A New Framework for Addressing Temporal Range Queries and Some Preliminary Results*

Qingmin Shi and Joseph JaJa
Institute for Advanced Computer Studies,
Department of Electrical and Computer Engineering,
University of Maryland, College Park, MD 20742
{*qshi,joseph@umiacs.umd.edu*}

January 29, 2003

Abstract

Given a set of n objects, each characterized by d attributes specified at m fixed time instances, we are interested in the problem of designing space efficient indexing structures such that arbitrary temporal range search queries can be handled efficiently. When $m = 1$, our problem reduces to the d-dimensional orthogonal search problem. We establish efficient data structures to handle several classes of the general problem. Our results include a linear size data structure that enables a query time of $O(\log n \log m / \log\log n + f)$ for one-sided queries when $d = 1$, where f is the number of objects satisfying the query. A similar result is shown for counting queries. We also show that the most general problem can be solved with a polylogarithmic query time using nonlinear space data structures.

1 Introduction

In this paper, we introduce a framework for exploring temporal patterns of a set of objects and discuss the design of indexing structures for handling temporal orthogonal range queries in such a framework. We assume that each object is characterized by a set of attributes, whose values are given for a sequence of time snapshots. The temporal patterns of interest can be defined as the values of certain attributes remaining within certain bounds, changing according to a given pattern (say increasing or decreasing), or satisfying certain statistical distributions. We focus here on temporal patterns characterized by orthogonal range values over the attributes. More specifically, we are aiming to design indexing structures to quickly find objects whose attributes fall within a set of ranges during a given time period specified at query time. In the dynamic case, either objects or time snapshots can be added or deleted.

*Supported in part by the National Science Foundation through the National Partnership for Advanced Computational Infrastructure (NPACI), DoD-MD Procurement under contract MDA90402C0428, and NASA under the ESIP Program NCC5300.

Our framework is very general and encompasses problems in multidimensional range search and temporal range search for data time series.

More formally, let S be a set of n objects $\{O_1, O_2, \cdots, O_n\}$, each of which is characterized by a set of d attributes whose values change over time. We are given m snapshots of each object at time instances $t_1, t_2, \ldots, t_m$. The set of values of the d attributes of object O_i at time instance t_j is denoted as a vector $\mathbf{v}(i,j) = [v_i^j(1), v_i^j(2), \ldots, v_i^j(d)]$.

We are interested in developing a data structure for S so that the following types of queries, called *temporal range queries*, can be handled very quickly:

> Given two vectors $\mathbf{a} = [a_1, a_2, \cdots, a_d]$ and $\mathbf{b} = [b_1, b_2, \cdots, b_d]$, and two time instances t_s and t_e. Find the set Q of objects such that for every $O_i \in Q$, $a_k \leq v_i^j(k) < b_k$ for all $1 \leq k \leq d$ and $t_s \leq t_j \leq t_e$.

Note that the general multidimensional orthogonal range search is a special case of our problem corresponding to a single time snapshot. Typically, we measure the complexity in terms of the storage cost of the data structure and the query time as functions of n,m, and d, where typically d is considered to be a constant.

Many applications fall in a natural way under our general framework. The following is a list of a few such examples.

- Climatologists are often interested in studying the climate change patterns for certain geographical areas, each characterized by a set of environmental variables such as temperature, precipitation, humidity, etc. Given a time series of such information for n regions, one would like to quickly explore relationships among such regions by asking queries of the following type: determine the regions where the annual precipitation is above 40 inches and the summer temperature is above 70°F between the years 1965 and 1975.

- In the stock market, each stock can be characterized by its daily opening price, closing price, and trading volume. Related interesting queries that fall under our framework are of the following type: determine the stocks, each of whose daily opening price is less than \$2 and whose daily trading volume is larger than 200 million shares during the year 2000.

- As an application related to data warehousing, consider a retail chain that has stores across the country, each of which reports their sales on a daily basis. A typical query will for example be to identify the stores whose sales exceeded \$100,000 for each of the past 12 months.

- Consider a set of n cities, each characterized by annual demographic and health data, for a period of 30 years. In exploring patterns among these cities, one may be interested in asking queries about the number of cities that had a high cancer rate and a high ozone level between 1990 and 2000.

The d-dimensional orthogonal range search problem, which is a special case of our problem, has been studied extensively in the literature. The best results do achieve linear space and polylogarithmic query time for three-sided reporting queries and four-sided counting queries for $d = 2$ [13, 3], and for dominance reporting queries for $d = 3$. Otherwise, all fast query time algorithms require nonlinear space, sometimes coupled with matching lower bounds under certain computational models[2, 5, 4]. Note that we can't treat our problem as an orthogonal range search problem by simply treating the time snapshots as just an extra time dimension to the d dimensions corresponding to the attributes. This is the case since the values of an object's attributes at different time instances cannot be treated simply as independent of each other. However we can combine all the attribute values of an object together to specify that object, resulting in a (md)-dimensional range search problem, which is clearly quite undesirable, especially for large m.

Another related class of problems that have been studied in the literature, especially the database literature, deals with a time series of data by appending a time stamp to each piece of data separately. Hence such an approach will be quite inefficient to capture temporal information about single objects since it will have to process the values at all the time steps between t_s and t_e. Examples of such techniques include those based on *persistent* data structures [6], such as the Multiversion B-tree [10] and the Multiversion Access Methods [20], and the Overlapping B^+-trees [12] and its extensions such as the Historical R-tree [14], the HR^+-tree [17], and the Overlapping Linear Quadtrees [18, 19]. To answer a query that involves a time period, the query time of these methods will often depend on the length of the time period, which is undesirable for our general problem since the temporal range query could cover a very long time period characterized by the two parameters t_s and t_e.

Another related topic involves the so-called *kinetic data structures*, which are used for indexing moving objects. Queries similar to ours involving both time periods and positions of objects have been studied, for example in the work of Ararwal et al. [1] and Saltenis et. al [15]. However, the objects are considered there to be points moving along a straight line and at a consistent speed. As a result, the positions of the objects need not be explicitly stored. In our case, such a problem will be formulated as the positions of each object at different time instances (that are the same for all the objects), without any assumption about expected trajectories or speeds.

Before stating our main results, let us introduce two main variations of temporal range queries, which are similar to those appearing in orthogonal range search queries. The *reporting query* requires that a list of the objects (or their indices) be generated as an answer to the query, while the *counting query* requires only that only the number of objects satisfying the query be generated. Our results include the following:

- A linear space data structure that handles temporal range queries for a single object in $O(1)$ time, assuming the number d of attributes is constant.

- Two data structures that handle temporal one-sided range reporting queries for a set of objects in $O(\log m \log n + f)$ [1], and $O(\log m \log n / \log \log n + f)$ time respectively, the first using $O(nm)$ space, and the second using $O(mn \log^{\epsilon} n)$, where f is the number of objects satisfying the query, and $d = 1$.

- Two data structures that use $O(nm \log(nm))$ and $O(nm \log^{1+\epsilon}(nm))$ space respectively to answer the temporal one-sided range counting queries. The first data structure

[1]In this paper, we always assume the logarithmic operations to be of base 2.

enables $O(\log^2(nm))$ query time and the second enables $O((\log(nm)/\log\log(nm))^2)$ time, under the assumption that $d = 1$.

- By a reduction to the $2d$-dimensional dominance problem, the most general problem can be solved in polylogarithmic query time using $O(nm^2\text{polylog}(n))$ space. When m is extremely large, we show that it is possible to use $o(nm^2)$ space to achieve polylogarithmic query time.

Before proceeding, we notice that the actual time instances $\{t_1, t_2, \cdots, t_m\}$ can be replaced by their subscripts $\{1, 2, \cdots, m\}$. By doing so, we introduce the additional complexity of having to convert t_s and t_e specified by the query to l_1 and l_2 respectively, where t_{l_1} is the first time instance no earlier than t_s and t_{l_2} is the last time instance no later than t_e. This conversion can be done in $O(\log m)$ time and $O(m)$ space using binary search or an asymptotically faster $O(\log m/\log\log m)$ algorithm with a larger constant behind the big-O notation and the same $O(m)$ space using the fusion tree of Fredman and Willard [7]. In the remaining of this paper, we assume that the time instances are represented by integers $\{1, 2, \cdots, m\}$ and the time interval in the query is represented by two integers l_1 and l_2. For brevity, we will use the $[i..j]$ to denote the set of integers $\{i, i+1, \cdots, j\}$. Without causing confusion, we will call the set of contiguous integers $[i..j]$ a time period.

The remainder of the paper is organized as follows. The next section discusses a special version of the temporal range search problem, which involves only a single object. The data structure for the reporting case of temporal one-sided range queries is covered in Section 3, while the counting version is covered in Section 4. In Section 5, we deal with the two-sided temporal range query, and conclude in Section 6.

2 Preliminaries: Handling Range Queries of a Single Object

Consider the case of temporal range queries involving only a single object O. We provide a simple solution to this case, which will be used to handle the more general case. Let the values of the attributes of O at time instance j be $[v^j(1), v^j(2), \cdots, v^j(d)]$. Given two real vectors $\mathbf{a} = [a_1, a_2, \cdots, a_l]$ and $\mathbf{b} = [b_1, b_2, \cdots, b_l]$, and two time instances l_1 and l_2, we will describe an efficient method to test whether the following predicate holds:

> P: For every time instances j that satisfies $l_1 \le j \le l_2$, $a_k \le v^j(k) \le b_k$ for all k between 1 and d.

Since we are assuming that d is a fixed constant, we can restrict ourselves to the following case. Let the object O be specified by $[v^1, v^2, \cdots, v^m]$, where each v^i is a real number. We develop a data structure that can be used to test the following predicate for any given parameters l_1, l_2, and a:

> P': For every time instances j satisfying $l_1 \le j \le l_2$, $v^j \ge a$.

We start by making the following straightforward observation.

Observation 1 *A predicate of type P' is true if and only if* $\min\{v^j | j \in [l_1..l_2]\} \geq a$.

Using this observation, our problem is reduced to finding the minimum value v^j of the object during the time period $[l_1..l_2]$ and comparing it against the value of a.

The problem of finding the minimum value in time period $[l_1..l_2]$ can be reduced to the problem of finding the nearest common ancestor in the so called *Cartesian tree*, as described in [8].

A Cartesian tree [21] for a sequence of m real numbers is a binary tree with m nodes. In our case, a Cartesian tree for time instances $[l..r]$ with $l \leq r$ has $r - l + 1$ nodes. The root stores the smallest value v^j over the time period $[l..r]$. If there are multiple v^j's with the smallest value, the earliest one is chosen to be stored at the root. The left subtree of the root is the Cartesian tree for time instances $[l..(i-1)]$ and the right subtree is the Cartesian tree for the time instances $[(i+1)..r]$. The left (resp. right) tree is null if $i = l$ (resp. $i = r$). The tree nodes are labeled l through r according to the in-order traversal of the tree (which correspond to their time instances). Figure 1 gives an example of the Cartesian tree.

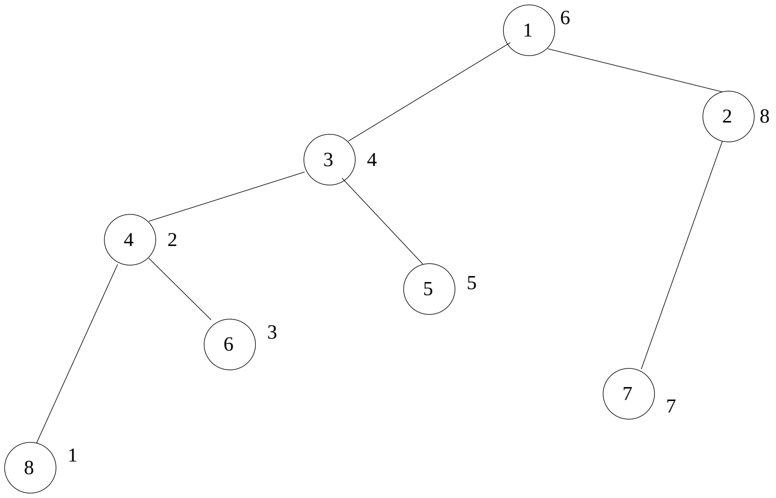

Figure 1: A Cartesian tree for the sequence $[8, 4, 6, 3, 5, 1, 7, 8]$. The number outside each node represents the time instance of the attribute value stored at the node.

It is easy to realize that the smallest value among $\{v^i, \ldots, v^j\}$ is the one stored in the nearest common ancestor of nodes i and j. This problem was addressed in [9], where the following result is shown.

Lemma 1 *Given a collection of rooted trees with n vertices, the nearest common ancestor of any two vertices can be found in $O(1)$ time, provided that pointers to these two vertices are given as input. This algorithm uses $O(n)$ space.*

It is easy to see if a tree is complete, we can easily solve the nearest common ancestor problem in linear space and constant time by labeling the tree nodes in the order of the in-order traversal and performing bit-operations on the labels corresponding to the two vertices. Harel and Tarjan solve the the same problem for any arbitrary tree by first transforming it into a compressed tree of logarithmic depth, augmenting the subtrees of it into complete trees without asymptotically increasing the overall storage cost, and applying the technique for complete trees. For details see [9].

Given the above lemma, we immediately have the following result.

Theorem 1 *Predicate P' can be evaluated using $O(1)$ time with $O(m)$ space data structure.*

If we build a Cartesian tree where an internal node stores the maximum instead of the minimum value, we can evaluate predicates involving upper bounds instead of lower bounds. We call the former Cartesian tree a *minimum Cartesian tree* and the latter a *maximum Cartesian tree*. By building both the minimum and the maximum Cartesian trees for each of the d attributes, we will be able to evaluate the general P predicates in linear space and constant time, which is optimal.

Corollary 1 *A P predicate can be evaluated using $O(1)$ time with $O(m)$ space data structure.*

3 Handling One-Sided Queries for an Arbitrary Number of Objects

In this section, we deal with temporal range queries for n objects with only one attribute, that is $d = 1$. Let v_i^j denote the value of object O_i at time instance j. We want to preprocess the data and construct a linear size data structure so that queries of the following type can be answered quickly:

> Q_1: Given a tuple (l_1, l_2, a), with $l_1 \leq l_2$, report all objects whose attributes are greater than or equal to a for all time instances between l_1 and l_2.

We call such queries *temporal one-sided reporting queries.*

Observation 1 is again very important in answering queries of type Q_1. A straightforward approach to solve our problem would be to determine for each possible time interval the set of minimal values, one for each object, and store the minima corresponding to each time interval in a sorted list. A query can then be immediately handled using the sorted list corresponding to the time interval $[l_1, l_2]$. However, the storage cost would then be $O(nm^2)$, which is quite high especially in the case when m is much larger then n. We will develop an alternative strategy that requires only linear space.

Assume that we have built a Cartesian tree C_i for object O_i. Then, each attribute v_i^j of this object can be associated with a sequence of contiguous time instances during which v_i^j is the smallest. We call this sequence the *dominant interval* of v_i^j. In fact, the dominant interval corresponds to the set of nodes in the subtree rooted at the node j in C_i. For example, the value v_i^4 of the object i whose corresponding Cartesian tree is shown in Figure 1 is associated with time interval $[1, 5]$. Let $[s_i^j..e_i^j]$ be the dominant interval of attribute v_i^j.

Consider the set of nm tuples $(v_i^j, s_i^j, e_i^j, i, j)$. One way for answering a Q_1 query would be to identify those 5-tuples that satisfy $[s_i^j..e_i^j] \supseteq [l_1..l_2]$ and $v_i^j \geq a$. However an object can be reported many times, which defeats our goal of achieving a query time of $O(\log^c(nm) + f)$, where c is a small constant and f is the number of objects satisfying the query. Consider for example the object given in Figure 1. A query with $l_1 = 2$, $l_2 = 3$, and $a = 0$ would report it three times, for the 5-tuples that correspond to time instances 2, 4, and 6. In fact, an object can be reported m times in the worst case.

This problem is taking care of in the next lemma.

Lemma 2 *An object O_i should be reported if and only if there exist a unique 5-tuple $(v_i^j, s_i^j, e_i^j, i, j)$ such that the following conditions are true: $[s_i^j..e_i^j] \supseteq [l_1..l_2]$; $j \in [l_1..l_2]$; and $v_i^j \geq a$.*

Proof:
Suppose an object O_i needs to be reported. This means its values during the time period $[l_1..l_2]$ are no smaller than a. Let $v_i^j = \min\{v_i^l | l_1 \leq l \leq l_2\}$. It is obvious that the 5-tuple $(v_i^j, s_i^j, e_i^j, i, j)$ satisfies the three conditions in Lemma 2. On the other hand, it is straightforward to see that the existence of such a 5-tuple ensures the presence of object O_i in the answer to the query. The uniqueness of the 5-tuple $(v_i^j, s_i^j, e_i^j, i, j)$ is guaranteed by the definition of dominant intervals $[s_i^j..e_i^j]$. Indeed, suppose we have another 5-tuples $(v_i^{j'}, s_i^{j'}, e_i^{j'}, i, j')$ that satisfies $[s_i^{j'}..e_i^{j'}] \supseteq [t_s..t_e]$, $j' \in [t_s..t_e]$, and $v_i^{j'} \geq a$. By definition, both j and j' are the smallest values during the time interval $[l_1..l_2]$. Without lose of generality, assume $j < j'$, then $s_i^{j'} > j$, which is in contradiction to the condition that $s_i^{j'} \leq l_1 \leq j$. □

Lemma 2 reduces the problem of determining the objects satisfying the query to finding a 5-tuple for each such object, which satisfies the three stated conditions. To solve the latter problem, we first single out those attributes that were taken during the time period $[l_1, l_2]$ and then filter them using the remaining two conditions.

We first construct a balanced binary tree T based on the m time instances. The jth leaf node starting from the left corresponds to time instance j. Each node v of this tree is associated with a set $S(v)$ of n tuples, one from each object. If v is the jth leaf node, then $S(v) = \{(v_i^j, s_i^j, e_i^j, i, j) | i = 1, \ldots, n\}$. If v is an internal node with two children u and w and the 5-tuples of object O_i in $S(u)$ and $S(w)$ are $(v_i^{j_1}, s_i^{j_1}, e_i^{j_1}, i, j_1)$ and $(v_i^{j_2}, s_i^{j_2}, e_i^{j_2}, i, j_2)$ respectively, then the 5-tuple of object O_i in $S(v)$ is $(v_i^j, s_i^j, e_i^j, i, j)$, where j is either j_1 or j_2, depending on whether $[s_i^{j_1}..e_i^{j_1}] \supseteq [s_i^{j_2}..e_i^{j_2}]$ or $[s_i^{j_2}..e_i^{j_2}] \supseteq [s_i^{j_1}..e_i^{j_1}]$. (The reason why one and only one of the above conditions must be true should be easy to understand by recalling the definition of dominant intervals.) To give an example, let us consider the case where $n = 2$ and $m = 8$. The values of the attributes of the two objects and the corresponding 5-tuples are given in Table 1. Figure 2 gives the corresponding tree structure.

Given a Q_1 query (l_1, l_2, a), we can easily find the set of at most $\log m$ *allocation nodes* in T, using the interval $[l_1, l_2]$. An allocation node is a node whose corresponding time interval is fully contained in $[l_1, l_2]$ and that of whose parent is not. If the query time interval is [2..6], for the example given in Figure 2, then the allocation nodes are b, k, and l. For each allocation node v, we know that all the n samples in $S(v)$ are taken during the time period $[l_1, l_2]$. Therefore, if a 5-tuple $(v_i^j, s_i^j, e_i^j, i, j) \in S(v)$ satisfies $[s_i^j..e_i^j] \supseteq [l_1, l_2]$ and $v_i^j \geq a$, then O_i should be reported. Otherwise, object O_i should not be reported. In either case, no further search on v's descendants is needed. This is true because of the following. First, if O_i is reported at node v, then there is no need to look for O_i any more. Second, if O_i is not reported at v, this means either $[s_i^j..e_i^j] \not\supseteq [l_1..l_2]$ or $v_i^j < a$. If the former is true, then no tuple of O_i stored in the descendants of v can cover $[l_1..l_2]$ because $[s_i^j..e_i^j]$ covers

j	object O_1 value	object O_1 5-tuple	object O_2 value	object O_2 5-tuple
1	8	(8,1,1,1,1)	5	(5,1,1,2,1)
2	4	(4,1,3,1,2)	2	(2,1,3,2,2)
3	6	(6,3,3,1,3)	4	(4,3,3,2,3)
4	3	(3,1,5,1,4)	1	(1,1,8,2,4)
5	5	(5,5,5,1,5)	7	(7,5,5,2,5)
6	1	(1,1,8,1,6)	3	(3,5,8,2,6)
7	7	(7,7,7,1,7)	6	(6,7,8,2,7)
8	2	(2,7,8,1,8)	8	(8,8,8,2,8)

Table 1: The values of two objects

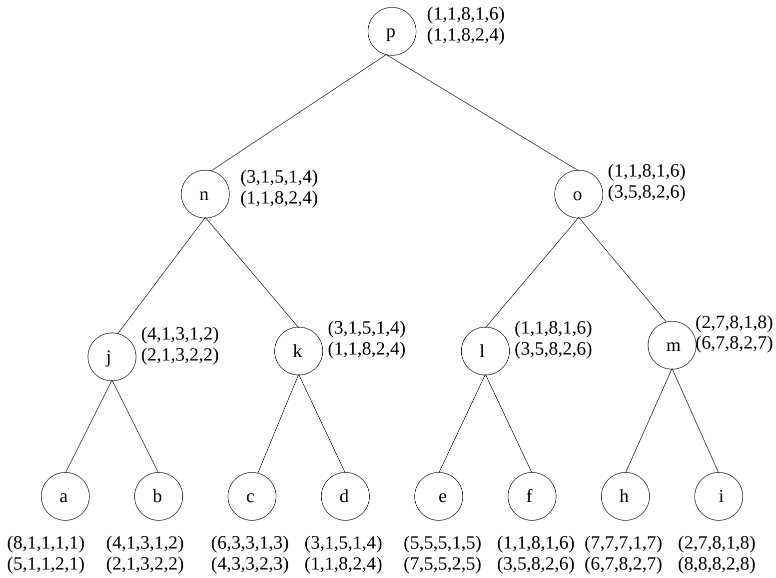

Figure 2: The tree structure corresponding to the data given in Table 1. Each node contains two 5-tuples, one from each object.

the dominant intervals of all the other values of O_i stored in the subtree rooted at v. If the latter is true, then we are sure O_i should not be reported at all.

One final note is that, even though an object is represented multiples times in the form of its tuples, it will be reported at most once. This can be justified as follows. If an object is reported, then only one of its m tuples satisfies the conditions derived from the query. Note that even though a tuple may be stored in up to $\log m$ nodes, these nodes form a partial path from the root to a leaf node and, as a result, only the node at the highest level corresponding to $[l_1, l_2]$ will be considered.

For each node v, looking for 5-tuples $(v_i^j, s_i^j, e_i^j, i, j) \in S(v)$ that satisfy $[s_i^j..e_i^j] \supseteq [l_1, l_2]$ and $v_i^j \geq a$ is equivalent to a three-dimensional dominance reporting problem, which can be solved in $O(\log n + f(v))$ time using the data structure of Makris and Tsakalidis [11], which we call the *dominance tree*. Here $f(v)$ is the number of objects reported when node v is visited. Note that there are $2m - 1$ nodes in the tree and each node is associated with a dominance tree of size $O(n)$. The overall size of the data structure is $O(nm)$. A query process involves identifying the $O(\log m)$ allocation nodes in $O(\log m)$ time and searching the dominance trees associated with these allocation nodes. Hence $O(\log n + f(v))$ time is spent at each such node v. Therefore, the complexity of the overall algorithm is $O(\log n \log m + f)$, where f is total number of objects reported.

In [16], we provide a faster algorithm for solving the the three-dominance query problem. The algorithm uses $O(n \log^{\epsilon} n)$ space and $O(\log n / \log\log n + f)$ query time, where ϵ is an arbitrarily small positive constant. Using this data structure instead of the dominance tree, we can further reduce the query complexity to $O(\log m \log n / \log\log n + f)$ at the expense of increasing the storage cost to $O(mn \log^{\epsilon} n)$. We thus have the following theorem.

Theorem 2 *Given n objects, each specified by the values of its attribute at m time instances, we can build an indexing structure so that any one-sided reporting query can be answered in $O(\log n \log m + f)$ time and $O(nm)$ space, or $O(\log m \log n / \log\log n + f)$ time and $O(mn \log^{\epsilon} n)$ space, where f is the number of objects satisfying the query and ϵ is an arbitrarily small positive constant.*

We next consider the counting query counterpart.

4 Handling One-Sided Counting Queries

In this section, we consider the following temporal range counting queries.

> Q_2: Given a tuple (l_1, l_2, a), with $l_1 \le l_2$, determine the number of objects whose values are greater than or equal to a for all time instances between l_1 and l_2.

The conditions stated in Lemma 2 (Section 3) can be expressed as $s_i^j \le l_1 \le j$, $j \le l_2 \le e_i^j$, and $v_i^j \ge a$; and there is at most one such instance. Hence the answer to the query is $|A(l_1, l_2, a)|$, where $A(l_1, l_2, a) = \{(i,j) | s_i^j \le l_1 \le j, j \le l_2 \le e_i^j, \textit{ and } v_i^j \ge a\}$.

Let

$$U(l_1, l_2, a) = \{(i,j) | v_i^j \ge a\},$$

$$B_1(l_1, l_2, a) = \{(i,j) | l_2 < j \textit{ and } v_i^j \ge a\},$$

$$B_2(l_1, l_2, a) = \{(i,j) | l_2 > e_i^j \textit{ and } v_i^j \ge a\},$$

$$B_3(l_1, l_2, a) = \{(i,j) | l_1 < s_i^j \textit{ and } v_i^j \ge a\},$$

$$B_4(l_1, l_2, a) = \{(i,j) | l_1 > j \textit{ and } v_i^j \ge a\},$$

$$C_1(l_1, l_2, a) = \{(i,j) | l_1 < s_i^j, l_2 < j \textit{ and } v_i^j \ge a\},$$

$$C_2(l_1, l_2, a) = \{(i,j) | l_1 > j, l_2 < j \textit{ and } v_i^j \ge a\},$$

$$C_3(l_1, l_2, a) = \{(i,j) | l_1 < s_i^j, l_2 > e_i^j \textit{ and } v_i^j \ge a\}, \text{ and}$$

$$C_4(l_1, l_2, a) = \{(i,j) | l_1 > j, l_2 > e_i^j \text{ and } v_i^j \geq a\}.$$

We have the following lemma:

Lemma 3 $|A| = |U| - |B_1| - |B_2| - |B_3| - |B_4| + |C_1| + |C_2| + |C_3| + |C_4|$.

Proof:
It is easy to see that $\overline{A} = U - A = B_1 \cup B_2 \cup B_3 \cup B_4$. Thus, $|\overline{A}| = \sum_{i=1,2,3,4} |B_i| - \sum_{i,j \in \{1,2,3,4\}, i \neq j} |B_i \cap B_j| + \sum_{i,j,k \in \{1,2,3,4\}, i \neq j \neq k} |B_i \cap B_j \cap B_k| - |\cap_{i=1,2,3,4} B_i|$. It is clear the third and the fourth terms in the right hand side of this equation are both zero. As for the second term, the only four non-empty intersections are $B_1 \cap B_3$, $B_1 \cap B_4$, $B_2 \cap B_3$, and $B_2 \cap B_4$, which correspond to the sets C_1, C_2, C_3, C_4 respectively. □

The problem of determining the size of each of the sets U, B_i or C_i can be viewed as a special version of three-dimensional dominance counting problem defined as follows:

Q_2': *Given a set V of n three dimensional points, preprocess V so that given a point (x, y, z), the number of points in V that are dominated by (x, y, z) can be reported efficiently.*

Unlike the reporting case, algorithms for the three-dimensional dominance counting problem that have linear space and polylogarithmic query time are not known to the authors' best knowledge. However Chazelle gives a linear space and $O(\log n)$ time algorithm [3] for the two-dimensional case. Using the scheme of the range tree, his result can easily be extended to the three-dimensional case by first building a binary search tree on the x-coordinates, and then associate with each node the data structure for answering two-dimensional dominance queries involving only the y- and z-coordinates. The resulting data structure provides an $O(n \log n)$ space and $O(\log^2 n)$ time solution.

By using the fusion tree techniques, we were able to improve the query time to $O((\log n / \log \log n)^2)$ at the expense of increasing the storage cost by a factor of $O(\log^\epsilon n / \log \log n)$. For details, see [16]. Since we have a total of nm tuples, Theorem 3 follows.

Theorem 3 *Given n objects, each characterized by the values of its attribute at m time instances, we can preprocess the input so that any one-sided counting query can be answered in $O(\log^2(nm))$ time using an $O(nm \log(nm))$ space data structure, or $O((\log(nm)/\log\log(nm))^2)$ time using an $O(nm \log^{1+\epsilon}(nm)/\log\log(nm))$ space data structure.*

5 Fast Algorithms for Handling Two-Sided Queries

In this section, we address the general type of queries for which the values of the objects to be reported are bounded between two values a and b during the time period $[l_1..l_2]$. More specifically,

Q_3: Given a tuple (l_1, l_2, a, b), with $l_1 \leq l_2$ and $a \leq b$, report all objects O_i, such that $a \leq v_i^j \leq b$ for all $j = l_1, \ldots, l_2$.

The following is a direct extension of Observation 1.

Observation 2 *An object O_i should be reported for a Q_3 query if and only if* $\min\{v_i^j | j \in [l_1..l_2]\} \geq a$ *and* $\max\{v_i^j | j \in [l_1..l_2]\} \leq b$.

In this section, we first show that, even for an arbitrary number d of attributes, the two-sided queries can be handled fast if we are willing to use $O(nm^2\text{polylog}(n))$ space for the indexing structure. We later show that we can achieve fast query time using $o(nm^2)$ space in the case when m is extremely large. We start by looking at the case when $d = 1$, which admits a simple solution.

To achieve a polylogarithmic query time, we compute for each pair of $(t_1, t_2) \in [1..m] \times [1..m]$ with $t_1 < t_2$ the minimum value $m_i^{t_1,t_2}$ and maximum value $M_i^{t_1,t_2}$ for each object O_i and index the n minimum-maximum pairs in a suitable data structure T^{t_1,t_2} designed to efficiently handle two-dimensional dominance queries. Pointers to these $O(m^2)$ structures can be stored in a array to allow constant-time access. Given any query (l_1, l_2, a, b), we use (l_1, l_2) to locate the appropriate data structure T^{l_1,l_2} in constant time and use it to answer the two-dimensional dominance query: $m_i^{t_1,t_2} \geq a$ and $M_i^{t_1,t_2} \leq b$.

A possible data structure for T^{t_1,t_2} is the priority tree [13] or the improved version of the priority tree that appeared in [22]. The former allows $O(\log n + f)$ query time and the latter allows $O(\log n / \log\log n + f)$ query time, both using linear space.

We can handle counting queries in a similar fashion using as T^{t_1,t_2} Chazelle's linear space data structure to achieve $O(\log n)$ query complexity or the one in [16] with $O(n \log^{\epsilon} n)$ space and $O(\log n / \log\log n)$ query time. Since we have $m(m-1)/2$ (t_1, t_2)-pairs, Theorem 4 follows.

Theorem 4 *Given n objects, each of which is specified by the values of its attribute at m time instances, it is possible to design an indexing structure so that the reporting version of any two-sided query can be answered in $O(\log n / \log\log n + f)$ time using $O(nm^2)$ space for the indexing structure. The counting version can be handled in $O(nm^2)$ space and $O(\log n)$ query time, or $O(nm^2 \log^{\epsilon} n)$ space and $O(\log n / \log\log n)$ query time.*

The strategy described above can be extended to handle any arbitrary number d of attributes describing each object. Our general problem will be reduced to $O(m^2)$ $2d$-dimensional dominance queries. Using the results of [16], we obtain the following theorem.

Theorem 5 *The general temporal range query problem, with n objects, each with $d > 1$ attributes specified at m time instances, can be handled with a data structure of size $O(m^2 \cdot n \log^{\epsilon} n (\log n / \log\log n)^{2d-3})$ and a query time $O((\log n / \log\log n)^{2d-2} + f)$. The counting query can be handled in $O((\log n / \log\log n)^{2d-1})$ time using $O(m^2 \cdot n \log^{\epsilon} n (\log n / \log\log n)^{2d-2})$ space.*

Clearly the space used to handle two-sided queries, even in the case when $d = 1$, is quite high. An interesting problem is whether there exists a data structure whose size is $o(nm^2)$, such that the general temporal range search problem can be solved in time that is polylogarithmic in nm and proportional to the number of objects found. We provide a partial answer to this question by showing that this is indeed the case when m is extremely large.

Theorem 6 *Given n objects, each characterized by the values of its attribute at m time instances such that $m > n!$, it is possible to design an indexing structure such that the reporting version of any two-sided query can be answered in $O(\log^c n + f)$ time using an $o(nm^2)$ space.*

Proof:
For each pair of time instances j_1 and j_2, let $m_i^{j_1,j_2} = \min\{v_i^j | j \in [j_1..j_2]\}$, and $M_i^{j_1,j_2} = \max\{v_i^j | j \in [j_1..j_2]\}$. Let $(i_1^{j_1,j_2}, i_2^{j_1,j_2}, \ldots, i_n^{j_1,j_2})$ be the permutation of $(1, 2, \cdots, n)$ such that $m^{j_1,j_2}_{i_1^{j_1,j_2}} \le m^{j_1,j_2}_{i_2^{j_1,j_2}} \le \cdots \le m^{j_1,j_2}_{i_n^{j_1,j_2}}$. Similarly, let $(I_1^{j_1,j_2}, I_2^{j_1,j_2}, \ldots, I_n^{j_1,j_2})$ be the permutation of $(1, 2, \ldots, n)$ such that $M^{j_1,j_2}_{I_1^{j_1,j_2}} \le M^{j_1,j_2}_{I_2^{j_1,j_2}} \le \ldots \le M^{j_1,j_2}_{I_n^{j_1,j_2}}$. We define two mappings f^{j_1,j_2} and F^{j_1,j_2}, such that $i^{j_1,j_2}_{f^{j_1,j_2}(k)} = k$ and $I^{j_1,j_2}_{F^{j_1,j_2}(k)} = k$ for $k = 1, 2, \cdots, n$. Thus an object O_i corresponds to two numbers $f^{j_1,j_2}(i)$ and $F^{j_1,j_2}(i)$ that basically give the ranks of O_i for the time period $[j_1..j_2]$ with regard to its maximum and minimum values respectively. In other words, point $(f^{j_1,j_2}(i), F^{j_1,j_2}(i))$ is the representation of object O_i in the two-dimensional rank space corresponding to the time period $[j_1..j_2]$.

Note that there are at most $O(n!)$ permutations of $(1, 2, \ldots, n)$. Therefore at most $O((n!)^2)$ different point sets are possible for each pair of j_1 and j_2. During preprocessing time, we simply build one priority tree for each possible point set and construct an array of m^2 entries that indicate for each pair (j_1, j_2) the corresponding priority tree.

Since the query is given as (l_1, l_2, a, b), we have to map the numbers a and b to the rank space of (l_1, l_2) before the corresponding priority tree can be searched. Let a^{j_1,j_2} and b^{j_1,j_2} be the parameters used to search the appropriate priority tree. Then a^{j_1,j_2} is equal to the number of points that are always greater than or equally a during the time period $[l_1, l_2]$ and b^{j_1,j_2} is equal to the number of points that are always less than or equal to b in that period. These two numbers can be independently computed using the results in Section 4. Even without using the fusion tree, this step still can be done in $O(\log^2(nm))$ time using $O(nm \log(nm))$ space.

The storage cost for the priority trees and the array is $O(m^2 + n(n!)^2 + nm\log(nm)) = o(nm^2)$. Therefore the total storage cost is $o(nm^2)$. After the ranks of a and b are determined, the query can be answered in $O(\log n + f)$ time. Thus the total computational time is $O(\log^2(nm) + f)$.□

6 Conclusion

We have introduced in this paper a general class of problems involving temporal range queries, which seems to be widely applicable. We have shown that this problem can be reduced to a number of multidimensional dominance search problems, and hence can in principle be solved fast using nonlinear space data structures. Special cases for one-sided queries were shown to admit elegant solutions using linear size data structures and polylogarithmic query time. A simple intriguing problem is whether the two-sided version for $d = 1$ can be solved in polylogarithmic time using linear space. Note that this problem can easily be reduced to solving the one-sided version for $d = 2$, and hence it is somewhat the easiest problem to tackle next.

References

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *19th ACM Symposium on Principles of Database Systems*, pages 175–186, 2000.

[2] B. Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal on Computing*, 15(3):703–724, Aug. 1986.

[3] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–463, June 1988.

[4] B. Chazelle. Lower bounds for orthogonal range search I. The arithmetic model. *Journal of the ACM*, 37(3):439–463, 1990.

[5] B. Chazelle. Lower bounds for orthogonal range search I. The reporting case. *Journal of the ACM*, 37(2):200–212, 1990.

[6] J. R. Driscoll, N. Sarnak, D. Sleattor, and R. E. Tarjan. Make data structures persistent. *J. of Compu. and Syst. Sci.*, 38:86–124, 1989.

[7] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences*, 48:533–551, 1994.

[8] H. N. Gabow, J. L. Bently, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pages 135–143, 1984.

[9] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[10] S. Lanka and E. Mays. Fully persistent B^+-trees. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 426–435, 1991.

[11] C. Makris and A. K. Tsakalidis. Algorithms for three-dimensional dominance searching in linear space. *Information Processing Letters*, 66(6), 1998.

[12] Y. Manolopoulos and G. Kapetanakis. Overlapping B^+-trees for temporal data. In *Proceedings of the 5th Jerusalem Conference on Information Technology*, pages 491–498, 1990.

[13] E. M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, May 1985.

[14] M. A. Nascimento and J. R. O. Silva. Towards historical R-trees. In *Proceedings of the ACM Symposium on Applied Computing*, pages 235–240, Feb. 1998.

[15] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 331–342, 2000.

[16] Q. Shi and J. JaJa. Fast algorithms for 3-d dominance reporting and counting. Technical report, Institute of Advanced Computer Study (UMIACS), Unveristy of Maryland, 2003.

[17] Y. Tao and D. Papadias. Efficient historical R-trees. In *Proceedings of the 13th International Conference on Scientific and Statistical Database Management*, pages 223–232, 2001.

[18] T. Tzouramanis, Y. Manolopoulos, and M. Vassilakopoulos. Overlapping Linear Quadtrees: A spatio-temporal access method. In *Proceedings of the 6th ACM Symposium on Advances in Geographic Information Systems (ACM-GIS)*, pages 1–7, Bethesda, MD, 1998.

[19] T. Tzouramanis, M. Vassilakopoulos, and Y. Manolopoulos. Processing of spatio-temporal queries in image databases. In *Proceedings of the 3rd East-European Conference on Advances in Databases and Information Systems (ADBIS'99)*, pages 85–97, Sept. 1999.

[20] P. J. Varman and R. M. Verma. An efficient multiversion access structure. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):391–409, 1997.

[21] J. Vuillemin. A unifying look at data structures. *Communications of the ACM*, 23(4):229–239, 1980.

[22] D. E. Willard. Examining computational geometry, van Emde Boas trees, and hashing from the perspective of the fusion three. *SIAM Journal on Computing*, 29(3):1030–1049, 2000.