Durable Queries over Non-synchronized Temporal Data

Yanqi Xie
Xiamen University of Technology
Wei Weng
Xiamen University of Technology
Jianmin Li (✉ jianminli_xmut@sina.com)
Xiamen University of Technology

Research Article

Keywords: Durable Top-k, Query Processing, Non-synchronized Temporal, Data, Sweep Line Algorithm

Posted Date: October 10th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2120736/v1

Durable Queries over Non-synchronized Temporal Data

Yanqi Xie · Wei Weng · Jianmin Li

Received: / Accepted:

Abstract Temporal data are ubiquitous nowadays and efficient management of temporal data is of key importance. A temporal data typically describes the evolution of an object over time. One of the most useful queries over temporal data are the durable top-k queries. Given a time window, a durable top-k query finds the objects that are frequently among the best. Existing solutions to durable top-k queries assume that all temporal data are sampled at the same time points (i.e., at any time, there is a corresponding observed value for every temporal data). However, in many practical applications, temporal data are collected from multiple data sources with different sampling rates. In this light, we investigate the efficient processing of durable top-k queries over temporal data with different sampling rates. We propose an efficient sweep line algorithm to process durable top-k queries over non-synchronized temporal data. We conduct extensive experiments on two real datasets to test the performance of our proposed method. The results show that our methods outperforms the baseline solutions by a large margin.

Keywords Durable Top-k · Query Processing · Non-synchronized Temporal Data · Sweep Line Algorithm

Yanqi Xie
School of Computer and Information Engineering, Xiamen University of Technology
E-mail: yqxie@xmut.edu.cn

Wei Weng
School of Computer and Information Engineering, Xiamen University of Technology
E-mail: wwweng@xmut.edu.cn

Jianmin Li (corresponding author)
School of Computer and Information Engineering, Xiamen University of Technology
E-mail: jianminli_xmut@sina.com, lijm@xmut.edu.cn

1 Introduction

Temporal data are ubiquitous nowadays. They can be found in many areas such health [1,2], energy [3,4], traffic [5,6], environment [7,8], etc. In fact, with the fast development and wide deployment of data collection devices, many practical data are temporal in nature. Generally speaking, a temporal data describes the evolution of some data *object* over time [9]. In many practical applications, a data object has a *value* at any time during its evolution. For example, a stock may have a *closing price* every day; a power generator may run at a varying *load* depending on the need of power consumption; a traffic sensor by the road may record a *traffic flow* every hour. In such applications, users often show particular interests to the *top* data objects (i.e., the data objects with the best scores) within a time period.

In fields such as data management and information retrieval, there have been extensive studies on querying temporal data. The quereis can be roughly classified into two categories: point-wise queries and period-wise queries. Point-wise queries measure a data object at each time point. For example, Lee et al. [10] studied the consistent top-k (CTop-k) queries over temporal data, aiming at finding all data objects whose scores are always among the k highest ones at each time point. In contrast, period-wise queries focus on some aggregated measurement (e.g., average score, total score, etc.) of a data object over a time period. For example, Jestes et al. studied aggregate top-k queries on temporal data [11], aiming at finding the k data objects with the highest aggregation scores.

Durable top-k (DTop-k) queries are a relatively novel type of queries over temproal data. Similar to consistent top-k queries, a durable top-k query has particular interests in data objects with the best k scores at each time point. Nonetheless, unlike the case with CTop-k queries, for a data object to be a DTop-k result, it is unnecessary that its score remains a top-k at *every* time point. Instead, within a given time period, any data object that becomes one of the top-k for *sufficiently many* time points is interesting enough to a DTop-k query. In this sense, DTop-k queries can be viewed as an extension of CTop-k queries.

U et al. [12] first proposed and studied durable top-k queries on document archives. A document in an archive may have different versions over time. For example, a Wikipedia document may be editted by different users from all the world. Specifically, U et al. wanted to ideantify the documents matching some given keywords for sufficiently many time points. Wang et al. [13] studied DTop-k queries over times series data. They argue that, comparing to archived documents (which are piecewise constant), time series are more dynamic than archived documents, thus it requires dedicated techniques for efficient processing of DTop-k queries.

A key assumption in [13] is that *the time series are synchronized*, i.e., each time series has an observation/score at every time point; hence, the top-k objects at every time point can be precomputed and organized in certain index structures. However, in many practical applications, massive temporal data are collected from multiple data sources, and different data sources may use different sampling rates [14, 15]. For example, cars with different GPS devices or speedometers may generate speed data at different frequencies using different mechanisms. Therefore, it is very unlikely that such temporal data could be synchronized. Instead, practical temporal data are more likely to be *non-synchronized* [16].

In this paper, we investigate efficient processing of DTop-k queries over non-synchronized temporal data. The main challenge from non-synchronicity is that the time is continuous instead of discrete. Existing index structures (e.g. [13]) for synchronized, thus discrete, data are hardly applicable. Therefore, we propose a novel sweep line algorithm (SLA) to efficiently deal with the non-synchronicity. The key insight of SLA is the property of intersections: When two temporal data intersect, they change their relative order, and vice versa. In light of this property, SLA answers DTop-k queries by tracking snapshot top-k objects. To sum up, we make the following contributions:

1. To the best of our knowledge, we are the first to investigate the problem of efficient DTop-k processing with non-synchronized temporal data.
2. We propose a novel sweep line algorithm (SLA) to efficiently answer DTop-k queries.
3. We conduct extensive experiments using two real datasets. The results show that SLA outperforms its competitors by a large margin.

The rest of the paper is organized as follows. Section 2 introduces some preliminary knowledge (such as concepts and existing methods). Section 3 presents the solutions to DTop-k queries, including a straightforward solution (Section 3.1) and our main proposal SLA (Section 3.2). Then, Section 4 shows the experimental results. Finally, Section 5 concludes the paper.

2 Preliminaries

2.1 Concepts and Problem Definition

Definition 1 (Temporal data) A temporal data describes the evolution of a data object over time. Formally, the j-th data object in a temporal database $\mathcal{D}$ can be represented as a sequence

$$o^j = \left\langle \left(t_1^j, v_1^j\right), \left(t_2^j, v_2^j\right), \cdots, \left(t_{T_j}^j, v_{T_j}^j\right) \right\rangle,$$

where v_ℓ^j is the score of object o^j at time t_ℓ^j $(\ell = 1, 2, \cdots, n_j)$ and T_j is the recorded length of the temporal sequence of object o^j.

Note that Definition 1 offers a general representation that naturally allows asynchronicity in $\mathcal{D}$. Indeed, given two data objects in the temporal database $o^i, o^j \in \mathcal{D}$, it is allowed that the time points $t_1^i, t_2^i, \cdots, t_{T_i}^i$ and $t_1^j, t_2^j, \cdots, t_{T_j}^j$ are not aligned (and even $T_i \neq T_j$).

Definition 2 (Snapshot Top-k Object) Given a data object

$$o = \langle (t_1, v_1), (t_2, v_2), \cdots, (t_T, v_T) \rangle,$$

the value of o at time t, denoted $o(t)$, is defined as

$$o(t) = \begin{cases} \kappa_\ell (t - t_\ell) + v_\ell, & \text{if there exists } 1 \leq \ell < T \text{ such that } t_\ell \leq t \leq t_{\ell+1}, \\ -\infty, & \text{otherwise,} \end{cases}$$

where $\kappa_\ell = (v_{\ell+1} - v_\ell) / (t_{\ell+1} - t_\ell)$. Given a temporal database of N objects, $\mathcal{D} = \left\{o^1, o^2, \cdots, o^N\right\}$, an object $o^j \in \mathcal{D}$ is called a snapshot top-k object at time t if $o^j(t)$ is no less than the k-th highest score among all $o(t)$ $(o \in \mathcal{D})$.

A clear implication in Definition 2 is that, given T discrete samples in time, we also view the temporal data $o = \langle (t_1, v_1), (t_2, v_2), \cdots, (t_T, v_T) \rangle$ as a concatenation of $T-1$ line segments. Thus $o(t)$, the score of object o at time t, is a linear interpolation on the line segment within $[t_\ell, t_{\ell+1}]$, with endpoint cases of $o(t_\ell) = v_\ell$ and $o(t_{\ell+1}) = v_{\ell+1}$. This view is closely related to the piecewise linear representation of temporal data [17, 18], which may lose accuracy because actual temporal data are usually smoother. Nonetheless, in this work, we do not replace or remove original data points to obtain any approximation of the temporal data. We leave the accuracy issue to the data generation mechanism (which is beyond the scope of this work) and assume that any data $o \in \mathcal{D}$, when viewed as a concatenation of line segments, is accurate enough for any potential application. Hence, in this work, we alternatively represent a temporal data $o = \langle (t_1, v_1), (t_2, v_2), \cdots, (t_T, v_T) \rangle$ as $o = \langle s_1, s_2, \cdots, s_{T-1} \rangle$ where s_i is the line segment with endpoints (t_i, v_i) and (t_{i+1}, v_{i+1}).

Definition 3 (DTop-k Query) A durable top-k (DTop-k) query over a temporal database $\mathcal{D}$ is defined by a triple $q = \langle W, k, \gamma \rangle$, where

1. $W = [t_{\text{begin}}, t_{\text{end}}]$ is a time interval ($t_{\text{begin}} < t_{\text{end}}$);
2. k is a positive integer; and
3. $\gamma \in (0, 1]$ is a threshold.

Given a data object $o \in \mathcal{D}$, define the *durability* of $o \in \mathcal{D}$ with respect to W and k as

$$\text{dur}(o; W, k) = \text{card}\left\{ t \in W \middle| o \in \text{Top}_t^k(\mathcal{D}) \right\} / |W|,$$

where $\text{Top}_t^k(\mathcal{D}) \subseteq \mathcal{D}$ denotes the set of all snapshot top-k objects in $\mathcal{D}$ at time t. Then, the DTop-k query $q = \langle W, k, \gamma \rangle$ finds all $o \in \mathcal{D}$ with $\text{dur}(o; W, k) \geq \gamma$.

Figure 1 shows an example temporal dataset $\mathcal{D} = \left\{o^1, o^2, o^3, o^4\right\}$ and an example DTop-k query $q = \langle W, k, \gamma \rangle$ with $W = [3.5, 8.5]$, $k = 2$, and $\gamma = 0.75$. Note that $\mathcal{D}$ is non-synchronized: the temporal data are sampled with different rates. Even, some data (e.g., o^3, o^4) are not sampled using regular frequencies. Via simple calculation, we know that $\text{dur}\left(o^1; W, k\right) = 0.94$, $\text{dur}\left(o^2; W, k\right) = 1.0$, $\text{dur}\left(o^3; W, k\right) = 0.0$, and $\text{dur}\left(o^4; W, k\right) = 0.06$. Since the query threshold is $q.\gamma = 0.75$, o^1 and o^2 are the only two results to the query q.

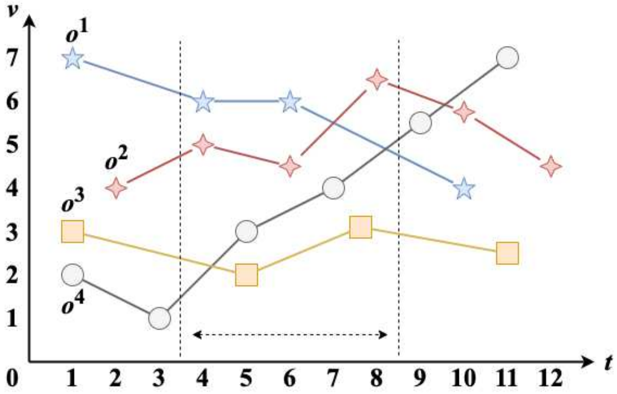

Fig. 1 Example temporal data and a DTop-k query $q = \langle[3.5, 8.5], 2, 0.75\rangle$

2.2 The Essence of Intersections

Previous studies have discovered the relationship between intersections of line segments and snapshot top-k objects (see, e.g., [13]). We summarize the main result as follows.

Lemma 1 (Intersection Lemma) *Let* $Top_t^k(\mathcal{D}) = \{o^1, o^2, \cdots, o^k\}$ *be the set of snapshot top-k objects at time t, where* $o^i(t) \geq o^j(t)$ *for any* $i < j$ *without loss of generality. Then, if some object* $o \notin Top_t^k(\mathcal{D})$ *becomes a snapshot top-k result at a later time* $t + \Delta t$*, then there must exist line segments* $s \in o$ *and* $s^k \in o^k$ *such that s and* s^k *have at least one intersection within* $[t, t + \Delta t]$.

We omit the proof of the intersection lemma as it is quite intuitive that, for a non-top-k object o to become a top-k, it must at least go up to beat the k-th best object o^k. In fact, the converse of Lemma 1 is also true: for any intersection (v, t) of two line segments, there must exist a particular k such that the snapshot top-k set $\text{Top}_t^k(\mathcal{D})$ changes at time t. Therefore, intersections of line segments play an important role in DTop-k query processing.

3 Query Processing

3.1 A Straightforword Method

In light of the intersection lemma (Lemma 1), in order to process a DTop-k query $q = \langle W, k, \gamma\rangle$ over non-synchronized temporal data, it suffices to compute the snapshot top-k sets at every intersection within the interval W. With all the snapshot top-k sets computed, it is relatively trivial to compute $\text{dur}(o)$ for each candidate data object o.

Algorithm 1 summarizes the above idea. Algorithm 1 first finds the set of all intersections within W (Lines 1-2), $P = (p_1, p_2, \cdots, p_M\}$. Note that the size of P (i.e., the number M) might be quadratic to the total number of line segments

Algorithm 1: Straightforward Algorithm

```
   Input: Temporal dataset $\mathcal{D} = \{o^1, o^2, \cdots, o^N\}$; DTop-$k$ query $q = \langle W, k, \gamma \rangle$
   Output: $\mathcal{R}$, the set of all data object $o \in \mathcal{D}$ with $\mathrm{dur}(o; W, k) \geq \gamma$
1  Let $T = \{t_1, t_2, \cdots, t_M\}$ be the set of all time points when intersection happens within $W$
2  $\tilde{T} \leftarrow \{t_0, t_1, t_2, \cdots, t_M, t_{M+1}\}$ where $t_0 = W.t_{\text{begin}}$ and $t_{M+1} = W.t_{\text{end}}$
3  for $i \leftarrow 0, 1, \cdots, M+1$ do
4      Compute the snapshot top-$k$ set $S_i \leftarrow \mathrm{Top}^k_{t_i}(\mathcal{D})$
5  for each object $o \in \bigcup_{i=0}^{M+1} S_i$ do
6      Initialize counters $o.c \leftarrow -1$ and $o.L \leftarrow 0$
7      for $i \leftarrow 0, 1, 2, \cdots, M+1$ do
8          if $o \in S_i$ then
9              if $o.c = -1$ then
10                 Start the counter $o.c \leftarrow p_i.t$
11             else
12                 Update $o.L \leftarrow o.L + p_i.t - o.c$
13                 Update $o.c \leftarrow p_i.t$
14         else
15             Update $o.c = -1$
16     if $o.L / |W| \geq \gamma$ then
17         Add $o$ into $\mathcal{R}$
18 return $\mathcal{R}$
```

within W. Algorithm 1 then finds the snapshot top-k set $S_i = \mathrm{Top}^k_{p_i.t}(\mathcal{D})$ at every intersection (Lines 3-4). After obtaining all the snapshot top-k sets, the rest of the algorithm becomes trivial. It uses a counter c to maintain the state of each object o. The counting procedure contains two cases.

- Case 1: $o \in S_i$. In this case, $o.c = -1$ means o just becomes a top-k at the intersection, thus the counter records the starting time (Lines 9-10); whereas $o.c \neq -1$ means that o is already a top-k before the intersection, thus the duration is calculated and accumulated to $o.L$ (Lines 11-13).
- Case 2: $o \notin S_i$. In this case, $o.c = -1$ means o is not a top-k before the intersection, thus there is nothing to do with o at this moment, whereas $o.c \neq -1$ means a top-k object o just gets replaced by some other object. In either case, the counter is reset to -1 (Lines 14-15).

After the counting, Algorithm 1 gets the durability of each candidate object o and returns the durable top-k results (Lines 16-17).

It remains to be clarified how to obtain the intersections (Line 1) and the snapshot top-k sets (Line 4).

3.1.1 Finding the Intersections

Given a set of line segments, finding all the intersections are a fundamental task in computer geometry. It can be efficiently solved via a *sweep line algorithm* [19]. Specifically, let $\mathcal{D} = \{o^1, o^2, \cdots, o^N\}$ be the temporal database and $S =$

$\left\{s_1^1, s_2^1, \cdots, s_{T_1-1}^1, \cdots, s_1^N, s_2^N, \cdots, s_{T_N-1}^N\right\}$ be the set of all line segments in $\mathcal{D}$. Note that each line segment $s_\ell^j \in S$ lies in the t-v plane, having a start point $\left(t_\ell^j, v_\ell^j\right)$ and an end point $\left(t_{\ell+1}^j, v_{\ell+1}^j\right)$. The sweep line algorithm first sorts all the $2|S|$ endpoints in the t-axis. Then, it moves a vertical line $t = \ell$ (i.e., the "sweep line") from left to right to scan over the sorted endpoints. The algorithm maintains two data structures, a *status* data structure $\mathrm{DS}_{\mathrm{status}}$ and a *stopping points* data structure $\mathrm{DS}_{\mathrm{stop}}$.

– The status data structure $\mathrm{DS}_{\mathrm{status}}$ is a balanced binary search tree storing the endpoints and the v-values (i.e., scores) of line segments, ordered by the scores. It supports efficient insertion, deletion, and adjacent neighbor searching of line segments.
– The stopping points data structure $\mathrm{DS}_{\mathrm{stop}}$ is a min-heap, storing all *stopping points* (i.e., endpoints and intersections), ordered by the t-values (i.e., time).

The line sweeping process works with the stopping points data structure $\mathrm{DS}_{\mathrm{stop}}$. It visits the stopping points in $\mathrm{DS}_{\mathrm{stop}}$ in the temporal order. Upon visiting a stopping point x, the main operations depend on the type of x:

– If x is a start point of some line segment $s \in S$, then s is inserted into $\mathrm{DS}_{\mathrm{status}}$. The intersections between s and its neighboring segments are computed and inserted into $\mathrm{DS}_{\mathrm{stop}}$. Delete the intersection of the original adjacent neighbors (if any) from $\mathrm{DS}_{\mathrm{stop}}$.
– If x is an end point of some s, then s is deleted from $\mathrm{DS}_{\mathrm{status}}$. The neighbors of s become adjacent and their intersection is computed and inserted into $\mathrm{DS}_{\mathrm{stop}}$.
– If x is an intersection of line segments s_1 and s_2, then swap the positions of s_1 and s_2 in $\mathrm{DS}_{\mathrm{status}}$.

In this way, all intersections can be computed in $O\left((|S| + K)\log|S|\right)$ time, where K is the actual number of intersections in S [19].

3.1.2 Finding the Snapshot Top-k

Efficient solutions to snapshot top-k queries have been studied in literature. Li et al. [20] proposed a SEB-tree index to support snapshot top-k queries (which are termed "top-$k(t)$ queries"). SEB-tree is a randomized index structure with deterministic correctness guarantees. Specifically, SEB-tree is based on p-samples of S. A p-sample $S_p \subseteq S$ is constructed by selecting each line segment $s \in S$ with probability p. To construct the SEB-tree index, Li et al. first construct a sequence of p_i-samples $S_{p_i} \subseteq S$ for $p_i = 2^{-i}$ $(i = 1, 2, \cdots)$, and then build a B-tree based on each S_{p_i}. A snapshot top-k query can then be answered using the SEB-tree index in $O\left(\log|S| + k\right)$ time.

3.1.3 Complexity Analysis

Given a temporal dataset $\mathcal{D}$ and the corresponding line segment set S, the intersections can be computed prior to any DTop-k query. When answering a

DTop-k query $q = \langle W, k, \gamma \rangle$, Algorithm 1 first takes $O(\log K)$ time to identify the intersections within W (Lines 1-2). Then, the main cost is to run M snapshot top-k queries, taking $O(M \cdot (|S| + K) \log |S|)$ time. Finally, Algorithm 1 takes $O(C \cdot M)$ time to generate the final durable top-k results, where C is the total number of candidates (i.e., the number of objects that ever is a top-k at any time in W).

3.2 A Sweep Line Method

Through the analysis in Section 3.1.3, we see that a major drawback of the straightforward method is that it issues many snapshot top-k queries, which is quite time-consuming. We argue that such a computational cost can be largely saved, as an intersection $x = (v, t)$ in fact carries much useful information.

In light of the intersection lemma (Lemma 1), an alternative solution to processing a DTop-k query $q = \langle W, k, \gamma \rangle$ over non-synchronized temporal data is to track the changes of the snapshot top-k set within the indicated time interval $W = [t_{\text{begin}}, t_{\text{end}}]$. Specifically, we may first find the snapshot top-k set $\text{Top}_t^k(\mathcal{D})$ for $t = t_{\text{begin}}$ and then scan all the intersections in temporal order to see whether and how $\text{Top}_t^k(\mathcal{D})$ changes as t goes to t_{end}. Algorithm 2 summarizes the above idea, follwing the paradigm of sweep line algorithms [19].

For each data object $o \in \mathcal{D}$, Algorithm 2 maintains a counter $(c_{\text{begin}}, c_{\text{end}}, L)$ (Line 1), which records the last time when o enters or leaves the snapshot top-k set (Lines 18-19). Whenever o leaves the snapshot top-k set at time t, L accumlates the total duration for which o has ever remained a snapshot top-k (Lines 11&17). According to the intersection lemma (Lemma 1), Algorithm 2 updates the counters whenever it discovers a new intersection x (Line 7). As will be explained later, an intersection x can provide Algorithm 1 with the information of the intersecting line segments as well as their corresponding objects (o_{in} and o_{out} at Line 7). If o_{out} is not the current k-th best, then it is either $o_{\text{in}}, o_{\text{out}} \in \text{Top}_t^k(\mathcal{D})$ or $o_{\text{in}}, o_{\text{out}} \notin \text{Top}_t^k(\mathcal{D})$, in either case $\text{Top}_t^k(\mathcal{D})$ remains unchanged.

The initial snapshot top-k set can be computed using the same technique as introduced in Section 3.1.2. The intersections used in Line 7 can be precomputed using the method introduced in Section 3.1.1. The precomputed intersections can be organized in a B$^+$-tree to support efficient positioning with respect to a query window W.

It remains to be clarified how to maintain the snapshot top-k set during the line sweeping process, i.e., how to efficiently retrieve the k-th best object (Line 15).

3.2.1 Maintaining the Snapshot Top-k

Considering fact that the parameter k is query-related, keeping track of the k-th best object is to maintain the entire ordering of the temporal database $\mathcal{D}$.

Algorithm 2: Sweep Line Algorithm

```
   Input: Temporal dataset $\mathcal{D} = \{o^1, o^2, \cdots, o^N\}$; DTop-$k$ query $q = \langle W, k, \gamma \rangle$
   Output: $\mathcal{R}$, the set of all data object $o \in \mathcal{D}$ with $\text{dur}(o; W, k) \geq \gamma$
1  Initialize counters $o.c_{\text{begin}} \leftarrow -1$, $o.c_{\text{end}} \leftarrow -1$, and $o.L \leftarrow 0$ for all $o \in C$
2  Find the snapshot top-$k$ set $C \leftarrow \text{Top}_t^k(\mathcal{D})$
3  for each $o \in C$ do
4      Update counters $o.c_{\text{begin}} \leftarrow W.t_{\text{begin}}$
5  $t \leftarrow W.t_{\text{begin}}$
6  while true do
7      Find the next intersection $x = (t', v)$ and the corresponding objects $o_{\text{in}}$ and $o_{\text{out}}$
       satisfying $t' > t$
8      if $t' \notin W$ then
9          for each $o \in C$ do
10             if $o.c_{begin} \neq -1$ then
11                 $o.L \leftarrow o.L + W.t_{\text{end}} - o.c_{\text{begin}}$
12             if $o.L/|W| \geq \gamma$ then
13                 Add $o$ into $\mathcal{R}$
14         break
15     if $o_{out}$ is not the k-th best object in $C$ then
16         continue
17     Update $o_{\text{out}}.L \leftarrow o_{\text{out}}.L + t' - o_{\text{out}}.c_{\text{begin}}$
18     Update $o_{\text{out}}.c_{\text{begin}} \leftarrow -1$ and $o_{\text{out}}.c_{\text{end}} \leftarrow t'$
19     Update $o_{\text{in}}.c_{\text{begin}} \leftarrow t'$ and $o_{\text{in}}.c_{\text{end}} \leftarrow -1$
20     Update $C \leftarrow C \cup \{o_{\text{in}}\} - \{o_{\text{out}}\}$
21     Update $t \leftarrow t'$
22 return $\mathcal{R}$
```

Since we focus on top-k objects where $k \ll |\mathcal{D}|$, in practice we may maintain the top-$k_{\max}$ set, where $k_{\max} \ll |\mathcal{D}|$ is the maximum possible query parameter.

Let $x_i = (t_i, v_i)$ $(i = 1, 2, \cdots, M)$ be all the intersections in $\mathcal{D}$ in temporal order. It is easy to see that the snapshot top-k set remains unchanged within every interval $[t_i, t_{i+1}]$, since there is no intersection between x_i and x_{i+1}. Therefore, we may precompute the snapshot top-$k_{\max}$ sets between each interval $[t_i, t_{i+1}]$, organizing the results in an index structure (e.g., a hash table). In this way, Line 15 of Algorithm 2 can be execute in constant time. Figure 2 illustrates the above idea. From Figure 2, it is also clear that the time efficiency is at the cost of index size. It requires $O(M \cdot k_{\max})$ space. Although $k_{\max} \ll |\mathcal{D}|$ by assumption, the total number of intersections M is usually at the magnitude of $O\left(|\mathcal{D}|^2\right)$. Thus the hash table index of all snapshot top-$k_{\max}$ takes quadratic space.

To strike a balance between the space cost and time efficiency, we may use a tree-based data structure similar to the status data structure $\text{DS}_{\text{status}}$ (Section 3.1.1) to facilitate the line sweeping. Specifically, we use a max-heap as an auxiliary data structure. The heap is initialized with the snapshot top-k ranking $\left(1, o^1\right), \left(2, o^2\right), \cdots, \left(k, o^k\right)$. Without loss of generality, here we assume that there is no tie in the ranking. Nonetheless, ties can be trivially handled

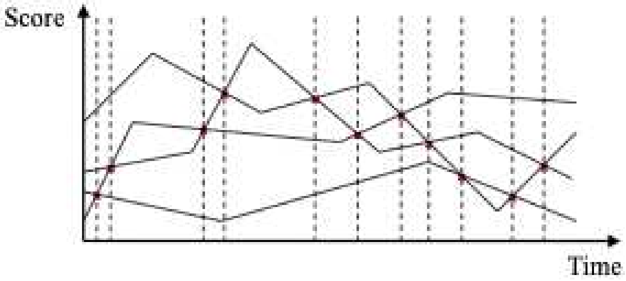

Fig. 2 Precomputing the snapshot top-$k_{\max}$ sets

in such a data structure. Using the auxiliary data structure, though it takes constant time to retrieve the k-th best object, it takes additional $O(\log k)$ time to maintain the max-heap property. Thus, the overall time for processing an intersection is $O(\log k)$.

3.2.2 Complexity Analysis

Algorithm 2 first takes $O(\log |S| + k)$ time to compute the initial snapshot top-k using the SEB-tree index, where S is the set of line segments derived from the temporal database $\mathcal{D}$. Then, it takes $O(\log K)$ time to positioning the intersections in the query interval W, where K is the total number of intersections in S. Assume that there are M intersections in W. Then, Algorithm 2 takes $O(M)$ or $O(M \cdot \log k)$ time to compute the durable top-k results, depending on whether a quadratic index is used. Comparing to the analysis in Section 3.1.3, we see that Algorithm 2 is much more efficient in processing DTop-k queries.

3.2.3 Early Termination

Given a DTop-k query $q = (k, W, \gamma)$, it is possible for Algorithm 2 to terminate before actually sweeping the entire query interval W. Let t be the current position of the sweep line and $r_t = W.t_{\text{end}} - t$. For any object o with a counter $(c_{\text{begin}}, c_{\text{end}}, L)$, the actually durability of o satisfies

$$\text{dur}(o) \le \text{dur}^*(o) = \begin{cases} (L + r_t) / |W|, & \text{if} c_{\text{begin}} = -1, \\ (L + t_{\text{end}} - c_{\text{begin}}) / |W|, & \text{otherwise.} \end{cases}$$

A clear observation is that o cannot be a DTop-k result if the upper bound $\text{dur}^*(o) < \gamma$. If at some time $t \in W$, all so-far recorded objects are either confirmed into or safely excluded from $\mathcal{R}$, Algorithm 2 can terminate without actually sweeping over the entire W.

4 Experiments

In this section, we evaluated our proposed methods. We simulate durable top-k queries over two real datasets and compare our proposed method with straightforward solutions. We also study the impact of query parameters.

4.1 Dataset

We use the Stock Market Data (SMD) as our dataset for the experiments. SMD records the daily stock market prices for all NASDAQ listed companies. Specifically, there are 1,557 companies. Each company is a temporal data from the day it was listed on NASDAQ until Aug 29, 2022. The earliest company was listed on Jan 2, 1970, whereas the latest one was just listed on Aug 24, 2022. So the SMD dataset has a timespan over 50 years.

Therefore, the temporal data are of various lengths in SMD. Each temporal data has 5,509.45 values in average, with the maximum and minimum lengths being 13,283 and 4. The oldest and youngest companies The distribution of temporal data lengths is shown in Figure 3.

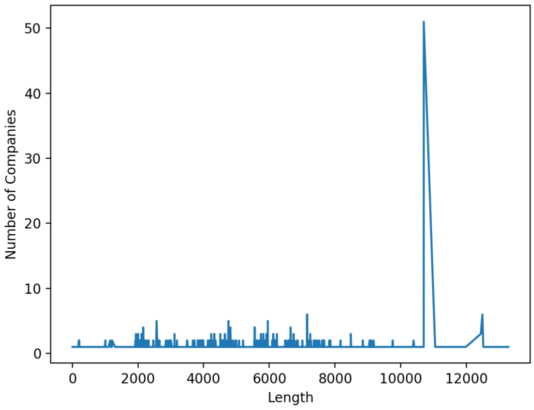

Fig. 3 Distribution of the lengths the temporal data in SMD

The original SMD dataset is synchronized in a daily manner. We randomly disturb the dataset a bit to generate a non-synchronized temporal dataset. Specifically, given a temporal object $o = \langle (t_1, v_1), (t_2, v_2), \cdots, (t_T, v_T) \rangle$, we add a random time shift Δt_i to time t_i $(i = 1, 2, \cdots, T)$. The time shift Δt_i is chosen in the range of ±1440 minutes uniformly at random. In this way, we turn the original synchronized data into a non-synchronized one without affecting much semantics of the data.

4.2 Efficiency of SLA

In this section, we compare SLA with some baseline methods. Specifically, we consider the following methods in this experiment.

- Baseline (Section 3.1). The straightforward method that issues a snapshot top-k query at every intersection.
- SLA-Q: The SLA algorithm with quadratic index (with all the top-k objects precomputed between any two intersections).
- SLA-H: The SLA algorithm with a heap structure to faciliate the line sweeping process.

We vary k in the range 5, 10, 15, 20, and 25. For each $k = 5, 10, 15, 20, 25$, we randomly generate 1,000 DTop-k queries with a fixed $\gamma = 0.75$. The length of the query interval W is also fixed to a week (i.e., 7 days, or 10,080 minutes), but the position of the interval W may vary from the first day (Jan 2, 1970) to the last (Aug 29, 2022). For each k, each of the 1,000 query is answered by each method 10 times to get an average time cost. Then, the performance of a method is measured by the average of its performance on 1,000 queries.

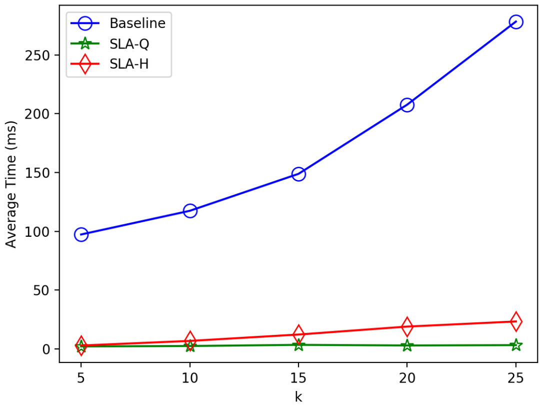

Fig. 4 Performance of SLA

Figure 4 show the results. As can be seen, the time costs of all methods increase as k gets larger. Baseline increases faster than SLA does. This is because when k increases, it costs more from snapshot top-k computation. In addition, it can be seen that SLA outperforms Baseline by orders of magnitude because SLA avoids snapshot top-k queries. Moreover, SLA-Q slightly outperforms SLA-H due to the quadratic index of precomputed top-k rankings. Nonetheless, the improvement of SLA-Q over SLA-H is not very significant. This implies a wise choice should be made to balance the space and time costs in practice.

4.3 Sensitivity to the Length of Query Interval

In this section, we investigate the impact of $|W|$, the length of the query interval, on the performance of SLA. We fix $k = 5$ and vary the length $|W|$ in the range 7, 14, 21, and 28 days. This range covers the time length from one week to approximately one month. Again, we randomly generate 1,000 DTop-k queries for each choice of $|W|$. Each query is answered by each algorithm 10 times. The average time costs are recorded and shown in Figure 5.

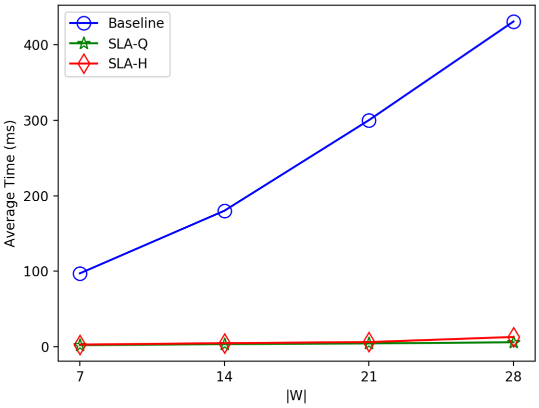

Fig. 5 Performance of SLA with varying $|W|$

From Figure 5, it is clear that, as the query interval W gets larger, the time cost of Baseline increases much faster than SLA does. This is expected because the number of intersections may be quadratic to the number of line segments in W. In addition, we see that SLA-Q and SLA-H are not as sensitive to $|W|$ as Baseline is. Indeed, comparing to the initial snapshot top-k computation, the line sweeping process over W is relatively cheap (either reading a precomputed top-k list or updating a size-k heap). In addition, the difference between SLA-Q and SLA-H is very small, which is also a reference when deciding the implementation details of SLA in practice.

5 Conclusions

In this paper, we study the problem of DTop-k queries over non-synchronized temporal data. The major challenge brought by non-synchronicity is that the time space becomes continuous and thus solutions based on discrete time space are no longer efficient. We propose an efficient sweep line algorithm SLA to process DTop-k queries over non-synchronized temporal data. The key insight of SLA is the property of intersections: When two temporal data intersect, they change their relative order, and vice versa. Using this property, SLA answers DTop-k queries by tracking snapshot top-k objects. We conduct extensive experiments on two real datasets to test the performance of our proposed method. The results show that our methods outperforms the baseline solutions by a large margin.

6 Declarations

6.1 Ethical Approval and Consent to Participate

Not applicable.

6.2 Human and Animal Ethics

Not applicable.

6.3 Consent for Publication

Not applicable.

6.4 Availability of Supporting Data

Not applicable.

6.5 Competing Interests

The authors declare that they have no competing interests.

6.6 Funding

The work is supported by the Fujian Province Science and Technology Plan Project (No. 2019J05123).

6.7 Authors' Contributions

The authors' contributions are as follows.

- Yanqi Xie: Problem formulation, algorithm design, coding, and paper writing.
- Wei Weng: Problem formulation, algorithm analysis, and experiment analysis.
- Jianmin Li: Problem formulation, algorithm design, algorithm analysis, and paper proofreading.

All authors have reviewed the manuscript.

6.8 Acknowledgements

The authors would like to thank the Science and Technology Department of Fujian Province, China, for its financial support to this work.

6.9 Authors' Information

Yanqi Xie

Yanqi Xie received his M.E. degree in College of Mathematics and Computer Science from Fuzhou University, China, in 2007. He is currently a lecturer in the School of Computer and Information Engineering, Xiamen University of Technology. His research interests cover data hiding, rough set, and pattern recognition. (Email: yqxie@xmut.edu.cn)

Wei Weng

Wei Weng is currently an associate professor with the School of Computer and Information Engineering, Xiamen University of Technology. He received the Ph.D. degree from Department of Automation, Xiamen University in 2019. His research interests include machine learning, artificial intelligence, and deep learning for graph representation. (Email: wwweng@xmut.edu.cn)

Jianmin Li (Corresponding author)

Jianmin Li received the M.S. degree in Computer Science Department from Xiamen University in 2009 and the Ph.D. degree in Department of Automation of Xiamen University in 2015. He is currently a faculty of School of Computer and Information Engineering, Xiamen University of Technology. His research interests include computer vision, machine learning, and pattern recognition. (Email: jianminli_xmut@sina.com, or lijm@xmut.edu.cn)

References

1. Deyu Deng, Carson K. Leung, Chenru Zhao, Yan Wen, and Hao Zheng. Spatial-temporal data science of COVID-19 data. In *BigDataSE*, 2021.
2. Tao Hu, Siqin Wang, Bing She, Mengxi Zhang, Xiao Huang, Yunhe Cui, Jacob Khuri, Yaxin Hu, Xiaokang Fu, Xiaoyue Wang, Peixiao Wang, Xinyan Zhu, Shuming Bao, Wendy Guan, and Zhenlong Li. Human mobility data in the COVID-19 pandemic: characteristics, applications, and challenges. *International Journal of Digital Earth*, 14(9):1126–1147, 2021.
3. Zhibin Niu, Junqi Wu, Xiufeng Liu, Lizhen Huang, and Per Sieverts Nielsen. Understanding energy demand behaviors through spatio-temporal smart meter data analysis. *Energy*, 226:120493, 2021.
4. Weixuan Lin, Di Wu, and Benoit Boulet. Spatial-temporal residential short-term load forecasting via graph neural networks. *IEEE Transactions on Smart Grid*, 12(6):5373–5384, 2021.
5. Haitao Yuan and Guoliang Li. A survey of traffic prediction: from spatio-temporal data to intelligent transportation. *Data Science and Engineering*, 6:63–85, 2021.
6. Xiyue Zhang, Chao Huang, Yong Xu, Lianghao Xia, Peng Dai, Liefeng Bo, Junbo Zhang, and Yu Zheng. Traffic flow forecasting with spatial-temporal graph diffusion network. In *AAAI*, 2021.

7. Xiansheng Liu, Hadiatullah Hadiatullah, Pengfei Tai, Yanling Xu, Xun Zhang, Jürgen Schnelle-Kreis, Brigitte Schloter-Hai, and Ralf Zimmermann. Air pollution in Germany: spatio-temporal variations and their driving factors based on continous data from 2008 to 2018. *Environmental Pollution*, 276:116732, 2021.
8. Shuang Zhao, Shiliang Liu, Xiaoyun Hou, Fangyan Cheng, Xue Wu, Shikui Dong, and Robert Beazley. Temporal dynamics of SO_2 and NO_x pollution and contributions of driving forces in urban areas in China. *Environmental Pollution*, 242:239–248, 2018.
9. Gowtham Atluri, Anuj Karpatne, and Vipin Kumar. Spatio-temporal data mining: a survey of problems and methods. *ACM Computing Surveys*, 51(4):Article 83, 2018.
10. Mong Li Lee, Wynne Hsu, Ling Li, and Wee Hyong Tok. Consistent top-k queries over time. In *DASFAA*, 2009.
11. Jeffrey Jestes, Jeff M. Phillips, Feifei Li, and Mingwang Tang. Ranking large temporal data. *PVLDB*, 5(11):1412–1423, 2012.
12. Leong Hou U, Nikos Mamoulis, Klaus Berberich, and Srikanta Bedathur. Durable top-k search in document archives. In *SIGMOD*, 2010.
13. Hao Wang, Yilun Cai, Yin Yang, Shiming Zhang, and Nikos Mamoulis. Durable queries over historical time series. *TKDE*, 26(3):595–607, 2014.
14. Haitao Wang, Jikun Ou, and Yunbin Yuan. Strategy of data processing for GPS rover and reference receivers using different sampling rates. *IEEE Transactions on Geoscience and Remote Sensing*, 49(3):1144–1149, 2011.
15. Han Su, Kai Zheng, Jiamin Huang, Haozhou Wang, and Xiaofang Zhou. Calibrating trajectory data for spatio-temporal similarity analysis. *The VLDB Journal*, 24:93–116, 2015.
16. Max Horn, Michael Moor, Christian Bock, Bastian Rieck, and Karsten Borgwardt. Set functions for time series. In *ICML*, 2020.
17. Hazem Elmeleegy, Ahmed K. Elmagarmid, Emmanuel Cecchet, Walid G. Aref, and Willy Zwaenepoel. Online piece-wise linear approximation of numerical streams with precision guarantees. In *VLDB*, 2009.
18. Ge Luo, Ke Yi, Siu-Wing Cheng, Zhenguo Li, Wei Fan, Cheng He, and Yadong Mu. Piecewise linear approximation of streaming time series data with max-error guarantees. In *ICDE*, 2015.
19. Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.
20. Feifei Li, Ke Yi, and Wangchao Le. Top-k queries on temporal data. *The VLDB Journal*, 19(5):715–733, 2010.