Processing Skyline Queries in Temporal Databases

Christos Kalyvas
Department of Information and Communication Systems Engineering, University of the Aegean, Samos, Greece
chkalyvas@aegean.gr

Theodoros Tzouramanis
Department of Information and Communication Systems Engineering, University of the Aegean, Samos, Greece
ttzouram@aegean.gr

Yannis Manolopoulos
Department of Informatics, Aristotle University of Thessaloniki, Greece
manolopo@csd.auth.gr

ABSTRACT
The skyline query aims to filter out a set of eligible points on the basis of a set of evaluation criteria and out of a potentially large dataset of points. The computation of this decision support problem has been studied across a wide range of environments and of types of data. A field of research that has remained unexplored in the context of this problem, and which would also greatly benefit from the study of the computation of the skyline query, is that of temporal databases. A solution for computing skyline queries and some of its variants over temporal data is put forward here. An experimental study indicates the promising effectiveness and practicability of the proposed extension of the skyline query processing in real-life temporal data applications.

CCS Concepts
• Information systems→Database query processing; • Information systems→Temporal data.

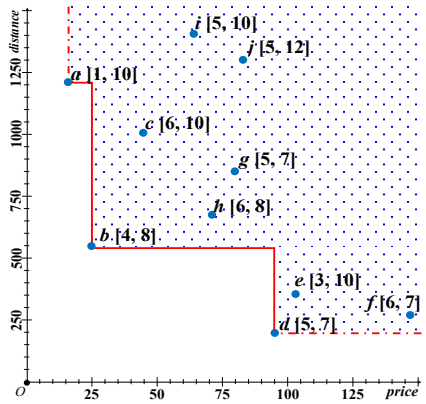
Keywords
Temporal databases, processing skyline-based queries, algorithms, experimentation, performance evaluation.

1. INTRODUCTION
In recent years, the skyline query [4] has received a considerable amount of attention because of its ability to highlight in an efficient way the most eligible subset of a set of objects on the basis of a bunch of user-defined criteria. Specifically, given a point dataset in a d-dimensional space, the skyline query retrieves the points which are not dominated by any other data point in the dataset. A point is said to dominate another point if it is as good or better in all dimensions and strictly better in at least one dimension. Without loss of generality, it is assumed that a point p dominates another point r if, for all dimensions, p has equal or smaller coordinate values than r and, in at least one dimension, the value of p is strictly smaller than r.

The following example better illustrates this concept: it is assumed that a traveler carries out a search for a hotel room. The price of a room is expected to increase as the distance of the hotel from the city center decreases; therefore a decision-support mechanism is needed to find the optimal combination between the two dimensions of distance and price. On the basis of the dataset of Figure 1(a), and by taking into account the first two columns as the primary decision criteria, the potential optimal selection for the user's preferences would be $\{a, b, d\}$.

Hotel	Price (€)	Distance from the city center (Km)	Season of operation (months of the year)	
			Start	End
a	15	1,200	1	10
b	25	550	4	8
c	45	1,000	6	10
d	95	200	5	7
e	103	350	3	10
f	147	275	6	7
g	80	850	5	7
h	70	670	6	8
i	65	1,400	5	10
j	83	1,300	5	12

(a)



(b)

Figure 1: (a) A dataset of hotels, **(b)** The skyline of the dataset.

The skyline operator has not yet been optimized to handle temporal data. In this class of data regarding time, the period of interest needs to be added as an additional constraint to be evaluated together with the decision criteria of the traditional skyline query. On this basis, the optimal selection that will cover the desired scenario, on the dataset for the example of Figure 1, for hotels operating in the 4th month of the year, would be the set of hotels $\{a, b, e\}$ which differs from the set retrieved by applying the traditional skyline query without considering the time domain.

The paper therefore studies the extension of the skyline query for temporal data and aims to demonstrate how the strategy for calculating the traditional skyline is affected when the time factor is also

SAC'17, April 3-7, 2017, Marrakesh,, Morocco.

http://dx.doi.org/10.1145/3019612.3019677

taken into consideration. Efficient algorithms for processing modified versions of the static, dynamic, and reverse skyline queries for temporal data will be proposed, together with a new dominant method for evaluating temporal data using the skyline operator.

Section 2 summarizes the related work on skyline query processing as well as the various indexing methods for storing and querying temporal data. Section 3 formulates the problem addressed in this work by introducing the relevant terminology. Section 4 presents the proposed algorithms for processing skyline queries in temporal databases. Section 5 reports on the experimentation results on real and synthetic data regarding the performance of the proposed algorithms. Section 6 concludes and looks ahead to possible extensions of this work.

2. RELATED WORK

Skyline query processing. The computation of the skyline in database research is equivalent to determining the *maximal vector problem* in computational geometry, equivalent to the *pareto optimal set* [13] problem in operations research. The first work to address the skyline computation problem in the databases context is [4]. The first index-based solutions for processing the skyline query were the Bitmap and the Index algorithms proposed in [23]. The Nearest Neighbor (NN) algorithm [12] that followed is the first to use the wide-spread R-tree index [8]. The Branch & Bound (BBS) algorithm [20] is an improvement on the NN algorithm and offers a state-of-the-art and I/O optimal solution to the problem since it traverses the R-tree only once.

A natural extension of the skyline query is the *dynamic skyline query* [20], in which the dynamic coordinates of every point object in the dataset are given by a set of functions that are based on the distance of the point to a given reference query point q. Intuitively, the dynamic skyline corresponds to the skyline on a transformed space in which the query point q becomes the new origin-point and all the distances are computed on the basis of this point. Dynamic skyline queries are quite useful when the user's preferences on every axis are defined explicitly, forming a vector of preferences on the d-dimensional space. The dynamic skyline problem has been studied in several domains such as spatial databases [22], subspaces [25], data streams [15], etc.

The *reverse skyline query* and the methods for processing it efficiently are introduced in [5] and [6]. It is based on the dynamic skyline and its goal is to identify the influence of a given vector of characteristics over a dataset of vectors of user preferences based on the distance between them and the given vector. Given the preferences of potential hotel customers as points in the two-dimensional (distance and price) space, the reverse skyline query can provide an answer if it makes sense to offer them a hotel room q at a specific distance from the city center and at a specific price. The hotel room q (becoming an origin-point) will be eligible for a potential traveler, if it belongs to the dynamic skyline of the vector of her/his preferences p. Many reverse skyline variants have been proposed for several domains such as for data streams [29], uncertain data [16], wireless sensor networks [28], etc.

Temporal data processing. The increasing interest in maintaining numerous time-varying data versions and in supporting queries and trends analysis for decision making using these data, has led to the publication of over 2,000 research papers, to a comprehensive glossary of terminology [9], surveys and books in temporal databases. These usually refer to two types of time, valid time and transaction time. The first corresponds to the time when a fact is true in the real world. The second is the time during which a piece of data is stored in the database. Databases that combine both these types of time are called bi-temporal.

Surveys of access methods for efficient query processing in temporal databases are found in [21] and [19]. A cluster of these methods are modifications of the traditional B^+-tree access structure such as the Multi-version B-tree [1] and the Overlapping B^+-trees [26]. They usually index tuples in the form $<k, t_1, t_2>$ in which k is a key of a database relation and $[t_1, t_2]$ is a time interval, which in most of these cases is the transaction time. Another cluster employs mapping strategies and transformations such as the mapping of time intervals to single-dimensional points in MAP21 [18] or the interval transformation in the Interval Space Transformation method [7]. These methods usually index valid time ranges of the form $[v_1, v_2]$. Another cluster is comprised of extensions of space partitioning indexing structures such as the 4R-tree [2], 3D R-tree [27], MV3R-tree [24] *etc*. Most of these methods can efficiently support tuples of the form $<k, t_1, t_2, v_1, v_2>$, and can index both temporal and bi-temporal data.

While several temporal queries, joins and semijoins have been explored for several application domains, a query that has not been discussed yet in temporal databases is the skyline query and its variants. This paper addresses the problem and proposes algorithms for computing efficiently the well-known static, dynamic and reverse skylines for temporal data. Closely related work is found in [11], in which the interval skyline query is introduced for time series applications. However, the properties that are valid for the time series environment differ entirely from those in the field of general temporal (non-time-series) data. Therefore the proposed algorithm is not applicable in temporal and bi-temporal databases. The present paper could be seen to complete the work of [14] and [10], both of which study the top-k query on temporal data, i.e. a query that belongs to the same broader family of ranking queries as the skyline query. Importantly, in order to support the convex skyline query for sets of spatiotemporal objects in privacy aware environments in which the disclosure of only aggregated values of objects is allowed, the authors in [17] refer to the same term *temporal skyline* but with a meaning which differs from the meaning used in this paper.

3. PROBLEM FORMULATION

The study involves the extensions of the static, dynamic and reverse skyline queries for the handling of temporal data. It will focus in one dimension of time, which can be either the transaction or the valid time and will comment on the extension of the proposed solution to handle both time dimensions. The following definitions will help to clarify the main angles of the paper.

Definition 1 - Temporal dominance: Given a time-varying point dataset P in a d-dimensional space D and a point p $(p_1, ..., p_d) \in P$ with validity in the time interval t_p, the point p *temporally dominates* in the time interval t another point r $(r_1, ..., r_d) \in P$ with validity in the time interval t_r, denoted as $p \prec_t r$, if and only if t is the non-null intersection between the time intervals t_p and t_r; and $\forall\, i \in \{1, ..., d\}$ we have $p_i \le r_i$ and $\exists\, j \in \{1, ..., d\}: p_j < r_j$.

Definition 2 - Temporal Skyline Query: Given a time-varying point dataset P in a d-dimensional space D, the *temporal skyline query* in the time interval t_s retrieves the set of time-varying points $SL_{ts}(P) \subseteq P$ which are not temporally dominated by any other point

in P in at least the non-null time interval $t \subseteq t_s$, that is, $SL_{ts}(P) = \{(p, t)$, where $t \subseteq t_s$ and $p \in P \mid \nexists\ r \in P: r \prec_t p\}$. $SL_{ts}(P)$ is called the temporal skyline of P in the time interval t_s.

Figure 1(a) illustrated the temporal database of ten data tuples represented in Figure 1(b) by time-varying points $P = \{a, b, ..., j\}$ in the two-dimensional space. Some data points in the figure temporally dominate others, such as point b which temporally dominates point c in the time interval [6, 8]. The temporal skyline of P in the time interval [3, 8] is the set $SL_{[3, 8]}(P) = \{(a, [3, 8]), (b, [4, 8]), (d, [5, 7]), (e, [3, 4]), (e, [8, 8])\}$. Note that point e is part of the temporal skyline of P in two different time intervals.

Definition 3 - Dynamic Temporal Dominance: Given a time-varying point dataset P in a d-dimensional space D and a query point q $(q_1, ..., q_d) \in D$ with validity in the time interval t_q, a point p $(p_1, ..., p_d) \in P$ with validity in the time interval t_p *dynamically temporally dominates* another point r $(r_1, ..., r_d) \in P$ with validity in the time interval t_r with regard to q in the time interval t, denoted as $p \prec_{(q, t)} r$, if and only if t is the non-null intersection between the time intervals t_p, t_r and t_q, and $\forall\ i \in \{1, ..., d\}$ we have $|q_i - p_i| \leq |q_i - r_i|$ and $\exists\ j \in \{1, ..., d\}: |q_j - p_j| < |q_j - r_j|$.

Definition 4 - Dynamic Temporal Skyline Query: Given a time-varying point dataset P in a d-dimensional space D and a query point q $(q_1,..., q_d) \in D$ with validity in the time interval t_q, the *dynamic temporal skyline query* of P with regard to q in the time interval t_q retrieves the set $SL_{(q, tq)}(P)$ of points in P which are not dynamically temporally dominated by any other point in P in at least the non-null time interval $t \subseteq t_q$, that is, $SL_{(q, tq)}(P)=\{(p,t)$, where $t \subseteq t_q$ and $p \in P \mid \nexists\ r \in P: r \prec_{(q, t)} p\}$. $SL_{(q, tq)}(P)$ is called the dynamic temporal skyline of P with regard to q in the time interval t_q.

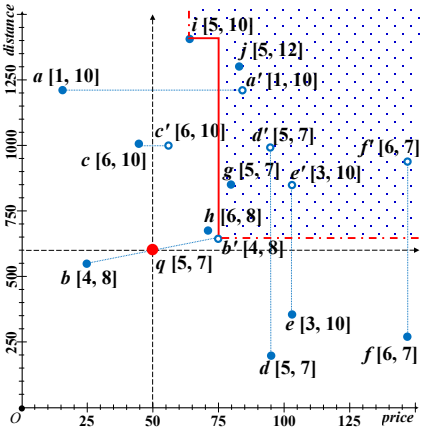


Figure 2: The dynamic temporal skyline of the dataset of Figure 1 with regard to a query point q in the time interval [5, 7].

In Figure 2 every database point p (p_x, p_y) in the original two-dimensional space of Figure 1(b) is transformed into a point p' $(|q_x - p_x|, |q_y - p_y|)$ in a new two-dimensional space, the origin in which is the query point q (50, 600) with validity in the time interval [5, 7]. The dynamic temporal skyline of P with regard to q consists of the set $SL_{(q, [5, 7])}(P) = \{(b, [5, 7]), (c, [6, 7]), (h, [6, 7]), (i, [5, 5])\}$. Again it is possible for a data point to be part of a dynamic temporal skyline in more than one subinterval.

Definition 5 - Reverse Temporal Skyline Query: Given a time-varying point dataset P in a d-dimensional space D and a reference query point q $(q_1, ..., q_d) \in D$ with validity in the time interval t_q, the *reverse temporal skyline query* of P with regard to q in the time interval t_q retrieves the set $RSL_{(q, tq)}(P)$ of points in P which take q as one of their dynamic temporal skyline points in at least the non-null time-interval $t \subseteq t_q$. This means that a point $p \in P$ with validity in the time interval t_p belongs to the set $RSL_{(q, tq)}(P)$ and therefore is a reverse temporal skyline of q in the time-interval t, if there does not exist any other point $r \in P$ with validity in the time interval t_r such that (1) t is the non-null intersection between the time intervals t_p, t_r and t_q, (2) $\forall\ i \in \{1, .., d\}: |r_i - p_i| \leq |q_i - p_i|$, and (3) $\exists\ j \in \{1, ..., d\}: |r_j - p_j| < |q_i - p_j|$. $RSL_{(q, tq)}(P)$ is called the reverse temporal skyline of P with regard to q in the time interval t_q.

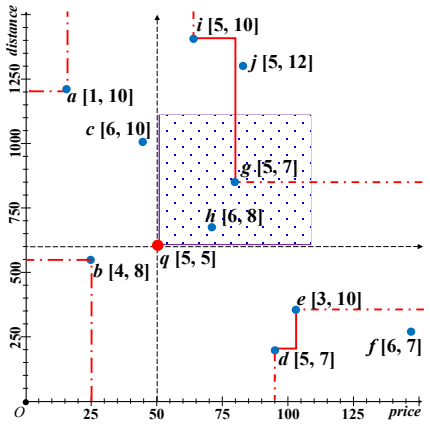


Figure 3: The reverse temporal skyline of the dataset of Figure 1 with regard to a query point q in the time instant 5.

In the example of Figure 3, the reverse temporal skyline of P with regard to query point q (50, 600) in the time interval [5, 5], i.e. in the time instant 5, consists of the set $RSL_{(q, [5, 5])}(P) = \{(a, [5, 5]), (b, [5, 5]), (g, [5, 5]), (i, [5, 5])\}$. For instance, since the dynamic temporal skyline of data point g in the time instant 5 contains the query point q (i.e. this holds because no any other data point exists in the grey range of Figure 3 in the time instant 5), g is a reverse skyline point of q in that time instant.

4. SKYLINE QUERY PROCESSING IN TEMPORAL DATA

4.1 The Temporal Skyline Query

The algorithm for computing the temporal skyline of a time-varying point dataset is an extension of the original BBS algorithm [20] for traditional (non-temporal) data. Since BBS uses a typical data-partitioning method, such as the R-tree, to serve as the backbone indexing method, in this paper the 3D R-tree access method [27] is considered to be the best choice for maintaining the temporal data. The reason for this choice is that the description of the 3D R-tree differs only slightly from that of the traditional R-tree in respect of its ability to store transaction and/or valid time data as extra data dimensions in the tree. Another reason for selecting the 3D R-tree is that it is accompanied by a simple implementation and requires the fewest possible modifications to the built-in functionalities of modern database management systems as compared to its competitors in the temporal databases domain. The 3D R-tree can also straightforwardly support as many user-defined data dimensions as required, and for any skyline query processing application, as compared to most of its temporal indices competitors, which can support only a single dimension for the key of the data tuples, plus of course one or two time dimensions.

```
Algorithm 1: The temporal skyline query ()
Input:       A dataset P, indexed using the 3D R-tree
             and a requested time interval t_s.
Output:      The temporal skyline SL_ts of P.
       SL_ts = H = ∅; // H is a heap
1:     FOR every 3D R-tree root entry e with validity
         in the time interval t_e DO
2:       IF t_e ∩ t_s ≠ ∅ THEN insert (e, t_e ∩ t_s) into H;
3:     WHILE H is not empty DO
4.       Remove top entry (e, t_e) of H;
5:       FOR every interval t ⊆ t_e in which e is not
           temporally dominated by any point in SL_ts DO
6:         IF e is an intermediate entry THEN
7:           FOR every child ee of e, with validity
               in the interval t_ee with t ∩ t_ee ≠ ∅ DO
8:             FOR every time interval t' ⊆ t ∩ t_ee
                 in which ee is not temporally
                 dominated by any point in SL_ts DO
9:               Insert (ee, t') into H;
10:        ELSE // e is a data point
11:          Insert (e, t) into SL_ts;
12:    RETURN SL_ts;
```

Algorithm 1: The temporal skyline query.

The *pseudo* code of the algorithm for computing the temporal skyline is illustrated in Algorithm 1. The proposed algorithm makes temporal dominance checks by considering independently the time dimension. The point dataset of Figure 1 will be used, organized in the four MBRs R_1, R_2, R_3 and R_4 that are illustrated in Figure 4. For simplicity, it will be assumed that the root node of the 3D R-tree holds only these four MBRs. The distances are computed according to L_1 norm, i.e. the *mindist* of a data point to the origin point O of the data space is equal to the sum of its coordinates, while the corresponding *mindist* of an MBR equals the *mindist* of its lower left-corner point.

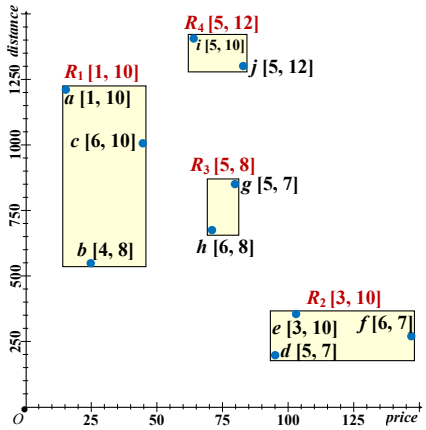


Figure 4: The dataset of Figure 1 organized in four MBRs.

The requested time interval to compute the temporal skyline is assumed to be the $t_s = [3, 8]$. The algorithm in Lines 1-2 starts from the 3D R-tree root node and inserts all its entries with time validity overlapping the requested time interval in a heap H, in the form $\{(R_2, [3, 8]), (R_1, [3, 8]), (R_3, [5, 8]), (R_4, [5, 8])\}$, sorted according to the MBRs' *mindist*. Then, by executing Lines 4-9 of the algorithm, the MBR entry (R_2, [3, 8]) with the minimum *mindist* will be replaced in the heap by its data entries, in the form: (d, [5, 7]), (f, [6, 7]), and (e, [3, 8]).

The next entry to be extracted from the heap according to Figure 5 is (d, [5, 7]), which, according to Line 11 of the algorithm, is inserted into the temporal skyline list. The next entry to be extracted from the heap is (f, [6, 7]) for which, in Line 5 of the algorithm, it is discovered that it is temporally dominated in every time instant in the interval [6, 7] by entry (d, [5, 7]) of the temporal skyline. The next entry to be extracted from the heap is (e, [3, 8]) for which, in Line 5 of the algorithm, it is discovered that it is not temporally dominated in the time subintervals [3, 4] and [8, 8], therefore the corresponding entries (e, [3, 4]) and (e, [8, 8]) are inserted in the temporal skyline. The MBR R_1 is then expanded and, as Figure 5 shows, its contents are inserted in the heap. Finally, after processing some more entries, the MBR R_4 is extracted from the heap, which, however, is temporally dominated in the entire requested time interval [3, 8] of the query.

action	H content	$SL_{[3,8]}$ ( ) content
expand root in [3, 8]	(R_2, [3, 8]), (R_1, [3, 8]), (R_3, [5, 8]), (R_4, [5, 8])	–
expand R_2 in [3, 8]	(d, [5, 7]), ~~(f, [6, 7])~~, (e, [3, 8]), (R_1, [3, 8]), (R_3, [5, 8]), (R_4, [5, 8])	(d, [5, 7]), (e, [3, 4]), (e, [8, 8])
expand R_1 in [3, 8]	(b, [4, 8]), ~~(R_3, [5, 8])~~, ~~(e, [6, 8])~~, (a, [3, 8]), ~~(R_4, [5, 8])~~	(d, [5, 7]), (e, [3, 4]), (e, [8, 8]), (b, [4, 8]), (a, [3, 8])

Figure 5: Processing steps of the example execution of Algorithm 1.

The correctness of the proposed algorithm is straightforwardly inherited from the corresponding correctness [20] of the BBS algorithm for traditional (non-temporal) data. This means that every data point added into the temporal skyline during the execution of the algorithm is guaranteed to be a final temporal skyline point for the time interval under consideration. Also, every data point in the 3D R-tree will be examined by the algorithm, unless one of its ancestor nodes has been pruned for the whole time interval of the validity of the data point. The proposed algorithm is also progressive, it provides neither false misses nor false hits and it is able to allow the user to determine the order in which skyline points will be returned.

In the case of bi-temporal data, the algorithm can perform temporal dominance checks by considering every time dimension independently, which means that a data point belongs to the temporal skyline only if it is not temporally dominated by any other point in the dataset in both the valid and transaction time dimensions.

4.2 The Dynamic Temporal Skyline Query

While the static temporal skyline evaluates the data objects on the basis of the minimum (or maximum) values of their coordinates, the dynamic temporal skyline evaluates the data objects in respect of a customer's given preference point q (q_1, …, q_d) within a specified time interval t_q (e.g. a hotel at 50 euros, at a 600 meters from the city center, the following April). Therefore, the dynamic temporal skyline query with regard to q in the time interval t_q, for every data point p (p_1, …, p_d) with validity in the time interval t_p which overlaps t_q, specifies d functions of the form $\forall\ i \in \{1, ..., d\}: f_i = |q_i - p_i|$, and the goal is to return the static temporal skyline of P in the time interval t_q, in the transformed/dynamic workspace which has q as its point of origin and the coordinates of every object p in every dimension are defined by the functions f_i.

Algorithm 1 is applicable to dynamic temporal skylines by storing in the heap the entries according to their *mindist* in the dynamic workspace. Please refer to [20] for more details. The main modifications that are needed, so that Algorithm 1 can process the dynamic temporal skyline query, is the replacement of the temporal dominance checks in Lines 5 and 8 by dynamic temporal dominance checks, as they are set out in Definition 3.

4.3 The Reverse Temporal Skyline Query

As with the dynamic temporal skyline, the reverse temporal skyline evaluates the data objects with regard to a given query point q on a specified time interval t_q. However, the main difference between these two queries is that the dynamic temporal skyline query can be seen as a query from the customer's perspective whereas the reverse temporal skyline can be seen as a query from the company's perspective (e.g. which customers –having their preferences represented by data points in the workspace– would be interested in a hotel room at 50 euros, at 600 meters from the city center, between October and May?).

Four different algorithms for processing the reverse skyline query for traditional (non-temporal) data are proposed in [5] and [6], with the Branch & Bound Reverse Skyline (BBRS) algorithm [5] to be the one selected to serve as a backbone algorithm for extension in order to support the reverse temporal skyline. The BBRS algorithm is chosen for the simplicity of its implementation and its ability to run without the need to preprocess the dynamic skyline of every point in the dataset. The drawback of the BBRS in comparison to its three competitors is that it requires that the index be traversed once for every candidate reverse skyline point that is found in the final filtering step of the algorithm. This can be easily overcome by ensuring that the algorithm is accompanied by a buffer to hold the most frequently –or the least recently– used nodes of the index in memory for faster potential future usage.

```
Algorithm 2: The reverse temporal skyline query ()
Input:       A dataset P, indexed using the 3D R-tree,
             a query point q and a time interval t_q.
Output:      The reverse temporal skyline RSL_(q, tq) of P
      RSL = H = ∅; // H is a heap
1:    FOR every 3D R-tree root entry e with validity
        in the time interval t_e DO
2:      IF t_e ∩ t_q ≠ ∅ THEN insert (e, t_e ∩ t_q) in H;
3:    WHILE H is not empty DO
4:      Remove top entry (e, t_e) of H;
5:      FOR every interval t ⊆ t_q ∩ t_e in which e is
          not globally temporally dominated by any
          point in RSL DO
6:        IF e is an intermediate entry THEN
7:          FOR every child ee of e, with validity
              in the interval t_ee with t ∩ t_ee ≠ ∅ DO
8:            FOR every time interval t' ⊆ t ∩ t_ee in
                which ee is not globally temporally
                dominated by any point in RSL DO
9:              Insert (ee, t') into H;
10:       ELSE // e is a data point
11:         Execute a range query based on e, q, t;
12:         IF the range query is empty in any time
              interval t' ⊆ t THEN
13:           Insert (e, t') into RSL;
14:   RETURN RSL;
```

Algorithm 2: The reverse temporal skyline query.

The *pseudo* code of the proposed algorithm is illustrated in Algorithm 2. The algorithm in Lines 5 and 8 makes global temporal dominance checks according to the following definition.

Definition 6 - Global Temporal Dominance: Given a time-varying point dataset P in a d-dimensional space D and a query point q (q_1, ..., q_d) $\in D$ with validity in the time interval t_q, a point p (p_1, ..., p_d) $\in P$ with validity in the time interval t_p *globally temporally dominates* another point r (r_1, ..., r_d) $\in P$ with validity in the time interval t_r with regard to q in the non-null time interval t if and only if (1) t is the non-null intersection between the time intervals t_p, t_r and t_q, (2) $\forall\, i \in \{1, ..., d\}$: $(p_i - q_i)(r_i - q_i) > 0$, (3) $\forall\, i \in \{1, ..., d\}$: $|p_i - q_i| \leq |r_i - q_i|$, and (4) $\exists\, j \in \{1, ..., d\}$: $|p_j - q_j| < |r_i - q_j|$.

On the basis of the definition and of the example of Figure 3, it can be said that the point g globally temporally dominates the point j in the time instant 5.

The global temporal dominance checks in the algorithm help with the pruning of intermediate index nodes (and data points) which cannot store (or be, respectively) reverse temporal skyline points. The first ten lines of the algorithm are executed in a manner similar to Algorithm 1 for the temporal skyline. However, for every point e with validity t_e overlapping the time interval t_q of the query q that is not globally temporally dominated in a time interval $t \subseteq t_e \cap t_q$, a further examination is required. This examination is performed in Lines 11-12 of the algorithm by issuing a range query, e being in the centre of the range window and q in its corner, similarly to that illustrated in grey around the data point g in Figure 3. As [5] shows for the case of non-temporal data, if this range query returns no data point for a time interval $t' \subseteq t$, then e is a reverse temporal skyline point with regard to q in t'. Therefore, in this case in Line 13 of the algorithm the tuple (e, t') is inserted into the $RSL_{(q, tq)}$ list.

In the reverse skyline query of Figure 3 with regard query point q (50, 600) and time instant 5, the data entries which are valid and not globally temporally dominated by any other point in this instant are a, i, h, b, d, and e. However, after performing the range query checks of Lines 11-12 of the algorithm, only the first four of these points is found to belong to the reverse temporal skyline.

The handling of bi-temporal data can be treated in a manner analogous to the temporal skyline query, by performing global temporal dominance checks in every time dimension independently.

5. EXPERIMENTAL STUDY

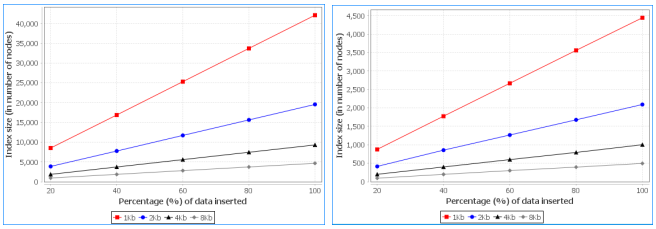
The proposed query algorithms were implemented in Java (JDK version 8). The 3D R-tree implementation is based on the R*-tree implementation that can be downloaded from the ChoroChronos portal[1], with the operational modification to preferably select time dimension splitting (time-split) in the case of an overflowing tree node split. The workstation that was used for evaluation was equipped with Intel I7 8GB RAM and Windows 10 Pro 64-bit.

The experiments used two datasets: a synthetic dataset constructed from 1,000,000 uniformly distributed time-varying two-dimensional points with a uniformly distributed time interval validity of a maximum of 20% of the lifetime of the scenario, which is 1,000 time instants; and the real-life Major Hotel Chain dataset [3] with 147,029 bookings collected from five U.S. hotels,

[1] http://chorochronos.datastories.org

of which 100,000 were selected randomly. The construction of a two-dimensional point for every single booking record considered the Nightly_Rate column, plus a uniformly distributed artificial column of values between 0 and 100, representing for example customer rating scores. Every booking's time validity interval is the combination of the Check_In_Date and Check_Out_Date columns as they are given by the data provider. The lifetime of the scene is 56 distinct time instants, i.e. 56 dates.

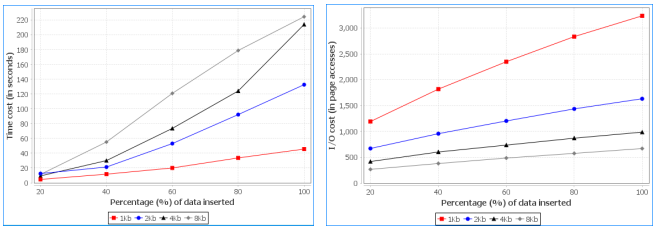
Every experiment was repeated 10 times and the average value of every measured parameter was calculated. As with the dynamic and reverse temporal skyline queries, at every run a different randomly selected query point was used. In the following, unless otherwise stated, the findings of the performance investigation of the proposed query processing algorithms are qualitatively comparable, whether synthetic or real data are used, whence the decision for partial depiction. Four different values are considered for the file system page size, *i.e.* 1Kb, 2Kb, 4Kb and 8Kb. The 3D R-tree node size is set to be equal to the page size. To get a crystal-clear view on the algorithms' performance, no buffer is introduced to hold any 3D R-tree node in main memory during the experiments.



(a) (b)

Figure 6: The 3D R-tree index size in a number of nodes, **(a)** for the synthetic dataset, and **(b)** for the real dataset.

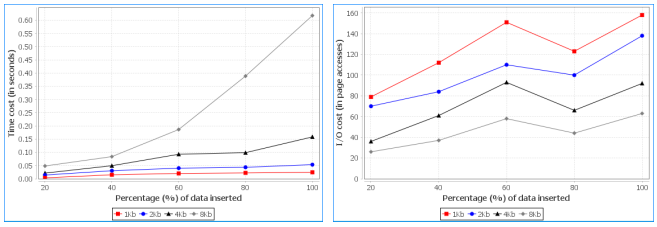
The graphs in Figure 6 illustrate the 3D R-tree index size for the synthetic (left) and real (right) datasets. The index size was measured at every 20% intervals of the data inserted. These results help to measure the percentage of the index that is accessed when processing the static and dynamic temporal skyline queries in the subsequent experiments, since every execution of their algorithms traverses the 3D R-tree only once.



(a) (b)

Figure 7: (a) The time cost, and **(b)** the I/O cost, for executing the temporal skyline query algorithm for the synthetic dataset.

The graphs in Figure 7 illustrate the time-cost in seconds (left) and the I/O cost in page accesses (right) for answering the temporal skyline query using the synthetic dataset. The query is executed at every 20% intervals of the data inserted, in every case for a time interval equal to the lifespan of the whole scenario, the skyline is thus computed for every time instant in the lifetime of the scene. By comparing the I/O cost to answer the query and the corresponding index size shown in Figure 6(a), it is concluded that the temporal skyline algorithm accesses about 8% to 23% of the nodes of the index. This cost is justified by the large number of the 1,000



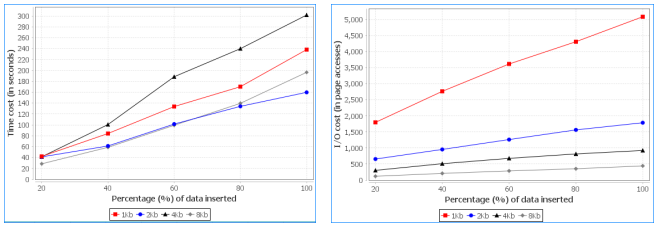
(a) (b)

Figure 8: (a) The time cost, and **(b)** the I/O cost, in both cases for executing the temporal skyline query algorithm for the real dataset.

time instants for which the skyline is computed with only a single tree traversal using the proposed Algorithm 1.

The graphs in Figure 8 illustrate the time (left) and the I/O (right) cost for answering the temporal skyline query using the real-life dataset. The query is executed again for all instants in the lifetime of the whole scene. Since the real dataset scene lifetime is much smaller than that of the synthetic dataset, the algorithm using the real dataset provides relatively faster answers due to much fewer temporal dominance checks.

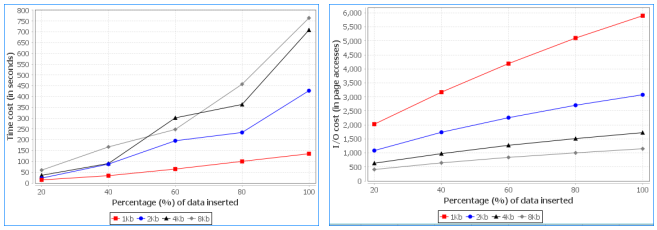
A comparison of the I/O cost for answering the temporal skyline query using the real dataset with the corresponding index size shown in Figure 6(b) indicates that the proposed query algorithm appears to accesses about 3% to 19% of the index. This percentage represents (as in the case of the synthetic dataset) a zone of tree nodes accessed by the algorithm that have MBRs located closely to the origin point *O* of the workspace.



(a) (b)

Figure 9: (a) The time cost, and **(b)** the I/O cost, for executing the dynamic temporal skyline query algorithm for the synthetic dataset.

The graphs in Figure 9 show the time (left) and the I/O (right) cost of the execution of the dynamic temporal skyline query algorithm using the synthetic dataset. The validity of every query point is set to be equal to the lifetime of the whole scene. Supporting this "dynamic" query incurs an expected cost that is larger than the cost of the corresponding "static" temporal skyline query. This cost is incurred because the corresponding algorithm accesses a larger portion of the dataset as the result of the fact that the reference query point is not always on a corner of the workspace (as it would be with the ordinary temporal skyline query).



(a) (b)

Figure 10: (a) The time cost, and **(b)** the I/O cost, for executing the reverse temporal skyline query algorithm for the synthetic dataset.

The last graphs in Figure 10 show the time (left) and I/O (right) cost for executing the reverse temporal skyline query algorithm using the synthetic dataset. The expected increase in cost in comparison to the dynamic temporal skyline, even if both queries access quite similar parts of the data space when using the same query points q, is due to the additional overhead of the reverse temporal skyline because of the empty/boolean range query executed by Algorithm 2 in Line 11.

All the experiments indicate that the computation of a temporal skyline-based query for the whole domain of the time dimension is a costly task, with the worst performance when every point in the dataset is active for a time interval which does not overlap with the time interval of any other point in the dataset. In this case all the data belong to the temporal skyline; therefore the algorithms will need to access all the nodes of the index. By contrast, when all the points in the dataset have identical time intervals, the performance of the temporal skyline-based algorithms coincides with the performance of the corresponding algorithms for traditional (non-temporal) data.

6. CONCLUSION

The skyline query is a decision support mechanism which retrieves the value-for-money options of a dataset by identifying the objects which present the optimal combination of the characteristics of the dataset. This is the first attempt to involve the time factor, while optimizing the skyline operator and its dynamic and reverse skyline variants. This work should hopefully pave the way for the construction of other solutions for processing efficiently skyline-based queries for a variety of temporal and bi-temporal data applications. The next step is to investigate the impact of the temporal indexing method on the cost performance of queries, through the performance comparison of indexing methods or by proposing new ones, and to investigate the impact on query execution performance in the case of many objects with a relatively small or large time interval lifetime. Future research needs also to consider the introduction of efficient algorithms for processing extensions of other skyline query variants that will be applicable to temporal data.

7. REFERENCES

[1] Becker, B., Gschwind, S., Ohler, T., Seeger, B., and Widmayer, P. An asymptotically optimal multiversion B-tree. *The VLDB Journal*, 5, 4 (1996), 264-275.

[2] Bliujute, R., Jensen, C. S., Saltenis, S., and Slivinskas, G. Light-weight indexing of general bitemporal data. In *the 12th SSDBM Conference* 2000, 125–138.

[3] Bodea, T., Ferguson, M., and Garrow, L. Data set-choice-based revenue management: Data from a major hotel chain. *Manufacturing & Service Operations Management 11*, 2 (2009), 356–361.

[4] Borzsony, S., Kossmann, D., and Stocker, K. The skyline operator. In *the 17th ICDE Conference,* 2001, 421–430.

[5] Dellis, E., and Seeger, B. Efficient computation of reverse skyline queries. In *the VLDB Conference*, 2007, 291–302.

[6] Gao, Y., Liu, Q., Zheng, B., and Chen G. On efficient reverse skyline query processing. *Expert Systems with Applications*, 41, 7 (2014), 3237-3249.

[7] Goh, C. H., Lu, H., Ooi, B.-C., and Tan, K.-L. Indexing temporal data using existing B+-trees. *Data & Knowledge Engineering 18*, 2 (1996), 147–165.

[8] Guttman, A. R-trees: a dynamic index structure for spatial searching, In the *ACM SIGMOD Conference*, 1984, 47–57.

[9] Jensen, C. S., Dyreson, C. E., et al. The consensus glossary of temporal database concepts - February 1998 version. In *Temporal Databases: Research and Practice*. 1998, 367–405.

[10] Jestes, J., Phillips, J. M., Li, F., and Tang, M. Ranking large temporal data. *Proceedings of the VLDB Endowment 5*, 11 (2012), 1412–1423.

[11] Jiang, B., and Pei, J. Online interval skyline queries on time series. In *the ICDE Conference*, 2009, 1036–1047.

[12] Kossmann, D., Ramsak, F., and Rost, S. Shooting stars in the sky: An online algorithm for skyline queries. In *the 28th VLDB Conference,* 2002, 275–286.

[13] Kung, H.-T., Luccio, F., and Preparata, F. P. On finding the maxima of a set of vectors. *Journal of the ACM (JACM) 22*, 4 (1975), 469–476.

[14] Li, F., Yi, K., and Le, W. Top-k queries on temporal data. *The VLDB Journal, 19*, 5 (2010), 715–733.

[15] Li, Z., Peng, Z., Yan, J., and Li, T. Continuous dynamic skyline queries over data stream. *Journal of Computer Research and Development 48*, 1 (2011), 77–85.

[16] Lian, X., and Chen, L. Reverse skyline search in uncertain databases. *ACM Transactions on Database Systems 35*, 1 (2010), 3.

[17] Morimoto, Y., and Siddique, M. A. Skyline sets query and its extension to spatio-temporal databases. In *the DNIS Workshop,* 2010, 317–329.

[18] Nascimento, M. A., and Dunham, M. H. Indexing valid time databases via B+-trees. *IEEE Transactions on Knowledge and Data Engineering 11*, 6 (1999), 929–947.

[19] Ozsoyoglu, G., and Snodgrass, R. T. Temporal and real-time databases: A survey. *IEEE Transactions on Knowledge and Data Engineering 7*, 4 (1995), 513–532.

[20] Papadias, D., Tao, Y., Fu, G., and Seeger, B. Progressive skyline computation in database systems. *ACM Transactions on Database Systems 30*, 1 (2005), 41–82.

[21] Salzberg, B., and Tsotras, V. J. Comparison of access methods for time-evolving data. *ACM Computing Surveys (CSUR) 31*, 2 (1999), 158–221.

[22] Sharifzadeh, M., and Shahabi, C. The spatial skyline queries. In *the 32nd VLDB Conference,* 2006, 751–762.

[23] Tan, K.-L., et al. Efficient progressive skyline computation. In *the 27th VLDB Conference,* 2001, 301–310.

[24] Tao, Y., and Papadias, D. The MV3R-tree: A spatio-temporal access method for timestamp and interval queries. In *the 27th VLDB Conference,* 2001, 431-440.

[25] Tao, Y., Xiao, X., and Pei, J. Efficient skyline and top-k retrieval in subspaces. *IEEE Transactions on Knowledge and Data Engineering 19*, 8 (2007), 1072-1088.

[26] Tzouramanis, T., Manolopoulos, Y., and Lorentzos, N. Overlapping B+-trees: an implementation of a transaction time access method. *Data & Knowledge Engineering 29*, 3 (1999), 381–404.

[27] Vazirgiannis, M., Theodoridis, Y., and Sellis, T. Spatio-temporal composition and indexing for large multimedia applications. *Multimedia Systems 6*, 4 (1998), 284–298.

[28] Wang, G., Xin, J., Chen, L., and Liu, Y. Energy-efficient reverse skyline query processing over wireless sensor networks. *IEEE Transactions on Knowledge and Data Engineering 24*, 7 (2012), 1259–1275.

[29] Zhu, L., Li, C., and Chen, H. Efficient computation of reverse skyline on data stream. In *the CSO Joint Conference*, 2009, 735–739.