Efficient Index for Temporal Core Queries over Bipartite Graphs

Anxin Tian
HKUST
atian@connect.ust.hk

Alexander Zhou
HKUST
atzhou@cse.ust.hk

Yue Wang*
Shenzhen Institute of Computing Sciences
yuewang@sics.ac.cn

Xun Jian*
HKUST
xjian@connect.ust.hk

Lei Chen
HKUST & HKUST(GZ)
leichen@ust.hk

ABSTRACT

Many real-world binary relations can be modelled as bipartite graphs, which can be inherently temporal and each edge is associated with a timestamp. The (α, β)-core, a popular structure that requires minimum degrees over two layers of vertices, is useful for understanding the organisation of bipartite networks. However, the temporal property has rarely been considered in cohesive subgraph mining in bipartite graphs. This gap prevents the finding of time-sensitive (α, β)-cores in real-world applications. In this paper, we aim at finding (α, β)-cores within any time window over a temporal bipartite graph. To address this problem, we propose a novel DAG (Directed Acyclic Graph)-like hierarchy with qualified time windows to describe the temporal containment property of the (α, β)-core. Furthermore, we construct the superior-optimized index which significantly optimizes space complexity and guarantees efficient query performance. We also propose a maintenance approach that can efficiently update the index by removing stale information and incorporating newly inserted temporal edges. Extensive experiments are conducted on eight real-world graphs and the results show the effectiveness and efficiency of our indexes.

PVLDB Reference Format:
Anxin Tian, Alexander Zhou, Yue Wang, Xun Jian, and Lei Chen. Efficient Index for Temporal Core Queries over Bipartite Graphs. PVLDB, 17(11): 2813-2825, 2024.
doi:10.14778/3681954.3681965

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ExpCodeBase/tabc.

1 INTRODUCTION

In bipartite graphs, vertices are divided into two disjoint sets such that each edge connects one vertex from each set. Bipartite graphs always arise when we need to model relationships across two different types of entities. Many real-world bipartite graphs are inherently temporal, where each edge is associated with a timestamp or an interval representing its occurrence (e.g. user-page networks [5], customer-product networks [33], collaboration networks [6, 25], affiliation networks [53]). For simplicity and without loss of generality, we consider the scenario where each edge has a single timestamp rather than a time interval.

As a popular cohesive subgraph structure, the (α, β)-core [4, 9, 18, 30, 31, 50, 64] is defined as a maximal subgraph of the given bipartite graph G whose vertices in the upper layer have a degree of at least α and vertices in the lower layer have a degree of at least β. The (α, β)-core has been widely used in community detection and search. In real-world applications, it is often necessary to focus on a specific time window when considering interactions between entities within a temporal bipartite graph. Given a specific time window, we can obtain a graph snapshot by collecting edges with timestamps falling within this time window into a subgraph. We show a Flickr temporal bipartite graph in Figure 1 (a) and its snapshot S over the time period of $\langle 3, 21 \rangle$ in Figure 1 (b). The $(2, 4)$-core of S is marked. The upper-layer vertices represent Flickr users and the lower-layer vertices represent Flickr groups. Considering the $(2, 4)$-core in the time window $\langle 3, 21 \rangle$, it is a subgraph where each user is involved in at least two groups, and each group includes at least four users. In this context, users in the $(2, 4)$-core can be considered as potentially sharing preferences of groups, enabling group recommendations based on common interests. We call the (α, β)-core of a snapshot over a time window as the temporal (α, β)-core.

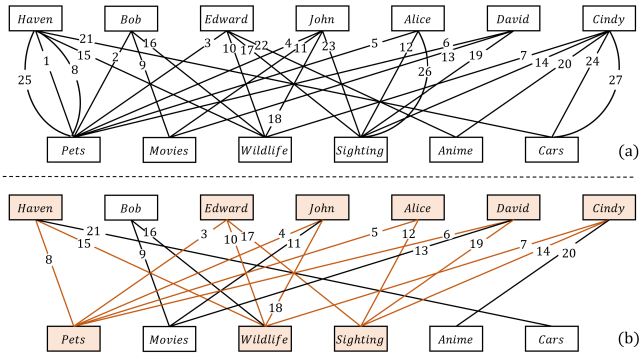


Figure 1: A Flickr users-groups temporal bipartite graph and its snapshot over the time period $\langle 3, 21 \rangle$

Addressing the problem of answering temporal (α, β)-core queries helps finding time-sensitive results and improving effectiveness in real-world applications, but existing solutions [9, 26, 30, 50, 64]

*Yue Wang and Xun Jian are the corresponding authors.

Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097.
doi:10.14778/3681954.3681965

cannot distinguish (α, β)-cores from different time windows. Therefore, in this paper, we aim at finding any (α, β)-core within any time window over a temporal bipartite graph efficiently.

1.1 Applications

Querying temporal (α, β)-cores on bipartite graph benefits many real-world applications [2, 5, 14, 30], we present some of them in the following examples.

Fraud Detection. In social networks like Twitter, users often share external URL links in their posts to express their interests. However, there are also many fraudulent accounts used to promote scam URL links. In this case, relationships between accounts and URL links can be modelled as a bipartite graph, in which each edge represents an account that mentions a URL link in their postings. For promoting scam links, fraudulent accounts tend to share the same scam links consistently, thus forming specific patterns [2, 5] (e.g. the $(10, 5000)$-core finds a huge amount of accounts that mention the few same URLs together). In real-world applications, there always exist suspicious time windows [11], which makes the target (α, β)-cores more effective than those non-temporal solutions [30]. Similarly, in a e-commerce network like Taobao [33], fraudsters control many dummy accounts to promote specific products by adding them into their shopping carts, which forms the same patterns that can be handled by the temporal (α, β)-core.

Temporal-aware Recommendation. Personalized recommendation is widely used in online shopping, movie or blog websites [3, 7, 62]. In temporal recommendation applications, both long-short-term recommendations [43, 55] and seasonal/periodic recommendations [13, 22, 24] have been extensively studied. Answering temporal (α, β)-core queries enables recommendation for long-term or short-term preferences by setting a proper time window. Additionally, specific recommendation results can be obtained within a particular time window [8, 19]. For example, recommending down jackets every winter [56], suggesting turkey or pumpkin for specific holidays each year [34], etc. It is shown that the (α, β)-core of each user is an effective fault-tolerant subspace cluster [14, 20], by varying α and β according to the degree of tolerance for missing values. In Figure 1 (b), we show the snapshot S of the Flickr users-groups graph. Within the $(2, 4)$-core, *Haven*, *Edward*, *John*, *Alice*, *David* and *Cindy* can be considered as potentially sharing preferences of groups. Except for *Edward*, who have already joined *Pets*, *Wildlife* and *Sighting*, one more group can be recommended to all other five users based on common interests.

1.2 Challenges

Naturally, one might wonder whether existing solutions to similar problems can directly address the problem of querying temporal (α, β)-cores. These analogous problems include the historical k-core query problem [60], the non-temporal (α, β)-core query problem [30] and the persistent (α, β)-core community search problem [26]. However, it turns out that the solutions to these similar problems pose the following significant challenges.

• *Challenge 1.* The online solution requires the peeling process for answering each temporal (α, β)-core query. As for the solution of the persistent (α, β)-core community search problem [26], it utilizes the (α, β)-core to find the persistent community of the given query vertex, where the community persistently preserves the (α, β)-core structure of the given time interval. Specifically, this method conducts the graph reduction first according to the given query vertex, which makes it not feasible for the temporal (α, β)-core query problem. Though a naive online approach can be derived from this method, the peeling process for each query results in an expensive cost for each query, which is unacceptable in real-world applications. Its time cost is given in Table 1.

• *Challenge 2.* The existing static solution is unaffordable for space and construction cost to solve this temporal problem. Considering the existing solution Bicore-Index for the non-temporal (α, β)-core query [30], the Bicore-Index cannot recognize timestamp information. We can derive a temporal version of the Bicore-Index by constructing the index for each snapshot over any possible time window. However, the construction of this approach incurs a time complexity of $O(t_{max}^2 \cdot \delta \cdot m)$ and a space complexity of $O(t_{max}^2 \cdot m)$ as shown in Table 1, where t_{max}^2 is unscalable in practice.

• *Challenge 3.* In addition to the challenges posed by the extension of existing methods, the problem itself presents inherently difficult challenges. Firstly, the (α, β)-core has two correlated parameters that need to be simultaneously considered, where α and β represents the threshold requirement of vertices in two layers. Consequently, the relationship between (α, β)-cores needs to be described using a two-dimensional method rather than a one-dimensional monotonicity, this rules out the feasibility of the historical k-core solution [60]. Furthermore, considering all time windows, describing the containment relationship among (α, β)-cores becomes exceedingly difficult. With the change of time windows, old edges are continuously excluded while new edges constantly come into consideration. This results in (α, β)-cores undergoing continuous changes, rendering the simple containment rule that was applicable in static scenarios no longer suitable.

1.3 Contributions

In this paper, we study the new problem of querying (α, β)-cores within a target time window $\langle t_s, t_e \rangle$. We summarize the proposed indexes in Figure 2. For given temporal bipartite graphs and temporal core queries, we construct the vertex-based index $\mathcal{I}_V$, which requires checking each sub-index to obtain the query results. To improve query performance, we restructure it into the query-optimized index $\mathcal{I}_{QO}$ by grouping vertices into the same (α, β)-core under qualified time windows. Furthermore, we construct the superior-optimized index $\mathcal{I}_{SO}$ from $\mathcal{I}_{QO}$, which significantly optimizes space complexity. Our main contributions are presented in detail below.

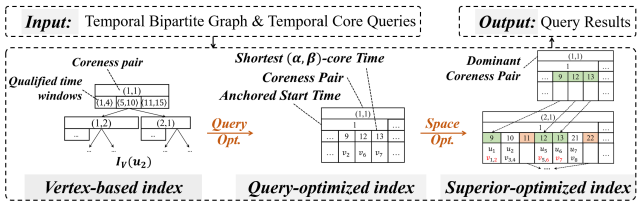


Figure 2: The proposed indexes

Comparisons over cost complexities among three proposed indexes and competitors are listed in Table 1.

• We study the new problem of answering temporal (α, β)-core queries, which helps finding time-sensitive results in applications.
• We propose a novel DAG (Directed Acyclic Graph)-like hierarchy with qualified time windows to describe the temporal containment property of the (α, β)-core. The associated vertex-based index $\mathcal{I}_V$ is effective for solving the problem.
• We construct the superior-optimized index $\mathcal{I}_{SO}$ via leveraging the dominant property between values of α and β, which significantly optimizes space complexity from $O(n \cdot m \cdot \mu)$ to $O(\epsilon \cdot m \cdot \mu)$, while efficient query performance is guaranteed.
• For handling the dynamic property of real-world temporal graphs, we propose efficient maintenance algorithms for $\mathcal{I}_{SO}$ by removing stale information and incorporating newly arrived temporal edges, where the removal also deletes the relevant outdated queries.

Table 1: Comparison over cost complexities, where Constr. means construction, n denotes the number of distinct vertices, m denotes the number of edges of the temporal bipartite graph, δ denotes the maximum value of existence of (α, β)-core such that $\alpha = \beta = \delta$, t_{max} denotes the number of distinct timestamps, $|R|$ denotes the size of the result, μ is the average number of qualified time windows, d_{max} is the maximum degree of all vertices, ϵ denotes the number of vertices that are not compressed ($\mu \ll t_{max}$ and $\epsilon \ll n$ in practice), ρ denotes the average appearance times of all vertices in R, and $O(\sim)$ denotes the construction cost of $\mathcal{I}_V$

Index	Space	Query time	Constr. time
Online	N/A	$O(\log(m) + \delta \cdot \lvert S\rvert)$	N/A
$\mathcal{I}_{TBI}$ [30]	$O(t_{max}^2 \cdot m)$	$O(\lvert R\rvert + \log t_{max}^2)$	$O(t_{max}^2 \cdot \delta \cdot m)$
$\mathcal{I}_V$ [§ 4.3]	$O(n \cdot m \cdot \mu)$	$O(n \cdot \log \mu)$	$O(\delta \cdot m + d_{max} \cdot \mu)$
$\mathcal{I}_{QO}$ [§ 5.1]	$O(n \cdot m \cdot \mu)$	$O(\lvert R\rvert + \log \mu)$	$O(\sim + n \cdot m \cdot \mu)$
$\mathcal{I}_{SO}$ [§ 6.1]	$O(\epsilon \cdot m \cdot \mu)$	$O(\rho \cdot \lvert R\rvert + \log \mu)$	$O(\sim + n \cdot m \cdot \mu)$

1.4 Outline

In Section 2, we review the related work on this topic. Section 3 gives definitions of notations, concepts and problems. The DAG-like hierarchy and the vertex-based $\mathcal{I}_V$ are introduced in Section 4. We present the query-optimized index $\mathcal{I}_{QO}$ and the superior-optimized index $\mathcal{I}_{SO}$ together with associated algorithms in Sections 5 and 6. Finally, we evaluate our methods with extensive experiments in Section 7 and conclude this work in Section 8.

2 RELATED WORK

In this section, we review closely related work to this problem. Many works are conducted on various cohesive subgraph structures over bipartite graphs and temporal cohesive subgraph query.

Cohesive Subgraph Structures over Bipartite Graphs. Effective cohesiveness metric is the keystone of real-world applications of cohesive subgraph mining. Among many types of structures proposed for different scenarios over bipartite graphs, we can classify them into several basic structures:

• *Clique-like:* Clique-like structures aim to capture completeness of subgraphs. Like the k-clique [17, 36, 61, 67] in general graphs, the biclique [33, 35, 57] is the complete subgraph of bipartite graphs, and the (a, b)-biclique [37] is a biclique with exact a vertices in upper vertex layer and exact b vertices in lower vertex layer. However, bicliques suffer from the hardness of computing, both the maximum edge biclique problem and the maximum balanced biclique problem are NP-Hard [37, 51]. For weakening the inherent hardness of cliques, truss-like structures and core-like structures are studied, e.g. the (α, β)-core decomposition can be solved in polynomial time [30].
• *Truss-like:* Truss-like structures follow the idea that the number of triangles each edge involves in [1, 10, 21, 44, 45, 47, 65]. Similarly, the bitruss [49, 52, 69] is the counterpart structure for bipartite graphs, in which actually no triangles exist. Thus a butterfly (a (2, 2)-biclique) [27, 41, 48, 68] works as a triangle-like concept to represent edge containment. What's more, the bi-triangle [59] is an alternative truss-like metric for bipartite graphs instead of defining based on the butterfly. Compared to these truss-like structures, the (α, β)-core is used to describe vertex-centric cohesiveness, while truss-like structures are used to describe edge-centric cohesiveness. This difference leads to varying effectiveness in real-world applications.
• *Core-like:* Core-like structures focus on number of connections among vertices. The (α, β)-core [14, 30, 32] is proposed based on the idea of the k-core [12, 40, 42, 63]. The (α, β)-core is much more efficient to compute compared with clique-like structures and truss-like structures over bipartite graphs, thus it is widely used as a powerful structure in many real-world applications.

Temporal Core-like Subgraph Query. Many different types of temporal core-like subgraph studies have already been explored. Among them, the (k, h)-core [54] investigates the impact of multi-edges in temporal graphs, where h denotes the number of different timestamps between the same two vertices. The (k, Δ)-core [16] tends to study the coexistence of edges within a given time span Δ. Much work revolves around the study of core-like structures in persistent communities in temporal graphs, with the (θ, τ)-persistent k-core [28] considering the time-span threshold θ and the persistence threshold τ. The (θ, τ)-continual k-core [29] extends on the (θ, τ)-persistent k-core by researching community search given a specific query vertex, and the (θ, τ)-persistent (α, β)-core [26] is a simple extension from the (θ, τ)-persistent k-core [28] to bipartite graphs. Similar to the problems studied in this paper, there are also many time-window-based queries, including the historical k-core [60], the scalable time-range k-core [58], and the user-defined temporal $(k, \mathcal{X})$-core [66]. Besides, some work explores the periodic [39] and bursting [38] properties of k-core in temporal graphs. The infeasibility of existing solutions [26, 60] is discussed and others are far from solving this problem.

3 PRELIMINARIES

In this section, we give the formal definition of concepts and the problem statement. A table of all notations can be found in Table 2.

3.1 Problem Definition

Our problem is defined over a temporal bipartite graph denoted by $G(V = (U \cup L), E)$. Here U and L denote disjoint sets of vertices from two layers, (i.e. $U \cap L = \emptyset$). $E = \{e = (u, v, t) \mid u \in U, v \in V\}$ denotes the set of edges, where each edge represents the intersection

between vertices u and v at time t. $\deg(u, G)$ is used to denote degree of $u \in V(G)$ in graph G. We use n to denote the number of distinct vertices in G and m to denote the number of edges of G. Without loss of generality, we consider a single timestamp instead of a time interval for each temporal edge.

For simplicity, we assume that the time span of the whole temporal bipartite graph is $\langle 1, t_{max}\rangle$. For a time window $\langle t_s, t_e\rangle$, we say $\langle t_s, t_e\rangle$ is a ***sub-window*** of $\langle 1, t_{max}\rangle$ if $(t_s \geq 1, t_e < t_{max})$ or $(t_s > 1, t_e \leq t_{max})$, denoted by $\langle t_s, t_e\rangle \subseteq \langle 1, t_{max}\rangle$. We also say $\langle 1, t_{max}\rangle$ is a ***super-window*** of $\langle t_s, t_e\rangle$, denoted by $\langle 1, t_{max}\rangle \supseteq \langle t_s, t_e\rangle$. In this paper, all the discussed time windows $\langle t_s, t_e\rangle$ are assumed to be sub-windows of $\langle 1, t_{max}\rangle$ by default. Given such $\langle t_s, t_e\rangle$, we define the snapshot of G formally.

Definition 1. ***Graph snapshot*** [23]. Given a temporal bipartite graph G with the time span of $\langle 1, t_{max}\rangle$ and a sub-window $\langle t_s, t_e\rangle$ of the time span, the snapshot of G is the subgraph $S_{\langle t_s,t_e\rangle}$ whose $E_{\langle t_s,t_e\rangle} = \{e = (u, v) \mid (u, v, t) \in E(G), t \in \langle t_s, t_e\rangle\}$. Note that multiple edges between the same pair of vertices with different timestamps within the time window are merged into a single edge.

Example 3.1. In Figure 3, the snapshot $S_{\langle 3,20\rangle}$ of the temporal bipartite graph G is the subgraph that excludes u_7, u_8, v_8 and v_9, where edges not present in time window $\langle 3, 20\rangle$ are removed.

The maximal (α, β)-core [14] is defined as follows.

Definition 2. ***Maximal*** (α, β)***-core*** [14]. Given a snapshot $S_{\langle t_s,t_e\rangle}$ and two positive integers α and β, a subgraph $C_{\alpha,\beta}$ is called a maximal (α, β)-core if: • $\deg(u, C_{\alpha,\beta}) \geq \alpha$ and $\deg(v, C_{\alpha,\beta}) \geq \beta$, $\forall u \in U(C_{\alpha,\beta})$, $\forall v \in L(C_{\alpha,\beta})$; • $C_{\alpha,\beta}$ is maximal (i.e. $C_{\alpha,\beta}$ is not contained in any other (α, β)-cores of $S_{\langle t_s,t_e\rangle}$).

The pair of values (α, β) is called **coreness pair** of the structure. In this paper, we simply use the (α, β)-core for short to denote the maximal (α, β)-core. Given a vertex $u \in V(S_{\langle t_s,t_e\rangle})$, we define the set of all coreness pairs of u as the **coreness pair set**, denoted by $\mathrm{CP}(u, S_{\langle t_s,t_e\rangle})$. Considering an (α, β)-core in a bipartite graph, there may exist several (α, β)-cores within it such that they are not connected with each other.

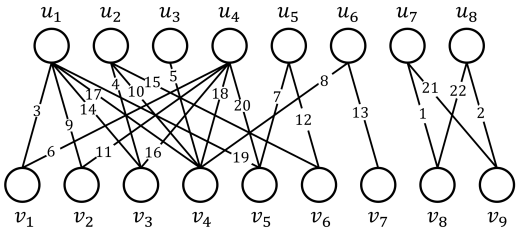


Figure 3: A toy temporal bipartite graph G

Example 3.2. In Figure 3, considering u_2 in $S_{\langle 3,20\rangle}$, the coreness pair set $\mathrm{CP}(u_2, S_{\langle 3,20\rangle})$ is $\{(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 1), (2, 2), (2, 3), (3, 1)\}$. Each coreness pair (α, β) means that u_2 is contained in the (α, β)-core of $S_{\langle 3,20\rangle}$.

Based on the concepts defined above, we formally present the problem statement of temporal (α, β)-core query below.

Problem Statement. ***Temporal*** (α, β)***-core Query***. Given a bipartite graph G, a time window $\langle t_s, t_e\rangle$ and two positive integer α and β, the *temporal* (α, β)*-core query* problem is to find the (α, β)-core in the snapshot $S_{\langle t_s,t_e\rangle}$. The query is denoted as $Q^{\alpha,\beta}_{t_s,t_e}$.

Example 3.3. Considering the temporal bipartite graph G in Figure 3, given the $(1, 5)$-core over $\langle 3, 20\rangle$ as the query $Q^{1,5}_{3,20}$, the result of this temporal $(1, 5)$-core query is $\{u_1, u_2, u_3, u_4, u_6\}$ from the upper layer, and $\{v_4\}$ from the lower layer.

Table 2: Notations

Notation	Description
$G = (V, E)$	temporal bipartite graph
$S_{\langle t_s,t_e\rangle}$	snapshot over $\langle t_s, t_e\rangle$ of G
$V(\cdot), E(\cdot)$	set of vertices and edges respectively
$U(\cdot), L(\cdot)$	set of vertices from the upper layer and the lower layer respectively
$\deg(u, S_{\langle t_s,t_e\rangle})$	the degree of u in $S_{\langle t_s,t_e\rangle}$
$C_{\alpha,\beta}$	the maximal (α, β)-core of $S_{\langle t_s,t_e\rangle}$
$\mathrm{CP}(u, S_{\langle t_s,t_e\rangle})$	the coreness pair set of u in $S_{\langle t_s,t_e\rangle}$
$\mathrm{DCP}(u, S_{\langle t_s,t_e\rangle})$	the dominant coreness pair set of u in $S_{\langle t_s,t_e\rangle}$
$\mathrm{ST}(u, (\alpha, \beta), t_s)$	the shortest (α, β)-core time of u for t_s
$\mathrm{QTW}(u, (\alpha, \beta))$	all qualified windows of u for the (α, β)-core

4 VERTEX-BASED INDEX

In this section, we first introduce the Directed Acyclic Graph (DAG)-like hierarchy of coreness pairs. Then we present the concept of the *qualified time window* for correctly querying temporal (α, β)-cores. Finally, we overview our proposed $\mathcal{I}_V$, showing how it is designed based on the DAG-like hierarchy. The theoretical proofs are in the technical report [46].

4.1 DAG-like Hierarchy

Before illustrating the DAG-like hierarchy, we study the temporal containment property of (α, β)-cores formally. Different from the k-core, the temporal containment property of (α, β)-cores is associated with two values α and β. For correctly describe the containment property of (α, β)-cores and distinguish it from the k-core's, Proposition 4.1 is derived below.

Proposition 4.1. *Considering a snapshot $S_{\langle t_s,t_e\rangle}$ of G, for the (α, β)-core and the (α', β')-core, we say the (α, β)-core* ***contains*** *the (α', β')-core if $(\alpha' > \alpha, \beta' \geq \beta)$ or $(\alpha' \geq \alpha, \beta' > \beta)$. We also say the (α', β')-core is* ***contained*** *by the (α, β)-core.*

Based on Proposition 4.1, we construct a DAG-like hierarchy for (α, β)-cores in Figure 4 to describe the containment relationships among all possible (α, β)-cores, where α_m denotes the maximum α value, β_m denotes the maximum β value and δ denotes the the maximum value of existence of (α, β)-core such that $\alpha = \beta = \delta$ and is bounded by $\sqrt{m}$ [30]. Unlike existing work [30, 50], which only examines the containment property of (α, β)-cores from a one-dimensional perspective (i.e. along either α value or β value), we unify the description of the containment property among (α, β)-cores via a two-dimensional DAG-like perspective.

However, such containment relationships are only guaranteed within a specific time window. When considering (α, β)-cores between different time windows, such containment properties cannot

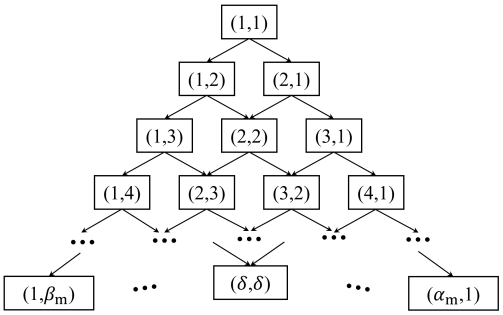


Figure 4: The DAG-like Hierarchy

be ensured. Therefore, to effectively describe the temporal containment relationships between (α, β)-cores across different time windows in bipartite graphs, we introduce the concept of *qualified time window* below.

4.2 Qualified Time Window

To correctly answer temporal (α, β)-core queries, it is essential to accurately describe the temporal containment relationships between (α, β)-cores across different time windows. Although the approach in the historical k-core query [60] is insightful, addressing the temporal containment property within (α, β)-cores poses significant challenges. In this subsection, we introduce the concept of *qualified time window* to aid in the appropriate extension of the DAG-like hierarchy in a temporal scenario.

The key reason why the containment property of (α, β)-cores is challenging to guarantee over different time windows is the uncertainty of the containment relationship of snapshots across different time windows. We obtain Proposition 4.2 by observing the containment relationship of snapshots between time windows.

Proposition 4.2. *Considering a temporal bipartite graph G and two time windows $\langle t_s, t_e\rangle$ and $\langle t'_s, t'_e\rangle$, the corresponding snapshot $S_{\langle t_s,t_e\rangle}$ is a subgraph of the other snapshot $S_{\langle t'_s,t'_e\rangle}$ if $\langle t_s, t_e\rangle \subseteq \langle t'_s, t'_e\rangle$.*

According to Proposition 4.2, we can ensure the containment property of (α, β)-cores across different time windows in the mentioned case. In order to obtain (α, β)-cores under different time windows, compared to the naive index solution that computes for all possible windows, a more efficient approach is to record the times at which the $\mathsf{CP}(\cdot)$ changes for each vertex. From this perspective, we define the concept of *shortest (α, β)-core time* below.

Definition 3. ***Shortest (α, β)-core time***. Given a temporal bipartite graph G, two positive integers α and β, a start time t_s and a vertex $u \in V(G)$, the shortest (α, β)-core time of u for t_s and (α, β) is the soonest time t_e such that u is in the (α, β)-core of $S_{\langle t_s,t_e\rangle}$, denoted by $\mathsf{ST}(u, (\alpha, \beta), t_s)$.

Example 4.1. Considering the vertex $u_2 \in G$ in Figure 3, let the start time t_s be 3, the shortest $(3, 1)$-core time is 15 because in the time interval $\langle 3, 9\rangle$, $deg(u_2) = 1$; in $\langle 3, 14\rangle$, $deg(u_2) = 2$; after the timestamp 15, $deg(u_2) = 3$ and u_2 is contained in the $(3, 1)$-core.

Lemma 4.1. *Given a temporal bipartite graph G, a start time t_s, a vertex $u \in V(G)$ and its shortest (α, β)-core time $\mathsf{ST}(u, (\alpha, \beta), t_s)$, u is contained in the (α, β)-core of $S_{\langle t_s,t_e\rangle}$ if $\mathsf{ST}(u, (\alpha, \beta), t_s) \le t_e$.*

Definition 4. ***Qualified time window***. Given a temporal bipartite graph G, two positive integers α and β, a start time t_s, a vertex $u \in V(G)$ and its the shortest (α, β)-core time $\mathsf{ST}(u, (\alpha, \beta), t_s)$, a time window $\langle t'_s, t'_e\rangle$ is a qualified time window if $t'_e = \mathsf{ST}(u, (\alpha, \beta), t_s)$ and t'_s is the smallest time such that $\mathsf{ST}(u, (\alpha, \beta), t'_s) = t'_e$. We denote all qualified windows of u for the (α, β)-core by $\mathsf{QTW}(u, (\alpha, \beta))$.

Example 4.2. Considering the vertex u_2 temporal bipartite graph G in Figure 3, there are 10 time windows such that u_2 is contained in the $(2, 1)$-core, including $\langle 1, 10\rangle, \langle 2, 10\rangle, \cdots, \langle 5, 15\rangle, \cdots, \langle 10, 15\rangle$. Only two of them are qualified time windows of the $(2, 1)$-core for u_2, i.e. $\mathsf{QTW}(u_2, (2, 1)) = \{\langle 1, 10\rangle, \langle 5, 15\rangle\}$.

4.3 Overview of the Index $\mathcal{I}_V$

Based on the containment property between (α, β)-cores in Proposition 4.1, we need to extend such property from core-based to vertex-based for the index design. We show the dominant property among each vertex's coreness pairs in Proposition 4.3.

Proposition 4.3. *Given a snapshot $S_{\langle t_s,t_e\rangle}$ of G, suppose a vertex $u \in V(S_{\langle t_s,t_e\rangle})$ is contained in both the (α, β)-core and the (α', β')-core, we say (α', β') **dominates** (α, β) if $(\alpha' > \alpha, \beta' \ge \beta)$ or $(\alpha' \ge \alpha, \beta' > \beta)$, denoted as $(\alpha', \beta') \succ (\alpha, \beta)$. We also say (α, β) is **dominated** by (α', β'), denoted as $(\alpha, \beta) \prec (\alpha', \beta')$.*

More specifically, in Lemma 4.2, we map such properties of $\mathsf{CP}(\cdot)$ for each vertex among different time windows.

Lemma 4.2. *Considering two snapshots $S_{\langle t_s,t_e\rangle}$ and $S_{\langle t'_s,t'_e\rangle}$ such that $\langle t_s, t_e\rangle \subseteq \langle t'_s, t'_e\rangle$, for any vertex $u \in S_{\langle t_s,t_e\rangle}$, there may exist a coreness pair in $\mathsf{CP}(u, S_{\langle t'_s,t'_e\rangle})$ that dominates one in $\mathsf{CP}(u, S_{\langle t_s,t_e\rangle})$, but not vice versa.*

By combining the DAG-like hierarchy and the qualified time window, we can construct an index for each vertex to answer the temporal (α, β)-core query. We provide the formal definition for this vertex-based index $\mathcal{I}_V$ below.

Definition 5. ***The vertex-based index $\mathcal{I}_V$***. Given a temporal bipartite graph G, the vertex-based index $\mathcal{I}_V$ consists of all sub-indexes $\mathcal{I}_V(u)$ for all vertices $u \in V(G)$. The sub-index $\mathcal{I}_V(u)$ organises its coreness pairs adhering to the DAG-like hierarchy, where each coreness pair (α, β) arranges all t_s-sorted qualified time windows $\mathsf{QTW}(u, (\alpha, \beta))$.

Example 4.3. Considering the vertex u_2 of the temporal bipartite graph G in Figure 3, the vertex-based index $\mathcal{I}_V(u_2)$ is given in Figure 5, where each coreness pair (α, β) is attached with all qualified time windows $\mathsf{QTW}(u_2, (\alpha, \beta))$, the DAG-like hierarchy correctly forms the temporal containment property of (α, β)-cores over the qualified time windows.

The correctness of the vertex-based index is shown in Theorem 4.3 and the space cost is shown in Proposition 4.4.

Theorem 4.3. *Given the vertex-based index $\mathcal{I}_V$ of G and a temporal (α, β)-core query $Q^{\alpha,\beta}_{t_s,t_e}$, let $\langle t'_s, t'_e\rangle$ be the last qualified time window in $\mathsf{QTW}(u, (\alpha, \beta))$ such that $t'_s \le t_s$ in the $\mathcal{I}_V(u)$. If $t'_e \le t_e$, the vertex u is guaranteed to be included in the result of $Q^{\alpha,\beta}_{t_s,t_e}$.*

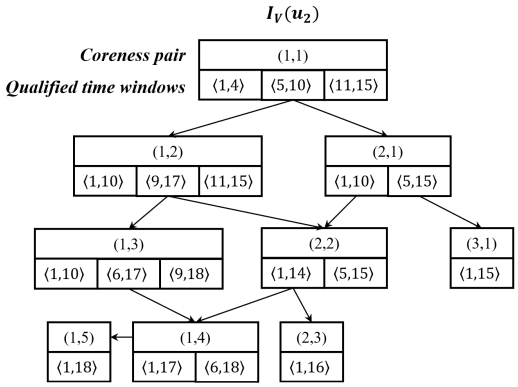


Figure 5: The vertex-based index $\mathcal{I}_V(u_2)$ for G in Figure 3

Proposition 4.4. *Given a vertex-based index $\mathcal{I}_V$ of G, the index size is bounded by $O(n \cdot m \cdot \mu)$, where μ denotes the average number of qualified time windows and $\mu \ll t_{max}$ in practice.*

Query Processing. We denote the query algorithm of the $\mathcal{I}_V$ by Qry_V. Given a correctly built index $\mathcal{I}_V$ of G and a temporal (α, β)-core query $Q^{\alpha,\beta}_{t_s,t_e}$, the query processing of Qry_V starts by traversing all vertices in $V(G)$. For each $\mathcal{I}_V(u)$, each coreness pair is set to be the query entry. If there does not exist the entry for this query (α, β), it turns out it is an invalid query and the correct result should be an empty set. After the target coreness pair entry being matched with the query (α, β), the checking process of qualified time windows takes $O(\log |\text{QTW}(u, (\alpha, \beta))|)$ by binary search. Thus the overall time cost is $O(n \cdot \log \mu)$, where μ is the average number of qualified time windows traversed.

4.4 Index Construction

For each vertex, we need to obtain qualified time windows for all possible α, β, the *anchored start time* and the *shortest (α, β)-core time*. A naive approach is to conduct decomposition for the snapshots over all possible anchored start time, and the shortest (α, β)-core time can be obtained during the decomposition. However, the time cost of this naive approach still requires $O(t^2_{max} \cdot \delta \cdot m)$, making it as impractical as the naive indexing solution. Therefore, a more efficient method for index construction is desired.

In real-world applications, temporal graphs inherently possess the characteristic of dynamically inserting new edges. Therefore, it is natural to consider techniques for (α, β)-core maintenance in dynamic graphs [30, 32]. Considering the containment property of (α, β)-cores over temporal bipartite graphs, there are two feasible ideas to obtain qualified time windows utilizing the maintenance technique: 1) bottom-up: from the unit time windows (i.e. $\langle 1, 2\rangle$, $\langle 2, 3\rangle$, $\cdots$, $\langle t_{max} - 1, t_{max}\rangle$), it expands these time windows like maintaining (α, β)-cores given edge insertions; 2) top-down: from the largest time window (i.e. $\langle 1, t_{max}\rangle$), it shrinks this time window like maintaining (α, β)-cores given edge deletions. Although the worst-case time complexity for both edge deletion and edge insertion is the same according to the SOTA method of (α, β)-core maintenance [32], the boundedness analysis for (α, β)-core maintenance remains unclear. To assist us in choosing a more efficient approach, we first investigate the boundedness property in (α, β)-core maintenance.

Boundedness Analysis. For the (α, β)-core maintenance problem [32], we denote the set of vertices whose coreness pair set changes by CNG. An incremental algorithm $\mathcal{A}$ is *bounded* [15] if its cost is a polynomial function of $\|\text{CNG}\|_c$, where $\|\cdot\|_c$ denotes the size of c-hop neighbors for a positive integer c. It has been demonstrated that both *k*-truss maintenance and *k*-core maintenance exhibit asymmetry concerning edge insertions and deletions [65]. Next, we present the theoretical conclusion in Theorem 4.4 and provide its proof in the technical report [46].

Theorem 4.4. *The (α, β)-core maintenance is bounded for edge deletions, but unbounded for edge insertions.*

Index Construction. Based on the boundedness analysis, it is evident that the top-down approach is guaranteed to be bounded when constructing the $\mathcal{I}_V$. We denote the index construction algorithm of the $\mathcal{I}_V$ in Algorithm 1 by Cons_V. In lines 1-2, it first decomposes the bipartite graph over the largest time window and initializes the auxiliary structures. In lines 3-4, it initializes the shortest (α, β)-core time of each vertex for all possible coreness pairs at the start time 1 by by deleting the edges at the specific timestamp and maintaining coreness pairs [30, 32]. Then in lines 6-8, it iteratively increases the start time t_s and updates the affected shortest (α, β)-core time. Finally, it constructs sub-indexes $\mathcal{I}_V(u)$ by correct qualified time windows for each vertex in lines 11-12. For the time complexity, line 1 costs $O(\delta \cdot m)$, line 2 costs $O(m)$, line 3-10 costs $O(d_{max} \cdot \mu)$. The overall cost is $O(\delta \cdot m + d_{max} \cdot \mu)$.

Algorithm 1: Cons_V: the $\mathcal{I}_V$ construction

Input: the temporal bipartite graph G
Output: the $\mathcal{I}_V$ of G
1 Call the SOTA decomposition algorithm [30] on $S_{\langle 1, t_{max}\rangle}$;
2 Initialize auxiliary structures for all $u \in V(G)$;
3 **foreach** $t \in \langle t_{max}, 2\rangle$ **do**
4 Delete edges at the timestamp t;
5 Maintain coreness pairs for edges within $\langle 1, t-1\rangle$;
6 **foreach** *the start time* $t_s \in \langle 2, t_{max}\rangle$ **do**
7 Delete edges at the timestamp $t_s - 1$ and update auxiliary structures due to the deletion;
8 Update the shortest (α, β)-core time for vertices only if necessary;
9 **foreach** $u \in V(G)$ **do**
10 $\mathcal{I}_V(u) \leftarrow$ qualified time windows over coreness pairs;
11 **return** the $\mathcal{I}_V$ of G;

5 QUERY-OPTIMIZED INDEX

While the vertex-based index $\mathcal{I}_V$ overcomes the challenges posed by the two-value-associated (α, β)-core and effectively unifies the containment property between time windows and cores, successfully solving the temporal (α, β)-core query problem, the query cost $O(n \cdot \log \mu)$ of the $\mathcal{I}_V$ consists of the term n, which denotes the number of vertices in the whole graph and is undesirable for query performance. Therefore, our goal is to optimize query performance to be only related to each query rather than the graph

size. In this section, we aim to optimize query performance and reconstruct the $\mathcal{I}_V$ to obtain the query-optimized index $\mathcal{I}_{QO}$ with a query cost of $O(|R| + \log \mu)$ without changing the space overhead, where $|R|$ denotes the size of the result. The theoretical proofs are in the technical report [46].

5.1 Overview of the index $\mathcal{I}_{QO}$

To optimize the term n in the query cost $O(n \cdot \log \mu)$ of the vertex-based index $\mathcal{I}_V$, a natural approach is to abandon the vertex-based structure. Instead, it is more reasonable to organize the coreness pairs and time windows related to a single query systematically and correctly place the result under each query entry. Based on this idea, we can perform a one-to-one mapping transformation on the vertex-based index $\mathcal{I}_V$ to achieve this goal. Formally, we define the query-optimized index $\mathcal{I}_{QO}$ as follows.

Definition 6. ***The query-optimized index $\mathcal{I}_{QO}$***. Given a temporal bipartite graph G, the query-optimized index $\mathcal{I}_{QO}$ consists of all sub-indexes for any coreness pair (α, β) of the given G, where the sub-index $\mathcal{I}_{QO}(\alpha, \beta)$ systematically arranges all t_s-sorted qualified time windows adhering to the DAG-like hierarchy. The corresponding vertex set is attached to each qualified time window.

Example 5.1. Considering the temporal bipartite graph G in Figure 3, a part of the query-optimized index $\mathcal{I}_{QO}$ is given in Figure 6 of the technical report [46], where we only present the part such that anchored start time $t_s = 1$ and six coreness pairs due to space limit. The DAG-like hierarchy correctly forms the containment property of temporal (α, β)-cores.

Proposition 5.1. *Given a query-optimized index $\mathcal{I}_{QO}$ of G, the index size is bounded by $O(n \cdot m \cdot \mu)$.*

Query Processing. We denote the query algorithm of the $\mathcal{I}_{QO}$ by Qry_{QO}. Given a correctly built index $\mathcal{I}_{QO}$ of G and a temporal (α, β)-core query $Q^{\alpha,\beta}_{t_s,t_e}$, the query processing of Qry_{QO} starts by retrieving the sub-index $\mathsf{Qry}_{QO}(\alpha, \beta)$ if it exists for the target (α, β). If there does not exist such $\mathsf{Qry}_{QO}(\alpha, \beta)$, it implies that there is no valid result for this (α, β)-core query. Within the $\mathsf{Qry}_{QO}(\alpha, \beta)$, it first checks the last anchored start time t'_s such that $t'_s \le t_s$, then it collects vertices in the vertex sets under qualified time windows until the last qualified time window $\langle t'_s, t'_e \rangle$ such that $t'_e \le t_e$. The overall time cost is $O(|R| + \log \mu)$, where $|R|$ is the size of the result.

The Construction of $\mathcal{I}_{QO}$. We introduce the transformation of $\mathcal{I}_V$ into $\mathcal{I}_{QO}$ through remapping. We denote the index construction algorithm of the $\mathcal{I}_{QO}$ by Cons_{QO}. It first initializes structures for the $\mathcal{I}_{QO}$. Then it traverses the vertex-based index $\mathcal{I}_V$ and conducts one-one remapping to obtain the sub-indexes $\mathcal{I}_{QO}(\alpha, \beta)$. The time cost of Cons_{QO} is as same as the size of the $\mathcal{I}_V$, i.e. $O(n \cdot m \cdot \mu)$. The overall cost of the $\mathcal{I}_{QO}$ construction is $O(\delta \cdot m + d_{max} \cdot \mu + n \cdot m \cdot \mu)$.

6 SUPERIOR-OPTIMIZED INDEX

While the query-optimized index $\mathcal{I}_{QO}$ achieves the great improvement of query performance from $O(n \cdot \log \mu)$ to $O(|R| + \log \mu)$, the high space complexity of $O(n \cdot m \cdot \mu)$ also presents a challenge. Furthermore, if we can identify and compress redundant information in the $\mathcal{I}_{QO}$, it can significantly improve the space efficiency. In this section, targeting space efficiency, we demonstrate the compressibility of coreness pairs in the DAG-like hierarchy, proposing the superior-optimized index $\mathcal{I}_{SO}$, which significantly optimizes space cost from $O(n \cdot m \cdot \mu)$ to $O(\epsilon \cdot m \cdot \mu)$ while ensuring an acceptable query cost $O(\rho \cdot |R| + \log \mu)$. Here ϵ denotes the number of vertices that are not compressed ($\epsilon \ll n$ in practice) and ρ denotes the average appearance times of all vertices in R.

6.1 Overview of the index $\mathcal{I}_{SO}$

We observe significant redundancy in existing indexes $\mathcal{I}_V$ and $\mathcal{I}_{QO}$, primarily stemming from the compressibility of coreness pairs for each vertex. Next, we introduce the dominant property among coreness pairs and illustrate how such a property can compress redundant information in the DAG-like hierarchy. Finally, we construct the superior-optimized index $\mathcal{I}_{SO}$ based on these properties. We give the definition of *the dominant coreness pair* formally.

Definition 7. ***Dominant coreness pair*** [32]. In a snapshot $S_{\langle t_s,t_e \rangle}$, given an (α, β)-core and an (α', β')-core that both contain $u \in V(S_{\langle t_s,t_e \rangle})$, if there does not exist any other coreness pairs (α'', β'') containing u which dominates (α', β'), the coreness pair (α', β') is a dominant (α', β') coreness pair of u.

Note that for a single vertex $u \in V(S_{\langle t_s,t_e \rangle})$, there may exist multiple dominant coreness pairs. We define them as **dominant coreness pair set** of the vertex u, denoted by $\mathrm{DCP}(u, \langle t_s, t_e \rangle)$.

Different from the existing work [32], which utilizes dominant coreness pairs to reduce the time overhead of dynamic maintenance, in our work, we leverage this property to design the storage scheme, thereby achieving efficient space and query efficiency.

Based on the definition of the dominant coreness pair, we elaborate on how it works in the DAG-like hierarchy in Figure 4. The arrows in the hierarchy denote the dominant relationship between two (α, β) coreness pairs, for example, $(1, 1) \to (1, 2)$ means that the coreness pair $(1, 1) \prec (1, 2)$. It is straightforward that the DAG-like hierarchy can cover all (α, β) cores of the given G.

Example 6.1. In the snapshot $S_{\langle 3,20 \rangle}$ of the temporal graph in Figure 3, u_2 appears in multiple different (α, β) cores. Among all coreness pairs that u_2 belongs to, the coreness pair $(1, 5) \succ \{(1, 1), (1, 2), (1, 3), (1, 4)\}$, the coreness pair $(2, 3) \succ \{(2, 1), (2, 2)\}$, thus the dominant coreness pair set $\mathrm{DCP}(u_2, \langle 3, 20 \rangle)$ is $\{(1, 5), (2, 3), (3, 1)\}$ since each coreness within $\mathrm{DCP}(u_2, \langle 3, 20 \rangle)$ cannot dominate each other.

With the dominant property among coreness pairs and the DAG-like hierarchy, it is natural to organise vertices by grouping them according to their dominant coreness pairs. This structure forms the foundation of the superior-optimized index $\mathcal{I}_{SO}$. Towards answering temporal (α, β)-core queries, we need to find all (α, β)-cores correctly. Thus we define the dominant coreness hub below and we guarantee the correctness of its specific linking rules.

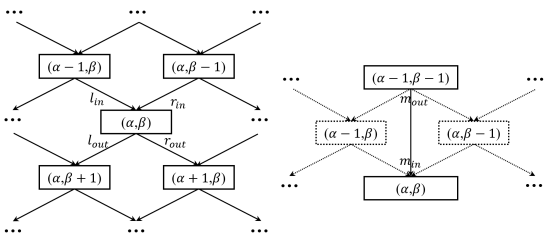
Definition 8. ***Dominant coreness hub***. Given a snapshot $S_{\langle t_s,t_e \rangle}$ and two positive integer α and β, a set of vertices $\mathcal{H}_{\alpha,\beta}$ is called a dominant coreness hub if all vertices with dominant (α, β) coreness pair are contained in this set. We call it a hub for short in this paper.

Example 6.2. Considering the snapshot $S_{\langle 3,20 \rangle}$ of the graph in Figure 3, we have $\mathcal{H}_{2,2} = \{u_5, v_6\}$, which consists of vertices which have the dominant coreness pair of $(2, 2)$. Though u_1 is also contained in a $(2, 2)$-core, it is assigned to $\mathcal{H}_{2,3}$ since $(2, 3) \succ (2, 2)$.

The linking rules among dominant coreness hubs are the key to organise all (α, β)-cores correctly, we elaborate on them below utilizing the property of dominance and discontinuity among coreness pairs in the hierarchy.

Linking Rules. Considering two coreness pairs (α, β) and (α', β'), we say they are continuous if the Manhattan distance between them is exactly 1. In real-world bipartite graphs, coreness pairs may not be continuous in the hierarchy, we need to handle the discontinuity which is essential to guarantee correct linking among hubs. Generally, given a dominant (α, β) hub, the linking rules to another dominant (α', β') hub are as follows.

- *Rule 1. Direct-link.* If (α, β) shares the same value with (α', β') over α (resp. β), i.e. if $\alpha = \alpha'$ (resp. $\beta = \beta'$), the other is with a gap $t = |\beta' - \beta|$ (resp. $t = |\alpha' - \alpha|$), then (α, β) and (α', β') can be *direct-linked* as long as t is minimized among all possible (α', β').
- *Rule 2. Skip-link.* If (α, β) does not share the same value with (α', β') over α or β, then they can be *skip-linked* only when the Manhattan distance between (α, β) and (α', β') is minimized among all possible (α', β').



(a) One case of *Direct-Link*. (b) One case of *Skip-Link*.

Figure 6: The linking rules of dominant coreness hubs

We illustrate *Rule 1* in Figure 6 (a). Considering (α, β), there are four hubs' values of which each increases or decreases of the coreness by exactly 1 over α or β, then they can be *direct-linked* to (α, β). We use *in* (resp. *out*) to denote the relationship that (α, β) dominates (resp. is dominated). For large graphs, the hierarchy is very likely to be compact and *Rule 1* can be fulfilled in most cases.

There are cases where *Rule 1* cannot be applied due to the discontinuity among coreness pairs. We show one common case that *Rule 1* can not be fulfilled in Figure 6 (b), where dashed boxes mean that both of the $(\alpha - 1, \beta)$ hub and the $(\alpha, \beta - 1)$ hub don't exist in the hierarchy. In this case, we can utilize *Rule 2* to find the (α', β') hub with the smallest Manhattan distance to $(\alpha - 1, \beta - 1)$, i.e. the (α, β) hub in Figure 6 (b), then (α, β) can be *skip-linked* to $(\alpha - 1, \beta - 1)$.

Correctness. According to Proposition 4.1, if $(\alpha', \beta') \succ (\alpha, \beta)$, $C_{\alpha',\beta'}$ is completely contained by $C_{\alpha,\beta}$. Given a snapshot $S_{\langle t_s, t_e \rangle}$, all dominant (α, β) hubs that are linked by *Rule 1* and *Rule 2*, can be utilized to find all possible (α, β)-cores of G. We show the correctness in Theorem 6.1. Its proof can be referred in the report [46].

Theorem 6.1. *With all dominant coreness hubs $\mathcal{H}_{\alpha,\beta}$ of $S_{\langle t_s,t_e \rangle}$ being linked by Rule 1 and Rule 2, it is feasible to find all possible (α, β)-cores of $S_{\langle t_s,t_e \rangle}$ correctly.*

However, it is not space-advantageous to compute dominant coreness hubs over all qualified time windows directly, as the space saved will be offset by introduced linkings. Thus we perform the hub computation over the same qualified time windows among sub-indexes in the $\mathcal{I}_{QO}$ to obtain the superior-optimized index $\mathcal{I}_{SO}$.

Definition 9. ***The superior-optimized index $\mathcal{I}_{SO}$***. Given a temporal bipartite graph G, the superior-optimized index $\mathcal{I}_{SO}$ systematically arranges all t_s-sorted qualified time windows adhering to the DAG-like hierarchy, where the corresponding vertex set is equal to the dominant coreness hub under same qualified time windows among dominant coreness pairs.

Example 6.3. Considering the graph G in Figure 3, a part of the superior-optimized index $\mathcal{I}_{SO}$ is given in Figure 7 due to space limit. Different from the $\mathcal{I}_{QO}$ in Figure 6 of the technical report [46], for the marked same time windows among different coreness pairs, vertices are only stored in the dominant coreness hubs.

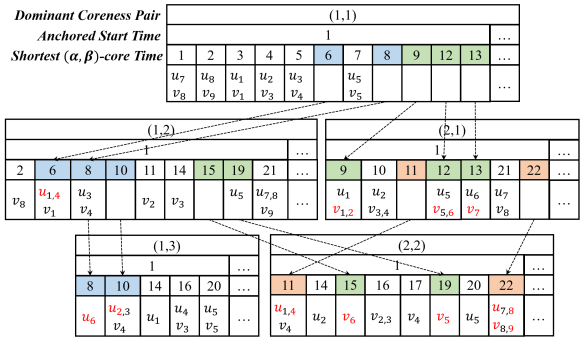


Figure 7: The superior-optimized index $\mathcal{I}_{SO}$ for G in Figure 3

We show the space cost of the superior-optimized index in Proposition 6.1, its proof can be referred in the report [46].

Proposition 6.1. *Given a superior-optimized index $\mathcal{I}_{SO}$ of G, the index size is bounded by $O(\epsilon \cdot m \cdot \mu)$.*

Query Processing. We denote Algorithm 2 by Qry_{SO}. Given the index $\mathcal{I}_{SO}$ of G and a temporal (α, β)-core query $Q^{\alpha,\beta}_{t_s,t_e}$, in lines 2-5, the query processing of Qry_{SO} starts by retrieving the sub-index $\text{Qry}_{SO}(\alpha', \beta')$ where (α', β') equals (α, β) if $\text{Qry}_{SO}(\alpha, \beta)$ exists, or equals the dominant coreness pair with the smallest Manhattan distance to (α, β). In line 6, if there does not exist such $\text{Qry}_{SO}(\alpha', \beta')$, it implies that there is no valid result for this (α, β)-core query. Then in lines 9-13, within the $\text{Qry}_{SO}(\alpha', \beta')$, it first checks the last anchored start time t'_s such that $t'_s \leq t_s$, then it collects vertices in the dominant coreness hub under qualified time windows until the last qualified time window $\langle t'_s, t'_e \rangle$ such that $t'_e \leq t_e$. The visited hubs (α', β') are marked. In lines 14-15, we check the linked dominant coreness pairs that dominate (α, β), and push them into the queue for BFS. In lines 16-19, we perform the query result checking from all hubs such that no other coreness pair dominates it to guarantee the correctness. From line 8 to line 19, the processing is performed recursively until there is no unvisited coreness pair that dominates (α, β). The overall time cost is $O(\rho \cdot |R| + \log \mu)$, where ρ denotes the average appearance times of all vertices in R, i.e. $\rho = \frac{1}{|R|} \cdot \sum_{u \in R} |\text{DCP}(u)|$, such that $\text{DCP}(u) \succ (\alpha, \beta)$.

Algorithm 2: Qry_{SO}: answering Q_{α,β,t_s,t_e} via the $\mathcal{I}_{SO}$

Input: $\mathcal{I}_{SO}$, $Q^{\alpha,\beta}_{t_s,t_e}$
Output: R

1 $R \leftarrow \emptyset$, $Queue \leftarrow \emptyset$, $(\alpha', \beta') \leftarrow \emptyset$;
2 **if** $\text{Qry}_{SO}(\alpha, \beta)$ *exists* **then**
3 $(\alpha', \beta') \leftarrow (\alpha, \beta)$;
else
5 $(\alpha', \beta') \leftarrow$ the dominant coreness pair with the smallest Manhattan distance to (α, β);
6 **if** $(\alpha', \beta') == \emptyset$ **then return** R;
7 $Queue \leftarrow (\alpha', \beta')$;
8 **while** $!Queue.size()$ **do**
9 $(\alpha', \beta') \leftarrow Queue.top(), Queue.pop()$;
10 Mark (α', β') as visited;
11 Find the last anchored t'_s in $\text{Qry}_{SO}(\alpha', \beta')$ such that $t'_s \le t_s$;
12 **foreach** *hub under* $\text{QTW}_{\langle t'_s, t'_e \rangle}$ *such that* $t'_e \le t_e$ **do**
13 $R \leftarrow$ all vertices within this hub;
14 **foreach** *coreness pair* (α'', β'') *that linked to* (α', β') **do**
15 $Queue \leftarrow Queue \cup (\alpha'', \beta'')$;
16 **foreach** *coreness pair* (α, β) *such that no other pair dominates it* **do**
17 **if** (α, β) *is NOT visited* **then**
18 $Queue \leftarrow Queue \cup (\alpha, \beta)$;
19 Repeat lines 8-15 similarly in the bottom-up manner;
20 **return** R;

6.2 The Construction of $\mathcal{I}_{SO}$

Addressing the significant redundancy in existing indexes $\mathcal{I}_V$ and $\mathcal{I}_{QO}$, the two key procedures are the computation and the linking of dominant coreness hubs. In this subsection, we first illustrate the construction framework of the $\mathcal{I}_{SO}$.

We denote the index construction algorithm of the $\mathcal{I}_{SO}$ in Algorithm 3 by Cons_{SO}. In lines 1-3, it first computes dominant coreness hubs over qualified time windows. Then the construction procedure of sub-indexes is similar to Cons_{QO} in line 4. Finally, it needs to link all sub-indexes $\mathcal{I}_{SO}(\alpha, \beta)$ according to the DAG-like hierarchy in line 5. Next, we focus on how to compute and link dominant coreness hubs correctly.

The Computation of Dominant Coreness Hubs. Utilizing the decomposition algorithm [30], we can obtain the coreness pair set $\text{CP}(u)$ for each vertex $u \in V(G)$ with time complexity of $O(\delta \cdot m)$, noting that coreness pairs within $\text{CP}(u)$ is sorted by α first and β then. In Algorithm 4, given the sorted $\text{CP}(u)$, we need to do two rounds of filtering for two layers of vertices in G respectively. In line 1, we initialize variables. In lines 2-6, we sieve out coreness pairs which are not dominant coreness pairs along α for vertices in the upper layer. In line 7, we conduct sieving similarly for vertices in the lower layer. After two rounds of filtering, we obtain $\text{DCP}(u)$ for each $u \in V(S_{\langle t_s, t_e \rangle})$, then we map vertices to hubs reversely based on obtained $\text{DCP}(u)$ from line 12 to 14. The time complexity of Algorithm 4 is $O(E(S_{\langle t_s, t_e \rangle}))$.

The Linking of Dominant Coreness Hubs. To ensure all hubs are linked correctly with their neighbor hubs, we design pointers following linking rules *Direct-link* and *Skip-link* for each hub. For

Algorithm 3: Cons_{SO}: the $\mathcal{I}_{SO}$ construction

Input: the $\mathcal{I}_V$ of G
Output: the $\mathcal{I}_{QO}$ of G

1 **foreach** $u \in V(G)$ **do**
2 **foreach** t_s *in associated qualified time windows in* $\mathcal{I}_V(u)$ **do**
3 Compute dominant coreness hubs by Algorithm 4 with same QTWs of neighbor coreness pairs;
4 Repeat Cons_{QO} similarly for these dominant coreness pairs;
5 Link all sub-indexes $\mathcal{I}_{SO}(\alpha, \beta)$ according to the DAG-like hierarchy by Algorithm 5;
6 **return** the $\mathcal{I}_{SO}$ of G;

Algorithm 4: Dominant Coreness Hubs Computation

Input: G, $\text{CP}(u)$ for each $u \in V(S_{\langle t_s, t_e \rangle})$
Output: $\text{DCP}(u)$ for each $u \in V(S_{\langle t_s, t_e \rangle})$

1 $\beta_m \leftarrow 0$, $\alpha_m \leftarrow 0$;
2 **foreach** $u \in U(S_{\langle t_s, t_e \rangle})$ **do**
3 **foreach** $\alpha \in \text{CP}(u)$ **do**
4 **foreach** $\beta \in \text{CP}(u)(\alpha)$ **do**
5 $\beta_m = \max(\beta, \beta_m)$;
6 $\text{DCP}(u) \leftarrow (\alpha, \beta_m)$;
7 Perform lines 2-6 similarly for all $v \in L(S_{\langle t_s, t_e \rangle})$;
8 **return** $\text{DCP}(u)$ for each $u \in V(S_{\langle t_s, t_e \rangle})$;

Algorithm 5: Dominant Coreness Hubs Linking

1 **foreach** $hub \in \mathcal{H}_{\alpha,\beta}$ **do**
2 $sFlag_{out} \leftarrow true$; $sFlag_{in} \leftarrow true$;
3 $(\alpha, \beta) \leftarrow$ the coreness of this hub; $\alpha' \leftarrow \alpha$, $\beta' \leftarrow \beta$;
4 Call $\text{DirectLink}_{\text{out}}$ for l_{out} and r_{out};
5 **if** $!l_{out}$ *and* $!r_{out}$ **then** $\text{SkipLink}_{\text{out}}(m_{out})$;
6 Call $\text{DirectLink}_{\text{in}}$ for l_{in} and r_{in};
7 **if** $sFlag_{in}$ **then** $\text{SkipLink}_{\text{in}}(m_{in})$;

Procedure. $\text{SkipLink}_{\text{out}}(m_{out})$
while $\beta' \le \beta_m$ **do**
  $\beta' \leftarrow \beta' + 1$;
  while $\alpha' \le \alpha_m$ **do**
  $\alpha' \leftarrow \alpha' + 1$;
  if $\mathcal{H}_{\alpha',\beta'}$ *exists* **then**
  $hub.m_{out} \leftarrow \mathcal{H}_{\alpha',\beta'}$; $\mathcal{H}_{\alpha',\beta'}.m_{in} \leftarrow hub$;
  $sFlag_{out} \leftarrow false$; break;
  if $!sFlag_{out}$ **then** break;

Procedure. $\text{DirectLink}_{\text{out}}(\cdot)$
Consider $\text{SkipLink}_{\text{out}}$, remove the outer loop for l_{out} (or the inner one for r_{out}), and interchange m_{out}, m_{in} with their counterparts;

Direct-link, we design four pointers named as l_{out}, r_{in}, r_{out}, l_{in}, corresponding with each of the four possible direct-linked neighbors, where l denotes the left-hand-side and r denotes the right-hand-side, subindexes *out* and *in* denote directions in the hierarchy shown in Figure 6 (a). For *Skip-link*, we set m_{in} and m_{out} similarly. Note that *Skip-link* is only applied when there exists no neighbors available with *Direct-link* along the same direction of *out* or *in*, e.g. m_{out} is only linked when neither l_{out} nor r_{out} is able to be linked correctly.

Algorithm 5 shows how we link all hubs according to the DAG-like hierarchy. In lines 2-3, we initialize related variables. In lines 4-5, we call $\mathsf{DirectLink}_{\mathsf{out}}$ for checking two to-be-directly-linked pointers of the out-direction and call $\mathsf{SkipLink}_{\mathsf{out}}$ if there are no available l_{out} and r_{out}. In lines 6-7, we conduct similarly for the in-direction. We omit $\mathsf{SkipLink}_{\mathsf{in}}(m_{in})$ and $\mathsf{DirectLink}_{\mathsf{in}}(\cdot)$ due to the space limit. The time complexity of Algorithm 5 is $O(m)$.

6.3 The Maintenance of $\mathcal{I}_{SO}$

Considering the inherent temporal property in the temporal (α, β)-core query problem, the size of our proposed index $\mathcal{I}_{SO}$ is not desired to be bounded by the graph size, especially as the temporal graph continues to grow larger and larger. Therefore, we focus on the maintenance over temporal graphs, which aims to eliminate staleness and efficiently merge insertions in the index. In this section, we propose the maintenance algorithm for the index $\mathcal{I}_{SO}$.

Algorithm 6: Main_{SO}: Temporal Maintenance of the $\mathcal{I}_{SO}$

Input: the $\mathcal{I}_{SO}$ over $\langle 1, t_{max}\rangle$, the stale time t_{sta}, new edges $E(S_{\langle t_{max}+1, t'_{max}\rangle})$

Output: the $\mathcal{I}_{SO}$ over $\langle t_{sta}+1, t'_{max}\rangle$

1 **foreach** $\mathcal{I}_{SO}(\alpha, \beta)$ **do**
2 Remove associated qualified time windows s.t. $t_s \in \langle 1, t_{sta}\rangle$;
3 Keep (α, β)-cores of the snapshot $S_{\langle t_{sta}+1, t_{max}\rangle}$ from the $\mathcal{I}_{SO}$;
4 Obtain (α, β)-cores of the snapshot $S_{\langle t_{sta}+1, t'_{max}\rangle}$ by maintaining $S_{\langle t_{sta}+1, t_{max}\rangle}$ given edge insertions $E(S_{\langle t_{max}+1, t'_{max}\rangle})$;
5 Repeat lines 2-10 in Algorithm 1 similarly over the time window $\langle t_{sta}+1, t'_{max}\rangle$;
6 Update qualified time windows and hubs only if necessary;
7 **return** the $\mathcal{I}_{SO}$ over $\langle t_{sta}+1, t'_{max}\rangle$;

Suppose that it is required to drop the stale edges within $\langle 1, t_{sta}\rangle$ and merge the new edges within $\langle t_{max}+1, t'_{max}\rangle$, where t'_{max} denotes the new latest timestamp. For the $\mathcal{I}_{SO}$, the deletion of the staleness information is straightforward and can be performed by removing qualified time windows whose $t_s \in \langle 1, t_{sta}\rangle$. However, dealing with newly inserted edges within $\langle t_{max}+1, t'_{max}\rangle$ remains a challenging task. We describe how to address this point specifically in Algorithm 6, which is denoted by Main_{SO}. In lines 1-2, it first drops the stale part of the existing index, which costs $O(\max_{u \in V(S_{\langle 1, t_{sta}\rangle})} |\mathsf{CP}(u)| \cdot \mu_{\langle 1, t_{sta}\rangle})$. In line 3, it keeps all (α, β)-cores of the snapshot $S_{\langle t_{sta}+1, t_{max}\rangle}$ from the existing index to avoid re-decomposition, which costs $O(m)$. Then in line 4, it obtains the result of the snapshot $S_{\langle t_{sta}+1, t'_{max}\rangle}$ by maintaining $S_{\langle t_{sta}+1, t_{max}\rangle}$ with edge insertions $E(S_{\langle t_{max}+1, t'_{max}\rangle})$ [32], which costs $O(\max_{u \in V(S')} \deg(u) \cdot m)$ in the worst case, where $S' = S_{\langle t_{max}+1, t'_{max}\rangle}$. In lines 5-6, the maintenance procedure is similar to the construction procedure in Algorithm 1, which costs $O(t'_{max} \cdot m)$. Note that qualified time windows and dominant coreness hubs are updated only if necessary. The overall time complexity of Algorithm 6 is $O((\max_{u \in V(S')} \deg(u) + t'_{max}) \cdot m)$, where $S' = S_{\langle t_{max}+1, t'_{max}\rangle}$. Though the time complexity of Algorithm 6 is equal to the re-construction solution Algorithm 3 in the worst case, Algorithm 6 is always much more efficient on real world graphs because the visited subgraph during its process is typically significantly smaller than the entire graph.

We omit the guideline for index selection due to space limit, which can be found in the technical report [46].

7 EXPERIMENTS

In this section, we evaluate the performance of the query processing, the index construction and maintenance, and the index size. We also conduct a case study of the temporal (α, β)-core query on the real-world dataset. All algorithms are implemented in C++ and all experiments are performed on a Linux server with an Intel Xeon Gold 6240R CPU @ 2.40 GHz and 256 GB main memory.

7.1 Experimental Setups

Datasets. We use eight real datasets in our experiments which are Stackoverflow (ST), Linux-kernel (LK), Citeulike (CU), Twitter (TW), Amazon Ratings (AR), Last.fm (LF), Wiktionary (WN) and Wikipedia (WP). All datasets can be found from KONECT [1]. The summary of datasets is shown in Table 3. $|E|$ denotes the number of edges, $|U|$ and $|L|$ denote the number of vertices in two layers respectively. d_{max} is the maximum degree of all vertices. t_{max} denotes the number of distinct timestamps. δ denotes the maximum value of existence of (α, β)-core such that $\alpha = \beta = \delta$. μ is the average number of qualified time windows for each coreness pair in our proposed indexes.

Algorithms. Our empirical studies are conducted based following algorithms. We use $\mathsf{Qry}_{(\cdot)}$ and $\mathsf{Cons}_{(\cdot)}$ to denote the query algorithm and the construction algorithm of a specific method respectively. We denote the pure online query solution by $\mathsf{Qry}_{\mathsf{OL}}$ and denote the construction algorithm of temporal Bicore-Index [30] by $\mathsf{Cons}_{\mathsf{TBI}}$. Similar notations are adopted for the $\mathcal{I}_V$, $\mathcal{I}_{QO}$ and $\mathcal{I}_{SO}$. We terminate algorithms that cannot be completed within 36 hours, note that $\mathsf{Cons}_{\mathsf{TBI}}$ cannot finish on all datasets.

Table 3: Summary of Datasets

G	$\lvert E\rvert$	$\lvert U\rvert$	$\lvert L\rvert$	d_{max}	t_{max}	δ	μ
ST	1.30M	545K	96.6K	6.11K	99.6K	22	16
LK	1.56M	42.0K	337K	31.7K	1.08B	12	43
CU	2.41M	153K	731K	189K	103K	27	87
TW	4.66M	175K	530K	19.8K	99.8M	23	74
AR	5.83M	2.14M	1.23M	12.1K	315M	26	91
LF	19.1M	992	1.08M	55.5K	272M	164	125
WN	44.7M	66.1K	5.82M	3.50M	513M	97	131
WP	129.8M	1.02K	5.91K	818K	509M	212	163

7.2 Query Processing

In this part, we investigate the performance of our proposed query processing algorithms $\mathsf{Qry}_{\mathsf{V}}$, $\mathsf{Qry}_{\mathsf{QO}}$ and $\mathsf{Qry}_{\mathsf{SO}}$ together with the competitor $\mathsf{Qry}_{\mathsf{OL}}$. For input parameters, we vary the time window size as 10%, 30%, 50%, 70%, 90% of t_{max} and 30% by default; we vary (α, β) randomly to be dominated by (α_m, β_m), these (α, β)s may be invalid for evaluating the ability of validity checking. We first test the algorithms on all eight datasets with 100 queries generated randomly for each dataset and we take the average to report in

[1] http://konect.cc/networks/

each test. Then, we evaluate the performance to process queries when varying (α, β) and time window size.

Performance over all datasets. The performance of query processing over all datasets is shown in Figure 8 (a), we can observe that Qry_{V}, Qry_{QO} and Qry_{SO} perform consistently better than Qry_{OL}. Qry_{QO} performs best among all methods, it is $1 \sim 2$ orders of magnitude faster than Qry_{V} and can be several orders better than Qry_{OL}. Qry_{SO} is comparable to Qry_{QO} but a bit slower than Qry_{QO} because of extra retrieval among dominant coreness pairs to collect compressed vertices. On the largest dataset WP, the advantages of Qry_{QO} and Qry_{SO} are most significant being 167× and 91× faster than Qry_{OL}.

Varying (α, β). We evaluate the query performance by varying the value of α and β and we report the results of two representative datasets TW and WP, the results on other datasets show similar trends. In Figure 8 (b), (c), both α and β are the same and varied together according to the ratio of δ. When the ratio is small, the query time of all algorithms are the longest, since the size of the query result is relatively large. As the ratio is getting larger, the size of the result and the number of qualified time windows become smaller, thus it takes less time for Qry_{V}, Qry_{QO} and Qry_{SO} to answer queries. Qry_{OL} is not sensitive to the ratio because of the online decomposition from the original graph. We can see that Qry_{QO} and Qry_{SO} have good scalability to the value of (α, β). Fixing either α or β has a similar impact on query performance compared to changing both α and β simultaneously.

Varying time window size. We evaluate the query performance by varying the time window size in Figure 8 (d), (e). We can see a considerable increase for Qry_{QO} and Qry_{SO}, while Qry_{V} is not sensitive to the size of time windows. Because as the time window gets larger, the query result also becomes larger, it takes more time for Qry_{QO} and Qry_{SO} to collect vertices in the result. Qry_{V} checks every vertex no matter how large the time window is, thus it keeps stable. As the window size gets larger, the difference between Qry_{SO} and Qry_{QO} becomes slightly larger since larger window size will invoke more retrieval among coreness pairs of Qry_{SO}.

7.3 Index Construction & Maintenance

Construction performance over all datasets. In Figure 8 (f), we can see that constructing $\mathcal{I}_{QO}$ and $\mathcal{I}_{SO}$ is slower than constructing $\mathcal{I}_{V}$, which is reasonable since $\mathcal{I}_{QO}$ needs time for remapping transformation from $\mathcal{I}_{V}$ and $\mathcal{I}_{SO}$ needs more time for the computation and the linking of dominant coreness hubs. Overall, the construction time increases with the increase in graph size (m). We can observe that the construction time on CU is longer than that of TW and AR, even though they have larger graph sizes. Because d_{max} of CU is relatively larger, making it denser TW and AR.

Construction performance when varying $|G|$. We also report the scalability of the construction methods in Figure 8 (g), (h). For each dataset, all edges are sorted in chronological order. We pick the first 10%, 30%, 50%, 70%, 90% of the edges from the original graph to perform the algorithms. As the graph gets larger, the cost of all methods become longer because all methods are dependent to the graph size and the time span of the dataset. Cons_{QO} and Cons_{SO} are comparable to Cons_{V} at all portions, which are consistent with the result in Figure 8 (f).

Maintenance performance when varying $|\Delta G|$. We report the scalability of Main_{SO} in Figure 8 (i), (j). We pick the first 5%, 10%, 15%, 20% of the edges from TW as the stale edges to delete and keep the last 5%, 10%, 15%, 20% of the edges as the new edges to insert. Similar settings are applied to WP as 4%, 8%, 12%, 16%. As the portion of updated edges gets larger, the reconstruction Cons_{SO} gets shorter because the updated graph size gets smaller. With more edges being updated, Main_{SO} needs more time to finish the maintenance of the index. Main_{SO} reaches the efficiency bound at around 20% for TW and at around 16% for WP.

7.4 Index Size

Performance over all datasets. In Figure 8 (k), we present the size of the original graph as the baseline. We can see that $\mathcal{I}_{QO}$ and $\mathcal{I}_{V}$ are the most space-costly indexes, which can be significantly larger than the original graph for all datasets, especially $\mathcal{I}_{QO}$ is 12× larger than the graph for LF. The size of $\mathcal{I}_{QO}$ and $\mathcal{I}_{V}$ are very close because $\mathcal{I}_{QO}$ is reformatted from $\mathcal{I}_{V}$. $\mathcal{I}_{SO}$ is space-efficient and around one order of magnitude smaller than $\mathcal{I}_{QO}$ and $\mathcal{I}_{V}$, and it is comparable to the graph size.

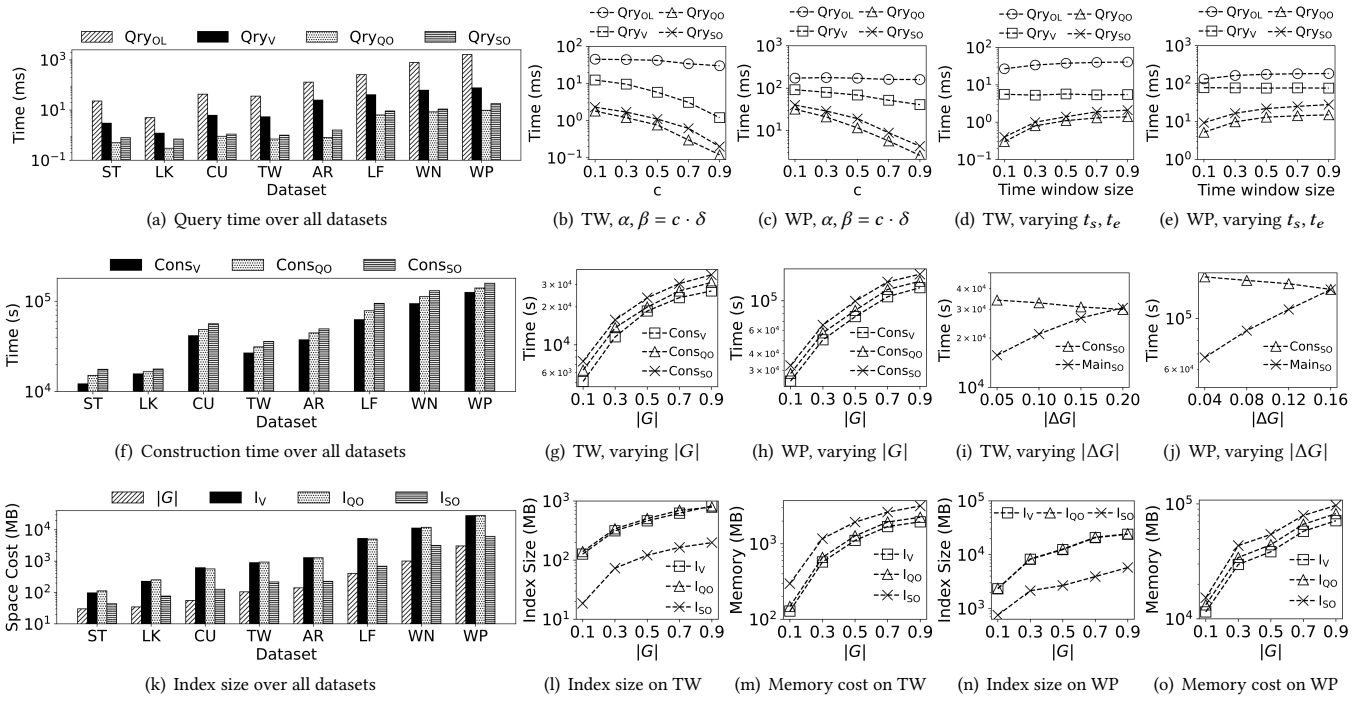
Scalability evaluation when varying $|G|$. We also conduct the scalability evaluation on TW and WP regarding the index size and peak memory usage of all indexes in Figure 8 (l), (m), (n), and (o). For each dataset, all edges are generated exactly same as (g) and (h). As the graph gets larger, the space cost of both index size and memory usage becomes larger near-linearly since they are dependent to the graph size and the time span of the dataset. For index size, we can observe that the data of $\mathcal{I}_{V}$ and $\mathcal{I}_{QO}$ are very close to each other, while $\mathcal{I}_{SO}$ exhibits a significantly better space efficiency by around an order of magnitude. For peak memory usage, though $\mathcal{I}_{SO}$ has slightly higher memory consumption compared to $\mathcal{I}_{V}$ and $\mathcal{I}_{QO}$, the peak memory usage is around 10^5 MB for the largest dataset WP, which is acceptable in practice.

7.5 Case Study

To show the effectiveness of temporal (α, β)-core queries in fault-tolerant recommendation, we conduct the case study on the DBLP graph of Jiawei Han. In the DBLP bipartite graph, one layer of vertices represents the co-authors of Jiawei Han, while the other layer of vertices represents the publications. For clarity, we have merged the publications into their venue. Jiawei Han is connected to every venue in the graph, and we have omitted those edges. As for other authors, we have only depicted their connections to the top four venues based on the number of papers.

We present the static $(50, 2)$-core of Jiawei Han's graph in Figure 9, starting from 1985 when he published his first paper, where all co-authors have published at least 50 papers with Jiawei Han. We can observe that Jiawei Han has a wide range of research interests, with a focus on data mining (KDD, WWW, CIKM, ICDM). Additionally, he has made significant contributions in the field of databases (TKDE, SIGMOD, VLDB, ICDE) and natural language processing (ACL, EMNLP).

Next, we consider the time window from 2017 to now, the temporal $(50, 2)$-core based on this snapshot excludes the authors highlighted in orange in Figure 9. We indicate below each author the



(a) Query time over all datasets (b) TW, $\alpha, \beta = c \cdot \delta$ (c) WP, $\alpha, \beta = c \cdot \delta$ (d) TW, varying t_s, t_e (e) WP, varying t_s, t_e

(f) Construction time over all datasets (g) TW, varying $|G|$ (h) WP, varying $|G|$ (i) TW, varying $|\Delta G|$ (j) WP, varying $|\Delta G|$

(k) Index size over all datasets (l) Index size on TW (m) Memory cost on TW (n) Index size on WP (o) Memory cost on WP

Figure 8: Performance evaluation

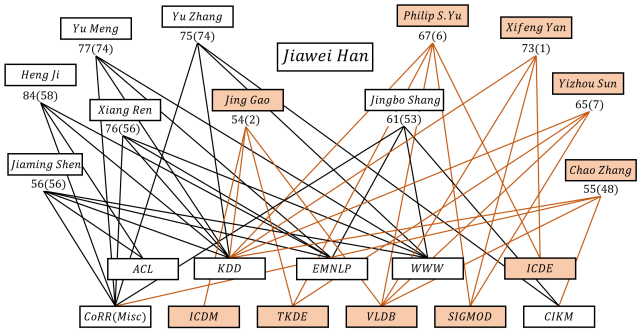


Figure 9: Case study

number of publications they have starting from 1985, with the number of publications starting from 2017 shown in parentheses. We can observe that, except for Chao Zhang, the other four authors excluded from the temporal $(50, 2)$-core are predominantly involved in the field of databases prior to 2017. However, their collaboration with Jiawei Han has significantly declined after 2017, while the collaboration between other authors and Jiawei Han remains relatively unchanged. This indicates that Jiawei Han has shifted his research focus to data mining and natural language processing after 2017, gradually moving away from the database field. Such distinctions cannot be observed from the static (α, β)-core.

8 CONCLUSION

In this paper, we study the problem of answering temporal (α, β)-core queries. We propose a novel DAG-like hierarchy and qualified time windows to accurately describe the temporal containment property of (α, β)-cores and construct an vertex-based index that successfully solves the problem. For enhancing its query performance and space efficiency, we propose the query-optimized index and the superior-optimized index. We also propose a maintenance approach that can efficiently update the the superior-optimized index. Extensive experimental results show the efficiency and effectiveness of our proposed indexes.

ACKNOWLEDGMENTS

Lei Chen's work is partially supported by National Key Research and Development Program of China Grant No. 2023YFF0725100, National Science Foundation of China (NSFC) under Grant No. U22B2060, the Hong Kong RGC GRF Project 16213620, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, CRF Project C2004-21G, Guangdong Province Science and Technology Plan Project 2023A0505030011, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Zhujiang scholar program 2021JC02X170, Microsoft Research Asia Collaborative Research Grant, HKUST-Webank joint research lab and HKUST(GZ)-Chuanglin Graph Data Joint Lab. Yue Wang is partially supported by China NSFC (No.62002235).

REFERENCES

[1] E. Akbas and P. Zhao. Truss-based community search: a truss-equivalence based indexing approach. *PVLDB*, 10(11):1298–1309, 2017.

[2] M. Allahbakhsh, A. Ignjatovic, B. Benatallah, S. Beheshti, E. Bertino, and N. Foo. Collusion detection in online rating systems. *APWeb*, 7808:196–207, 2013.

[3] S. Amer-Yahia, S. B. Roy, A. Chawla, G. Das, and C. Yu. Group recommendation: Semantics and efficiency. *PVLDB*, 2(1):754–765, 2009.

[4] W. Bai, Y. Chen, D. Wu, Z. Huang, Y. Zhou, and C. Xu. Generalized core maintenance of dynamic bipartite graphs. *Data Min. Knowl. Discov.*, pages 1–31, 2022.

[5] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. *WWW*, pages 119–130, 2013.

[6] X. Cai, X. Ke, K. Wang, L. Chen, T. Zhang, Q. Liu, and Y. Gao. Efficient temporal butterfly counting and enumeration on temporal bipartite graphs. *PVLDB*, 2023.

[7] L. A. M. C. Carvalho and H. T. Macedo. Users' satisfaction in recommendation systems for groups: an approach based on noncooperative games. *WWW*, pages 951–958, 2013.

[8] O. Celma. Music recommendation. In *Music recommendation and discovery: The long tail, long fail, and long play in the digital music space*, pages 43–85. Springer, 2010.

[9] M. Cerinsek and V. Batagelj. Generalized two-mode cores. *Soc. Networks*, 42:80–87, 2015.

[10] H. Chen, A. Conte, R. Grossi, G. Loukides, S. P. Pissis, and M. Sweering. On breaking truss-based communities. *KDD*, pages 117–126, 2021.

[11] D. Cheng, X. Wang, Y. Zhang, and L. Zhang. Graph neural network for fraud detection via spatial-temporal attention. *TKDE*, 34(8):3800–3813, 2020.

[12] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu. Efficient core decomposition in massive networks. *ICDE*, pages 51–62, 2011.

[13] G. de Souza Pereira Moreira, F. Ferreira, and A. M. da Cunha. News session-based recommendations using deep neural networks. In *DLRS*, pages 15–23, 2018.

[14] D. Ding, H. Li, Z. Huang, and N. Mamoulis. Efficient fault-tolerant group recommendation using alpha-beta-core. *CIKM*, pages 2047–2050, 2017.

[15] W. Fan, C. Hu, and C. Tian. Incremental graph computations: Doable and undoable. In *SIGMOD*, pages 155–169, 2017.

[16] E. Galimberti, M. Ciaperoni, A. Barrat, F. Bonchi, C. Cattuto, and F. Gullo. Span-core decomposition for temporal networks: Algorithms and applications. *TKDD*, 15(1):1–44, 2020.

[17] E. Gregori, L. Lenzini, and S. Mainardi. Parallel k-clique community detection on large-scale networks. *TPDS*, 24(8):1651–1660, 2012.

[18] Y. Guan, R. Lu, Y. Zheng, S. Zhang, J. Shao, and G. Wei. Achieving efficient and privacy-preserving (α, β)-core query over bipartite graphs in cloud. *IEEE TDSC*, 2022.

[19] J. A. Gulla, C. Marco, A. D. Fidjestøl, J. E. Ingvaldsen, and Ö. Özgöbek. The intricacies of time in news recommendation. In *UMAP*, 2016.

[20] S. Günnemann, E. Müller, S. Raubach, and T. Seidl. Flexible fault tolerant subspace clustering for data with missing values. *ICDM*, pages 231–240, 2011.

[21] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. *SIGMOD*, pages 1311–1322, 2014.

[22] K. Keerthika and T. Saravanan. Enhanced product recommendations based on seasonality and demography in ecommerce. In *ICACCCN*, pages 721–723, 2020.

[23] U. Khurana and A. Deshpande. Efficient snapshot retrieval over historical graph data. In *ICDE*, pages 997–1008. IEEE, 2013.

[24] T. Kramár and M. Bieliková. Context of seasonality in web search. In *ECIR 2014*, pages 644–649. Springer, 2014.

[25] M. Ley. The DBLP computer science bibliography: Evolution, research issues, perspectives. *SPIRE*, 2476:1–10, 2002.

[26] M. Li, Z. Xie, and L. Ding. Persistent community search over temporal bipartite graphs. In *ADMA*, pages 324–339. Springer, 2023.

[27] R. Li, P. Wang, P. Jia, X. Zhang, J. Zhao, J. Tao, Y. Yuan, and X. Guan. Approximately counting butterflies in large bipartite graph streams. *TKDE*, 2021.

[28] R.-H. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai. Persistent community search in temporal networks. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 797–808. IEEE, 2018.

[29] Y. Li, J. Liu, H. Zhao, J. Sun, Y. Zhao, and G. Wang. Efficient continual cohesive subgraph search in large temporal graphs. *WWW*, 24:1483–1509, 2021.

[30] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou. Efficient (α, β)-core computation: An index-based approach. *WWW*, pages 1130–1141, 2019.

[31] Q. Liu, X. Liao, X. Huang, J. Xu, and Y. Gao. Distributed (α, β)-core decomposition over bipartite graphs. In *ICDE*, pages 909–921. IEEE, 2023.

[32] W. Luo, Q. Yang, Y. Fang, and X. Zhou. Efficient core maintenance in large bipartite graphs. *SIGMOD*, 1(3):1–26, 2023.

[33] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, and J. Zhou. Maximum biclique search at billion scale. *PVLDB*, 13(9):1359–1372, 2020.

[34] L. Ma, J. H. Cho, S. Kumar, and K. Achan. Seasonality-adjusted conceptual-relevancy-aware recommender system in online groceries. In *ICBD*, pages 4435–4443. IEEE, 2019.

[35] Z. Ma, Y. Liu, Y. Hu, J. Yang, C. Liu, and H. Dai. Efficient maintenance for maximal bicliques in bipartite graph streams. *WWW*, pages 1–21, 2021.

[36] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.

[37] R. Peeters. The maximum edge biclique problem is np-complete. *Discrete. Appl. Math.*, 131(3):651–654, 2003.

[38] H. Qin, R.-H. Li, Y. Yuan, G. Wang, L. Qin, and Z. Zhang. Mining bursting core in large temporal graphs. *PVLDB*, 2022.

[39] H. Qin, R.-H. Li, Y. Yuan, G. Wang, W. Yang, and L. Qin. Periodic communities mining in temporal networks: Concepts and algorithms. *TKDE*, 34(8):3927–3945, 2020.

[40] A. E. Sarıyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and Ü. V. Çatalyürek. Incremental k-core decomposition: algorithms and evaluation. *VLDBJ*, 25(3):425–447, 2016.

[41] J. Shi and J. Shun. Parallel algorithms for butterfly computations. *SIAM*, pages 16–30, 2020.

[42] B. Sun, T.-H. H. Chan, and M. Sozio. Fully dynamic approximate k-core decomposition in hypergraphs. *TKDD*, 14(4):1–21, 2020.

[43] K. Sun, T. Qian, T. Chen, Y. Liang, Q. V. H. Nguyen, and H. Yin. Where to go next: Modeling long-and short-term user preferences for point-of-interest recommendation. In *AAAI*, volume 34, pages 214–221, 2020.

[44] Z. Sun, X. Huang, J. Xu, and F. Bonchi. Efficient probabilistic truss indexing on uncertain graphs. *WWW*, pages 354–366, 2021.

[45] A. Tian, A. Zhou, Y. Wang, and L. Chen. Maximal d-truss search in dynamic directed graphs. *PVLDB*, 16(9):2199–2211, 2023.

[46] A. Tian, A. Zhou, Y. Wang, X. Jian, and L. Chen. Efficient index for temporal core queries over bipartite graphs [technical report]. https://github.com/ExpCodeBase/tabc/blob/main/full.pdf [Online]. Accessed: 2024-07-09.

[47] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.

[48] J. Wang, A. W.-C. Fu, and J. Cheng. Rectangle counting in large bipartite graphs. *Int. Congr. Big Data*, pages 17–24, 2014.

[49] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang. Efficient bitruss decomposition for large-scale bipartite graphs. *ICDE*, pages 661–672, 2020.

[50] K. Wang, W. Zhang, X. Lin, Y. Zhang, L. Qin, and Y. Zhang. Efficient and effective community search on large-scale bipartite graphs. *ICDE*, pages 85–96, 2021.

[51] Y. Wang, S. Cai, and M. Yin. New heuristic approaches for maximum balanced biclique problem. *Information Sciences*, 432:362–375, 2018.

[52] Y. Wang, R. Xu, X. Jian, A. Zhou, and L. Chen. Towards distributed bitruss decomposition on bipartite graphs. *PVLDB*, 15(9):1889–1901, 2022.

[53] S. Wasserman and K. Faust. *Social network analysis - methods and applications*, volume 8 of *Structural analysis in the social sciences*. Cambridge University Press, 2007.

[54] H. Wu, J. Cheng, Y. Lu, Y. Ke, Y. Huang, D. Yan, and H. Wu. Core decomposition in large temporal graphs. In *ICBD*, pages 649–658. IEEE, 2015.

[55] R. Xie, Y. Wang, R. Wang, Y. Lu, Y. Zou, F. Xia, and L. Lin. Long short-term temporal meta-learning in online recommendation. In *WSDM*, pages 1168–1176, 2022.

[56] H. Yang, P. Gupta, R. Fernández Galán, D. Bu, and D. Jia. Seasonal relevance in e-commerce search. In *CIKM*, pages 4293–4301, 2021.

[57] J. Yang, Y. Peng, and W. Zhang. (p, q)-biclique counting and enumeration for large sparse bipartite graphs. *PVLDB*, 15(2):141–153, 2021.

[58] J. Yang, M. Zhong, Y. Zhu, T. Qian, M. Liu, and J. X. Yu. Scalable time-range k-core query on temporal graphs. *PVLDB*, 16(5):1168–1180, 2023.

[59] Y. Yang, Y. Fang, M. E. Orlowska, W. Zhang, and X. Lin. Efficient bi-triangle counting for large bipartite networks. *PVLDB*, 14(6):984–996, 2021.

[60] M. Yu, D. Wen, L. Qin, Y. Zhang, W. Zhang, and X. Lin. On querying historical k-cores. *PVLDB*, 2021.

[61] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang. Diversified top-k clique search. *VLDBJ*, 25(2):171–196, 2016.

[62] Q. Yuan, G. Cong, and C. Lin. COM: a generative model for group recommendation. *SIGKDD*, pages 163–172, 2014.

[63] Y. Zhang and S. Parthasarathy. Extracting analyzing and visualizing triangle k-core motifs within networks. *ICDE*, pages 1049–1060, 2012.

[64] Y. Zhang, K. Wang, W. Zhang, X. Lin, and Y. Zhang. Pareto-optimal community search on large bipartite graphs. *CIKM*, pages 2647–2656, 2021.

[65] Y. Zhang and J. X. Yu. Unboundedness and efficiency of truss maintenance in evolving graphs. *SIGMOD*, pages 1024–1041, 2019.

[66] M. Zhong, J. Yang, Y. Zhu, T. Qian, M. Liu, and J. X. Yu. A unified and scalable algorithm framework of user-defined temporal $(k, \mathcal{X})$-core query. *TKDE*, 2024.

[67] A. Zhou, Y. Wang, and L. Chen. Finding large diverse communities on networks: the edge maximum k*-partite clique. *PVLDB*, 13(12):2576–2589, 2020.

[68] A. Zhou, Y. Wang, and L. Chen. Butterfly counting on uncertain bipartite graphs. *PVLDB*, 15(2):211–223, 2021.

[69] Z. Zou. Bitruss decomposition of bipartite graphs. *DASFAA*, pages 218–233, 2016.