# Efficient Aggregation over Moving Objects*

Peter Revesz Yi Chen
Computer Science and Engineering Department
University of Nebraska-Lincoln
Lincoln, NE68588, USA
{revesz,ychen}@cse.unl.edu

## Abstract

*We study two types of aggregation queries over a set $S$ of moving point objects. The first asks to* count *the number of points in $S$ that are dominated by a query point $Q$ at a given time $t$. The second asks to find the* maximum *number of points in $S$ that are dominated by a query point at any time. These queries have several applications in the area of Geographic Information Systems and spatiotemporal databases. For the first query and any fixed dimension $d$, we give two different solutions, one using $O(\sqrt{N})$ time and $O(N)$ space and another using $O(\log N)$ time and $O(N^2)$ space, where $N$ is the number of moving points. When each of the points in $S$ is moving piecewise linearly along the the same line and the total number of pieces is $O(N)$, then we can do the count query in $O(\sqrt{N})$ time and $O(N)$ space.*

*For the second query, when all objects move along the $x$-axis we give a solution that uses $O(\log N)$ time and $O(N^2)$ space in the worst case. Our solutions introduce novel search structures that can have other applications.*

## 1. Introduction

Aggregation operators are frequently used in database queries. The efficiency of database queries with aggregate operators is well understood and studied in the context of traditional relational data. However, aggregation operators are also important for more complex data that cannot be represented in relational databases.

**Example 1.1** Suppose that a large company has a number of manufacturing plants $P_1, P_2, P_3, \ldots$. Each plant produces four different products $X_1, X_2, X_3$ and $X_4$. The profit at each plant for each product changes over time as shown in Table 1.1.

*This research was supported in part by NSF grant EIA-0091530 and a Gallup Research Professorship.

**Table 1. Profits for various plant and product combinations.**

| Id | X1 | X2 | X3 | X4 | T |
|---|---|---|---|---|---|
| 1 | $t^2+2t+10$ | 80 | $t+30$ | $5t-10$ | $t$ |
| 2 | $t^3-8t-10$ | $10t$ | $t^2-2t$ | $t^3-3t+4$ | $t$ |
| 3 | $t^2-50$ | $3t$ | $5t-10$ | $t-10$ | $t$ |
| 4 | $t^4-16$ | $7t$ | $5t^2$ | $t-30$ | $t$ |
| 5 | $t^3+81$ | $4t$ | $t^3-21$ | $t+10$ | $t$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

The company has the opportunity to buy a new plant $Q$ where profits are rising rapidly. The board of directors would approve the buy only if five years from now $Q$ will be more profitable for each product than 10 of the current plants.

In this case, the input relations $P(Id, X_1, X_2, X_3, X_4, T)$ and $Q(X_1, X_2, X_3, X_4, T)$ form a constraint database [10, 12, 16]. Therefore, we can find out how many plants are less profitable in 2007 by the following SQL query:

```
select count(Id)
from P, Q
where P.X1 < Q.X1 and
      P.X2 < Q.X2 and
      P.X3 < Q.X3 and
      P.X4 < Q.X4 and
      P.T  = 2007 and
      Q.T  = 2007;
```

Suppose that the company has a long-term plan to eliminate all products except $X1$. Therefore, the board of directors gives an approval for the purchase plan subject to the following extra condition: $Q$ should have the potential to some day be more profitable on product $X1$ than 20 of their current plants. We can find out the maximum number

Proceedings of the 10th International Symposium on Temporal Representation and Reasoning and
Fourth International Conference on Temporal Logic (TIME-ICTL'03) 1530-1311/03 $17.00 © 2003 IEEE

of plants that will be less profitable than $Q$ by the following SQL query:

```
select count(Id)
from P, Q
where P.X1 < Q.X1 and P.T  = Q.T
group by T
having count(Id) >= all
      (select count(Id)
       from P, Q
       where P.X1 < Q.X1 and P.T  = Q.T
       group by T);
```

While Example 1.1 can be extended to any higher dimension, many practical aggregation queries use only 1, 2 or 3-dimensional moving objects.

**Example 1.2** Consider a set of ships moving on the surface of the ocean. The locations of these ships are known by an enemy submarine which moves secretly underwater at constant depth. If the submarine fires, it calls attention to itself. Hence the submarine wants to wait until the maximum number of ships are within its firing range (which is some rectangle with the submarine in the center) before firing at as many ships as possible.

Let us assume that we have the relations $Ship(Id, X, Y, T)$ and $Range(X, Y, T)$, which describe the ships and the firing range of the submarine, respectively. A ship is in the firing range at a time instance if its $(X, Y)$ location is equal to a point in the $Range$ at the same time instance. Hence the above can be expressed by the following SQL query using a maximum aggregation operator.

```
select max(ship-count))
from  (select count(Id) as ship-count
       from   Ship, Range
       where  Ship.X = Range.X and
              Ship.Y = Range.Y and
              Ship.T = Range.T
       group-by T);
```

There are many alternatives to express in SQL the same query. For example, the above SQL query could be also expressed in by another SQL query that has a structure similar to the second SQL query in Exercise 1.1. It is a practical problem to recognize that these different SQL structures both express *max-count* queries, which we will define below. In this paper, we do not deal with the parsing problem.

**Example 1.3** We show in Figure 1 three cars driving along three path, which can be represented by piecewise linear constraints. We assume each car travels at a constant speed in each line segment. Assume a plane flying in the air keeps taking pictures of the ground, which is represented as the rectangular area in Figure 1. Given a time instance, find out how many cars will be covered in the picture at that time.

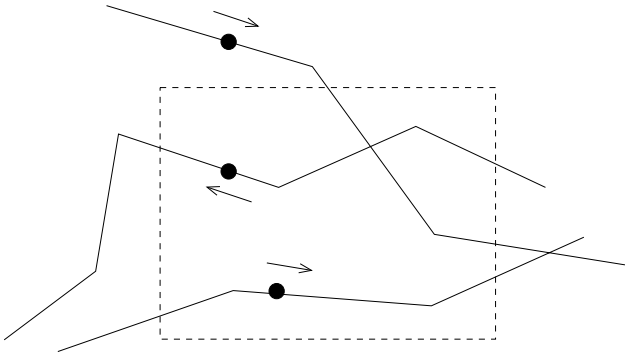

**Figure 1. Aggregations on piecewise linearly moving points**

Examples 1.1 and 1.2 are both cases of a group of frequently occurring aggregation problems where the input data can be visualized as a set $S$ of $N$ number of $k$-dimensional moving points. In Example 1.1 each point represents one plant and the value of the $i$th dimension represents the profit of the $i$-th product at that plant. In Example 1.2 each point represents one ship in 2-dimensions using latitude and longitude. In Example 1.3, the speed and direction of the cars change as they enter new line segments, but the movement can still be represented by piecewise linear constraints.

We say that point $P_i$ *dominates* point $P_j$ if and only if $P_i$ has a larger value than $P_j$ has for each dimension. Then the queries in Examples 1.1 and 1.2 can be generalized as follows:

*Count:* Given a moving point $Q$ and a time instance $t$, find the number of points in $S$ dominated by $Q$ at time $t$.

*Max-Count:* Given a moving point $Q$, find the maximum number of points in $S$ that $Q$ can dominate at any time.

In this paper we focus only on the above two aggregation queries, because several other more complex aggregation queries can be reduced to them or can be solved similarly. For example:

*Range-Count:* Given a time instance $t$ and two moving points $Q_1$ and $Q_2$, find the number of points in $S$ located in the hyper-rectangle defined by $Q_1$ and $Q_2$. (This reduces to a sequence of count queries.)

*Max-Time:* Given a moving point $Q$ and time instance $t$, find out whether $Q$ dominates at time $t$ the maximum possible number of points in $S$. (This reduces to testing whether the results of a *count* and a *max-count* query are the same.)

*Sum:* Assign a value to each moving point. Then given a moving point $Q$ and time instance $t$, find the sum of the values of the points in $S$ dominated by $Q$ at time $t$. (This requires only minor changes in the index structures that we develop for *count* queries.)

Aggregation queries can be evaluated in $O(N \log N)$ time and $O(N)$ space (see Appendix). However, this performance is not acceptable in applications where the input data is large and the query evaluation speed is critical, like in Example 1.2. The goal of this paper is to develop novel indexing structures that can greatly speed up *count* and *max-count* aggregate query evaluation.

There are some indexing structures for moving objects [1, 2, 5, 11, 17]. One may use these indices to answer the *count* and the *range-count* query by first finding the set of points $S' \subseteq S$ dominated by a new point $Q$ or being within a hyper-rectangle defined by $Q_1$ and $Q_2$, and then counting the number of points in $S'$. However, the counting may require linear time in the size of $S'$. Our goal is to find the count in logarithmic time. Further, these indices cannot be used to answer the *max-count* and *max-time* queries.

As shown by Zhang et al. [20], if we have a *static* set of points, then the *range-count* problem can be solved by generalizing some earlier work on dominance by Bentley [3]. Zhang and Tsotras [19] also considered the *max-count* aggregation problem for static sets of points in $S$. However, these methods are not easily generalizable to moving points, which is our focus in this paper. Lazaridis and Mehrotra [13] , Choi and Chung [6] and Tao et al. [18] study the approximation of aggregate queries for spatial and spatiotemporal data. In contrast to them, our algorithm produce precise answers without a significant loss in performance.

This paper is organized as follows. In Section 2, we review some basic concepts, including partition trees for indexing moving objects proposed by Agarwal et al. [1]. In Section 3, we consider two different methods for answering count aggregation queries. The first method extends partition trees, to *partition aggregation trees*. The second method uses a novel data structure called *dominance-time graphs*. Dominance-Time graphs are faster than partition aggregation trees and they can also be used when the position of moving points are represented by polynomial functions of time. In Section 4 we consider *max-count* aggregation queries. Finally, in Section 5 we discuss some open problems. Our main results are summarized in Table 2,

**Table 2. Computational complexity of aggregation on moving objects.**

| Query | I/O | S | D | Function | Method |
|---|---|---|---|---|---|
| Count | $\sqrt{N}$ | $N$ | d | linear | PA tree |
| Count | $log\ N$ | $N^2$ | d | polynomial | DT graph |
| Max | $log\ N$ | $N^2$ | 1 | linear | Dome subdiv |

where D means dimensions and S means space requirements.

## 2. Basic Concepts

We review two basic concepts. *Duality* [8] allows mapping $k$-dimensional moving points into $2k$-dimensional static points. *Partition Trees* proposed by Agarwal et al. [1] are search trees for moving points.

### 2.1 Duality

Suppose the positions of the moving points in each dimension can be represented by linear functions of time of the form $f(t) = a{\cdot}t{+}b$, which is a line in the plane. We may represent this line as a point $(a, b)$ in its dual plane. Similarly, a point $(c, d)$ can be represented as a line $g(t) = c{\cdot}t{+}d$ in its dual plane. Suppose line $l$ and point $P$ have dual point $L'$ and dual line $p'$ respectively. Then, $l$ is below $P$ if and only if $L'$ is below $p'$.

**Lemma 2.1** Let $P = a_P{\cdot}t{+}b_P$ and $Q = a_Q{\cdot}t{+}b_Q$ be two moving points in one dimensional space, and $P'(a_P, b_P)$ and $Q'(a_Q, b_Q)$ be their corresponding points in the dual plane. Suppose $P$ overtakes $Q$ or vice versa at time instance $t$, then

$$t = -\frac{b_P - b_Q}{a_P - a_Q}$$

Let $Slope(P'Q')$ denote the slope of the line $P'Q'$. Then we have $t = -Slope(P'Q')$, that is, $t$ is equal to the negative value of the slope of the line $P'Q'$.

Hence, given a time instance $t$, the problem of finding how many points are dominated by $Q$ reduces to the problem of finding how many points are below $l$, where $l$ is a line crossing $Q$ in the dual plane with the slope $-t$.

**Definition 2.1** Let $S$ be a set of $N$ points and $l$ be a line in the plane. We define the function **CountBelow(l)** as follows. If $l$ is a vertical line with $r_1$ points on the left and $r_2$ points on the right, then $CountBelow(l) = max(r_1, r_2)$. Otherwise, if $r$ number of points are below $l$, then $CountBelow(l) = r$.

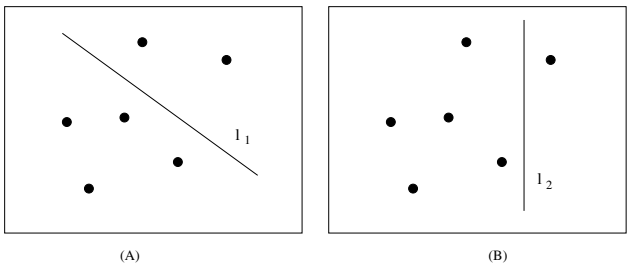


**Figure 2. Rank of a line.**

Note that Definition 2.1 is logical, because if $l$ is a vertical line, then we can always tilt it slightly left or right to get another line that has the same value of $CountBelow$ as we defined.

**Example 2.1** Figure 2 shows a set of points and two lines $l_1$ and $l_2$. There are four points below $l_1$, hence $CountBelow(l_1) = 4$. There are five points to the left and one point to the right of $l_2$, which is a vertical line. Hence $CountBelow(l_2) = 5$.

### 2.2 Partition Trees

Given a set $S$ of $N$ points in two dimensional space, we represent a *simplicial partition* of $S$ as $\Pi = \{(S_1, \Delta_1), (S_2, \Delta_2), ..., (S_m, \Delta_m)\}$, where $S_i$'s are mutually disjoint subsets of $S$ whose union is $S$, and $\Delta_i$ is a triangle that contains all points of $S_i$. For a given parameter $r$, $1 \leq r < N$, we say this simplicial partition is *balanced* if each subset $S_i$ contains between $N/r$ and $2N/r$ points.

Figure 3(A) shows an example of balanced simplicial partition for 35 points with $r = 6$. The *crossing number* of a simplicial partition is the maximum number of triangles crossed by a single line. The following is known about crossing numbers:

**Theorem 2.1 (Matousek[14])** Let $S$ be a set of $N$ points in the plane, and let $1 < r \leq N/2$ be a given parameter. For some constant $\alpha$ (independent of $r$), there exists a balanced simplicial partition $\Pi$ of size $r$, such that any line crosses at most $cr^{1/2}$ triangles of $\Pi$ for a constant $c$. If $r \leq N^{\alpha}$ for some suitable $\alpha < 1$, $\Pi$ can be constructed in $O(Nlog\ r)$ time.

Using Theorem 2.1, it is possible to recursively partition a set of points in the plane. This gives a partition tree.

## 3. Count Aggregation Queries

In this section, we first make a simple extension of partition trees, described in Section 3.1. With the modification, we can answer the *count* query in $O(\sqrt{N})$ time. Then we describe a more novel data structure, called an *dominance-time graph*, that needs only logarithmic time.

### 3.1 Partition Aggregation Trees

**Definition 3.1** Let $S$ be a set of $N$ points in $k$ dimensional space and $T$ be a multi-level partition tree for $S$. Let $v_i$ be an internal node in $T$, which stores a triangle $\Delta_i$. We attach a new value $A_i$ to node $v_i$, such that $A_i$ is the number of points in $S_i$. We call the new tree structure *Partition Aggregation Tree* (PA Tree).

**Theorem 3.1** *PA Tree* is a linear size data structure that answers the count query in $O(\sqrt{N})$ I/Os.

**Example 3.1** Figure 3(B) shows a partition tree with four top level triangles $A, B, C$ and $D$. The query line $q$ crosses two top level triangles $A$ and $B$. There are three second level triangles $A4, B2$ and $B3$ that are crossed by $q$. Figure 3(C) shows the structure of the PA-tree. For simplicity, we only show for each node the triangle name and the count of the points contained in that triangle.

To find $CountBelow(q)$, we start from the root of the PA-tree, load all top level triangles into memory and compare them to the query line $q$. Since both triangles $C$ and $D$ are below the line, we add the precomputed value to the result $CountBelow(q) = 12 + 17 = 29$. For the triangles $A$ and $B$, we traverse their children recursively. In this case, triangle $B4$ is below $q$, then we have $CountBelow(q) = CountBelow(q) + CountIn(B4) = 29 + 4 = 33$, where $CountIn(B4)$ is the number of points in the subset associated with $B4$. When we reach the leaf nodes of the PA-tree, we compare each point in the node with $q$, and add the number of points below $q$. There is one point in triangle $B3$ that is below $q$. Finally, the answer to the aggregation problem is 34. In Figure 3(C), we indicate using double sided rectangles those nodes that are accessed by this algorithm.

In Example 1.3, the movement of a car can be represented by piecewise linear functions. When the direction or speed changes, we may consider the car to be replaced by a new car with different direction or speed. We have the following theorem for the piecewise linearly moving points in one dimensional space:

**Theorem 3.2** Let $S$ be a set of piecewise linearly moving points with $N$ number of pieces in one dimensional space. The dominance-sum problem of $S$ can be answered in $O(\sqrt{N})$ I/Os with $O(N)$ space.

The above talks about one dimensional space. That may occur when each car is going on a straight highway, but each car may slow down in certain intervals due to road construction or heavy traffic, and they change direction only if they

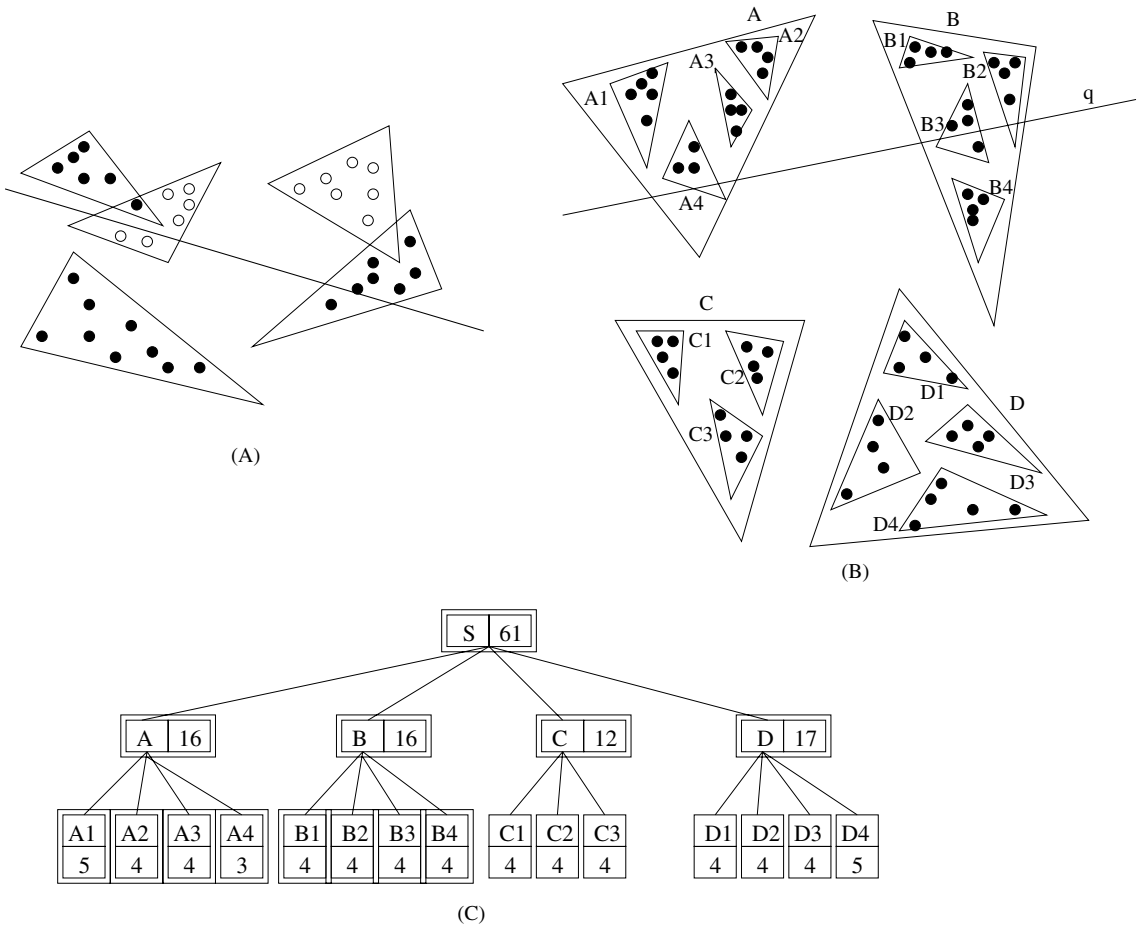


**Figure 3. A partition aggregation tree.**

make U-turns. It is an open problem to find a similarly efficient solution for two or higher dimensional space.

### 3.2 Dominance-Time Graph

Partition aggregation trees are limited because they only work when the points are moving linearly. In this section we introduce *dominance-time graphs*, a novel index data structure that can handle polynomial functions of time.

**Definition 3.2** For two $k$-dimensional moving points $P = (f_1, ..., f_k)$ and $Q = (g_1, ..., g_k)$, we say $P$ dominates $Q$ at time $t$, denoted as **dom(P,Q,t)**, if and only if $f_i(t) > g_i(t)$ for $1 \leq i \leq k$. If $P$ does not dominate $Q$ at time $t$, then we write **ndom(P,Q,t)**.

**Definition 3.3** Let $S$ be a set of $N$ moving points in $k$ dimensional space. The **dominance-time graph** $G(V, E)$ for $S$ is a directed labeled graph, such that for each point in $S$, there exists a corresponding vertex in $V$, and there is an edge in $G$ from $P$ to $Q$ labeled by the set of disjoint intervals $\{(a_1, b_1), ..., (a_m, b_m)\}$, if and only if $dom(P, Q, t)$ is true for time instance $t$ that is within any of the open intervals. Note that any real number and $-\infty$ and $+\infty$ are allowed as interval endpoints.

**Example 3.2** Suppose that we are given the following set of two dimensional moving points:

$$P_1 = (t + 10,\ t - 5)$$

$$P_2 = (2t,\ 2t - 10)$$

$$P_3 = (3t + 5,\ 3t - 15)$$

$$P_4 = (4t - 5,\ 0)$$

The dominance-time graph of these moving points is shown in Figure 3. Note that for any time instance $t \in (5, 10)$ the condition $dom(P_3, P_4, t)$ is true. Hence the edge from $P_3$ to $P_4$ is labeled $\{(5, 10)\}$. The labels on the other edges can be found similarly.

**Definition 3.4** Let $P$ and $Q$ be two moving points and $t_0$ and t be two time instances such that $t_0 < t$. We say that

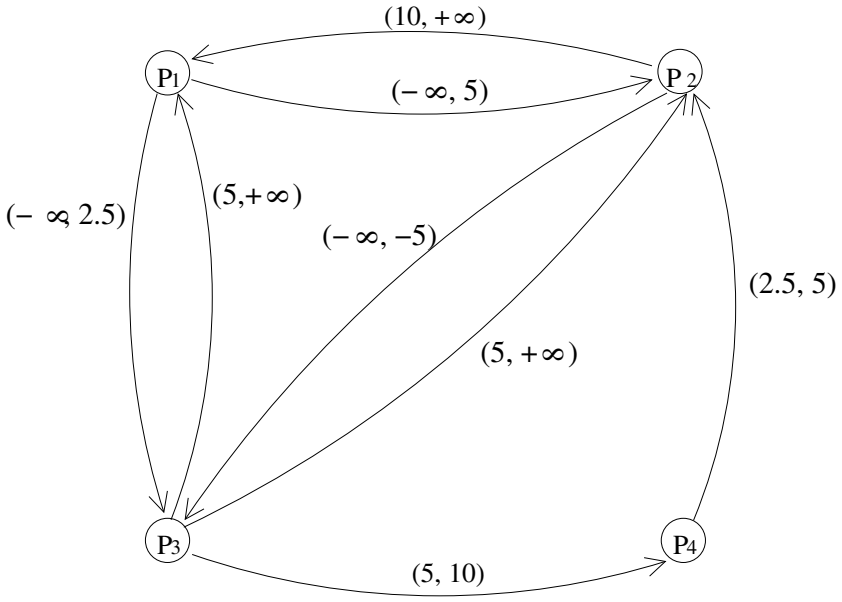


**Figure 4. A dominance-time graph.**

between $t_0$ and $t$ an increment event happens to $P$ with respect to $Q$ if $ndom(P, Q, t_0)$ and $dom(P, Q, t)$. Similarly, we say that between $t_0$ and $t$ a decrement event happens to $P$ with respect to $Q$ if $dom(P, Q, t_0)$ and $ndom(P, Q, t)$.

**Definition 3.5** Let **Rank(P,t)** be the number of points that are dominated by $P$ at time $t$.

**Lemma 3.1** An increment event happens to $P$ with respect to $Q$ if and only if there is an outgoing edge from $P$ that has a label in which no interval contains $t_0$ and some interval contains $t$. Similarly, a decrement event happens to $P$ with respect to $Q$ if and only if there is an outgoing edge from $P$ that has a label in which some interval contains $t_0$ and no interval contains $t$.

**Lemma 3.2** Let $t_0$ and $t$ be two time instances such that $t_0 < t$. Let $P$ be any vertex in a dominance-time graph. Let $m$ (and $n$) be the number of increment (and decrement) events that happen to $P$ with respect to different other vertices between $t_0$ and $t$. Then the following is true:

$$Rank(P, t) = Rank(P, t_0) + m - n$$

**Example 3.3** Table 3 shows the rank of each point of Example 3.2 at time instances $t = -8$ and $t = 12$. Note that $dom(P_2, P_3, -8)$ and $ndom(P_2, P_3, 12)$ are both true. Hence, an increment event happened to $P_2$ between time $t = -8$ and $t = 12$. Similarly, $ndom(P_2, P_1, -8)$ and $dom(P_2, P_1, 12)$ are also both true.Hence a decrement event happens to $P_2$ between the same times. Thus, according to Lemma 3.2, we have

$$Rank(P_2, 12) = Rank(P_2, -8) + 1 - 1 = 1$$

**Table 3. Location and rank of points at times $t = -8$ and $t = 12$.**

| Point | Location t = -8 | Rank t = -8 | Location t = 12 | Rank t = 12 |
|---|---|---|---|---|
| $P_1$ | (2, -13) | 2 | (22, 7) | 0 |
| $P_2$ | (-16, -26) | 1 | (24, 14) | 1 |
| $P_3$ | (-19, -39) | 0 | (41, 21) | 2 |
| $P_4$ | (-37, 0) | 0 | (43, 0) | 0 |

### 3.3 Time and Space Analysis

In this section we describe the basic structure of *dominance-time trees* and show how to use them to answer *count* aggregation queries in $O(log\, mN)$ I/Os, where $N$ is the number of moving points and $m$ is the maximum degree of the polynomial functions used to represent the position of the points.

A *dominance-time tree* for point $P$ is a $B$-tree to index the consecutive time intervals:

$$(-\infty, t_1), (t_1, t_2), \ldots, (t_i, t_{i+1}), \ldots, (t_n, +\infty)$$

such that during each interval $(t_i, t_{i+1})$, the rank of $P$ remains unchanged. The rank of $P$ during these intervals and the $t_i$ endpoints of these intervals can be precomputed and stored in the $B$-tree. Therefore, the $B$-tree can find the rank of $P$ for any time instance in $(-\infty, +\infty)$.

Let $S$ be a set of $N$ moving points. For any point $P$ in $S$, we may compute (precisely for polynomials up to degree 5

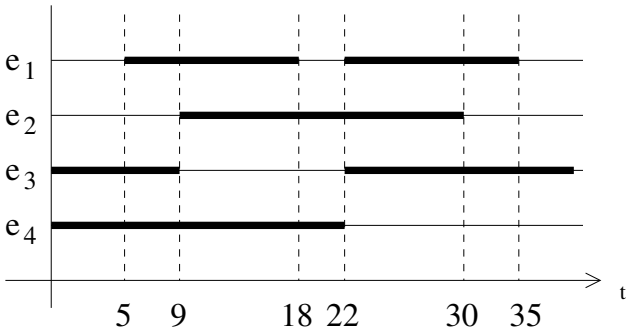


**Figure 5. A time line.**

and approximately for higher degree polynomials) a set of $n$ time instances $t_i$ $(1 \leq i \leq n)$ such that during each interval $(t_{i-1}, t_i)$ the rank of $P$ remains unchanged.

**Example 3.4** Suppose in a dominance-time graph, there are four outgoing edges, $e_1$, $e_2$, $e_3$ and $e_4$ for a point $P$. They are labeled as the following respectively:

$e_1 : (5, 18),\ (22, 35)$

$e_2 : (9, 30)$

$e_3 : (0, 9),\ (22, +\infty)$

$e_4 : (0, 22)$

Figure 5 shows the intervals contained in the labels with thick line segments. In this case, the $B$-tree contains the time instances $0, 5, 9, 18, 22, 30, 35$ and the following time intervals:

$(-\infty, 0)$,$(0, 5)$,$(5, 9)$,$(9, 18)$,$(18, 22)$, $(22, 30)$,$(30, 35)$, $(35, +\infty)$

**Definition 3.6** Suppose $G$ is a dominance-time graph for a set of moving points and $P$ is a vertex in $G$. A *Dominance-Time Tree* $T_P$ is a data structure based on a $B$-tree, which indexes all end points of time intervals contained in the labels of outgoing edges from $P$.

The leaf node of the dominance-time tree contains a list of consecutive time instances, $t_1, t_2, ..., t_b$, and $b + 1$ data fields $v_1, v_2, ..., v_{b+1}$ where $b$ is chosen according to the size of the disk pages. For each field $v_i$ for $1 \leq i \leq b$ we store the precomputed rank of $P$ during the interval $(t_{i-1}, t_i)$. Given a time instance $t$, the rank of $P$ can be found by searching the dominance-time tree until we find the leaf node with the interval that contains $t$.

Now we can prove the following.

**Theorem 3.3** Let $S$ be a set of $N$ moving points in $k$-dimensional space. Let $m$ be a fixed constant and assume that the position of each moving point in each dimension is represented by a polynomial function of degree at most $m$. Given a point $P$ in $S$ and a time instance $t$, the *Dominance-Time Tree* for each $P \in S$ requires $O(N)$ space. Hence the *count* aggregation problem can be done in $O(log_B\ N)$ I/Os using a total of $O(N^2)$ space.

The preprocessing of the dominance-time tree structure involves computation of polynomial functions. However, for a moving point which is represented by a polynomial function, it is not difficult to use piecewise linear functions to approximately represent its trajectory. Using this approximation method, the number of time intervals when the rank of a particular point remain unchanged will remain unchanged.

## 4. Max-Count Aggregation Queries

Our *max-count* aggregation algorithm uses a novel data structure built on the concept of *domes*, which we introduce here as a new type of spatial partition of the dual plane of a set of one-dimensional moving points. We start this section with a few definitions.

**Definition 4.1** Let $S$ be any set of points in the plane. For any new point $Q$, we define **MaxBelow(Q)** to be the maximum number of points below any line that passes through $Q$.

**Definition 4.2** Let $S$ be any set of points in the plane. Let $L$ be the set of lines that cross at least two points in $S$ or cross at least one point in $S$ and are vertical. For $0 \leq i \leq N$, we define $L_i = \{l \in L | CountBelow(l) + CountOn(l) \geq i\}$, where $CountOn(l)$ is the number of points in $S$ crossed by line $l$.

**Definition 4.3** For any line $l$, let $Below(l)$ be the half-plane below $l$, or if it is a vertical line, then the half-plane on that side of the line that contains more points. Let **Below($L_k$)** be the intersection of the half-planes associated with the lines in $L_k$. Let **k-dome**, denoted as $d_k$, be the boundary of the region $Below(L_k)$.

The intuition is that any point above $d_k$ has a line through it with at least $k$ points below.

**Definition 4.4 Layer(k)**$= \{Q | Q \in Below(L_{k+1})$ and $Q \notin Below(L_k)\}$.

**Example 4.1** We show in Figure 6 a set of seven points. In this case, $L_7$ is composed of the dotted lines (i.e., the lines crossing $P_2P_3$, $P_3P_4$,$P_4P_5$ and the two vertical lines

crossing $P_2$ and $P_5$), while $L_6$ is composed of the union of the dotted and dashed lines (i.e, the lines crossing $P_2P_7$, $P_3P_5$, $P_4P_6$, $P_4P_7$ and the two vertical lines crossing $P_3$ and $P_6$). The two thick polygonal lines in the figure are $d_7$ and $d_6$, respectively, and $Layer(6)$ is the area between them.

Now we prove some properties of the above concepts.

**Lemma 4.1** For any $i$ and $j$ such that $i \leq j$, the following hold.
(1) $L_i \subseteq L_j$.
(2) $Below(L_i) \subseteq Below(L_j)$.
(3) no point of dome $d_i$ is above any point of dome $d_j$.

**Lemma 4.2** Layer(k) consists of those points that are strictly outside $d_k$ and on or inside $d_{k+1}$.

**Lemma 4.3** Each point belongs to only one layer.

We can now show the following characterization of layers.

**Theorem 4.1** $Q \in Layer(m) \leftrightarrow MaxBelow(Q) = m$.

Theorem 4.1 implies that the layers partition the plane in such a way that there is a one-to-one correspondence between any element of the partition and the $MaxBelow$ value of the points in that element. We can use this theorem to build a data structure for efficiently identifying which element of the partition a new point is located in, using the following well-known result from computational geometry.

**Theorem 4.2** [15] Point location in an N-vertex planar subdivision can be effected in $O(log\ N)$ time using $O(N)$ storage, given $O(Nlog\ N)$ preprocessing time.

**Lemma 4.4** Any dome $d_k$ has $O(N)$ edges.

**Lemma 4.5** Let $S$ be any set of $N$ points in the plane and $Q$ a query point. Then we can find in $O(\log N)$ time using an $O(N^2)$ space data structure $MaxBelow(Q) = m$.

**Lemma 4.6** Let $S$ be a set of $N$ points and $Q$ a query point moving along the $x$ axis. Let $S'$ and $Q'$ be the duals of $S$ and $Q$, respectively. Then the following hold.

(1) For any time instance $t$ the moving point $Q$ dominates $CountBelow(l)$ number of points in $S$, where line $l$ crosses $Q'$ and has slope $-t$. (2) The maximum number of points that $Q$ dominates is $MaxBelow(Q')$.

Finally, we have the following theorem.

**Theorem 4.3** The *Max-Count* aggregation query can be answered using an $O(N^2)$ size data structure in $O(\log N)$ query time and $O(N^2 \log N)$ preprocessing time.

The above considers only objects that exist at all times. Suppose that objects only exist between times $t_1$ and $t_2$. That means that only lines passing $Q$ and having slopes between $-t_2$ and $-t_1$ are interesting solutions. Let $L_i^{(t_1,t_2)}$ be the modification of $L_i$ that allows only lines that have slopes between $-t_2$ and $-t_1$ and cross two or more points or cross only one point and have slopes exactly $-t_2$ or $-t_1$. With this modification, we can correspondingly modify the definition of layers. Then Theorems 4.1 and 4.3 still hold.

## 5. Further Work

There are several interesting open problems. We list below a few of these.

1. Are there *count* or *max-count* aggregation algorithms that are more efficient in time or space than our algorithms, or can a tight lower bound be proven for these aggregation problems?

2. Can the *count* aggregation algorithm for piecewise linear moving points in one dimension be extended to higher dimensions while keeping the $O(\sqrt{N})$ time and $O(N^2)$ space in the worst case?

3. Can the *max-count* aggregation algorithm in one dimension be extended to higher dimensions while keeping the $O(\log N)$ time and $O(N^2 \log N)$ space in the worst case?

4. How can we make the data structures dynamic, that is, allow efficient deletions and additions of new moving points? We have partial solution to this problem when only insertions are considered.

5. What is the average case of the *count* and *max-count* algorithms?

6. Can the algorithms be improved by considering approximations? As described in Section 1, approximations for the *count* aggregation query were considered in the work of [13, 6, 18]. However, there are no approximation algorithms for the *max-count* aggregation problem.

7. Moving objects can be represented not only by moving points but also by *parametric rectangles* [4], by *geometric transformation objects* [7, 9], or by some other constraint representation [12, 16]. These constraint representations are more general because they also represent the changing (growing, shrinking) shape of the objects over time. It is possible to consider *count* and *max-count* aggregation queries on these more general moving objects. Is it possible to solve these queries within the same time complexity?

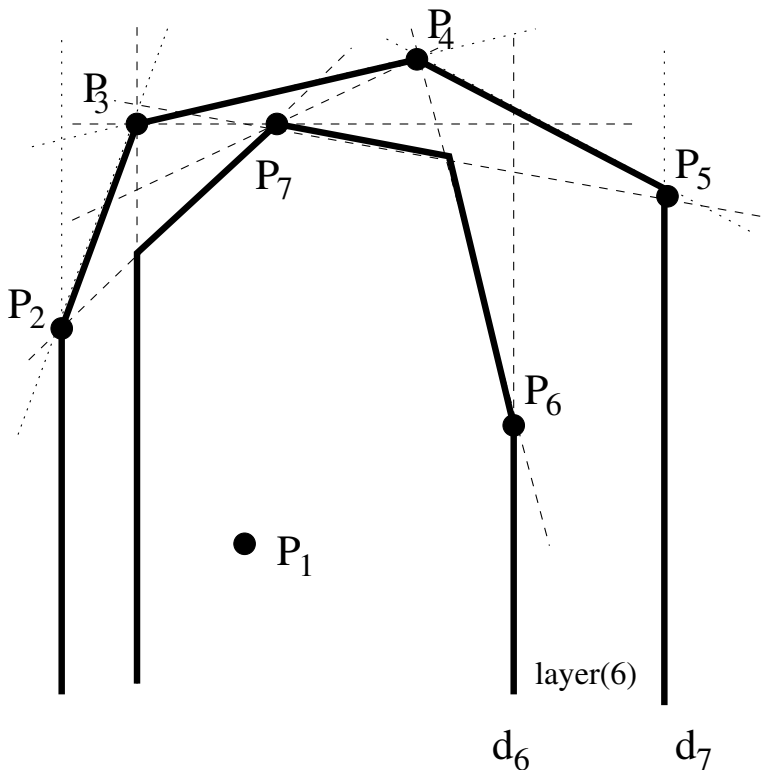


**Figure 6.** $Layer(6)$ **for seven points.**

We are also interested in implementations of these algorithms and testing them on real data, for example, aviation data sets, and truck delivery data sets.

## References


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

[2] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1997.

[3] J. L. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4), 1980.

[4] M. Cai, D. Keshwani, and P. Revesz. Parametric rectangles: A model for querying and animating spatiotemporal databases. In *Proc. 7th International Conference on Extending Database Technology*, volume 1777, pages 430–44. Springer-Verlag, 2000.

[5] M. Cai and P. Revesz. Parametric r-tree: An index structure for moving objects. In *Proc. 10th COMAD International Converence on Management of Data*, pages 57–64, 2000.

[6] Y.-J. Choi and C.-W. Chung. Selectivity estimation for spatio-temporal queries to moving object s. In *SIGMOD*, 2002.

[7] J. Chomicki and P. Revesz. A geometric framework for specifying spatiotemporal objects. In *Proc. International Workshop on Time Representation and Reasoning*, pages 41–6, 1999.

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer Verlag, Berlin, 1997.

[9] S. Haesevoets and B. Kuijpers. Closure properties of classes of spatio-temporal objects under Boolean set operations. In *Proc. International Workshop on Time Representation and Reasoning*, pages 79–86, 2000.

[10] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, 1995.

[11] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *ACM Symp. on Principles of Database Systems*, pages 261–272, 1999.

[12] G. Kuper, L. Libkin, and J. Paredaens. *Constraint Databases*. Springer Verlag, 2000.

[13] L. Lazaridis and S. Mehrotra. Progressive approximate aggregate queries with a multi-resolution t ree structure. In *SIGMOD*, 2001.

[14] J. Matousek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.

[15] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer Verlag, New York, 1985.

[16] P. Revesz. *Introduction to Constraint Databases*. Springer Verlag, 2002.

[17] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD Conference*, pages 331–342, 2000.

[18] Y. Tao, J. Sun, and D. Papadias. Selectivity estimation for predictive spatio-temporal queries. In *ICDE*, 2003.

[19] D. Zhang and V. J. Tsotras. Improving min/max aggregation over spatial objects. In *ACM-GIS*, pages 88–93, 2001.

[20] D. Zhang, V. J. Tsotras, and D. Gunopulos. Efficient aggregation over objects with extent. In *Symposium on Principles of Database Systems*, pages 121–132, 2002.

Proceedings of the 10th International Symposium on Temporal Representation and Reasoning and
Fourth International Conference on Temporal Logic (TIME-ICTL'03) 1530-1311/03 $17.00 © 2003 IEEE