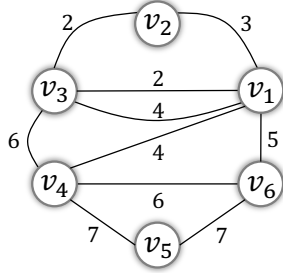
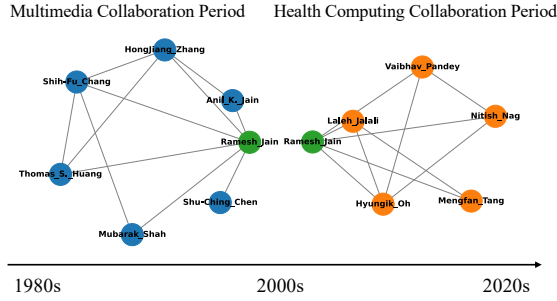


**Figure 15: Illustrating the key properties of temporal  $k$ -cores on timestamp-based temporal graph and interval-based temporal graph.**



**Figure 16: An example of the temporal graph.**



**Figure 14: A case study on DBLP co-authorship network.**

**Table 9: Statistics of datasets: number of nodes ( $n$ ), edges ( $m$ ), maximum timestamp ( $t_{\max}$ ), and number of communities.**

Dataset	$n$	$m$	$t_{\max}$	# of communities
Patent	12,214	41,916	891	6
S-DBLP	28,085	236,894	27	10
School	327	188,508	7,375	9

## A MORE DISCUSSIONS

### A.1 Case study

We analyze the DBLP co-authorship network by using AIT to identify research groups of “Ramesh Jain”, a renowned computer scientist in the multimedia area. Specifically, we set  $k = 5$ , the query vertex  $q$  to “Ramesh Jain”, and the query time interval to  $[1980, 2020]$ .

As shown in Figure 14, our TDC model captures Jain’s research evolution by identifying two distinct communities: one focused on multimedia from 1980-2000, and another interested in health computing from 2000-2020.

### A.2 Extended to the lifespan temporal graph

In the literature, there are two temporal graph models, which are timestamp-based and interval-based temporal graphs, and they associate each edge with a timestamp [9, 49] and a time interval (lifespan) [100, 101], respectively. Since a single timestamp can be considered a time interval whose start and end timestamps are the same, the interval-based temporal graph can be regarded as an extension of the timestamp-based temporal graph, implying that the timestamp-based temporal graph is more fundamental. Actually, our proposed solutions on timestamp-based temporal graph can also be easily extended to interval-based temporal graph.

Let us briefly explain the key idea of extension: First, we extend the  $k$ -core model on the timestamp-based temporal graph for the interval-based temporal graph. Given an interval-based temporal graph, its temporal  $k$ -core can be a subgraph, where each vertex has at least  $k$  neighbors and all the temporal edges have a non-empty intersected interval, denoting the valid time interval for this  $k$ -core. Afterwards, we can easily extend our proposed algorithms for identifying the communities from the interval-based temporal graph. Specifically, given a query time window, to find the TDC on an interval-based temporal graph, an online solution is to first enumerate all sub-windows within the given window, treating them as intersection intervals for constructing the corresponding  $k$ -core, then compute each duration, and finally select the temporal  $k$ -core with the longest duration as the TDC.

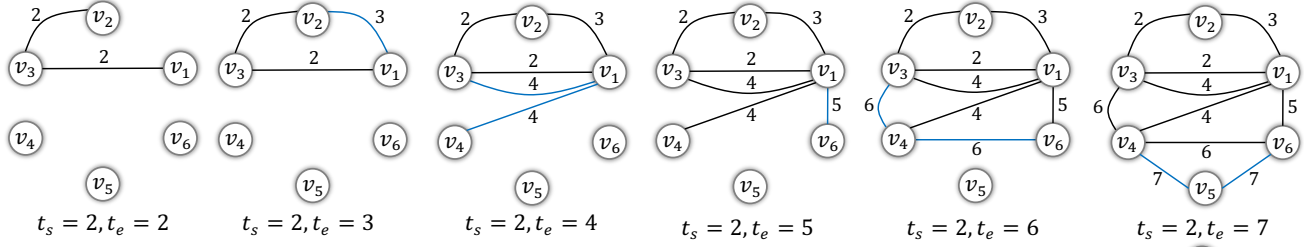
Regarding the index-based solution, recall that our original indexes rely on a key property of the temporal  $k$ -core on timestamp-based temporal graph: *Given a fixed start time  $t_s$  and an integer  $k$ , for all the temporal  $k$ -cores for the intervals  $[t_s, t_e]$ , when we increase  $t_e$  from  $t_s$  to  $t_{\max}$ , they can only add new edges and no edge deletes as  $t_e$  increases.* However, for interval-based temporal graphs, when considering the intersection interval  $[t_s, t_e]$  of all the edges in the temporal  $k$ -core, the set of edges can only shrink as  $t_e$  increases from  $t_s$  to  $t_{\max}$ . This is because only edges whose intervals fully cover the interval  $[t_s, t_e]$  are included, so some edges may drop out as the interval grows. To tackle this issue, we only need to slightly revise our original indexing algorithm to build the index. Specifically, we first enumerate  $t_e$  from  $t_{\max}$  to  $t_1$ , and then for each specific  $t_e$ , we decrease  $t_s$  from  $t_e$  to  $t_1$ , during which we progressively derive the  $k$ -cores on all the possible intervals. The above index can also efficiently support the TDC search.

For example, we illustrate the key properties of temporal  $k$ -cores on two types of temporal graphs in Figure 15. Consider the timestamp-based and interval-based temporal graphs shown in Figures 15(a) and (b), respectively. Fixing  $k = 2$  and  $t_s = 2$ , we increase  $t_e$  from 2 to 3. In the timestamp-based temporal graph, the temporal  $k$ -core in  $[2, 3]$  adds two edges and one vertex compared to  $[2, 2]$ . In contrast, in the interval-based temporal graph, the temporal  $k$ -core in  $[2, 3]$  removes two edges and one vertex compared to  $[2, 2]$ .

### A.3 An example of temporal $k$ -core

We would like to clarify that, for a temporal  $k$ -core with a fixed start time  $t_s$ , the temporal  $k$ -core can only be merged and never split as the end time  $t_e$  increases, unlike the general  $k$ -core in static

➤ **Projected Graph over each time interval with  $t_s = 2$**



➤ **Temporal 2-core over each time interval with  $t_s = 2$**

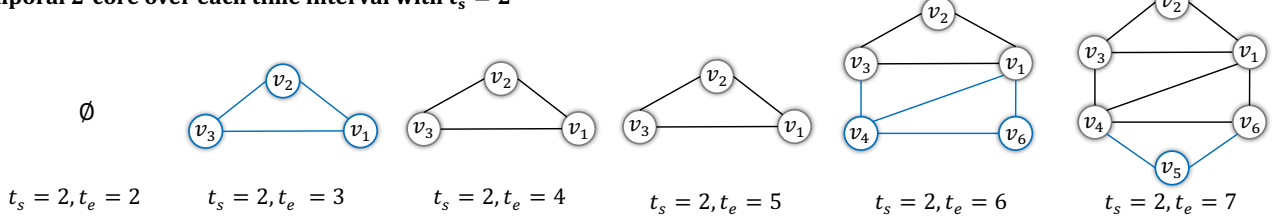


Figure 17: Projected graphs and their corresponding temporal 2-cores for each time window ( $t_s = 2$ ).

graphs. This property arises from the nature of the projected graph: when  $t_s$  is fixed, the projected graph over the time window  $[t_s, t_e]$  can only gain edges as  $t_e$  increases, but never loses any existing edges. Consequently, both vertices and edges are only ever added to the temporal  $k$ -core as  $t_e$  increases; there is no removal of vertices or edges from the core. To illustrate this, we provide a detailed example as follows:

**EXAMPLE 7.** Consider the temporal graph shown in Figure 16. We present a chain of projected graphs and their corresponding temporal  $k$ -cores for  $t_s = 2$  and  $t_e$  varying from 2 to 7 (i.e.,  $t_{max}$ ) in Figure 17. It can be observed that as  $t_e$  increases, new edges are incrementally added into the projected graph, and thus the corresponding temporal  $k$ -core either remains the same or grows larger by merging new vertices or edges (marked in blue color). No edges or vertices are ever deleted from the core during this process. This behavior directly follows from the definition of the temporal  $k$ -core on the projected graph.

## B ADDITIONAL PROOFS OF LEMMAS AND THEOREMS

**LEMMA 3.1** The total time cost of ONCE is  $O(t_{max}^2 \cdot \log t_{max} \cdot (n+m))$ , where  $n$ ,  $m$  and  $t_{max}$  denotes the number of vertices, edges and the number of distinct time slots in a temporal graph  $G$ , respectively.

**PROOF.** The proof of Lemma 3.1 is straightforward. There are at most  $t_{max}^2$  time windows to check. For each time window, ONCE requires  $O(m)$  time to compute the temporal  $k$ -core containing  $q$  and at most  $O(\log t_{max} \cdot (n+m))$  time to determine the duration. Therefore, the lemma holds.  $\square$

**LEMMA 4.4** The community duration of the temporal  $k$ -core  $S_k[t_l, t_r]$  can be calculated by using the active time of its key edge subtraction  $(t_r + 1)$ .

**PROOF.** We prove this by contradiction. Denote  $e^*$  as the key edge of  $S_k[t_s, t_e]$ . Assume there exists a time window  $[t_s, t'_e]$  such that  $\exists u \in S_k[t_s, t'_e] \wedge u \notin S_k[t_s, t_e]$ , where  $t'_e < t(e^*)$ . For an edge

$e' = (u, v)$  with  $v \in S_k[t_s, t_e]$ , we have  $\mathcal{L}_{t_s}(e')_k < t'_e < \mathcal{L}_{t_s}(e^*)_k$ , which contradicts the definition of the key edge. Hence, this lemma can be proved.  $\square$

**LEMMA 4.7** Given a temporal graph  $G$ , an integer  $k$ , an edge  $e = (u, v)$  and two start times  $t_s, t'_s$ , if  $t_s < t'_s$ , then  $\mathcal{L}_{t_s}(e)_k \leq \mathcal{L}_{t'_s}(e)_k$ .

**PROOF.** Given that  $t_s < t'_s$ , all edges in the projected graph of  $[t'_s, \mathcal{L}_{t_s}(e)_k]$  must exist in that of  $[t_s, \mathcal{L}_{t_s}(e)_k]$ . The core number of  $u$  and  $v$  over  $[t'_s, \mathcal{L}_{t_s}(e)_k - 1]$  is smaller than  $k$  based on Definition 4.1. Therefore, the lemma holds.  $\square$

**LEMMA 4.10** Given a temporal graph  $G$ , a start time  $t_s$ , and an integer  $k$ , the ASF-index of the updated graph  $G'$ , by decreasing the active times of some existing edges in  $G$ , is equivalent to directly inserting these edges into the ASF-index of  $G$ .

**PROOF.** Clearly, decreasing the weight of an edge is equivalent to first adding the edge with the new, lower weight and then removing the edge with the old, higher weight. Here, we denote the edges in  $E$  with lower weights as  $E'$ . Hence, the ASF-index after inserting  $E'$  into  $G$  is equivalent to inserting all edges from  $E'$  into  $\mathcal{F}[k, t]$ , as shown in lemma 4.9. When we delete all edges  $E$  from  $G$ , there are two types of edges in  $E$ : those not in the index and those in the index. For the former, deleting them does not affect the index. For the latter, during the insertion process, we have already deleted them by inserting the same edges with smaller weights.  $\square$

**LEMMA 4.12** The total time cost of ATGIC is  $O(k_{max} \cdot m \cdot \bar{c} \log n)$ , where  $\bar{c}$  denotes the average number of changes for the active time of an edge. Typically,  $\bar{c} \ll t_{max}$ .

**PROOF.** For each  $k$ , and start time  $t_s$  ( $t_s > 1$ ), ATGIC needs to insert all the edges that active times decreased and edges with the timestamps  $t_s$  into the index of the previous start time (i.e.,  $t_s - 1$ ). We use  $\bar{c}$  to denote the average number of changes for the active time of one edge, so there are at most  $((\bar{c} + 1) \cdot m)$  edges that would be inserted into the index. In addition, inserting an edge into the

index takes  $O(\log n)$  time [33, 67]. Hence, the total construction time is  $O(k_{\max} \cdot m \cdot \bar{c} \log n)$ .  $\square$

LEMMA 4.13 The ATG-index requires  $O(k_{\max} \cdot \bar{m})$  space, where  $\bar{m}$  represents the number of all edges in the index, which be calculated by multiplying the number of edges by the number of times each edge appears in the index. Note that  $\bar{m}$  is bounded by  $m \cdot \bar{c}$  and  $\bar{m} \ll n \cdot t_{\max}$ , as shown in our experimental results.

PROOF. For each  $k$ , our ATG-index is a graph with  $n$  vertices and  $\bar{m}$  edges ( $n < \bar{m}$ ), so the index costs  $O(k_{\max} \cdot \bar{m})$  memory. Note that when an edge's active time changes, it appears multiple times in the index. This indicates that the number of times each edge appears in the index is always less than or equal to  $\bar{c}$ .  $\square$

LEMMA 5.1 The total time cost of BIT is  $O(t_{\max} \cdot \min\{n, t_{\max}\} \cdot n)$ , where  $n$  denotes the number of vertices in a temporal graph  $G$ .

PROOF. There are at most  $t_{\max} \cdot \{n, t_{\max}\}$  time windows, since for each start time  $t_s$  only the timestamps of the edges in  $\mathcal{F}[t_s, k]$  need to be considered as the end time. For each time window, BIT needs  $O(n)$  time to calculate the temporal  $k$ -core containing  $q$  and its corresponding community duration in the index  $\mathcal{F}[t_l, k]$ . Hence, the above lemma holds.  $\square$

LEMMA 5.4 Given a temporal  $k$ -core  $S_k[t_l, t_r]$ , and its key edge  $e^* = (u, v)$ , assuming  $u \in S_k[t_l, t_r]$  and  $v \notin S_k[t_l, t_r]$ ,  $v$  must be included in  $S_k[t_l, t'_r]$ , if  $t'_r \geq \mathcal{L}_{t_l}(e^*)_k$ .

PROOF. We prove this lemma by contradiction. Suppose there exists an edge  $e^* = (u, v)$  and the end time  $t'_e$  satisfies the conditions, but  $v$  is not in  $S_k[t_s, t'_e]$ . This implies that the smallest active time required to include  $v$  in  $S_k[t_s, t'_e]$  is greater than  $t'_e$ , contradicting the initial assumption. Therefore, the lemma holds.  $\square$

THEOREM 5.5 The algorithm 5 can correctly return the TDC.

PROOF. To prove the correctness of algorithm 5, we only need to prove that for a fixed  $k$ , the  $S_k[t_s, t_e + 1]$  can be correctly calculated by  $S_k[t_s, t_e]$ . Supposing that there exists a vertex  $v \in S_k[t_s, t_e + 1]$  but not be included in it by our algorithm. That is to say, for an edge  $e^* = (u, v)$ ,  $\mathcal{L}_{t_s}(e^*)_k > t_e + 1$ , which contradicts the definition of active time. Hence, the above theorem holds.  $\square$

LEMMA 5.6 The total time cost of AIT is  $O(t_{\max} \cdot n + t_{\max} \cdot \min\{t_{\max}, n\} \cdot \log n)$ , where all variables are defined as in Lemma 3.1.

PROOF. First, computing MSF takes  $O(t_{\max} \cdot n)$  time. And then, for each start time  $t_s$ , AIT takes  $O(\min\{t_{\max}, n\} \cdot \log n)$  time to compute the temporal core with the largest duration for all time windows which using  $t_s$  as the start time. Specifically, for the start time  $t_s$ , there are at most  $\min\{t_{\max}, n\}$  end times in our index, and for each window, AIT can retrieve  $e^*$  from  $\mathcal{E}(k)$  in  $O(\log n)$  time. Hence, the above lemma holds.  $\square$