Efficient Frequency-Aware k-Core Query on Temporal Graphs

Zhongfan Du*, Ming Zhong*✉, Yuanyuan Zhu*, Tieyun Qian*, Mengchi Liu†, and Jeffrey Xu Yu‡
*School of Computer Science, Wuhan University, Wuhan, China
†South China Normal University, Guangzhou, China
‡The Chinese University of Hong Kong, Hong Kong, China
{zhongfandu, clock, yyzhu, qty}@whu.edu.cn, liumengchi@scnu.edu.cn, yu@se.cuhk.edu.hk

2025 IEEE 41st International Conference on Data Engineering (ICDE) | 979-8-3315-3603-9/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICDE65448.2025.00179

***Abstract*—In temporal graphs, time and topology are considered to be intertwined. As an evidence, it is observed that the vertices in more cohesive subgraphs have more frequent and more numerous interactions between each other in the history. Motivated by that, we study a novel frequency-aware k-core query problem. Different from previous studies that focus on finding k-cores in the projected subgraphs of given time intervals, we look for the subgraphs of k-core in which neighbor vertices have at least a certain number of high-frequency interactions. To address the problem, we propose 1) a minimum slope algorithm for computing the frequency in linear time, 2) a space-efficient index that stores the distinct "core frequency" of vertices for addressing arbitrary queries, 3) a propagation algorithm that collects core frequencies by message passing for index construction, and 4) efficient algorithms for retrieving a specific or all skyline results from the index respectively. The experimental results show that, our algorithms achieve several orders of magnitude improvement on efficiency compared to corresponding baselines, and meanwhile, the size of index is even smaller than that of graph unless the graph has very few timestamps on each edge. More importantly, by both statistics and case study, it is verified that the frequency-aware k-core query indeed find more cohesive subgraphs in the static k-core.**

***Index Terms*—temporal graph, k-core, interaction frequency, core frequency, skyline.**

I. Introduction

Background. The temporal graph has attracted enormous research interests recently. In the database community, there have been a variety of graph query models that take time into consideration, such as temporal path query [1, 2], temporal k-core/truss query [3–10], temporal motif/triangle/butterfly counting [11, 12], etc. In the machine learning community, the representation learning [13, 14] of temporal graphs becomes a major task, which aims to capture the dynamic representation varying with time. In the web search community, the centrality [15, 16] in temporal graphs is also an emerging topic.

In related studies, the main difference between temporal and non-temporal graphs is the presence of timestamped edges. Figure 1 shows a toy example of temporal social graph. Each edge has an ordered list of integer timestamps, which denotes two people had multiple interactions such as follows, comments, retweets, and likes at different times in the past. From the perspective of database, the temporal graph is a set of historical transactions, and in contrast, the non-temporal graph can be seen as the result of grouping transactions by endpoints without retaining time.

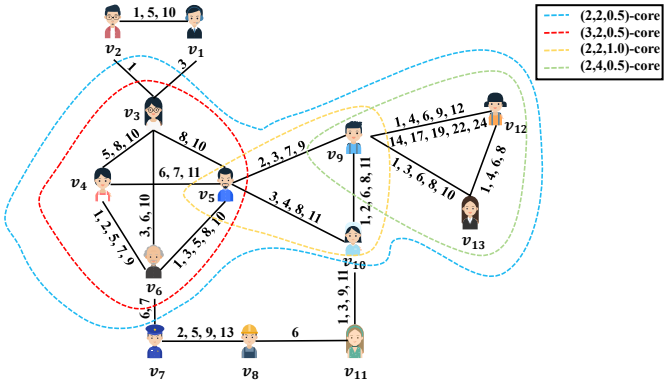
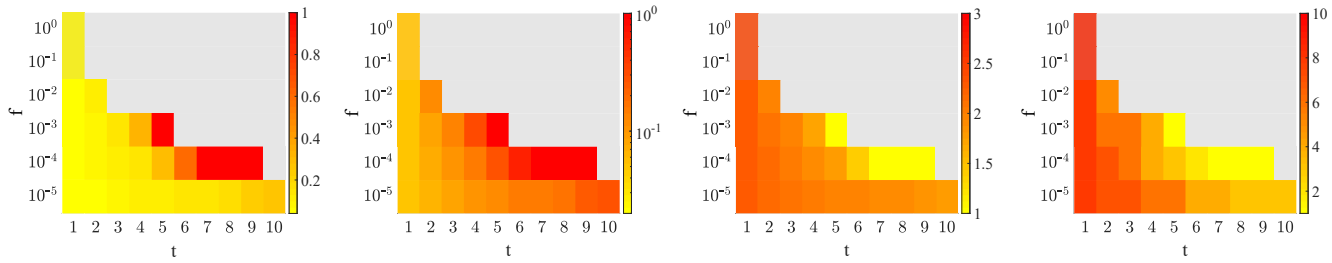


Fig. 1: A running example of temporal social graph. The edge labels are lists of timestamps to indicate when the connected people had interactions. The dashed frames remark several frequency-aware k-cores in the graph.

Motivation. Traditional graph queries like the k-core query can benefit from constraints on the lists of timestamps on edges. For example, the list length represents the number of interactions between two vertices, which measures the strength of the relationship accumulated between the two. However, a large number alone may result from occasional interactions over a long period, thereby failing to reflect sustained engagement. Therefore, the list length divided by the span of timestamps, which measures how frequently two vertices interact, also offers unique insights. Different from existing temporal k-core queries [4, 5, 7] that consider two vertices as mutual neighbors if they have at least one interaction, it is more reasonable to do so if they have a certain number of high-frequency interactions. This allows to find a more cohesive subgraph of k-core, where each vertex has at least k neighbors in both strong and frequent contact.

To reveal the positive effect of interaction frequency constraints on the k-core, we conduct an empirical study on a real-world temporal social network called Email. In brief, one constraint is to require a pair of neighbor vertices have at least t interactions, and the other is to require the maximum frequency of these at least t interactions is no less than f. We calculate four widely-used community metrics [17] (see details in Section VI-F) for k-cores with varying values of t

2375-026X/25/$31.00 ©2025 IEEE
DOI 10.1109/ICDE65448.2025.00179



(a) Internal density. (b) Clustering coefficient. (c) Average distance. (d) Number of communities.

Fig. 2: The heat maps of various k-core metrics with respect to the frequency constraints (namely, t and f) on Email dataset. Note that, the metrics (a) and (b) are the greater the better, while (c) and (d) are the less the better.

and f. The results are shown as heat maps in Figure 2. We can see that, the increase of t or f indeed improves the quality of k-core as a community.

Application. There are many potential application scenarios of the frequency-aware k-core query.

Social networks. The k-core is widely used to identify influential users [18, 19]. Obviously, a user has the stronger influence to another if they have more high-frequency contacts. For example, in online forums, users with frequent interactions can trigger discussion cascades faster than those with sparse ties. Active participants who frequently engage in discussions and reply to posts play a crucial role in spreading information and engaging other members of the community. Thus, after applying interaction frequency constraints to the k-core, we can identify more influential users.

Transaction networks. The detection of fraud groups can be assisted by the k-core query [20, 21]. Since only the transfers whose amounts are greater than a certain value will be supervised, fraudsters usually use many high-frequency small-amount microtransactions to evade regulation. Such fraud groups can be modeled by the frequency-aware k-core.

Communication networks. The k-core is also used to locate crucial devices, so that we can strengthen the connections between them in order to enhance the stability of networks [22, 23]. By considering the number and frequency of communications between devices, we can further narrow down the scope of crucial devices, thereby reducing maintenance costs.

Challenge. On the one hand, the online query processing based on core decomposition [24] is inefficient on large-scale temporal graphs, not to mention the frequency of each edge needs to be recalculated with respect to given parameters such as t and f. On the other hand, a straightforward index that stores the result vertices for all possible parameters is too redundant and costly in space. Although index compression techniques such as [25, 26] can potentially address the problem, they only allow one extra parameter other than k. For more parameters, the delta encoding scheme is too complicated to be as effective as in previous work [27]. Moreover, since there are multiple parameters, query writing could be difficult when faced with an unfamiliar graph.

Contribution. In this paper, we study the frequency-aware k-core query problem on temporal graphs. Our contributions are summarized as follows.

Query model (Section II). As the cornerstone, we define a novel metric of temporal edge called t-frequency, which is the highest frequency of at least t interactions on the edge. Consequently, we define a frequency-aware k-core called (k, t, f)-core, in which each vertex has k neighbor vertices such that the t-frequencies of temporal edges between them are no less than a given threshold $f \in [0, 1]$. The (k, t, f)-core is flexible compared to other existing definitions. In particular, it becomes the historical/temporal k-core [4, 5] when $t = 1$ or the span-core [28] when t equals to the time span.

Frequency computation (Section III). We present a linear-time algorithm to compute the t-frequency for temporal edges. The basic idea is to convert the t-frequency problem to an equivalent minimum slope problem in a particular Cartesian coordinate system. Then, we leverage the upper convex curve to prune candidates, thereby reducing the time complexity from quadratic to linear in terms of the number of timestamps.

Index and Query processing (Section IV). To address the (k, t, f)-core query problem, we propose both an online core decomposition algorithm with the worst case time complexity $O(|\mathcal{E}| \cdot |T|_{avg})$ and a sophisticate Core Frequency Index (CF-Index) that needs at most $O(|\mathcal{V}| \cdot \log |T|_{avg})$ time to process a query, where $|\mathcal{E}|$ is the number of edges, $|\mathcal{V}|$ is the number of vertices, and $|T|_{avg}$ is the average number of timestamps on each edge. The CF-Index is designed on top of a novel concept called "core frequency", which is the maximum value of f such that a vertex can appear in the (k, t, f)-core with given k and t. By exploiting the monotonicity of core frequency, the CF-Index is space-efficient and normally smaller than the original graph. Moreover, we develop a scalable index construction algorithm, which gradually updates the core frequencies by propagation in the graph until convergence.

Skyline algorithm (Section V). We further study the problem of finding skyline (k, t, f)-cores. By revealing the intrinsic relation between skyline (k, t, f) triples and core frequencies stored in the CF-Index, we develop an efficient algorithm to obtain skyline (k, t, f) triples directly from the CF-Index. When faced with an unfamiliar temporal graph, the obtained skyline (k, t, f) triples can help to determine the valid range of each query parameter regarding the other two.

Experimental evaluation (Section VI). We perform compre-

hensive experiments on both real-world and artificial datasets. Our algorithms for frequency computation, index construction, and single/skyline query processing achieve several orders of magnitude improvement over the respective baselines. Moreover, both statistics and case study demonstrate that, time and topology are "intertwined" in temporal graphs, and the vertices in more cohesive subgraphs usually have more frequent interactions. Thus, the (k,t,f)-core query can find more cohesive subgraphs of the k-core effectively and efficiently.

II. Problem Formulation

A *temporal graph* can be represented by $\mathcal{G}=(\mathcal{V},\mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ represent the set of vertices and the set of indirected temporal edges between the vertices, respectively. Each *temporal edge* in $\mathcal{E}$ can be represented by a triple (u,v,T), where u and v are the vertices in $\mathcal{V}$ and T is an ascending list of distinct integer timestamps. It indicates that u and v have interactions at those moments. Figure 1 shows a temporal social graph as our running example. The temporal edge between the vertices v_1 and v_2 is represented by $(v_1,v_2,[1,5,10])$.

We represent the weight of a temporal edge by an inherent time-dependent metric, namely, (interaction) *frequency*. For static graphs, there exist a variety of metrics of edge weight, such as the distance in road networks. In contrast to these time-independent metrics, the frequency $F(u,v)=F(T)=\frac{n}{t_n-t_1+1}\in(0,1]$ of a temporal edge (u,v,T) with $T=[t_1,t_2,\cdots,t_n]$ $(n\geq 1)$ naturally measures how intimately two vertices interact from the time perspective.

In addition to the frequency that represents the relative number of interactions divided by time, we also consider the absolute number (namely, regularity). It is to overlook the edges with only a few timestamps, because they usually represent occasional relationships. Thereby, we propose the following adjusted metric.

Definition 1 (t-Frequency)**.** *Given a temporal edge (u,v,T) with $T=[t_1,t_2,\cdots,t_n]$ $(n\geq 1)$ and an integer $t\geq 1$, the t-frequency $F_t(u,v)$ between u and v is defined as the highest frequency of at least t interactions recorded in T. Formally, we have $F_t(u,v)=F_t(T)=\max\{F(T')|T'\subseteq T,|T'|\geq t\}$.*

Example 1. *In Figure 1, for the temporal edge $(v_7,v_8,[2,5,9,13])$, the sublists of timestamps whose sizes are no less than 3 are $[2,5,9]$, $[2,5,13]$, $[5,9,13]$, $[2,9,13]$, and $[2,5,9,13]$, and their frequencies are 0.375, 0.25, 0.333, 0.25, and 0.333, respectively. As a result, $F_3(v_7,v_8)=F_3([2,5,9,13])=F([2,5,9])=0.375$.*

With the t-frequency, we can identify which vertices have intimate enough mutual relationships by the following concept.

Definition 2 ((t,f)-Neighbor)**.** *Given a temporal graph $\mathcal{G}$, an integer $t\geq 1$, and a float $f\in[0,1]$, two vertices $u,v\in\mathcal{V}$ are mutually (t,f)-neighbors, if and only if there is a temporal edge $(u,v,T)\in\mathcal{E}$ between them such that $F_t(u,v)\geq f$. For convenience, we denote by $\mathcal{N}_{t,f}(u)$ the set of (t,f)-neighbors of the vertex u.*

Example 2. *In Figure 1, since $F_3(v_7,v_8)=0.375$, v_7 and v_8 are mutually (t,f)-neighbors if $t=3$ and $f\leq 0.375$.*

Then, we formulate a novel concept of cohesive subgraph called (k,t,f)-*core* for temporal graphs. This concept reshapes the traditional concept of (static) k-core in order to guarantee that each vertex in a cohesive subgraph has enough intimate neighbors in terms of interaction regularity and frequency. In other words, a vertex will not appear in the (k,t,f)-core if most of its neighbors have only occasional or infrequent interactions with it.

Definition 3 ((k,t,f)-Core)**.** *Given a temporal graph $\mathcal{G}$, an integer $k\geq 1$, an integer $t\geq 1$, and a float $f\in[0,1]$, the (k,t,f)-core (also called frequency-aware k-core) denoted by $\mathcal{C}^k_{t,f}$ is the maximal subgraph of $\mathcal{G}$ such that each vertex u in $\mathcal{C}^k_{t,f}$ has at least k (t,f)-neighbors, namely, $|\mathcal{N}_{t,f}(u)|\geq k$.*

Example 3. *In Figure 1, we remark several (k,t,f)-cores by dashed frames. Compared to the $(2,2,0.5)$-core, the other three cores increase the requirement of k, t, and f, respectively. We can see that, these query parameters can effectively find different cohesive subgraphs.*

Intrinsically, the (k,t,f)-core enhances the k-core by introducing two parameters t and f with respect to timestamps on edges. Obviously, the (k,t,f)-core is an even more cohesive subgraph of the k-core for temporal graphs. In particular, they are equivalent if $t=1$ and $f=0$.

We aim to address the problem of querying (k,t,f)-core in this paper, which returns the set of vertices in a specific (k,t,f)-core. Moreover, we also consider how to find the parameter triples (namely, (k,t,f)) of all skyline (k,t,f)-cores, in order to assist query writing.

III. Frequency Computation

In this section, we address a preliminary sub-problem of (k,t,f)-core query, namely, computing the t-frequency for a specific temporal edge (u,v,T). Straightforwardly calculating the frequency for each sublist of T no shorter than t incurs $\sum_{i=t}^{|T|}\binom{|T|}{i}$ times of computation, which is too costly for downstream indexing or querying tasks on temporal graphs with a great average number of timestamps on edges.

A. Minimum Slope Problem

To solve the computation of the t-frequency efficiently, we convert it to another equivalent minimum slope problem.

Firstly, we have an observation that a sublist of T can achieve the highest frequency only if its timestamps are consecutive in T.

Lemma 1. *Given a timestamp list $T=[t_1,t_2,\cdots,t_n]$ $(n\geq 1)$ and an integer $t\geq 1$, let $T'\subseteq T$ be a sublist no shorter than t whose frequency is equal to the t-frequency $F_t(T)$, there exists an integer $i\in[1,n-t+1]$ such that $T'=[t_i,t_{i+1},\cdots,t_{i+|T'|-1}]$.*

Proof. All proofs in this paper can be found at https://github.com/graphlab-whu/KTFC/blob/main/proof.pdf. □

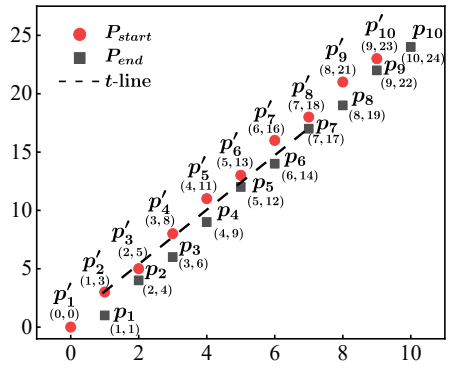


Fig. 3: The illustration of two point sets and a t-line between them for the example temporal edge (v_9, v_{12}, T).

Due to Lemma 1, we can overlook the inconsecutive sublists of T and only need to compute the frequency $\sum_{i=1}^{n-t+1}(n - i - t + 2)$ times, which is still exponential.

In order to further reduce the time complexity, we map the timestamps in T to points in a Cartesian coordinate system. The x-axis represents the indexes of the timestamps in T, and the y-axis represents the values of the timestamps. In particular, each timestamp t_i in T is mapped to two points, namely, $p_i = (i, t_i)$ and $p'_i = (i-1, t_i - 1)$. The set of points $p_1, p_2, \cdots, p_n$ is referred to the end point set denoted by P_{end}, and in contrast, the set of points $p'_1, p'_2, \cdots, p'_n$ is referred to the start point set denoted by P_{start}.

Then, the frequency of any consecutive sublist of T can be represented by the slope of a particular line that connects two points in P_{start} and P_{end} respectively. Consider a line between $p'_i = (i-1, t_i - 1) \in P_{start}$ and $p_j = (j, t_j) \in P_{end}$ with $j \geq i$, its slope is denoted by $\mathrm{slp}(p'_i, p_j) = \frac{t_j - t_i + 1}{j - i + 1}$, which is exactly the reciprocal of the frequency of a consecutive sublist $[t_i, t_{i+1}, \cdots, t_j]$ of T. Thus, the t-frequency of a specific temporal edge can be obtained by computing the slope of each line between p'_i and p_j with $j - i + 1 \geq t$, which is called **t-line**.

Lemma 2. *Given a timestamp list $T = [t_1, t_2, \cdots, t_n]$ $(n \geq 1)$ and an integer $t \geq 1$, $F_t(T)$ is the reciprocal of the minimum slope of all t-lines between $p'_i = (i-1, t_i - 1) \in P_{start}$ and $p_j = (j, t_j) \in P_{end}$ with $j \geq i$.*

Example 4. *Figure 3 shows the two point sets P_{start} and P_{end} for $T = [1, 4, 6, 9, 12, 14, 17, 19, 22, 24])$. Take the t-line (p'_2, p_7) as an example ($t \leq 6$). The reciprocal of its slope is the frequency of the sublist $T' = [4, 6, 9, 12, 14, 17]$: $\frac{1}{\mathrm{slp}(p'_2, p_7)} = \frac{7-1}{17-3} = \frac{|T'|}{17-4+1}$.*

B. Linear-Time Solution

To address the minimum slope problem for t-lines, we propose a linear-time algorithm. Given the start point set P_{start} = $\{p'_1, p'_2, \cdots, p'_n\}$ and the end point set P_{end} = $\{p_1, p_2, \cdots, p_n\}$, we compute the minimum slope $\mathrm{mslp}(\cdot, p_j) = \min\{\mathrm{slp}(p'_i, p_j) | 1 \leq i \leq j - t + 1\}$ for the group of t-lines that connect each end point p_j, so that the minimum value of $\mathrm{mslp}(\cdot, p_j)$ for $1 \leq j \leq n$ is obviously the result. Moreover, the most important optimization is to address the sub-problems, namely, computing $\mathrm{mslp}(\cdot, p_j)$ incrementally. The optimization is based on the following step-by-step observations.

Observation 1. We introduce the *Upper Convex Curve* (UCC) on start points denoted by $\mathrm{UCC}_j \subseteq P_{start}$ for a specific end point $p_j \in P_{end}$, which serves as a pruned solution space of computing $\mathrm{mslp}(\cdot, p_j)$. Among the start points p'_1, p'_2, $\cdots$, p'_{j-t+1} that can be connected to the end point p_j by t-lines, only the slopes of the t-lines from the ones in UCC_j could be the minimum. In other words, any start point p'_i such that $\mathrm{slp}(p'_i, p_j) = \mathrm{mslp}(\cdot, p_j)$ must be in UCC_j.

Definition 4 (Upper Convex Curve)**.** *Given an ordered list of start points p'_1, p'_2, $\cdots$, p'_{j-t+1} that can be connected to the end point p_j by t-lines, the upper convex curve UCC_j for p_j is a sublist p'_{c1}, p'_{c2}, $\cdots$, p'_{cm} such that $\mathrm{slp}(p'_{ci-1}, p'_{ci}) \geq \mathrm{slp}(p'_{ci}, p'_{ci+1})$ for $c1 < ci < cm$. Intuitively, assume there exist a curve connecting the start points in UCC_j consecutively, all the other start points in the given list lie below this curve.*

Lemma 3. *For an end point p_j and an integer t, any start point p'_i with $1 \leq i \leq j - t + 1$ such that $\mathrm{slp}(p'_i, p_j) = \mathrm{mslp}(\cdot, p_j)$ must satisfy that $p'_i \in \mathrm{UCC}_j$.*

Observation 2. To compute the minimum slope $\mathrm{mslp}(\cdot, p_j)$ for a given end point p_j, we only need to traverse the start points in UCC_j until the "inflection point" is found. Assume p'_l be an *Optimal Start Point* (OSP) such that $\mathrm{slp}(p'_l, p_j) = \mathrm{mslp}(\cdot, p_j)$. For the start points in UCC_j before p'_l (including p'_l), the slopes of the t-lines from them to p_j decrease monotonically in traversal order. While, for the start points in UCC_j after p'_l (including p'_l), the slopes of the t-lines from them to p_j increases monotonically in traversal order. Thus, the traversal can be stopped early if the current slope is greater than the previous slope.

Lemma 4. *The slopes of the t-lines from points in UCC_j to endpoint p_j decreases monotonically first and then increases. (In boundary cases, it may be monotonically decreasing or increasing).*

Observation 3. For an end point p_j, we can only use the UCC of the previous end point (namely, UCC_{j-1}) to obtain its UCC (namely, UCC_j). Because the start points not in UCC_{j-1} are surely not in UCC_j. In other words, once a start point is pruned from the UCC for any end point, it is permanent for the rest end points.

Lemma 5. *Those start points other than p'_{j-t+1} that do not exist in UCC_{j-1} cannot exist in UCC_j.*

Observation 4. For an end point p_j, we only need to start the traversal on UCC_j from the previous OSP of p_{j-1} but not from the first start point. Because the start points in UCC_{j-1} before the OSP of p_{j-1} must not be the OSP of p_j. It can be leveraged to further prune the start points in UCC_j, so that each start point will be traversed only once in the procedure

Algorithm 1: Linear-time t-frequency computation

Input: An ascending timestamp list T, an integer $t \geq 1$
Output: $F_t(T)$

1 initialize a start point list UCC, $mslp \leftarrow \infty$;
2 **for** $j \leftarrow t$ **to** $|T|$ **do**
3 append p'_{j-t+1} at the end of UCC;
  // Maintain the UCC and OSP
4 **while** $|UCC| > 1$ *and* $\text{slp}(UCC[|UCC|], p'_{j-t+1}) \geq \text{slp}(UCC[|UCC|-1], UCC[|UCC|])$ **do**
5 remove the last point from UCC; // Lemma 3
6 **while** $|UCC| > 1$ *and* $\text{slp}(UCC[1], p_j) > \text{slp}(UCC[2], p_j)$ **do**
7 remove the first point from UCC; // Lemma 6
8 $mslp \leftarrow \min(mslp, \text{slp}(UCC[1], p_j))$;// UCC[1] is the OSP for p_j
9 **return** $1/mslp$;

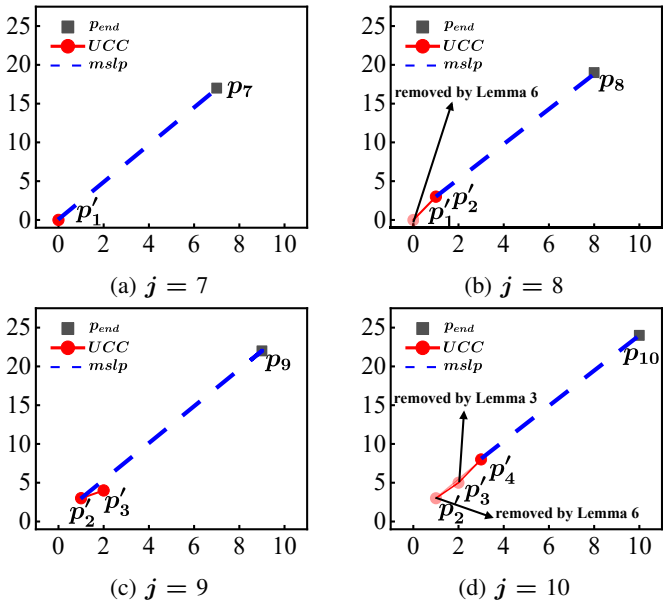
of computing $\text{mslp}(\cdot, p_j)$ for all $1 \leq j \leq n$.

Lemma 6. *When iterating to a new end point p_j, points before its OSP can not be the start point to obtain the minimum slope, which means that they should be removed from UCC.*

Algorithm. Based on the above observations, we design an efficient algorithm to compute the minimum slope for all t-lines. Specifically, we enumerate the end points iteratively in ascending time order and maintain the UCCs and OSPs for them. In each iteration, we continue to enumerate the start points in the current UCC from the previous OSP until the new OSP is identified. Lastly, the minimum slope from the OSPs to the corresponding end points is returned.

The pseudo code is given in Algorithm 1. It uses a double-ended queue UCC to maintain UCC_j for each end point p_j and a variable $mslp$ to record the latest minimum slope (Line 1). When an end point p_j is enumerated, the start point p'_{j-t+1} is added to UCC (Line 3). Recalling Lemma 3, we need to remove some points after the addition of p'_{j-t+1} to maintain the upper convex curve (Lines 4-5). Then, we find the OSP for p_j by scanning the start points in UCC and can stop early due to the monotonicity of slope (Lemma 4). Meanwhile, we can keep removing points that are not OSP until we find it based on Lemma 6 (Lines 6-7). We update $mslp$ at the end of each iteration (Line 8), and eventually return the reciprocal of $mslp$ as the t-frequency $F_t(T)$ (Line 9).

Example 5. *Figure 4 shows the procedure of computing the 7-frequency of $T = [1, 4, 6, 9, 12, 14, 17, 19, 22, 24]$ by Algorithm 1. We start to enumerate the end point p_j from $j = 7$. For $j = 7$, the start point p'_1 is added into* UCC_7. *Obviously, the OSP is p'_1, and we have* $\text{mslp}(\cdot, p_7) = \text{slp}(p'_1, p_7) = \frac{17-0}{7-0} = 2.43$. *Thus, the minimum slope $mslp$ is updated to 2.43. For $j = 8$, p'_2 is added into* UCC_8 *copied from* UCC_7. *Since* $\text{slp}(p'_2, p_8) = \frac{19-3}{8-1} = 2.29 < \text{slp}(p'_1, p_8) = \frac{19-0}{8-0} = 2.38$, *$p'_2$ is the OSP in* UCC_8, *and thereby p'_1 is removed from* UCC_8 *by Lemma 6. Now $mslp$ is updated to 2.29. For $j = 9$, p'_3 is added into* UCC_9 *copied from* UCC_8. *and we can calculate that p'_2 is still the OSP in* UCC_9 *similarly. Thus, $mslp$ remains the same. For $j = 10$, p'_4 is added into* UCC_{10} *copied from* UCC_9. *Since* $\text{slp}(p'_2, p'_3) = \frac{5-3}{2-1} = 2 < \text{slp}(p'_3, p'_4) = \frac{8-5}{3-2} = 3$, *$p'_3$ does not satisfy the upper convex condition and is removed from* UCC_{10} *by Lemma 3. Then, we can calculate that p'_4 is the OSP in* UCC_{10} *(namely,* $\text{mslp}(\cdot, p_{10}) = \text{slp}(p'_4, p_{10}) = \frac{24-8}{10-3} = 2.29$*). Thus, p'_2 is removed from* UCC_{10} *by Lemma 6, and $mslp$ remains the same. Finally, the 7-frequency is the reciprocal of $mslp$, namely,* $\frac{1}{2.29} = 0.437$.



(a) $j = 7$ (b) $j = 8$ (c) $j = 9$ (d) $j = 10$

Fig. 4: The procedure of computing the 7-frequency of the example temporal edge (v_9, v_{12}, T) by Algorithm 1.

Complexity. Let us analyze the complexity of Algorithm 1. Since each start point or end point is enumerated at most once, the time complexity of Algorithm 1 is $O(|T|)$, where T is the given timestamp list of a temporal edge. Moreover, the space complexity of Algorithm 1 is also $O(|T|)$, as the only data structure UCC has each start point enqueued only once. Consequently, the computation of t-frequency for temporal edges can be accomplished in linear time and space, and thus can scale well on temporal graphs in which vertices have intensive interactions, such as social networks, communication networks, transaction networks, etc.

IV. Index and Query Processing

In this section, we propose both index-free and index-based algorithms to address the (k, t, f)-core query problem. The index-free algorithm extends the classic core decomposition algorithm [24] by filtering edges with respect to frequency. Since the time complexity of core decomposition is $O(|\mathcal{E}|)$, the index-free algorithm is not efficient on large-scale graphs. Therefore, we introduce a novel concept called "core frequency", and design a compact index structure on top of that, so that a specific (k, t, f)-core can be efficiently obtained from the index.

Algorithm 2: Online frequency-aware k-core query.

Input: $\mathcal{G}$, k, t, f
Output: (k,t,f)-core

```
obtain G' = (V', E') from G by [24];
for (u, v, T) ∈ E' do
    if |T| < t or F_t(T) < f then
        remove (u, v, T) from E';
obtain G'' = (V'', E'') from G' by [24];
return V'';
```

Algorithm 3: CF-Index query

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, k, t, f
Output: (k,t,f)-core

```
C^k_{t,f} ← ∅;
for u ∈ V do
    obtain cf(u, k, t) from the CF-Index;
    if cf(u, k, t) ≥ f then
        C^k_{t,f} ← C^k_{t,f} ∪ {u};
return C^k_{t,f}
```

A. Online Query Algorithm

The online query algorithm is composed of three steps: 1) obtain the static k-core by core decomposition, 2) remove the temporal edges whose t-frequency is less than f, and 3) obtain the (k,t,f)-core by core decomposition again. The rationale is to avoid to compute the t-frequency for all edges. The pseudo code of the online algorithm is given in Algorithm 2.

Its correctness is guaranteed by the following observation.

Lemma 7. *Given a temporal graph $\mathcal{G}$, $\mathcal{C}^k_{t',f'} \subseteq \mathcal{C}^k_{t,f}$ if $t' \geq t$ and $f' \geq f$.*

According to Definition 3, the (k,t,f)-core reduces to the k-core if $t = 1$, so that the (k,t,f)-core is a subgraph of the k-core for $\mathcal{C}^k_{t,f} \subseteq \mathcal{C}^k_{1,f}$.

The worst case time complexity of Algorithm 2 is $O(|\mathcal{E}| \cdot |T|_{avg})$, where $|T|_{avg}$ is the average number of timestamps on edges. As k increases, the actual time cost will decrease. Notably, although precomputing the core number of each vertex and storing it in the index can accelerate the initial core decomposition process, this approach does not reduce the theoretical time complexity of the overall algorithm. This is because the time complexity is mainly determined by the second step, as detailed in the experiment in Section VI-C.

B. Index-based Query Algorithm

In order to improve query efficiency, we propose an index that can answer a specific (k,t,f)-core query without searching the graph.

Key Idea. Lemma 7 implies that the (k,t,f)-core gradually has vertices removed with the increase of f if k and t are fixed. In other words, a vertex will only disappear in the (k,t,f)-core and never reappear as f increases.

Inspired by the above observation, we introduce a concept "core frequency", which is similar to the existing concepts core number (namely, the maximum value of k such that a given vertex is contained by the k-core) and core time [4](namely, the earliest end time t_e such that a given vertex is contained by the historical k-core during the period $[t_s, t_e]$ for given k and start time t_s).

Definition 5 (Core Frequency)**.** *Given integers k and t, the core frequency of a vertex v is the maximum value of f such that v is contained by the (k,t,f)-core, which is denoted by $\mathrm{cf}(v,k,t)$. Formally, there exists no float $f > \mathrm{cf}(v,k,t)$ such that $v \in \mathcal{C}^k_{t,f}$. In particular, $\mathrm{cf}(v,k,t) = -1$ if v is not contained by the (k,t,f)-core for any $f \in [0,1]$.*

Example 6. *Consider the vertex v_4 in Figure 1. Given $k = 3$ and $t = 2$, the maximum value of f such that v_3 is contained by the (k,t,f)-core is 0.5, namely, $\mathrm{cf}(v_4,k,t) = 0.5$. For $f > 0.5$, v_3 cannot exist in the (k,t,f)-core.*

Index Design. Based on the core frequency, we propose a novel index called *Core Frequency Index* (CF-Index). The purpose is to precompute and store the core frequency of all vertices for each possible value of k and t. Thus, in order to determine whether a vertex v is contained by the (k,t,f)-core, we only need to retrieve $\mathrm{cf}(v,k,t)$ from the index and compare it with f. If $\mathrm{cf}(v,k,t) \geq f$, we have $v \in \mathcal{C}^k_{t,f}$.

The storage format of CF-Index is shown in Figure 5. Firstly, the CF-Index has a partition for each k. In the partition of a specific k, there is a record for each vertex $v \in \mathcal{V}$, which stores the tuples $(t, \mathrm{cf}(v,k,t))$ in ascending order of t. For example, in the partition of $k = 2$, the record of the vertex v_5 contains five mappings for t = 1, 2, 3, 4, and 5, respectively.
Observation. Moreover, it is observed that, the core frequency $\mathrm{cf}(v,k,t)$ decreases monotonically with the increase of t in each record for specific k and v.

Lemma 8. *Given a vertex v and an integer k, we have $\mathrm{cf}(v,k,t) \geq \mathrm{cf}(v,k,t')$ if $t < t'$.*

Index Optimization. Due to the monotonicity of core frequency, we can remove the duplicated core frequencies in each record to reduce the space overhead of CF-Index without losing information. Specifically, only the tuple $(t, \mathrm{cf}(u,k,t))$ with the minimum t will be stored for each distinct $\mathrm{cf}(u,k,t)$. For example, in the record of the vertex v_5 in the partition of $k = 1$, the two tuples $(2,1)$ and $(5,0.5)$ can be removed. Then, to retrieve $\mathrm{cf}(v_5,1,5)$, we only need to find the maximum $t \leq 5$ (namely, 4) in the record and return the corresponding core frequency $\mathrm{cf}(v_5,1,4) = 0.5$.
Query Processing. Algorithm 3 leverages the CF-Index to deal with the (k,t,f)-core query. It scans the partition of given k. For each record of a specific vertex u, it retrieves the core frequency $\mathrm{cf}(u,k,t)$ by a binary search and then determines whether u should appear in the result by comparing $\mathrm{cf}(u,k,t)$ and f.
Complexity. The time complexity of Algorithm 3 is bounded by $O(|\mathcal{V}| \cdot \log |T|_{avg})$, which is surely lower than that of Algorithm 2 as $|\mathcal{V}| << |\mathcal{E}|$. Moreover, the space complexity

of CF-Index is bounded by $O(k_{max} \cdot |\mathcal{V}| \cdot |T|_{avg})$, where k_{max} is the maximum value of k for a specific graph.

C. Index Construction

The construction of CF-index is a non-trivial task. Let us consider a naive implementation firstly. Initially, for each temporal edge, we compute its t-frequency f for each possible value of t. Then, for each combination of k and (t, f) pair, we obtain the corresponding (k, t, f)-core by Algorithm 2. Lastly, for each vertex u in each (k, t, f)-core, we use (t, f) to update the record of u in the partition of k. Obviously, this naive implementation involves a huge amount of redundant computation and is not scalable.

Pipeline. In order to reduce redundant computation, we propose a propagation algorithm that avoids to induce any (k, t, f)-core and directly produces the distinct core frequencies of vertices for each k and t. Specifically, we have the following three nested steps.

Step 1. For each $t \in [2, |T|_{max}]$, we compute the t-frequency of each edge, and identify the set of edges E_t whose t-frequency has changed with respect to its $(t-1)$-frequency, since the change could lead to the change of core frequency of the vertices V_t linked by the edges in E_t. We use $C_t^k \subseteq \mathcal{V}$ to store the vertices whose core frequency $\mathrm{cf}(u, k, t)$ could change with respect to $\mathrm{cf}(u, k, t-1)$. Initially, we check each $u \in V_t$ and put those whose core frequency has changed into C_t^k for each k less than the core number of u.

Step 2. In an iteration of t, for each $k \in [1, k_{max}]$, we enumerate u from C_t^k until C_t^k is empty, and deal with it as follows. 1) We try to update $\mathrm{cf}(u, k, t)$ to a decreased upper bound that is derived only from the core frequencies of the neighbors of u. 2) We maintain a particular neighbor vertex set $N_t^k(u)$ for each vertex $u \in \mathcal{V}$ to determine whether the core frequency $\mathrm{cf}(u, k, t)$ has changed with respect to $\mathrm{cf}(u, k, t-1)$ in $O(1)$ time. 3) For each neighbor v of u, we add v into C_t^k if $\mathrm{cf}(v, k, t)$ has changed due to the change of $\mathrm{cf}(u, k, t)$.

Step 3. Once C_t^k has become empty, for each $u \in \mathcal{V}$, it is guaranteed that $\mathrm{cf}(u, k, t)$ has converged to the true value during the propagation in Step 2. Then, we put the core frequency $\mathrm{cf}(u, k, t)$ that has changed with respect to $\mathrm{cf}(u, k, t-1)$ into the record of u in the partition of k for the CF-Index according to Definition 5.

Observation 1. Our first key finding is an easy-to-compute condition to determine that $\mathrm{cf}(u, k, t)$ has changed with respect to $\mathrm{cf}(u, k, t-1)$. The core frequency of a vertex u actually depends on two factors. As t increases, the core frequency of the neighbors of u could decrease (Lemma 8), and the t-frequency between u and its neighbors also could decrease. Thereby, the neighbors could disappear in the $(k, t, \mathrm{cf}(u, k, t-1))$-core. As a result, u may not have enough $(t, \mathrm{cf}(u, k, t-1))$-neighbors to appear in the $(k, t, \mathrm{cf}(u, k, t-1))$-core, so that $\mathrm{cf}(u, k, t)$ could be less than $\mathrm{cf}(u, k, t-1)$.

Lemma 9. *Given k and t, let $N_t^k(u)$ be the set of neighbors of a vertex u such that for each $v \in N_t^k(u)$ we have $\mathrm{cf}(v, k, t) \geq \mathrm{cf}(u, k, t-1)$ and $F_t(u, v) \geq \mathrm{cf}(u, k, t-1)$. Consequently, $\mathrm{cf}(u, k, t) < \mathrm{cf}(u, k, t-1)$ if and only if $|N_t^k(u)| < k$.*

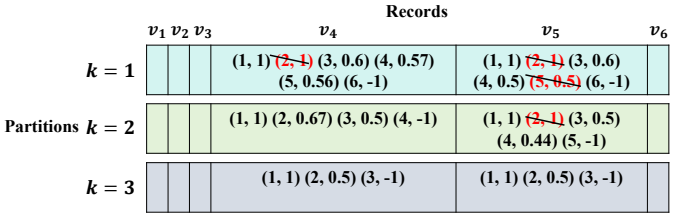


Fig. 5: The example of CF-Index structure.

Note that, when dealing with a vertex u for specific t and k in Step 2, $\mathrm{cf}(v, k, t)$ may not have been calculated for a neighbor v of u. If so, we assume that $\mathrm{cf}(v, k, t)$ is not changed (namely, is equal to the upper bound $\mathrm{cf}(v, k, t-1)$), so that there will be no false positive (namely, mistakenly determining that $\mathrm{cf}(u, k, t)$ has changed). In the meantime, there will be no false negative, because u could be added into C_t^k again whenever $\mathrm{cf}(v, k, t)$ has been updated later.

Observation 2. Taking Lemma 9 one step further, we have the following observation.

Lemma 10. *Given t and k, $\mathrm{cf}(u, k, t)$ equals to the maximum float $f \leq 1$ such that the number of neighbor vertices (denoted by v) of u, which satisfy $\mathrm{cf}(v, k, t) \geq f$ and $F_t(u, v) \geq f$, is no less than k.*

Based on Lemma 10, we can obtain $\mathrm{cf}(u, k, t)$ in a propagation way. Once u is enumerated from C_t^k, it means at least one neighbor of u cause the decrease of $\mathrm{cf}(u, k, t)$. Thereby, we recompute the maximum float f in Lemma 10 and update $\mathrm{cf}(u, k, t) = f$ accordingly. Intuitively, the value of $\mathrm{cf}(u, k, t)$ will gradually decrease and converge to its true value with the core frequencies of all neighbors of u have been updated to the true values.

Algorithm. Algorithm 4 gives the pseudo code of building the CF-Index. Firstly, we perform the core decomposition [24] to obtain the core number of all vertices (Line 1). Then, for each vertex u and each $k \leq core(u)$, we perform three initialization operations (Lines 2-7). 1) We add $(1, 1)$ to $Record_k(u)$, which can answer the (k, t, f)-core query for $t = 1$ (namely, the original k-core query). 2) We set the core frequency $\mathrm{cf}(u, k, 1) = 1$. 3) We initialize the neighbor set N_1^k as the neighbors of u in the corresponding k-core.

Then, for each $t \geq 2$ in ascending order, we gradually update $\mathrm{cf}(u, k, t)$ by propagation. Initially, we keep $\mathrm{cf}(u, k, t)$ and $N_t^k(u)$ of t as the same as that of $t-1$ for each possible k and u because $\mathrm{cf}(u, k, t) \leq \mathrm{cf}(u, k, t-1)$ (Lines 10-11). As t increases, the t-frequencies of a set of edges E_t have changed with respect to their $(t-1)$-frequencies, which probably causes the change of the core frequencies of the vertices linked by them. Therefore, as a seed, each linked vertex u has $N_t^k(u)$ maintained (Lines 12-17). For an edge $(u, v, T) \in E_t$, if $F_t(u, v) < \mathrm{cf}(u, k, t)$ (or $\mathrm{cf}(v, k, t)$), u (or v) is removed from $N_t^k(v)$ (or $N_t^k(u)$) (Line 29). After that, if $\mathrm{cf}(u, k, t)$ (or $\mathrm{cf}(v, k, t)$) is determined to be changed (Lemma 9), u (or v) is added into C_t^k, namely, activated for propagation (Lines 30-31). Thus, we may have a number of

Algorithm 4: CF-Index construction

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Output: CF-Index of $\mathcal{G}$

1 get the core number $core(u)$ of each vertex $u \in \mathcal{V}$ by [24];
2 **for** $k \leftarrow 1$ **to** k_{max} **do**
3 **for** $u \in \mathcal{V}$ **do**
4 **if** $core(u) \geq k$ **then**
5 insert $(1, 1)$ to $Record_k(u)$;
6 $\mathrm{cf}(u, k, 1) \leftarrow 1$;
7 $N_1^k(u) \leftarrow \{v | core(v) \geq k, (u, v, T) \in \mathcal{E}\}$;
8 get E_t for each $t \in [2, |T|_{max}]$ by Algorithm 1;
9 **for** $t \leftarrow 2$ **to** $|T|_{max}$ **do**
10 $\mathrm{cf}(u, k, t) \leftarrow \mathrm{cf}(u, k, t-1)$ for all u and k;
11 $N_t^k(u) \leftarrow N_{t-1}^k(u)$ for all u and k;
12 **for** $(u, v, T) \in E_t$ **do**
13 **for** $k \leftarrow 1$ **to** $\min\{core(u), core(v)\}$ **do**
14 **if** $u \in N_t^k(v)$ *and* $F_t(u, v) < \mathrm{cf}(v, k, t)$ **then**
15 `maintain(u, v, k, t)`;
16 **if** $v \in N_t^k(u)$ *and* $F_t(u, v) < \mathrm{cf}(u, k, t)$ **then**
17 `maintain(v, u, k, t)`;
18 **for** $k \leftarrow 1$ **to** k_{max} **do**
19 **while** $C_t^k \neq \emptyset$ **do**
20 $u \leftarrow C_t^k$.pop();
21 $\mathrm{cf}(u, k, t) \leftarrow$ `updateCF`(u, k, t);
22 update $f = \mathrm{cf}(u, k, t)$ of (t, f) in $Record_k(u)$;
23 **for** *each neighbor* v *of* u **do**
24 **if** $\mathrm{cf}(v, k, t) \geq \mathrm{cf}(u, k, t)$ *and* $F_t(u, v) \geq \mathrm{cf}(u, k, t)$ **then**
25 add v to $N_t^k(u)$;
26 **if** $u \in N_t^k(v)$ *and* $\mathrm{cf}(u, k, t) < \mathrm{cf}(v, k, t)$ **then**
27 `maintain(u, v, k, t)`;
28 **Function** `maintain`(u, v, k, t)**:**
29 remove u from $N_t^k(v)$;
30 **if** $|N_t^k(v)| < k$ *and* $v \notin C_t^k$ **then**
31 add v to C_t^k;

activated vertices other than the seeds in each C_t^k of different k. For each k, we continue to enumerate the vertices in C_t^k (Line 20), and use the `updateCF` function (Algorithm 5) to calculate the new (upper bound of) core frequency according to Lemma 10 (Line 21). If $\mathrm{cf}(u, k, t)$ has changed, there must be $(t, f) \in Record_k(u)$, so that we update f to the new $\mathrm{cf}(u, k, t)$ (Line 22). Moreover, for each neighbor v of u, we check whether v can be added into $N_t^k(u)$ since $\mathrm{cf}(u, k, t)$ has decreased (Lines 24-25), and whether $\mathrm{cf}(v, k, t)$ has changed (Lines 26-27). When C_t^k becomes empty, the update of (t, f) in all records in the partition of k is finished.

Algorithm 5 gives the pseudo code of updating the (upper bound of) core frequency of a vertex u for given k and t. Firstly, we traverse each neighbor v of u, and put the lesser of $\mathrm{cf}(v, k, t)$ and $F_t(u, v)$ into a list F (Lines 1-4). If the length of F is less than k, u cannot appear in the (k, t, f)-core even

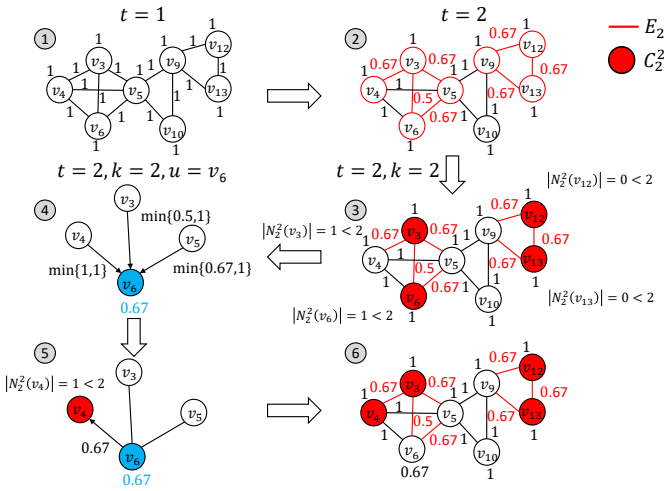


Fig. 6: An example of propagation process in Algorithm 4.

Algorithm 5: updateCF(u, k, t)

1 $F \leftarrow \emptyset$;
2 **for** *each neighbor* v *of* u **do**
3 **if** $\mathrm{cf}(v, k, t) > 0$ *and* $F_t(u, v) \neq null$ **then**
4 $F.add(\min\{\mathrm{cf}(v, k, t), F_t(u, v)\})$;
5 **if** $|F| \geq k$ **then**
6 **return** *The k-th largest value in the set* F;
7 **else**
8 **return** *-1*;

if $f = 0$, so that we return -1 as the core frequency. Otherwise, we return the k-th greatest value in F as the new upper bound of $\mathrm{cf}(u, k, t)$ according to Lemma 10.

Example 7. *Figure 6 shows an example of propagation process in Algorithm 4. 1) When $t = 1$, the core frequencies of all vertices and the t-frequencies of all edges are 1. 2) When $t = 2$, the t-frequencies of the edges in red color decreases, which have been stored in E_2 in advance. 3) For $k = 2$, the neighbor sets $N_2^2(v_3)$, $N_2^2(v_4)$, $N_2^2(v_5)$, $N_2^2(v_6)$, $N_2^2(v_9)$, $N_2^2(v_{12})$, and $N_2^2(v_{13})$ of linked vertices should be updated. For example, since $F_t(v_3, v_6)$ and $F_t(v_5, v_6)$ decrease to be less than $\mathrm{cf}(v_6, 2, 2) = 1$, v_3 and v_5 are removed from $N_2^2(v_6)$, and there is only v_4 left in $N_2^2(v_6)$, which means v_6 should be added to C_2^2 for $|N_2^2(v_6)| < 2$. As a result, the vertices in red color are added into C_2^2 as seeds. 4) Assume v_6 be popped from C_2^2 firstly. We update $\mathrm{cf}(v_6, 2, 2)$ by Algorithm 5. Intuitively, the neighbors of v_6 pass the values $\min\{\mathrm{cf}(v_3, 2, 2), F_t(v_3, v_6)\}$=0.5, $\min\{\mathrm{cf}(v_4, 2, 2), F_t(v_4, v_6)\}$=1, $\min\{\mathrm{cf}(v_5, 2, 2), F_t(v_5, v_6)\}$ =0.67 to v_6, and then $\mathrm{cf}(v_6, 2, 2)$ is aggregated to the 2nd greatest value 0.67. 5) Since $\mathrm{cf}(v_6, 2, 2)$ is decreased, the new core frequency is passed to the neighbors of v_6 whose neighbor sets contain v_6, so that we update $N_2^2(v_4)$ due to $v_6 \in N_2^2(v_4)$. 6) After that, v_6 has been removed from $N_2^2(v_4)$, and thereby v_4 is added to C_2^2 for $|N_2^2(v_4)| < 2$.*

Complexity. The time complexity of Algorithm 4 has two parts. During preprocessing, the t-frequencies of all temporal edges need to be calculated for each t, so the complexity of preprocessing is roughly $O(|\mathcal{E}| \cdot |T|^2_{avg})$ while using Algorithm 1 to compute t-frequency. The overall time complexity of a single propagation over all vertices is $O(\mathcal{E})$. However, for specific t and k, the times of propagation necessary before convergence is not known theoretically. It is observed that generally only a few core frequency updates for each vertex are sufficient to stop the algorithm in our experiments. Thus, the overall time complexity of Algorithm 4 can be estimated as $O(|\mathcal{E}| \cdot |T|^2_{avg} + |\mathcal{E}| \cdot |T|_{max} \cdot k_{max} \cdot c)$, where c is a small variable with respect to specific temporal graphs.

V. Skyline Query Algorithm

In this section, we propose an efficient algorithm based on the CF-Index to find the skyline (k,t,f)-cores for a given temporal graph. There are two applications of the skyline query algorithm. Firstly, we can obtain the Pareto optimal subgraphs of k-cores in the sense of interaction frequency. Secondly, based on the results, we can suggest valid query parameters (namely, k, t, and f) adaptively, thereby assisting query writing. The formal definition of skyline (k,t,f)-core is as follows.

Definition 6 (Skyline (k,t,f)-core)**.** *Given a temporal graph $\mathcal{G}$, a (k,t,f)-core is a skyline (k,t,f)-core if and only if there does not exist another (k',t',f')-core that dominates it, namely, $k' \geq k$, $t' \geq t$, and $f' \geq f$ hold at the same time.*

Baseline. In order to find the skyline (k,t,f)-cores, we can simply enumerate all possible (k,t,f) triples, and obtain the corresponding (k,t,f)-cores by Algorithm 3. However, the time complexity is obviously too high. While SkylineComm2D [29] incorporated pruning mechanisms based on this framework, its computational efficiency remains constrained by the substantial overhead from repeated invocations of Algorithm 3 during query processing. As a result, the efficiency remains relatively low.

Observations. Interestingly, since the CF-Index preserves the greatest value of f (namely, the maximum core frequency $\mathrm{cf}(u,k,t)$ of all vertices $u \in \mathcal{V}$) for all possible k and t, we can directly derive the values of k, t, and f for all skyline (k,t,f)-cores from the CF-Index. Specifically, we have the following observations.

Lemma 11. *For any skyline (k,t,f)-core, f is a core frequency of at least one vertex, and thus is preserved in the CF-Index.*

Lemma 11 guarantees the completeness of obtaining skyline (k,t,f)-cores from the CF-Index.

Lemma 12. *For any skyline (k,t,f)-core, we have $t = t' - 1$ for a pair (t',f') preserved in the partition of k.*

Lemma 12 indicates the possible positions of f (namely, the corresponding t) in the CF-Index.

Algorithm 6: Skyline (k,t,f) triple query algorithm

```
Input: G = (V, E)
Output: All skyline (k,t,f) triples of G
1  initialize S_k for each k;
2  for k ← 1 to k_max do
3      for u ∈ V do
4          f' ← 0;
5          for (t,f) ∈ Record_k(u) do
6              if S_k.get(t−1) = null and t > 1 then
7                  S_k.set(t−1, f'); // Lemma 12
8              else
9                  if S_k.get(t−1) < f' then
10                     S_k.set(t−1, f');
11             f' ← f;
12     f' ← 0;
13     for (t,f) ∈ S_k do
14         if f = f' then
15             delete the previous pair from S_k;
16         f' = f; // case 1 in Lemma 13
17 for k ← 2 to k_max do
18     for (t,f) ∈ S_k do
19         if S_{k−1}.get(t) ≠ null and S_{k−1}.get(t) ≤ f then
20             delete (t,f) from S_{k−1};
               // case 2 in Lemma 13
```

Based on these two observations, we only need to scan the CF-Index once to find the candidates of non-dominated (k,t,f) triples, and then we can remove the candidates that do not meet the skyline requirements.

Algorithm. The pseudo code of the skyline query is given in Algorithm 6. We initialize a number of map structures S_k that store (t,f) key-value pairs in order of t for each k (Line 1). Then, we scan the CF-Index to obtain the skyline (k,t,f) triple candidates based on Lemma 11 and 12 (Lines 2-16). For each partition of $k \in [1, k_{max}]$, we traverse the records of each vertex $u \in \mathcal{V}$, and collect induced (t,f) pairs only if they are not dominated by others. For that, we only keep the greatest f for each t in S_k (Lines 8-10). Then, we compare each pair (t,f) only with the successive pair (t',f') in S_k due to Lemma 8, and remove (t,f) if $f = f'$ (Lines 12-16). Moreover, We still need to remove the (t,f) pairs that violate the global skyline requirement, namely, are dominated by others in different S_k (Lines 17-20). An important observation is that, due to the hierarchical containment of k-core (Lemma 13), we only need to check whether (t,f) in S_k is dominated by (t,f') in S_{k+1} but not all other pairs.

Lemma 13. *For two candidate triples (k,t,f) and (k',t',f') obtained from the CF-Index by Lemma 12 (Lines 2-11 in Algorithm 6), (k,t,f) is dominated by (k',t',f') in two cases: 1) $k = k'$, $t < t'$, $f = f'$, or 2) $k < k'$, $t = t'$, $f = f'$.*

Note that, Algorithm 6 only returns the parameter triples of skyline (k,t,f)-cores. It can be extended to return the vertex

TABLE I: The statistics of datasets. k_{max} is the maximum core number of vertices. $|T|_{max}$ and $|T|_{avg}$ are the maximum and average timestamp number of edges, respectively.

Name	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	$\|\mathcal{E}\| \times \|T\|_{avg}$	k_{max}	$\|T\|_{max}$	$\|T\|_{avg}$	unit
Enron	184	3,125	125,235	16	1183	40.1	sec
Email	986	16,064	332,334	34	4,878	20.7	sec
Trade	255	34,211	507,497	132	31	14.8	year
SocialEvo	74	4,486	2,099,519	56	13,690	468.0	sec
Contacts	694	79,530	2,426,860	98	2,428	30.5	sec
WikiTalk	1,140,149	2,787,967	7,833,140	124	1,561	2.8	sec
DBLP	1,824,701	20,766,016	29,487,744	286	41	1.42	year
Contacts∗	10,675	1,045,591	23,041,562	155	2,428	22.1	sec
WikiTalk+	1,140,149	2,787,967	68,867,052	124	1,561	24.7	sec
DBLP+	1,824,710	20,766,016	209,736,761	124	41	10.1	year

sets of skyline (k, t, f)-cores.

Complexity. Since Algorithm 6 sequentially scans and filters (t, f) pairs in the CF-Index three times, its time complexity is equal to the space complexity of the CF-Index, namely, $O(k_{max} \cdot |\mathcal{V}| \cdot |T|_{avg})$.

VI. Experiments

In this section, we perform comprehensive experiments to verify the effectiveness and efficiency of our proposed algorithms on a Windows machine equipped with an Intel Core i7 CPU at 2.30GHz and 32GB of RAM. Each algorithm is implemented in C++ and compiled using the g++ compiler with -O3 optimization.

A. Dataset

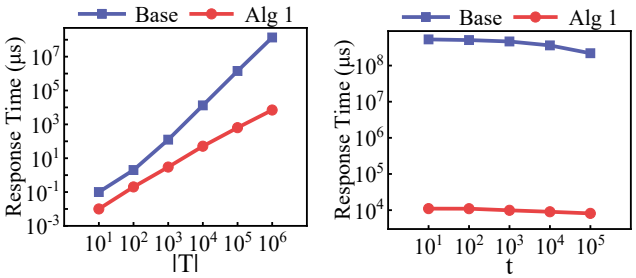
We select seven real-world temporal social network datasets and two artificial datasets. Email and WikiTalk are from SNAP [30]. DBLP is from KONECT [31]. Enron, Trade, SocialEvo, and Contacts are from TGX [32, 33]. Contacts∗ is generated by the DyGraph generator [34] to expand the size with respect to the original distribution. WikiTalk+ and DBLP+ are generated by adding more random timestamps to edges of WikiTalk and DBLP.

The basic information about the datasets is presented in Table I. Note that, many datasets used in previous studies only have a small average number of timestamps on each edge ($|T|_{avg}$), because they are sampled from the original graphs under certain rules. In contrast, we focus on more "real" datasets in which vertices interact more frequently, like at least dozens of times on average.

B. Efficiency of Frequency Computation

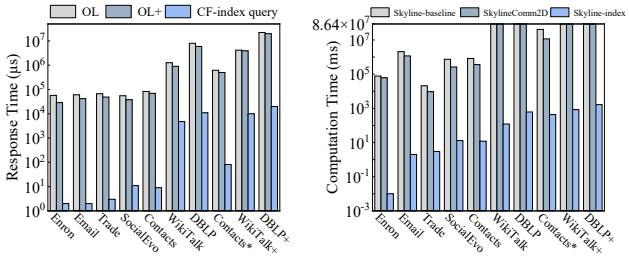
To evaluate the efficiency of Algorithm 1, we compare it with a baseline algorithm that enumerates all possible consecutive sublits of T no shorter than t. We generate 100 random timestamp lists for each group of selected values of t and $|T|$. The mean response time of two competitors are shown in Figure 7.

Our observations are as follows. 1) Algorithm 1 outperforms the baseline significantly, which confirms the effectiveness of UCC and OSP based pruning. 2) When t is fixed, the response time of Algorithm 1 increases non-exponentially with the increase of $|T|$. When $|T|$ is fixed, Algorithm 1 is not sensitive to the variation of t. Both confirm the linear time complexity of Algorithm 1.



(a) $t = \frac{|T|}{2}$ (b) $|T| = 10^6$

Fig. 7: The efficiency of t-frequency computation for T.



(a) Single query. (b) Skyline query.

Fig. 8: Query efficiency on various datasets.

C. Efficiency of Query Processing

Firstly, we compare the response time of OL (online query Algorithm 2), OL+ that improves OL by precomputing the core number of each vertex, and the CF-Index query (Algorithm 3). For each dataset, we randomly generate 20 groups of query parameters (namely, k, t, and f) and report the average response time in Figure 8a. With the help of the skyline query algorithm, each query parameter is guaranteed to be in the valid range. The advantage of the CF-Index query is quite significant. It outperforms the online query by 2∼5 orders of magnitude. Especially on the Enron dataset, the average response time is reduced from 57,182 μs to just 2 μs.

Moreover, we examine the query parameter sensitivity of the CF-Index query on two real-world datasets.

Impact of $\boldsymbol{k}$. For three groups of t and f values, we vary k from 2 to 10 in steps of 2. As shown in Figures 9a and 9b, the response time decreases gradually as k increases. The reason is that the records of vertices are smaller in the partition of greater k.

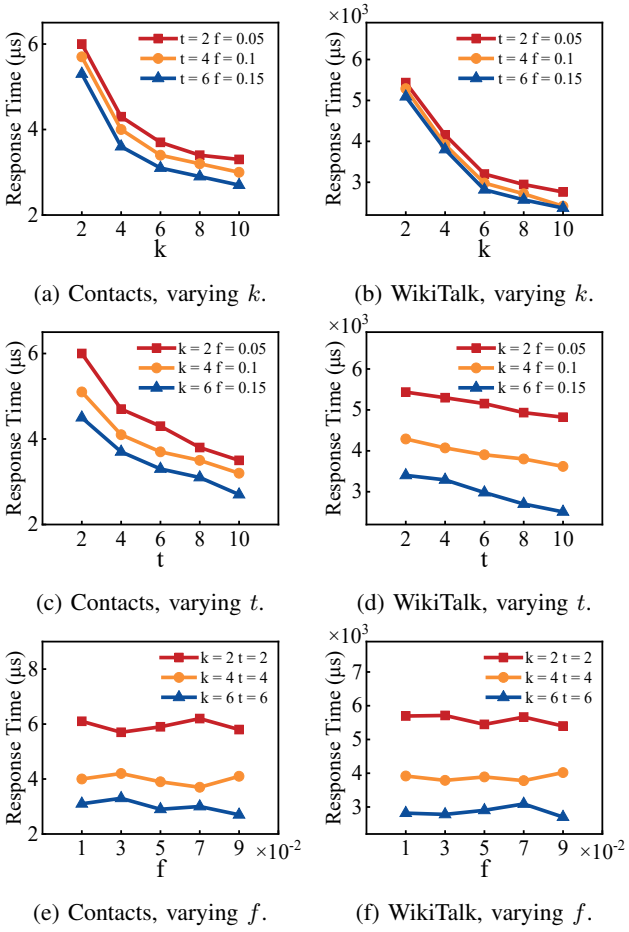
Impact of $\boldsymbol{t}$. For three groups of k and f values, we vary t from 2 to 10 in steps of 2. As shown in Figures 9c and 9d, the response time decreases gradually as t increases. Because as t grows, it will exceed the maximum value of t in more and more records, so that the binary search in such records can be skipped.

Impact of $\boldsymbol{f}$. For three groups of k and t values, we vary f from 0.01 to 0.09 in steps of 0.02. As shown in Figures 9e and 9f, there is no explicit trend of the response times, because f does not affect the complexity of Algorithm 3.

Lastly, we compare the skyline (k, t, f)-core query algo-

TABLE II: The statistics of CF-Index construction.

Dataset	Baseline (ms)	Alg. 4 CF-Index construction (ms)			CFreq	Graph Size	Index Size	Percentage
		Preproc (Base | Alg. 1)	Indexing	Total	#	(MB)	(MB)	(%)
Enron	2,955,267	312 | 6 (↓ 98%)	64	70	9,381	1.66	0.15	9%
Email	31,865,189	4,822 | 109 (↓ 97%)	860	969	75,070	5.07	1.28	25%
Trade	15,053,121	14 | 7 (↓ 50%)	1,129	1,136	247,880	5.62	2.44	43%
SocialEvo	>86,400,000	537,988 | 1,292 (↓ 99%)	1,047	2,339	148,207	15.98	2.28	14%
Contacts	>86,400,000	20,171 | 306 (↓ 98%)	20,709	21,015	651,820	16.00	10.20	63%
WikiTalk	>86,400,000	7,268 | 3,790 (↓ 48%)	337,508	341,298	7,497,253	93.09	636.07	676%
DBLP	>86,400,000	12,109 | 7,167 (↓ 41%)	398,285	405,452	24,754,718	449.95	2299.15	511%
Contacts∗	>86,400,000	92,687 | 4,731 (↓ 95%)	19,518,265	19,522,996	21,091,198	362.68	280.77	77%
WikiTalk+	>86,400,000	39,074 | 8,225 (↓ 79%)	3,618,957	3,627,182	34,855,744	1480.98	1341.42	91%
DBLP+	>86,400,000	61,712 | 17,896 (↓ 71%)	7,172,034	7,189,930	112,891,483	4018.23	3696.77	92%



(a) Contacts, varying k. (b) WikiTalk, varying k.

(c) Contacts, varying t. (d) WikiTalk, varying t.

(e) Contacts, varying f. (f) WikiTalk, varying f.

Fig. 9: Query efficiency under varying parameters.

rithm (Algorithm 6) with the baseline algorithm and SkylineComm2D mentioned in Section V. Their response time is presented in Figure 8b. The CF-Index based skyline algorithm is at least three orders of magnitude faster than the baseline algorithm and SkylineComm2D. Even for the largest DBLP+ dataset, it only spends less than two seconds. It is worth mentioning that the response time of Algorithm 6 on different datasets is almost proportional to the size of the corresponding CF-Index (in Table II), since Algorithm 6 primarily involves traversing the CF-Index.

D. Overhead of Index Construction

Firstly, let us observe the construction time shown in Table II. The baseline algorithm mention in Section IV-C is obviously too costly, and even cannot stop in 24 hours for more than half datasets. In contrast, the propagation style Algorithm 4 is much more efficient, and stops in less than 1 hour on almost all datasets except Contact*, which incurs massive upper bound updates because of the heavily skewed distribution of timestamps on edges. More specifically, the construction time of Algorithm 4 has two parts. The preprocessing part can be reduced significantly by using Algorithm 1 due to its superiority on frequency computation. The indexing part normally costs more time than preprocessing, except on SocialEvo that has an extraordinarily great value of $|T|_{avg}$.

We also report the space overheads of the constructed CF-Index in Table II. The CF-Index is smaller than the original temporal graph for almost all datasets except WikiTalk and DBLP, whose $|T|_{avg}$ is very few. By comparing CFreq# (denoting the total number of core frequencies stored in the CF-Index) and $|\mathcal{E}| \times |T|_{avg}$ in Table I, it is verified that the CF-Index effectively reduces the space overhead by leveraging the discreteness of core frequency variation with respect to t. Therefore, the CF-Index can scale to large-scale temporal graphs.

E. Application of Skyline Algorithm

Faced with unfamiliar temporal graphs, it is difficult to find a valid combination of query parameters that ensures to return non-empty results immediately, and thus we need the help of skyline (k,t,f)-core query (Algorithm 6). Figure 10 shows the surfaces fitted from the skyline (k,t,f) triples returned by Algorithm 6 on Email. Thus, we can gradually select query parameters in the feasible space under the skyline surface. Take the Email dataset as an example. Firstly, if we set $k = 12$, the valid range of t is $[1,9]$ and f is $[0,1]$. Subsequently, if we set $t = 3$, the valid range of f is $[0,0.5]$, or if we set $f = 0.2$, the valid range of t is $[1,3]$. There are totally 224 skyline cores for the Email dataset. Moreover, we report the sizes of some skyline cores in the above table for observation.

F. Effectiveness of (k,t,f)-Core Query

From the perspective of statistics, a strong evidence of effectiveness is the improvement on metrics that evaluate the

k	2		3		4	
t	3	4	2	3	2	3
f	0.6	0.09	1	0.08	0.16	0.02
size	4	4	22	5	52	6

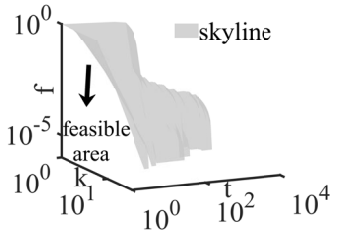
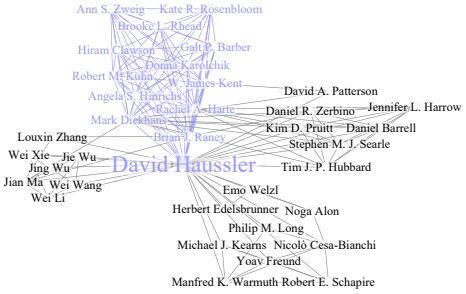
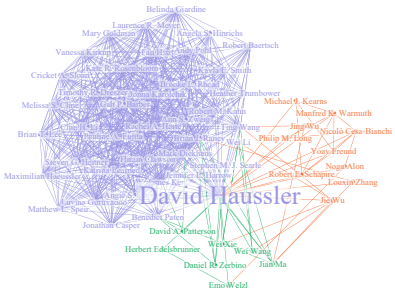


Fig. 10: The surface of skyline (k, t, f) triples on Email.



(a) (9,4,0)-core vs (9,7,0)-core.



(b) (7,5,0)-core vs (7,5,0.5)-core vs (7,5,1)-core.

Fig. 11: The comparison of ego-networks of a same author (David Haussler) in different (k, t, f)-cores on DBLP.

quality of the (k, t, f)-core as a community with respect to the k-core. As shown in Figure 2, the heat maps clearly indicate that there are incremental improvement on four classical metrics with the increase of t or f if k is fixed. The internal density $\frac{2\times|\mathcal{E}|}{|\mathcal{V}|\times(|\mathcal{V}|-1)}$ is the most straightforward measurement of the connection tightness within a specific community. The clustering coefficient $\frac{3\times triangle(\mathcal{G})}{triplet(\mathcal{G})}$ measures the tendency of vertices to be clustered together, where $triangle(\mathcal{G})$ and $triplet(\mathcal{G})$ are the numbers of triangles and triplets in $\mathcal{G}$ respectively. Moreover, the average value of the distances between all pairs of vertices and the number of communities obtained by the Louvain algorithm [35] in $\mathcal{G}$ are also common metrics. Note that, the first two metrics are the greater the better, and the last two metrics are the less the better.

Moreover, we conduct a case study on DBLP, which is a temporal graph of coauthorship. Figure 11a shows the difference between two ego-networks of a same author David Haussler in the (9,4,0)-core and the (9,7,0)-core respectively. We can see that there are four separate communities in the (9,4,0)-core, while the (9,7,0)-core extracts one from them with the increase of t from 4 to 7. Then, we observe the effectiveness of parameter f. Figure 11b shows the ego-networks in the (7,5,0)-core, (7,5,0.5)-core, and (7,5,1)-core. After f is increased to 0.5, a part of vertices (in orange color) are removed. When f is further increased to 1, more vertices (in green color) are removed. Eventually, the most closely connected vertices are left.

In conclusion, the above observations demonstrate that, time and topology are "interwound" in temporal graphs, and the vertices in more cohesive subgraphs usually have more frequent interactions. Conversely, the (k, t, f)-core query can indeed help to find more cohesive subgraphs of the k-core.

VII. Related Work

There are several typical query conditions for temporal graphs: 1) filtering edges by timestamp [4, 5, 36–44], 2) restricting the order of timestamps on directed edges of a path or triangle [2, 11, 45–47], and 3) constraining the duration or time pan of a subgraph [10, 28, 48–51]. Different from them, we focus on a new query condition with respect to the interaction frequency between vertices.

As the most similar query model, the (k, h)-core [3] requires that there are at least h interactions between each pair of vertices. However, this restriction does not really take time into consideration. In contrast, we adopt a novel time-dependent metric, namely, t-frequency, which considers both the number of interactions and the time span over them. Although the previous work [52] aims to identify frequently occurring patterns in temporal graphs, the subject of frequency is the subgraph but not the edge.

Moreover, as an ordinary time-dependent concept, the frequency is widely used in the fields of temporal graph representation learning, time series data analytics, etc. TF-GCL [53] is a frequency-aware graph contrastive learning framework for temporal networks to address challenges in capturing temporal dynamics and differentiating between low and high frequency vertices. FEDformer [54] demonstrates how (both low and high) frequency components contribute to long-term forecasting by capturing global trends and critical events in data. TS-TFC [55] is a temporal-frequency co-training framework, which leverages the complementary strengths of time-domain and frequency-domain representations for semi-supervised time series learning. These studies have also inspired our research.

VIII. Conclusion

In this paper, we propose the first frequency-aware k-core query model on temporal graphs, and design a compact core frequency index to address the query. Moreover, a collection of efficient algorithms are developed for frequency computation, index construction, and single/skyline query processing, respectively. Our experiments verify that, the frequency-aware k-core query indeed improve the cohesiveness of the result subgraph significantly, and the algorithms achieve several orders of magnitude optimization on the respective tasks.

Acknowledgment

This work was supported by the Natural Science Foundation of Hubei Province (No. JCZRYB202500322), the NSFC (No. 61202036, 62272353, and 62276193), and the RGC of Hong Kong (No. 14205520).

REFERENCES

[1] H. Wu, J. Cheng, Y. Ke, S. Huang, Y. Huang, and H. Wu, "Efficient algorithms for temporal path computation," *TKDE*, vol. 28, no. 11, pp. 2927–2942, 2016.

[2] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *PVLDB*, vol. 7, no. 9, pp. 721–732, 2014.

[3] H. Wu, J. Cheng, Y. Lu, Y. Ke, Y. Huang, D. Yan, and H. Wu, "Core decomposition in large temporal graphs," in *Big Data*, 2015, pp. 649–658.

[4] M. Yu, D. Wen, L. Qin, Y. Zhang, W. Zhang, and X. Lin, "On querying historical k-cores," *PVLDB*, vol. 14, no. 11, pp. 2033–2045, 2021.

[5] J. Yang, M. Zhong, Y. Zhu, T. Qian, M. Liu, and J. X. Yu, "Scalable time-range k-core query on temporal graphs," *PVLDB*, vol. 16, no. 5, p. 1168–1180, Jan. 2023.

[6] J. Yang, M. Zhong, Y. Zhu, T. Qian, M. Liu, and J. X. Yu, "Evolution forest index: Towards optimal temporal k-core component search via time-topology isomorphic computation," *PVLDB*, vol. 17, no. 11, pp. 2840–2853, 2024.

[7] 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, χ)-core query," *TKDE*, vol. 36, no. 7, pp. 2831–2845, 2024.

[8] Y. Li, J. Liu, H. Zhao, J. Sun, Y. Zhao, and G. Wang, "Efficient continual cohesive subgraph search in large temporal graphs," *World Wide Web*, vol. 24, pp. 1483–1509, 2021.

[9] H. Qin, R.-H. Li, Y. Yuan, G. Wang, L. Qin, and Z. Zhang, "Mining bursting core in large temporal graphs," *PVLDB*, vol. 15, no. 13, p. 3911–3923, 2022.

[10] C. Hu, M. Zhong, Y. Zhu, T. Qian, T. Yu, H. Chen, M. Liu, and J. X. Yu, "Querying cohesive subgraph regarding span-constrained triangles on temporal graphs," in *ICDE*, 2024, pp. 3338–3350.

[11] N. Pashanasangi and C. Seshadhri, "Faster and generalized temporal triangle counting, via degeneracy ordering," in *KDD*, 2021, p. 1319–1328.

[12] P. Liu, A. R. Benson, and M. Charikar, "Sampling methods for counting temporal motifs," in *WSDM*, 2019, p. 294–302.

[13] G. Xue, M. Zhong, J. Li, J. Chen, C. Zhai, and R. Kong, "Dynamic network embedding survey," *Neurocomputing*, vol. 472, pp. 212–223, 2022.

[14] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart, "Representation learning for dynamic graphs: a survey," *J. Mach. Learn. Res.*, vol. 21, no. 1, 2020.

[15] T. Zhang, Y. Gao, J. Zhao, L. Chen, L. Jin, Z. Yang, B. Cao, and J. Fan, "Efficient exact and approximate betweenness centrality computation for temporal graphs," in *Proceedings of the ACM on Web Conference*, 2024, pp. 2395–2406.

[16] T. Zhang, J. Fang, Z. Yang, B. Cao, and J. Fan, "Tatkc: A temporal graph neural network for fast approximate temporal katz centrality ranking," in *Proceedings of the ACM on Web Conference 2024*, 2024, pp. 527–538.

[17] T. Chakraborty, A. Dalmia, A. Mukherjee, and N. Ganguly, "Metrics for community analysis: A survey," *ACM Comput. Surv.*, vol. 50, no. 4, 2017.

[18] F. Zhang, C. Li, Y. Zhang, L. Qin, and W. Zhang, "Finding critical users in social communities: The collapsed core and truss problems," *TKDE*, vol. 32, no. 1, pp. 78–91, 2020.

[19] C. Li, L. Wang, S. Sun, and C. Xia, "Identification of influential spreaders based on classified neighbors in real-world complex networks," *Applied Mathematics and Computation*, vol. 320, pp. 512–523, 2018.

[20] J. Jiang, Y. Li, B. He, B. Hooi, J. Chen, and J. K. Z. Kang, "Spade: A real-time fraud detection framework on evolving graphs," *PVLDB*, vol. 16, no. 3, pp. 461–469, 2022.

[21] F. Liu, Z. Li, B. Wang, J. Wu, J. Yang, J. Huang, Y. Zhang, W. Wang, S. Xue, S. Nepal, and Q. Z. Sheng, "eriskcom: an e-commerce risky community detection platform," *The VLDB Journal*, vol. 31, no. 5, p. 1085–1101, 2022.

[22] J. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani, "K-core decomposition of internet graphs: hierarchies, self-similarity and measurement biases," *Networks and Heterogeneous Media*, vol. 3, 12 2005.

[23] C. Chen, Q. Zhu, R. Sun, X. Wang, and Y. Wu, "Edge manipulation approaches for k-core minimization: Metrics and analytics," *TKDE*, vol. 35, no. 1, pp. 390–403, 2023.

[24] V. Batagelj and M. Zaversnik, "An o(m) algorithm for cores decomposition of networks," *ArXiv*, vol. cs.DS/0310049, 2003.

[25] S. Li, K. Wang, X. Lin, W. Zhang, Y. He, and L. Yuan, "Querying historical cohesive subgraphs over temporal bipartite graphs," in *ICDE*, 2024, pp. 2503–2516.

[26] Y. Fang, R. Cheng, S. Luo, and J. Hu, "Effective community search for large attributed graphs," *PVLDB*, vol. 9, no. 12, pp. 1233–1244, 2016.

[27] C. Luo, Y. Zhu, Q. Liu, Y. Gao, L. Chen, and J. Xu, "Mcr-tree: An efficient index for multi-dimensional core search," *Proc. ACM Manag. Data*, vol. 2, no. 3, 2024.

[28] E. Galimberti, M. Ciaperoni, A. Barrat, F. Bonchi, C. Cattuto, and F. Gullo, "Span-core decomposition for temporal networks: Algorithms and applications," *TKDD*, vol. 15, no. 1, Dec. 2020.

[29] R.-H. Li, L. Qin, F. Ye, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng, "Skyline community search in multi-valued networks," in *SIGMOD*, 2018, p. 457–472.

[30] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[31] J. Kunegis, "Konect: the koblenz network collection," in *WWW*, 2013, pp. 1343–1350.

[32] R. Shirzadkhani, S. Huang, E. Kooshafar, R. Rabbany,

and F. Poursafaei, "Temporal graph analysis with tgx," *WSDM*, 2024.
[33] F. Poursafaei, S. Huang, K. Pelrine, and R. Rabbany, "Towards better evaluation for dynamic link prediction," in *NeurIPS*, 2022.
[34] A. McCrabb, H. H. Nigatu, A. Getachew, and V. Bertacco, "Dygraph: a dynamic graph generator and benchmark suite," *Proceedings of the 5th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, 2022.
[35] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, oct 2008.
[36] R.-H. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai, "Persistent community search in temporal networks," in *ICDE*, 2018, pp. 797–808.
[37] L. Li, Y. Zhao, Y. Li, F. Wahab, and Z. Wang, "The most active community search in large temporal graphs," *Knowledge-Based Systems*, vol. 250, p. 109101, 2022.
[38] L. Chu, Y. Zhang, Y. Yang, L. Wang, and J. Pei, "Online density bursting subgraph detection from temporal graphs," *PVLDB*, vol. 12, no. 13, p. 2353–2365, 2019.
[39] A. Tian, A. Zhou, Y. Wang, X. Jian, and L. Chen, "Efficient index for temporal core queries over bipartite graphs," *PVLDB*, vol. 17, pp. 2813–2825, 2024.
[40] Y. Chen, J. Jiang, S. Sun, B. He, and M. Chen, "Rush: Real-time burst subgraph detection in dynamic graphs," *PVLDB*, vol. 17, no. 11, pp. 3657–3665, 2024.
[41] I. Sarpe, F. Vandin, and A. Gionis, "Scalable temporal motif densest subnetwork discovery," in *SIGKDD*, 2024, pp. 2536–2547.
[42] J. M. da Trindade, J. Shun, S. Madden, and N. Tatbul, "Kairos: Efficient temporal graph analytics on a single machine," *arXiv preprint arXiv:2401.02563*, 2024.
[43] Z. Chen, F. Zhang, Y. Chen, X. Fang, G. Feng, X. Zhu, W. Chen, and X. Du, "Enabling window-based monotonic graph analytics with reusable transitional results for pattern-consistent queries," *PVLDB*, vol. 17, no. 11, pp. 3003–3016, 2024.
[44] Z. Wang, M. Zhong, Y. Zhu, T. Qian, M. Liu, and J. X. Yu, "On more efficiently and versatilely querying historical k-cores," *PVLDB*, vol. 18, no. 5, pp. 1335–1347, 2025.
[45] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke, "Reachability and time-based path queries in temporal graphs," in *ICDE*, 2016, pp. 145–156.
[46] Y. Jin, Z. Chen, and W. Liu, "Enumerating all multi-constrained s-t paths on temporal graph," *Knowl. Inf. Syst.*, vol. 66, no. 2, p. 1135–1165, 2023.
[47] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *SIGMOD*, 2014, p. 1311–1322.
[48] L. Lin, P. Yuan, R.-H. Li, and H. Jin, "Mining diversified top-r lasting cohesive subgraphs on temporal networks," *IEEE Transactions on Big Data*, vol. 8, no. 6, pp. 1537–1549, 2022.
[49] Y. Zeng, H. Qin, R.-H. Li, K. Wang, G. Wang, and X. Lin, "Mining quasi-periodic communities in temporal network," in *ICDE*, 2024, pp. 2476–2488.
[50] H. Qin, R.-H. Li, Y. Yuan, G. Wang, W. Yang, and L. Qin, "Periodic communities mining in temporal networks: Concepts and algorithms," *TKDE*, vol. 34, no. 8, pp. 3927–3945, 2022.
[51] H. Qin, R.-H. Li, G. Wang, L. Qin, Y. Cheng, and Y. Yuan, "Mining periodic cliques in temporal networks," in *ICDE*, 2019, pp. 1130–1141.
[52] Q. Zhang, D. Guo, X. Zhao, L. Yuan, and L. Luo, "Discovering frequency bursting patterns in temporal graphs," in *ICDE*, 2023, pp. 599–611.
[53] S. Tan, J. You, and D. Li, "Temporality- and frequency-aware graph contrastive learning for temporal network," in *CIKM*, 2022, p. 1878–1888.
[54] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, "FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting," in *ICML*, 2022, pp. 27 268–27 286.
[55] Z. Liu, Q. Ma, P. Ma, and L. Wang, "Temporal-frequency co-training for time series semi-supervised learning," in *AAAI*, 2023.