TEMPORAL AND VIDEO CONSTRAINT DATABASES

by

Rui 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 Z. Revesz

Lincoln, Nebraska

December, 2000

TEMPORAL AND VIDEO CONSTRAINT DATABASES

Rui Chen, Ph.D.

University of Nebraska, 2000

Adviser: Peter Z. Revesz

This dissertation proposes a general, flexible and reusable software architecture for constraint database systems. Our architecture contains several independent and coherent modules dealing with approximation, update, data representation, query evaluation, visualization and export conversion. We give a high-level description of these modules and their components, which can be modified and reused several different ways in building other systems. We implement a constraint database system based on our proposed architecture, TAQS, which is built on a spatiotemporal constraint database with linear constraints.

We propose an $O(n)$ time piecewise linear approximation algorithm to approximate the temporal data into a compact constraint database representation. We also approximate the spatial data by using TIN transformation, and represent it in constraint databases. Experiments show that the piecewise linear approximation provides a significant data reduction, and high coefficient correlation between the original data and its approximation.

We describe the queries based on the approximation, including simple algebraic queries, similarity queries and GIS-based queries. We evaluate the performance of the queries by using precision and recall parameters. Since the approximate database is much smaller than the original, the query evaluation becomes much faster while keeping very high precision and recall.

We also describe the update of the approximate temporal data and spatial data. The data is visualized by color bands display which yields static images and isometric color animation which yields a sequence of video clips for a series of spatiotemporal data.

ACKNOWLEDGMENTS

I would like to express my sincerest gratitude to my advisor, Professor Peter Z. Revesz, for introducing me to this exciting research area. He consistently guided me and gave me a great source of inspiration. I would also like to thank the members of my supervisory committee for their time and commitment. These committee members are: Professor Hong Jiang, Professor Spyros Magliveras and Professor Ram Narayanan.

I would like to thank my colleague Min Ouyang for his help in my research work. I would also like to take this opportunity to thank my friends who made me enjoy this period with so much fun.

I dedicate this dissertation to my parents for their continuous support and encouragement and for their unflinching faith in my abilities. Without their unselfish love and affection, I would not have finished this dissertation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Constraint databases, introduced in [16], are useful in geographic information systems, computer-aided design and other spatiotemporal applications. Constraint databases also provide a compact approximate representation of temporal data, which would require too much storage space in relational databases.

Because of their advantages, there is an increasing number of implemented constraint database systems such as CCUBE [3]. The architecture of these constraint database systems is very similar. However, they all contain several special features which never occur in relational and object-oriented database system design. These features may include modules for constraint representation, data approximation and special data visualization.

The goal of this dissertation is to help design a temporal and video constraint database, which can deal with temporal data and even spatiotemporal data. It can also evaluate queries and output the query results as static images or a sequence of video clips.

We propose a general, flexible and reusable software architecture for constraint database systems. Our architecture contains several independent and coherent modules dealing with approximation, update, data representation, query evaluation, visualization and export conversion. We give a high-level description of these modules and their components, which can be modified and reused several different ways in building other systems. We implement a prototype system, TAQS (short for *T*hree-dimensional *A*nimation and *Q*uery *S*ystem), which has several special design features over other general database systems. TAQS is a constraint database system that implements a spatiotemporal database with linear constraints and visualizes query results with color bands display and isometric color animation. The animation can be viewed as a sequence of video clips.

We mainly consider the time series data since it occurs in many applications. For example, a time series could be a sequence of data points to represent temperature or precipitation for a given location as it changes over time. Since a fine granularity of time may be needed, the traditional representation of time series data as a set of data points requires too much computer storage space and allows only inefficient data retrieval and querying.

We also study the approximation of spatial data. We propose a method to transform the spatial data into a constraint database representation. We use the Triangulated Irregular Networks (TIN) to transform a set of discrete spatial data points to be represented as a continuous surface.

The operations to approximate the temporal data or spatial data into constraint databases are performed at the time of the data entry; that is, the data is stored as a constraint database [16, 25, 38], where the constraints are parametric functions of time or space that interpolate the data. This approach is advantageous because it is possible to build powerful database systems (for example, CCUBE [3], DEDALE [14] and MLPQ [34]) that can be queried by standard relational database query languages. This enables a potentially much wider range of users to use the database.

Applications of constraint database systems were, until now, severely limited to a few well-understood areas of constraint representation. One such example is GIS, where convex polygonal areas were represented as conjunctions of linear inequality, i.e., half-plane intersection, constraints. Our work on interpolation functions as a natural source of constraint data opens up a range of uses of constraint databases beside these narrow focus applications.

It is very important to present the data to a user in a form that is easily understandable. Many current constraint database systems have a poor graphical user interface. MLPQ/GIS [17] probably has the most advanced user interface that allows a number of iconic queries, including the option to ask the system to show an animation of a 2-D object (a moving polygon). We describe an advanced GIS-oriented user interface that can animate various spatiotemporal data. Such an animation has a potential to reveal many interesting features to a user that would be hard or impossible to notice otherwise.

1.1 Outline of the Dissertation

The rest of the dissertation is organized as follows. Chapter 2 reviews the basic concepts related to our work, including the relational databases, constraint databases, time series data, geographic information systems, and video databases.

Chapter 3 proposes a general constraint database architecture, describes each module in much detail and introduces the TAQS system implemented based on this architecture.

Chapter 4 provides the piecewise linear approximation algorithm to transform a time series into a constraint database. It also shows the experimental results of the approximation.

Chapter 5 introduces the approximation for the spatial data. Here it uses TIN to transform the discrete spatial points into a continuous surface and represent the spatial data in constraint databases.

Chapter 6 describes the queries on the approximate constraint relations. It also introduces the queries on GIS applications, and similarity queries. Finally, it shows that the evaluation on the approximate constraint relations can yield a rather accurate query output.

Chapter 7 discusses the update operations on the approximate temporal data and spatial data.

Chapter 8 describes two visualization methods: color bands display and isometric color animation. The first method is to display the static image of the query output. The second method is to show the spatiotemporal query output with animations.

Finally, Chapter 9 conclude the dissertation and list some directions for future work.

Chapter 2

Review of Basic Concepts

This chapter reviews some basic concepts which are related to our work. Along with the development of the database techniques, the database systems are evolved in several striking steps.

The primitive database management systems were file management systems that realized some database functions by executing the defined subroutines for the data. The underlying data model, Network and Hierarchical Model, provides users with a network view of databases, including record types and one-to-many relationships among the record types. The network model allows a record type to be involved in more than one relationship, while the hierarchical data model allows a record type to be involved in only one relationship as a child.

The next type of database was the relational data model, which provides more flexible and powerful organization of data to generate very large databases. The relational query languages, for instance, SQL (Structure Query Language) are declarative query languages; that is, users just tell the system what they want to get from the database, and the remaining thing is finished by the system automatically. So, it is very easy to learn and use.

There were also several proposals for database models. Most of them were not implemented for commercial use. One of the very important proposals was the semantic data model, whose primary use is in database design phase.

Another database model is extended- and object-relational model, which extends the relational model by incorporating some object-oriented features. The extensions are involved in data representation, object identity, operations, programming interface, and so on.

Nested relational model is another extension of the relational data model. It allows the values of attributes to be relations, hence it can build up a tree-structured object space. In some cases, this model is more natural to represent the relationships of objects.

In the late 80's, object-oriented (O_2) databases came into being. The relationships among objects are represented by the definitions of those objects themselves. In some sense using this kind of representation is much easier to describe our real world.

Along with the requirement of the use of multimedia information, databases started to support imaging and multimedia data. By extending current database systems, they have some ability to provide services to users with multimedia information. Since multimedia data, especially video data, has some special characteristics compared with other kinds of data types, more tasks associated with it need to be done.

Now, constraint databases attract the attention of more and more researchers in this research area. Constraint databases can be applied to geographic information systems, computer-aided design and other spatiotemporal applications.

The following are structured as follows. Section 2.1 gives a brief review of the relational databases. Section 2.2 describes the constraint databases. Section 2.3 introduces the time series data. Section 2.4 gives an introduction of geographic information systems. Finally, in Section 2.5 briefly describes several video database models and systems.

2.1 Relational Databases

Relational databases are mainly used as commercial data-processing systems. A relational database consists of a collection of tables, each of which is assigned a unique name as the table name. Each table has a structure of an array, in which each row is a set of related values, called a record. A row in a table represents their relationship. The following is an example of the table in a relational database.

Example 2.1.1 Suppose there are some courses in a department stored in a relational table. A course has several attributes, which are callno, course name and credit. The table is shown in Table 2.1.

callno	**course_name**	**credit**
813	Database Systems	3
930	Computer Architecture	3
977	Cryptography	3

Table 2.1: The $Course$ Table

Table 2.1 stores three courses, which are Database Systems, Computer Architecture and Cryptography. Their call numbers are 813, 930 and 977, respectively, and all of them are 3 credits.

There are several query languages to retrieve the data from a relational database, such as relational algebra, SQL, Datalog and so on. A relational algebra query consists of a set of operations that take one or two relations as input and produce a new relation as their results. The fundamental operations are *select*, *project*, *union*, *product*, and so on. Another query language, SQL, has established itself as the standard relational-database language. The basic Structure of an SQL query consists of three clauses: *select*, *from* and *where*. The *select* clause is used to list the attributes which the users are interested in. The *from* clause is used to tell the system which tables the data is retrieved from. The *where* clause is used to specify the query condition to limit the query result. SQL is a procedural language. However, Datalog is a declarative query language, which is based on the logic-programming language Prolog. A Datalog program consists of a set of rules.

Each rule is defined by several conjunctive expressions. The head of a rule is the output relation, while the body is the input relations and conditions. If the body of a rule is true by instantiating the variables with concrete values, the head of the rule is true with those corresponding values.

Example 2.1.2 Given the Table *Course* in Example 2.1.1, find the course name with the callno 813. The following are queries using relational algebra, SQL and Datalog query languages, respectively.

Relational algebra query:

$$\Pi_{course\_name}(\sigma_{callno=813}(course))$$

SQL query:

Select $course\_name$

from $course$

where $callno = 813$;

Datalog query:

$$course\_813(course\_name)\ :-\ course(callno, course\_name, credit),\ callno = 813.$$

2.2 Constraint Databases

Besides the relational databases, there is another type of database systems, constraint databases, which stimulate more and more database researchers to devote themselves to such a field. Constraint databases are built on the constraint data model which can represent an infinite data set with a finite representation. In the constraint data model, each attribute is associated with an attribute variable and the value of the attributes in a relation is specified implicitly using constraints.

A constraint database is a finite set of constraint relations, each of which is a finite set of constraint tuples, where each tuple is a conjunction of atomic constraints using the same set of attribute variables. The following is an example of a relation in a constraint database.

Example 2.2.1 Suppose there is a road, a pond and a park in some place whose locations are shown in Figure 2.1. Their constraint representation is shown in Table 2.2.

id	**x**	**y**	
road	x	y	$5x - 11y = -18.$
pond	x	y	$2x - 5y \leq -22,\ 3x + y \leq 35,\ 5x - 4y \geq -4.$
park	x	y	$x \geq 7,\ x \leq 11,\ y \geq 1,\ y \leq 4.$

Table 2.2: The Constraint Relation for A Place

2.3 Time Series Data

A time series [24] is a set of time-stamped data entries. A time series allows a natural association of data collected over intervals of time. For example, summaries of stock market trading or banking transactions are typically collected daily, and

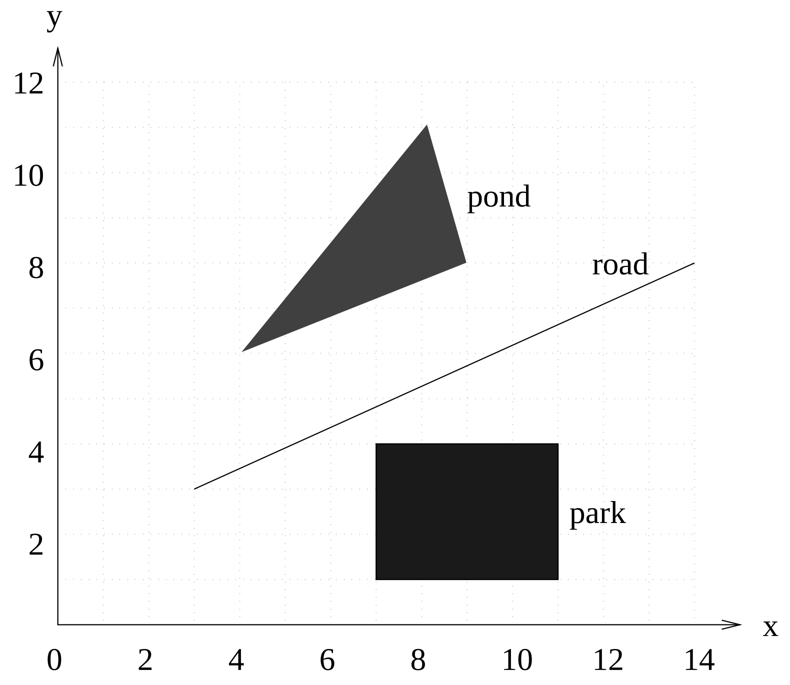

Figure 2.1: The Map of a Place

are naturally modeled with a time series. A time series can be regular or irregular, depending on whether or not the time series has an associated calendar.

A regular time series has an associated calendar. In a regular time series, data arrives predictably at predefined intervals. For example, daily summaries of stock market data form regular time series, and such time series might include the set of trade volumes and opening, high, low, and closing prices for a stock.

An irregular time series does not have an associated calendar. Often, irregular time series are data-driven, where unpredictable bursts of data arrive at unspecified points in time or most time-stamps cannot be characterized by a repeating pattern. For example, account deposits and withdrawals from a bank automated teller machine (ATM) form an irregular time series. An irregular time series may have long periods with no data or short periods with bursts of data.

For time series data, there are some methods to reduce the storage space. For example, we can use a polynomial function to interpolate these data within some specified error tolerance. In some cases, it can achieve great data reduction. However, there are also some disadvantages. For instance, when the data is retrieved from the polynomial function, it needs more complex computation.

In the area of cryptographic research, the linear complexity[20] of a given sequence is defined by the length of the shortest linear feedback shift register [21] which generates the sequence followed by some arbitrary sequence. It means that is the lowest polynomial function to generate the given sequence by iterations. The polynomial function of a sequence with the linear complexity k would have the form:

$$x_i = \prod_{j=i-k}^{i-1} (c_j x_j).$$

Example 2.3.1 Given a sequence of binary numbers $S = 01010000010110010011$ in the field F_2, denoted by $\{x_i \mid 0 \leq i \leq 19\}$. Find the linear complexity of this sequence.

The linear complexity of this sequence is 10, and the polynomial function is:

$$x_i = x_{i-10} + x_{i-9} + x_{i-8} + x_{i-7} + x_{i-6} + x_{i-5} + x_{i-2},\ 10 \leq i \leq 19.$$

By using the above polynomial function, we can generate the remaining sequence $[x_{10} .. x_{19}]$. For instance, we can get x_{14} by using the function:

$$\begin{aligned} x_{14} &= x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_1 2 \\ &= 0 + 0 + 0 + 0 + 0 + 1 + 1 \\ &= 0 \pmod 2. \end{aligned}$$

Note that if there is a very long, even infinite sequence which can be generated by a polynomial function, using the polynomial function can yield a great data reduction. But for the practical measurements, such as temperature or precipitation, it is not possible to have such kind of sequence. And this method is not efficient in retrieving arbitrary elements of the sequence, not easy to update the data, and there is no interpolation for the given data.

2.4 Geographic Information Systems

Geographic information systems (GIS) [36, 35] technology can be used for scientific investigations, resource management, and development planning. For example, a GIS might allow emergency planners to easily calculate emergency response times in the event of a natural disaster, or a GIS might be used to find wetlands that need protection from pollution.

In the strictest sense, a GIS is a computer system capable of assembling, storing, manipulating, and displaying geographically referenced information , i.e. data identified according to their locations. Practitioners also regard the total GIS as including operating personnel and the data that go into the system.

A GIS can be used to emphasize the spatial relationships among the objects being mapped. While a computer-aided mapping system may represent a road simply as a line, a GIS may also recognize that road as the border between wetland and urban development, or as the link between two streets.

With a GIS you can "point" at a location, object, or area on the screen and retrieve recorded information about it from off-screen files. Using scanned aerial photographs as a visual guide, you can ask a GIS about the geology or hydrology of the area or even about how close a swamp is to the end of a street. This kind of analytic function allows you to draw conclusions about the swamp's environmental sensitivity.

Traditional maps are abstractions of the real world, a sampling of important elements portrayed on a sheet of paper with symbols to represent physical objects. People who use maps must interpret these symbols. Topographic maps show the shape of land surface with contour lines. The actual shape of the land can be seen only in the mind's eye. Graphic display techniques in GIS's make relationships among map elements visible, heightening one's ability to extract and analyze information.

2.5 Video Databases

The use of multimedia is an irreversible trend in computing since it enhances the overall quality and quantity of information. Therefore, more and more information systems and applications are providing the support for multimedia information. Among various multimedia data, such as text, static images, audio, 2D or 3D pictures, and video data, video is the most important media for visual representation, making dramatic improvements to the whole human-computer interaction.

A video database is a video information management system, the integration of representation, storage, retrieval, processing and transmission of video data. Since a video database is also a special type of database, it is very similar, in some aspects, to other databases. For example, both video databases and relational databases need to deal with queries and indexing of stored data in the databases for speeding up the information retrieval. However, the representation and querying for video data are quite different from those of other databases.

The following are some characteristics [26, 18] of video database systems, a few of which may have a great influence on the design and implementation of video database systems.

Large sizes: This property of video objects requires that video database systems could have the ability to store and retrieve large amount of data. Also, the communication cost for the transmission need to be considered seriously.

Real-time nature: This property needs high-speed retrieval of video data and low communication cost for transmission.

Raw nature of video data: Contents of video data are binary in nature. Therefore, video database systems should derive and store some extra necessary information associated with the video for the sake of retrieval from the databases.

Spatial data types and queries: Some objects in videos have spatial relationships with respect to one another. Often users want to query this kind of information to make some analysis.

Interactive querying, relevance feedback and refining: Since it is very hard to be highly accurate in retrieving video information from video database systems, the output may have several alternatives related to the desired result. Users can choose one as the best. If there is no such desired result, users could send some feedback and then refine the query.

Automatic feature extraction and indexing: Conventional databases themselves have the explicit description of the stored data, while in video databases there is no such description. When there are huge amount of video data to be input into databases, the automatic extraction and indexing techniques are highly desirable.

There are several video database models and implemented video database systems. For example, there are video database models such as the Extension of

Relational Database for Video Data Model [18], Parametric Rectangle Constraint Data Model [4], Video Object Data Model [13], Object Composition and Playback Models [12] and Constraint Video Data Model [9]; and video database systems such as QBIC [11], PReSTO [4], STORM [23] and OVID [13] systems.

Chapter 3

The Architecture of Constraint Database Systems

In this chapter we propose a general, flexible and reusable software architecture for constraint database systems. The presentation below is based on our work in [32].

The software architecture includes several relatively independent and coherent modules dealing with approximation, update, data representation, query evaluation, visualization and export conversion. We describe these modules in general and illustrate them with our recently implemented TAQS constraint database system.

Constraint databases, introduced in [16], are useful in geographic information systems, computer-aided design and other spatiotemporal applications. Constraint databases also provide a compact approximate representation of temporal data which would require too much storage space in relational databases.

Because of their advantages, there is an increasing number of implemented constraint database systems such as CCUBE [3]. The architecture of these constraint database systems is very similar. On the other hand, they all contain several special features which never occur in relational and object-oriented database system design. These features may include modules for constraint representation, data approximation and special data visualization.

TAQS is a constraint database system that implements a spatiotemporal database with linear constraints and visualizes query results by three-dimensional animation, including volume animation and isometric color animation.

We outline the proposed software architecture in Section 3.1. We describe the designed modules in Section 3.2.

3.1 Overview of the Constraint Database Architecture

We propose the software architecture for constraint database systems shown in Figure 3.1. As can be seen, the architecture consists of six main modules. These modules and their roles in the architecture can be described as follows.

Data Representation: The data representation module is responsible for the internal representation of constraint databases and communicates with the external constraint database storage structures. If it cannot find some needed constraint relation in the constraint database store, then it calls the approximation module that searches its relational database store and provides a converted constraint re-

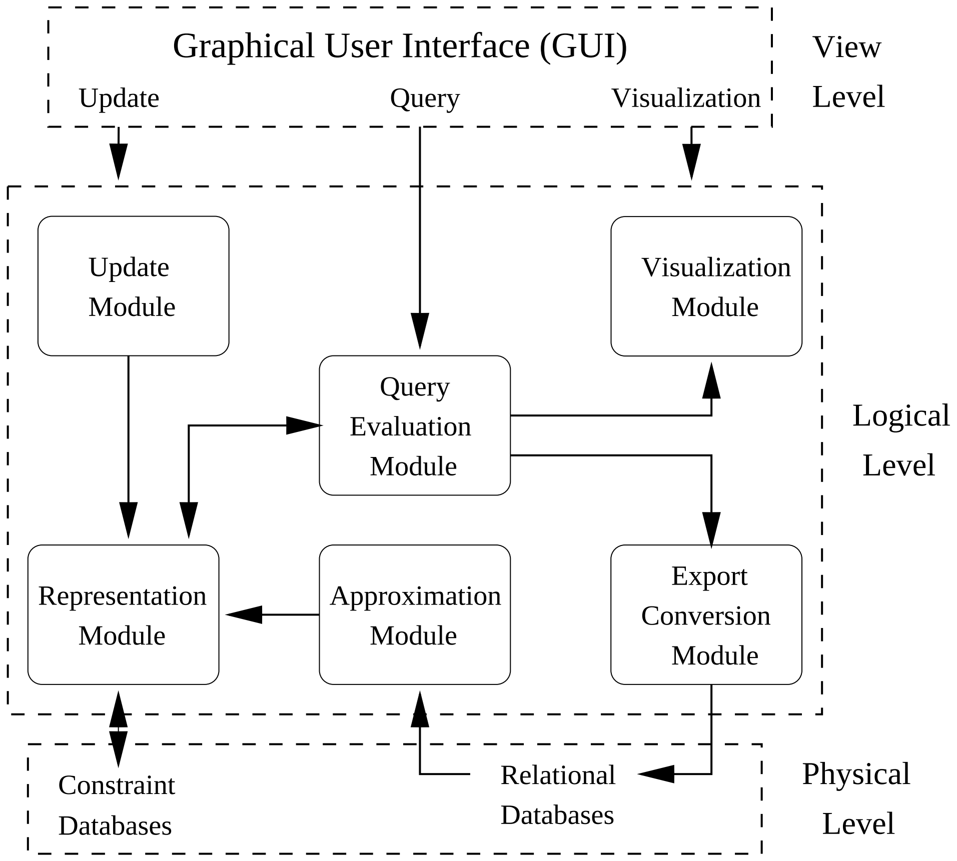

Figure 3.1: Constraint Database Architecture

lation. The data representation module provides output to the query evaluation module.

Query Evaluation: The query evaluation module is used to evaluate the queries provided by the user via a graphical user interface (GUI). There are several kinds of constraint query languages, such as relational algebra, SQL, Datalog and their extensions. In addition, there are special commands available as either a part of the query languages or special iconic operators that are useful for data visualization or exporting.

The output of this module is another constraint relation, which may be added to the database, therefore it may be passed to the data representation module. Alternatively, it may be a relation that is passed to the visualization module, if the query involves query visualization operators. Finally, it could also pass the data to the export conversion module, if the query asks for exporting to a relational database.

Visualization: The visualization module shows on the GUI the data that the user asks to see. The visualization module should provide a large number of available options. One of the interesting options that is special to spatiotemporal applications is animation of three dimensional spatiotemporal data. The input of this module is from the query evaluation module and the output is to the GUI.

Approximation: This is an optional module that can be used to convert from relational databases to constraint databases. Typically, the conversion involves a significant data reduction and approximation of the original relational data. The input is from relational database storage, and the output is to the data representation module.

Update: This module is responsible for the update of the constraint relations as requested by the user. There are several update languages which the user may use to express updates or use iconic operators. There also may be several ways to implement the update requests.

Export Conversion: This module provides conversion from constraint databases to relational databases. Note that this option may not be always available, because constraint databases are sometimes equivalent to infinite relational databases.

We discussed each of the above modules in a general way. In the following section, we expand on our discussion and illustrate each module using the TAQS constraint database system as an example. In the following chapters, we describe the approximation, query, update and visualization in more detail.

3.2 Descriptions of the Modules

This section describes all of the six modules in our proposed constraint database architecture.

3.2.1 Data Representation Module

We start with the description of this module because the data model is at the heart of any database system. In the constraint data model, each constraint database consists of a set of constraint tables, and each constraint table consists of a set of constraint tuples [16].

There are many types of constraint databases depending on the type of constraint domains and constraints used. For example, we may use real numbers as the domain and polynomial inequality constraints [38], set order constraints [25], or Boolean equality constraints [28]. For some other types of constraints see the survey [27].

The TAQS system, like the CCUBE [3], the DEDALE [14] and the MLPQ [34, 17] systems, uses as the internal representation a set of *rational linear constraints*. In fact, we reused within the TAQS system some of the data representation modules of the MLPQ system, which was implemented earlier at the University of Nebraska-Lincoln. We will describe the rational linear constraint database representation with an example.

Example 3.2.1 Suppose that we are representing the temperature at several weather stations. The weather changes piecewise linearly at each weather station. For example, as can be seen from the table below, the temperature at weather station 1 is $2t + 75$ Fahrenheit degrees at any time before one hour (counting from some given zero hour), then $9t + 68$ between one and two hours, and finally $2t + 82$ after two hours. The representation for this is shown in Table 3.1.

Note that in the table we use only linear equality constraints, and inequality constraints. In general, we allow in the TAQS system linear inequality constraints of the form:

$$a_1x_1 + \ldots + a_nx_n \ \theta \ b$$

where each a_i and b is a rational constant and each x_i is a variable, and θ is one

SN	**Temp**	**t**	
sn	$temp$	t	$sn = 1,\ temp = 2t + 75,\ t \geq 0,\ t \leq 1.$
sn	$temp$	t	$sn = 1,\ temp = 9t + 68,\ t > 1,\ t \leq 2.$
sn	$temp$	t	$sn = 1,\ temp = 2t + 82,\ t > 2,\ t \leq 4.$
sn	$temp$	t	$sn = 2,\ temp = 3.75t + 70,\ t \geq 0,\ t \leq 4.$
sn	$temp$	t	$sn = 3,\ temp = 6t + 80,\ t \geq 0,\ t \leq 1.$
sn	$temp$	t	$sn = 3,\ temp = -2.67t + 88.67,\ t > 1,\ t \leq 4.$
sn	$temp$	t	$sn = 4,\ temp = -2.25t + 85,\ t \geq 0,\ t \leq 4.$

Table 3.1: The *Temperature* Relation

of the comparison operators $<, >, \leq, \geq, =$.

The meaning of a constraint database is the set of regular relational tuples that can be obtained by substituting constants of the allowed domain (in this case the rational numbers) into the variables of the constraints in each constraint tuple. If all the constraints are true, then the tuple after substitution is also true. Hence, for example, the tuple $(1, 77, 1)$ is true, because if we substitute into the first constraint tuple the value 1 for s and t and 77 for $temp$, then the constraint $sn = 1$, $temp = 2t + 75$, and $t \leq 1$ are all satisfied.

3.2.2 Query Evaluation Module

The query evaluation module can be broken up into several smaller modules. These include a number of modules. First, the query representation module, which accepts the user's request from the GUI for a query. Second, the security module, which checks that the user is accessing only relations that he/she is allowed to use. Third, the evaluation module. The last one may have as its component in a multi-user, distributed environment a transactions module, which ensures that each operation of each user is executed as a whole, not only in parts.

Constraint databases can be queried by a number of popular query languages, such as relational algebra, SQL and Datalog, and their extensions [37, 2]. Here we discuss Datalog queries on constraint databases, because they are a powerful query language and illustrate well the idea of constraint query evaluation. We also omit discussion of transaction control.

Datalog is a declarative query language which is based on the logic-programming language Prolog. A Datalog program consists of a set of rules. Each rule is defined by several conjunctive expressions. The head of a rule is the output relation, while the body is the input relations and conditions. If the body of a rule is true by instantiating the variables with concrete values, the head of the rule is true with those corresponding values. Datalog query language can easily execute the functions of the algebraic query operators, such as select, project, union, set difference, Cartesian product, rename, natural join, division and assignment.

Our TAQS system provides all these operators as well as some aggregate operators, such as minimum (Min) and maximum (Max) of a linear objective function. For the implementation of these last two operators, we used a linear programming software package. TAQS calls the linear programming package and outputs a minimum or maximum value for each linear constraint tuple. On the other hand, other kinds of queries could also be accepted if a transformation subroutine for query expressions is implemented.

3.2.3 Visualization Module

The visualization module can be broken up into smaller modules that provide different types of visualization. TAQS system provides several types of visualizations including color bands display and isometric color animation.

The isometric color animation is like a video provided on the GUI. Constraint databases are particularly well-adapted to provide animations because they represent an infinite sequence of data, hence they allow any granularity for the animation without requiring a huge data storage.

3.2.4 Approximation Module

The approximation module can be used to convert data from relational to constraint databases. We are not interested here in trivial conversions where one relational tuple is converted into one constraint tuple. Instead, we are interested in conversions that take full advantage of constraints to represent data approximately but very compactly. In such approximations, dozens or may be hundreds of relational tuples could be represented by a single constraint tuple.

We propose a piecewise linear approximation method to approximate a time series into piecewise linear function in Chapter 4. Then the resulting piecewise linear function can be represented in a constraint database as a constraint relation.

We also propose a method to approximate a set of spatial data points and represent them by constraint databases in Chapter 5.

3.2.5 Update Module

The update module is responsible for modifying the database as the user requests. Usually the query languages are augmented with special language constructs to express updates. The most important types of update operations are insertions and deletions of tuples in relational databases.

As for updating a piecewise linear function transformed from a time series, we need some special techniques to deal with that. We also consider the update on spatial data. Chapter 7 gives a detailed description of how to update approximated temporal data and spatial data.

3.2.6 Export Conversion Module

The export conversion module deals with the conversion from constraint databases to relational databases. Constraints in general cannot be converted to a relational database representation because that would require an infinite number of tuples. However, if there are finite set of time instances which are of interest, then we can generate from the constraint database a relational database that contains the values for the needed time instances. This conversion is provided in the TAQS system and is illustrated in an example below.

Example 3.2.2 Suppose that we are given the constraint database relation *Temperature* shown in Table 3.1 and we are interested in the temperatures at times 1 and 1.5. Then we can convert the constraint database to the relational database shown in Table 3.2.

SN	**Temp**	**t**
1	77.0	1.0
1	81.5	1.5
2	73.7	1.0
2	75.6	1.5
3	86.0	1.0
3	84.7	1.5
4	82.7	1.0
4	81.6	1.5

Table 3.2: The Conversion Result

Note that we may not obtain the above table from the corresponding relational database because that may not give the temperature for the time $t = 1.5$. By using the interpolation implicitly in the piecewise linear approximation that actually yields the constraint database in Table 3.1, we could access an approximate value of the temperature at any time instance between 1 and 4 including 1.5. The constraint database representation is better in this case because we do not know ahead what time instances the users may be interested in.

Chapter 4

Temporal Data Approximation

This chapter gives a piecewise linear approximation for temporal data. The presentation below is based on our work in [6, 5].

Section 4.1 introduces the piecewise linear approximation. Section 4.2 discuss how to store a piecewise linear function. Finally, Section 4.3 shows some experimental results of the piecewise linear approximation for a given data set.

4.1 The Piecewise Linear Approximation

There are many types of approximations and interpolations that could be used. Let us consider approximations for time series data. A time series S is a sequence of data points $(t_1, y_1), \cdots, (t_n, y_n)$ where the ts are monotone increasing time values. Note that in a time series the ts need not always increase uniformly with the same increment.

Example 4.1.1 The recording of temperature at a weather station is a time series. For example, for weather station 1 a time series may be $(0, 75)$, $(1, 77)$, $(2, 86)$, $(3, 87)$ and $(4, 90)$. This can be represented by the first five tuples of relation $Temperature$ as shown in Table 4.1. For other weather stations $(2$, 3, and $4)$ we can represent similarly their corresponding time series by adding more tuples to the $Temperature$ relation.

SN	**Temp**	**t**
1	75	0
1	77	1
1	86	2
1	87	3
1	90	4
2	70	0
2	72	1
2	75	2
2	80	3
2	85	4
3	80	0
3	86	1
3	81	2
3	80	3
3	78	4
4	85	0
4	83	1
4	81	2
4	78	3
4	76	4

Table 4.1: The $Temperature$ Relation

We can see that the $Temperature$ relation can get quite large. In the TAQS system we can compress this relation by using a *Piecewise Linear Approximation*

(PLA). A piecewise linear function is a continuous function that is the union of a set of linear functions whose domains are disjoint. For example, the union of the linear function $2t + 75$ when $t \leq 1$, the function $9t + 68$ when $1 < t \leq 2$, and the function $2t + 82$ when $t > 2$ is a piecewise linear function.

Given a time series S as above and an error tolerance constant Ψ, the piecewise linear approximation problem is the problem of finding a piecewise linear function f such that:

$$|f(t_i) - y_i| \leq \Psi \text{ for each } (t_i, y_i) \in S. \tag{4.1}$$

In general, the smaller Ψ is the more pieces the piecewise linear function will contain. Below we present a piecewise linear approximation algorithm that runs in $O(n)$ worst case time. First let us define some terms to be used in explaining the algorithm.

Definition 4.1.1 On the time interval $[t_b,\ t_e]$, let us define $Y_{b,e}(t)$ to be the linear function:

$$Y_{b,e}(t) = \frac{y_e - y_b}{t_e - t_b}(t - t_b) + y_b \tag{4.2}$$

Note: The linear function $Y_{b,e}(t)$ can be drawn as a line segment with end-points (t_b, y_b) and (t_e, y_e).

Definition 4.1.2 Given two points (t_b, y_b) and (t_e, y_e) where $(b < e)$, and the maximum approximation error threshold is Ψ, then the *lower line* for these two points is the line which passes through the points (t_b, y_b) and $(t_e, y_e - \Psi)$, denoted as:

$$L_{b,e}(t) = \frac{(y_e - \Psi) - y_b}{t_e - t_b}(t - t_b) + y_b \tag{4.3}$$

and the *upper line* for these two points is the line which passes through the points (t_b, y_b) and $(t_e, y_e + \Psi)$, denoted as:

$$U_{b,e}(t) = \frac{(y_e + \Psi) - y_b}{t_e - t_b}(t - t_b) + y_b \tag{4.4}$$

Note: The lower line $L_{b,e}(t)$ can be drawn as a line that has one endpoint (t_b, y_b) and passes through $(t_e, y_e - \Psi)$. Similarly, the upper line $U_{b,e}(t)$ can be drawn as a line that has the same endpoint (t_b, y_b) and passes through $(t_e, y_e + \Psi)$.

Our piecewise linear approximation algorithm is shown below. The algorithm initializes the values of b and e to be 1 and 2. Note that the line segment with endpoints $(t_1, y_1), (t_2, y_2)$ is the smallest possible first piece of the piecewise linear function.

In the *if* clause of the *while* loop, the algorithm creates a new piece $Y_{b,e}(t)$ if the sequence $(t_b, y_b), \ldots, (t_e, y_e)$ can be approximated but the one longer sequence $(t_b, y_b), \ldots, (t_{e+1}, y_{e+1})$ cannot be approximated as required by Formula (4.1).

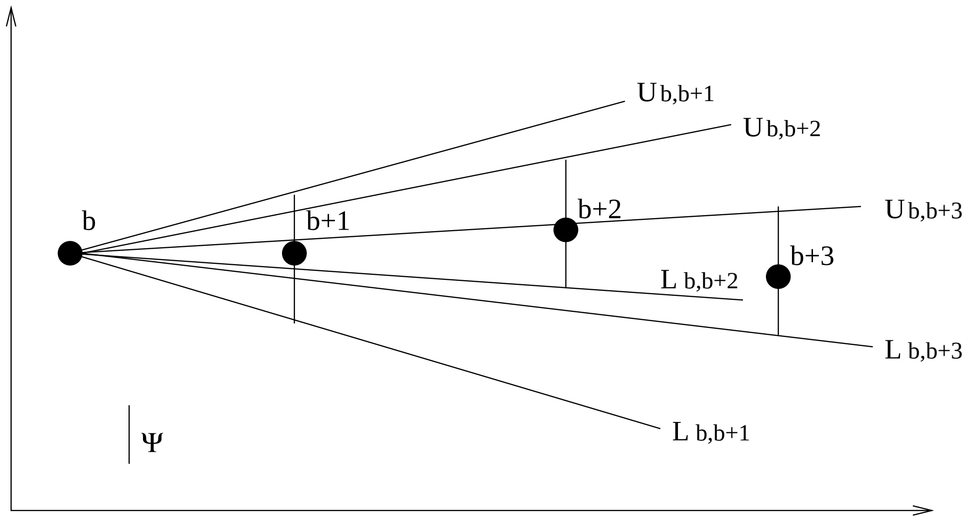

Figure 4.1: The Upper and Lower Lines

In the *else* clause of the *while* loop, the algorithm extends the current sequence by one point and if necessary tightens both the current lower and the current upper lines. The working of the *else* clause is illustrated in Figure 4.1. There we see three pairs of lower and upper lines. Suppose that we enter three times successively the *while* loop and each time execute the *else* clause. Then the lower line L will after the first iteration $L_{b,b+1}$, in the second $L_{b,b+2}$, and in the third $L_{b,b+2}$. Similarly, the upper line U will be after the first iteration $U_{b,b+1}$, in the second $U_{b,b+2}$, and in the third $U_{b,b+3}$. Note that we get the highest slope lower bound line and the smallest slope upper line as the final result for L and U, respectively.

PIECEWISE LINEAR APPROXIMATION ALGORITHM:

Input: A time series $S = (t_1, y_1), \ldots, (t_n, y_n)$ and n.

Ψ the maximum error threshold in the approximation.

Output: A piecewise linear function approximation of S.

Local Vars: The b and e are integer variables such that the sequence $(t_b, y_b), \ldots, (t_e, y_e)$ can be approximated by one piece.
L and U stand for the current lower and upper lines.

```
$b := 1$ and $e := b + 1$
$L := L_{b,e}$ and $U := U_{b,e}$

while $e < n$ do
    if $y_{e+1} < L(t_{e+1})$ or $y_{e+1} > U(t_{e+1})$ then
        Create a piece $Y_{b,e}$ defined by Formula (4.2)
        $b := e$ and $e := b + 1$.
        $L := L_{b,e}$ and $U := U_{b,e}$.
    else
        if $L_{b,e+1}(t_{e+1}) > L_{b,e}(t_{e+1})$ then
            $L := L_{b,e+1}$
        end-if

        if $U_{b,e+1}(t_{e+1}) < U_{b,e}(t_{e+1})$ then
            $U := U_{b,e+1}$
        end-if
        $e := e + 1$
    end-if
end-while

Create a piece $Y_{b,e}$ defined by Formula (4.2)
```

Example 4.1.2 Consider the time series $S = (0, 75),\ (1, 77),\ (2, 86),\ (3, 87),$ $(4, 90)$ from Example 4.1.1. This time series has five elements hence $n = 5$. If we call the piecewise linear approximation algorithm with S and n and $\Psi = 3$, then it will work as follows.

Initialization: The algorithm makes $b = 1,\ e = 2$, the lower line $L = L_{1,2} = y = -t + 75$, and the upper line $U = U_{1,2} = y = 5t + 75$.

First iteration: Since $e = 2 < n = 5$, the algorithm enters the *while* loop. Since $y_{e+1} = y_3 = 86 > U(t_{e+1}) = U(t_3) = U(2) = 85$, the algorithm also enters the *then* clause. There it creates a piece $Y_{1,2} = 2t + 75$, then updates $b = 2,\ e = 3$, the lower line $L = L_{2,3} = 6t + 71$ and $U = U_{2,3} = 12t + 65$.

Second iteration: Since $e = 3 < n = 5$, the algorithm enters again the *while* loop. Since $y_{e+1} = y_4 = 87 < L(t_{e+1}) = L(t_4) = L(3) = 89$, the algorithm enters the *then* clause. There it creates a piece $Y_{2,3} = 9t + 68$, then updates $b = 3$, $e = 4$, the lower line $L = L_{3,4} = -2t + 90$ and $U = U_{3,4} = 4t + 78$.

Third iteration: Since $e = 4 < n = 5$, the algorithm enters again the *while* loop. Since $y_{e+1} = y_5 = 90 < L(t_{e+1}) = L(t_5) = L(4) = 82$ is false and $y_{e+1} = y_5 = 90 > U(t_{e+1}) = U(t_5) = U(4) = 94$ is also false, the algorithm now enters the *else* clause. There since $L_{3,5}(t_{e+1}) = 0.5t_{e+1} + 85 = 87 > L_{3,4}(t_{e+1}) = 82$, the lower line is updated $L = 0.5t+85$. Also, since $U_{3,5}(t_{e+1}) = 3.5t_{e+1}+79 = 93 < U_{3,4}(t_{e+1}) = 94$, the upper line is updated $U = 3.5t + 79$. It also updates $e = 5$.

Last line: Finally, since $e = 5 < n = 5$ is false, the algorithm exists the *while* loop and in the last line of the program creates the piece $Y_{3,5} = 2t + 82$.

Note that the sequence of pieces is $2t + 75$ (from 0 to 1), $9t + 68$ (from 1 to 2), and $2t + 82$ (from 2 to 4).

The other constraint tuples can be obtained by calling the piecewise linear approximation algorithm once for each of the other time series data corresponding to the other weather stations. The resulting constraint relation is shown in Table 4.2.

Now we prove the correctness of the algorithm in the next theorem.

Theorem 4.1.1 The piecewise linear approximation algorithm is correct for any error tolerance value Ψ and time series S.

Proof: We have to show that for any S and Ψ the algorithm finds a piecewise

SN	**Temp**	**t**	
sn	$temp$	t	$sn = 1,\ temp = 2t + 75,\ t \geq 0,\ t \leq 1.$
sn	$temp$	t	$sn = 1,\ temp = 9t + 68,\ t > 1,\ t \leq 2.$
sn	$temp$	t	$sn = 1,\ temp = 2t + 82,\ t > 2,\ t \leq 4.$
sn	$temp$	t	$sn = 2,\ temp = 3.75t + 70,\ t \geq 0,\ t \leq 4.$
sn	$temp$	t	$sn = 3,\ temp = 6t + 80,\ t \geq 0,\ t \leq 1.$
sn	$temp$	t	$sn = 3,\ temp = -2.67t + 88.67,\ t > 1,\ t \leq 4.$
sn	$temp$	t	$sn = 4,\ temp = -2.25t + 85,\ t \geq 0,\ t \leq 4.$

Table 4.2: The Constraint Relation *Temperature*

linear function f that satisfies Formula 4.1. To show that, it is enough to prove the following invariant condition for each entry of the *while* loop, by induction on the number of entries:

Invariant Condition (1): The line segment $Y_{b,e}(t)$ on the time iterval $[t_b, t_e]$ satisfies Formula 4.1.

For the first entry of the *while* loop, the initialization implies the line segment $Y_{1,2}$. Clearly, that satisfies Formula 4.1 because it gives zero error for both t_1 and t_2.

Now we assume that invariant condition (1) holds before some entry of the *while* loop, and then we show that it will also hold before the next entry or exit from the *while* loop.

There are two basic cases. The first case is when we enter the *if* clause of the *while* loop. In that case we add the current $Y_{b,e}(t)$ to the piecewise linear function, and that is correct. Then we reset the values of b and e to be two consequtive points. Therefore, similarly to the initialization, these also will satisfy Formula 4.1.

For the *else* clause in the *while* loop, we prove the following condition that holds before entering the *else* clause.

Invariant Condition (2): If $f(t)$ is a line that passes through (t_b, y_b) and is between L and U (i.e., has a higher slope than L but a smaller slope than U has), then $|f(t_i) - y_i| \leq \Psi$ for each $b \leq i \leq e$.

We fix the value of b and prove the condition by induction on $e - b$. (This corresponds to repeatedly executing the *else* clause.)

If $e = b + 1$, then condition (2) obviously holds.

Now assume that the condition holds for some $k = e - b$. Then we prove it for $k + 1$. Clearly k can increase only if e has increased by one since we fixed the value of b.

If we increased e since the last entry of the *else*, then we must have also updated L and U. Let the previous values be L' and U' and the new values be simply L and U. Since the slope of L is greater than or equal to that of L', and the slope of U is less than or equal to that of U', there can be only a smaller or equal region between L and U than between L' and U'.

Let $f(t)$ be any line that passes through (t_b, y_b) and is between L and U. Then by the above, $f(t)$ also is between L' and U'. Therefore, by the induction hypothesis $|f(t_i) - y_i| \leq \Psi$ holds for $b \leq i \leq e$. Also, by the definition of L and

U, we have that $f(t_e) - y_e| \leq \Psi$. This proves that invariant condition (2) must hold.

Finally, note that invariant condition (2) implies invariant condition (1). That is because if we are in the *else* clause, then we can choose for $f(t)$ the linear function $Y_{b,e}(t)$.

Finally, if we exit the while loop, then we get the last piece, which also must satisfy Formula 4.1 because none of our arguments above depended on the value of n. Hence if we had more values we could continue by entering again the *while* loop, that is, invariant condition (1) still must hold. ■

Next we analyze the computational complexity of our approximation algorithm.

Theorem 4.1.2 The computational complexity of the piecewise linear approximation algorithm is $O(n)$ time in the worst case where n is the number of points in the time series to be approximated.

Proof: There is only loop, the *while* loop in the piecewise linear approximation algorithm. The *while* loop is executed only at most $n-2$ times, because initially the value of e is two, then it is incremented by one in each iteration until $e = n$.

We also have to show that each iteration of the *while* loop takes only a constant time. Within the *while* loop the top level *if* statement has two clauses. The *then* clause takes a constant time, because there we only do a fixed number of comparison and assignment operations and add one piece to the piecewise linear function, which will be the output of the algorithm. By keeping the pieces in a linked list and a pointer to the end of the last list, we can do the addition in constant time. In the *else* clause we again do only fixed number of comparison and assignment operations.

Therefore, the worst case computational complexity of this algorithm is $O(n)$ time. ■

Remark: There are other advantages of the piecewise linear approximation algorithm beside data compression. First, we do interpolation as well as conversion of data. For example, this allows us to ask what was the temperture at time instances other than the original time instances when the original time series measurements were taken. Second, because the data is much smaller, querying and visualizations could be also done faster using the compressed data.

4.2 The Storage Representation of Piecewise Linear Function

There are several methods to store a piecewise linear function.

Method 1: One simple method is to store the endpoints of the pieces in a piecewise linear function. In some sense, this kind of representation is equivalent to storing some selected data points from the original time series only.

Example 4.2.1 Given the constraint relation *Temperature* shown in Table 4.2.

Using the above method will only store the data in Table 4.3.

SN	**Temp**	**t**
1	75	0
1	77	1
1	86	2
1	90	4
2	70	0
2	85	4
3	80	0
3	86	1
3	78	4
4	85	0
4	76	4

Table 4.3: The End Point of Each Piece

Suppose that the original time series contains n points and the algorithm returns a piecewise linear approximation with m pieces. Then the approximation can be represented by those points that begin or end a piece, that is, only $m+1$ points. Hence the piecewise linear approximation yields a degree of data compression that depends on the ratio $\frac{n}{m}$ which is the average number of original time series points spanned by each piece.

Method 2: Another kind of storage representation is to store the first start point, then for each piece it just stores its slope and ending time. When retrieving the data, from the first point and slope we can get the first piece and the endpoint of this piece which is also the startpoint of the next piece. Then we can get the next piece, and so on for other pieces. Compared with the first representation method, this one is a little discouraging since if we want to get some intermediate piece we need to scan the pieces from the first one.

Example 4.2.2 Given the constraint relation *Temperature* shown in Table 4.2. Using the second method will store the data in Table 4.4 and Table 4.5.

SN	**Temp**	**t**
1	75	0
2	70	0
3	80	0
4	85	0

Table 4.4: The Start Point for Each Weather Station

SN	**slope**	**t**
1	2	1
1	9	2
1	2	4
2	3.75	4
3	6	1
3	-2.67	4
4	-2.25	4

Table 4.5: The Slope and the End Time of Each Piece

4.3 Experimental Results

We used the temporal data set U.S. monthly precipitation for 96 years which contains $96 \times 6,726$ temporal data points, between the year 1990 and 1997 from $6,726$ weather stations throughout the continental United States [22]. The precipitation values ranged between 0 and $4,957$ with an average value of 295.91 and a standard deviation of 269.95.

We tested the transformation accuracy of our algorithm with different values of Ψ between 10 and 320. After the piecewise linear approximation function was found, we checked the differences between the value of the interpolation function and the original values. We ran separately for each weather station the transformation algorithm and made the correlation tests.

Table 4.6 shows the average number of generated pieces of the piecewise linear function among 96 weather stations and the transformation accuracy for different values of maximum error Ψ. The unit of Ψ is hundredths of inches for the precipitation.

Maximum Error Ψ	**10**	**20**	**40**	**80**	**160**	**320**
Average # of Pieces	89.03	84.29	76.09	63.22	45.72	25.30
Correlation Coefficient	0.9999	0.9999	0.9993	0.9956	0.9748	0.8775

Table 4.6: Experimental Results of Piecewise Linear Approximation

The results of the correlation coefficients show that the transformation is highly accurate when the Ψ is lower than the average value of the data. The number of pieces in the piecewise linear functions decreases as Ψ increases.

The maximum number of generated linear pieces for n data points is $n - 1$. The relationships between the percent of the number of linear pieces over $n - 1$ and the correlation coefficient are shown in Figure 4.2 when Ψ varies from 10 to 320.

Also, we can see that the piecewise linear function transformation has very high correlation with few number of linear pieces. We believe that this holds for any reasonable data set. Form the point view of the storage space, this property shows that the piecewise linear approximation provides a certain ability of data compression.

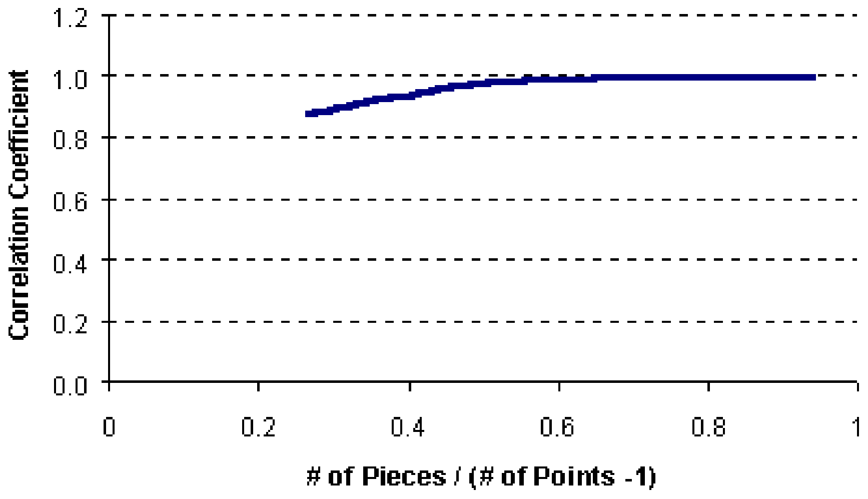

Figure 4.2: Maximum Error Ψ Varies from 10 to 320

Chapter 5

Spatial Data Approximation

This chapter deals with the approximation of the spatial data, and then represents it in constraint databases. Spatial data records the spatial information of the objects we are interested in. In our research, we consider two-dimensional spatial points, whose corresponding information could be any property related to that point, such as temperature, precipitation, elevation, and so on.

There are two kinds of spatial data models, namely the vector data model and the raster data model, that are used in Geographic Information Systems (GISs) [10, 35, 1]. Raster-based GISs are often used when continuous data are available, while vector-based are more suited than raster GISs when there are only discrete sampled data, such as temperature, precipitation and so on. For vector data model, triangles are the spatial framework, which is named Triangulated Irregular Networks (TIN).

Chomicki and Revesz [8, 29] and Grumbach et al. [15] describe algorithms to transform TIN data structure to constraint representation.

The plan of this chapter is as follows. In Section 5.1 we introduce the representation of the spatial data by using triangulated irregular networks. In Section 5.2 we present a transformation algorithm to transform a TIN to a constraint database based on the work [8, 29] and [15].

5.1 The TIN Representation of Spatial Data

This section shows the transformation of a set of spatial data into the TIN structure. A TIN makes it possible for a set of discrete data points to be represented as a continuous surface. The most popular software ARC/INFO supplies a very good function to generate the TIN structure by transforming some given sample points.

We use the ARC/INFO TIN generation module to create the TIN structure for a given set of points in the three dimensional space. TIN is a vector data structure based on two basic elements: sample points with their coordinates and a series of edges by joining these points to form triangles.

Example 5.1.1 Given four weather stations 1 to 4 located in $(10, 20)$, $(20, 40)$, $(50, 25)$ and $(30, 10)$ respectively as shown in Table 5.1, where SN stands for station number and X and Y the coordinates of the station locations.

We set the initial z value for each point as 0, then use the TIN transformation. The generated TIN for the weather stations is shown in Table 5.2.

Note that the third column in the section EDGES is a special parameter to

SN	X	Y
1	10	20
2	20	40
3	50	25
4	30	10

Table 5.1: The Locations of Weather Stations

NODES				
	1	10	20	0
	2	20	40	0
	3	50	25	0
	4	30	10	0
EDGES				
	1	1	2	0
	2	2	3	0
	3	3	4	0
	4	1	4	0
	5	2	4	1
TRIANGLES				
	1	1	4	5
	2	2	3	5
END				

Table 5.2: The TIN Structure for Weather Stations

denote whether the edge is an internal edge or a boundary edge. The graph to represent this TIN structure is shown in Figure 5.1.

So, we can get the node list $Node(id, x, y)$, the edge list $Edge(id, p_1, p_2, ifb)$, and the triangle list $Triangle(id, e_1, e_2, e_3)$, where x and y stand for the position of spatial points in a planar surface, ifb is 0 means this edge is a boundary edge or an internal edge otherwise, p_1, p_2 are the ids of the nodes corresponding to the edge, and e_1, e_2, e_3 are the ids of the edges corresponding to the triangle.

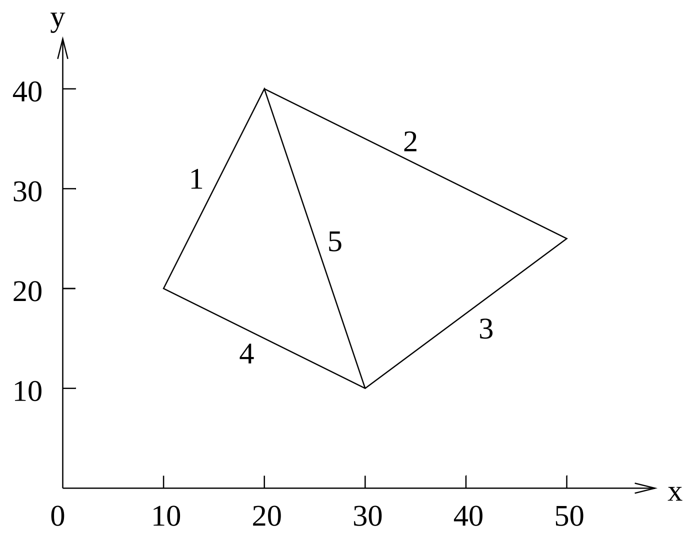

Figure 5.1: The TIN Graph of Weather Stations

Experiment: We did some empirical analysis for the transformation. We use United States weather stations [22] as the spatial data set in our experiment. The original data set includes more than $20,000$ weather stations.

First, we use the weather stations as the spatial points to generate the TIN structure. In the TIN transformation, the maximum and minimum X and Y values are computed and a bounding triangle is formed which contains all of the vertices to be triangulated. So, the TIN algorithm only generates convex triangles no matter how the real shape should be. In our implementation, we made some modifications for the TIN output by using a control variable *valid_edge_length* to control the edge length in the TIN graph. If one of the edges in a triangle is greater than *valid_edge_length*, then this triangle is not displayed. By doing so, we can get better TIN graph as shown in Figure 5.2, which is composed of $8,021$ points and

16,023 triangles generated by TIN.

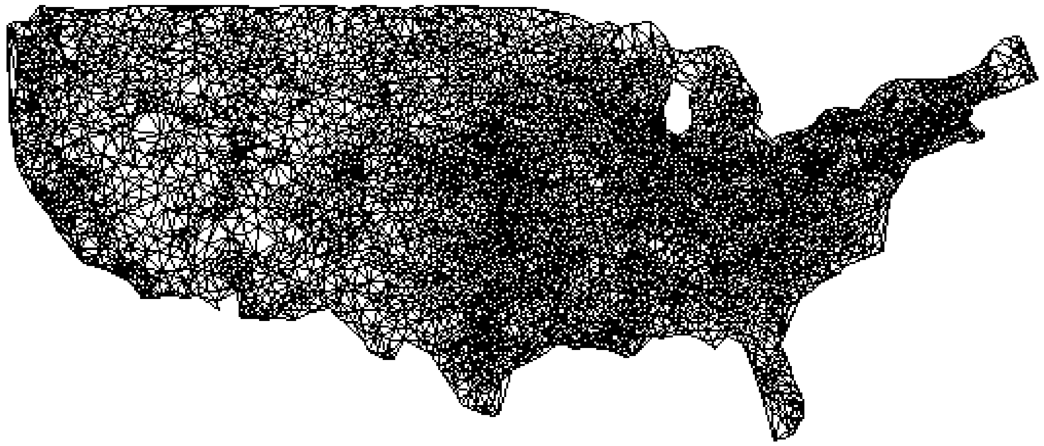

Figure 5.2: The TIN Graph for 8,021 Weather Stations

5.2 Transform TIN to Constraint Databases

The triangles in a TIN structure can be represented in constraint databases. Each triangle is represented by three equality constraints which compose a tuple in a constraint database.

First we define some basic terms to be used in the TIN transformation algorithm.

Definition 5.2.1 Give two spatial points $P_i(x_i, y_i)$ and $P_j(x_j, y_j)$, let us define $Y_{i,j}(x,\ \theta)$ to be the linear constraint:

$$Y_{i,j}(x,\ \theta) := \begin{cases} y\ \theta\ \frac{y_j - y_i}{x_j - x_i}(x - x_i) + y_i, & \text{if } x_i \neq x_j; \\ x\ \theta\ x_i, & \text{if } x_i = x_j. \end{cases} \tag{5.1}$$

where θ can be $\geq$, $\leq$ or $=$ symbols. We also denote the value of $Y_{i,j}(x,\ =)$ is the value of y if $x_i \neq x_j$ or x if $x_i = x_j$.

Lemma 5.2.1 Given three points $P_i(x_i, y_i)$, $P_j(x_j, y_j)$ and $P_k(x_k, y_k)$. The linear constraint $Y_{i,j}(x,\ \theta)$ contains the P_k if the following condition holds:

$$\begin{array}{ll} y_k\ \theta\ Y_{i,j}(x_k,\ =) & \text{if } x_i \neq x_j; \\ x_k\ \theta\ Y_{i,j}(x_k,\ =) & \text{if } x_i = x_j. \end{array} \tag{5.2}$$

Proof: First let us prove it holds when $x_i \neq x_j$. There are three cases: (1) θ is $\geq$; (2) θ is $\leq$; (3) θ is $=$.

Case 1: The linear constraint is $Y_{i,j}(x,\ \geq)$, that is, $y \geq \frac{y_j - y_i}{x_j - x_i}(x - x_i) + y_i$. The condition shows that $y_k \geq Y_{i,j}(x_k,\ =)$; that is, $y_k \geq \frac{y_j - y_i}{x_j - x_i}(x_k - x_i) + y_i$. So, the half surface defined by $Y_{i,j}(x,\ \geq)$ contains the point P_k.

Case 2: This is similar to Case 1.

Case 3: The linear constraint define a line and the point P_k is on the line, so the linear constraint contains it.

Next, we prove the Lemma holds when $x_i = x_j$. There are also three cases according to θ.

Case 1: The linear constraint is $Y_{i,j}(x,\ \geq)$, that is, $x \geq x_i$. The condition shows that $x_k \geq Y_{i,j}(x_k,\ =)$; that is, $x_k \geq x_i$. So, the half surface defined by $Y_{i,j}(x,\ \geq)$ contains the point P_k.

Case 2: This is similar to Case 1.

Case 3: The linear constraint define a line and the point P_k is on the line, so the linear constraint contains it.

Therefore, the Lemma holds. ■

In the following, we give an algorithm to transform the TIN to a constraint database.

TIN TRANSFORMATION ALGORITHM:

Input: A TIN structure with the node list $Node(id, x, y)$,
the edge list $Edge(id, p_1, p_2, ifb)$,
and the triangle list $Triangle(id, e_1, e_2, e_3)$.

Output: A constraint representation of the TIN triangles.

```
for each triangle in TIN loop
    Get the three edges of this triangle
    for each edge in the triangle loop
        Get the two nodes of this edge
        for each node in the edge loop
            get the $x$ and $y$ of this node
        end-for
        Get the linear constraint $Y_{i,j}(x,\ \theta)$ defined in Formula 5.1 with
        these two nodes as $P_i$ and $P_j$.
        Use the other node in this triangle as $P_k$ and assign $\theta$ to contain $P_k$
        by Lemma 5.2.1.
    end-for
    Create a tuple in the result by using these three linear constraints
end-for
```

The above algorithm translates each TIN triangle into a constraint tuple, which represents the triangle in the constraint database.

Example 5.2.1 For the TIN generated in Example 5.1.1, the constraint representation is shown in Table 5.3.

ID	X	Y	
1	x	y	$2x - y \geq 0,\ x + 2y \geq 50,\ 3x + y \leq 100.$
2	x	y	$x + 2y \leq 100,\ 3x - 4y \leq 50,\ 3x + y \geq 100.$

Table 5.3: The Representation of Spatial Data

Chapter 6

Approximate Queries

This chapter discusses the basic approximate queries in Section 6.1 based on our work in [6], similarity queries in Section 6.2, queries on GIS applications in Section 6.3 based on our work in [7] and the approximate query evaluation in Section 6.4 based on our work in [33]. We illustrate the concepts using the Datalog query language.

6.1 Basic Queries

In this section, we use the relation $Temperature(SN, Temp, t)$ in Table 3.1 as one of the input relation.

Select: The select operator returns the tuples which satisfy the select condition.

Project: This operator is used to reorder the columns of a relation or to eliminate some columns of a relation. It creates a new relation which contains the specified columns of the original relation.

Example 6.1.1 For relation $Temperature$ the query

$$Query(SN, Temp) \ :- \ Temperature(SN, Temp, 1.5).$$

finds for each weather station the temperature at time $t = 1.5$. The output relation is shown in Table 6.1.

SN	**Temp**	
sn	$temp$	$sn = 1,\ temp = 81.5.$
sn	$temp$	$sn = 2,\ temp = 75.625.$
sn	$temp$	$sn = 3,\ temp = 84.667.$
sn	$temp$	$sn = 4,\ temp = 81.625.$

Table 6.1: Find Temperatures at $t = 1.5$

Intersection: This operation returns the intersection points of two relations to the user. The two relations should have the same attribute names and types.

SN	**Temp**	**t**	
sn	$temp$	t	$sn = 1,\ temp = 3t + 70,\ t \geq 0,\ t \leq 1.$
sn	$temp$	t	$sn = 1,\ temp = 8t + 65,\ t > 1,\ t \leq 4.$
sn	$temp$	t	$sn = 2,\ temp = 3.75t + 75,\ t \geq 0,\ t \leq 4.$
sn	$temp$	t	$sn = 3,\ temp = 5t + 80,\ t \geq 0,\ t \leq 3.$
sn	$temp$	t	$sn = 3,\ temp = -3t + 104,\ t > 3,\ t \leq 4.$
sn	$temp$	t	$sn = 4,\ temp = -2.25t + 80,\ t \geq 0,\ t \leq 4.$

Table 6.2: The $Temperature2$ Relation

Example 6.1.2 Given the relations $Temperature$ and $Temperature2$, shown in Table 6.2, the query

$$Query(SN, Temp, t) \ :- \ \ Temperature(SN, Temp, t),$$

$$Temperature2(SN, Temp, t).$$

returns the relation in Table 6.3 that contains those tuples that occur in both input relations:

SN	**Temp**	**t**	
sn	$temp$	t	$sn = 1,\ temp = 87.66,\ t = 2.83.$
sn	$temp$	t	$sn = 3,\ temp = 80,\ t = 0.$
sn	$temp$	t	$sn = 3,\ temp = 85.65,\ t = 1.13.$

Table 6.3: Intersection of $Temperature$ and $Temperature2$

Join: This operator executes the natural join operation for two relations A and B which have some attributes in common. It will match these same attributes, then returns the tuples whose projection onto the attributes of A belong to A and whose projection onto the attributes of B belong to B.

Example 6.1.3 The natural join of $Temperature$ and relation $Precipitation$, shown in Table 6.4, can be found by the query:

$$\begin{aligned} Query(SN, Temp, Prep, t) \ :- \ & Temperature(SN, Temp, t), \\ & Precipitation(SN, Prep, t). \end{aligned}$$

which gives the output relation shown in Table 6.5.

Min/Max: The Min and Max operators return the minimum and maximum values, respectively, for any linear function, called the objective function. The objective function can be as simple as a single variable.

SN	**Prep**	**t**	
sn	$prep$	t	$sn = 1,\ prep = 50t + 1050,\ t \geq 0,\ t \leq 4.$
sn	$prep$	t	$sn = 2,\ prep = 35t + 980,\ t \geq 0,\ t \leq 4.$
sn	$prep$	t	$sn = 3,\ prep = -20t + 1040,\ t \geq 0,\ t \leq 4.$

Table 6.4: The *Precipitation* Relation

SN	**Temp**	**Prep**	**t**	
sn	$temp$	$prep$	t	$sn = 1,\ temp = 2t + 75,\ prep = 50t + 1050,$ $t \geq 0,\ t \leq 1.$
sn	$temp$	$prep$	t	$sn = 1,\ temp = 9t + 68,\ prep = 50t + 1050,$ $t > 1,\ t \leq 2.$
sn	$temp$	$prep$	t	$sn = 1,\ temp = 2t + 82,\ prep = 50t + 1050,$ $t > 2,\ t \leq 4.$
sn	$temp$	$prep$	t	$sn = 2,\ temp = 3.75t + 70,\ prep = 35t + 980,$ $t \geq 0,\ t \leq 4.$
sn	$temp$	$prep$	t	$sn = 3,\ temp = 6t + 80,\ prep = -20t + 1040,$ $t \geq 0,\ t \leq 1.$
sn	$temp$	$prep$	t	$sn = 3,\ temp = -2.67t + 88.67,\ prep = -20t + 1040,$ $t > 1,\ t \leq 4.$

Table 6.5: Join of *Temperature* and *Precipitation*

Example 6.1.4 The query

$$Query(SN, \min(Temp))\ :-\ Temperature(SN, Temp, t)$$

finds for each weather station the minimum temperature. The result of the query is shown in Table 6.6.

6.2 Similarity Queries

It is possible to make similarity queries in constraint databases. In practice, people may want to know which map in a set of maps is similar to a given map. This can

SN	**Temp**	
sn	$temp$	$sn = 1,\ temp = 75.$
sn	$temp$	$sn = 2,\ temp = 70.$
sn	$temp$	$sn = 3,\ temp = 78.$
sn	$temp$	$sn = 4,\ temp = 76.$

Table 6.6: Find the Minimum Temperature

be done by using similarity queries.

The definition of the similarity depends on the actual situation. There is no universal formula to define the similarity. Our system allows a user to change the evaluation rule for the similarity measure.

Example 6.2.1 Suppose there are the relation $R_{Prec}(State, Area, Prec, t)$ for precipitation. The user wants to find in which year the average precipitation of the U.S. is the closest to that in the year 1997.

To query this information, the user may define the similarity measure rule for precipitation. The similarity of two instances A and B with n states and each state with area a_i and precipitation values $A.z_i$ and $B.z_i$, respectively, for $1 \leq i \leq n$ may be defined as follows:

$$sim(A, B) = \sum_{i=1}^{n} a_i \mid A.z_i - B.z_i \mid$$

Based on the specified similarity rule, the constraint database system can calculate the similarity value for each time instance, then select the year when its similarity value is closest to that in the year 1997.

6.3 GIS-based Queries

Representation: A geo-temporal, or more generally, a spatiotemporal data set (x, y, z, t) has for each location (x, y) some value (z) that varies with time (t). Such a spatiotemporal data set can be obtained by observations, for example weather stations where x and y could be longitude and latitude and z could be temperature at time instance t.

Such a point set could be stored in a relational database as a single relation with four attributes and real number attribute values, but this representation would be inconvenient for querying. For example, if the weather station recorded the temperature every 72 hours for a location, then it is not easy to tell what is the best estimate for the temperature at say 100 hours at that location.

Therefore, we transform a spatial data set to a constraint database representation in Chapter 5. Suppose that we have input constraint relations $Elevation(x, y, e)$ and $Slope(x, y, s)$ and $Aspect(x, y, a)$ that express for each location (x, y) by piecewise linear functions the elevation, the slope and the aspect, respectively.

Queries: Suppose that for a given region there are several linear equations obtained empirically to estimate various other spatiotemporal variables, for example the Mean Annual Air Temperature (m). Then we could find an estimate of m for each location as follows:

$$
\begin{aligned}
&MAAT(x, y, m)\ :-\\
&\qquad Elevation(x, y, e),\ Aspect(x, y, a),\ Slope(x, y, s),\\
&\qquad m = 74.3 - 0.000005x - 0.000011y - 0.00524e - 0.000203a - 0.0432s.
\end{aligned}
$$

where the second line is the estimate of m in ^oC at each location (x, y) in terms of the elevation (z), aspect (a) and slope (s). This generates a new defined relation $MAAT$ that also has the third argument a function of time.

We can also define new relations based on the input relations and the previously generated relations. For example, we can find p at each location by the following Datalog query:

$$
\begin{aligned}
&PE(x, y, p)\ :-\\
&\qquad Elevation(x, y, z),\ Aspect(x, y, a),\ Slope(x, y, s),\ MAAT(x, y, m),\\
&\qquad p = 670.21 - 0.0000395x - 0.0000927y - 0.103228z + 0.000876a\\
&\qquad\qquad -0.571s + 5.3m.
\end{aligned}
$$

where the second line is the estimator for p in terms of z, a, s and m. Similarly, we can estimate the mean annual precipitation $\overline{p}$ by defining a relation MAP.

As a final example, the annual water balance b can be estimated as the difference between MAP and PE. This can be expressed as a Datalog query:

$$AWB(x, y, b)\ :-$$

$$MAP(x, y, \overline{p}),\ PE(x, y, p),$$

$$b = \overline{p} - p.$$

6.4 Approximate Query Evaluation

In this section we made some experiments based on approximate queries to check if the piecewise linear approximation is appropriate for the queries. We used the temporal dataset containing 10 years' daily high temperature and daily low temperature datasets between the year 1987 and 1996 from the weather station in the state of Nebraska (station number: 252820). We use different error tolerance Ψ values to restrict the accuracy of the approximation. The original weather data comes from the website of the National Climatic Data Center at http://www.ncdc.noaa.gov. Since the daily high temperature and daily low temperature are changing too much for each day, we use the running average within 7 days to smooth the occasional changes.

Table 6.4 gives the number of pieces of the piecewise linear approximation of the high temperature dataset and low temperature dataset with different error tolerance values. — means that we do not use the approximation in that row. Here the unit of Ψ for the temperature is degrees Fahrenheit.

$R_{high}(day,\ high\_temp)$ is the high temperature relation and $R_{low}(day,\ low\_temp)$ is the low temperature relation. During the comparison, R_{high} and R_{low} are renamed as R'_{high} and R'_{low} respectively. Also, we get the relation

Ψ (^oF)	**#Pieces in High Temperature**	**# Pieces in Low Temperature**
—	3,653	3,653
0.5	2,199	1,735
1.0	1,426	1,084
1.5	1,017	774
2.0	790	594
2.5	625	502

Table 6.7: Number of Pieces in Datasets

$R(day,\ high\_temp,\ low\_temp)$ by joining the relation R_{high} and R_{low}.

The following are some experimental tests for simple and composite Datalog queries based on these three relations. We use the MLPQ [17, 31, 30, 34] system to evaluate the queries. For an approximate query output, it can be divided into four categories [19] shown in Figure 6.1:

1. *Relevant retrieved*: This part belongs to the actual output and is retrieved.
2. *Relevant not retrieved*: This part belongs to the actual output but is not retrieved.
3. *Non-relevant retrieved*: This part does not belong to the actual output but is retrieved.
4. *Non-relevant not retrieved*: This part does not belong to the actual output and is not retrieved.

As the division of those four categories, the relevant retrieved and the non-

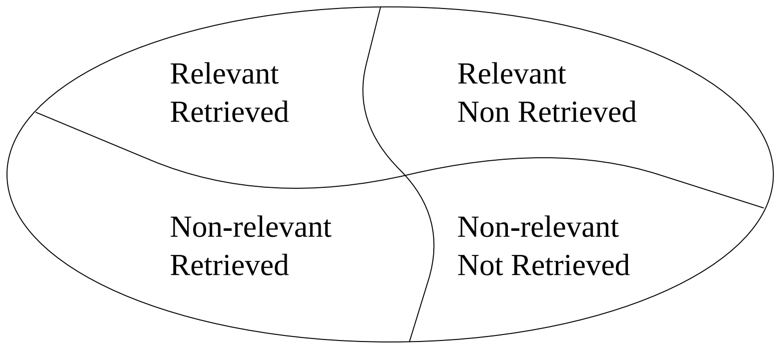

Figure 6.1: The Query Output Categories

relevant not retrieved for the approximate queries are equivalent to the actual query output; but the relevant not retrieved is missed in approximate queries, and the non-relevant retrieved is extraneous output in approximate queries.

We compare the accurate query results with our approximate query results by using *precision* and *recall* parameters, which are defined by:

$$Precision = \frac{\#\ Relevant\ Retrieved}{\#\ Total\ Retrieved}$$

$$Recall = \frac{\#\ Relevant\ Retrieved}{\#\ Possible\ Revelent}$$

6.4.1 Simple Queries

Example 6.4.1 We try to find all pairs of days such that for each the high temperature in one day is greater than or equal to that in the other. The Datalog query in MLPQ is as follows.

$$Pair(day1,\ day2)\ :-\ R_{high}(day1,\ high\_temp1),$$

$$R_{high}(day2,\ high\_temp2),$$

$$high\_temp1 \geq high\_temp2.$$

The test results are as Table 6.4.1.

Ψ (^oF)	**# MLPQ Constraints**	**# Converted Solutions**	**Precision**	**Recall**
—	6,676,854	6,676,854	100.00%	100.00%
0.5	2,421,002	6,859,132	99.69%	99.52%
1.0	1,017,929	7,049,422	99.24%	99.03%
1.5	517,763	7,239,712	98.79%	98.56%
2.0	312,486	7,428,668	98.36%	98.12%
2.5	195,636	7,614,284	97.95%	97.71%

Table 6.8: $R_{high} \geq R'_{high}$

Example 6.4.2 We try to find all pairs of days such that for each the low temperature in one day is greater than or equal to that in the other. The Datalog query in MLPQ is as follows.

$$Pair(day1,\ day2)\ :-\ R_{low}(day1,\ low\_temp1),$$

$$R_{low}(day2,\ low\_temp2),$$

$$low\_temp1 \geq low\_temp2.$$

The test results are as Table 6.9.

Ψ (^oF)	**# MLPQ Constraints**	**# Converted Solutions**	**Precision**	**Recall**
—	$6,678,121$	$6,678,121$	100.00%	100.00%
0.5	$1,506,890$	$6,874,458$	99.58%	99.37%
1.0	$588,189$	$7,082,147$	99.10%	98.85%
1.5	$299,962$	$7,291,840$	98.65%	98.40%
2.0	$176,720$	$7,498,194$	98.14%	97.88%
2.5	$126,257$	$7,701,877$	97.69%	97.42%

Table 6.9: $R_{low} \geq R'_{low}$

6.4.2 Composite Queries

Example 6.4.3 We try to find all pairs of days such that for each the high temperature in one day is greater than or equal to that in the other and the low temperature in one day is also greater than or equal to that in the other. The Datalog query in MLPQ is as follows.

$$\begin{aligned} Pair(day1,\ day2) \ :- \ & R(day1,\ high\_temp1,\ low\_temp1), \\ & R(day2,\ high\_temp2,\ low\_temp2), \\ & high\_temp1 \geq high\_temp2, \\ & low\_temp1 \geq low\_temp2. \end{aligned}$$

The test results are as Table 6.10.

Example 6.4.4 We try to find all pairs of days such that for each the high temperature in one day is greater than or equal to that in the other or the low temperature in one day is also greater than or equal to that in the other. The Datalog query in

Ψ (^oF)	**# MLPQ Constraints**	**# Converted Solutions**	**Precision**	**Recall**
—	$6,123,916$	$6,123,916$	100.00%	100.00%
0.5	$3,648,265$	$6,314,605$	99.61%	99.42%
1.0	$1,840,304$	$6,516,973$	99.11%	98.88%
1.5	$1,054,473$	$6,723,440$	98.65%	98.44%
2.0	$641,163$	$6,929,223$	98.18%	98.01%
2.5	$453,096$	$7,136,790$	97.80%	97.76%

Table 6.10: $R_{high} \geq R'_{high}$ and $R_{low} \geq R'_{low}$

MLPQ is as follows.

$$\begin{aligned}
Pair(day1,\ day2) \ &:- \ R(day1,\ high\_temp1,\ low\_temp1),\\
&\qquad R(day2,\ high\_temp2,\ low\_temp2),\\
&\qquad high\_temp1 \geq high\_temp2.\\
Pair(day1,\ day2) \ &:- \ R(day1,\ high\_temp1,\ low\_temp1),\\
&\qquad R(day2,\ high\_temp2,\ low\_temp2),\\
&\qquad low\_temp1 \geq low\_temp2.
\end{aligned}$$

The test results are as Table 6.11.

6.4.3 Query Analysis

From the experimental results of the above queries, we know that we can achieve a great data reduction while keeping a very high precision and recall in queries by using piecewise linear approximation. We think this result holds for any reasonable time series datasets. As shown in Example 6.4.4, if we use $\Psi = 2.5$ to transform

Ψ (^oF)	**# MLPQ Constraints**	**# Converted Solutions**	**Precision**	**Recall**
—	$7,231,059$	$7,231,059$	100.00%	100.00%
0.5	$4,319,038$	$7,418,987$	99.66%	99.47%
1.0	$2,179,384$	$7,614,382$	99.23%	99.01%
1.5	$1,242,753$	$7,808,235$	98.82%	98.55%
2.0	$757,325$	$7,997,888$	98.37%	98.05%
2.5	$532,039$	$8,179,840$	97.92%	97.49%

Table 6.11: $R_{high} \geq R'_{high}$ or $R_{low} \geq R'_{low}$

the daily high temperatures and daily low temperatures, the composite query of finding the pairs of days has retrieved $532,039$ constraint tuples, which can be converted into $8,179,840$ regular tuples. Therefore, it compressed data with the compression ratio of $(8179840 - 532039)/8179840 = 93.5\%$. The precision and recall are 97.92% and 97.49% respectively in this case.

Figure 6.2, Figure 6.3, Figure 6.4 and Figure 6.5 show the relationship between precision, recall and the data compression for the above four queries respectively.

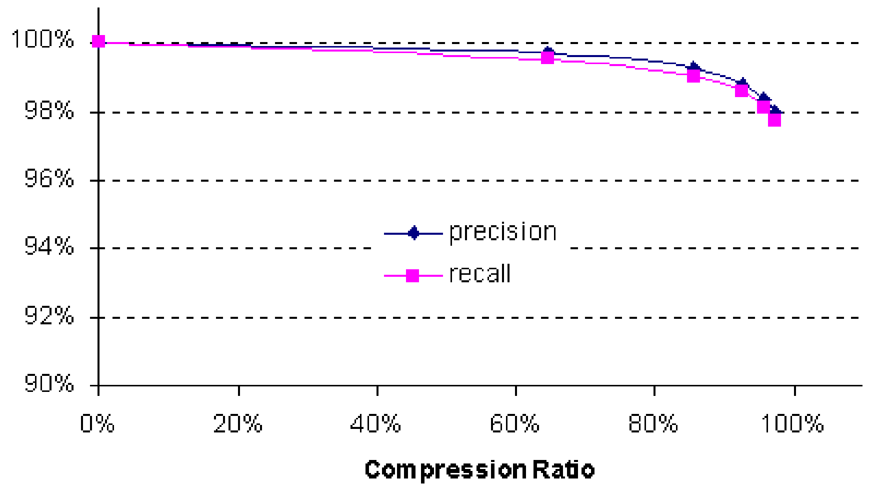

Figure 6.2: For Query $R_{high} \geq R'_{high}$

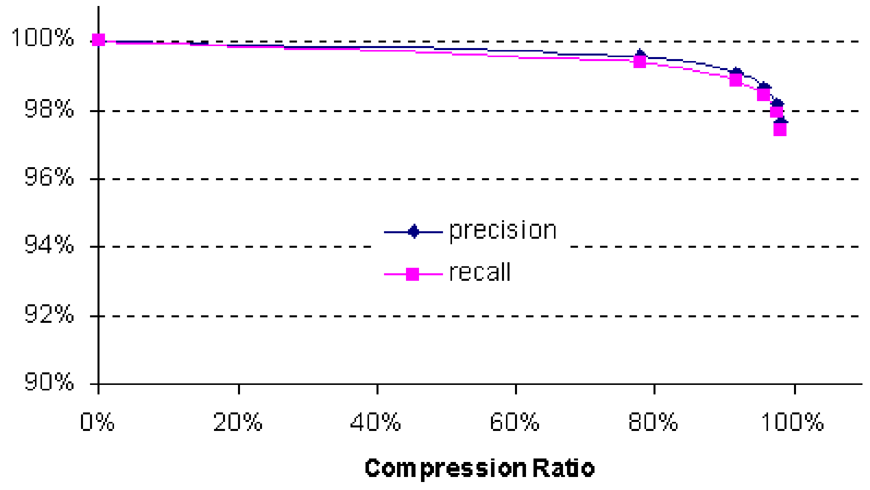

Figure 6.3: For Query $R_{low} \geq R'_{low}$

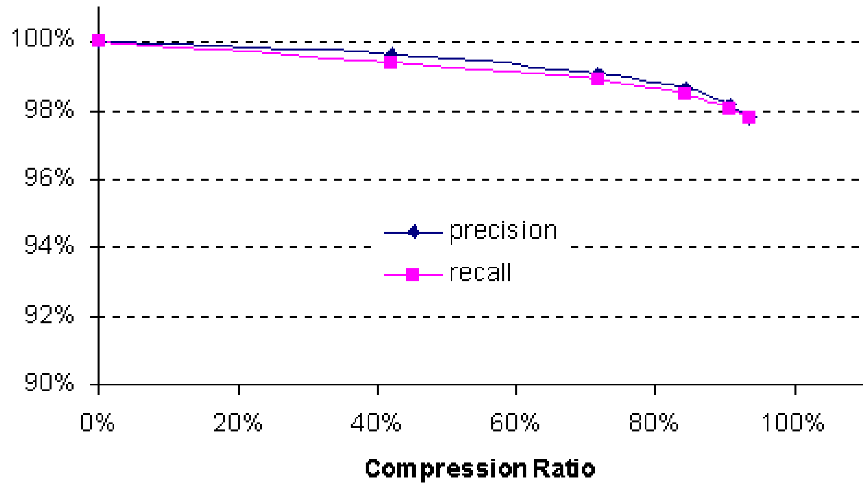

Figure 6.4: For Query $R_{high} \geq R'_{high}$ and $R_{low} \geq R'_{low}$

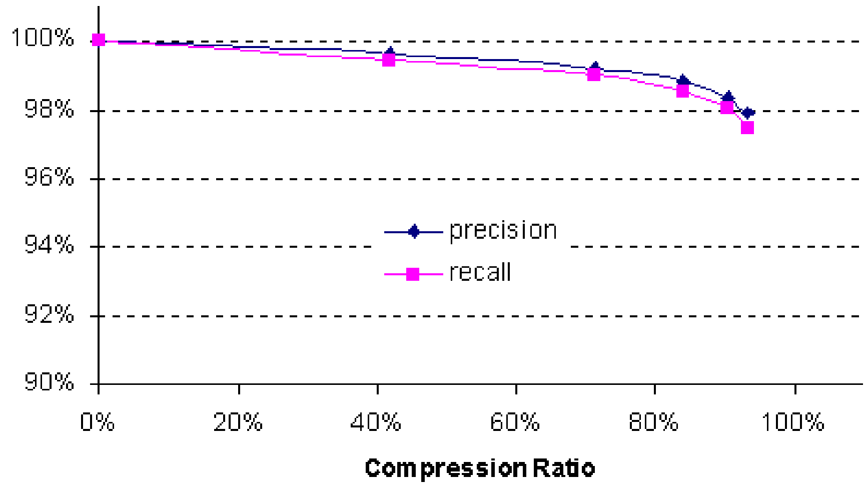

Figure 6.5: For Query $R_{high} \geq R'_{high}$ or $R_{low} \geq R'_{low}$

Chapter 7

Update Data Approximations

This chapter describes the update on the data approximations. Section 7.1 describes the update on the approximations of the temporal data, which is based on our work [32]. Section 7.2 describes the update on the approximations of the spatial data.

7.1 Update Temporal Data Approximations

In this section we consider what happens if some relational database that represents a time series is approximated by a constraint database and the user requests an insertion or deletion of a point in the time series. Note that the user can request insertions and deletions of time series points (i.e., tuples of the relational database) and not the constraint database, because the constraint database representation is hidden from the user.

If the user requests a deletion of a time series data point, then the request can be ignored because the approximation function still satisfies the error tolerance for the remaining points.

The insertion of points is much more complex. In this case, we have to update the piecewise linear function. Consider Figure 7.1. There the original piecewise linear function is shown as a solid black line in (1). In (2) the point P_1 is to be inserted, but the piecewise linear function is not changed since P_1 is in the error tolerance Ψ. In (3) the point P_2 is to be inserted, and the piecewise linear function is updated by splitting the middle piece into two pieces. In (4) the point P_3 is to be inserted, and the piecewise linear function is updated by splitting the third piece into two pieces.

The temporal insertion algorithm for a single data point (t_α, y_α) is shown below.

Remark: We have mentioned that a piecewise linear approximation can be represented as a set of endpoints of the pieces before. Now, we add a Boolean tag o to each point. The tag will be *true* if it is a point that is an original point, otherwise it is a point which was inserted and the tag will be *false*. This allows us to reconstruct from any updated piecewise linear function f, the original piecewise linear function, denoted f_o, as the sequence of points with the *true* tags. Finally, we assume that for no point (t_α, y_α) to be inserted is there already a point with time t_α.

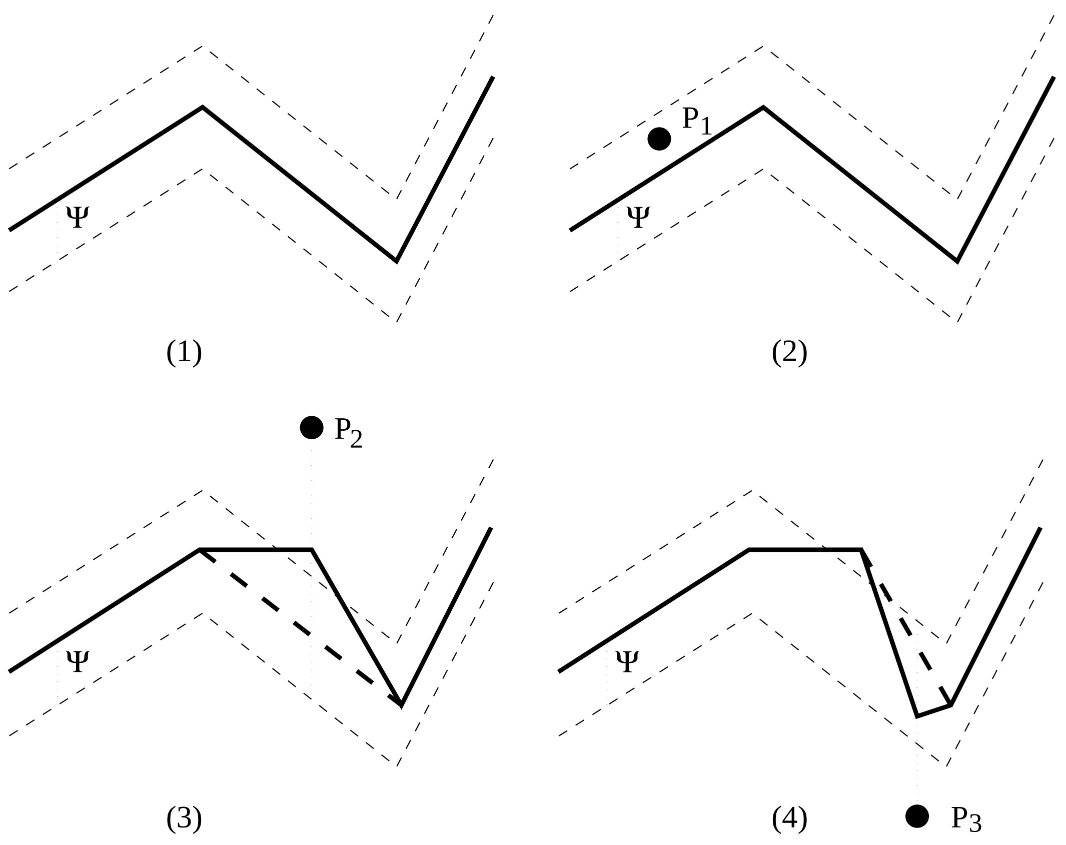

Figure 7.1: Inserting Points into Piecewise Linear Function

TEMPORAL INSERTION ALGORITHM:

Input: A piecewise linear function f represented as a sequence of points $(t_1, y_1, o_1), \ldots, (t_n, y_n, o_n)$.

Ψ the maximum error threshold in the approximation.

(t_α, y_α) the point to be inserted.

Output: An updated piecewise linear function.

```
**if** $t_\alpha < t_1$ **then**
```

```
    Add $(t_\alpha, y_\alpha, false)$ as the first point in $f$.
**else if** $t_\alpha > t_n$ **then**
    Add $(t_\alpha, y_\alpha, false)$ as the last point in $f$.
**else if** $f_o(t_\alpha) - \Psi > y_\alpha$ **then**
    Add $(t_\alpha, \frac{1}{2}((f_o(t_\alpha) + \Psi) + y_\alpha), false)$ between points with times $t_i < t_\alpha$
    and $t_{i+1} > t_\alpha$.
**else if** $f_o(t_\alpha) + \Psi < y_\alpha$ **then**
    Add $(t_\alpha, \frac{1}{2}((f_o(t_\alpha) - \Psi) + y_\alpha), false)$ between points with times $t_i < t_\alpha$
    and $t_{i+1} > t_\alpha$.
**end-if**
```

We can show the following theorem.

Theorem 7.1.1 Suppose that a time series S is approximated by a piecewise linear function f_o with n "pieces" and Ψ error tolerance. Then any set I of m insertions such that each insertion point is at most some constant $\delta \geq \Psi$ distance from f_o can be done by the temporal insertion algorithm such that the updated piecewise linear function f has at most $n + m$ "pieces" and the following holds.

$$|f(t_i) - y_i| \leq \frac{\Psi + \delta}{2} \text{ for each } (t_i, y_i) \in S \cup I.$$

Proof: From the algorithm above, there are four cases to insert one point (t_α, y_α), which are (1): $t_\alpha < t_1$, (2): $t_\alpha > t_n$, (3): $t_1 < t_\alpha < t_n$ and $f_o(t_\alpha) - \Psi > y_\alpha$, and (4): $t_1 < t_\alpha < t_n$ and $f_o(t_\alpha) + \Psi < y_\alpha$.

First we prove that the updated piecewise linear function f has at most $n+m$ pieces after m insertions. In either of the four cases, the algorithm adds one point to f. In other cases, the algorithm does not add any point to f. Therefore, for a sequence of m insertions, at most m points are added to f. Further, no points are ever deleted by the insertion algorithm. Hence, f has at most $n+m$ points.

Next, we prove that $|f(t_i) - y_i| \leq \frac{\Psi+\delta}{2}$ for each $(t_i, y_i) \in S \cup I$. We prove this by induction. Let us assume that after a sequence of insertions the condition is true and now we are inserting some new point (t_α, y_α). We prove that the condition also holds after the insertion of (t_α, y_α).

Case (1): The insertion algorithm in effect adds a new piece with endpoints (t_α, y_α) and (t_1, y_1) to f. The condition is clearly true in this case, because the point (t_α, y_α) is contained in the new piece.

Case (2): This is similar to case (1).

Case (3): In this case, the old piece between t_i, y_i and (t_{i+1}, y_{i+1}) is deleted and replaced with two new pieces, one with endpoints $A(t_i, y_i)$ and $C(t_\alpha, \frac{1}{2}((f_o(t_\alpha) + \Psi) + y_\alpha)$, and the other with endpoints $C(t_\alpha, \frac{1}{2}((f_o(t_\alpha) + \Psi) + y_\alpha)$ and $B(t_{i+1}, y_{i+1})$.

Note that by the induction hypothesis, all points in S before A or after B still satisfy the condition. Hence we only have to prove that the condition is still true for the points of f that are between A and B. First let us consider the points on the piece AC.

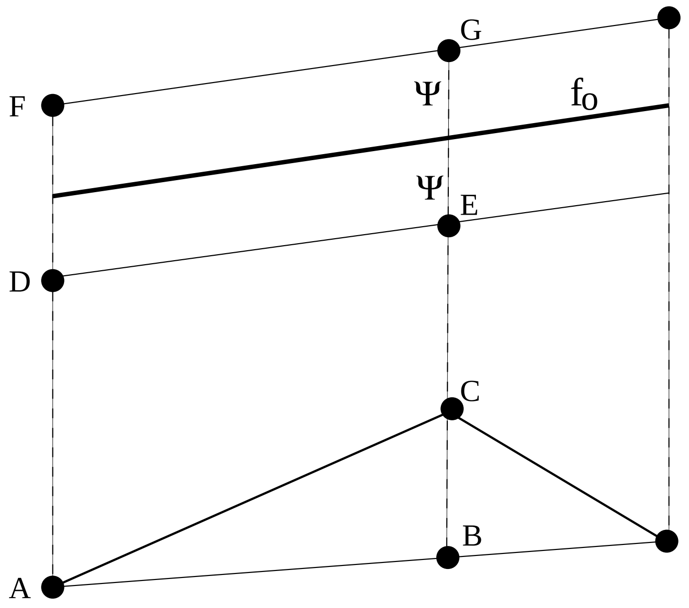

Figure 7.2: Insert a Temporal Point

Consider the original piecewise linear function between A and B. Let D and E be points on the line $f_o(t) - \Psi$, and F and G be points on the line $f_o(t) + \Psi$ as shown in Figure 7.2. The coordinates of these four points can be calculated to be $D(t_i, f_o(t_i) - \Psi)$, $E(t_\alpha, f_o(t_\alpha) - \Psi)$, $F(t_i, f_o(t_i) + \Psi)$ and $G(t_\alpha, f_o(t_\alpha) + \Psi)$. For the point to be inserted (t_α, y_α), we calculate the following:

$$\begin{array}{l} |f(t_\alpha) - y_\alpha| \\ = \frac{1}{2}((f_o(t_\alpha) + \Psi) + y_\alpha) - y_\alpha \text{ by y coordinate of C} \\ = \frac{1}{2}((f_o(t_\alpha) + \Psi) - y_\alpha) \\ = \frac{1}{2}(\Psi + (f_o(t_\alpha) - y_\alpha)) \\ \leq \frac{\Psi+\delta}{2}. \end{array} \tag{7.1}$$

The last inequality follows form the condition that each point inserted is at most δ distance from the original piecewise linear function. Hence (t_α, y_α) satisfies the condition.

Note that there cannot be any other I point H before (t_α, y_α) that is between A and C and has more than Ψ distance from f_o. If we had, then we would have to use either AH or HB instead of AB when we are inserting (t_α, y_α).

Now we can assume that all S and I points before (t_α, y_α) and between A and C are at most Ψ distance from f_o. Therefore, these all fall into the trapezoid region $DEGF$. We show that any point within $DEGF$ satisfies the condition.

At first we show the condition for the corner vertices. For D and F the condition is true by the induction hypothesis; that is, they are both at most $\frac{\Psi+\delta}{2}$ distance from A. Note that y_α is at most δ and both E and G are at most Ψ distance from f_o. Since the y coordinate of C is at the midpoint of y_α and the y coordinate of G, both G and E are at most $\frac{\Psi+\delta}{2}$ distance from C.

Let A' be the point exactly $\frac{\Psi+\delta}{2}$ below F, and let C' be the point exactly $\frac{\Psi+\delta}{2}$ below G. Suppose that $M = (t, y)$ is any point within $DEGF$. Let M_1 be the point directly above M and intersecting the line segment FG and M_2 be the point directly below M and intersecting the line segment $A'C'$ as shown in Figure 7.3.

Clearly, the distance between M and AC is less than the distance between M_1 and M_2, which is exactly $\frac{\Psi+\delta}{2}$. Hence M must satisfy the condition. Therefore,

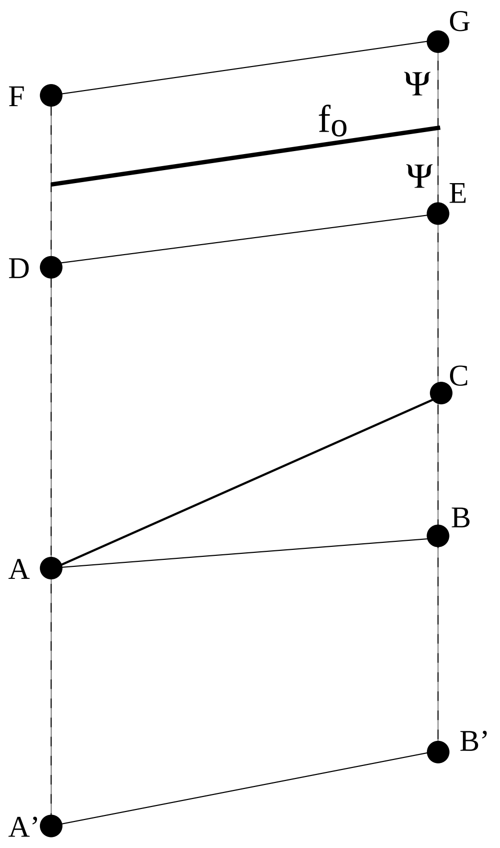

Figure 7.3: Proof Condition for Point M

all points within $DEGF$ satisfy the condition.

The above took care for points between A and C. We can prove similarly that all points in S and in I before (t_α, y_α) and between C and B also satisfy the condition.

Case (4): It is similar to Case (3). ■

For example, if $\delta = 3\Psi$, then the error tolerance for the updated piecewise linear approximation will be 2Ψ for all original and newly inserted data points.

Example 7.1.1 Let a time series T be the sequence of points: $(1,1)$, $(3,3)$, $(10,6)$, $(13,9)$, $(15,7)$, $(18,4)$, $(22,4)$, $(23,6)$, $(28,9)$ and $(29,12)$, the piecewise linear function with the error tolerance $\Psi = 2$ is shown in Table 7.1 and Figure 7.4.

Y	**t**	
y	t	$3y - 2t = 1,\ t \geq 1,\ t \leq 13.$
y	t	$y + t = 22,\ t > 13,\ t \leq 18.$
y	t	$y = 4,\ t > 18,\ t \leq 22.$
y	t	$7y - 8t = -148,\ t > 22,\ t \leq 29.$

Table 7.1: The Piecewise Linear Function of T with $\Psi = 2$

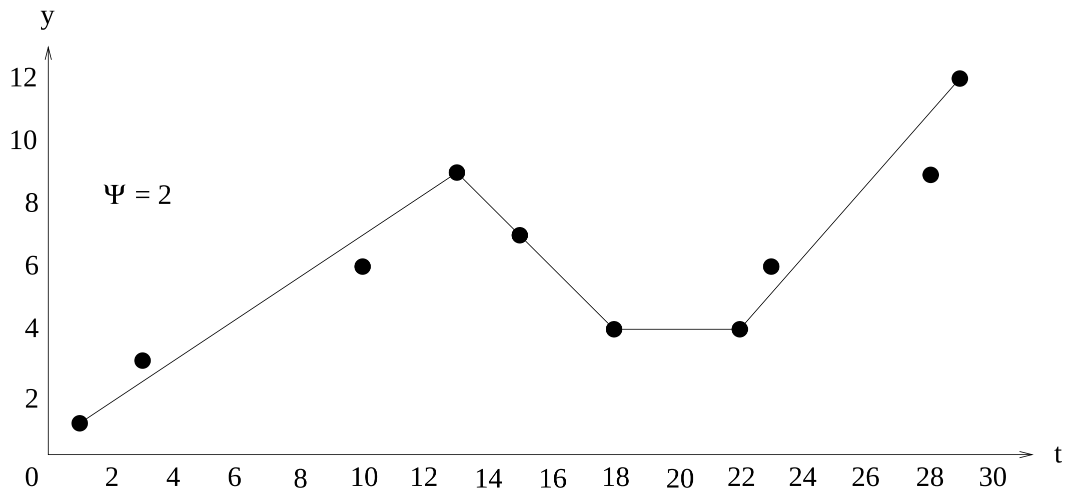

Figure 7.4: The PLA of T with $\Psi = 2$

Then insert four points $P1(16,5)$, $P2(26,5)$, $P3(5,12)$, and $P4(8,2)$. The resulting piecewise linear function is shown in Table 7.2 and Figure 7.5.

7.2 Update Spatial Data Approximations

Section 7.1 describes the update on temporal data approximations.

Y	**t**	
y	t	$2y - 2.9t = -0.9,\ t \geq 1,\ t \leq 5.$
y	t	$3y + 2t = 30.4,\ t > 5,\ t \leq 8.$
y	t	$5y - 4.2t = -9.6,\ t > 8,\ t \leq 13.$
y	t	$y + t = 22,\ t > 13,\ t \leq 18.$
y	t	$y = 4,\ t > 18,\ t \leq 22.$
y	t	$2y - 1.9t = -33.8,\ t > 22,\ t \leq 26.$
y	t	$y - 1.4t = -28.6,\ t > 26,\ t \leq 29.$

Table 7.2: Inserting Points in Piecewise Linear Function of T

For deleting a spatial data point, one kind of method is just to ignore the request for the internal spatial point, and delete the triangles with this point as the node if it is a boundary point.

Another method is to find the influenced triangles with the point to be deleted, then get the nodes of these triangles and generate new TIN for these remaining influenced nodes. We then recalculate the temporal approximation for each new generate triangles and delete the old triangles and their related temporal approximations.

For inserting a spatial data point, if it is not in any triangles, then we just find the nearest edge and construct a new triangle with this point. In the following we discuss the insertion of a spatial data point in a general case. We assume that the inserted point is fallen in some current triangle.

The following is an insertion algorithm for inserting a spatial data point $P_\alpha(x_\alpha, y_\alpha)$ into the spatial approximations.

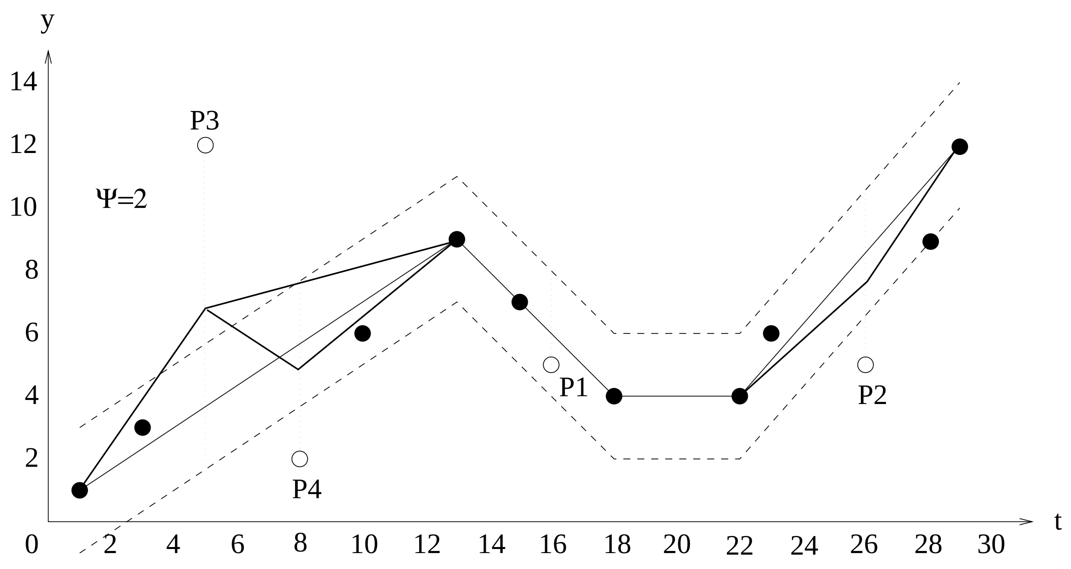

Figure 7.5: Inserting Points in Piecewise Linear Function of T

SPATIAL INSERTION ALGORITHM:

Input: A spatial representation with a triangle relation T

(x_α, y_α) the point to be inserted.

Output: An updated spatial representation.

Find the triangle T_β where the point (x_α, y_α) is contained by using intersection.

Calculate the nodes P_i, P_j and P_k of T_β.

Delete T_β, and insert $T_{i,j,\alpha}$, $T_{j,k,\alpha}$ and $T_{i,k,\alpha}$ into T.

Figure 7.6 illustrate the spatial insertion algorithm, which split the triangle into three smaller triangles.

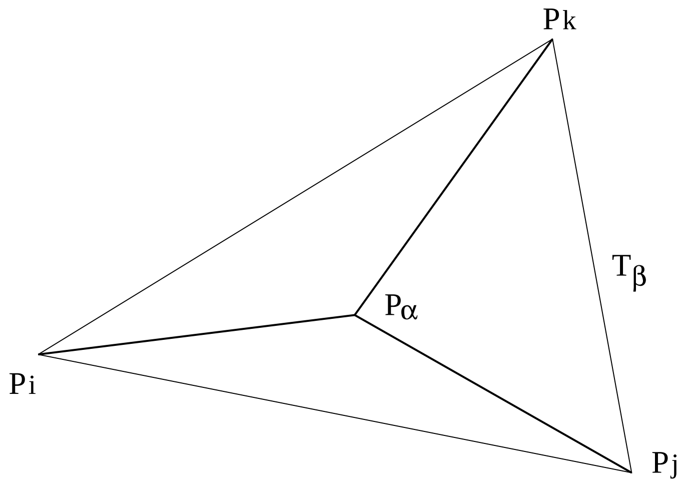

Figure 7.6: Insert a Spatial Data Point

Chapter 8

Visualization

This chapter discusses the visualization techniques, including color bands display and isometric color animation. Section 8.1 is based on our work in [31, 30], Section 8.2 is based on our work in [7, 29].

8.1 Color Bands Display

In this section, we first give an example of color bands display. Then we provide a algorithm to display a constraint relation with color bands. Finally, we introduce several GIS-based applications as examples.

In the linear constraint data model, each database consists of a finite set of constraint relations which indicate spatial, temporal or spatiotemporal information. And each constraint tuple is a conjunction of linear constraints. For example, we define a polygon as follows.

$$\begin{array}{ll}
Polygon(x,y)\ :- & Polygon(x,y)\ :- \\
\quad x \geq 0, & \quad -x \geq 0, \\
\quad y \geq 0, & \quad y \geq 0, \\
\quad 5x + 1.34y \leq 50, & \quad -5x + 1.34y \leq 50, \\
\quad x + y \leq 13.66, & \quad -x + y \leq 13.66, \\
\quad 1.34x + 5y \leq 50. & \quad -1.34x + 5y \leq 50.
\end{array}$$

$$\begin{array}{ll}
Polygon(x,y)\ :- & Polygon(x,y)\ :- \\
\quad -x \geq 0, & \quad x \geq 0, \\
\quad -y \geq 0, & \quad -y \geq 0, \\
\quad -5x - 1.34y \leq 50, & \quad 5x - 1.34y \leq 50, \\
\quad -x - y \leq 13.66, & \quad x - y \leq 13.66, \\
\quad -1.34x - 5y \leq 50. & \quad 1.34x - 5y \leq 50.
\end{array}$$

The above constraint tuples represent a polygon which is shown in Figure 8.1.

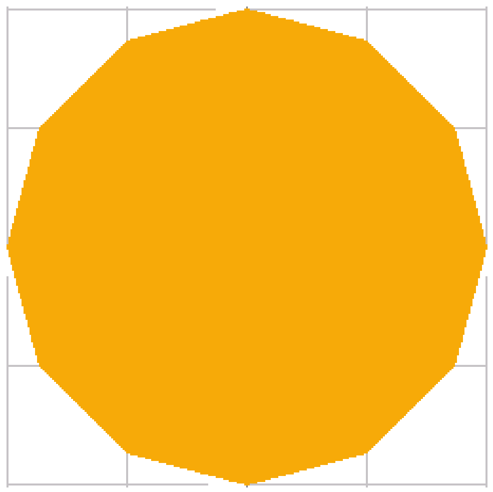

Figure 8.1: A Polygon

Now, by modifying the constraint representation, we can get the colored output of this polygon. Suppose that we have a relation equation between the elevations and x, y coordinates, and we want to display the polygon that shows different elevation intervals with different colors. In TAQS system with the function to display color bands, it just needs the user to add a variable parameter z which indicates the elevation and insert an elevation equation with respect to the x and y values for each constraint tuple. The new constraint representation for the corresponding color polygon is shown as follows.

$$\begin{array}{l} New\_Polygon(x,y,z)\ :- \\ \quad x \geq 0, \\ \quad y \geq 0, \\ \quad 5x + 1.34y \leq 50, \\ \quad x + y \leq 13.66, \\ \quad 1.34x + 5y \leq 50, \\ \quad x + y + z = 10. \end{array}$$

$$\begin{array}{l} New\_Polygon(x,y,z)\ :- \\ \quad -x \geq 0, \\ \quad y \geq 0, \\ \quad -5x + 1.34y \leq 50, \\ \quad -x + y \leq 13.66, \\ \quad -1.34x + 5y \leq 50, \\ \quad -x + y + z = 10. \end{array}$$

$$\begin{array}{l} New\_Polygon(x,y,z)\ :- \\ \quad -x \geq 0, \\ \quad -y \geq 0, \\ \quad -5x - 1.34y \leq 50, \\ \quad -x - y \leq 13.66, \\ \quad -1.34x - 5y \leq 50, \\ \quad -x - y + z = 10. \end{array}$$

$$\begin{array}{l} New\_Polygon(x,y,z)\ :- \\ \quad x \geq 0, \\ \quad -y \geq 0, \\ \quad 5x - 1.34y \leq 50, \\ \quad x - y \leq 13.66, \\ \quad 1.34x - 5y \leq 50, \\ \quad x - y + z = 10. \end{array}$$

Then, after the user sets the colors for different elevation intervals, the system will automatically generate the color representation of the polygon and display the polygon whose elevations are represented by the different color patterns after the user clicks the *ColorRelation* button. The color figure of the polygon is shown in Figure 8.2.

There are many benefits to display the objects with multiple specified colors to denote distinct value intervals for a certain attribute. It can let us know more details about the query results and analyze them more easily. We can make the most important part displayed by a striking color and other parts displayed by general colors. It is obvious to know where the most important part is and how

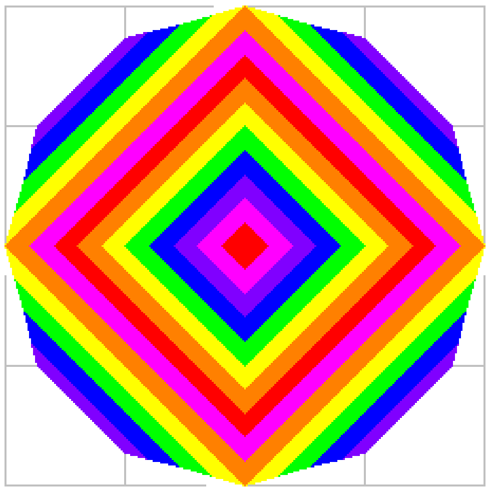

Figure 8.2: The Color Polygon

much it involves.

8.1.1 Display Algorithm

In this section we show how to make the new representation of the constraint tuples to be displayed with the designated colors.

The spatial data is represented in constraint relations, which have the following form:

$$R(id, x, y) :- \text{set of linear inequality constraints over } x \text{ and } y \text{ variables.}$$

And the non-spatial data is stored in regular relations of the form $R(id, a_1, a_2, ..., a_n)$.

Now, if there is a new attribute (say z) which can be expressed by an equation with the variables x, y, and itself z, add this equation into the constraint relations. The new relation R^* has the form:

$$\begin{aligned} R^*(id,x,y,z) \quad :- \quad & \text{set of linear inequality constraints over } x,y \text{ variables,} \\ & \text{one equality constraint with the form } c_1x + c_2y + c_3 = z. \end{aligned}$$

Suppose a user wants to display the result with n different colors according to the corresponding z intervals. The following algorithm will convert the constraint relation tuple to n (may be less than n) new constraint relation tuples with their corresponding colors.

COLOR BANDS DISPLAY ALGORITHM:

Input: A constraint relation R with the form of R^*.

n number of z intervals $[z1_1, z1_2], \cdots, [zn_1, zn_2]$.

n corresponding colors $color_1, \cdots, color_n$.

Output: A new constraint relation R'.

```
for each constraint tuple in $R$ do
    Find the equality constraint with the form $c_1x + c_2y + c_3 = z$.
    for each $z$ interval $[zi_1, zi_2]$ do
        Create a new tuple $T$ which is the same as $R$ except changing
        the equality constraint to two inequalities constraints:
        $c_1x + c_2y + c_3 \geq zi_1$ and $c_1x + c_2y + c_3 \leq zi_2$.
        Delete the variable $z$ from $T$.
        Set the color of this tuple to be $color_i$.
        Decide if $T$ is satisfiable. If it is, add $T$ to $R'$.
    end-for
end-for
```

Then, the system will display the results with the corresponding specified colors for the different variable parameter intervals. The computational complexity of this algorithm is $O(nm)$ time in the worst case, where n is the number of intervals of the variable parameter values, m is the number of constraint tuples. In practice there almost always exist some new created relation tuples which are not satisfiable, so less than $n \times m$ new relation tuples to be created in the general case.

8.1.2 GIS-based Applications

GIS-based applications can use color bands display to improve the data visualization. For a geographic map, we can use different colors to display the temperature distribution, evapotranspiration distribution, precipitation distribution, and so on.

The Nebraska State Map: First, we transform the TIN structure of the Nebraska state map into a constraint relation $MAP(x, y, z, a, s)$, where x and y are the longitude and latitude of the sampled points on the map. For each tuple in MAP relation, it represents a triangle of the map. z is represented by an equality constraint of x and y. a is aspect of the triangle, and s is the slope of the triangle. The MAP relation is shown as follows.

$$
\begin{aligned}
&MAP(x, y, z, a, s) :- \\
&\quad a_{i1}x + b_{i1}y < c_{i1}, \\
&\quad a_{i2}x + b_{i2}y < c_{i2}, \\
&\quad a_{i3}x + b_{i3}y < c_{i3}, \\
&\quad a_{i4}x + b_{i4}y + z = c_{i4}, \\
&\quad a = c_{i5}, \\
&\quad s = c_{i6}.
\end{aligned}
$$

where $1 \leq i \leq n$, and n is the number of triangles which compose the whole geographic map.

Now if we want to show the elevation distribution, just specify the elevation intervals and their colors. The Nebraska color map for the elevation is shown in Figure 8.3.

The Mean Annual Air Temperature of the State of Nebraska: By adding the new attribute, Mean Annual Air Temperature (MAAT), and its model equation,

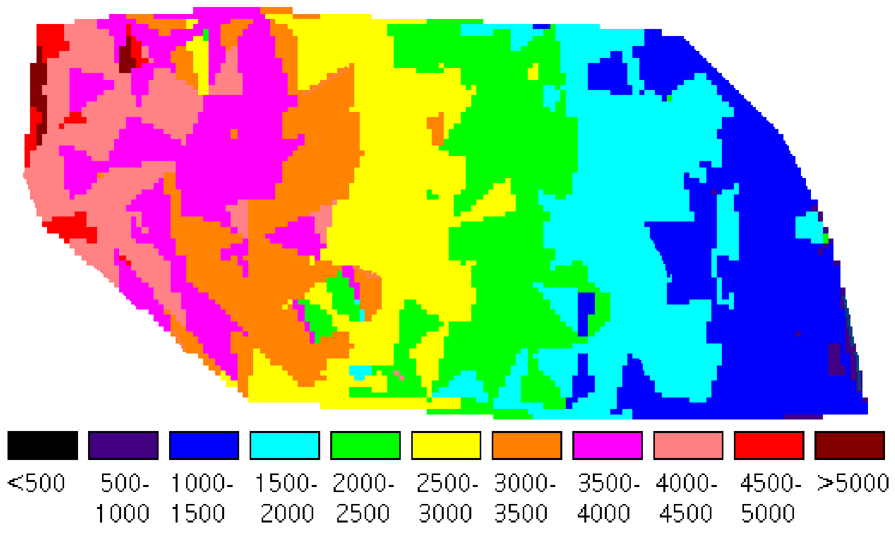

Figure 8.3: The Nebraska Elevation Map

the MAAT (unit is ^oC) distribution graph can be drawn according to the specified colors. The Datalog query is shown below.

$$\begin{aligned} &MAAT(x, y, maat) :- \\ &\quad 0.000005\ x + 0.000011\ y + 0.00524\ z + 0.000203\ a + 0.0432\ s + maat \\ &\qquad = 74.3, \\ &\quad MAP(x, y, z, a, s). \end{aligned}$$

where the first rule in the definition is the model equation between MAAT and other geographic attributes. The result of this quey is shown in Figure 8.4.

The Potential Evapotranspiration of the State of Nebraska: Similarly, we can use the model equation of the potential evapotranspiration (PET) (unit is mm) to draw the potential evapotranspiration map as shown in Figure 8.5.

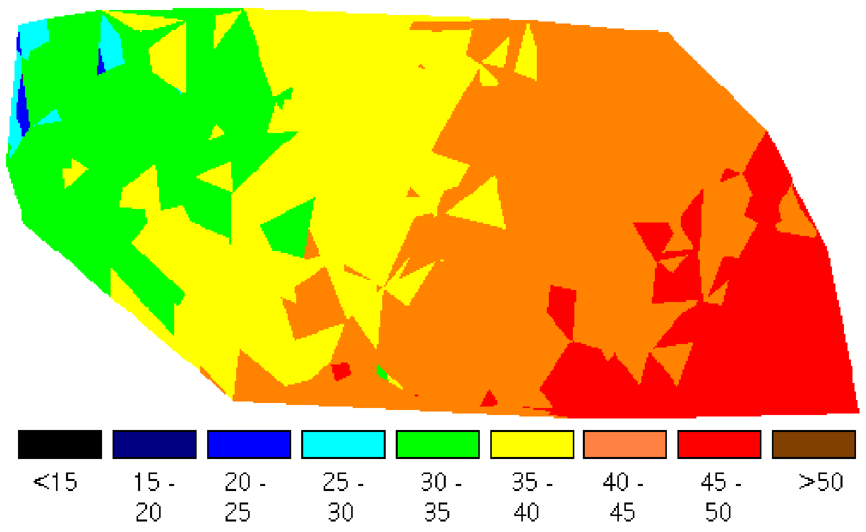

Figure 8.4: The Nebraska Mean Annual Air Temperature Map

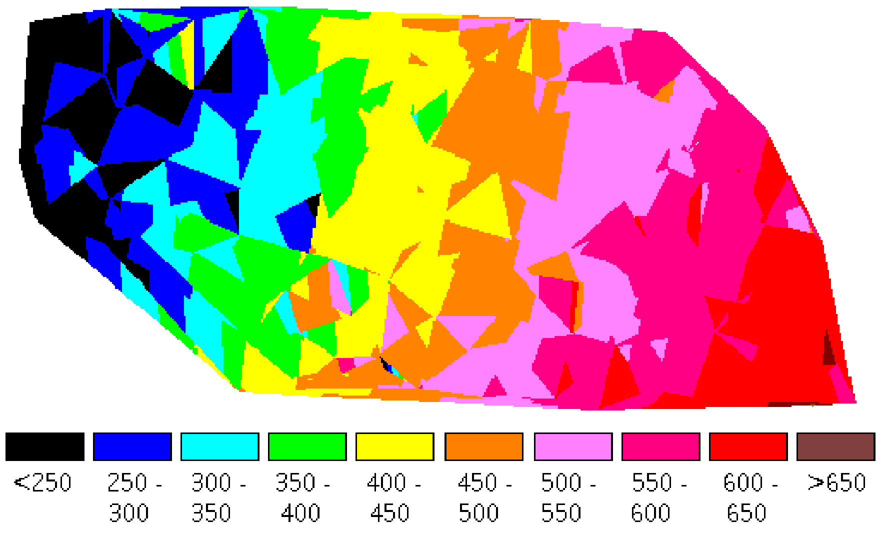

Figure 8.5: The Nebraska Potential Evapotranspiration Map

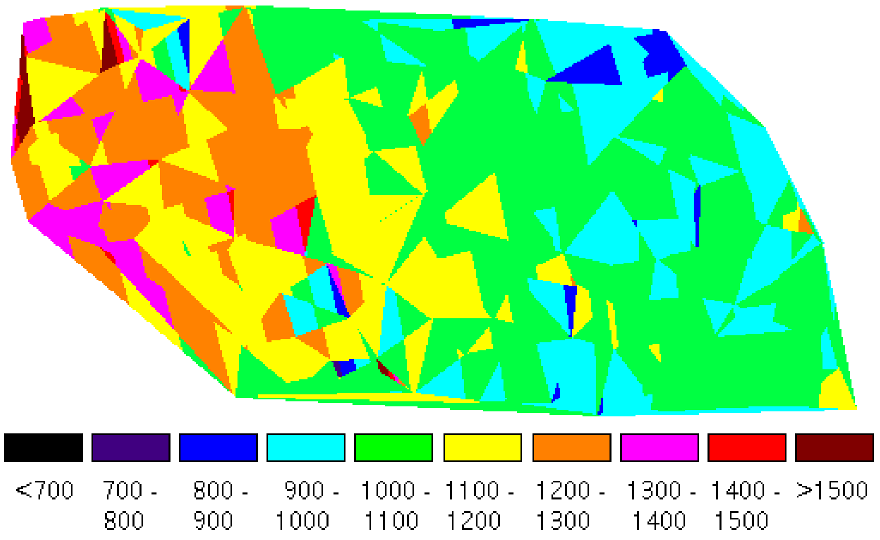

Figure 8.6: The Nebraska Mean Annual Precipitation Map

Other Distribution Maps for the State of Nebraska: We also have mean annual precipitation model equation (PREC) (unit is mm) and frost-free period model equation (FFP) (unit is day) (base 32 ^oF), and Annual Water Balance (AWB) (unit is mm) model equation. By using these model equations, we can plot their distribution map shown in Figure 8.6, Figure 8.7 and Figure 8.8, respectively.

8.2 Isometric Color Animation

By using temporal and spatial approximation, a set of spatiotemporal data can be represented as two constraint relations, one is for the spatial information and the other is for the temporal information of each spatial point. The spatial points are translated to triangles in 2-dimensional space, and their temporal information related to each point is translated into a piecewise linear function.

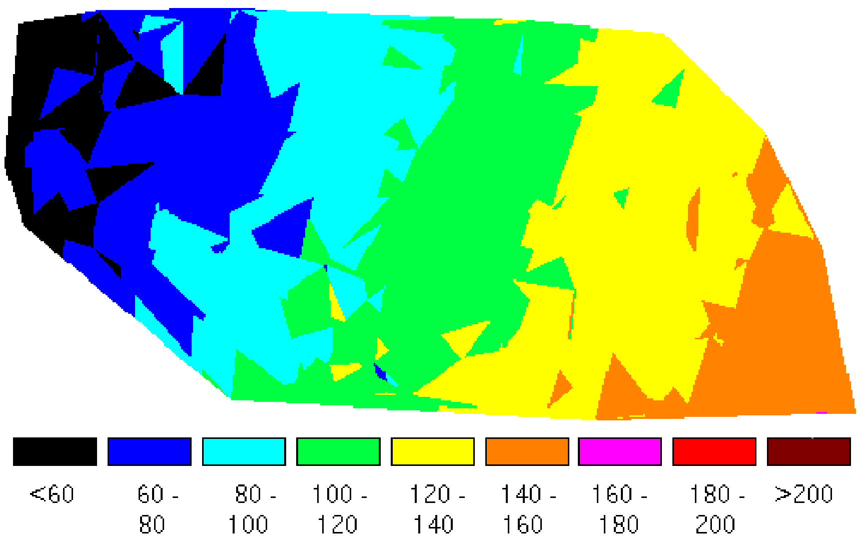

Figure 8.7: The Nebraska Frost-Free Period Map

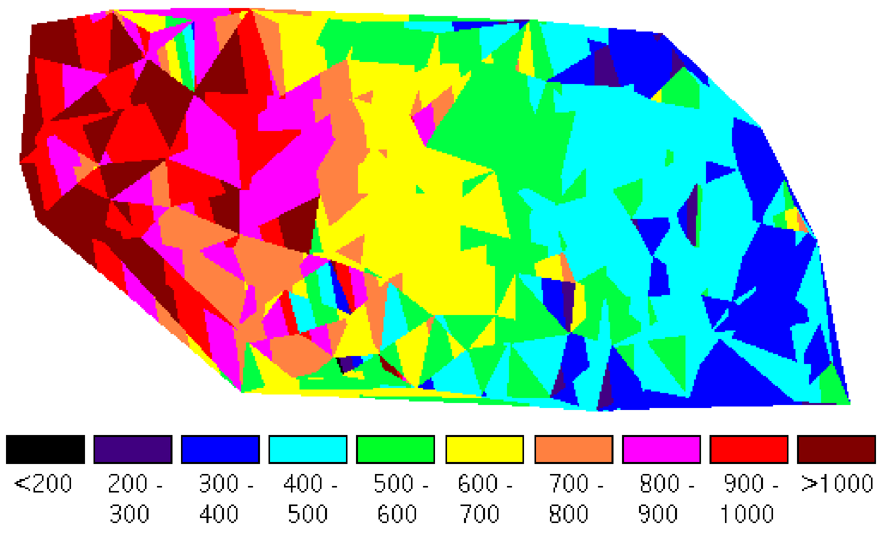

Figure 8.8: The Nebraska Annual Water Balance Map

To get a snapshot of the animation, substitute the variable t in each piecewise linear functions with the specified time value to get the property values at that time. Then draw the triangles with the colors whose ranges cover their property values, respectively.

Figure 8.9, Figure 8.10, Figure 8.11 and Figure 8.12 show four snapshots of the isometric color animation for the U.S. precipitation in January, April, July and October in the year 1997, respectively. The high precipitation areas are displayed by blue (darker gray) shade colors and the low precipitation areas by red (lighter gray) shade colors.

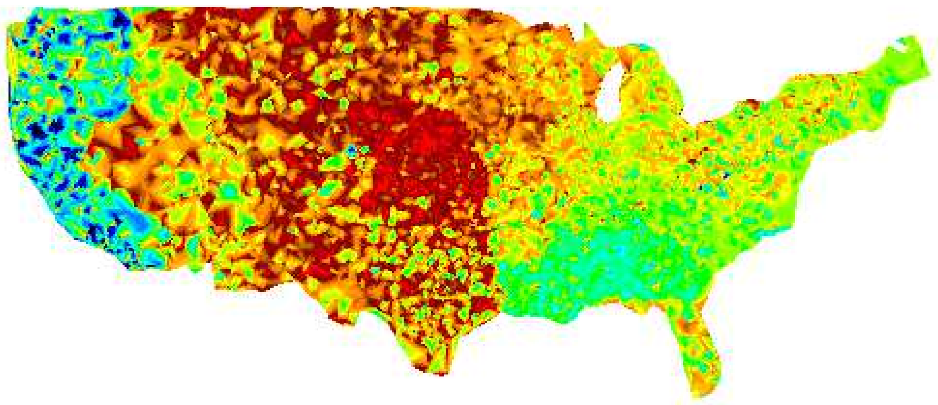

Figure 8.9: The Snapshot of U.S. Precipitation in January, 1997

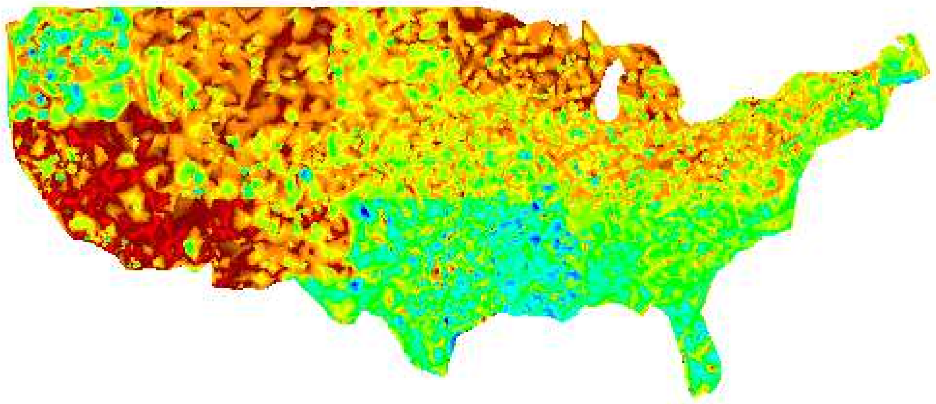

Figure 8.10: The Snapshot of U.S. Precipitation in April, 1997

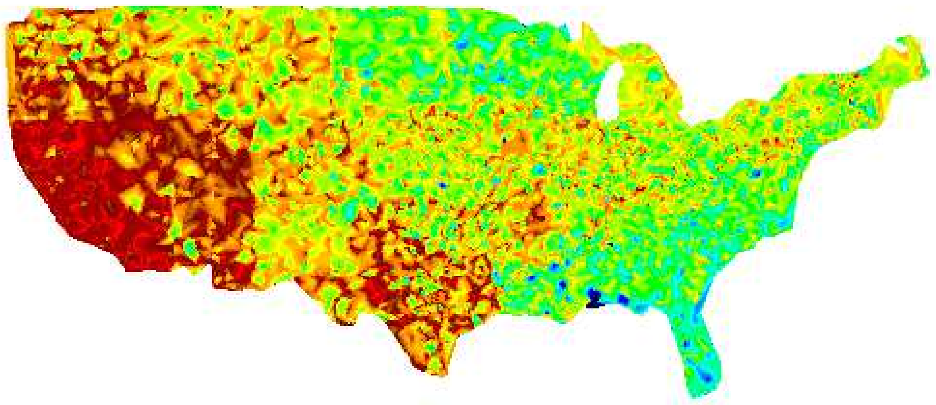

Figure 8.11: The Snapshot of U.S. Precipitation in July, 1997

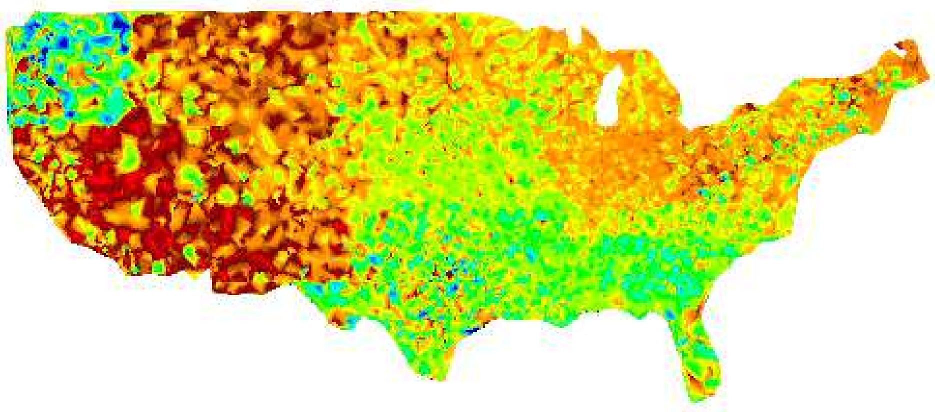

Figure 8.12: The Snapshot of U.S. Precipitation in October, 1997

Chapter 9

Conclusions

In this chapter we conclude the dissertation, and list some directions for future work in Section 9.1.

We propose a software architecture of constraint database systems, and describe the TAQS system in detail as an illustration of the implementation based on this architecture.

The experiments show that the transformation of temporal data into piecewise linear functions is highly accurate, and they can be easily represented and queried in constraint databases. We also approximate spatial data and represent it in a constraint database.

We also consider the approximate queries, query evaluation, and update. And we provide color bands display and isometric color animation to visualize the query results .

9.1 Future Work

For future work, one thing we are trying to do is to find more efficient piecewise linear approximation algorithms; that is, to find optimal algorithms with lower complexity, or to find approximate algorithms which are more closer to optimal solutions. Another research direction is to find better approximate evaluation algorithms that have tighter upper and lower bounds.

For spatiotemporal data, we currently just concentrate on the two-dimensional space. However, in the real world many problems are three-dimensional. Thus, displaying the spatial objects in three dimensional space is very necessary. Another issue is the controllability of the spatial objects. People would have the power to control the movement of the objects as they desired during the animation. And the animation system should have some certain predictability for the future according to the past and present information. Moreover, how to get the accurate TIN structure from the sampled points need also to be studied further.

Bibliography

[1] N. Adam and A. Gangopadhyay. *Database Issues in Geographic Information Systems.* Kluwer Academic Publishers, 1997.

[2] A. Aho and J. Ullman. Universality of Data Retrieval Languages. In *Proc. the 6th Symposium on Principles of Programming Languages, Texas*, pages 110–120, January 1979.

[3] A. Brodsky, V. Segal, J. Chen, and P. Exarkhopoulo. The CCUBE Constraint Object-Oriented Database System. In *Proc. ACM SIGMOD*, pages 577–579, 1999.

[4] M. Cai, D. Keshwani, and P. Revesz. Parametric Rectangles: A Model for Querying and Animating Spatiotemporal Databases. In *Proc. the 7th International Conference on Extending Database Technology*, LNCS 1777, pages 430–444. Springer, 2000.

[5] R. Chen, M. Ouyang, and P. Revesz. A Time Series Data Model. In *2001 Annual Meeting, Association of American Geographers*, New York, NY, 2001.

[6] R. Chen, M. Ouyang, and P. Revesz. Approximating Data in Constraint Databases. In *Proc. the 4th International Symposium on Abstraction, Reformulation and Approximation, Horseshoe Bay, Texas*, volume 1864 of *Lecture Notes in Artificial Intelligence*, pages 124–143. Springer-Verlag, July 2000.

[7] R. Chen and P. Revesz. Geo-Temporal Data Transformations and Visualization. In *Proc. the 1st International Conference on Geographic Information Science*, pages 240–242, Savannah, GA, October 2000.

[8] J. Chomicki and P. Revesz. Constraint-based interoperability of spatiotemporal databases. *Geoinformatica*, 3:3:211–243, 1999.

[9] C. Decleir, M. Hacid, and J. Kouloumdjian. A Database Approach for Modeling and Querying Video Data. In *the 15th International Conference on Data*

Engineering, Sydney, Australia, pages 6–13. IEEE Computer Society, March 1999.

[10] B. Dent. *Cartography Thematic Map Design.* McGraw-Hill, 1999.

[11] M. Flickner, H. Sawhney, and et al. Query by Image and Video Content: The QBIC System. In *IEEE Computer*, pages 23–32, September 1995.

[12] B. Furht. *Multimedia Systems and Technologies.* Kluwer Academic, 1996.

[13] W. Grosky, R. Jain, and R. Mehrotra. *The Handbook of Multimedia Information Management.* Prentice Hall, 1997.

[14] S. Grumbach, P. Rigaux, and L. Segoufin. The DEDALE System for Complex Spatial Queries. In *Proc. the ACM SIGMOD International Conference on Management of Data*, pages 213–224, Seattle, WA, 1998.

[15] S. Grumbach, P. Rigaux, and L. Segoufin. Manipulating Interpolated Data is Easier than You Thought. In *Proc. International Conference on Very Large Databases*, 2000.

[16] P. Kanellakis, G. Kuper, and P. Revesz. Constraint Query Languages. *Journal of Computer and System Sciences*, 51:26–52, 1995.

[17] P. Kanjamala, P. Revesz, and Y. Wang. MLPQ/GIS: A GIS Using Linear Constraint Databases. In C. S. R. Prabhu, editor, *Proc. the 9th COMAD International Conference on Management of Data*, pages 389–393. Tata McGraw Hill, 1998.

[18] S. Khoshafian and B. Baker. *Multimedia and Imaging Databases.* Morgan Kaufmann, 1996.

[19] G. Kowalski. *Information Retrieval Systems – Theory and Implementation.* Kluwer Academic Publishers, 1997.

[20] S. Magliveras and N. Memon. The Linear Complexity Profile of Cryptosystem PGM. In *Congressus Numerantium, Winnipeg, Canada*, volume 72, pages 51–60, January 1990.

[21] J. Massey. Shift-register Synthesis and BCH Decoding. In *IEEE Transactions on Information Theory*, volume 15, pages 122–127, January 1969.

[22] National Climatic Data Center (NCDC). *Monthly Precipitation Data for U.S. Cooperative & NWS Sites.* http://www.ncdc.noaa.gov/pub/data/coop-precip/.

[23] K. Nwosu, B. Thuraisingham, and B. Berra. *Multimedia Database Systems*. Kluwer Academic, 1996.

[24] Oracle. *Oracle8i Time Series User's Guide*. Oracle Corporation.

[25] J. Paredaens. Spatial Databases, the Final Frontier. In G. Gottlob and M.Y. Vardi, editors, *Proc. the 5th International Conference on Database Theory*, LNCS 893, pages 14–32. Springer-Verlag, 1995.

[26] B. Prabhakaran. *Multimedia Database Management Systems*. Kluwer Academic, 1997.

[27] P. Revesz. Constraint Databases: A Survey. In L. Libkin and B. Thalheim, editors, *Semantics in Databases*, LNCS 1358. Springer-Verlag, 1998.

[28] P. Revesz. The Evaluation and the Computational Complexity of Datalog Queries of Boolean Constraint Databases. *International Journal of Algebra and Computation*, 8(5):472–498, October 1998.

[29] P. Revesz. *Introduction to Constraint Databases*. Springer-Verlag, 2000.

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

[31] P. Revesz, R. Chen, P. Kanjamala, Y. Li, Y. Liu, and Y. Wang. The MLPQ/GIS Constraint Database System. In *Proc. the ACM SIGMOD International Conference on Management of Data, Dallas, Texas*, May 2000.

[32] P. Revesz, R. Chen, and M. Ouyang. A Software Architecture for Constraint Database Systems. In *Software Architectures, Components, and Frameworks*, submitted.

[33] P. Revesz, R. Chen, and M. Ouyang. Constraint Approximation and Querying of Relational Databases. *Journal of Constraints*, submitted.

[34] P. Revesz and Y. Li. MLPQ: A Linear Constraint Database System with Aggregate Operations. In *Proc. the 1st International Database Engineering and Applications Symposium*, 1997.

[35] J. Star and J. Estes. *Geographic Information Systems: An Introduction*. Prentice-Hall, Inc., 1989.

[36] U.S. Geological Survey. *Geographic Information Systems*. http://www.usgs.gov/research/gis/title.html.

[37] J. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I and II. Computer Science Press, 1988-1989.

[38] L. Vandeurzen, M. Gyssens, and D. Van Gucht. On the Desirability and Limitations of Linear Spatial Query Languages. In M. J. Egenhofer and J. R. Herring, editors, *Proc. the 4th International Symposium on Spatial Databases*, LNCS 951, pages 14–28, Berlin, 1995. Springer-Verlag.