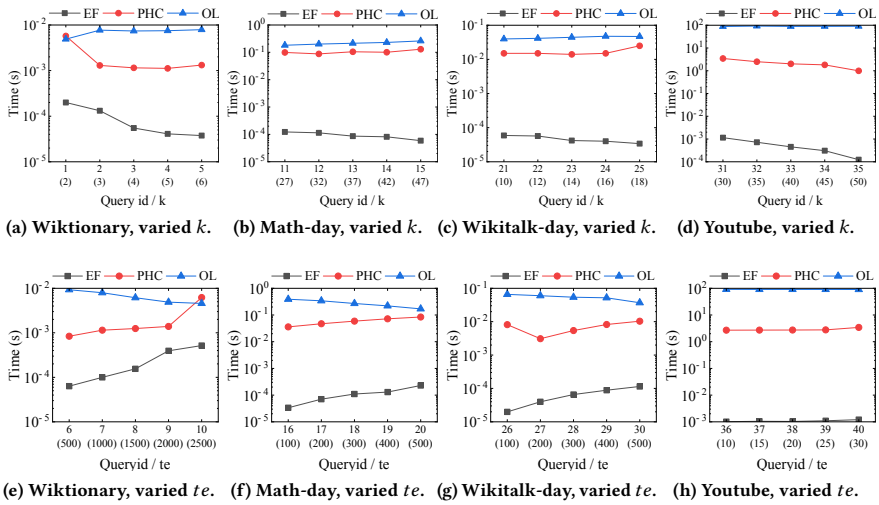
5.4 Evaluation of Search Efficiency

We compose the forty TCCS instances shown in Table 4 for testing search efficiency, which combines different graphs, cohesiveness



(a) Wiktionary (b) Math-day (c) Wikitalk-day (d) Youtube

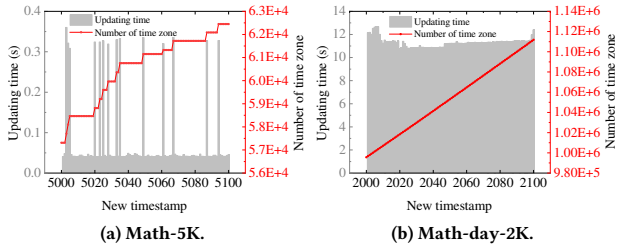
Figure 9: Time costs of building EF-Index.



(a) Wiktionary, varied k. (b) Math-day, varied k. (c) Wikitalk-day, varied k. (d) Youtube, varied k.

(e) Wiktionary, varied te. (f) Math-day, varied te. (g) Wikitalk-day, varied te. (h) Youtube, varied te.

Figure 10: The response time of TCCS algorithms on forty test instances.



(a) Math-5K. (b) Math-day-2K.

Figure 11: Time costs of maintaining EF-Index.

and time spans. The sizes of their result communities are also provided. For each query, we use three algorithms to process it respectively, including our **EF-Index** based search (Algorithms 4 and 5), **PHC-Index** based search (the latest indexing method [45]), and **Online** search implemented by using TCD operation (the latest online querying method [44]) to get the temporal k-core for search.

Figure 10 shows the response time of three algorithms on the forty TCCS instances. We have the following observations. 1) EF-Index is at least 10^1-10^3 times faster than the state-of-the-art PHC-Index on all instances, and PHC-Index is generally faster than Online. Because PHC-Index can only assist to filter the vertices not in results, and EF-Index directly provides the minimum set of vertices needed to be traversed. Even for the large-scale datasets like Youtube, EF-Index can still process each query only in less than 10^{-3} sec. 2) The response time of EF-Index decreases monotonically with increasing k, since the time cost of MTSF search (Algorithm 5) is mainly determined by the size of results, which correlates to k inversely. 3) The response time of EF-Index increases monotonically with increasing te, since the result component grows gradually with expanding time interval. Note that, the lookup time (Algorithms 4) that varies with respect to the size of lineage graph has been included in the response time of EF-Index.

6 RELATED WORK

There are typically two categories of k-core studies on temporal graphs, with respect to the **direct** or **indirect** temporal condition. The direct condition is normally a specific time interval, and thus the related studies focus on efficiency or scalability of general query processing, such as those [12, 41, 44, 45, 47] mentioned in Section 1. The indirect condition involves a variety of time-relevant metrics, such as interaction frequency [1, 41], persistence [21], burstiness [4, 31], periodicity [32], continuity [23] and reliability [37], and thus the related studies focus on devising dedicated solutions.

To the best knowledge we have, our work is the first study to address component search in the first category (while [27] belongs to the other). In addition, many other works [3, 6–8, 10, 15, 17, 25, 35, 36, 43, 46] on non-temporal k-core or other kinds of community also provide valuable insights.

7 CONCLUSION

To address the TCCS problem, we design a novel EF-Index based on the lineage relation in temporal k-core evolution, and propose a time-topology isomorphic computation paradigm to obtain lineage relation efficiently. EF-Index guarantees the same size of search space as directly searching an MSF for a given TCCS intance and thereby is time-optimal. Meanwhile, EF-Index only stores a minimum set of compactest MTSFs and thereby is near compactest. Moreover, EF-Index can be maintained incrementally in real time following dynamical graph updates.

ACKNOWLEDGMENTS

This work was supported by the Research Fund of the National Natural Science Foundation of China (No. 61202036, 62272353, and 62276193), the Guangzhou Key Laboratory of Big Data and Intelligent Education (No. 201905010009), and the Research Grants Council of Hong Kong, China (No. 14205520).

REFERENCES

[1] Wen Bai, Yadi Chen, and Di Wu. 2020. Efficient temporal core maintenance of massive graphs. *Information Sciences* 513 (2020), 324–340. https://doi.org/10.1016/j.ins.2019.11.003

[2] Xinwei Cai, Xiangyu Ke, Kai Wang, Lu Chen, Tianming Zhang, Qing Liu, and Yunjun Gao. 2023. Efficient Temporal Butterfly Counting and Enumeration on Temporal Bipartite Graphs. arXiv:2306.00893 [cs.DS]

[3] Yankai Chen, Jie Zhang, Yixiang Fang, Xin Cao, and Irwin King. 2021. Efficient community search over large directed graphs: An augmented index-based approach. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 3544–3550. https://doi.org/10.24963/ijcai.2020/490

[4] 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. https://doi.org/10.14778/3358701.3358704

[5] Manuel Cáceres, Massimo Cairo, Brendan Mumey, Romeo Rizzi, and Alexandru I. Tomescu. 2022. Sparsifying, Shrinking and Splicing for Minimum Path Cover in Parameterized Linear Time. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 359–376. https://doi.org/10.1137/1.9781611977073.18

[6] Yixiang Fang, Reynold Cheng, Yankai Chen, Siqiang Luo, and Jiafeng Hu. 2017. Effective and efficient attributed community search. *The VLDB Journal* 26, 6 (2017), 803–828. https://doi.org/10.1007/s00778-017-0482-5

[7] Yixiang Fang, Reynold Cheng, Xiaodong Li, Siqiang Luo, and Jiafeng Hu. 2017. Effective community search over large spatial graphs. *Proceedings of the VLDB Endowment* 10, 6 (2017), 709–720. https://doi.org/10.14778/3055330.3055337

[8] 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. https://doi.org/10.1007/s00778-019-00556-x

[9] Yixiang Fang, Zhongran Wang, Reynold Cheng, Hongzhi Wang, and Jiafeng Hu. 2018. Effective and efficient community search over large directed graphs. *IEEE Transactions on Knowledge and Data Engineering* 31, 11 (2018), 2093–2107.

[10] 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. https://doi.org/10.14778/3380750.3380756

[11] Michael Fredman and Michael Saks. 1989. The cell probe complexity of dynamic data structures. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*. Association for Computing Machinery, 345–354. https://doi.org/10.1145/73007.73040

[12] 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*. Association for Computing Machinery, 107–116. https://doi.org/10.1145/3269206.3271767

[13] Zhongqiang Gao, Chuanqi Cheng, Yanwei Yu, Lei Cao, Chao Huang, and Junyu Dong. 2022. Scalable Motif Counting for Large-scale Temporal Graphs. In *Proceedings of the 38th IEEE International Conference on Data Engineering (ICDE)*. IEEE, 2656–2668. https://doi.org/10.1109/ICDE53745.2022.00244

[14] John E. Hopcroft and Richard M. Karp. 1973. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Comput.* 2, 4 (1973), 225–231. https://doi.org/10.1137/0202019

[15] 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*. Association for Computing Machinery, 1311–1322. https://doi.org/10.1145/2588555.2610495

[16] Xuanwen Huang, Yang Yang, Yang Wang, Chunping Wang, Zhisheng Zhang, Jiarong Xu, Lei Chen, and Michalis Vazirgiannis. 2022. Dgraph: A large-scale financial dataset for graph anomaly detection. *Advances in Neural Information Processing Systems* 35 (2022), 22765–22777.

[17] Md Saiful Islam, Mohammed Eunus Ali, Yong-Bin Kang, Timos Sellis, Farhana M Choudhury, and Shamik Roy. 2022. Keyword aware influential community search in large attributed graphs. *Information Systems* 104 (2022), 101914. https://doi.org/10.1016/j.is.2021.101914

[18] Jérôme Kunegis. 2013. Konect: the koblenz network collection. In *Proceedings of the 22nd international conference on world wide web*. Association for Computing Machinery, 1343–1350. https://doi.org/10.1145/2487788.2488173

[19] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. Association for Computing Machinery, New York, NY, USA, 177–187. https://doi.org/10.1145/1081870.1081893

[20] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[21] 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. https://doi.org/10.1109/ICDE.2018.00077

[22] Tianpeng Li, Wenjun Wang, Pengfei Jiao, Yinghui Wang, Ruomeng Ding, Huaming Wu, Lin Pan, and Di Jin. 2022. Exploring Temporal Community Structure via Network Embedding. *IEEE Transactions on Cybernetics* (2022), 1–13. https://doi.org/10.1109/TCYB.2022.3168343

[23] 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, 5 (2021), 1483–1509. https://doi.org/10.1007/s11280-021-00917-z

[24] Zhe Lin, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Zhihong Tian. 2021. Hierarchical core maintenance on large dynamic graphs. *Proceedings of the VLDB Endowment* 14, 5 (2021), 757–770. https://doi.org/10.14778/3446095.3446099

[25] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks VS Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2020. Efficient algorithms for densest subgraph discovery on large directed graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 1051–1066. https://doi.org/10.1145/3318464.3389697

[26] Naoki Masuda and Renaud Lambiotte. 2020. *A Guide to Temporal Networks*. World Scientific Publishing Europe Ltd.

[27] Lutz Oettershagen, Athanasios L. Konstantinidis, and Giuseppe F. Italiano. 2023. Temporal Network Core Decomposition and Community Search. arXiv:2309.11843 [cs.SI]

[28] Lutz Oettershagen, Nils M. Kriege, Claude Jordan, and Petra Mutze. 2023. A Temporal Graphlet Kernel For Classifying Dissemination in Evolving Networks. In *Proceedings of the 2023 SIAM International Conference on Data Mining (SDM)*. Society for Industrial and Applied Mathematics, 19–27. https://doi.org/10.1137/1.9781611977653.ch3

[29] Lutz Oettershagen, Petra Mutzel, and Nils M. Kriege. 2022. Temporal Walk Centrality: Ranking Nodes in Evolving Networks. In *Proceedings of the ACM Web Conference 2022*. Association for Computing Machinery, 1640–1650. https://doi.org/10.1145/3485447.3512210

[30] Noujan Pashanasangi and C. Seshadhri. 2021. Faster and Generalized Temporal Triangle Counting, via Degeneracy Ordering. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, 1319–1328. https://doi.org/10.1145/3447548.3467374

[31] Hongchao Qin, Rong-Hua Li, Ye Yuan, Guoren Wang, Lu Qin, and Zhiwei Zhang. 2022. Mining Bursting Core in Large Temporal Graphs. *Proceedings of the VLDB Endowment* 15, 13 (2022), 3911–3923. https://doi.org/10.14778/3565838.3565845

[32] 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. https://doi.org/10.1007/978-3-031-25158-0_38

[33] Ursula Redmond and Pádraig Cunningham. 2013. Temporal Subgraph Isomorphism. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE, 1451–1452. https://doi.org/10.1145/2492517.2492586

[34] Uriel Singer, Ido Guy, and Kira Radinsky. 2019. Node Embedding over Temporal Graphs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 4605–4612. https://doi.org/10.24963/ijcai.2019/640

[35] 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*. Association for Computing Machinery, 939–948. https://doi.org/10.1145/1835804.1835923

[36] Renjie Sun, Chen Chen, Xiaoyang Wang, Ying Zhang, and Xun Wang. 2020. Stable community detection in signed social networks. *IEEE Transactions on Knowledge and Data Engineering* 34, 10 (2020), 5051–5055. https://doi.org/10.1109/TKDE.2020.3047224

[37] Yifu Tang, Jianxin Li, Nur Al Hasan Haldar, Ziyu Guan, Jiajie Xu, and Chengfei Liu. 2022. Reliable Community Search in Dynamic Networks. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2826–2838. https://doi.org/10.14778/3551793.3551834

[38] Kai Wang, Wenjie Zhang, Xuemin Lin, Ying Zhang, and Shunyang Li. 2022. Discovering hierarchy of bipartite graphs with cohesive subgraphs. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2291–2305.

[39] Dong Wen, Bohua Yang, Ying Zhang, Lu Qin, Dawei Cheng, and Wenjie Zhang. 2021. Span-Reachability Querying in Large Temporal Graphs. *The VLDB Journal* 31, 4 (nov 2021), 629–647. https://doi.org/10.1007/s00778-021-00715-z

[40] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. 2014. Path Problems in Temporal Graphs. *Proceedings of the VLDB Endowment* 7, 9 (may 2014), 721–732. https://doi.org/10.14778/2732939.2732945

[41] 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. https://doi.org/10.1109/BigData.2015.7363809

[42] 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.

[43] Guotong Xue, Ming Zhong, Tieyun Qian, and Jianxin Li. 2024. PSA-GNN: An augmented GNN framework with priori subgraph knowledge. *Neural Networks* 173 (2024), 106155. https://doi.org/10.1016/j.neunet.2024.106155

[44] Junyong Yang, Ming Zhong, Yuanyuan Zhu, Tieyun Qian, Mengchi Liu, and Jeffrey Xu Yu. 2023. Scalable Time-Range k-Core Query on Temporal Graphs. *Proceedings of the VLDB Endowment* 16, 5 (2023), 1168–1180. https://doi.org/10.14778/3579075.3579089

[45] Michael Yu, Dong Wen, Lu Qin, Ying Zhang, Wenjie Zhang, and Xuemin Lin. 2021. On querying historical k-cores. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2033–2045. https://doi.org/10.14778/3476249.3476260

[46] Chen Zhang, Fan Zhang, Wenjie Zhang, Boge Liu, Ying Zhang, Lu Qin, and Xuemin Lin. 2020. Exploring finer granularity within the cores: Efficient (k, p)-core computation. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 181–192. https://doi.org/10.1109/ICDE48307.2020.00023

[47] Ming Zhong, Junyong Yang, Yuanyuan Zhu, Tieyun Qian, Mengchi Liu, and Jeffrey Xu Yu. 2023. A Unified and Scalable Algorithm Framework of User-Defined Temporal $(k, \mathcal{X})$-Core Query. To appear in IEEE Transactions on Knowledge and Data Engineering. arXiv:2309.00361 [cs.DB]