# AGGREGATION OPERATIONS ON SPATIOTEMPORAL DATABASES

by

Yi Chen

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Doctor of Philosophy

Major: Computer Science

Under the Supervision of Professor Peter Revesz

Lincoln, Nebraska

December, 2003

# AGGREGATION OPERATIONS ON SPATIOTEMPORAL DATABASES

Yi Chen, Ph.D.

University of Nebraska, 2003

Advisor: Peter Revesz

The efficient evaluation of aggregation queries is key to the success of relational database systems and Geographic Information Systems. However, the aggregation queries for spatiotemporal databases that represent a set of moving point objects is a relatively new area. In this dissertation, we provide for the first time efficient aggregation algorithms for spatiotemporal databases. Our algorithms introduce several novel data structures called *Partition Aggregation Trees*, *Dominance-Time Graphs*, and *Dome Subdivisions* that are also interesting on their own and could be used for solving other problems beyond aggregation queries.

We also propose a novel mediation system architecture for spatiotemporal data. The new architecture makes it possible to collect and summarize the information from heterogeneous data sources. We also propose within the architecture a subsystem called DataFoX, that can evaluate Datalog-like queries on constraint databases and spatiotemporal XML documents in either the VML or the GML format. DataFoX also supports our new spatiotemporal aggregation operations that are not supported in other database and Geographic Information systems.

# ACKNOWLEDGEMENTS

I thank Professor Peter Revesz, my academic advisor, who influenced my research with many original ideas. Without his patient guidance this thesis would not be possible.

I thank also Professors Bogardi, Reichenbach and Surkan, the other members of my Ph.D. dissertation committee. They have advised me during various stages of this work.

Many thanks to Professor Yuanzhen Wang, my M.S. thesis advisor at the Huazhong University of Science and Technology, for introducing me to database systems and encouraging me to pursue a Ph.D. in the United States. I also want to thank many of my fellow students at the Department of Computer Science and Engineering at the University of Nebraska–Lincoln. In particular, Xin Liu, Lin Lin, Shasha Wu, and Mingtian Ni helped me in many ways during my years at the University of Nebraska–Lincoln.

I would like to thank Maggie, my wife, for her understanding and support during the past years. Finally, my parents receive my deepest gratitude and love for their dedication throughout my whole life. I dedicate this dissertation to them.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

Geographic Information Systems (GIS) [23, 39, 62] increasingly are important in various industries including digital government, e-commerce [14], and telecommunications. Many GIS applications require efficient manipulation of spatiotemporal information for efficient decision support [30, 6, 33].

There are various query languages used in GIS. These query languages often contain *aggregate operators* such as *average*, *count*, *max*, and *area*, which take in a set of values and return a single value. For example, a GIS concerned with traffic monitoring may need to *count* during a given time interval the total number of vehicles that enter a highway at a particular entrance. Efficient evaluation of aggregate operators is essential for the effective database systems and GIS.

Aggregation operations are very important for database related applications, especially in GIS systems and decision support systems. The efficiency of database queries with aggregate operators is well understood and studied in the context of traditional relational data. However, aggregation operators also are important for

more complex data that cannot be represented in relational databases.

The efficient evaluation of aggregate operators is always based on some indexing structure [5, 37, 44, 53, 57] for spatial and spatiotemporal data. For example, Bentley [7] describes the ECDF-tree data structure that enables efficient evaluation of some spatial aggregation operators on multi-dimensional set of points. Zhang et al. [63] extended Bentley's structure to other aggregate operators that deal with static spatial objects that are hyperrectangles.

However, aggregation on moving objects was not considered by either Bentley [7] or Zhang et al. [63]. Some more recent papers deal with moving objects, but only when those objects are moving along a line or a fixed set of lines [44]. That is still too limited for expressing many practical problems. For example, suppose several vehicles are move in the plane with fixed speed and direction, and we need to find out what is the maximum number of cars in a certain "window" or rectangular area at any time. Many indexing structures can answer such *window queries* on static objects or those that move along a line, but they fail to answer this query. There are no efficient proposed solutions for this spatiotemporal aggregation problem. Hence the study of aggregation operators is still an important problem, and it is a main topic of this thesis.

It is difficult to answer complex queries that need to use information together from several different Internet sites, beacuse of the great variety of data models in GIS [11, 56, 60]. If different sites are using different data models, then some data needs to be translated from one data model to another, which is commonly referred to as the problem of *data interoperability*. While *eXtensible Markup Language* (XML) is touted as a standard data model for data representation and exchange on the Internet, even XML has several different versions to describe spatial data. In this thesis we consider two of these versions: the *Vector Mark-up Language* (VML) and the

*Geography Markup Language* (GML), which is proposed by the OpenGIS consortium of companies.

The main challenges in this dissertation are the development of efficient algorithms to evaluate aggregation operators on spatiotemporal data, and the interoperability of spatiotemporal data. These issues are studied first in a theoretical sense. Second, the best algorithms also are implemented in a prototype system called DataFox. A detailed description of contributions is listed in Section 1.4, after a brief review of some related work.

## 1.2 Related Work for Chapters 2-4

### 1.2.1 Constraint Databases

*Constraint databases*, introduced by Kanellakis, Kuper & Revesz [35], is still a growing research area. Kuper, Libkin & Paredaens [38] and Revesz [48] are two recent books on constraint databases. Constraint databases generalize relational databases by finitely representable infinite relations. In the constraint data model, each attribute is associated with an attribute variable and the values of the attributes in a relation are specified implicitly using constraints.

A *constraint database* is a finite set of constraint relations. A *constraint relation* is a finite set of constraint tuples, where each *constraint tuple* is a conjunction of *atomic constraints* using the same set of attribute variables. For example, linear inequality constraints are considered atomic constraints. A conjunction of linear inequality constraints is a constraint tuple.

**Example 1.2.1** 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 1.2.1. In this table, each of the rows is a polynomial constraint tuple.

Table 1.1: Profits for various plant and product combinations.

| **Id** | **X1** | **X2** | **X3** | **X4** | **T** | |
|---|---|---|---|---|---|---|
| 1 | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $t$ | $x_1 = t^2 + 2t + 10$, $x_2 = 80$, $x_3 = t + 30$, $x_4 = 5t - 10$ |
| 2 | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $t$ | $x_1 = t^3 - 8t - 10$, $x_2 = 10t$, $x_3 = t^2 - 2t$, $x_4 = t^3 - 3t + 4$ |
| 3 | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $t$ | $x_1 = t^2 - 50$, $x_2 = 3t$, $x_3 = 5t - 10$, $x_4 = t - 10$ |
| 4 | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $t$ | $x_1 = t^4 - 16$, $x_2 = 7t$, $x_3 = 5t^2$, $x_4 = t - 30$ |
| 5 | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $t$ | $x_1 = t^3 + 81$, $x_2 = 4t$, $x_3 = t^3 - 21$, $x_4 = t + 10$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |

**Example 1.2.2** We show in Figure 1.1 a L-shape park and it's two different representations in constraint database. We use two different constraint database tables, as shown in Table A and Table B in the following, to represent this spatial object.

Table A

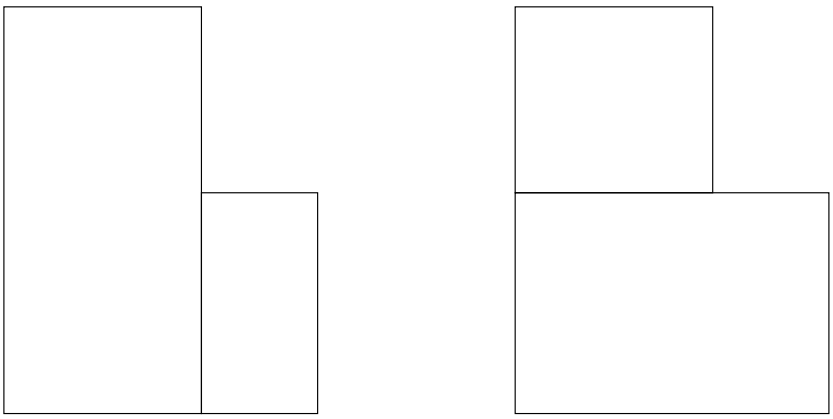
| **X** | **Y** | |
|---|---|---|
| x | y | $0 \leq x \leq 20$, $0 \leq y \leq 40$ |
| x | y | $20 \leq x \leq 30$, $0 \leq y \leq 20$ |

Table B

| **X** | **Y** | |
|---|---|---|
| x | y | $0 \leq x \leq 30$, $0 \leq y \leq 20$ |
| x | y | $0 \leq x \leq 20$, $20 \leq y \leq 40$ |

### 1.2.2 SQL

SQL is the standard query language for relational database systems. The SQL query language does not support spatio-temporal queries directly. However, it can be extended to query spatio-temporal information represented by constraint databases [35, 31, 46, 47].



(A) (B)

Figure 1.1: Different representations for the Park

We assume some familiarity with the SQL language [35, 46]. However, we illustrate below with a few examples how SQL can be applied to the constraint database shown in Table 1.2.1.

**Example 1.2.3** 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 [35, 38, 48]. Therefore, we can find out how many plants are less profitable in 2008 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  = 2008 and
      Q.T  = 2008;
```

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 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 1.2.1 can be extended to any higher dimension, many practical aggregation queries use only one, two or three dimensional moving objects.

**Example 1.2.4** 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 $Ship(Id, X, Y, T)$ and $Range(X, Y, T)$ be two relations, 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 by another SQL query that has a structure similar to the second SQL query in Example 1.2.1.

### 1.2.3 Datalog

Datalog is a rule-based query language, which is a natural language for querying constraint databases. Datalog has some features that SQL does not have. For example, Datalog support recursive queries.

**Example 1.2.5** Consider the the following query on the constraint database shown in Example 1.2.2:

*Compute the intersection of the park with a rectangular area represented by relation Rectangle(x,y).*

For this, we may use the following Datalog query:

$Intersect\_area(x, y) : -Park(x, y), Rectangle(x, y)$

**Example 1.2.6** Compute the area of the L-shape object shown in Example 1.2.2.

$Park\_Area(area < x, y >) : -Park(x, y).$

Every spatial object on the map is assumed to have two spatial attributes, $x$ and $y$. The area operator, denoted as *area<x,y>*, takes the relation as the input, and calculates the area of the object represented by the relation.

**Example 1.2.7** Let a spatiotemporal database relation `LinconCityMap(id, x, y, t)` represent the growing city. The following query finds the area of the city in the year 2003:

*LincolnArea(area<x,y>) :- LincolnCityMap(id, x, y, 2003).*

### 1.2.4 Spatial and Spatiotemporal Aggregate Operations

Many recent papers study aggregate operations for multi-dimensional point datasets [34, 59, 13, 20, 29].

Some common definitions related to aggregate operators follow.

**Definition 1.2.1** (**dominance**) Given two $d$-dimensional points $x = (x_1,...,x_d)$, $y = (y_1, ..., y_d)$, $x$ *dominates* $y$ if $x_i > y_i$ for every $i$, $1 \leq i \leq d$.

**Definition 1.2.2** (**dominance-sum**) Given a set $S$ of points and a query point $Q$, compute the *sum* of the points in $S$ that are dominated by $Q$.

**Example 1.2.8** Figure 1.2 shows a set $S$ of eight points in the two dimensional space, and a query point $Q$. Five points are dominated by $Q$ in $S$. That is, $dom-sum(Q) = 5$.

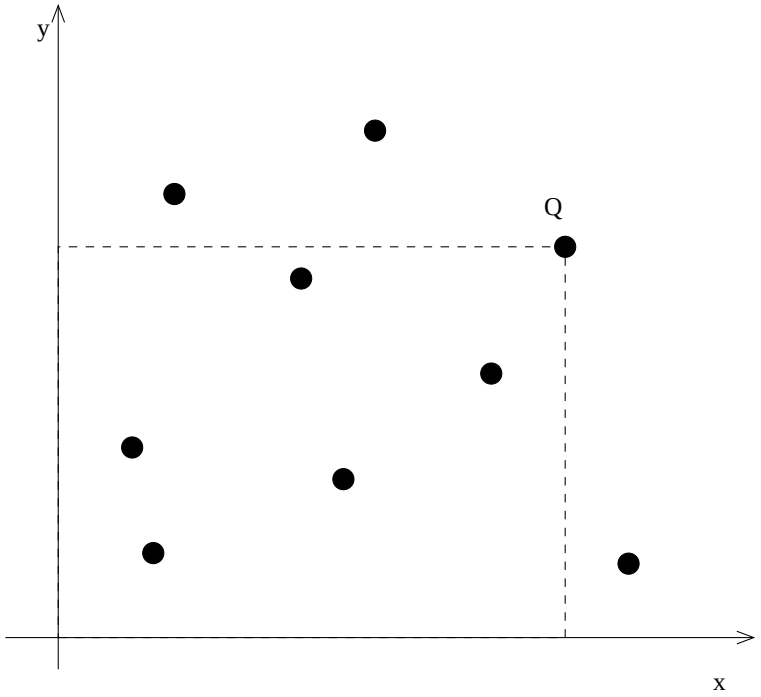



Figure 1.2: Dominance-sum for static points

**Definition 1.2.3** (**range-sum**) Given a set $S$ of points and a query box $q$, compute the *sum* of all points in $S$ that are contained in $q$.

Range-sum can be reduced to dominance-sum. ECDF-tree [7] is a multi-level data structure that answers the dominance-sum queries. The main branch of a *d*-dimensional ECDF-tree is a binary search tree whose leaves store the data points, ordered by their position in the first dimension. Each internal node of the main branch stores a pointer, which points to a *(d-1)*-dimensional ECDF-tree. This sub-level data structure is called the *border*.

**Example 1.2.9** Figure 1.3 (A) shows a set of eight points in the two dimensional space. $b_1$ is the $x$ coordinate of the vertical line, such that half of points in $S$ lie to the left of the line and the rest lie to the right. All points to the left of the vertical line $x = b_1$ are sorted by their $y$-coordinates and stored in the border as shown in Figure 1.3 (D). For a given query point $q$, $q.x > b_1$, three points in the border, $p_3$, $p_5$ and $p_7$ have smaller $y$-coordinates then $q$, hence they are dominated by $q$.

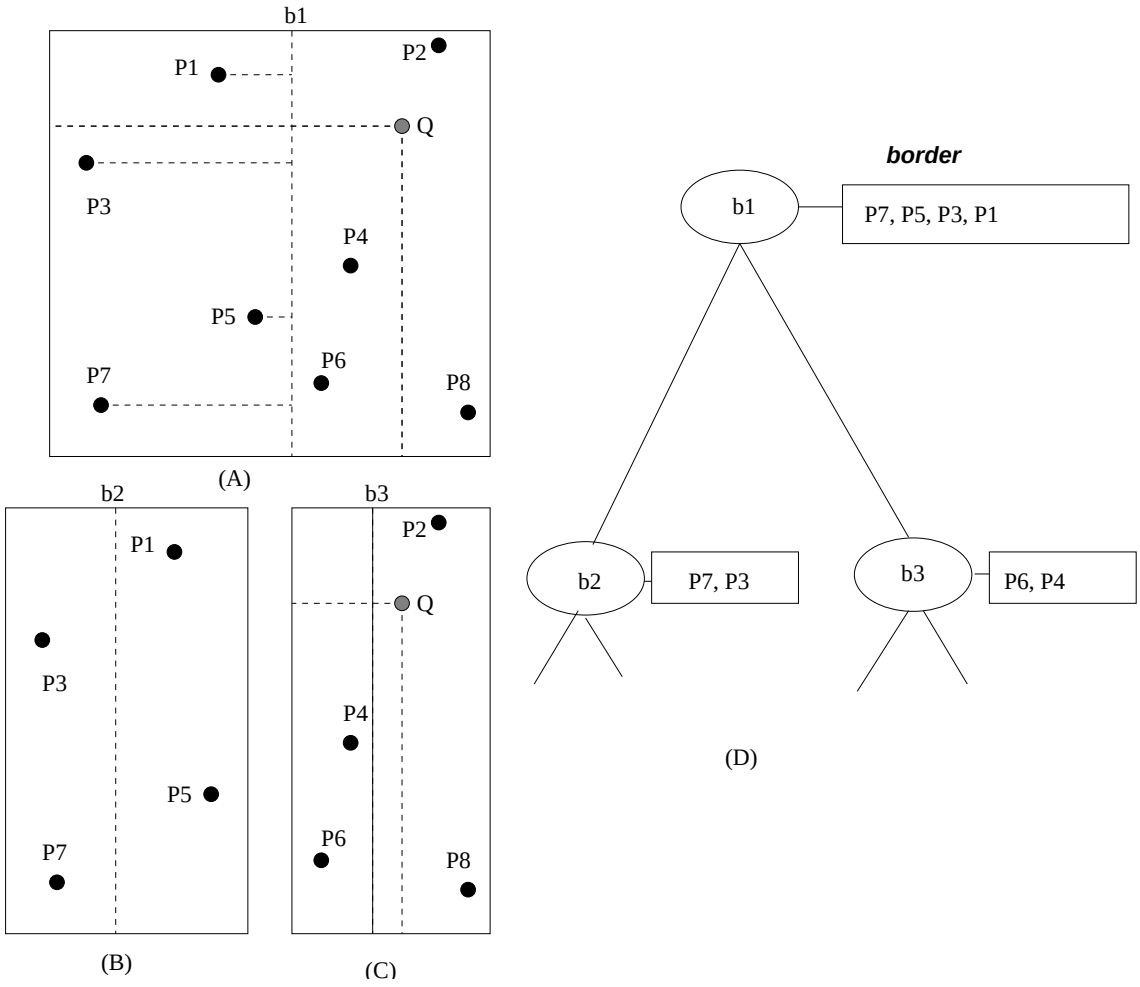



Figure 1.3: ECDF tree

Zhang et al. [63] addressed *box-sum* aggregations like *sum*, *count* and *avg* with related to spatial objects with extent. A typical box-sum query is: "*Find the total volume of pesticide sprayed in Orange County for March 1999.*"

The following spatial aggregations are also addressed:

**Definition 1.2.4** Given a set of objects, each having a box and a value function, and a query box $q$, compute the total value of all objects that intersect $q$, where the value contributed by an object $r$ is the integral of the value function of $r$ over the intersection between $r$ and $q$.

## 1.3 Related Work for Chapter 5

### 1.3.1 XML and Spatiotemporal XML

The *eXtensible Markup Language* (XML) is a tag-based notation for "marking" documents. It is a simple and flexible text format derived from SGML [1]. Originally designed to meet the challenges of large-scale electronic publishing, XML is playing an increasingly important role in the exchange of a wide variety of data on the Internet.

#### XML

Elements are the basic content units in XML. An element may contain character data, or other elements. Element tags in XML are defined by text surrounded by angle brackets, i.e., <...>. Tags generally come in matching pairs, with a beginning tag and a matching ending tag that is the same text starting with a slash.

**Example 1.3.1** In the following sample XML document, the root tag is `addressbook`. There are two entries surrounded by the tags <`contact`> and </`contact`>. There is an `email` element for the first entry and a `phone` element for the second entry. Each of the `name` elements is surrounded by the tags <`name`> and </`name`> and contains two sub-elements, namely `first-name` and `last-name`, which contain only character data.

```
<addressbook>
   <contact>
       <name>
           <first-name> Yi </first-name>
           <last-name> Chen </last-name>
       </name>
       <email> ychen@cse.unl.edu </email>
   </contact>
   <contact>
       <name>
           <first-name> Lin </first-name>
           <last-name> Lin </last-name>
       </name>
       <phone> 402-742-7719 </phone>
   </contact>
</addressbook>
```

### Document Type Definition

XML documents are required to satisfy a *Document Type Definition* (DTD), which specify components that are available for a particular type of document and the way those components can be mixed in order to produce a valid instance.

The general structure of a DTD is:

<!DOCTYPE *root-tag*[

<!ELEMENT *element-name (components)*>

⋮

]>

The *root-tag* is the name of a document. An *element* is described by its name, which is the tag used to surround portions of the document that represent that element, and a parenthesized list of components. The latter are tags that must appear within the tags for the element being described. `(#PCDATA)` after an element name means that element has a value that is text.

**Example 1.3.2** The following DTD specifies the structure of `addressbook.xml`.

The root element is `addressbook`, which contains zero or more `contact` elements. The first element definition says that inside the matching pair of tags `<addressbook>...</addressbook>` we find zero or more `contact` tags, each representing the contact information of a person.

```
<!DOCTYPE addressbook [
  <!ELEMENT addressbook  (contact*)>
  <!ELEMENT contact      (name, phone*, email*)>
  <!ELEMENT name         (first-name, last-name)>
  <!ELEMENT phone        (#PCDATA)>
  <!ELEMENT email        (#PCDATA)>
  <!ELEMENT first-name   (#PCDATA)>
  <!ELEMENT last-name    (#PCDATA)>
]>
```

### Spatial XML Documents

Several XML-based languages have been proposed for both two dimensional vector rendering, and for encoding geographic data. Among these the *Vector Markup Language* (VML) format provides a mechanism for encoding graphic primitives for rendering in a Web browser. For example, VML is implemented in Microsoft Internet Explorer. It is also the graphics interchange format within the Microsoft Office 2000 suite. The *Geographic Markup Language* (GML) provides a set of semantic tags for encoding coordinates of OpenGIS features.

### VML

*Vector Markup Language (VML)* is an XML-based exchange, editing, and delivery format for high-quality vector graphics on the Web. VML is currently supported by

Microsoft Internet Explorer 5 or greater for Windows 95, Windows 98, and Window NT 4.0 or greater.

VML was proposed as a standard for vector graphics on the Web [40]. VML is supported by Microsoft Office 2000 Beta 2 or greater. Microsoft Word, Microsoft Excel, and Microsoft PowerPoint can be used to create VML graphics.

Within VML the content is composed of paths described using connected lines and curves. The markup gives semantic and presentation information for the paths.

Basic types for VML include `boolean`, `fraction`, `ordinate`, `length`, `measure`, `angle`, `color`, `font`, `bitmap`, and `vector`.

**Example 1.3.3** The following code draws a rectangle with width 150 and height 50 and fills it with yellow color.

```
<html xmlns:v="urn:schemas-microsoft-com:vml"> <head>
 <title>Simple VML Example</title>
 <style>
   v:* {behavior: url(#default#VML);}
 </style>
</head>

<body>
  <v:rect style = "width:150pt;height:50pt" fillcolor="yellow">
  </v:rect>
</body>
</html>
```

The advantages of VML can be summarized as follows:

- VML makes writing easier yielding greater productivity for users and authors.

It facilitates the exchange and subsequent editing of vector graphics between a wide variety of productivity and design applications.

- VML provides faster graphic downloads and a better user experience. It allows the delivery of high-quality, fully integrated, scalable vector graphics to the Web, in an open text-based format.
- VML is open and standards-based. It is an XML-based format.

**GML**

*Geography Markup Language* (GML) [21] is an XML-based encoding standard for geographic information developed by the OpenGIS Consortium. A digital representation of the real world can be thought of as a set of features. The state of a feature is defined by a set of properties, where each property can be thought of as a {name, type, value} triple. The number of properties a feature may have, together with their names and types, are determined by its type definition.

GML encoding already allows for quite complex features. A feature can, for example, be composed of other features. A single feature like an airport might be composed of other features such as taxi ways, runways, hangers and air terminals. The geometry of a geographic feature can also be composed of many geometric elements. A geometrically complex feature can thus consist of a mix of geometric types, including:

- Point
- LineString
- LinearRing
- Polygon
- MultiPoint

- MultiLineString
- MultiPolygon
- MultiGeometry

In addition, there are `coordinates` and `coord` elements for encoding coordinates. There is also a `Box` element for defining the dimension of rectangle objects.

**Example 1.3.4** The following GML feature describes a lecture building. This `Feature` contains the properties `NumFloors` and `NumStudents`, and a basic geometry class `Box`. The `Box` element is defined by the coordinates of the lower-left and the upper-right vertices.

```
<Feature fid="142" featureType="building"
                   Description="A lecture building">
  <Property Name="NumFloors" type="Integer" value="3"/>
  <Property Name="NumStudents" type="Integer" value="987"/>
  <Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
    <coord>
      <X> 0.0 </X>
      <Y> 0.0 </Y>
    </coord>
    <coord>
      <X> 100.0 </X>
      <Y> 100.0 </Y>
    </coord>
  </Box>
</Feature>
```

The current version of GML uses a FeatureCollection as the basis of its document. A FeatureCollection is a collection of GML Features combined with an Envelope (which bounds the set of Features), a collection of Properties that apply to the `FeatureCollection` and an optional list of *Spatial Reference System Definitions*.

**Example 1.3.5** Figure 1.4 is an example of a feature collection, which contains a `Point` and a `Box`.

```
<FeatureCollection xmlns:ogcgml="http://www.opengis.org/gml#" >
  <Feature>
    <Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      <coord>
        <X> 5.0  </X>
        <Y> 40.0 </Y>
      </coord>
    </Point>
  </Feature>
  <Feature>
    <Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      <coord>
        <X> 0.0 </X>
        <Y> 0.0 </Y>
      </coord>
      <coord>
        <X> 100.0 </X>
        <Y> 100.0 </Y>
      </coord>
    </Box>
  </Feature>
</FeatureCollection>
```

Figure 1.4: An example of feature collections

### 1.3.2 XML Document as Database

Even though XML is mainly regarded as a data exchange and transportation model on the Internet, the large volume of XML documents online also motivates the idea of using XML

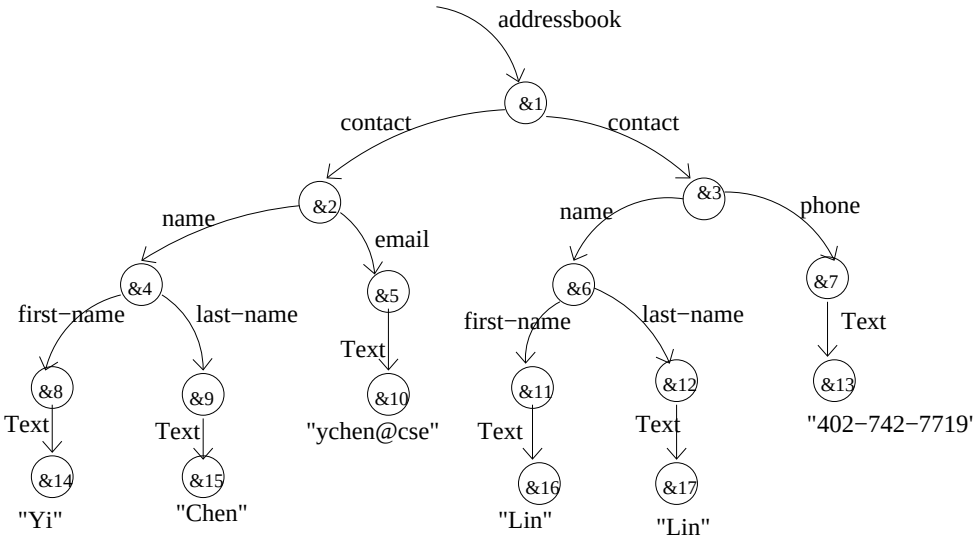



Figure 1.5: The LOREL model for an XML document

as a data storage model [25, 36]. Hence, it is important to find efficient technologies for XML query processing. Traditional database query languages, such as *SQL*, are not appropriate for querying XML, because XML is a semistructured data model without a rigid structure. Novel languages for querying XML documents have been proposed [2, 32, 27, 10, 52].

**Example 1.3.6** Figure 1.5 shows the LOREL graph model which represents the `addressbook.xml` in Example 1.3.1.

### 1.3.3 XML-based Information Integration

Information integration systems take data that is stored in two or more databases (*information sources*) and build from them one large database, possibly virtual, containing information from all the sources, so that data can be queried as a unit. The sources may be conventional databases or other types of information [28].

There are several ways that databases or other distributed information sources can be made to work together. Typical solutions include *federated databases*, *warehousing*, and *mediation*.

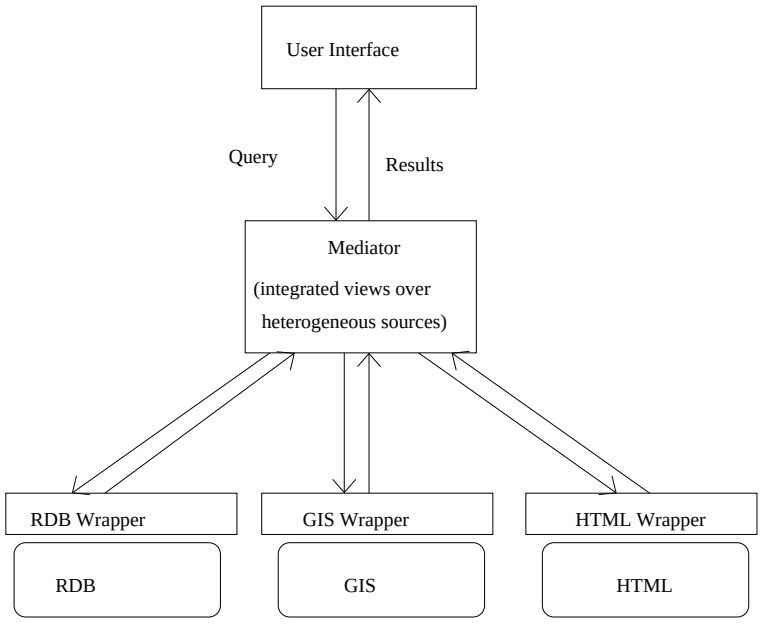



Figure 1.6: A typical mediator system architecture

*Mediator* systems [61] are systems that integrate multiple heterogeneous data sources, providing an integrated global view of the data and providing query facilities on the global view. Figure 1.6 shows the typical architecture of a mediator system. The *wrapper* is the software component developed for each data source, which provides a view of the local data in the global schema. Wrappers also translate queries on the global schema into queries into queries on the local schema, and translate results back into the global schema.

## 1.4 Overview of Research Contributions

The main contribution of this dissertation are the following.

1. Chapter 2, which is based on [50], defines the *Count* aggregation operation, illustrates its applications, and describes two novel solutions. In particular:

   (a) Section 2.2.1 develops the *Partition Aggregation Tree* or *PA Tree* data structure, which can answer *Count* aggregation queries over a set of moving objects that

move according to a linear function in $O(\sqrt{N})$ time (measured in number of disk access operations) and $O(N)$ space.

(b) Section 2.2.2 develops another novel data structure called *Dominance-Time Graph* or *DT graph*, which also can answer *Count* aggregation queries in $O(log\ N)$ time and $N^2$ space. DT graphs allow moving objects to move according to a polynomial function of time.

2. Chapter 3, which is based on [50], defines the *Max-Count* aggregation operation, which also has a wide range of applications, and describes a novel solution for it. For this problem, we introduce the *Dome Subdivision* data structure, which can answer *Max-Count* aggregation queries with $O(log\ N)$ time and $O(N^2)$ space when the objects move linearly along the $x$-axis.

3. Chapter 4, which is based on [17], introduces the use histograms for the estimation of *Max-Count* aggregation queries over moving objects. The resulting algorithm can work in any fixed constant time and space, which is a constant chosen by the user. In general, the larger the chosen constant the more accurate the estimation will be. The results of an extensive set of computer experiments evaluate the impact of various parameters on the estimation accuracy.

4. Chapter 5, which is based on [16, 15, 18], explains the limitations of current extensions of XML documents in representing spatiotemporal data. This chapter also describes the limitations of current *query languages* for XML documents in expressing some high-level queries. This chapter also describes the design of a constraint database-based spatiotemporal database mediator system that allows querying heterogeneous spatiotemporal data sources. In particular:

   (a) Section 5.1 designs a system architecture for querying heterogeneous spatiotemporal documents.

   (b) Section 5.2 describes our design of the underlying data model used for DataFoX.

Table 1.2: Computational complexity of aggregation on moving objects.

| **Query** | **Time** | **Space** | **Dimension** | **Function** | **Method** |
|---|---|---|---|---|---|
| Count | $O(\sqrt{N})$ | $O(N)$ | d | linear | PA tree |
| Count | $O(log\ N)$ | $O(N^2)$ | d | polynomial | DT graph |
| Max | $O(log\ N)$ | $O(N^2)$ | 1 | linear | Dome subdiv |
| Max | $O(c)$ | $O(c)$ | 1 | linear | Histogram |

(c) Section 5.3 introduces the *Layer Algebra* and illustrates each of the operators in the *Layer Algebra*.

(d) Section 5.4 introduces the DataFoX query language and explains the evaluation of DataFoX queries, based on a translation into the *Layer Algebra* of Section 5.3.

(e) Section 5.5 illustrates several detailed examples for the DataFoX system, which was implemented on the top of the MLPQ/PReSTO constraint database system [51, 43, 49, 48].

The complexity of all the aggregation query algorithms using the various novel data structures is summarized in Table 1.2, where $c$ is a constant.

# Chapter 2

# Efficient Spatiotemporal Aggregation: Count

## 2.1 Basic Concepts

*Duality* [22] allows mapping $k$-dimensional moving points into $2k$-dimensional static points. *Partition Trees* proposed by Agarwal et al. [5] are search trees for moving points.

### 2.1.1 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. This line can be represented 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 2.1.1** 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 $P'(b_P, a_P)$ and $Q'(b_Q, a_Q)$ be their corresponding points in the dual

plane. Suppose $P$ overtakes $Q$ or vice versa at time instance $t$, then

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

Let $Slope(P'Q')$ denote the slope of the line $P'Q'$. Then we have $t = -\frac{1}{Slope(P'Q')}$.

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 $-1/t$.

**Definition 2.1.1** 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 $CountBelow(l) = r$.

Note that Definition 2.1.1 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 2.1.1** Figure 2.1 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 $CountBelow(l_2) = 5$.

### 2.1.2 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.

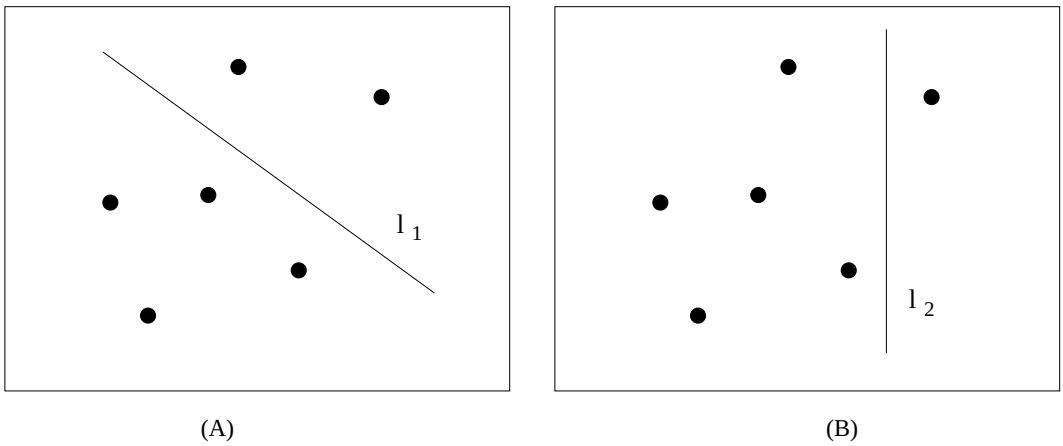



Figure 2.1: Rank of a line.

Figure 2.2(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 2.1.1 (Matousek [41])** 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 $O(N log\ r)$ time.

Using Theorem 2.1.1, it is possible to recursively partition a set of points in the plane. This gives a partition tree.

## 2.2 Count Aggregation Queries

Section 2.2.1 explains an extension of the partition tree. With the modification, the *count* query can be answered in $O(\sqrt{N})$ time. Then a more novel data structure, called a *dominance-time graph*, is described. This data structure can answer the *count* query in logarithmic time.

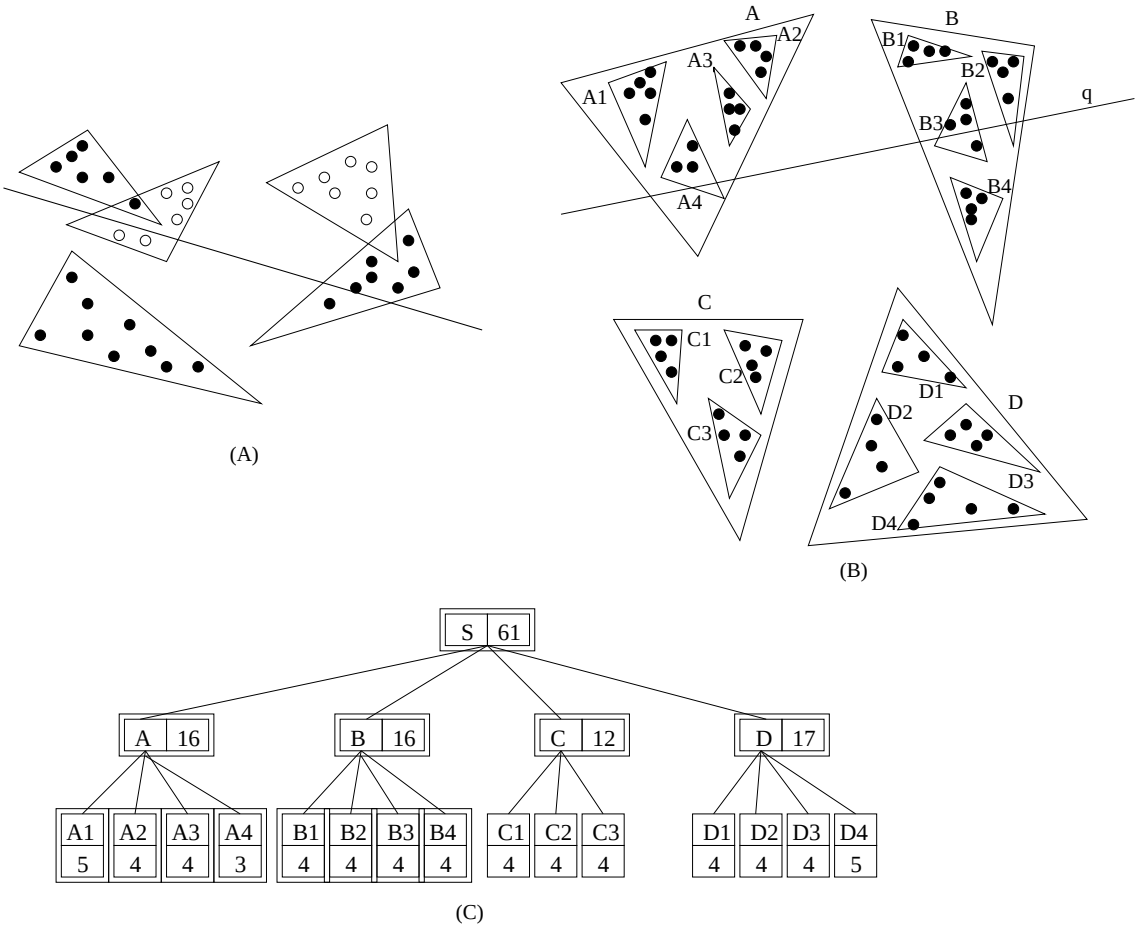



Figure 2.2: A partition aggregation tree.

### 2.2.1 Partition Aggregation Trees

**Definition 2.2.1** 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 $\Delta_i$. A new value $A_i$ is attached to node $v_i$, such that $A_i$ is the number of points in $S_i$. The new tree structure is called the *Partition Aggregation Tree* (PA Tree).

**Theorem 2.2.1** *PA Tree* is a linear size data structure that answers the count query in $O(\sqrt{N})$ time.

**Proof** First, consider the case of one dimensional moving points, that is, when the dual is the plane. To construct the data structure that answers the aggregation operation,

first build a partition tree $T$. Following [5], choose the degree of the node $v$ as $r_v = min\{cB, 2N_v\}$, where $B$ is the size of a disk block, $c$ is a constant integer, and $N_v$ is the number of points represented by the node $v$. Then, with a bottom-up fashion, the aggregate value of each triangle in the partition tree is recursively computed and attached to the node $v$.

To answer the aggregation problem by the query line $l$, visit $T$ in a top down fashion. Suppose a node $v$ in the tree is reached. If $v$ is an internal node associated with triangle $\Delta_i$, test if $\Delta_i$ is below the query line $l$. If so, add $A_i$ to the final result. If $\Delta_i$ is above $l$, just ignore it. If $\Delta_i$ intersects with $l$, then traverse recursively to the subtree of $v$.

Following [5], the query can be answered with $O(\sqrt{N})$ time and the data structure requires $O(N)$ space. The generalization to higher dimensional case follows the generalization of partition trees in [5].

**Example 2.2.1** Figure 2.2(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 2.2(C) shows the structure of the PA-tree. For simplicity, this figure only shows for each node the triangle name and the count of the points contained in that triangle.

To find $CountBelow(q)$, start from the root of the PA-tree, load all top level triangles into memory and compare them to the query line $q$. Because both triangles $C$ and $D$ are below the line, add the precomputed value to the result $CountBelow(q) = 12 + 17 = 29$. For the triangles $A$ and $B$, traverse their children recursively. In this case, triangle $B4$ is below $q$, then $CountBelow(q) = CountBelow(q) + CountIn(B4) = 29 + 4 = 33$, where $CountIn(B4)$ is the number of points in the subset associated with $B4$. When the leaf nodes of the PA-tree are reached, 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 2.2(C), those nodes that are accessed by this algorithm are indicated using double-sided rectangles.

**Example 2.2.2** Figure 2.3 shows three cars driving along three pathes. 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 2.3. Given a time instance, find out how many cars will be covered in the picture at that time.

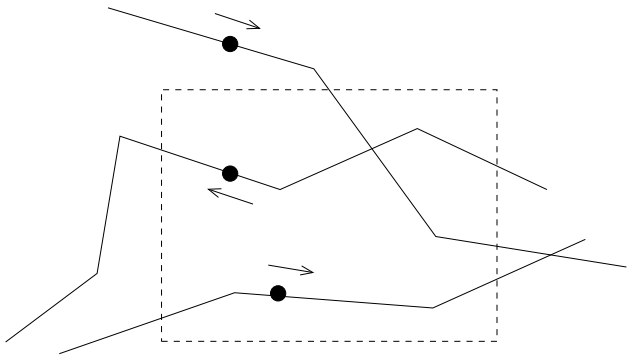


Figure 2.3: Aggregations on piecewise linearly moving points

In Example 2.2.2, the movement of a car can be represented by piecewise linear functions. When the direction or speed changes, it is like replacing a car 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 2.2.2** 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 $O(\sqrt{N})$ time with $O(N)$ space.

This theorem addresses the case in one-dimensional space, such as the situation 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 make U-turns. It is an open problem to find a similarly efficient solution for two or higher dimensional space.

### 2.2.2 Dominance-Time Graph

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

**Definition 2.2.2** For two $k$-dimensional moving points $P = (f_1, ..., f_k)$ and $Q = (g_1, ..., g_k)$, $P$ 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)**.

**Example 2.2.3** Figure 2.4 shows the positions of two moving points $P$ and $Q$ in the two dimensional space at time $t_1$ and $t_2$, $t_1 < t_2$. According to the above definitions, both $ndom(P, Q, t_1)$ and $dom(P, Q, t_2)$ are true.

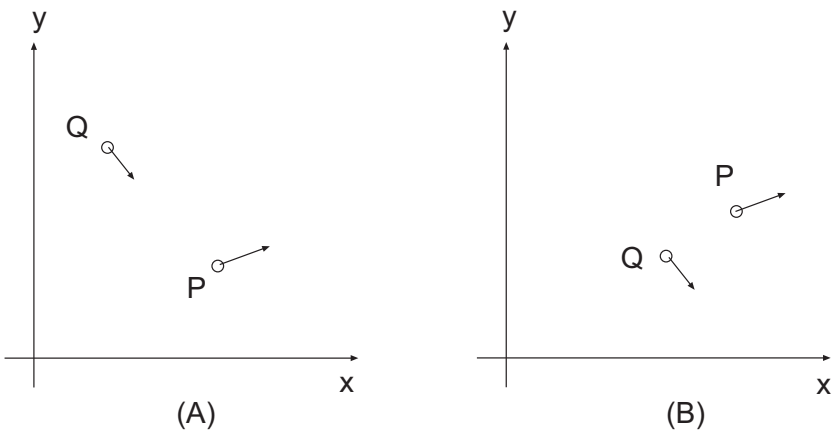



Figure 2.4: the positions of $P$ and $Q$ at time $t_1$ and $t_2$

**Definition 2.2.3** 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 $+\infty$ can be an interval endpoint.

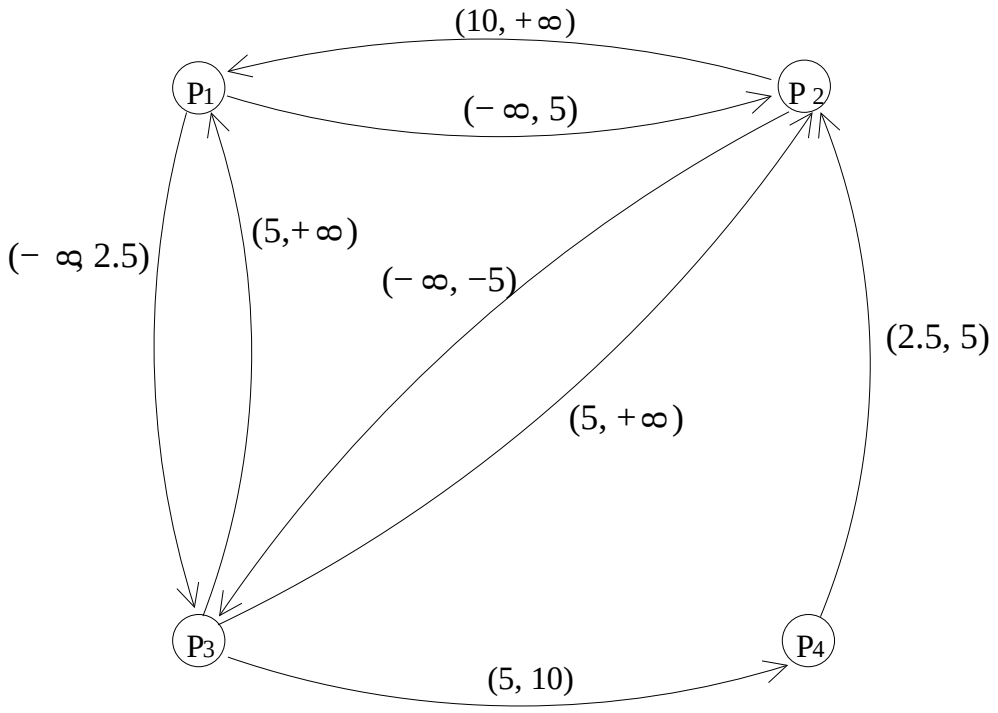



Figure 2.5: A dominance-time graph.

**Example 2.2.4** 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 2.2.4** Let $P$ and $Q$ be two moving points and $t_0$ and t be two time instances such that $t_0 < t$. 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, 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 2.2.5** Let **Rank(P,t)** be the number of points that are dominated by $P$ at time $t$.

**Lemma 2.2.1** 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 2.2.2** 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:

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

**Proof** There are $m$ increment events and $n$ decrement events that happen between $t_0$ and $t$. Each of the increment events increases by one the number of points dominated by $P$. Similarly, each of the decrement events decreases by one the number of points dominated by $P$. Therefore, the formula must be true.

**Example 2.2.5** Table 2.1 shows the rank of each point of Example 2.2.4 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 2.2.2, we have

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

Table 2.1: 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 |

### 2.2.3 Time and Space Analysis

This section describes the basic structure of *dominance-time trees* and show how to use them to answer *count* aggregation queries in $O(log\, mN)$ time, 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.

**Lemma 2.2.3** 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 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.

**Proof** In the dominance-time graph of $S$, for each time interval $(t_i, t_j)$ that contained in labels of outgoing edges from $P$, $t_i$ and $t_j$ indicate the time instances of an increment event and a decrement event for $P$. The set of time instances $t_i$ can be obtained by the following steps:

First, order all ending points of these intervals incrementally and denote them as $t_1, ...,$ $t_i, t_{i+1}, ..., t_j$. If there exists two consecutive instances $t_i = t_{i+1}$, then delete $t_{i+1}$. If there exist two consecutive intervals $(t_i, t_{i+1}), (t_{i+1}, t_{i+2})$ in which the ranks of P are the same, then delete $t_{i+1}$.

**Example 2.2.6** 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 2.6 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 2.2.6** 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. The precomputed rank of $P$ during the interval $(t_{i-1}, t_i)$ is associated with each field $v_i$ for $1 \leq i \leq b$. Given a time instance $t$, the rank of $P$ can be found by searching the dominance-time tree until the leaf node with the interval that contains $t$ is reached.

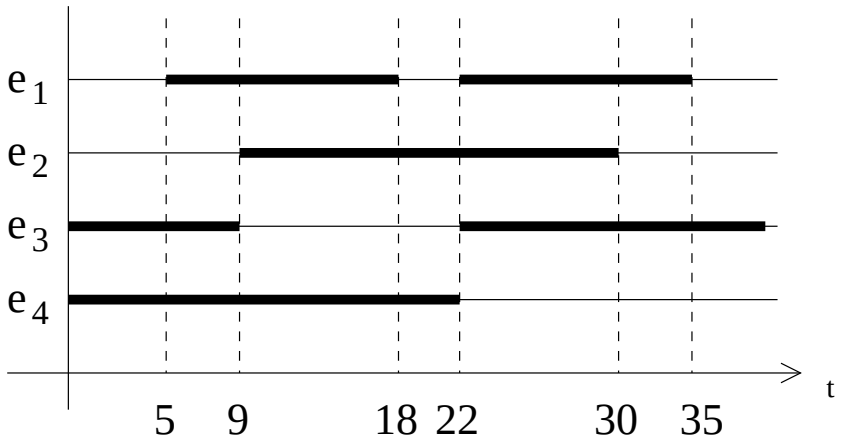



Figure 2.6: A time line.

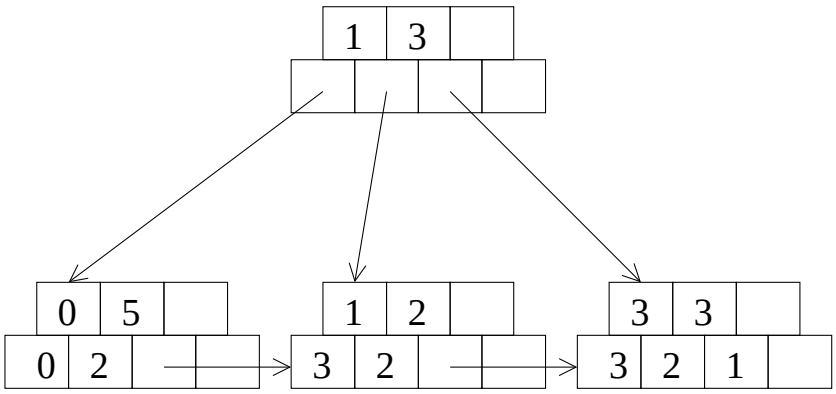



Figure 2.7: A dominance-time tree.

**Example 2.2.7** Figure 2.7 shows the rank tree $T_P$ for $P$, whose outgoing edges are described in Example 3. In this case, each node of the tree can contain at most three time instances. To find the rank of $P$ at 20, start from the root of the tree, compare each time instance in the internal node and search the sub-tree recursively, until the leaf node which contains the interval $(18, 22)$ is found. Hence, $Rank(P, 20) = 2$ because the data field in the leaf node associated with the interval contains the precomputed rank 2.

**Theorem 2.2.3** 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 $O(log_B\ N)$ time 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.

**Proof** The total number of time instances for $N$ moving points is $O(mN)$ where $m$ is the maximum degree of the polynomial functions of time. The dominance-time tree is a $B$-tree, which is a linear space data structure. Hence, the dominance-time tree $T_P$ requires a total of $O(mN/B)$ blocks where $B$ is the size of disk blocks.

Now, consider the time for the *count aggregation* problem. For a given point $P$ in a set of $N$ moving points, there are at most $N$ outgoing edges starting from $P$. Because the max degree of the polynomial functions is $m$, there are at most $m$ time intervals for each outgoing edges. Hence, the total number of time intervals for outgoing edges of $P$ is at most $mN$. Each of the $2mN$ ending point of these intervals indicates an increment or decrement event to $P$. Note that for any two consecutive time instances $t_i$ and $t_{i+1}$, the rank of $P$ can be precomputed. These time instances partition the time line into at most $(2mN + 1)$ segments. Let $B$ be the block size, these segments can be organized into a $B$+-tree structure, such that given a time instance $t$, the segment where $t$ is contained can be identified in $O(log_B(mN))$ time.

# Chapter 3

# Efficient Spatiotemporal Aggregation: Max-Count

## 3.1 Max-Count Defined

Many aggregation problems can be answered by the *query-and-aggregate* method, given an efficient index structure that can answer the query efficiently. For example, a naive way to answer the aggregation query: *"Find the number of points that will be taken over by q"* is: (i) use spatiotemporal index data structure to find out what are those points that will be taken over by $q$; (ii) count the result set of the above query. Observe that the aggregation performance depends on the size of the query result.

This chapter introduces the *max-count* spatio-temporal aggregation, which has not been studied before, and shows that it cannot be answered by the query-and-aggregate method. An example of max-count query is: *"A group of enemy tanks are moving with fixed speed and direction. When and where should the bomber plane drop a bomb so that it can hit the max number of tanks?"*

Observe that this query is different from traditional spatio-temporal *window queries*, that the query time could be open ended. We define the max-count query as the following:

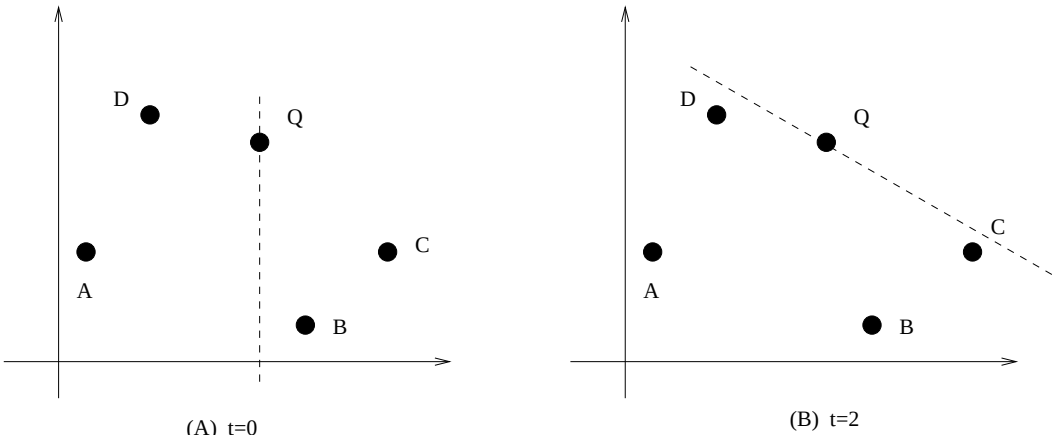



Figure 3.1: Max-Count for moving points in one dimensional space

**Definition 3.1.1** (Max-Count)

Given a collection $S$ of moving points and a moving query point $q$, compute MAX{COUNT($p$) | $p \in S$ and $p$ is dominated by $q$}.

## 3.2 Max-Count in Dual Plane

According to Lemma 2.1.1, given a set of moving points in one dimensional space, a moving query point $Q$ and 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 $-1/t$. Similarly, the problem of finding the max number of points dominated by $Q$ reduces to the problem of finding what is the max number of points that are below $l$ at any given time.

**Example 3.2.1** Figure 3.1 shows a set of four moving points in one dimensional space, which are represented by static points in a two dimensional space, and a query point $Q$. At time $t = 0$, $A$ and $D$ are dominated by $Q$. At time $t = 2$, all four points are dominated by $Q$.

There is always a line crossing $Q$, such that all points in the set are below the line, if $Q$ lies above the convex hull of the point set, to the left of all points, or to the right of all

points.

**Example 3.2.2** Figure 3.2 (A) shows the area described in the above as shaded area. Observe that this area is the union of four areas, including the area above line $AD$ as shown in Figure 3.2(B), the area above line $CD$ as shown in Figure 3.2(C), the area left to the vertical line crossing $A$ as shown in Figure 3.2(D), and the area right to the vertical line crossing $C$ as shown in Figure 3.2(E). Obviously, if $Q$ lies in the shaded area in Figure 3.2(B), there is always a line crossing $Q$, which is parallel to the line $AD$, such that all points are below this line. Similarly, this line always exists if $Q$ lies in the shaded area. Besides, if $Q$ lies below the shaded area, it is impossible to find a line crossing $Q$ such that all four points will be below that line. That is, $Max\_Count(Q) = 4$. Observe that these areas are layered.

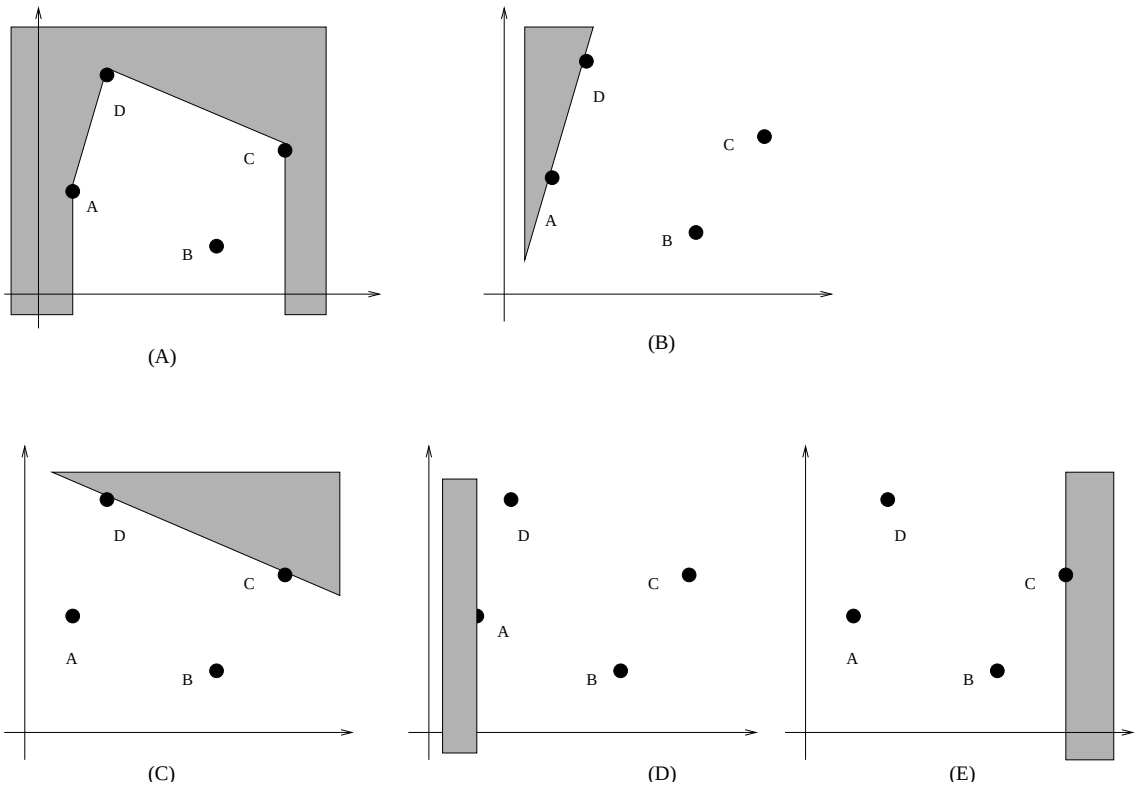



Figure 3.2: If $Q$ lies above the convex hull of the point set with 4 points, then $Max - Count(Q) = 4$

Similarly, if $Q$ lies above the line $AC$, there is always a line $l$ crossing $Q$, such that there are at least three points below $l$. Because $Q$ is above $AC$, there exists a line $l$ crossing $Q$ that

parallels to $AC$, such that all points below $AC$ and the points **on** $AC$ will all lie below $l$. So, if $Q$ lies in the shaded area shown in Figure 3.3(A), then $Max\_Count(Q) \geq 3$. Combine this area with the shaded area shown in Example 3.2.2. If $Q$ lies in the light-shaded area as shown in Figure 3.3(B), then $Max\_Count(Q) = 3$.

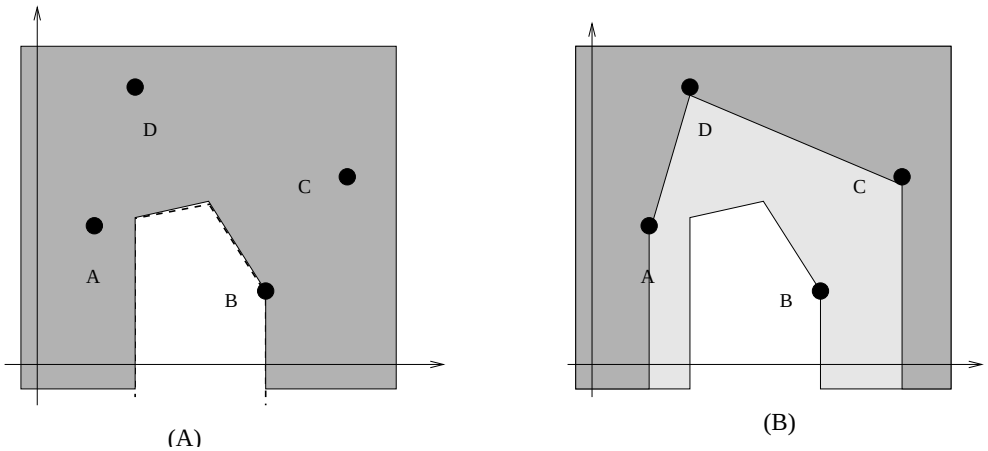



Figure 3.3: If $Q$ lies in the shaded area in (A), then $Max - Count(Q) \geq 3$. If $Q$ lies in the light-shaded area in (B), then $Max - Count(Q) = 3$

## 3.3 Dome and Layer

Our *max-count* aggregation algorithm uses a novel data structure built on the concept of *domes*, which is introduced here as a new type of spatial partition of the dual plane of a set of one-dimensional moving points.

**Definition 3.3.1** Let $S$ be any set of points in the plane. For any new point $Q$, define **MaxBelow(Q)** to be the maximum number of points below any line that passes through $Q$.

**Example 3.3.1** Example 3.4 shows a set of four points $A$,$B$,$C$,$D$, and a point $Q$ in the two dimensional space. In this example, $MaxBelow(Q) = 3$, because for any line crossing $Q$, there is at least one point that is above this line.

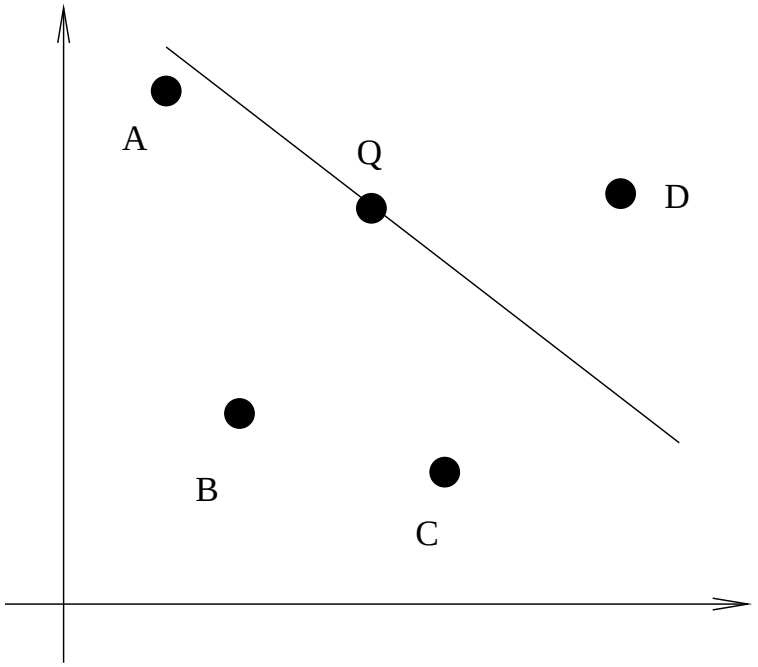



Figure 3.4: MaxBelow(Q) = 3

**Definition 3.3.2** 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 $l$.

**Example 3.3.2** Figure 3.5 shows a set of four points $A$,$B$,$C$, $D$. $L_3$=$\{l_1, l_2, l_3\}$. $l_4$ doesn't belong to $L_3$ because $CountBelow(l_4) + CountOn(l_4) = 2$.

**Definition 3.3.3** 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.

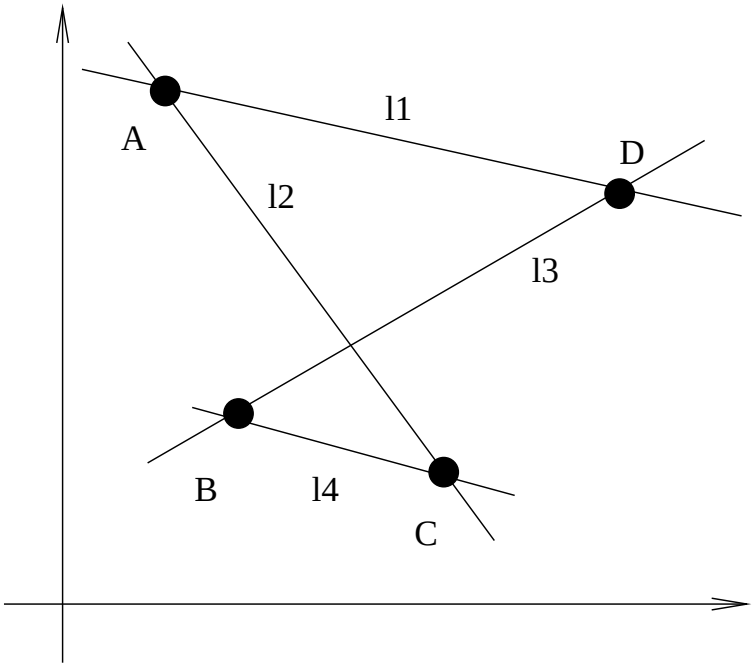



Figure 3.5: The line set $L_3$

**Definition 3.3.4 Layer(k)**= $\{Q|Q \in Below(L_{k+1})$ and $Q \notin Below(L_k)\}$.

**Example 3.3.3** In Figure 3.3(B), $Layer(3)$ is the light-shaded area. The upper boundary of the light-shaded area is $d_4$, the lower boundary of the area is $d_3$.

**Example 3.3.4** Figure 3.6 shows 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.

**Lemma 3.3.1** For any $i$ and $j$ such that $i \leq j$, the following holds.

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

**Lemma 3.3.3** Each point belongs to only one layer.

**Proof** Suppose that $Q$ belongs to both Layer(i) and Layer(j) where $i < j$. By Definition 3.3.4, $Q$ must be between $d_i$ and $d_{i+1}$ and also between $d_j$ and $d_{j+1}$. That means that $Q \in Below(L_{i+1})$ and $Q \notin Below(L_j)$. By Part (2) of Lemma 3.3.1, it must be the case that $Below(L_{i+1}) \subseteq Below(L_j)$ because $i + 1 \leq j$. However, there is no $Q$ that satisfies all of the above conditions. This is a contradiction, showing that the lemma holds.

**Theorem 3.3.1** $Q \in Layer(m) \leftrightarrow MaxBelow(Q) = m$.

**Proof** It is easy to see that region $Below(k)$ contains at most $k$ points. Because these points are a subset of $S$, $k \leq N$ must hold. In the worst case, these points will be all on the boundary of $Below(k)$, that is, on dome $d_k$.

Theorem 3.3.1 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. A data structure can be built based on this theorm for efficiently identifying which element of the partition a new point is located in, using the following well-known result from computational geometry.

**Theorem 3.3.2** [45] In an N-vertex planar subdivision the point location problem can be solved in $O(log\ N)$ time using $O(N)$ storage, given $O(Nlog\ N)$ preprocessing time.

**Lemma 3.3.4** Any dome $d_k$ has $O(N)$ edges.

**Lemma 3.3.5** Let $S$ be any set of $N$ points in the plane and $Q$ a query point. Then, $MaxBelow(Q) = m$ can be found in $O(\log N)$ time using an $O(N^2)$ space data structure.

**Proof** By Lemma 3.3.4, each dome has $O(N)$ edges. There are only $N/2$ different domes, namely $d_N, d_{N-1}, \ldots, d_{N/2}$. Therefore, the total number of edges and vertices on all of these domes is $O(N^2)$. First, draw all of the domes and find the partition of the plane according to layers. Using this partitioning and Theorem 3.3.2, the layer in which $Q$ lies can be found in $O(log\ N)$ time. Then, by Theorem 3.3.1, $MaxBelow(Q)$ can be calculated.

**Lemma 3.3.6** 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 holds.

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 $MaxBelow(Q')$.

**Theorem 3.3.3** 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.

**Proof** Let $Q'$ be the dual of the query point. By Lemma 3.3.6, the max-count aggregation problem can be answered by finding $MaxBelow(Q')$. By Lemma 3.3.5, $MaxBelow(Q')$ can be found in the required time and space.

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, the definition of layers can be modified correspondingly. Then, Theorems 3.3.1 and 3.3.3 still hold.

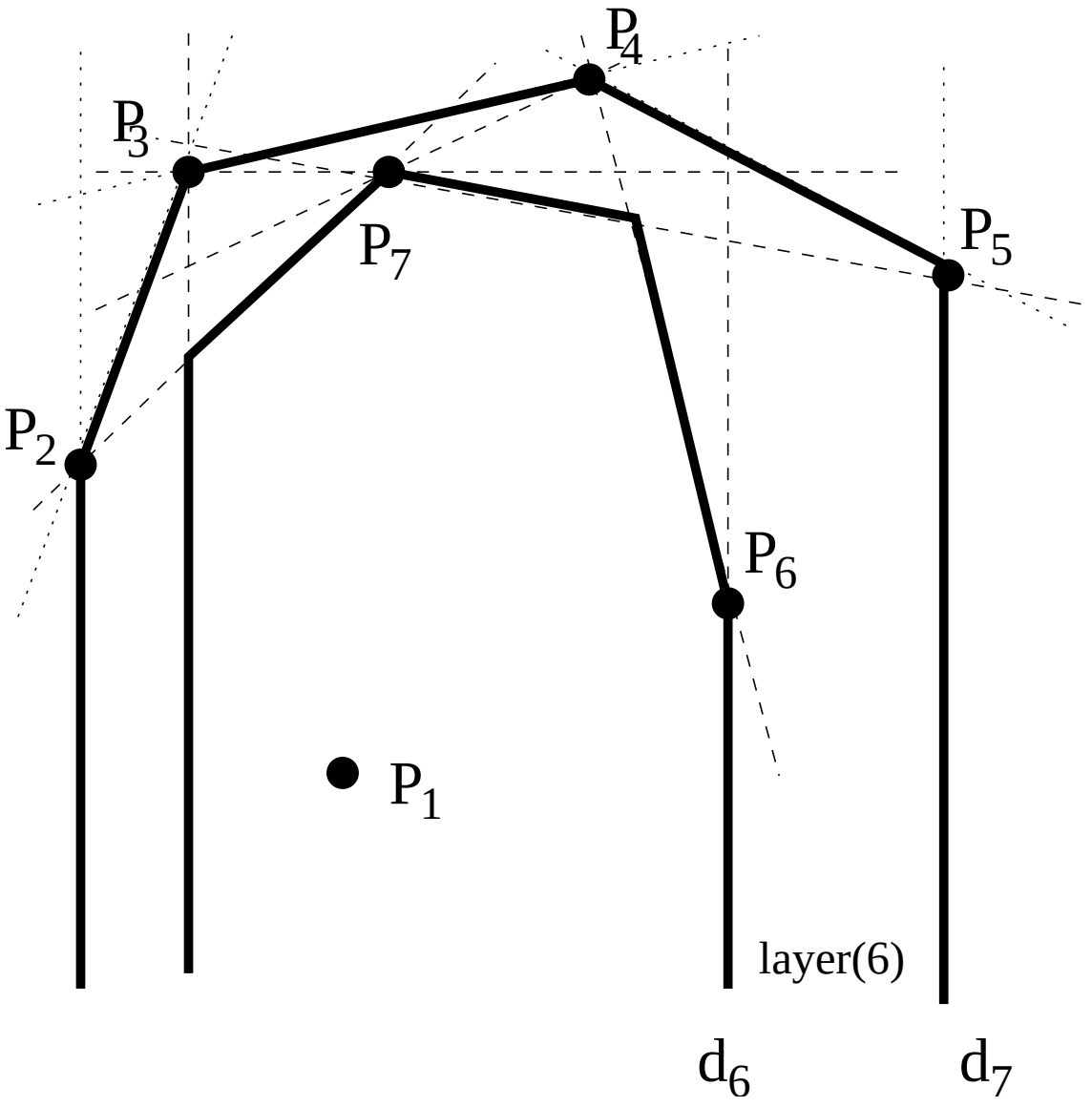



Figure 3.6: *Layer*(6) for seven points.

# Chapter 4

# Efficient Aggregation: Approximation

Many spatio-temporal aggregate operations can be evaluated in logarithmic time. For example, the *Count* aggregation 2 and *Max-Count* 3 can both be answered in $O(log\ N)$ time for a set of $N$ moving points.

However, in many time critical applications even logarithmic query time may be too slow. Accuracy can be sacrificed to obtain more efficient performance. Several authors considered recently the estimation of aggregation operators on spatial data.

## 4.1 Max-Count Aggregation Estimation

The special case is studied when the set of moving points in one dimensional space has uniform distribution of initial position (at time $t = 0$) and velocity.

Let $S$ be a set of $N$ moving points in one dimensional space. The position of a point $P_i \in S$ at time $t$ can be represented by a linear function of time $P_i(t) = a_i t + b_i$. In the dual plane, this point can be represented as a static point with the coordinate $(a_i, b_i)$. Suppose that the $N$ points represented in the dual plane are distributed uniformly in a rectangular

area $R$ as shown in Figure 4.1. The following lemma is based on the histogram method.

**Definition 4.1.1** The spatio-temporal *histogram* consists of a partitioning into a set of rectangular areas, called *buckets*, the two dimensional dual space of the one-dimensional moving point set. Each bucket is described by its corner vertices and the total number of points in it.

**Lemma 4.1.1** Let $S$ be a set of $N$ moving points which are all mapped within a rectangular area $R$ in the dual plane, and let $Q_1(t)$ and $Q_2(t)$ be two moving query points. Then, the number of points in $S$ that lie between $Q_1(t)$ and $Q_2(t)$ at time $t$ are the points that lie in the intersection of the rectangular area $R$ and the *query band area* defined as the area between the two parallel lines $l_1$ and $l_2$, which cross the dual points of $Q_1(t)$ and $Q_2(t)$ respectively and have slopes $-1/t$. Suppose the area of the intersection is $A$, then the number of points that lie between $Q_1(t)$ and $Q_2(t)$ at time $t$ can be estimated to be $N \cdot A/R$ assuming a uniform distribution of the points in $R$.

**Proof** The proof follows from the fact each moving point of $S$ and the query points are mapped into points in the dual plane such that the query band at time $t$ contains those points which are between $l_1$, which is $y-d_1 = \frac{-1}{t}(x-v_1)$ and $l_2$, which is $y-d_2 = \frac{-1}{t}(x-v_2)$, where $v_1$ and $v_2$ are the velocities and $d_1$ and $d_2$ are the initial positions of $Q_1(t)$ and $Q_2(t)$.

That is, if the area of the intersection can be calculated, the estimated aggregation result can be efficiently calculated. If $l_1$ is below $l_2$, the area of the intersection $A$ can be represented as $A = A_2 - A_1$, where $A_1$ is the *area* in the rectangle that is below $l_1$, and $A_2$ is the area in the rectangle that is below $l_2$. On the contrary, if $l_1$ is above $l_2$, then $A = A_1 - A_2$.

It is also clear that $A_1$ and $A_2$ can be calculated in a constant time. For example, given a time instance $t$, (i) if $l_1$ is above the rectangular area, then $A_1 = R$; (ii) if $l_1$ is below the rectangular area, then $A_1 = 0$; (ii) if $l_1$ intersects the rectangular area, we have the following cases as shown in Figure 4.2:

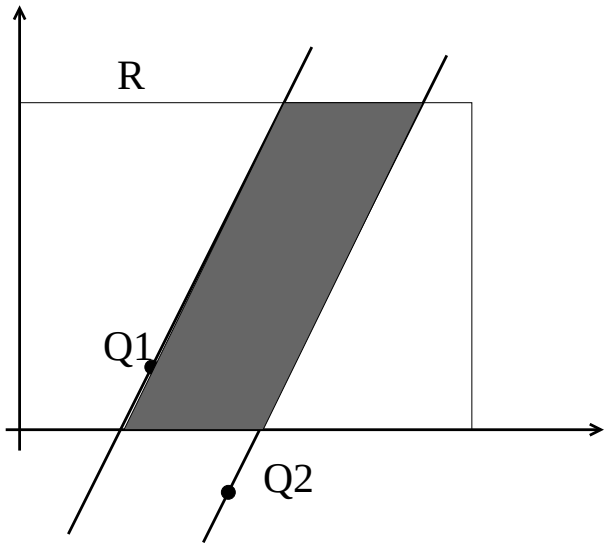



Figure 4.1: Estimation idea assuming uniformly distributed point sets.

1. Only the upper-right vertex is above $l_1$.

2. Only the upper-left vertex is above $l_1$.

3. Both the upper-left and upper-right vertexes are above $l_1$.

4. Both the upper-left and lower-left vertexes are above $l_1$.

5. Both the upper-right and lower-right vertexes are above $l_1$.

6. Only the lower-left vertex is below $l_1$.

7. Only the lower-right vertex is below $l_1$.

The above lemma shows that a constant number of calculations are needed to find the *count* aggregate. When we pose *Max-Count* aggregates on spatiotemporal points, we are given a query time range $(t^{\lfloor}, t^{\rfloor})$. The following shows that only a constant number of calculations are needed to find *Max-Count*, given a query time range.

**Lemma 4.1.2** Let $S$ be a set of moving points in one dimensional space, and they are uniformly distributed in a rectangular area $R$ in the dual plane. Let $Q_1$ and $Q_2$ be two moving

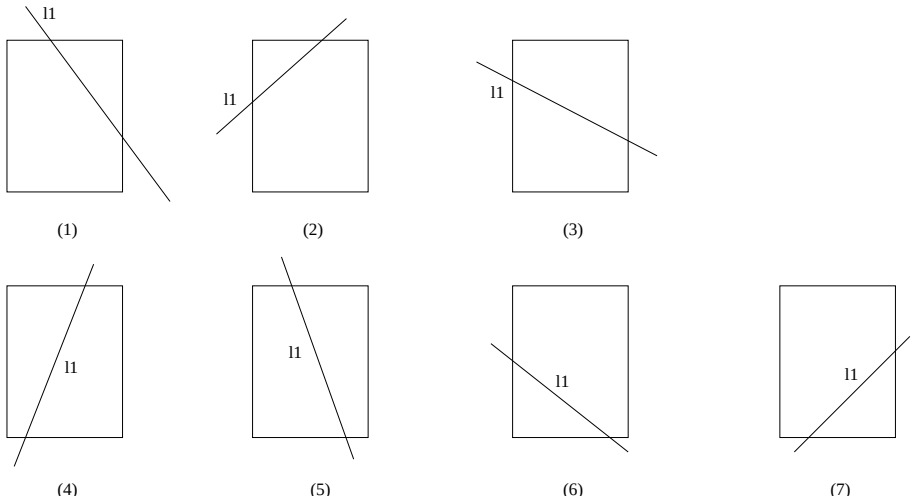



Figure 4.2: Cases with one bucket and one line.

points. Given a query time range $(t_1, t_2)$, the *Max-Count* aggregation can be computed by a constant number of calculations.

**Proof** Let $(t'_1, t'_2)$ be a time interval, such that $t_1 \leq t'_1 \leq t'_2 \leq t_2$, and during this time range, the two lines $l_1$ and $l_2$ do not sweep through any of the corner vertices of the rectangle. Then, the *Max-Count* can be computed during this time range $(t'_1, t'_2)$. This is because when none of the corner vertices is swept through by $l_1$ or $l_2$, then the relationship between $l_1$, $l_2$ with the rectangle remains the same. For example, if $l_1$ crosses the rectangle at time $t'_1$, it might be one of the seven cases described above, and it will remain the same case during the whole time range from $t'_1$ to $t'_2$. This is because if $l_1$ doesn't sweep through any corner vertex, then the area below $l_1$ can always be represented by a function of time $A_1 = at + \frac{b}{t} + c$, where $a$, $b$ and $c$ are constants. Hence, neither $l_1$ nor $l_2$ sweeps through any corner vertex, then the intersection area $A$ can also be represented by a function of time of the form $A = at + \frac{b}{t} + c$. Obviously, given the time range, the maximum value of this function can be computed. There are only four vertices for a rectangle, hence there are at most eight time intervals during which $l_1$ and $l_2$ do not sweep through any corner vertex. That is, in the worst case, the maximum value of eight functions need to be calculated to find the estimated *Max-Count* value.

**Lemma 4.1.3** Let $R$ be a rectangle, such that the lower-left and upper-right corner points are represented as $(x_1, y_1)$ and $(x_2, y_2)$. Let $l$ be a line that crosses a point $(x_0, y_0)$. Then, the area in $R$ that is below or above $l$ can always be represented by a function of the form $A = a \cdot t + \frac{b}{t} + c$, where $a$, $b$ and $c$ are constants.

**Proof** Without loss of generality, first consider two cases. In the first case, $l$ intersects two perpendicular edges of $R$, and, in the second case, $l$ crosses two parallel edges of $R$.

**Case (1):** In this case, the area below $l$ is a triangle as shown in Figure 4.3(A). Suppose the lower-left and upper-right corner points of $R$ are $(x_1, y_1)$ and $(x_2, y_2)$, and $l$ crosses a point $(x_0, y_0)$, then $l$ can be represented by

$$\frac{y - y_0}{x - x_0} = -\frac{1}{t},$$

when $x \neq x_0$. The above can be rewritten in terms of $x$ and then $y$ as:

$$x = x_0 - t(y - y_0) \qquad y = y_0 - \frac{(x - x_0)}{t}.$$

Now, the triangular area which is the intersection of $R$ and the area below $l$ can be represented as follows.

$$
\begin{aligned}
Area(R \cap Below(l)) &= \frac{1}{2} \cdot (x_3 - x_1)(y_3 - y_1) \\
&= \frac{1}{2} \cdot ((x_0 - t(y_1 - y_0)) - x_1) \cdot ((y_0 - \frac{(x_1 - x_0)}{t}) - y_1) \\
&= \frac{1}{2} \cdot (-t(y_1 - y_0) + x_0 - x_1) \cdot (y_0 - y_1 - \frac{(x_1 - x_0)}{t}) \\
&= \frac{1}{2} \cdot (t(y_0 - y_1) + (x_0 - x_1)) \cdot ((y_0 - y_1) + \frac{(x_0 - x_1)}{t}) \\
&= \frac{1}{2} \cdot (t(y_0 - y_1)^2 + 2(x_0 - x_1)(y_0 - y_1) + \frac{(x_0 - x_1)^2}{t}) \\
&= \frac{t(y_0 - y_1)^2}{2} + (x_0 - x_1)(y_0 - y_1) + \frac{(x_0 - x_1)^2}{2t} \\
&= a \cdot t + \frac{b}{t} + c
\end{aligned}
$$

where $a = \frac{(y_0 - y_1)^2}{2}$, $b = \frac{(x_0 - x_1)^2}{2}$, and $c = (y_0 - y_1)(x_0 - x_1)$. Similarly, the area above $l$ is:

$$
\begin{aligned}
Area(R \cap Above(l)) &= Area(R) - Area(R \cap Below(l)) \\
&= (x_2 - x_1)(y_2 - y_1) - at - \frac{b}{t} - c \\
&= a_1 t + \frac{b_1}{t} + c_1
\end{aligned}
$$

where $a_1 = -a$, $b_1 = -b$ and $c_1 = (x_2 - x_1)(y_2 - y)1) - c$.

**Case (2):** In this case, without loss of generality, consider the situation shown in Figure 4.3(B). Here, the area in $R$ that is below $l$ can be represented as follows.

$$
\begin{aligned}
Area(R \cap Below(l)) &= \frac{1}{2} \cdot (y_2 - y_1)((x_3 - x_1) + (x_4 - x_1)) \\
&= \frac{1}{2} \cdot (y_2 - y_1)(x_0 - t(y_1 - y_0) + x_0 - t(y_2 - y_0) - 2x_1) \\
&= \frac{1}{2} \cdot (y_2 - y_1)(2x_0 - 2x_1 + t(2y_0 - y_1 - y_2)) \\
&= \frac{(y_2 - y_1)(2y_0 - y_1 - y_2)}{2} \cdot t + (y_2 - y_1)(x_0 - x_1) \\
&= a_2 \cdot t + \frac{b_2}{t} + c_2
\end{aligned}
$$

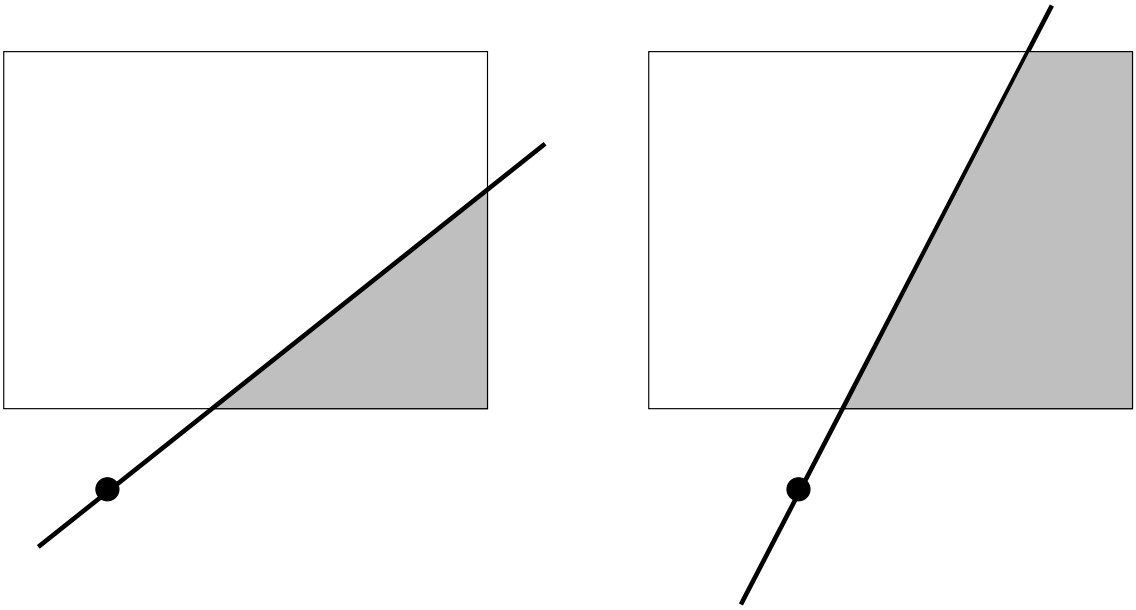


Figure 4.3: The intersection of a bucket and the area below a line.

Now, the following lemma shows that the area in $R$ that are between two parallel lines $l_1$ and $l_2$ can also be represented by a function of time in a similar form.

**Lemma 4.1.4** Let $S$ be a set of moving points in one dimensional space, let $H$ be the histogram of $S$, and let $Q_1(t)$ and $Q_2(t)$ be two moving query points. Then, the estimated number of points that are located between these two points at time $t$ can be represented by

a function of the form

$$count(t) = a \cdot t + \frac{b}{t} + c$$

where $a$, $b$ and $c$ are constants.

**Proof** The query band area is always the sum or difference of the basic areas shown in Figure 4.2. By Lemma 4.1.3 the basic areas are also of the stated form, which is closed under difference and addition.

**Lemma 4.1.5** Suppose the dual plane is partitioned into rectangular buckets. We can calculate the *Max-Count* of a query band during a query time interval when the query band covers the same set of corner points of the buckets.

**Proof** For a function of time in the form $f(t) = a \cdot t + \frac{b}{t} + c$ and a given time interval, the maximum value of $f(t)$ during that time interval can always be calculated in a constant time.

**Definition 4.1.2** Let $H$ be a histogram. Let $Q_1(t)$ and $Q_2(t)$ be two query points. Let $(t^{\lfloor}, t^{\rfloor})$ be the query time interval. Define the *Time Partition Order* to be the set of time instances $TP = \{t_1, t_2, ..., t_i, ..., t_k\}$, such that $t_1 = t^{\lfloor}$ and $t_k = t^{\rfloor}$ and for each time interval $[t_i, t_{i+1})$ the set of bucket corner vertices that lie within the query band remains the same.

The slopes of $l_1$ and $l_2$ change with $t$. For the query band to remain in one of the states shown in Figure 4.2 during a time interval $[t_i, t_{i+1})$, it cannot change so much that it either leaves a corner vertex or adds a new corner vertex of a bucket.

Therefore, throughout the time interval $[t_i, t_{i+1})$, the number of points within the query band can be estimated by the same function of the form $a \cdot t + \frac{b}{t} + c$, where $a$,$b$ and $c$ are constants.

All the above lemmas and observations lead to the following algorithm to estimate the *Max-Count* value.

**Algorithm 1** *Max-Count Algorithm*

**Input:** A histogram $H$, query points $Q_1(t)$ and $Q_2(t)$ and a query time interval $(t^{\lfloor}, t^{\rfloor})$.

**Output:** The approximated *Max-Count* value.

1. Find all bucket corner vertices in $H$. Find the lines between the corner vertices and the dual of the query points. Order the lines by their slopes. Find the Time Partition Order of the time interval $(t^{\lfloor}, t^{\rfloor})$.
2. For each time interval associated with the Time Partition Order calculate the function of time having the form $a \cdot t + \frac{b}{t} + c$, where $a$, $b$ and $c$ are constants.
3. For each such function of time, calculate the maximum value within the corresponding time interval. Store the result in a list.
4. The maximum value in the list is the final result.

**Theorem 4.1.1** Let $H$ be a histogram with $B$ number of buckets. Let $Q_1(t)$ and $Q_2(t)$ be two moving query points, and let $(t^{\lfloor}, t^{\rfloor})$ be a time interval. It takes $O(B \log B)$ time to calculate the estimated *Max-Count* value.

**Proof** Because the histogram has $B$ number of buckets, the total number of corner vertices in the histogram is $O(B)$. To find the time intervals, calculate the slope of the line from each corner vertex to the dual points of $Q_1(t)$ and $Q_2(t)$. Then, order all these lines according to their slopes. This takes $O(B \log B)$ time. Note that the slopes are equivalent to $-1/t$. Hence when ordering the lines by their slopes, the time instance associated with them are also ordered. Note that only the lines that have slopes between $-1/t^{\lfloor}$ and $-1/t^{\rfloor}$ where $t$ is nonnegative are considered.

During any time interval $[t_i, t_{i+1})$ used by Algorithm 1, the set of bucket corner vertices within the query band remains the same. Hence, according to Lemma 4.1.2, the *Max-Count* value during that time interval can be calculated in constant time. Finally, because the number of corner vertices is $O(B)$, the total number of time intervals is $O(B)$.

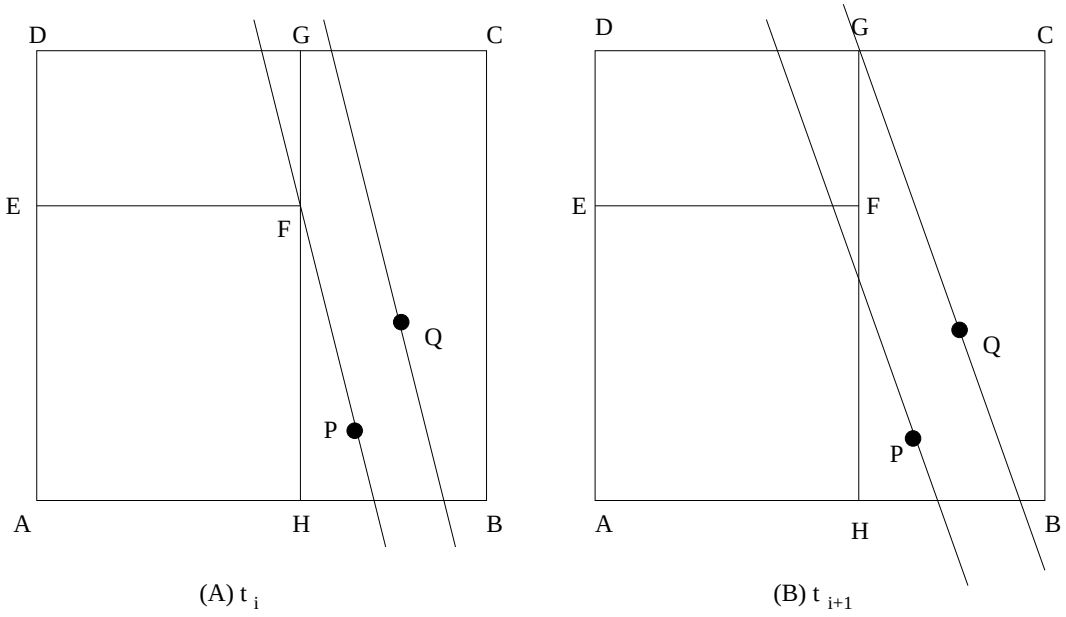



Figure 4.4: The query band at two different times.

**Example 4.1.1** Figure 4.4 shows a histogram which contains three buckets and in which $P$ and $Q$ are the duals of the two moving query points. There are a total of eight corner vertices for the buckets in the histogram, as shown in the figure. Figure 4.4(A) shows the query band at time $t_i$. The query band consists of two parallel lines which have the slope $-1/t_i$. The line crossing $P$ also crosses $F$. This means that at time $t_i$, $G$ lies in the query band, and $F$ enters the query band. Sweep the query band counterclockwise and at a later time $t_{i+1}$, $F$ still lies in the query band, but $G$ is leaving the query band, as shown in Figure 4.4(B).

During the time interval $[t_i, t_{i+1})$, the query band intersects with all three buckets. Moreover, the intersection between each bucket and the query band remains in one of the

states shown in Figure 4.2. For example, for the intersection of the query band and bucket $DEFG$ remains in the case shown in Figure 4.2(5). Hence, the area of the intersection between the query band and bucket $DEFG$ can be represented by the same function of time. According to Lemmas 4.1.1 and 4.1.3, the number of points can be estimated by a function of time $f_1 = a_1 \cdot t + \frac{b_1}{t} + c_1$. Similarly, the number of points in the intersection of the query band and buckets $AHFE$ and $HBCG$ can be estimated by functions $f_2 = a_2 \cdot t + \frac{b_2}{t} + c_2$ and $f_3 = a_3 \cdot t + \frac{b_3}{t} + c_3$. Then, the total number of points during the time interval $[t_i, t_{i+1})$ can be estimated by the function of time $f = f_1 + f_2 + f_3 = (a_1 + a_2 + a_3) \cdot t + \frac{b_1 + b_2 + b_3}{t} + (c_1 + c_2 + c_3)$. Observe that the *Max-Count* value during this time interval can be calculated with constant time. Because the number of such time intervals formed by *Time Partition Order* is $O(B)$, it takes $O(B)$ time to calculate the *Max-Count* of all such intervals and the final result.

## 4.2 Experiments

This section studies the impact of various parameters for the performance of the algorithm. We systematically generate several synthetic datasets that consist of a large number of one-dimensional moving points. Both the initial positions and the speeds of these points are distributed between 0 and $10,000$ according to the Zipf distribution. In the Zipf distribution, there are more points with bigger displacement value for the same speed, and there are more points with higher speed for the same displacement. That is similar to the dataset used in [19, 58]. Therefore, in the dual plane the dataset was distributed within a rectangular area with height $10,000$ and width $10,000$ with a greater density of points in the upper and right regions of the histogram.

### 4.2.1 Experimental Parameters

We consider the estimation accuracy with respect to the following parameters:

**Number of Buckets:** This is the number of rectangles that the plane is divided into using

the histogram algorithm of [4]. In this study, the number of buckets was either 10 or 20.

**Number of Points:** This is the number of points in the histogram. Because the same Zipf distribution in is used all of our experiments, the higher number of points also mean a higher density of the points. We varied the number of points from 8000 to 40000.

**Query Range:** This is the distance between the *duals* of the two moving query points. The query range is changed from 400 to 2000, that is, from 2% to 20% of the width of the histogram.

**Query Type:** The position of the dual of the two moving query points can be either in a dense region or a sparse region of the histogram. One dense and one sparse query were studied in the experiments.

Initially, the query type was not considered as a parameter. However, it is included because the experiments show that it is actually an important parameter. Presenting only an average running time of a set of different queries would actually hide an interesting and non-obvious relationship.

### 4.2.2 Dense Queries

In our first set of experiments, the location of the duals of the moving query points was in a dense region of the histogram. These queries are called *dense queries*.

#### Bucket Size 10

The bucket size was fixed to be 10 and varied the number of points and the query range.

Figure 4.5 shows that the error rate decreases exponentially as the number of points increase. The error rate also decreases slightly as the query range increases.

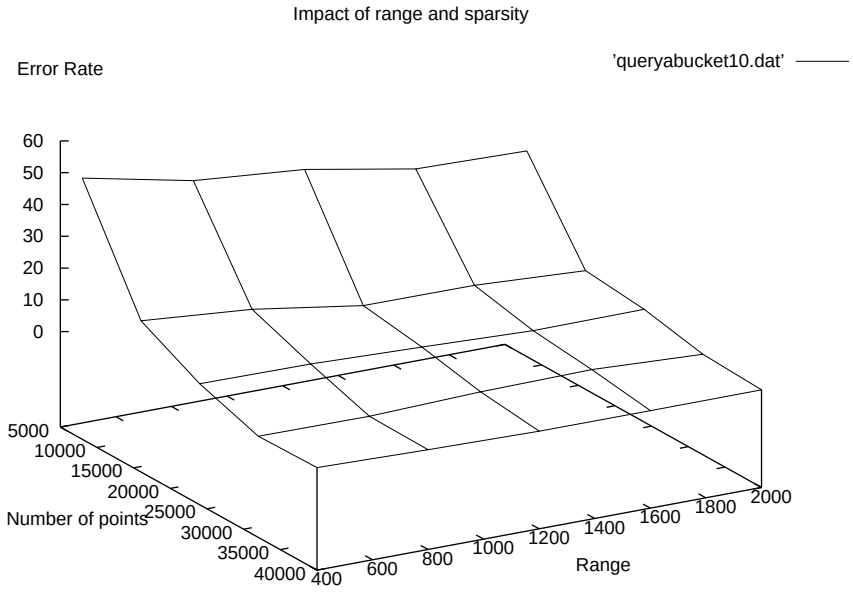



Figure 4.5: Performance measures for a dense query and 10 buckets.

*Discussion:* These findings were as expected. Obviously, as the number of points increases, the histogram does not change, but the distribution of the points within the buckets will tend to be more uniform. Hence the accuracy increases.

The query range data is harder to explain. Intuitively, in general the higher the intersection area between a bucket and the query band the less is the error rate. When the query range is wider, the intersection areas between the buckets and the query band tends to be greater, in fact, the query band may completely cover many buckets. For those buckets that are covered completely, the estimation would be accurate.

### Bucket Size 20

Figure 4.6 shows that the error rate decreases exponentially as the number of points increase. The error rate also decreases slightly as the query range increases. This results are similar to the results in Section 4.2.2, with a slightly lower error rate here than in the previous section for most combinations of number of points and query ranges.

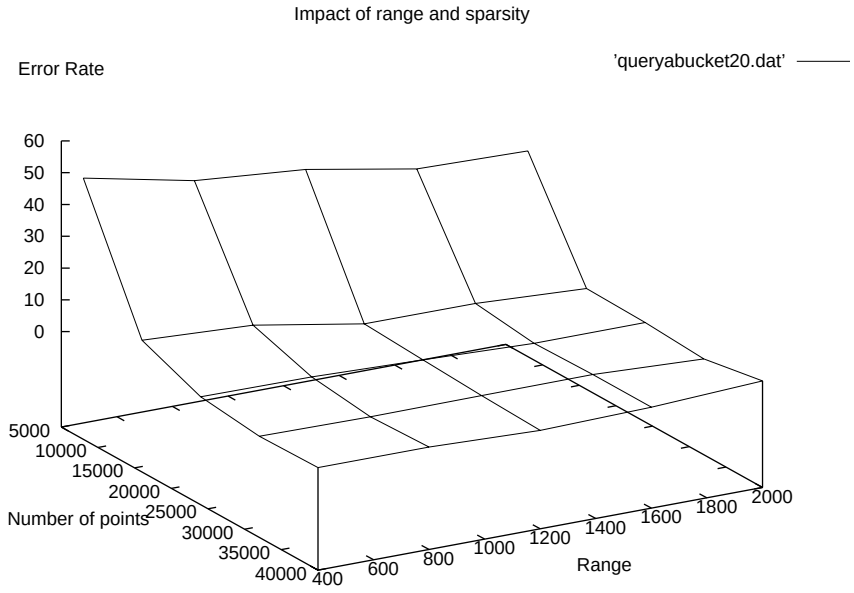



Figure 4.6: Performance measures for a dense query and 20 buckets.

*Discussion:* Intuitively, when more buckets are allowed in the histogram, the distribution in each bucket is more uniform, hence the total error rate should be lower. However, there is no visible decrease of the error rate when the number of buckets increases from 10 to 20. Apparently most of the extra buckets do not intersect with the query band, hence increasing the number of buckets does not significantly lower the error rate.

For dense queries, with either 10 or 20 buckets, a slight change in time could result in a large change in the estimate of the number of points in the query band. This explains why the error rate can be high (up to 50%) in the case of a relatively few number of points but remains low in the case of a high number of points. Apparently, the estimate and the actual values change more in tandem with the higher density.

### 4.2.3 Sparse Queries

*Sparse queries* mean those queries that are the opposite of dense queries. In sparse queries, the duals of the moving query points are located in a sparse region of the histogram.

### Bucket Size 10

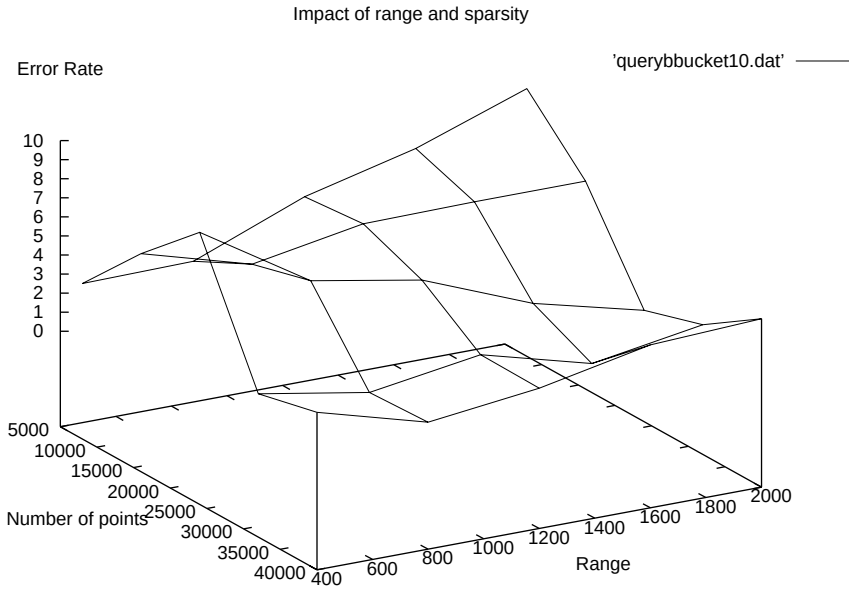



Figure 4.7: Performance measures for a sparse query and 10 buckets.

Figure 4.7 shows the relationship of the error rate, number of points and query range when the bucket size is 10 in the case of sparse queries.

The error rate is always relatively small, always below 10%. There is no clear relationship between the error rate and query range. In fact, the error rate decreases about linearly when the number of points is 24,000, but it increases linearly when the number of points is 8,000 and 40,000. Similarly, there is no clear relationship between the error rate and the number of points. For example, the error rate goes up and down for query range 400 and down and up for query range $2,000$.

*Discussion:* The lack of a clear relationship between the error rate and the query rate in this case may be due simply to the fact that the error rate remains lower than 7% in most cases. With such a relatively small error rate, the ups and downs in Figure 4.7 cannot be statistically significant.

### Bucket Size 20

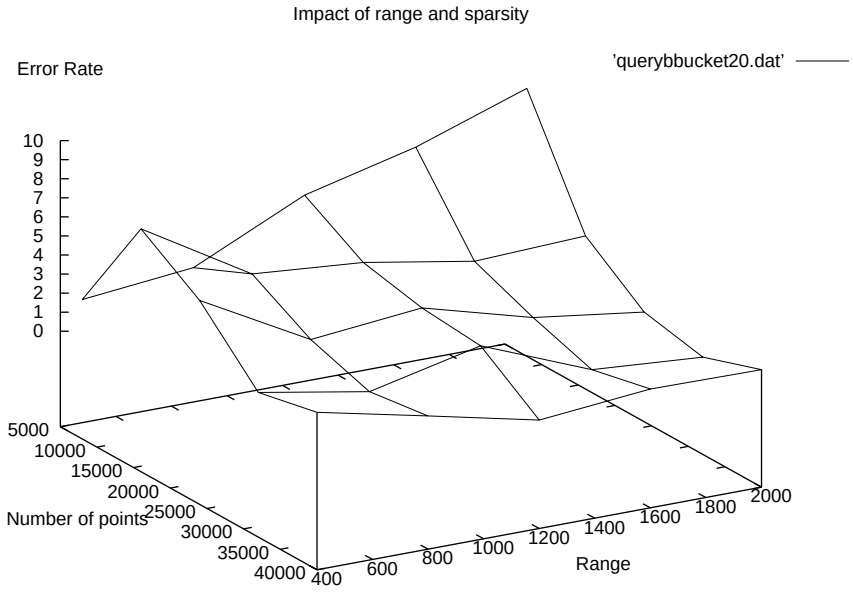



Figure 4.8: Performance measures for a sparse query and 20 buckets.

Figure 4.8 shows that the error rate is again very small in most cases of sparse queries when the bucket size 20. The highest error rate occurs in one corner of the picture when the number of points is $8,000$ and the query range is $2,000$. There seems to be a decrease in the error rate from that corner in any direction, either decreasing the query range or increasing the number of points.

*Discussion:* In many ways, these results are similar to those in Section 4.2.3. The most surprising result again is that the error rate is small, always less than 10%. It was also noteworthy that in the case of sparse queries the average error rate seems to be slightly lower with 20 buckets than with 10 buckets.

For sparse queries, with either 10 or 20 buckets, a slight change in time results only in a small change in the estimate of the number of points in the query band. This explains why the error rate is always small even when there are relatively few number of points.

Our experiments show that the query type is an important, perhaps the most important,

parameter in the performance of the *Max-Count* estimation algorithm. That is surprising, because it is a less obvious variable than the others. However, even in the case of dense queries a good performance can be guaranteed if the number of points is high, the query range is not too small, and the bucket size is 10 or higher.

## 4.3 Summary

The *Max-Count* aggregation problem arises as a natural problem for moving object databases. Our estimation algorithm is a practical solution when the objects move along one line, the number of moving objects is large, and the query range is not too small. Extensive experiments show that a histogram with a small number of buckets can can achieve high estimation accuracy.

# Chapter 5

# DataFoX: An Constraint-based Spatiotemporal XML Mediator System

XML has emerged as the standard data model for data representation and exchange on the Internet. Several XML-based languages encode geographic information. XML is easily extended to represent almost any spatio-temporal data model. Vector Markup Language (VML) can represent vector objects, and Geography Markup Language (GML) can represent coordinates of OpenGIS features. The parametric rectangle data model is encoded easily in XML. We refer to this encoding as PRML. However, the proposed XML query languages, for example, XQuery [11], Quilt [12], Lorel [3], XML-QL [24] and XPathLog [42] do not fully support geographic queries of spatio-temporal XML documents. Other XML query solutions, such as querying XML documents using relational database systems [9, 26, 28, 55, 54], do not support spatio-temporal queries.

The previously proposed XML query languages have the following limitations.

- The method of querying XML using relational database systems includes both the translation of XML schemas into relational schemas and the translation of tree-

structured XML data into relational tables. Some semantic information that is encoded in the tree structure might lost. For example, a rectangle and a straight-line object in GML have identical structures. These is no automatic translation algorithm that can maintain the XML schema information.

- Tree algegras do not allow grammar heterogeneity of XML data. For example, a single spatial object with rectangle shape also can be encoded as the combination of two rectangles. The structure and content of the GML data that result from these different combinations are different. The proposed tree algebra regards them as two different objects.

- Many XML standards provide a strict structure for part of the data. For example, spatial attribute data in GML are well structured. Unfortunately, most previously proposed XML query languages and XML algebras do not provide sufficient support for well-structured data in a semi-structured environment. For example, previously proposed XML query languages do not support spatio-temporal queries on GML although the spatio-temporal attributes are well structured in GML.

To overcome the above problems, a rule-based XML query language, Datalog For XML (DataFoX), is presented, which combines the simplicity of Datalog with the support for spatio-temporal data in constraint databases [35, 48]. A *Layer Algebra*, which is an extension of relational algebra for XML is also introduced. This algebra provides the basis of query evaluation and optimization for XML queries. The primary challenges we address are: (i) how to identify and represent trees in the query language, and (ii) how to evaluate the tree-based query language.

## 5.1 The DataFoX System Architecture

The architecture of the DataFoX system is outlined in Figure 5.1. XML Data sources in different formats are wrapped into a uniform representation called *Layer Constraint*

*Databases*, which are powerful for capturing the XML tree structure and GML, VML, and PRML spatio-temporal information. The DataFoX query language is translated into a Datalog query, which is further translated into the layer algebra. The query is evaluated and the query result may be translated back to XML encodings using wrappers.

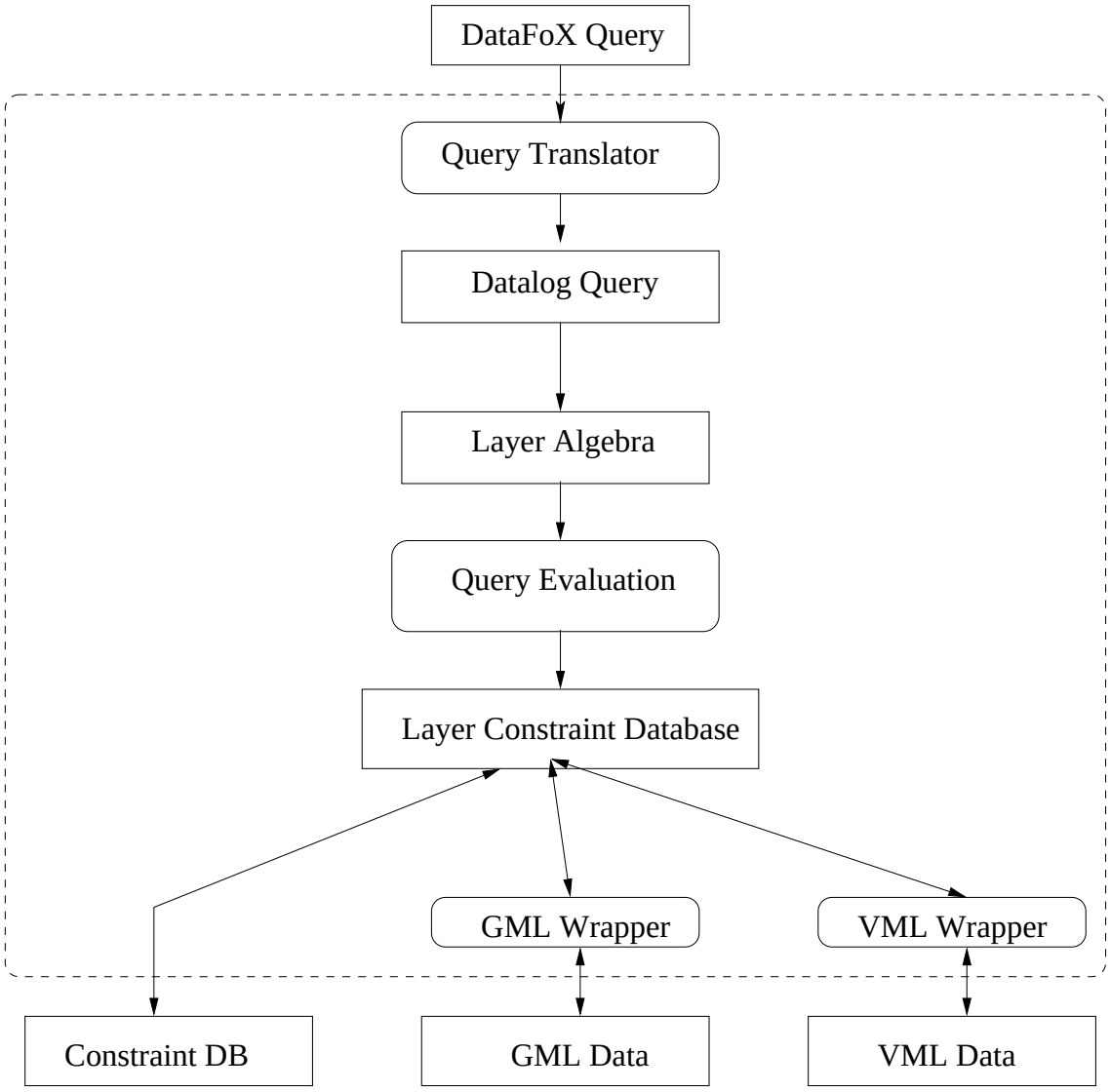



Figure 5.1: The DataFoX Architecture

Note that this architecture not only supports spatio-temporal queries using the constraint data model, but it also offers the power to integrate heterogeneous data sources. Consider the following example shown in Figure 5.2. This digital map of part of an uni-

versity campus is composed of different data sources with different formats. The building data are represented in GML shown in Figure 5.3. The stadium, which has the shape of an ellipse, is represented in VML as shown in Figure 5.4. A moving bus, following a rectangle route with a constant speed, and a helicopter, flying northwest, are represented in PRML as shown in Figure 5.5, the XML encoding of Parametric Rectangles [8].

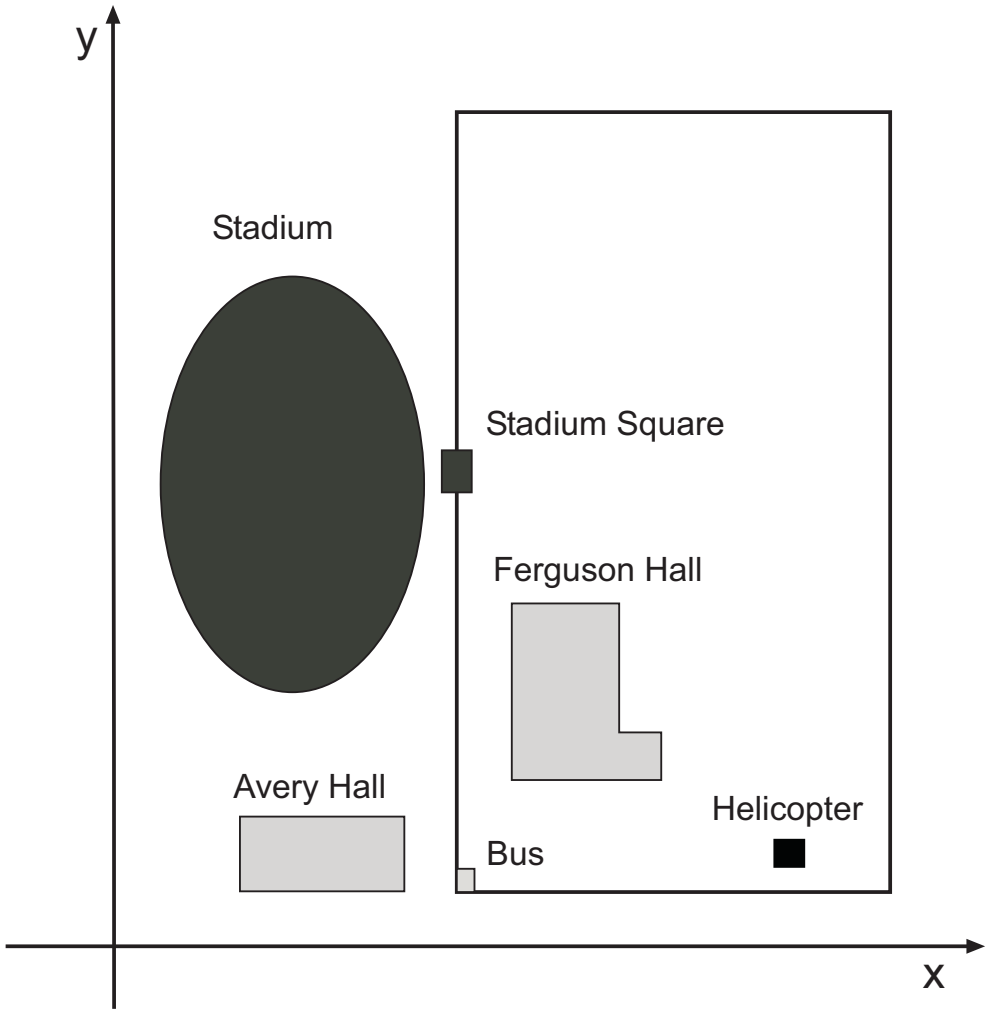



Figure 5.2: Campus Map

Typical queries on this map can be stated as:

Query 1 Find out when the bus will pass the east gate of the football stadium.

Query 2 Find out when the helicopter will fly over the stadium.

```
<buildings>
  <lecture_hall>
    <name> Ferguson Hall </name>
    <dept> Computer Science </dept>
    <boundedby>
      <rectangle>
        <coord> <x> 35 </x> <y> 15 </y> </coord>
        <coord> <x> 45 </x> <y> 30 </y> </coord>
      </rectangle>
      <rectangle>
        <coord> <x> 45 </x> <y> 15 </y> </coord>
        <coord> <x> 50 </x> <y> 20 </y> </coord>
      </rectangle>
    </boundedby>
  </lecture_hall>
  .
  .
  .
</buildings>
```

Figure 5.3: A fraction of GML document

Query 3 Find out if the helicopter will fly right above the bus. If yes, find out when the helicopter will fly right above the bus.

In the DataFoX system, the three different data sources can be translated into layered constraint databases.

## 5.2 Layer Model

In XML, the type of each node is defined as an "element". nodes are regarded as the counterpart of tuples in relational databases. The difference is that the data type of the field in a certain element could be another element. All of the nodes of the same element type belong to a *layer*, which is the counterpart of a relation in relational databases. The layer name is the same as the element name defined for these nodes. Layer A is the parent

```
<sport_facilities>
  <stadium>
    <name> Husker Stadium </name>
    <shape>
      <ellipse>
        <x> 15 </x>
        <y> 40 </y>
        <w> 10 </w>
        <h> 15 </h>
      </ellipse>
    </shape>
  </stadium>
  <square>
    <name> Stadium Square </name>
    <shape>
      <rectangle>
        <x> 30 </x>
        <y> 40 </y>
        <w> 5 </w>
        <h> 5 </h>
      </rectangle>
    </shape>
  </square>
</sport_facilities>
```

Figure 5.4: VML Document

layer of layer B if the element B is the sub-element of element A in the XML document.

Define *Inferred Layers* of a layer $L$ as the set of layers, which are child layers of $L$. Similarly, the *Inferred Layer of a node* is the sub-tree that is rooted from this node. Thus, the inferred nodes of a certain node $n$ and the edges between these nodes form an XML treewith the root $n$. Assume that each node has a virtual attribute called *nid* as the identifier of the node and a hidden attribute *pid* as a pointer to the parent node. we may use the *nid* to access a node, from which we may access its child nodes. In DataFoX, we also use this "nid" to identify the sub-tree that is rooted with the current node.

```
<transportation>
  <vehicle>
    <name> Bus </name>
    <schedule>
      <prectangle>
        <x>
          <from> <a> 0 </a> <b> 32 </b> </from>
          <to> <a> 0 </a> <b> 33 </b> </to>
        </x>
        <y>
          <from> <a> 1 </a> <b> 5 </b> </from>
          <to> <a> 1 </a> <b> 6 </b> </to>
        </y>
        <t>
          <from> 0 </from>
          <to> 55 </to>
        </t>
      </prectangle>
      .
      .
      .
    </schedule>
  </vehicle>
  .
  .
  .
</transportation>
```

Figure 5.5: PRML Document

Also assume *nid* is ordered, to maintain the order information of the XML tree model. For those elements defined as primitive data types, assign an internal attribute named *data* for the elements defined as primitive data types, and the value of this attribute is the content of the element. For example, the node in XML document `<author>Maggie</author>` can be represented by the schema `Author(nid, data)` such that the value of "data" is "Maggie" in this case.

Apply the same notation used to represent relational schema to represent the layer schema. However, the domain of variables for the layer schema is "tree". For example,

the layer schema `LAYERNAME(`$L_1$`,...,`$L_n$`)` means that there is an element named "LAYERNAME" defined in the XML document, which has $n$ child elements, with the layer names $L_1$, $L_2$, ..., $L_n$ respectively. The data type of the child layers can be atomic (e.g.: string, integer etc.) or a tree type defined by another layer schema. An instance of a layer and all of its inferred layers is a tree, whose root is an instance of the root layer.

The layer definition can be generated from the element definition of the XML document. Assume that every XML document comes with a Data Type Definition (DTD). XML Schema is a stronger schema definition of XML, but DTD is adequate. (Note: we can always generate DTD from the XML schema to create the layer data model.) With the layer data model, an XML document is treated as a collection of layers, and the traversal of the XML document is actually the traversal between the layers.

It is emphasized that the layer data model shares common features with the relational data model. However, in DataFoX, the layer model is a logical representation of XML, and the XML is not translated into flattened tables.

**Example 5.2.1** (Layer Model)

The XML document shown in Figure 5.6 can be modeled in the layer model shown in Figure 5.7. Every layer corresponds to an "element" definition in DTD. The *book* layer and *author* layer are called the *inferred layers* of the *library* layer. The layers with primitive data types like "PCDATA" are merged into their parent layer for simplicity.

## 5.3 DataFoX: Layer Algebra

With the self-description feature of XML, every spatio-temporal data can be encoded easily in XML. Typically, GML defines a class of spatial features to represent OpenGIS data and VML encodes vector information in XML. The Parametric Rectangle data model, used to model moving objects, also can be encoded in XML, which we refer to as PRML.

```
<!ELEMENT library (book*)>
<!ELEMENT book    (title, year,
                   author+)>
<!ELEMENT title   #PCDATA>
<!ELEMENT year    #PCDATA>
<!ELEMENT author  #PCDATA>

<library>
   <book>
       <title>Landscape</title>
       <year>1997</year>
       <author> Steve </author>
   </book>
   <book>
       <title>Portrait</title>
       <year>2000</year>
       <author>John</author>
       <author>Maggie</author>
   </book>
</library>
```

Figure 5.6: Library Document

These data models share a common characteristic, namely, the structures of the spatio-temporal attribute are well formatted, and generally form a well formatted sub-tree in the document (although the structure could be very complex.) Applying the layer data model, the upper layer is used to *aggregate* the object using constraints. The wrapper performs this work.

### 5.3.1 The Operators

A layer in XML can be regarded as a relation in relational databases. A tree-type variable is acceptable in a layer, and it is defined as child layers. We can define the operators on layers, which are very similar to those of relational databases. In this section, we introduce the Layer Algebra.

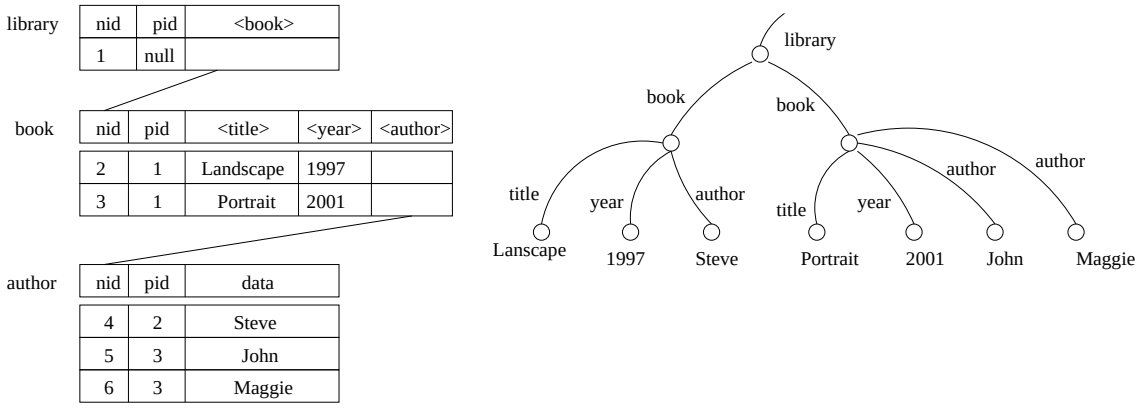



Figure 5.7: Layer Model for Library Document

### Selection

The "selection" operation identifies all nodes in a layer which satisfy some selection predicate. "selection" is a "horizontal" operation on XML in the sense that no traverse between layers is involved. The input of selection in layer algebra is a layer $L$, and a set of predicates $SL$ as parameters. It returns an output layer $O$, which has the same schema as the input layer $L$. The children of the output layer are lost except for those fields with atomic data types. The selection can be denoted as $\sigma^L_{SL}(L)$. A node $n$ belongs to the output layers if it satisfies the selection predicate $S$.

**Example 5.3.1** Consider the layer algebra query: $\sigma^L_{year=1997}(Book)$. The Figure 5.9 shows the result of selection operation on the layer model shown in Figure 5.7.

### Projection

Define the projection operation as an orthogonal operation to the selection operation. It is a "vertical" operation on XML in the sense that the traverse between layers is involved in the operation. The input of projection is a layer $L$ (and all of its inferred layers), and a set of projection list $PL$ as projection parameters. The projection list is a set of child layer names of the input layer $L$, which are associated with a set of sub-elements of the element defined for the input layer $L$. The output of projection operation is an output layer

$O$, such that only those child layers that satisfy the projection predicates are kept as the inferred layer of output layer $O$. That is, the child layers in the output layer are a subset of child layers in the input layer, and all child layers of these returned layers are copied to the output. The projection can be denoted as $\pi^L_{PL}(L)$. Formally, the projection operation is defined as follows.

- A node $n$ in the input layer belongs to the output if and only if the element type of it is included in the projection list $PL$.
- A node $m$ in the input layers belongs to the output layers, if the parent node of $m$ belongs to the output layers.

**Example 5.3.2** Consider the projection operation $\pi^L_{title,year}(Book)$ on the XML shown in Figure 5.7. The result of projection is shown in Figure 5.10.

**Example 5.3.3** The projection operation should be considered as the vertical tree traversal operation, that return a sub-tree from a set of nodes. For example, the selection operation shown in Example 5.3.1 only returns a single node, and the author information which is the child of the returned node is lost. The projection operation will return a sub-tree from the current node according to the schema of the source document as shown in Figure 5.11. Note that the internal attribute *nid* is used to denote the whole node.
Consider the query: *Find all information about books that were published in 1997.*
This can be expressed as: $\pi^L_{nid}(\sigma^L_{year=1997}(Book))$

### Product and Join

The input of a product operation is two layers, $L_1$ and $L_2$, with the schema $L_1(L_{1,1}, ..., L_{1,m})$ and $L_2(L_{2,1}, ..., L_{2,n})$. The output of the product operation is a new layer $O$, with the layer schema $O(L_{1,1}, ..., L_{1,m}, L_{2,1}, ..., L_{2,n})$. Formally, the product operation can be defined as follows.

For every pair of nodes $n_1$ and $n_2$, such that $n_1$ belongs to layer $L_1$ and $n_2$ belongs to layer $L_2$, a new node $n$ is created for the output layer $O$. The new nodes have $m + n$ children, such that the first $m$ children are the same as the children of $n_1$, and the remaining $n$ children are the same as the children of $n_2$.

The join operation can be expressed as selection operation posed on the result layer of the product operation. The formal definition of the join operation can be defined as follows.

**Definition 5.3.1 (Join)** Let $L_1$ and $L_2$ be two layers, with the schema $L_1(x_1, ..., x_i, ..., x_m)$ and $L_2(y_1, ..., y_j, ...y_n)$, respectively. Then the *join operation* $\bowtie$ creates a new layer $L$, such that

- The layer $L$ has the schema $L(x_1, ..., x_i, ..., x_m, y_1, ..., y_j, ..., y_n)$.
- If a node $n_1$ belongs to layer $L$, there must be two nodes $n_1$ and $n_2$ that belongs to layer $L_1$ and $L_2$ respectively, such that the inferred layer $x_i$ of node $n_1$ has the same value with that of the inferred layer $y_j$ of node $n_2$.

The join operation can be denoted as $L_1 \bowtie^L_{x_i=y_j} L_2$.

### Path Join

We introduce *Path Join* to join two layers with path predicates. The input is two layers $A$ and $B$, and the output of the path join operation is two layers, $A'$ and $B'$, which has the same schema with $A$ and $B$ respectively. The path join operation will have the form of $A \bowtie^L_{PP} B$, while $PP$ i s the path predicates. The path join can be defined formally as follows.

- A node $n$ belongs to the output layer of $A'$ if and only if $n$ also belongs to $A$, and there exists a node $m$ in $B'$, such that $n$ and $m$ satisfy the path predicate $n\theta m$.
- A node $m$ belongs to the output layer of $B'$ if and only if $m$ also belongs to $B$, and there exists a node $n$ in output layer $A'$, such that $m$ and $n$ satisfy the path predicate $n\theta m$.

**Example 5.3.4** The query in the Example 5.3.3 also can be expressed in path join in the following: $(\sigma^{L}_{year=1997} Book) \bowtie^{L}_{Book.nid/Author.nid} Author$

### 5.3.2 Translation of Layer Algebra

Layer Algebra is the basis for query evaluation and optimization. This section shows the translation of Layer Algebra to Relational Algebra.

**Theorem 5.3.1** Let $L$ be an expression of Layer Algebra. There is an expression $E$ in Relational Algebra that is equivalent to $L$.

**Proof** A Layer Algebra expression $L$ that consists of projection, selection and join can be translated into a Relation Algebra expression $E$ because there are equivalent operators for each layer algebra operator.

- (Selection) Selection operation only returns a set of nodes that belong to the input layer. Thus, the layer algebra $\sigma^{L}_{SL}(L)$ can be translated into relational algebra expression $\sigma_{SL}(R)$, such that the relation $R$ has the same schema with the layer $L$.
- (Projection) For a projection operation with the form $\pi^{L}_{PL}(L)$, such that the $PL$ is the list of fields to be projected, translate the expression into relational operation $\pi_{PL}(L)$, followed by $\sigma_{IL}(L)$, in which $IL$ is the inferred layers of $L$.
- (Join) The join operation on two layers can be translated straightforwardly into a join operation on two relations.

For the selection, projection and join operation on a layer that do not have inferred layers, the translation is straightforward.

**Example 5.3.5** Consider the join operation between the book layer and the publisher layer, $Book \bowtie^{L}_{Book.title=Publish.title} Publish$.

Figure 5.12 (a) and (b) show the original data, Figure 5.12 (c) shows the schema of the output for the join operation, and Figure 5.12 (d) shows the tree structure result of the join operation.

## 5.4 DataFoX Query Language

DataFoX is a Datalog-like declarative query language. It is a high-level language that extends Datalog with tree type variables. A DataFoX query can be translated into a constraint query, which can be evaluated in the running constraint database system MLPQ.

### 5.4.1 Syntax

The input of a DataFoX query is a set of XML documents, and the output of a DataFoX query is also an XML document. Each DataFoX query consists of a finite set of rules of the form:

$$Q(y_1, y_2, ..., y_m) : -R_1(x_{1,1}, ..., x_{1,k_1}), ..., R_n(x_{n,1}, ..., x_{n,k_n})$$

where each $R_i$ is either a name of element in the XML document or a defined relation name, including predefined spatio-temporal function names. The $x$'s and $y$'s are either variables, or constants. When $x_{i,j}$ is a variable, it is bounded to the $j$th child of the $R_i$ element. The sub-element type of the variable $x_{i,j}$ is allowed, by adding the element name $e_{i,j}$, with the form of $e_{i,j} : x_{i,j}$.

The rule head relation $Q$ is the root element name of the query result. The variables $y_i$ define the values of the elements which are the sub-elements of the resulting root element $Q$. Each variable $y_i$ has to appear somewhere in the rule body.

#### DataFoX Query Body

We categorize the predicates in the DataFoX query body into three classes: Extensional and Intensional Predicates, Built-in Predicates, and user defined functions. Similar to Datalog, the extensional predicates are relational database relations and the intensional predicates are the database relations defined by the rules.

Built-in predicates include arithmetic comparison predicates, $=$, $<$, and so on. "/" and "//" are two path predicates which refer to the parent-child and ascendant-descendant

relationships of two nodes. For example, $x/y$ means the node $x$ is the parent node of the $y$.

A variable in DataFoX can denote a tree in XML. An internal attribute node ID (*nid*) for an extensional predicate refers to the current node that satisfies the predicate, and this nid is used to identify the tree with the root node nid. For example, in the predicate $A(a, B : b, C : c)$ the variable $a$ refers to the current node with the element type $A$, which has two sub-elements $B$ and $C$. Two boolean predicates will hold from this extensional predicate: $a/b$ and $a/c$, because the node $b$ and $c$ are both children of node $a$.

**Example 5.4.1** Consider the following query on the XML document show in Figure 5.3.2: *Find all the lecture halls that have offices for the computer science department.* The query is expressed as:

```
CS_Lecture_Hall(lecture_hall:lh):- Lecture_Hall(lh, dept:"Computer Science")}.
```

The variable *lh* in the query body is an internal attribute referring to the node with the type `lecture_hall` that satisfies the query condition. The query result is an XML document with the root element "CS_Lecture_Hall" and a set of "lecture_hall" trees as the children of the root element.

User-defined predicates are introduced to handle the operation between tree type variables. Tree data is more complex then atomic data. The predicate between two tree variables can not simply be mapped to the predicates of the nodes and edges of the tree. For example, a single spatial object can be divided into two rectangles in two different ways. Thus, this spatial object can be encoded in GML in two different ways, but they are describing exactly the same object. Equality, as applied to two trees, means that there exists a mapping between nodes or edges of the two trees. This concept focuses on the equality of structure detail, but misses the semantic information provided by the tree as a whole.

The introduction of user-defined predicates also allows hiding of structure detail of variable from user. This provides a data integration mechanism.

**Example 5.4.2** Consider the query:

*Find all the buses that will intersect with the bus named "schoolbus" from time 0 to time 100.*

This can be expressed as:

```
Vehicle(bus: u):- Bus(v, name:''schoolbus''),
                  Bus(u),
                  Intersect(v, u, x, y, t),
                  0<=t, t<=100.
```

In this example, buses $u$ and $v$ have the same data type, defined as "Bus" element in the document, but the structural details of the data are hidden. The user-defined predicate *Intersect* handles the details of these two trees.

### DataFoX Query Head

Each DataFoX query head has two roles: (1) it specifies the projection operation on the relation created by the query body and (2) it defines the schema for each variable.

This means that each variable in the DataFoX rule head must appear in the rule body. The data type of a variable in the rule head is the same as that of the matching variable in the rule body. The query result is tagged as an XML document according to the query head.

Consider the query in Example 5.4.1, the query head `CS_Lecture_Hall( lecture_hall:lh)` defines the result schema, with the root element `CS_Lecture_Hall`. The element contains a child element `Lecture_Hall` which has the value $lh$. Note that $lh$ is a tree variable, the projection operation on this variable copies the whole substree start from this element.

### 5.4.2 Tree Operation

One of the advantages of Layer Algebra for XML is that it provides a data abstraction based on the tree data types. A sub-tree in XML is abstracted as a tree type variable in the root layer of this sub-tree. The details of the tree variable (the structure and content) are all hidden from the user. In implementation, the corresponding XML wrapper will be triggered when a tree variable is used in the query.

A typical application of tree operation is the manipulation of spatio-temporal data encoded in XML.

**Example 5.4.3** Consider the query:

*Find the intersect time and position of two vehicle.*

This can be expressed as:

```
Intersect_time(x, y, t) :- Vehicle(name:''Bus'', schedule:s1),
                           Vehicle(name:''Shuttle'',schedule:s2),
                           Intersect(s1, s2, x, y, t).
```

The schedule layer and its inferred layers encode spatio-temporal information of the vehicle in a tree structure. Tree variables $s_1$ and $s_2$ are used to abstract the tree and take part in the intersect spatio-temporal operation. The tree structure details are hidden from the user.

The operations on tree structures are very difficult to express in other XML query languages. Consider the intersection of two line segments $AB$ and $CD$ represented in GML. The coordinates of the four points can be queried by XQuery. Suppose these four points are represented in coordinates $A(0,0)$, $B(5,5)$, $C(0,5)$, $D(5,0)$. The coordinates of the intersection point is $(2.5, 2.5)$, which does not exist in the XML source. Handling spatial data types is an even more complex problem. For example, the intersection of two rectangular objects could be another rectangle, but it also could be a line segment or a single

point. To represent this query with XQuery language, the user has to take into account each case separately.

In the layer model, a tree type variable is used to represent the spatio-temporal object and introduce a set of built-in operations defined on these variables. For example, a function *area*() can be defined on the Boundedby element in GML to compute the area of the region represented by this element. The task of translating the spatio-temporal objects to a constraint representation is shifted to the wrapper. That makes the query language easier to understand.

### 5.4.3 Recursive Queries

As a rule-based query language, DataFoX also can express recursive queries. Figure 5.6 shows part of a large an XML document that contains descriptions of books. Suppose author Maggie wants to find potential co-authors for her future books. The potential co-authors should be co-authors of her books, or those recommended by the co-authors. Suppose that everyone will recommend to Maggie his/her co-authors and everyone who is recommended to him/her. The query can be expressed as follows:

```
Co-Author-Book(book: bk) :- Book( bk, author: ‘‘Maggie’’).
Co-Author-Book(book: bk) :- Co-Author-Book(book: bk0),
                            Book(bk0, author: auth),
                            Book(bk, author: auth).
Recommended_Author(author: auth):- Co-Author-Book(book: bk),
                                   Book(bk, author: auth).
```

The first rule finds all books authored by Maggie. The body says the sub-tree bk has an element author with the value “Maggie”. The second rule finds those books that have a common author with some book written by recommended authors. The third rule projects all authors involved in those books. This example shows the power of recursive queries in DataFoX, which is not supported in XQuery.

### 5.4.4 DataFoX Evaluation

To support the tree data type, DataFoX includes path predicates and tree operations. This section addresses the problem of DataFoX evaluation.

**Theorem 5.4.1** (Least Fixpoint Evaluation) The least fixpoint of any DataFoX query and input database (XML document) with path constraints is closed-form evaluable.

**Proof** Each of the extensional database sub-goals in the DataFoX rule body is associated with a layer with the scheme defined in the input database, which is an XML document. The path constraints with the form $a/b$ and $a//b$ can be expressed in Datalog as $parent(a, b)$ and $ancestor(a, b)$ respectively. Thus, there is an equivalent Datalog with set data types where the set variables range over R3 and are specified with linear constraint. It is well-known that this case of Datalog is closed-form evaluable in this case [48].

### 5.4.5 Translating DataFoX Queries

DataFoX uses a similar syntax to Datalog, and Layer Algebra is an extension of Relational Algebra with tree operations. This section discusses the translation of DataFoX rule bodies into Layer Algebra.

**Algorithm 2 Computing the layer for a DataFoX rule body using Layer Algebra**
**INPUT:** The body of a DataFoX rule $r$, which consists of subgoal s $S_1, ..., S_n$. For each $S_i = p_i(A_{i,1}, ..., A_{i,k_i})$ with an ordinary predicate, there i s a layer $L_i$ already computed where the $A$'s are either an internal attribute $nid$ for the layer or a variable term with the form $e_{i,k} : x_{i,k}$, or a constant term with the form $e_{i,k} : c_{i,k}$.
**OUTPUT:** An expression of layer algebra.
**METHOD:**

1. For each subgoal $S_i$, let $Q_i$ be the expression $\pi^L_{PL}(\sigma^L_{SL}(L_i))$, such that all variables appear in the predicate are included in the projection term $pl$, and $sl$ is the conjunction of the following conditions: (i) If there is a constant $a$ appear in the subgoal with the

form $e_{i,k} : a$, then $sl$ has the term $e_{i,k} = a$; and (ii) if two sub-elements have the same variable with the form $e_{i,k} : x, e_{i,l} : x$, then $sl$ has the term $e_{i,k} = e_{i,l}$.

2. For each subgoal $S_i$ which is path predicate with the form $a\theta b$, let $Q_i$ be the expression $L_{i,j} \bowtie_{L_{i,j}.nid\theta L_{i,k}.nid} L_{i,k}$, such that the variable $a$ and $b$ appears as the internal attributes in two other subgoals $S_j$ and $S_k$ respectively.

**Theorem 5.4.2** Translating a DataFoX Query

Let $Q$ be a DataFoX query not involving recursion, function calls (spatio-temporal functions). Then, there is an expression $E$ in layer algebra that is equivalent to $Q$.

**Proof** The DataFoX query body can be translated into Layer Algebra expression by the Algorithm 2, with the result expression $E$. The query head which consist of variables $X_1, ..., X_n$ can be expressed in layer algebra by the projection of the $n$ variables from the expression $E$. That is, the query statement can be expressed as $\pi^L_{X_1,...,X_n}(E)$.

The input of a DataFoX query is a set of EDBs (Extensional Databases) in the form of XML and a set of IDBs (Intensional Databases) which can be generated from other rules. The major challenge of DataFoX evaluation resides in the manipulation of tree type variables. The essential steps in DataFoX evaluation include elimination of tree variables and substitution of spatio-temporal operators. The result of these two steps is a Datalog query with constraints.

The elimination of tree variables in path predicates is straightforward. The evaluation of the path predicate $a/b$ can be translate to relational algebra to check if there exists a tuple $(a, b)$ in the relation edge.

### 5.4.6 Translation to Layer Algebra

With Algorithm 2 all spatio-temporal operations can be converted into constraint queries. This section focuses on the translation to layer algebra. We also show how to eliminate path predicates.

The query body of DataFoX can be translated into Datalog easily, provided a schema information of the XML document (DTD format).

To simplify the query statement, DataFoX extensional and intensional predicates allow the explicit specification of the field name. The complete predicate can be retrieved by looking up the internal schema information.

The translation of DataFoX to layer algebra can be summarized by the following steps:

1. If any constant appears in a predicate, then the predicate is translated into a "selection" operation. For example, the predicate $Book(title :'' portrait'', ...)$ can be translated into $\pi_{title='\ portrait'}(Book)$.

2. For any variable that appears in more than one predicate, a "join" operation is used between these predicates. For example, the DataFoX query Example 5.3.5 can be expressed as:

```
Publishers():- Book(title:t),
               Publish(title:t).
```

   The same variable $t$ appears in two predicates, the corresponding algebra is $Book \bowtie^{L}_{Book.title=Publish.title} Publish$.

3. Path predicates, such as $a/b$, are regarded as a special path join operation between the two nodes and the edge relation. Example 5.4.4 shows how to translate a path predicate into layer algebra. Note that by giving an edge relation that stores all edges, the descendant predicate with the form $a//b$ also is translated into a set of join operations on the edge relation. The descendant predicate is described as recursive Datalog as follows:

```
Descendant(a, b) :- Parent(a,b).
Descendant(a, b) :- Parent(a,c),
                    Descendant(c,b).
```

DataFoX maintains a lightweight tree to abstract the path information and use a bottom-up traversal to bypass the evaluation of recursive join operations.

**Example 5.4.4** Consider this query of finding the book authored by "Steve".
The DataFoX statement can be expressed as follows:

```
Book(title: t) :- Book(id1, title: t),
                  Author(id2, ‘‘Steve’’),
                  id1/id2.
```

In this example, the book predicate is linked to the extensional database relation by looking up the schema information. The predicate $Book(id : id1, title : t)$ can be translated into relational algebra: $\pi_{id,title}Book$.

The query body can be translated into relational algebra as follows:
$\sigma_{Author.data='Steve'}(Book \bowtie_{Book.nid/Author.nid} Author)$

## 5.5 Implementations

Our implementation of the DataFoX query language is based on the MLPQ/PReSTO [43] constraint database system. We have implemented the wrapper for GML and PRML. Figure 5.13 shows in MLPQ the integrated digital map from different XML formats, GML and PRML. The map is represented internally by constraint databases.

In this example, the campus building data are encoded in GML, as shown in Figure 5.3. The moving bus represented in PRML is shown in Figure 5.5.

The DataFoX query to determine the time when the helicopter flies over the stadium is expressed in DataFoX as the following:

```
Fly_over_Stadium(x, y, t) :- Vehicle(name:’’Helicopter’’, schedule:p),
                             Lecture_Hall(name:’’Ferguson Hall’’, boundedby:q),
                             Intersect( p, q, x, y, t).
```

The DataFoX parser detects and aggregates the spatio-temporal variables in the statement and then translate the statement into Datalog. The corresponding Datalog representation of this query is:

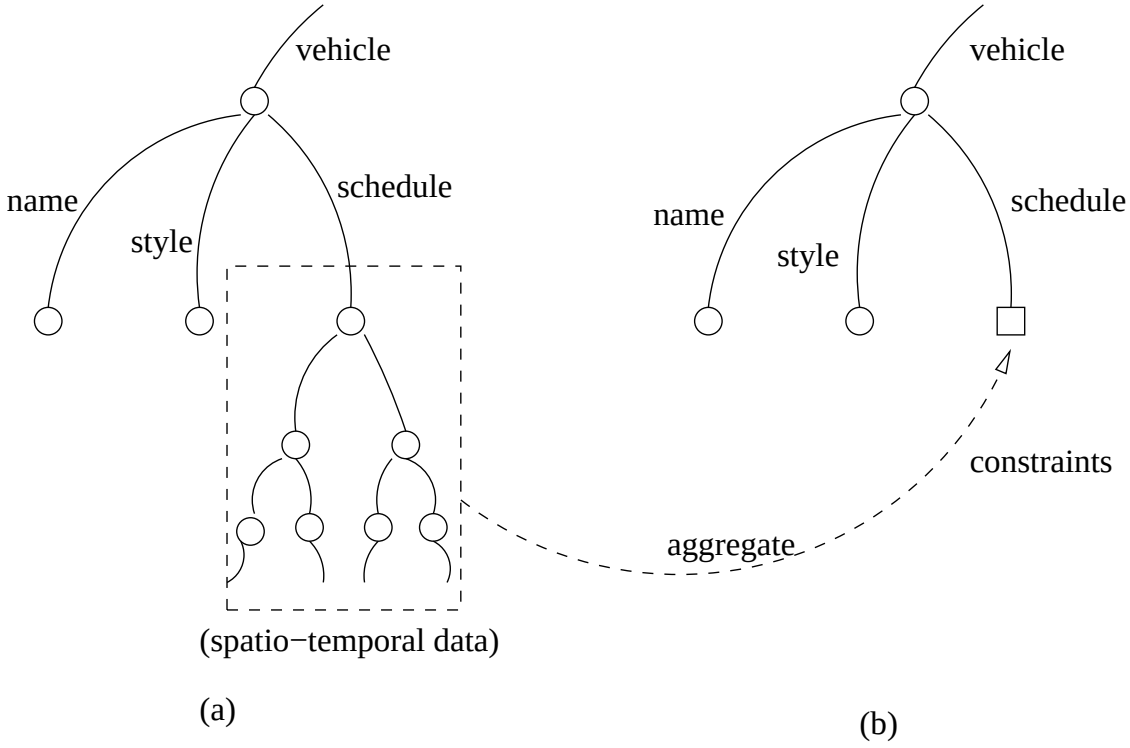
```
Fly_over_Stadium(x,y,t) :- Vehicle(id1, pid1, ‘‘Helicopter’’,x, y, t),
                           Lecture_Hall(id2, pid2, ‘‘Ferguson Hall’’,dept,x,y).
```

The constraints generated from the original GML data model shown in Figure 5.3 are as follows. The above Datalog query regards these constraints as the data source.

```
Lecture_Hall(id, pid, name, dept, x, y):- id=10, pid = 8,
                                          name = ‘‘Ferguson Hall’’,
                                          dept = ‘‘Computer Science’’,
                                          35 <=x, x <= 45,
                                          15 <=y, y <= 30.

Lecture_Hall(id, pid, name, dept, x, y):- id=10, pid = 8,
                                          name = ‘‘Ferguson Hall’’,
                                          dept = ‘‘Computer Science’’,
                                          45 <= x, x <= 50,
                                          15 <= y, y <= 20.
```

The Datalog query is evaluated and executed in MLPQ/PReSTO system and the query result is represented in the form of linear constraints. The wrapper produces the result with the corresponding XML encoding.




vehicle

| nid | <name> | <style> | <schedule> |
|---|---|---|---|
| ... | ... | ... | ... |
| 1 | Bus | ... | x=10, y=2t, 0<=t<=10 |
| 2 | shuttle | ... | ... |

(c)

Figure 5.8: Spatio-Temporal XML Aggregation

book

| nid | pid | <title> | <year> | <author> |
|---|---|---|---|---|
| 2 | 1 | Landscape | 1997 | |

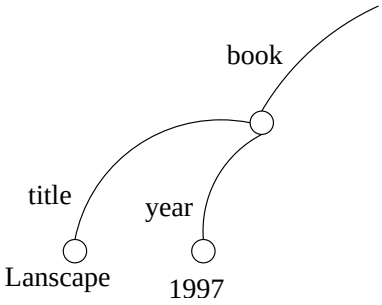



Figure 5.9: Result of Selection Operation

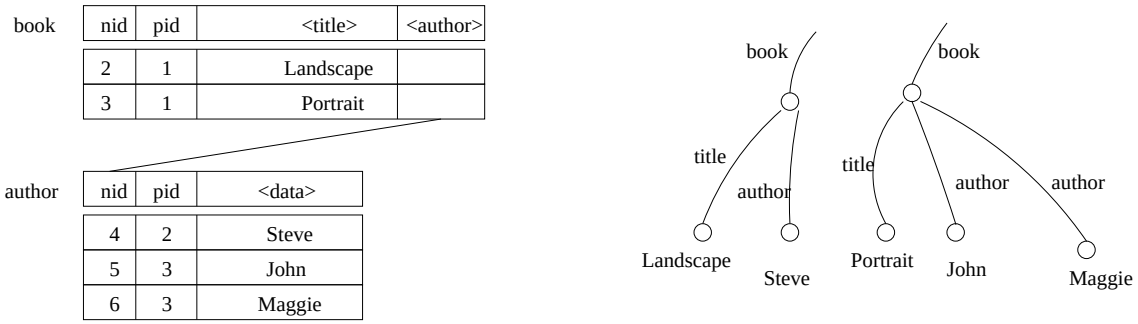



Figure 5.10: Result of Projection Operation

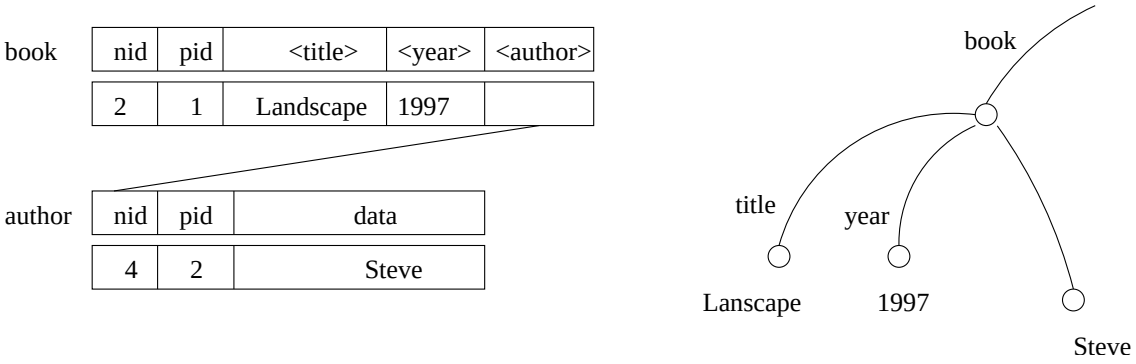



Figure 5.11: Vertical Traversal by Projection Operation

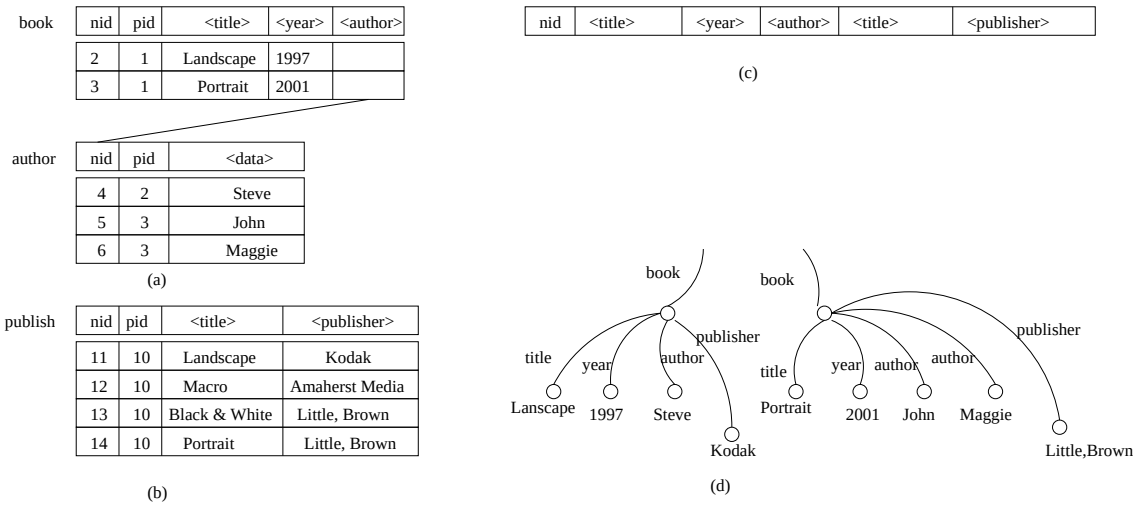



Figure 5.12: Join Operation

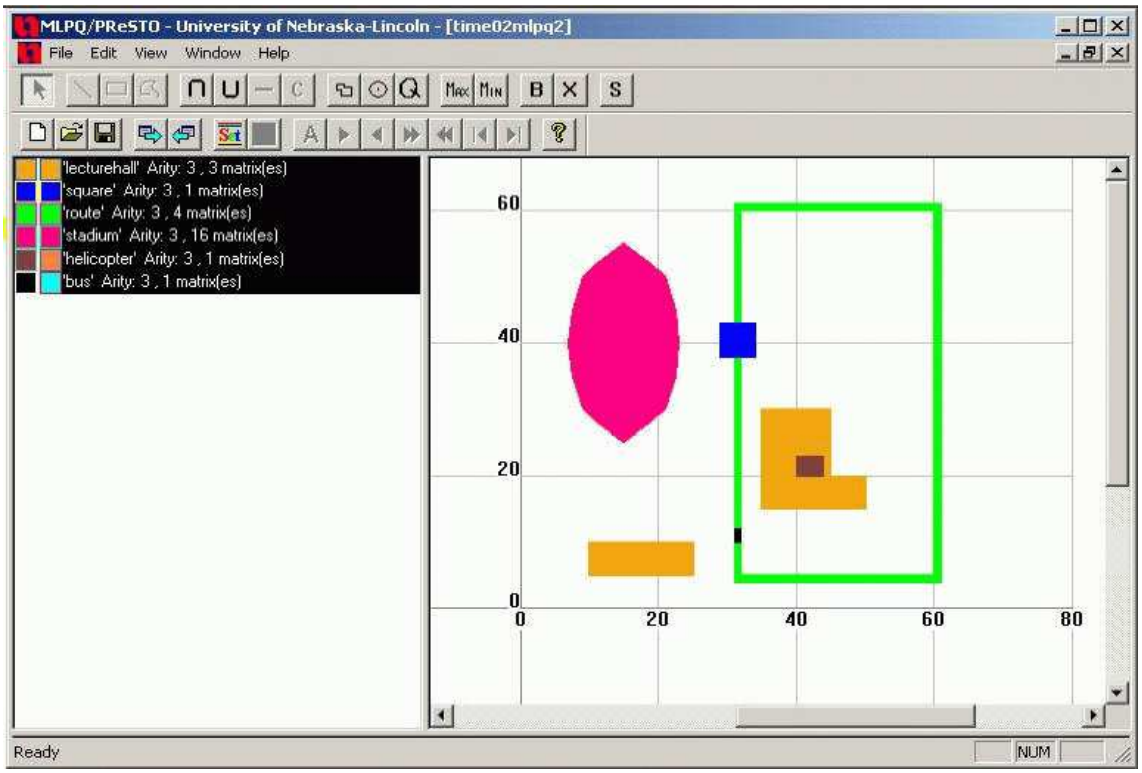



Figure 5.13: DataFoX Implementation on MLPQ/PReSTO

# Chapter 6

# Conclusion

This dissertation discusses aggregate operations over spatiotemporal databases. Efficient data structures are proposed to answer *count* aggregates, and a new aggregate operation *max-count* is introduced.

Constraint databases can easily represent moving objects. Linear functions of time were used to represent and approximate continuously moving objects. Either *PA tree* or *DT graph* data structures can answer *Count* aggregate queries efficiently.

The efficiency of the algorithms is examined from the perspective of computation time. Due to the space requirements of the above two data structures, these methods might not be practical enough in most commercial GIS systems for large data sets.

A new spatiotemporal aggregate operator *Max-Count* is addressed for the first time. The *dome* data structure can be used to answer *Max-Count* aggregation queries efficiently in logarithmic time. The histogram-based approximation algorithm also answers the same with $O(B)$, where $B$ is a chosen constant. The experiments show that a very small $B$ can be chosen to calculate the aggregation approximation without sacrificing much accuracy. This work may be very suitable for time-critical applications and seems the most practical of our algorithms.

The work discussed in this dissertation can be extended in several directions. Here are

some open problems:

1. Dynamic data structures should be studied, which are needed when for example the course of a moving object changes or a new moving object is introduced. In this case, the data structure should be efficiently updated.

2. Implementing polynomial constraints that represent moving objects is still an open problem. It is more accurate to use polynomial constraints to represent scientific data (for example, the moving of the planets along an elliptical course). Spatiotemporal aggregations with *Max-Count* should be investigated.

3. We also should study how to *estimate* efficiently the aggregate queries over moving objects that are represented by polynomial constraints.

4. The *Max-Count* aggregation problem needs to be investigated when the moving point objects move in a multidimensional space, not all along the $x$-axis. Answering this open problem probably would require another novel data structure.

# Bibliography

[1] International Organization for Standardization. Information processing - Text and office systems - Standard Generalized Markup Language(SGML).

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[4] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity Estimation in Spatial Databases. In *SIGMOD*, pages 13–24, 1999.

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

[6] M. Anderson and W. Woessner. *Applied Groundwater Modeling.* Academic Press, 1992.

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

[8] M. Cai, D. Keshwani, and P. Revesz. Parametric rectangles: A model for querying and animation of spatiotemporal databases". In *Proc. Seventh Conference on Extending Database Technology (EDBT), Springer-Verlag LNCS 1777*, pages 430–444, 2000.

[9] M. J. Carey, D. Florescu, Z. G. Ives, Y. Lu, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian. XPERANTO: Publishing object-relational data as XML. In *WebDB (Informal Proceedings)*, pages 105–110, 2000.

[10] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: A graphical language for querying and restructuring XML documents. In *Sistemi Evoluti per Basi di Dati*, pages 151–165, 1999.

[11] D. Chamberlin. XQuery: An XML query language, 2002.

[12] D. D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML query language for heterogeneous data sources. In *WebDB (Informal Proceedings)*, pages 53–62, 2000.

[13] C. Chan and Y. E. Ioannidis. Hierarchical cubes for range-sum queries. In *Proc. of VLDB*, pages 675–686, 1999.

[14] Y. Chen, J. Erickson, Y. Jiao, and M. Fayad. Building A Stable Model-Based Bidding/Quotation System. In *17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Workshop Presentation, 2002.

[15] Y. Chen and P. Revesz. XML-based integration of geographic data and queries. In *Proceedings of the Southwestern & Rocky Mountain Division, American Association for the Advancement of Science*, page 35, Omaha, Nebraska, 2001.

[16] Y. Chen and P. Revesz. CXQuery: A Novel XML Query Language. In *International Conference on Advances in Infrastructure for Electronic Business, Education, Science, and Medicine on the Internet*, CD-ROM 5 pages, L'Aquila, Italy, 2002.

[17] Y. Chen and P. Revesz. Indexing and aggregation estimation for moving points. In *SIGMOD (Submitted)*, 2003.

[18] Y. Chen and P. Revesz. Querying Spatiotemporal XML Using DataFoX. In *IEEE International Conference on Web Intelligence (WI)*, pages 301–309, Halifax, Canada, 2003.

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

[20] S.-J. Chun, C.-W. Chung, J.-H. Lee, and S.-L. Lee. Dynamic update cube for range-sum queries. In *The VLDB Journal*, pages 521–530, 2001.

[21] O. G. Consortium. Geography markup language.

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

[23] M. N. Demers. *Fundamentals of Geographic Information Systems*. John Wiley & Sons, New York, 2000.

[24] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A query language for XML.

[25] A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with STORED. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 431–442, 1999.

[26] A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with STORED. In *SIGMOD 99*, pages 431–442, 1999.

[27] A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. "XML-QL: A Query Language for XML". In *WWW The Query Language Workshop (QL)*, Cambridge, MA, , 1998.

[28] M. Fernandez, W. Tan, and D. Suciu. Silkroute: Trading between relations and XML. In *Processings of the Ninth International World Wide Web Conference*, 2000.

[29] S. Geffner, D. Agrawal, and A. Abbadi. The dynamic data cube. *Lecture Notes in Computer Science*, 1777:237–??, 2000.

[30] S. Goddard, S. Harms, S. Reichenbach, T. Tadesse, and W. Waltman. Geospatial decision support for drought risk management. *Communications of the ACM*, 46(1):35–38, 2003.

[31] D. Q. Goldin and P. C. Kanellakis. Constraint query algebras. *Constraints*, 1(1/2):45–83, 1996.

[32] R. Goldman, J. McHugh, and J. Widom. From semistructured data to XML: Migrating the lore data model and query language. In *Workshop on the Web and Databases (WebDB '99)*, pages 25–30, 1999.

[33] Herbert, Anderson, and M. Wang. *Introduction to Groundwater Modeling.* Academic Press, 1982.

[34] C. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in OLAP data cubes. In *SIGMOD*, pages 73–88, 1997.

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

[36] C.-C. Kanne and G. Moerkotte. Efficient storage of XML data. In *ICDE*, page 198, 2000.

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

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

[39] P. Longley, M. F. Goodchild, and D. D.J.& Rhind. *Geographic Information Systems and Science.* John Wiley, Chichester, 2001.

[40] B. Mathews, D. Lee, B. Dister, J. Bowler, H. Cooperstein, A. Jindal, T. Nguyen, P. Wu, and T. Sandal. Vector markup language (VML), 1998.

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

[42] W. May. XPath-Logic and XPathLog: A logic-based approach for declarative XML data manipulation.

[43] P. Kanjamala and P. Revesz and Y. Wang. MLPQ/GIS: A GIS using Linear Constraint Databases. In *9th International Conference On Management Of Data (COMAD' 98)*, 1998.

[44] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches in query processing for moving object trajectories. In *The VLDB Journal*, pages 395–406, 2000.

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

[46] P. Revesz. *Constraint Query Languages*. PhD thesis, University, 1991.

[47] P. Revesz. Safe stratified datalog with integer order programs. In *Principles and Practice of Constraint Programming*, pages 154–169, 1995.

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

[49] P. Revesz, R. Chen, P. Kanjamala, Y. Li, Y. Liu, and Y. Wang. The MLPQ/GIS Constraint Database System. *ACM SIGMOD Record*, 29(2), 2000.

[50] P. Revesz and Y. Chen. Efficient Aggregation on Moving Objects. In *10th International Symposium on Temporal Representation and Reasoning and 4th International Conference on Temporal Logic (TIME-ICTL)*, pages 118–127, 2003.

[51] P. Revesz and Y. Li. MLPQ: A linear constraint database system with aggregate operators. In *Proc. 1st International Database Engineering and Application Symposium*, pages 132–7, 1997.

[52] J. Robie, J. Lapp, and D. Schach. XML Query Language (XQL). In *Proc. of the Query Languages Workshop*, 1998.

[53] 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.

[54] J. Shanmugasundaram, E. Shekita, J. Kiernan, R. Krishnamurthy, E. Viglas, J. Naughton, and I. T. Rinov. A general technique for querying XML documents using a relational database system. In *SIGMOD Record*, pages 20–26, 2001.

[55] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *The VLDB Journal*, pages 302–314, 1999.

[56] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 422–432, 1997.

[57] Y. Tao and D. Papadias. MV3R-Tree: A spatio-temporal access method for timestamp and interval queries. In *The VLDB Journal*, pages 431–440, 2001.

[58] Y. Tao, J. Sun, and D. Papadias. Selectivity Estimation for Predictive Spatio-Temporal Queries. In *ICDE*, 2003.

[59] J. S. Vitter, M. Wang, and B. Iyer. Data cube approximation and histograms via wavelets. In *Proceedings of Seventh International Conference on Information and Knowledge Management*, pages 96–104, 1998.

[60] Y. Wang and Y. Chen. Materialized Views for Layer-Relational Structured Database and Incremental maintenance. *Journal of Huazhong University of Science and Technology*, 10:75–83, 1999.

[61] G. Wiederhold. Mediators in the architecture of future information systems. In M. N. Huhns and M. P. Singh, editors, *Readings in Agents*, pages 185–196. Morgan Kaufmann, San Francisco, CA, USA, 1997.

[62] M. Worboys. *GIS: A Computing Perspective*. Taylor & Francis, 1995.

[63] 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.