ANALYSIS AND MODELING TECHNIQUES FOR GEO-SPATIAL AND SPATIO-TEMPORAL DATASETS

by

KULSAWASD JITKAJORNWANICH

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2014

To the memory of my grandmother, Hianghung Saetang, I miss you…

Acknowledgements

First of all, I would like to express my sincerest thanks to my professor Dr. Ramez Elmasri for his invaluable guidance that helps me get to this point. This day will never come without him. I also would like to thank him for being there for me in several unexpected dilemma situations I encountered during my PhD studies. These good and bad experiences make me realize how such a great person he is. Not only is he a knowledgeable/creative professor, but also warm-hearted Samaritan. It has been such a great time working and learning from him. It is definitely one of the great experiences in my life that I will never forget.

Next, I would like to thank all my committee members (Prof. Chengkai Li, Prof. David Levine, and Prof. Leonidas Fegaras) for their priceless advice and feedback that helped me perfect my research work. Also, I am grateful to Prof. John McEnery for his support and guidance, and for allowing me to work on a real-world hydrology project, and for educating me on hydrology concepts.

I also would like to thank all my friends from our lab (MAST), especially Upa Gupta and Sakthi Kumaran Shanmuganathan for our effective discussions that make a better work.

Finally, I would like to thank all my family members for their continuous support, especially my grandmother, who passed away a semester prior in August 23, 2013. She will always be in my heart.

April 17, 2014

Abstract

ANALYSIS AND MODELING TECHNIQUES FOR GEO-SPATIAL AND SPATIO-TEMPORAL DATASETS

Kulsawasd Jitkajornwanich, PhD

The University of Texas at Arlington, 2014

Supervising Professor: Ramez Elmasri

In recent years, spatio-temporal data has received a lot of attention and increasingly plays an important role in our everyday lives as we can witness from the fast-growing mobile technologies and its location-based application development. By spatio-temporal data, we mean data that is associated with specific spatial locations that change over time. For example, a cellphone or car with GPS will generate the object location at regular time intervals. Another example would be the track of a storm center as it moves. Spatio-temporal data could be thought of as a huge data warehouse, which contains hidden and meaningful information. However, to analyze the available spatio-temporal data directly from its original formats and locations is not easy because the data is often in a format that is difficult to analyze and is usually 'big'. Our research goals focus on spatio-temporal datasets and how to summarize, model, and conceptualize them for analysis and mining. Five main parts of this dissertation include: 1) spatio-temporal knowledge representation, 2) identifying meaningful concepts from raw data, 3) converting raw data to conceptual data, 4) analysis and mining of conceptual data, and 5) a general framework for big data analysis and mining.

In the first part of the dissertation, we look at the spatio-temporal datasets in general by considering spatio-temporal data semantics using techniques similar to those utilized in the “Semantic Web”. We work towards creating a spatio-temporal ontology framework, which can be used to represent and reason about spatio-temporal data. In the next three parts, we focus on the spatio-temporal datasets in a specific domain, which is rainfall precipitation data in the hydrology domain. However, the techniques and methodology that we use can be adapted to different types of hydrological data such as soil moisture, water level, etc., as well as other types of big spatio-temporal data. Therefore, in the final part, we propose a generalized framework for analyzing and mining big data in any given domain. The framework allows big data in a particular domain to be conceptually analyzed and mined by using ontologies and EER.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction

In recent years, spatio-temporal data has received a lot of attention and increasingly plays an important role in our everyday lives, as we can witness from the fast-growing mobile technologies and their location-based application development. Consequently, spatio-temporal data is widely available and used in an increasing number of applications. By spatio-temporal data, we mean data that is associated with specific spatial locations that change over time. For example, a cellphone or car with GPS will generate the object location at regular time intervals. Another example would be the track of a storm center as it moves. The enormous amount of spatio-temporal data that is being generated every day could be thought of as a huge data warehouse, which contains hidden and meaningful information. Our work is a contribution towards extracting this hidden, meaningful information from large amount of spatio-temporal data.

To analyze the available spatio-temporal data directly from its original formats and locations is not easy because the data is often in a format that is difficult to analyze and is usually 'big'. Our research goals focus on spatio-temporal datasets and how to summarize, model, and format them for analysis and mining. There are five main parts of this dissertation:

1. Spatio-temporal knowledge representation (see Chapter 2)
2. Identifying meaningful concepts from raw data (see Chapter 3)
3. Converting raw data to conceptual data (see Chapter 4)
4. Analysis and mining of conceptual data (see Chapter 5)
5. A general framework for big data analysis and mining (see Chapter 6)

In the first part of the dissertation, we look at spatio-temporal datasets in general by considering spatio-temporal data semantics using techniques similar to those utilized in the “Semantic Web” [7][13]. We work towards creating a spatio-temporal ontology framework, which can be used to represent and reason about spatio-temporal data. We first formalize a spatial ontology by specifying the concepts and operations relevant to spatial data. We then apply a technique called “light-weight” [22][21] (which will be discussed further in the dissertation), so that we can integrate our spatial ontology with a well-known temporal ontology [2][3]. The resulting spatio-temporal ontologies are eventually implemented in an ontology editor, which in our case, is Protégé OWL 3.4.4 [30]. The spatio-temporal aspects of the data can be reasoned, inferenced, and queried using our framework. In the remaining parts of the dissertation, we focus on spatio-temporal datasets in a specific domain, which is rainfall precipitation data in the hydrology domain.

In the second part of the dissertation, we examine the structure and format of the raw rainfall data and discuss the process to prepare it for analysis and mining. Because no two different types of spatio-temporal data, even in the same domain, have the same characteristics and concepts, the conceptual model that we are going to come up with must make sense to the domain experts, have potential in effective implementation, and be able to provide better and more robust analysis and mining beyond traditional methods. Two main processes are conducted: investigating raw rainfall data description and structure, and studying rainstorm concepts. Consequently, we model the meaningful concepts of rainstorms using formalization. Three storm configurations are formalized: local storm, hourly storm, and overall storm, which will be discussed further in the dissertation.

In the third part, after the model is defined through formalization, we discuss the algorithms designed to convert the raw rainfall data into the formalized rainfall concepts. Two different approaches were developed: CUAHSI-based and MapReduce-based. Both share the same goal, which is to identify meaningful storm concepts, but with different focuses. The first approach focuses on converting raw rainfall data that follows the CUAHSI standard, which is based on the standard database schema called “CUAHSI ODM” to store hydrological data in a relational database. Both input and output are processed and stored in a relational database. Although there are benefits from this approach, including easy integration with CUAHSI APIs, user-friendly analysis/mining through SQL, the main drawback is the slow performance. As a result, the second approach is proposed with the focus on drastically improved performance for the data conversion algorithms. In this approach, every component of the storm identification algorithms are re-designed starting from the original structure of raw data to applying the most recent distributing computing technology (namely MapReduce and Hadoop®) to speed up the performance. The storm outputs are verified and the performance of each approach is measured. The conceptual storm data are identified and stored in a relational database. The size of conceptual storm data is significantly reduced when compared to the size of the raw data, and can easily be analyzed and mined.

In the fourth part of the dissertation, we show how identified storm outputs can be analyzed and mined. The analysis and mining tasks can be divided into two parts: 1) traditional hydrology analysis and 2) more general storm analysis and mining, which are more diverse and mostly have to do with our proposed concept of “overall” characteristics of the storm, including speed, track, total rainfall, coverage, boundary, etc. In the traditional hydrology analysis part, we examine characteristics of storms at a particular location (location-based) by considering [36][50][39][80]: 1) storm statistical properties, 2) relationships between/among characteristics of storms, and 3) focusing on extreme precipitation values of storms. In the more general storm analysis and mining part, we show some examples of how more robust/flexible storm analysis and mining can be done on our storm data. This also includes a storm visualization tool to visualize how an overall storm is formed and moves over time.

Although the techniques and methodology that we use are for big raw rainfall data, it can be adapted to different types of hydrological data such as soil moisture, water level, etc., as well as other types of big spatio-temporal data in other application domains. Therefore, we propose a generalized framework for conceptual analysis and mining of big data using ontologies and EER in the fifth part.

In the conclusion chapter, we summarize our contributions, and discuss future work. Thus, the main contributions of our research are:

1. Developing a spatial ontology and integrating it with a temporal ontology.
2. Proposing formalized storm concepts that enable easier analysis and mining of raw rainfall data.
3. Developing efficient algorithms to convert raw rainfall data into meaningful storm concepts, using the map-reduce paradigm.
4. Applying analysis and mining techniques to conceptual storm data.
5. Developing a framework that can be applied to other types of big spatio-temporal raw data to reduce the data and convert it into concepts for better analysis and mining.

Chapter 2

Spatio-Temporal Knowledge Representation

The goal of this chapter is to allow spatio-temporal data semantics of a dataset to be analyzed, modeled, and reasoned about by using ontology [10][20]. Ontology is an excellent tool in knowledge modeling. With the reasoning, inference, and representation mechanisms associated with an ontology, it becomes possible that systems with different definitions of the same concepts can interoperate with each other. In addition, a nearly complete description of concepts in a particular area of knowledge becomes readily available for interested users. In this chapter, we develop a spatio-temporal ontology framework, which can be used to represent and reason about spatio-temporal data. Figure 2-1 shows an overview of our spatio-temporal ontology framework.

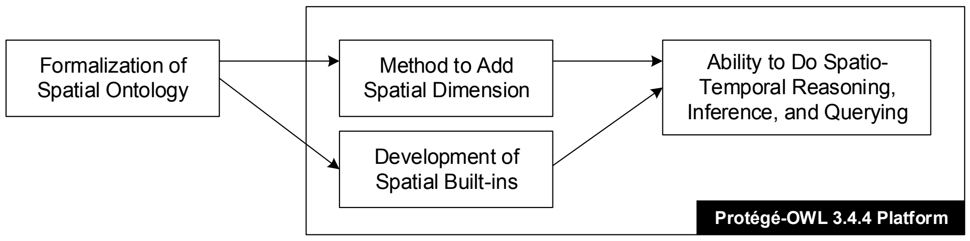

Formalization of Spatial Ontology
Method to Add Spatial Dimension
Development of Spatial Built-ins
Ability to Do Spatio-Temporal Reasoning, Inference, and Querying
Protégé-OWL 3.4.4 Platform

Figure 2-1 Overview of Our Spatio-Temporal Ontology Framework

We first formalize a spatial ontology in Section 2.1. We then show how the defined spatial ontology can be realized in Protégé in Section 2.2. Since the spatial ontology is formalized based on OpenGIS®, we also show that OpenGIS® can actually capture spatial concepts in Section 2.3, by proving that OpenGIS® SQL operations are complete with respect to the full spatial ontology operations. Finally, related work is discussed in Section 2.4.

2.1 Spatial Ontology Formalization

In this section, we first discuss the motivation of spatial ontology formalization and its comparison to temporal ontology as it is closely related and often analyzed along with spatial ontology. In fact, spatial operations can be considered as temporal operations with an additional dimension. Additionally, we will eventually embed temporal ontology into our spatial ontology to create spatio-temporal ontology. We then give a formal specification of spatial ontology covering spatial object definitions and spatial operation definitions.

2.1.1 Background and Motivation

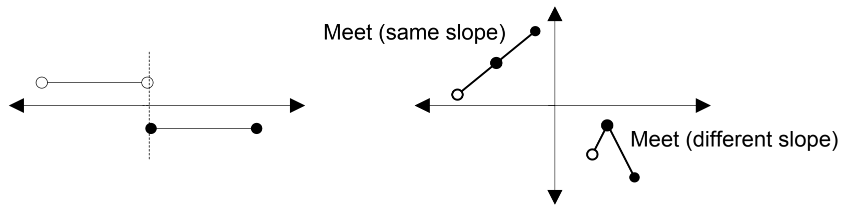
Representing spatial-related knowledge is a basic problem in many applications, such as GIS and map applications. In the past years, work on spatial ontologies has focused on two main areas: spatial database integration [8][9][12] and spatial ontology creation [5][29][11][14]. In spatial database integration, a spatial ontology is used as a tool to integrate different spatial databases. In spatial ontology creation, there are two different major approaches. First, by analyzing a collection of existing spatial databases and methodologies, a spatial ontology model is defined based on those databases [5]. However, this leads to the problem that the created spatial ontology will be limited to those databases and consequently will not be sufficient to be a standard for representing a complete formal spatial ontology. The second approach in spatial ontology creation is to define a complete spatial ontology model. For example, in [29], they propose to create a spatio-temporal ontology based on the MADS model, which allows a regular database to model spatial and temporal characteristics [24][25]. However, this approach has not been materialized in an implemented system, and there is no formal specification of spatial ontology developed from this approach. In addition, it is limited to the polygon data type only. Thus, the complete set of operations among point, line, and polygon is lacking. Finally, in [11][14], they propose using the RCC8 calculus [26] for spatial reasoning on regions, but they do not propose a complete spatial ontology.

Many researchers have worked in the area of temporal and spatial ontology [19][29][8][5][9][18][22][13][6][12][1][4]. A specification of temporal ontology was introduced in [13]. It clearly discussed temporal ontology formalization, and comprehensively defines temporal concepts and operations. It is based on the temporal logic developed by Allen [2][3]. The following is an example of the *Meet* operation between two time intervals formalized by [13], assuming that T_1, T_2 are two time intervals and t is a time instant.

$$Meet(T_1,T_2) \equiv (\exists t)[ends(t,T_1) \wedge begins(t,T_2)]$$

A complete formalization of ontology forms the basis and reference for ontology implementation. In addition, since the temporal ontology specification in [13] was intended to capture all temporal reasoning, it is gradually becoming the standard for temporal ontology specification.

Although spatial concepts and operations have been specified in many works [23][24][25][28][31][86], there are few attempts at specifying a complete formal ontology for spatial concepts. Spatial operations are more complex than temporal operations, and can be defined over multiple dimensions, especially two and three dimensions, whereas temporal operations are only on one dimension. Figure 2-2 shows how the *Meet* operation is different in one dimension and two dimensions. Additionally, temporal operations have only two directions (before and after) whereas for two dimensional spatial operations, there are continuous directions along 360º of a two dimensional space. Figure 2-3 shows eight directions, at 45º intervals. (East is 0º, north is 90º, west is 180º, etc.)



a) Temporal operation (1-D)   b) Spatial operation (2-D)

Figure 2-2 Comparison between Meet Operation in 1-D and 2-D

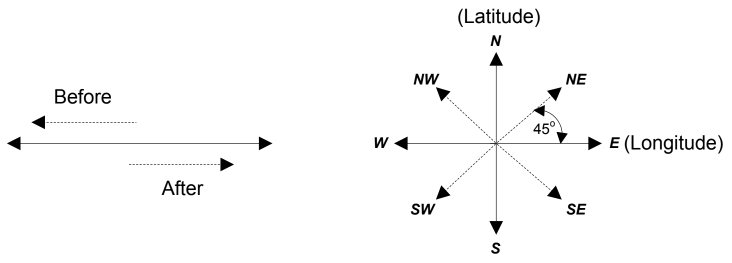


Figure 2-3 Difference of Direction between Temporal Operation (1-D, left) and Spatial Operation (2-D, right)

2.1.2 Formalization of Spatial Object Definitions

We formalize spatial ontology based on OpenGIS® [23] (as it is a well-known standard for spatial-related concepts) with some modifications. Our proposed spatial ontology consists of two parts [17]: spatial object definitions and spatial operation definitions. In [23], a geometry class hierarchy is proposed for 2-D spatial objects. The hierarchy shown in Figure 2-4 is based on the one in [23], with some minor modifications to allow our formalization.

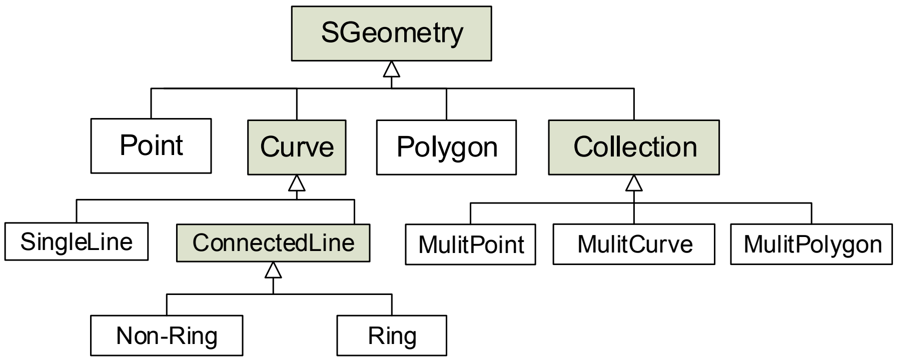


Figure 2-4 Geometry Class Hierarchy

Considering the leaf nodes in the hierarchy of Figure 2-4, the geometry objects can be categorized into 8 types as shown in Figure 2-5.

1. Point (*p*)
2. Single Line (*sl*)
3. Connected Line (*cl*): Non-Ring (*nr*)
4. Connected Line (*cl*): Ring (r)
5. Polygon (*a*)
6. MultiPoint (*mp*)
7. MultiCurve (*mc*)
8. MultiPolygon (*ma*)

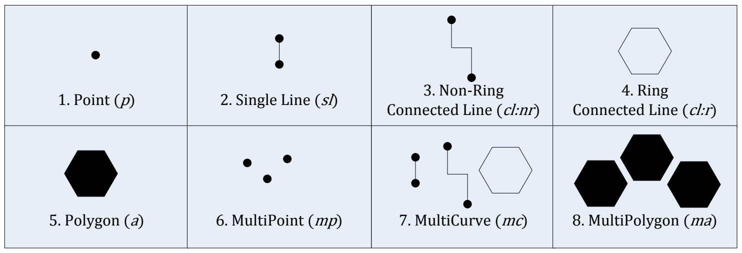


Figure 2-5 Types of Geometry Object in 2-D Space

In order to be used in the ontology, the spatial object definitions of all geometry object types in two-dimensional space (see Figure 2-4 and 2-5) have to be defined. For simplicity, we shall assume the 2-D coordinate system based on longitude and latitude, although the formalization can be adapted to other 2-D coordinate systems.

2.1.2.1 Point (*p*)

Point can be defined by longitude (*x*) and latitude (*y*).

$- Point(p) \equiv (x, y)\ where\ x = longitude\ and\ y = latitude.$

2.1.2.2 Single Line (*sl*)

Single line can be defined by any two points.

$- SingleLine(sl) \equiv (p_1, p_2)\ where\ p_1, p_2\ are\ points\ and\ p_1 \neq p_2.$

Single line also includes some unary operations, such as *Slope(m,sl)* and *Distance(d,sl),* which are defined in Section 2.1.3

2.1.2.3 Connected Line (*cl*): Non-Ring (*nr*)

Non-ring connected line can be defined by a sequence of points $(p_1, p_2, \dots, p_N)$, $N \geq 3$, and $p_i \neq p_j\ for\ i \neq j$, which in turn defines a sequence of single lines $(sl_1, sl_2, \dots, sl_{N-1})$ and each $sl_i = (p_i, p_{i+1})$. In other words, the sequence must contain at least two single lines and each p_i is connected to p_{i+1} for $i < N$.

$- NonRingConnectedLine(clnr)$

$\equiv (p_i | 1 \leq i \leq N, N \geq 3, p_i \neq p_j\ for\ i \neq j, sl_i = (p_i, p_{i+1}), Meet(sl_i, sl_{i+1}) = true$

$for\ i = 1,2, \dots, N-1)$

2.1.2.4 Connected Line (*cl*): Ring (*r*)

Ring connected line is defined as a sequence of points $(p_1, p_2, \dots, p_N)$, $N \geq 3$, and $p_i \neq p_j\ for\ i \neq j$, which in turn defines a sequence of single lines $(sl_1, sl_2, \dots, sl_N)$ where each $sl_i = (p_i, p_{i+1})$ for *i* = 1,2,…,*N-1* and $sl_N = (p_N, p_1)$. However, the area inside the ring is not part of the connected line.

$- RingConnectedLine(clr)$

$\equiv (p_i | 1 \leq i \leq N, N \geq 3, p_i \neq p_j\ for\ i \neq j, sl_i = (p_i, p_{i+1})\ for\ i = 1,2, \dots, N-1\ and$

$sl_N = (p_N, p_1), Meet(sl_i, sl_{i+1}) = true\ for\ i = 1,2,\dots,N-1, Meet(sl_N, sl_1) = true)$ $and\ area(clr) = 0$.

2.1.2.5 Polygon (*a*)

Polygon is defined as a sequence of points $(p_1, p_2, \dots, p_N)$, $N \geq 3$, and $p_i \neq p_j\ for\ i \neq j$, which in turn defines a sequence of single lines $(sl_1, sl_2, \dots, sl_N)$ where each $sl_i = (p_i, p_{i+1})$ for *i* = 1,2,…,*N-1* and $sl_N = (p_N, p_1)$, and the area within is part of the polygon.

$- Polygon(a)$

$\equiv (p_i | 1 \leq i \leq N, N \geq 3, p_i \neq p_j\ for\ i \neq j,\ sl_i = (p_i, p_{i+1})\ for\ i = 1,2,\dots,N-1\ and$ $sl_N = (p_N, p_1), Meet(sl_i, sl_{i+1}) = true\ for\ i = 1,2,\dots,N-1, Meet(sl_N, sl_1) = true,$ $Cross(sl_i, sl_j) = false\ for\ all\ i,j\ where\ i \neq j)\ and\ area(a) \neq 0.$

2.1.2.6 MultiPoint (*mp*)

MultiPoint can be defined as a set of two or more points.

$- MultiPoint(mp) \equiv \{p_i | 1 \leq i \leq N, N \geq 2\}$

2.1.2.7 MultiCurve (*mc*)

MultiCurve can be defined as a set of single line(s) or connected line(s).

$- MultiCurve(mc) \equiv \{c_i | c_i = sl \vee c_i = clnr \vee c_i = clr; 1 \leq i \leq N, N \geq 2\}$

2.1.2.8 MultiPolygon (*ma*)

MultiPolygon is defined as a set of two or more polygons.

$- MultiPolygon(ma) \equiv \{a_i | 1 \leq i \leq N, N \geq 2\}$

2.1.3 Formalization of Spatial Operation Definitions

As mentioned earlier, spatial operations are much more complex than temporal operations. Therefore, we will only focus on a subset of two-dimensional spatial operations. Considering spatial relationships, there are six major types as follows: 1) Point VS Point, 2) Point VS Line, 3) Point VS Polygon, 4) Line VS Line, 5) Line VS

Polygon, and 6) Polygon VS Polygon. Only spatial relationships between Point and Line (i.e., 1), 2), and 4)) are covered. We can divide these operation definitions into six different categories depending on the pair of spatial objects.

1. Point and Point
2. Point and Single Line
3. Point and Connected Line
4. Single Line and Single Line
5. Single Line and Connected Line
6. Connected Line and Connected Line

2.1.3.1 Point and Point

There are two relationships between point and point.

2.1.3.1.1 Equal. Two points are equal if and only if they have exactly the same longitude and latitude respectively.

$- Equal(p_1, p_2) \equiv p_1.x = p_2.x \land p_1.y = p_2.y$

2.1.3.1.2 Disjoint. Two points are disjoint when they are not equal.

$- Disjoint(p_1, p_2) \equiv \neg Equal(p_1, p_2)$

2.1.3.2 Point and Single Line

Considering point and single line, there are three possible spatial operations: *Endpoint, Ontheline,* and *Disjoint*.

2.1.3.2.1 Endpoint. Endpoint is a relationship between point and single line. In the formalization, we sometimes need to distinguish unambiguously the two endpoints so we define two operations on a line: *EndpointNe* and *EndpointSw*. North and south will have precedence in distinguishing the endpoints and east and west will be used only if a line is horizontal. That is, the endpoint of the line with higher latitude will be classified as

EndpointNe regardless of whether the longitude is either east or west. Figure 2-6(a) shows some examples of how the endpoints are categorized.

A point *p* will be an endpoint of single line *sl* if and only if *p* is equal to either one of the single line endpoints.

$- Endpoint(p, sl) \equiv p = sl.p_1 \vee p = sl.p_2$

$- EndpointNe(p, sl) \equiv$

$Endpoint(p, sl) \wedge (\exists p_1)[Endpoint(p_1, sl) \wedge p \neq p_1 \wedge \big((sl.p.y > sl.p_1.y) \vee$

$(sl.p.y = sl.p_1.y \wedge sl.p.x > sl.p_1.x)\big)]$

$- EndpointSw(p, sl) \equiv$

$Endpoint(p, sl) \wedge (\exists p_1)[Endpoint(p_1, sl) \wedge p \neq p_1 \wedge \big((sl.p.y < sl.p_1.y) \vee$

$(sl.p.y = sl.p_1.y \wedge sl.p.x < sl.p_1.x)\big)]$

The relationship between *EndpointNe* and *EndpointSw* can be specified as follows:

$- EndpointNe(p_1, sl) \wedge EndpointSw(p_2, sl) \rightarrow$

$[(sl.p_1.y_1 > sl.p_2.y_2) \vee (sl.p_1.y_1 = sl.p_2.y_2 \wedge sl.p_1.x_1 > sl.p_2.x_2)]$

We can now define two unary operations on single line, *Slope* and *Distance*, as follows (these are used in our specification of some of operations):

$- Slope(m, sl) \equiv$

$(\exists p_1, p_2)[EndpointNe(p_1, sl) \wedge EndpointSw(p_2, sl) \wedge m = \left[\frac{p_1.y - p_2.y}{p_1.x - p_2.x}\right]]$

$- Distance(d, sl) \equiv$

$(\exists p_1, p_2)[EndpointNe(p_1, sl) \wedge EndpointSw(p_2, sl) \wedge$

$d = [squareroot((p_1.y - p_2.y)^2 + (p_1.x - p_2.x)^2)]]$

2.1.3.2.2 Ontheline. Ontheline (see Figure 2-6(d)) is a relationship between point and single line. If a point *p* is on the single line *sl* then the slope of the point *p* to one of the endpoints of the single line must be equal to the slope of the single line. In addition, the point *p* has to fall on the single line. We also have a function to create a single line given two points: *CreateSingleLine(sl,x,y)* where *sl* is the created single line and *x*,*y* are the two endpoints.

$- Ontheline(p, sl) \equiv$

$Slope(m_1, sl) \wedge (\exists p_1)[CreateSingleLine(sl_1, p, sl.p_1)] \wedge Slope(m_2, sl_1) \wedge$

$m_1 = m_2 \wedge (sl.p_1.x_1 \leq p.x \leq sl.p_2.x_2 \vee sl.p_2.x_2 \leq p.x \leq sl.p_1.x_1)$

An endpoint is also considered to be on the line.

$- Endpoint(p, sl) \rightarrow Ontheline(p, sl)$

2.1.3.2.3 Disjoint. A point *p* and single line *sl* are disjoint when the point *p* is not on the single line *sl*.

$- Disjoint(p, sl) \equiv \neg Ontheline(p, sl)$

2.1.3.3 Point and Connected Line

In the following, we will discuss the formalization of spatial operations between point and connected line. Considering point and connected line, there are four possible spatial operations: *Endpoint*, *InteriorEndpoint, Ontheline,* and *Disjoint*.

2.1.3.3.1 Endpoint. Endpoint (see Figure 2-6(j)) is one of the relationships between point and connected line. In the formalization, we also need to distinguish unambiguously the two endpoints (same as operations between point and single line) so we define two operations on a line: *EndpointNe* and *EndpointSw*. The directions will be considered in the same fashion.

A point *p* will be an endpoint of connected line *cl* if and only if *p* is equal to p_1 or p_N of the connected line *cl*.

$- Endpoint(p, cl) \equiv p = cl.p_1 \vee p = cl.p_N$

$- EndpointNe(p, cl) \equiv$

$Endpoint(p, cl) \wedge (\exists p_1)[Endpoint(p_1, cl) \wedge p \neq p_1 \wedge$

$\big((cl.p.y > cl.p_1.y) \vee (cl.p.y = cl.p_1.y \wedge cl.p.x > cl.p_1.x)\big)]$

$- EndpointSw(p, cl) \equiv$

$Endpoint(p, cl) \wedge (\exists p_1)[Endpoint(p_1, cl) \wedge p \neq p_1 \wedge$

$\big((cl.p.y < cl.p_1.y) \vee (cl.p.y = cl.p_1.y \wedge cl.p.x < cl.p_1.x)\big)]$

The relationship between *EndpointNe* and *EndpointSw* can be specified as follows:

$- EndpointNe(p_1, cl) \wedge EndpointSw(p_2, cl) \rightarrow$

$$[(cl.p_1.y_1 > cl.p_2.y_2) \vee \begin{pmatrix} cl.p_1.y_1 = cl.p_2.y_2 \\ \wedge\, cl.p_1.x_1 > cl.p_2.x_2 \end{pmatrix}]$$

2.1.3.3.2 InteriorEndpoint. InteriorEndpoint (see Figure 2-6(k)) is another relationship between point and connected line when a point falls in the joint between two single lines of connected line.

$- InteriorEndpoint(p, cl) \equiv (\exists p_1 \in \{cl.p_i; i \neq 1, N\})[p = p_1]$

2.1.3.3.3 Ontheline. A point *p* is on the connected line *cl* if and only if *p* is on one of the single lines sl_i of connected line *cl* (see Figure 2-6(l)).

$- Ontheline(p, cl) \equiv (\exists sl \in \{cl.sl_i; sl_i = (p_i, p_{i+1})\})[Ontheline(p, sl)]$

The relationship among these three operations is as follows:

$- Endpoint(p, cl) \vee InteriorEndpoint(p, cl) \rightarrow Ontheline(p, cl)$

2.1.3.3.4 Disjoint. A point *p* and a connected line *cl* are disjoint when the point p is not on the connected line cl.

$- Disjoint(p, cl) \equiv \neg\, Ontheline(p, cl)$

2.1.3.4 Single Line and Single Line

There are seven main relationships between single lines.

2.1.3.4.1 Equal. Two single lines are equal if and only if they have exactly the same points.

$- Equal(sl_1, sl_2) \equiv$

$(\exists p_1, p_2)[EndpointNe(p_1, sl_1) \wedge EndpointNe(p_1, sl_2) \wedge$

$EndpointSw(p_2, sl_1) \wedge EndpointSw(p_2, sl_2)]$

2.1.3.4.2 Meet. In our formalization, two single lines meet when they share one endpoint. We can further divide the operation into two more cases: *MeetSameSlope* and *MeetDiffSlope* as follows (see Figure 2-6(b,c)).

$- Meet(sl_1, sl_2) \equiv$

$(\exists p \in \{sl_1.p_1, sl_1.p_2, sl_2.p_1, sl_2.p_2\})\ [Endpoint(p, sl_1) \wedge Endpoint(p, sl_2)]$

$- MeetSameSlope(sl_1, sl_2) \equiv$

$Meet(sl_1, sl_2) \wedge Slope(m_1, sl_1) \wedge Slope(m_2, sl_2) \wedge m_1 = m_2$

$- MeetDiffSlope(sl_1, sl_2) \equiv$

$Meet(sl_1, sl_2) \wedge Slope(m_1, sl_1) \wedge Slope(m_2, sl_2) \wedge m_1 \neq m_2$

2.1.3.4.3 Cross. In our formalization, two single lines cross when their slopes are different and the intersecting point fall on both lines (see Figure 2-6(e)). To make formalization easier, we will first define *Between* and *ProperBetween* relations as follows. We also have an operation *IntersectPoint(p,sl$_1$,sl$_2$)* that returns the point of intersection *p* between two single lines *sl$_1$*, *sl$_2$*, if the two lines do not have the same slope.

$- Between(x, x_1, x_2) \equiv x_1 \leq x \leq x_2 \vee x_2 \leq x \leq x_1$

$- ProperBetween(x, x_1, x_2) \equiv x_1 < x < x_2 \vee x_2 < x < x_1$

$- Cross(sl_1, sl_2) \equiv$

$Slope(m_1, sl_1) \wedge Slope(m_2, sl_2) \wedge m_1 \neq m_2 \wedge IntersectPoint(p, sl_1, sl_2) \wedge$

$Between(p.x, sl_1.p_1.x, sl_1.p_2.x) \wedge Between(p.x, sl_2.p_1.x, sl_2.p_2.x)$

$- ProperCross(sl_1, sl_2) \equiv$

$Slope(m_1, sl_1) \wedge Slope(m_2, sl_2) \wedge m_1 \neq m_2 \wedge IntersectPoint(p, sl_1, sl_2) \wedge$

$ProperBetween(p.x, sl_1.p_1.x, sl_1.p_2.x) \wedge ProperBetween(p.x, sl_2.p_1.x, sl_2.p_2.x)$

In this operation, we have one special case when the intersecting point is also one of the endpoints of the line. In other words, one of the endpoints of a line lies on the other line. We will call this case of operation *TCross* (see Figure 2-6(f)).

$- TCross(sl_1, sl_2) \equiv$

$Cross(sl_1, sl_2) \wedge (\exists p)[(Endpoint(p, sl_1) \wedge Ontheline(p, sl_2) \wedge \neg Endpoint(p, sl_2))$

$\vee (Endpoint(p, sl_2) \wedge Ontheline(p, sl_1) \wedge \neg Endpoint(p, sl_1))]$

2.1.3.4.4 Overlap. In our formalization, two single lines overlap if and only if their slopes are equal and there is one endpoint of one line lying in another line but not at the endpoint (see Figure 2-6(g)).

$- Overlap(sl_1, sl_2) \equiv$

$Slope(m_1, sl_1) \wedge Slope(m_2, sl_2) \wedge m_1 = m_2 \wedge$

$(\exists p)[\big(Endpoint(p, sl_1) \wedge Ontheline(p, sl_2) \wedge \neg Endpoint(p, sl_2)\big) \vee$

$(Endpoint(p, sl_2) \wedge Ontheline(p, sl_1) \wedge \neg Endpoint(p, sl_1))]$

2.1.3.4.5 Within. For Within operation, we can divide it into two more types: *CompleteWithin* (can also be called *ProperWithin*) and *SharedEndpointWithin* (see Figure 2-6(h,i)).

$- Within(sl_1, sl_2) \equiv (\forall p)[Endpoint(p, sl_1) \wedge Ontheline(p, sl_2)]$

$- CompleteWithin(sl_1, sl_2) \equiv$

$Within(sl_1, sl_2) \wedge (\forall p)[\neg(Endpoint(p, sl_1) \wedge Endpoint(p, sl_2))]$

$- SharedEndpointWithin(sl_1, sl_2) \equiv$

$Within(sl_1, sl_2) \wedge (\exists p)[Endpoint(p, sl_1) \wedge Endpoint(p, sl_2)]$

$- CompleteWithin(sl_1, sl_2) \vee SharedEndpointWithin(sl_1, sl_2) \equiv Within(sl_1, sl_2)$

2.1.3.4.6 Contain. Contain operation is defined as a reverse Within operation as follows:

$- Contain(sl_1, sl_2) \equiv Within(sl_2, sl_1)$

$- CompleteContain(sl_1, sl_2) \equiv \; CompleteWithin(sl_2, sl_1)$

$- SharedEndpointContain(sl_1, sl_2) \equiv SharedEndpointWithin(sl_2, sl_1)$

$- CompleteContain(sl_1, sl_2) \vee SharedEndpointContain(sl_1, sl_2) \equiv Contain(sl_1, sl_2)$

2.1.3.4.7 Disjoint. Two single lines are disjoint when they meet one of the following conditions: 1) if both single lines have the same slope, then all endpoints of each line do not fall on another line, or 2) if both single lines have different slopes, their intersecting point are not on both lines.

$- Disjoint(sl_1, sl_2) \equiv$

$$(Slope(m_1, sl_1) \wedge Slope(m_2, sl_2) \wedge m_1 = m_2 \wedge$$

$$(\forall p)\begin{bmatrix} Endpoint(p, sl_1) \wedge \neg Ontheline(p, sl_2) \wedge \\ Endpoint(p, sl_2) \wedge \neg Ontheline(p, sl_1) \end{bmatrix}) \vee$$

$$(Slope(m_1, sl_1) \wedge Slope(m_2, sl_2) \wedge m_1 \neq m_2 \wedge$$

$$[IntersectPoint(p, sl_1, sl_2) \wedge \neg Ontheline(p, sl_2) \wedge \neg Ontheline(p, sl_1)])$$

The relationships among these seven operations are as follows.

MeetDiffSlope and TCross operations are also considered as Cross operation.

$- MeetDiffSlope(sl_1, sl_2) \; \vee TCross(sl_1, sl_2) \rightarrow Cross(sl_1, sl_2)$

Equal, Meet, Cross, Overlap, Within, and Contain operations are also considered as *Intersect*.

$- Equal(sl_1, sl_2) \vee Meet(sl_1, sl_2) \vee Cross(sl_1, sl_2) \vee Overlap(sl_1, sl_2) \vee Within(sl_1, sl_2) \vee$

$Contain(sl_1, sl_2) \rightarrow Intersect(sl_1, sl_2)$

$- Disjoint(sl_1, sl_2) \rightarrow \neg Intersect(sl_1, sl_2)$

CompleteWithin and CompleteContain operations are also considered as Overlap.

$- CompleteWithin(sl_1, sl_2) \vee CompleteContain(sl_1, sl_2) \rightarrow Overlap(sl_1, sl_2)$

Equal, SharedEndpointWithin, SharedEndpointContain operations are also considered as Meet.

$-Equal(sl_1, sl_2) \vee SharedEndpointWithin(sl_1, sl_2) \vee SharedEndpointContain(sl_1, sl_2)$
$\rightarrow Meet(sl_1, sl_2)$

Equal operation is also considered as SharedEndpointWithin and SharedEndpointContain operations.

$-Equal(sl_1, sl_2) \rightarrow SharedEndpointWithin(sl_1, sl_2) \wedge SharedEndpointContain(sl_1, sl_2)$

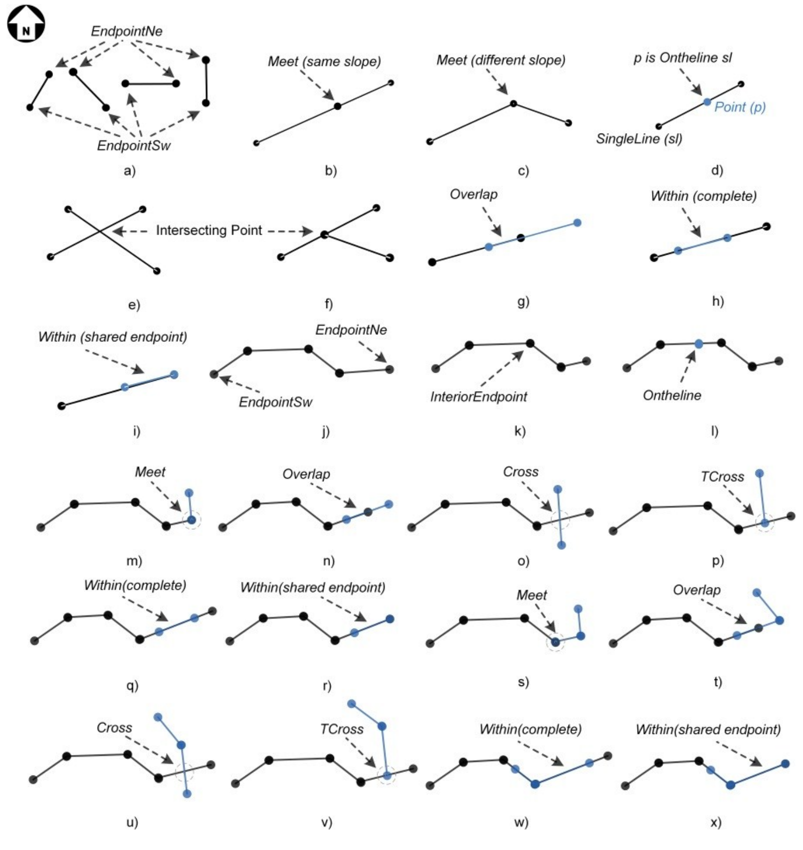


Figure 2-6 Types of Spatial Operations

2.1.3.5 Single Line and Connected Line

There are five main relationships between single line and connected line.

2.1.3.5.1 Meet. A single line *sl* meets a connected line *cl* when they share one of the endpoints (see Figure 2-6(m)).

$- Meet(sl, cl) \equiv (\exists p)[Endpoint(p, sl) \land Endpoint(p, cl)]$

2.1.3.5.2 Cross. A single line *sl* and a connected line *cl* are crossed if and only if there is a single line sl_i of connected line *cl* crossing the single line *sl* (see Figure 2-6(o)).

$- Cross(sl, cl) \equiv (\exists sl_1 \in cl.sl_i)[Cross(sl, sl_1)]$

A special case of cross is when one of the endpoints of one line lies on another line but not at the endpoint (see Figure 2-6(p)).

$- TCross(sl, cl) \equiv$

$Cross(sl, cl) \land (\exists p)[(Endpoint(p, sl) \land Ontheline(p, cl) \land \neg Endpoint(p, cl)) \lor$

$(Endpoint(p, cl) \land Ontheline(p, sl) \land \neg Endpoint(p, sl))]$

2.1.3.5.3 Overlap. A single line *sl* and connected line *cl* are overlapped when single line *sl* is overlapped or equal to one of the single line sl_i of connected line *cl* (see Figure 2-6(n)).

$- Overlap(sl, cl) \equiv (\exists sl_1 \in cl.sl_i)[Overlap(sl, sl_1) \lor Equal(sl, sl_1)]$

2.1.3.5.4 Within. A single line *sl* is completely within a connected line *cl* if and only if a single line *sl* is completely within or shared-endpoint within a single line of connected line *cl* but does not share any endpoint with the connected line *cl* (see Figure 2-6(q)).

$- CompleteWithin(sl, cl) \equiv$

$(\exists sl_1 \in cl.sl_i)[CompleteWithin(sl, sl_1) \lor SharedEndpointWithin(sl, sl_1)] \land$

$(\forall p)[\neg(Endpoint(p, sl) \land Endpoint(p, cl))]$

A single line *sl* is considered as shared-endpoint within connected line *cl* if and only if they share an endpoint and a single line *sl* is shared-endpoint within single line sl_1 or sl_{N-1} of connected line (see Figure 2-6(r)).

$- SharedEndpointWithin(sl, cl) \equiv$

$(\exists sl_1 \in cl.sl_i, i = 1, N-1)[SharedEndpointWithin(sl, sl_1)] \wedge$

$(\exists p)[Endpoint(p, sl) \wedge Endpoint(p, cl)]$

2.1.3.5.5 Disjoint. A single line *sl* and a connected line *cl* are disjoint when the single *sl* is disjoint from all single lines of the connected line *cl*.

$- Disjoint(sl, cl) \equiv (\forall sl_1 \in cl.sl_i)[Disjoint(sl, sl_1)]$

2.1.3.6 Connected Line and Connected Line

There are seven main relationships between connected lines.

2.1.3.6.1 Equal. Two connected lines are equal if and only if they have exactly the same list of ordered points.

$- Equal(cl_1, cl_2) \equiv$

$\left(\forall p_1 \in cl_1.p_i, p_2 \in cl_2.p_j\right)[(p_1 = p_2; if\ i = j) \wedge$

$(EndpointNe(p_1, cl_1) \wedge (EndpointNe(p_2, cl_2); if\ i = j = 1)]$

2.1.3.6.2 Meet. A connected line *cl*$_1$ meets connected line *cl*$_2$ when they share one of the endpoints (see Figure 2-6(s)).

$- Meet(cl_1, cl_2) \equiv (\exists p)[Endpoint(p, cl_1) \wedge Endpoint(p, cl_2)]$

2.1.3.6.3 Cross. A connected line *cl*$_1$ and connected line *cl*$_2$ are crossed if and only if there is a single line *sl*$_i$ of connected line from both connected lines crossing each other (see Figure 2-6(u)).

$- Cross(cl_1, cl_2) \equiv \left(\exists sl_1 \in cl_1.sl_i, sl_2 \in cl_2.sl_j\right)[Cross(sl_1, sl_2)]$

A special case of cross is when one of the endpoints of one line lies on another line but not at the endpoint (see Figure 2-6(v)).

$- TCross(cl_1, cl_2) \equiv$

$Cross(cl_1, cl_2) \land (\exists p)[(Endpoint(p, cl_1) \land (\exists sl_2 \in cl_2.sl_i)[Ontheline(p, sl_2)] \land$

$\neg Endpoint(p, cl_2)) \lor (Endpoint(p, cl_2) \land (\exists sl_1 \in cl_1.sl_i)[Ontheline(p, sl_1)] \land$

$\neg Endpoint(p, cl_1))]$

2.1.3.6.4 Overlap. A connected line *cl₁* and connected line *cl₂* are overlapped when their single lines are overlapped or equal (see Figure 2-6(t)).

$- Overlap(cl_1, cl_2) \equiv (\exists sl_1 \in cl_1.sl_i, sl_2 \in cl_2.sl_j)[Overlap(sl_1, sl_2) \lor Equal(sl_1, sl_2)]$

2.1.3.6.5 Within. A connected line cl_1 is completely within a connected line cl_2 if and only if each single line of connected line cl_1 is shared-endpoint within one of the single lines of connected line cl_2 but two connected lines do not share any endpoint (see Figure 2-6(w)).

$- CompleteWithin(cl_1, cl_2) \equiv$

$(\forall sl_1 \in cl_1.sl_i, sl_2 \in cl_2.sl_j)[SharedEndpointWithin(sl_1, sl_2)] \land$

$(\forall p)[\neg(Endpoint(p, cl_1) \land Endpoint(p, cl_2))]$

A connected line cl_1 is considered as shared-endpoint within connected line cl_2 when each single line of connected line cl_1 is shared-endpoint within one of the single lines of connected line cl_2 and both connected lines share at least one of the endpoints (see Figure 2-6(x)).

$- SharedEndpointWithin(cl_1, cl_2) \equiv$

$(\forall sl_1 \in cl_1.sl_i, sl_2 \in cl_2.sl_j)[SharedEndpointWithin(sl_1, sl_2)] \land$

$(\exists p)[Endpoint(p, cl_1) \land Endpoint(p, cl_2)]$

2.1.3.6.6 Contain. Similar to the Contain operation between single lines, Contain operation between connected lines can be defined as a reverse of Within operation as follows.

$- CompleteContain(cl_1, cl_2) \equiv CompleteWithin(cl_2, cl_1)$

$- SharedEndpointContain(cl_1, cl_2) \equiv SharedEndpointWithin(cl_2, cl_1)$

2.1.3.6.7 Disjoint. Two connected lines are disjoint when each single line of one connected line is disjoint from all single lines of another connected line.

$- Disjoint(cl_1, cl_2) \equiv (\forall sl_1 \in cl_1.sl_i, sl_2 \in cl_2.sl_j)[Disjoint(sl_1, sl_2)]$

2.2 OWL/Protégé Realization

After formalizing the spatial ontology in the previous section, we now discuss a method to add spatial dimension to existing ontologies. Particularly, we use a technique similar to the "light-weight" temporal ontology introduced in [22] so that we can incorporate a spatial layer into existing ontologies without requiring significant changes to the original ontology. We also implement spatial built-ins in Protégé based on the object and operation definitions in the spatial ontology formalization to do spatial reasoning, inference and querying on ontology.

2.2.1 Protégé Background and Preparation

Protégé [30] is a well-known and widely-used open-source platform for ontology management including creation, visualization, and manipulation of ontology [15][84]. Moreover, Protégé is also user-friendly and domain-customizable because it allows user-defined or imported plug-ins. Our work uses Protégé-OWL 3.4.4 which comes with SWRL Tab [16] supporting reasoning and query. We also use Java JDK version 1.5.0.11 and Jess® rule engine [27] to do the inferences. In Protégé-OWL 3.4.4, there are three types of properties: object property, datatype property, and annotation property. An object property is used to define a relationship between individuals. A datatype property is used to define relationship between individuals and data literals such as integer, string, etc. An annotation property is used to attach metadata to classes, individuals or properties [83].

Protégé also allows users to customize a property as functional, non-functional, symmetric, or transitive. The following examples are from [85]. If a property is functional, the cardinality between subject and object is N:1. For example, *hasBirthday* property is functional because one person can have only one birthday whereas the same birthday can belong to many people. In contrast, if a property is non-functional, the cardinality between subject and object is 1:N. For example, *hasISBN* property is non-functional because, a book can have multiple ISBNs whereas an ISBN can belong to only one book. If a property is symmetric, both following statements are true: subject-property-object and object-property-subject; for example, *isFriendOf* property is symmetric because if John *is a friend of* David, then David *is also a friend of* John. If a property is transitive, the following statement: subject-property-object2 is true if subject-property-object1 and object1-property-object2 are true; for example, *subClassOf* property is transitive because if class A is a subclass of class B and class B is a subclass of class C, then class A is also a subclass of class C.

2.2.2 Adding Spatial Dimension to Ontology

We use our formal specification of spatial ontology and adopt one of the approaches proposed in [22] for adding temporal dimension to existing ontologies. Figure 2-7 gives an overview of our approach.

In Figure 2-7, SProposition2D class has *hasGeoShape* relationship with SGeometry class, which contains geometry types: point, single line, non-ring connected line, and ring connected line. Point consists of two numbers: longitude and latitude. Single line consists of exactly two points. Connected line consists of three points or more. There are two types of connected lines: non-ring and ring. *hasDistance* is a relationship between SGeometry class and DistanceUnit class. DistanceUnit contains the units of distance measurements such as km., mile, yard, etc.

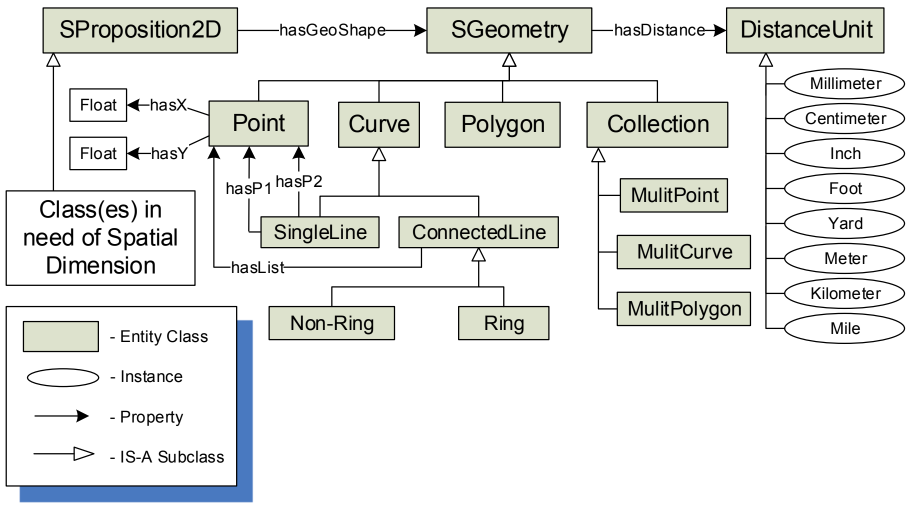


Figure 2-7 Method to Add Spatial Dimension to Existing Ontology

The principle is that any class in need of spatial dimension will be added as a subclass of SProposition2D class. Consequently, the class will automatically have *hasGeoShape* property that enables a class entity to be modeled as one of the spatial data types.

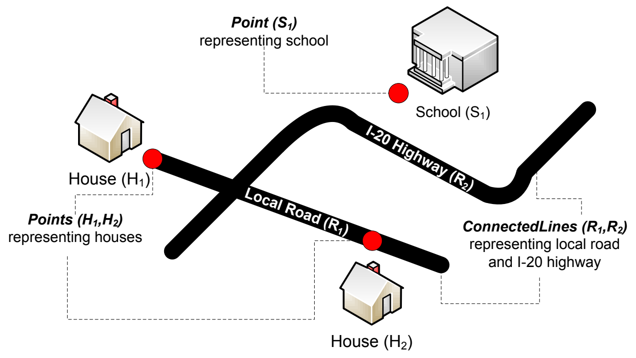

Point (S1) representing school
School (S1)
House (H1)
I-20 Highway (R2)
Local Road (R1)
Points (H1,H2) representing houses
ConnectedLines (R1,R2) representing local road and I-20 highway
House (H2)

Figure 2-8 Example of How Entities are Modeled in Spatial Dimension

An example is shown in Figure 2-8 Suppose we have a yellow page ontology which contains contact information of individuals, businesses, parks, trains rails, roads and highways, etc. We would like to add a spatial dimension to it. As a result, we will add house, park, train rail, road and highway classes to be subclasses of SPropostion2D class as shown in Figure 2-7 so that those classes will have spatial features of one of the spatial data types. In this example, houses and schools are modeled as points, roads and highways are modeled as connected lines, and parks are modeled as polygons. The choice of geometry type depends on the application scenarios.

In Protégé, a point is defined by two functional datatype properties, *hasX* and *hasY* of type float. A point is used to model an entity which does not have area. A single line is defined by exactly two distinct points. *hasP1* and *hasP2* are functional object properties of a single line. A connected line is defined as a list of three or more ordered points. *hasList* is a functional datatype property of a connected line, of type string, which has a following string pattern:

$$\{n,\ p_1,\ p_2, \ldots,\ p_n\}$$

where *n* is number of points, $n \geq 3$ and p_i is point *i*. This string pattern will be eventually parsed by Protégé built-ins into number of point instances.

2.2.3 Developing Spatial Built-ins in Protégé

According to Figure 2-1, for spatio-temporal reasoning, inference, and querying in Protégé, we develop the spatial operations as Protégé built-ins based on our formalization of spatial operations. We implemented 33 major spatial operations along with additional 6 utility built-ins in Protégé as shown in Table 2-1 and 2-2. Spatial built-ins can be divided into six categories depending on the combinations of geometry data types: point versus point, point versus single line, point versus connected line, single line versus single line, single line versus connected line, and connected line versus connected line.

Table 2-1 33 Major Spatial Built-Ins

<u>**Spatial Built-ins**</u>	**Point (1)**	**Single Line (2)**	**Connected Line (3) [Non-Ring & Ring]**
Point (1)	Equal	Endpoint EndpointNe EndpointSw Ontheline	Endpoint* EndpointNe* EndpointSw* Ontheline InteriorPoint
Single Line (2)		Equal Meet Cross Tcross Overlap Within: - Complete - SharedEndpoint Intersect	Meet* Cross Tcross Overlap Within: - Complete - SharedEndpoint* Intersect
Connected Line (3) [Non-Ring & Ring]			Equal Meet* Cross Tcross Overlap Within: - Complete - SharedEndpoint* Intersect

*Operations apply only on non-ring connected lines.

Table 2-2 6 Additional Spatial Utility Built-ins

<u>**Additional Built-ins**</u>	Slope, IntersectPoint, Between, ProperBetween, Distance, CreateSingleLine

In our notation, 1 represents point, 2 represents single line, and 3 represents connected line (non-ring and ring), which are appended to the operation (built-in) name. When defining a spatial rule in Protégé, the built-ins have to be specified as the following pattern:

```
spatial:<BuiltinName>XY(<GeoTypeX>,<GeoTypeY>)
```

where X and Y are ordered number corresponding to geometry data types. For example, to define an *EndPoint* built-in of point and single line, we specify:

```
spatial:endpoint12(<point>,<single line>)
```

This is because different combinations of geometry data types have different ways of implementing the same operation. As a result, we need to indicate the exact operation we are using.

2.2.4 Spatial Reasoning, Inference and Querying

Once we have a spatial dimension added to an ontology and have spatial built-ins ready in Protégé, we can do spatial reasoning, inference and querying.

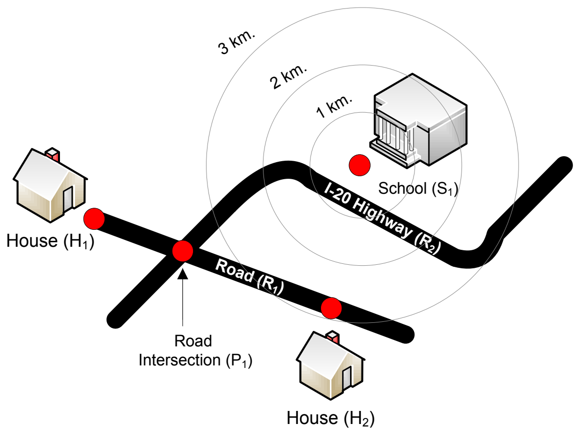


Figure 2-9 Example of How Spatial Operations can be used in Reasoning

We enable spatial reasoning by defining rules in SWRL Tab [16][30] and we use the Jess® [27] rule engine to perform inference on those rules. Spatial built-ins that we developed can be combined with other built-in libraries including SWRLB [16], TEMPORAL [22], SQWRL [21], etc. to create complicated rules. The following example shows how spatial built-ins can be used in reasoning and inference. Suppose we have 2 houses, 1 school and 2 roads as illustrated in Figure 2-8. We would like to define a rule regarding a school zone. Suppose, we define a location in a school zone as a point within 3 kilometers radius centered at a school as shown in Figure 2-9. This definition can be implemented in Protégé by the following rule:

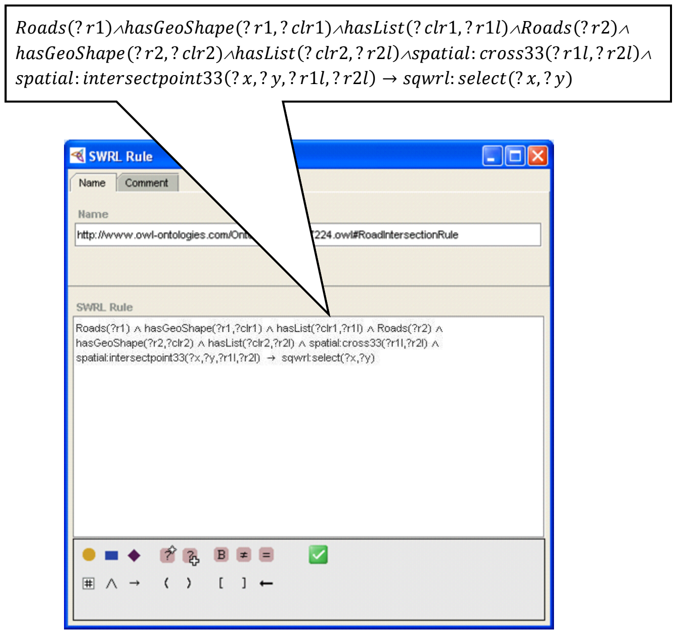
$$Point(?p) \wedge hasX(?p,?px) \wedge hasY(?p,?py) \wedge School(?s) \wedge hasGeoShape(?s,?ps) \wedge hasX(?ps,?sx) \wedge hasY(?ps,?sy) \wedge spatial{:}distance(?d,?px,?py,?sx,?sy) \wedge swrlb{:}lessthan(?d,3) \rightarrow SchoolZonePoints(?p)$$

For the next example, we would like to define highway-connected local roads as local roads that intersect with any highway at some point. The corresponding rule is:

$$Roads(?r) \wedge hasGeoShape(?r,?clr) \wedge hasList(?clr,?rl) \wedge Highways(?h) \wedge hasGeoShape(?h,?clh) \wedge hasList(?clh,?hl) \wedge spatial{:}Intersect33(?rl,?hl) \rightarrow HighwayConnectedLocalRoads(?r)$$

When we use Jess®, the above rules will classify points *?p* and roads *?r* as SchoolZonePoints and HighwayConnectedLocalRoads respectively if they satisfy the rules above.

To query, we use the *select* built-in from the SQWRL library [21]. The following example (see Figure 2-10) shows how spatial operations can be used in querying functionality. Suppose we have 3 roads in our ontology defined as *r1={3,1,1,1,3,9,6}*, *r2={4,2,5,8,2,6,7,9,5}* and *r3={3,1,4,1,6,3,9}* and we would like to list all road intersections *(x,y)*. The query can be defined as:



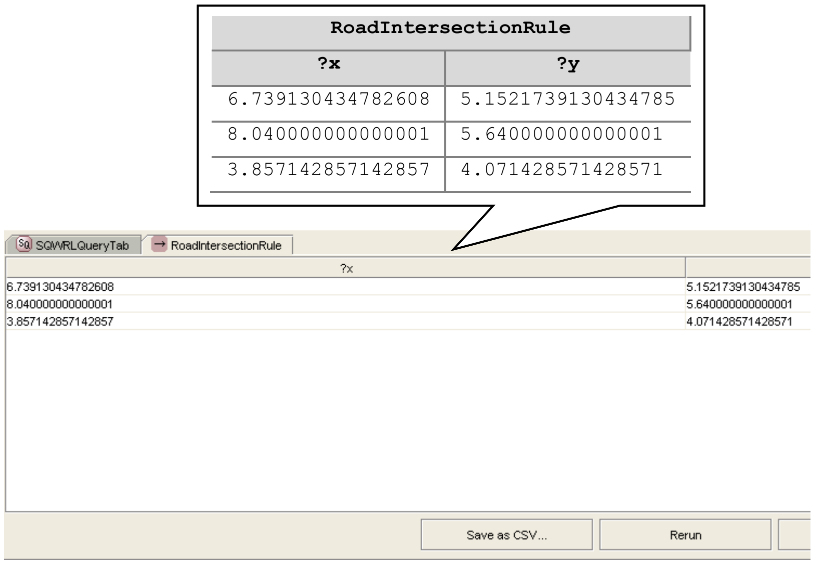
$Roads(?r1) \wedge hasGeoShape(?r1, ?clr1) \wedge hasList(?clr1, ?r1l) \wedge Roads(?r2) \wedge hasGeoShape(?r2, ?clr2) \wedge hasList(?clr2, ?r2l) \wedge spatial{:}cross33(?r1l, ?r2l) \wedge spatial{:}intersectpoint33(?x, ?y, ?r1l, ?r2l) \rightarrow sqwrl{:}select(?x, ?y)$

SWRL Rule
Name
Comment
Name
http://www.owl-ontologies.com/Ont 224.owl#RoadIntersectionRule
SWRL Rule

Roads(?r1) ∧ hasGeoShape(?r1,?clr1) ∧ hasList(?clr1,?r1l) ∧ Roads(?r2) ∧
hasGeoShape(?r2,?clr2) ∧ hasList(?clr2,?r2l) ∧ spatial:cross33(?r1l,?r2l) ∧
spatial:intersectpoint33(?x,?y,?r1l,?r2l) → sqwrl:select(?x,?y)

Figure 2-10 How to Define Rule in Protégé

When we processed the above query, the following table resulted from Protégé's SQWRL Query Tab (Figure 2-11).



RoadIntersectionRule	
?x	?y
6.739130434782608	5.1521739130434785
8.040000000000001	5.640000000000001
3.857142857142857	4.071428571428571

?x	
6.739130434782608	5.1521739130434785
8.040000000000001	5.640000000000001
3.857142857142857	4.071428571428571

Figure 2-11 Query Result Table in Protégé

2.3 OpenGIS® SQL Completeness

Our spatial ontology formalization is based on OpenGIS® [23]. So, it is important to prove that OpenGIS can actually capture the complete set of operations that cover spatial concepts. OpenGIS SQL is a standard for incorporating GIS and spatial concepts into the SQL standard database language. The objective of this section is to prove the completeness of OpenGIS SQL spatial relationships and operations. There are three types of 2-D spatial objects: Point (P), LineString (or Curve) (L), and Polygon (or Region) (A). Egenhofer shows that the spatial relationships between points, lines and polygons described in [32][33][34] are complete; that is, all possible 2-D spatial relationships are given in that paper. [33] first considers all cases based on the nine-intersection matrix (9IM) model and then eliminates cases that cannot occur. The number of all possible relationships between points, lines, and polygons identified in [33] is shown in Table 2-3. Thus, the complete set of 2-D spatial relationships is comprised of 68 relationships.

We show that OpenGIS SQL spatial relationships and operations are also complete by proving that all possible spatial 2-D relationships between points, lines, and polygon described in [33] can be expressed by OpenGIS SQL relationships and operations. Although OpenGIS SQL does provide a *Relate* operation, which is used to test exhaustively for intersections between the interior, boundary, and exterior of the two spatial 2-D objects given the dimensionally extended nine-intersection matrix (DE-9IM) [23], and hence can easily be used for the completeness proof, we found that the Relate operation is not needed since all the 68 possible spatial relationships between points, lines, and polygon can be described by using other OpenGIS SQL relationships and operations as you will see in our proof. In addition, the Relate operation may be considered as inefficient as it tests exhaustively for the intersections between the interior, boundary, and exterior of the given two spatial objects and also allows users to specify the spatial relationships that are not possible in real life. Also, as the OpenGIS SQL document states [23, Page 2-15], relate "has the disadvantage that it is a lower level building block and does not have a corresponding natural language equivalent", so it will be difficult for user to utilize the relate operation. Therefore, in our proof, we use a limited number of OpenGIS SQL spatial relationships and operations (excluding Relate operation) to describe all possible cases of the complete set of 68 spatial relationships identified in [33].

Table 2-3 Number of Possible Cases between 2-D Spatial Objects

Pairs	P/P	P/L	P/A	L/L	L/A	A/A
# cases	2	3	3	33	19	8

2.3.1 Summary of OpenGIS SQL Spatial Relationships and Operations

There are four main groups of methods in OpenGIS SQL:

- Basic methods on geometric objects
- Methods for testing spatial relationships between geometric objects
- Methods for supporting spatial analysis
- Miscellaneous methods for each type of geometric objects

We briefly discuss each of these groups next. For complete details about these methods, please refer to [23].

2.3.1.1 Basic methods for geometric objects

These operations are used to determine the properties and representation method for the 2-D objects. Table 2-4 is the summary of these methods along with the compatible geometric objects: P, L, and A.

Table 2-4 Summary of Basic Methods of OpenGIS SQL

Operations	P	L	A
Dimension( )	✓	✓	✓
GeometryType( )	✓	✓	✓
SRID( )	✓	✓	✓
Envelope( )	X	✓	✓
AsText( )	✓	✓	✓
AsBinary( )	✓	✓	✓
IsEmpty( )	✓	✓	✓
IsSimple( )	X	✓	✓
Is3D( )	✓	✓	✓
IsMeasured( )	✓	✓	✓
Boundary( )	X	✓	✓

2.3.1.2 Methods for testing spatial relationships between geometric objects

These are eight main binary boolean relationships that return TRUE if the particular topological relationship exists between the two given spatial objects. Table 2-5 shows a summary of the methods along with the compatible pairs of geometric objects: P/P, L/L, A/A, P/L, P/A, and L/A.

Table 2-5 Summary of 8 Main Spatial Relationships of OpenGIS SQL

Relationships	P/P	L/L	A/A	P/L	P/A	L/A
Equals( )	✓	✓	✓	X	X	X
Disjoint( )	✓	✓	✓	✓	✓	✓
Touches( )	X	✓	✓	✓	✓	✓
Crosses( )	X	✓	X	✓	✓	✓
Within( )	✓	✓	✓	✓	✓	✓
Contains( )	✓	✓	✓	✓	✓	✓
Overlaps( )	✓	✓	✓	X	X	X
Intersects( )	✓	✓	✓	✓	✓	✓

Our proof uses these operations in Table 2-5, which cover all binary spatial relationships in OpenGIS SQL. The first 7 operations above (Equals – Overlaps) can be summarized in the following decision tree (Figure 2-12), which is useful for understanding the correct definition for each relationship [23]. In Figure 2-12, the root of the decision tree is the leftmost node. Interior nodes describe spatial conditions, and leaf nodes represent OpenGIS SQL operations, where all the condition from root to leaf must be satisfied for the operation.

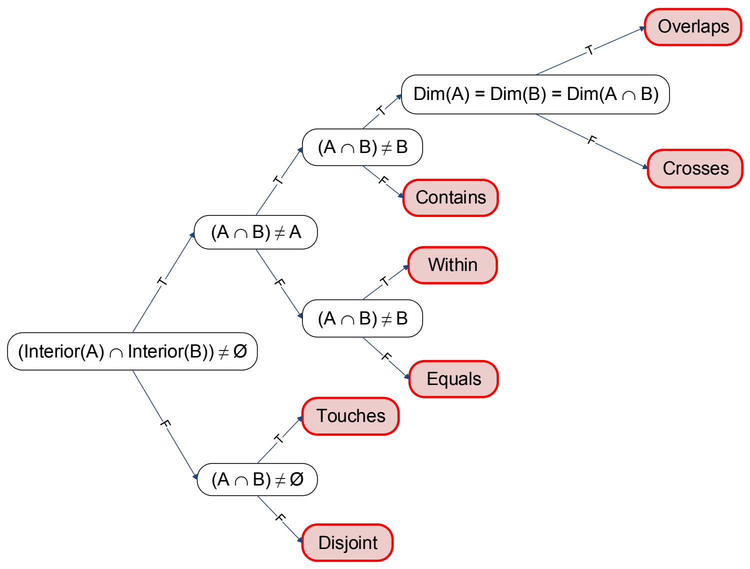


Figure 2-12 Summary Decision Tree of First 7 OpenGIS SQL Spatial Relationships (Equals - Overlaps) [23][81]

2.3.1.3 Methods that support spatial analysis

These operations either return metric values, or create new objects out of existing ones. Table 2-6 shows a summary of the methods along with the compatible geometric objects, and their pairs: P, L, A, P/P, L/L, A/A, P/L, P/A, and L/A.

Table 2-6 Summary of Methods that Support Spatial Analysis of OpenGIS SQL

Operations	P	L	A	P/P	L/L	A/A	P/L	P/A	L/A
Distance( )	X	X	X	✓	✓	✓	✓	✓	✓
Intersection( )	X	X	X	✓	✓	✓	✓	✓	✓
Union( )	X	X	X	✓	✓	✓	✓	✓	✓
Difference( )	X	X	X	✓	✓	✓	✓	✓	✓
SymDifference( )	X	X	X	✓	✓	✓	✓	✓	✓
Buffer( )	✓	✓	✓	X	X	X	X	X	X
ConvexHull( )	✓	✓	✓	X	X	X	X	X	X

2.3.1.4 Miscellaneous methods for each type of geometric objects

OpenGIS SQL also provides miscellaneous operations for each type of geometric object as summarized below.

1) Operations for Point (P)
- X(*P*)
- Y(*P*)
- Z(*P*) (for 3-D – we do not use Z)
- M(*P*) (allows application to associate any measure when needed for its environment with the point values)

2) Operations for Curve (or LineString: L)
- Length(*L*)
- StartPoint(*L*)
- EndPoint(*L*)
- IsClosed(*L*)
- IsRing(*L*)
- NumPoints(*L*)
- PointN(*L,i*) (return i^{th} point)

3) Operations for Polygon (A)
- Area(*A*)
- Centroid(*A*)
- PointOnSurface(*A*)
- ExteriorRing(*A*)
- NumInteriorRing(*A)*
- InteriorRingN(*A,i*) (return i^{th} interior ring)

In our proof, we will show how combinations of these OpenGIS SQL spatial relationships and operations can be used to describe all relationships in [33][32][34].

2.3.2 Proof of Completeness

We prove the completeness of OpenGIS SQL spatial relationships and operations by using proof by cases [102][35]. To prove by cases, we list all possibilities of the statement to be proved into cases [102]. We then show that in all cases, the statement is true [102]. Our proof first displays the complete set of 68 relationships [33] (each of which is numbered) in a graphical form as shown in Figure A-1. We then show how each relationship can be specified in OpenGIS SQL. Since the proof takes substantial amount of space, we described the proof in the Appendix A.

2.4 Related Work

Related work can be divided into two subsections: temporal ontology formalization and temporal information representation and querying in OWL [22][87].

2.4.1 Temporal Ontology Formalization

Hobbs [13] presented temporal ontology formalization which can be embedded to OWL [87] to capture temporal reasoning on the semantic web. There are four main concepts that are discussed on the paper: temporal relation reasoning, measuring duration, clock/calendar, and describing time and duration.

Temporal relation concept defines the reasoning of time instant and time interval, where time instant stores a point in time and time interval stores starting and ending times. In addition, the temporal operation reasoning is also discussed, such as *before*, *inside*, *timebetween*, *properbetween*, *equal*, *meet*, *overlap*, etc. Measuring duration concept defines a model to measure the unit of time and converts one unit of time to another. Clock and calendar concepts are used to appropriately reason about time zone and handle months with different days such as 28 days in the month of February.

Describing time and duration concepts are used to capture time and duration with different kinds of formats such as timestamp.

2.4.2 Temporal Information Representation and Querying in OWL

O'Connor and Das [22] proposed a technique to represent time dimension in OWL and demonstrated how temporal reasoning and querying over those ontologies on Protégé can be done. They presented two approaches in modeling temporal dimension: via user-defined property and via super class relationship. With regard to the first approach, they directly specify spatial features of entity through user-defined property. In the second approach, to represent time in OWL on Protégé, they first create a Proposition class which has a relationship *hasValidTime* to ValidTime class. The ValidTime class contains two types of time concepts: time instant and time interval. When we want to add temporal features to any class, the class will be added as a subclass of the Proposition class and it will inherit the time characteristics. They also mentioned that the second approach is more powerful for two main reasons. The first reason is the ease in distinguishing the class with and without temporal features just by looking at which classes are subclasses of Proposition class. The second reason is the eligibility to have additional co-existing spatial representation in the future through multiple inheritance, by having more than one parent classes, without causing any effects and deleting the prior ones. They used the second approach for adding temporal components into an existing ontology. The technique they used is known as "light-weight". That is, time dimension could be easily added to existing ontologies with minimal changes. The built-ins temporal operations were implemented based on [2][3].

Chapter 3

Identifying Meaningful Concepts from Raw Data

Starting from this chapter, we now turn our focus to a specific type of spatio-temporal data in a particular application domain. Identifying meaningful concepts for a particular application domain is a very important step because each spatio-temporal type of data has its own characteristics, which need to be specifically designed according to their domain concepts. In our work, the spatio-temporal data used is raw rainfall data in the hydrology domain. Our goal for this chapter and the next is to summarize the raw rainfall data into a model that facilitates storm analysis and mining. We identify meaningful storm concepts from the raw rainfall data in this chapter and discuss two different approaches to convert raw rainfall data into the meaningful storm concepts in the next chapter. The preliminary approach for storm analysis and mining is discussed in Chapter 5. This chapter is organized as follows. Section 3.1 discusses background and motivation of identifying meaningful storm concepts from raw rainfall data. The description of raw rainfall data is discussed in Section 3.2. Finally, storm formalization is defined in Section 3.3. (Related work of this chapter is discussed in Chapter 4, Section 4.5.)

3.1 Background and Motivation

In hydrology, most storm analysis has mainly focused on location-specific analysis (either site-specific or region-specific) [36][37][38][39], meaning that each location is considered independently when analyzing a storm. An example would be determining how many storms occurred at site location 376501 in the year 2011. But in reality, a storm covers many locations over a period of time, so location-specific analysis is insufficient. Our goal is to analyze rainfall data in a storm-specific way by considering all the locations over time for each storm, so we can determine storm-specific characteristics such as how big the storm is, how many sites are covered, the storm track, and what is its overall depth and duration. Analyzing the whole storm can give more insight and information since it reflects how a storm actually behaves in nature. In particular, a storm can start at one location and end at another, and the storm typically covers multiple locations at each time point. However, it is very difficult to analyze storms directly from the raw data for several reasons. First, the quantity of data is very large that it qualifies as big data [54][55]. Second, the data is stored in a manner that makes it difficult to identify the storms. The data has been gathered as frequently as every five minutes and covers a huge area of observation fields. Traditionally, the data is recorded and stored in either printed or file/folder format. As a result, attempting to do storm analysis with such a large amount of data and the traditional way of storing the data will require manually combining all data across an enormous number of folders and processing them together. This makes it nearly impossible to do storm analysis [40]. As a result, to enable storm-centric analysis, we first identify meaningful storm concepts by incorporating hydrology concepts (as we will discuss in Section 3.3). We then develop algorithms to identify the different types of storms as described in the formalization. Finally, we store the identified storms in a custom designed database schema. The big picture of our methodology is shown in Figure 3-1.

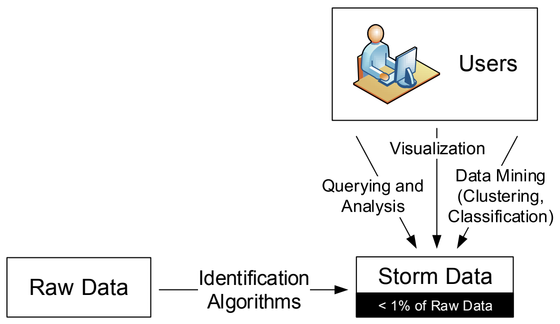


Figure 3-1 Overview of Our Methodology

This will allow domain experts to do more robust and flexible analysis and help them better understand their own data and potentially can lead to new discoveries.

3.2 Description of Raw Rainfall Data

The raw rainfall data that we used comes from National Weather Service (NWS) – West Gulf River Forecast Center (WGRFC) [42] and is called *Multi-sensor Precipitation Estimates* or *MPE* [42][45][77][79] reflecting how it is calculated, which is estimated by using a combination of radars and physical rain gauges (multi-sensors). The raw data is provided as hourly text files containing four attributes: observation time, row number, site id, and precipitation value, covering three main states (Texas, New Mexico, and Oklahoma) and some surrounding areas including parts of Arizona, Utah, Colorado, Kansas, Missouri, Arkansas, Louisiana, and Mexico [42]. (Figure 3-2 shows the coverage of our MPE observations.) The observation time is indicated in the file name (e.g., 2011041323_2011041400), whereas the remaining attributes are stored in the file. Each row of the file content consists of row number, site id, and precipitation value (inches), and reports one observation per one site. The total number of covered observed site locations are 165,750. Site points are four kilometers apart to the north, south, east, and west. The raw rainfall data is supplied as text files. Each hourly text file contains precipitation data for that particular hour for all sites. This means that the number of records inserted per hour, day, month, and year is 165,750, 3,978,000, 119,340,000, and 1,432,080,000, respectively.

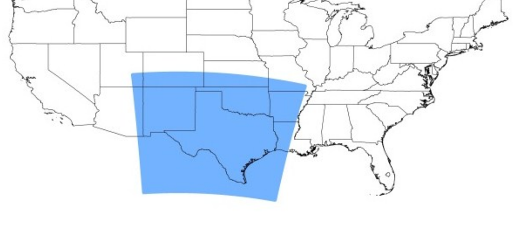


Figure 3-2 Coverage of WGRFC Observations [42]

The raw rainfall data text file is based on the *HRAP (Hydrologic Rainfall Analysis Project)* standard grid coordinate system [49][58] and is ordered by site id in a row major order from west to east and south to north. Each row has 425 sites and each column has 390 sites. Because of the systematic grid structure, given any site, we can determine the neighboring sites by using the relationships described in Figure 3-3. Moreover, given any site id, we can determine its HRAP local X and Y coordinates, using equations (1) and (2); where *a* is the minimum HRAP *x* coordinate and *b* is the minimum HRAP *y* coordinate, *c* is the first site id, and *d* is a value difference between site ids in adjacent rows in the same column. Additionally, given any (*x*, *y*) coordinate, we can calculate the corresponding site id using equation (3). Equation (3) is derived from (1) and (2), by considering $(x-a)$ and $(y-b)$ as outputs of the operations (*mod* and *div*) of the same operands ($(SiteID-c)$ and *d*). In our case, *a*, *b*, *c*, and *d* are 290, 10, 15599, and 1701, respectively.

$$x = a + \big((SiteID - c)\ mod\ d\big) \tag{1}$$

$$y = b + \big((SiteID - c)\ div\ d\big) \tag{2}$$

$$SiteID = d(y-b) + (x-a) + c \tag{3}$$

Our raw data contains 16 years of historical MPE data from 1997 to 2012. The size of the raw data in textual format is approximately 480 GB, and when loaded to the CUAHSI ODM relational database [45][47][58][44], it is about 5.4 TB. Every year an additional 30 GB/348 GB is added to textual/CUAHSI data.

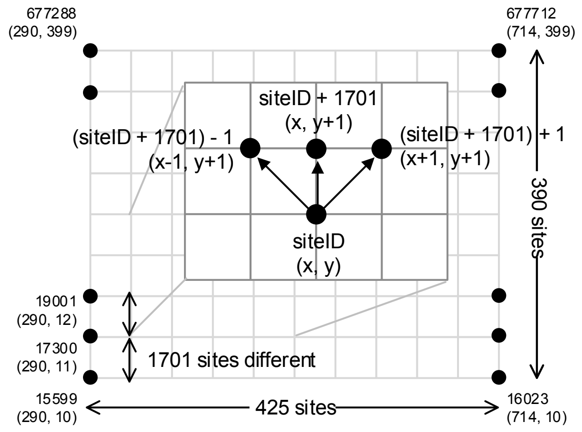


Figure 3-3 Relationships among Neighboring Sites

3.3 Rainstorm Formalization

The goal of our storm formalization is to analyze storms as a whole and at the same time, still allow traditional location-specific storm analysis. Since a storm can start at one place and stop at another, we slice the whole storm into several pieces by hour. We then assemble each slice back together into the original overall storm. Each slice of storm is, in fact, an hourly storm. We formalize storms into three different categories (local storms, hourly storms, and overall storms).

Before defining our storm-related concepts, we specify some predicates (relationships) and terminology that will be used in the definitions.

- *neighbor*(s_a, s_b, *d*): means that sites s_a and s_b are adjacent. Referring to Figure 4-4, if s_a is the central site, s_b can be any of the other sites. *d* is the direction from s_a to s_b, and is one of (N, S, E, W, NE, NW, SE, and SW).
- *area*(*s*): the area of site *s.*
- *storm area*: the total areas of a storm.

Next, we define the concepts of *local storm*, *hourly storm*, and *overall storm*.

3.3.1 Local Storm

Generally speaking, *local storm* is a site-specific storm, which considers each site location independently when analyzing a storm, e.g., determining how many storms occurred at site location 355879 in 2011. An example of local storms is the sequence of storms that occurred at site location 355879 last month. Two distinct local storms are separated by at least *h* consecutive time points with zero precipitation, where *h* is called the *inter-event time* [36][37][52]. In our case, inter-event time (*h*) is set to 6 hours as suggested in [36][37][103]. There may be some consecutive time points with zero precipitation within a local storm, as long as it is less than *h* time points. For any local storm, there will not be a subsequence of *h* or more consecutive zeroes in the series. Local storm is one type of storm, which was researched by most hydrologists [36][37][38][39]. Figure 3-4 shows some examples of local storms at site id, 355879. The formal specification of local storm is described in Definition 1.

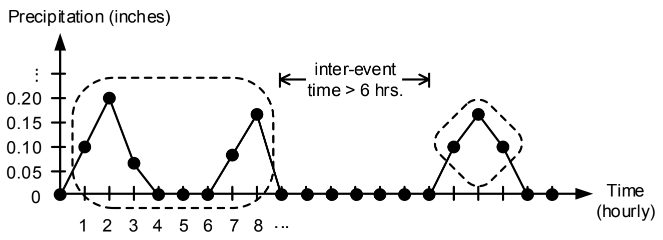


Figure 3-4 Examples of Local Storms at Site ID, 355879

Definition1. If L is a *local storm*, then:

- $s(L)$ = site location id for local storm L
- $T(L) = (t_1, t_2, \ldots, t_n)$ is the sequence of consecutive time points for local storm L at site $s(L)$, such that:
  - t_1 = start time *and* t_n = end time
  - $p(t_i)$ is the precipitation measured at site $s(L)$ for time t_{i-1} to t_i (inches)

- $p(t_1) > 0$ and $p(t_n) > 0$
- $t_i - t_{i-1} = 1, i \neq 1$ *(all time points in T(L) are consecutive.)*
- $p(t_k) = p(t_{k+1}) = \ldots = p(t_{k+(h-1)}) = 0$ is false, where $k = 1, 2, \ldots, (n-h)+1$ and h = inter-event time (maximum number of $h-1$ consecutive zero precipitation time points in a local storm)

Let LL be a list of all local storms at a particular site (i.e., $LL = (L_1, L_2, \ldots, L_m)$). LL must satisfy the following properties:

- L_i.*start_time* - L_{i-1}.*end_time* $> h, i \neq 1$
- L_{i+1}.*start_time* - L_i.*end_time* $> h, i \neq m$

(These formalize the concept that two different local storms at site *s* must be apart by at least inter-event time.)

Storm characteristics for this storm type include:

- *duration*: the duration of a local storm (*n hours)* [51]
- *total rainfall*: the amount of precipitation occurring throughout the storm duration at a particular site ($\sum_{i=1}^{n} p(t_i)\ inches$) [51].
- *intensity*: the total rainfall divided by duration ($\frac{total\ rainfall}{duration}\ inches/hr$) [51].

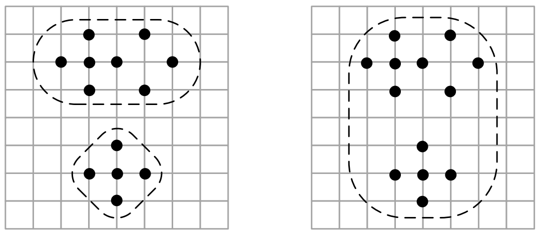
3.3.2 Hourly Storm

Informally, *hourly storm* is a time-specific storm, which is an orthogonal concept to local storm. It considers each hour independently when analyzing a storm. An example of hourly storms is a set of storms that occurred between 9:00 am and 10:00 am today across various site locations. Hourly storm considers a specific time point (an hour) instead of considering a particular site location. In other words, local storm fixes one site and covers its data over many time points, whereas hourly storm fixes a time point and covers its data over many adjacent sites. Two different hourly storms are separated by *space-tolerance n* [62]. Space tolerance specifies a maximum number of sites (*n*) with zero precipitation allowed in between any two non-zero precipitation sites to be considered as part of the same hourly storm. The space tolerance concept allows non-zero precipitation sites to still be categorized as part of an hourly storm even if they are not in adjacent neighboring sites but are *indirect neighboring* sites within a certain number of intermediate sites (see Definition 2).

Definition 2. We say that site *b* is an *i-indirect neighbor* of site *a* if:

$$neighbor(a, x_1), neighbor(x_1, x_2), \dots , neighbor(x_i, b)$$

That is, when space-tolerance is set to *n*, the neighbors of site *a* will include direct neighbors, as well as all *i-indirect neighbors* of *a* for *i* = 1, 2, … , *n*. Figure 3-5 compares space-tolerance of 0 and 1. With the same set of non-zero precipitation values, represented by dots at a particular hour, with space-tolerance *n=0*, 2 hourly storms are identified whereas space-tolerance *n=1*, only 1 hourly storm is identified.



a) Space-tolerance, *n=0* b) Space-tolerance, *n*=1

Figure 3-5 Comparison between Different Space-Tolerance Values

In our work, space-tolerance is set to zero. That is, all non-zero precipitation sites of an hourly storm must be adjacent. With space-tolerance set to zero, an hourly storm can be formally defined by a set of adjacent sites with non-zero precipitation values at a particular hour as described in Definition 3.

Definition 3. If *H* is an *hourly storm*, then:

- *t(H)* = time point (a particular hour)
- *S(H)* = $\{s_1, s_2, \ldots, s_n\}$ is a set of site location ids s_i such that:
    - given any s_i in the set, $p(s_i) > 0$, where *p(s)* is precipitation measured at time *t(H)* for site *s.*
    - if $|S(H)| = 1$, then it contains a single site with no neighbors.
    - otherwise, *S(H)* satisfies these properties:
        - *Connectivity*: $\forall s_a \in S(H) \exists s_b \in S(H): neighbor(s_a, s_b)$ (every site in the set must have at least one neighbor that is also in the set)
        - *Maximality* [98]:

          $\forall s_a, s_b \in all\_sites: (s_a \in S(H) \wedge neighbor(s_a, s_b)) \rightarrow s_b \in S(H)$

          (for any two neighboring sites s_a, s_b with non-zero rainfall in the same hour, they belong to the same hourly storm)

The last property can also be stated as follows. Let *HH* be a set of all hourly storms at a particular hour (i.e., *HH* = $\{H_1, H_2, \ldots, H_m\}$). *HH* must satisfy the following property:

- $\forall H_i, H_j \in HH: (s_a \in H_i \wedge s_b \in H_j) \rightarrow \neg neighbor(s_a, s_b)$ (two different hourly storms at the same hour do not have overlapped site(s))

The characteristics of hourly storm are listed below:

- *total rainfall*: the total amount of precipitation occurring at a particular hour for the sites of an hourly storm ($\sum_{i=1}^{n} p(s_i)\ inches$).
- *coverage*: number of sites covered by an hourly storm ($|S(H)|$ sites).
- *average*: the average precipitation for an hourly storm ($\frac{total\ rainfall}{coverage}\ inches/site$).
- *center*: a storm center based on hydrology concept. It is defined as the site coordinate(x_{center}, y_{center}) with the highest precipitation at a particular hour. If

there are more than one site with highest precipitation, the x_{center} and y_{center} will be averaged based on those sites coordinates.

- *centroid*: a storm center based on geometry concept. It does not take into account the precipitation values when calculating the storm center. The center of the storm will be calculated solely based on the average of *x* and *y* coordinates of all the sites of an hourly storm.
- *boundary*: a minimum bounding rectangle (MBR) covered by an hourly storm, defined by two HRAP sites coordinates: one with the lowest *x* and *y* (x_{min}, y_{min}) and another one with the highest *x* and *y* (x_{max}, y_{max}).

3.3.3 Overall Storm

Unlike local storm and hourly storm that consider either a site location or time (an hour) independently, *overall storm* considers both location and time together when analyzing a storm. So, the result is the capture of storm as a whole, called *overall storm*, which can capture storm movement and other "overall" storm characteristics that could not be found in most hydrology papers [36][37][38][39][53]. Overall storm is a combination of hourly storms that satisfy two requirements: *grouping-window g* and *spatial-window s* [62]. Grouping-window is the time interval within which storms will be considered to be part of the same storm. Spatial-window is the number of common site(s) shared between any two successive hourly storms. This definition of overall storm ensures that hourly storms that move to the same path will be considered as the same overall storm. According to hydrology concepts, it is very unlikely that two different paths of hourly storms with different origins and/or destinations could end up being part of the same overall storm. However, if that is the case, the final path of the overall storm will be averaged out based on those two paths. In this work, grouping-window and spatial-window are set to 1 hour and 1 site, respectively. An example of overall storm and its corresponding hourly storms is shown in Figure 3-6. The formal specification of overall storm is described in Definition 4.

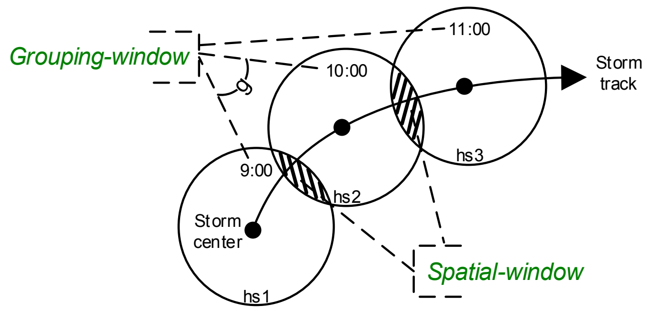


Figure 3-6 Example of Overall Storm and its Corresponding Hourly Storms

Definition 4. If *O* is an *overall storm*, then:

- *O* = (V_1, V_2, … , V_n) such that:
    - V_i is an hourly storm set at hour *i* where *i* = 1, 2, …, *n*, V_1 is a set of hourly storms in the first hour (i.e., {$H_{1,1}$, $H_{1,2}$,…}), and V_n is a set of hourly storms in the last hour (i.e., {$H_{n,1}$, $H_{n,2}$,…})
    - if |*O*| = *1,* then the overall storm is equivalent to an hourly storm
    - otherwise*, O* satisfies these properties:
        - $\forall a \in V_k \exists b \in V_{k+1}: t(b) - t(a) \leq g,$ where $k \neq n$ and *g* is *grouping-window*
        - $\forall a \in V_k \exists b \in V_{k+1}: |S(b) \cap S(a)| \geq p,$ where $k \neq n$ and *p* is *spatial-window*
        - *Maximality* [98]: $\forall H_a, H_b \in all\_hourlystorms: (H_a \in O.V_i \wedge |t(H_b) - t(H_a)| \leq g \wedge |S(H_b) \cap S(H_a)| \geq p) \rightarrow H_b \in O.V_j$

The maximality property of overall storms can also be stated as follows. Let *OO* be a set of all overall storms (i.e., *OO* = {O_1, O_2,…, O_m}). *OO* must satisfy the following property:

- $\forall V \in OO.O_i, W \in OO.O_j: (H_a \in V \wedge H_b \in W) \rightarrow$

$$\neg (|t(H_b) - t(H_a)| \leq g \wedge |S(H_b) \cap S(H_a)| \geq p)$$

The following are overall storm characteristics:

- *total rainfall*: the total amount of precipitation occurring throughout the storm duration across the hourly storms
- *duration*: the duration of an overall storm (*n hours)*
- *intensity*: the total rainfall divided by the duration ($\frac{total\ rainfall}{duration}$ inches per hour).
- *coverage*: the number of “distinct” sites covered an overall storm
- *num sites*: the summation of each hourly storm coverage of an overall storm (Note that a site in num sites can be repeated whereas one in coverage cannot.)
- *average*: the average precipitation (per site) for an overall storm ($\frac{total\ rainfall}{num\ sites}\ inches\ per\ site$).
- *intensity per site (or average per hour):* the intensity divided by num sites (or the average divided by duration). This characteristic gives an idea of how intense a rainstorm is for any given site location covered by an overall storm.
- *total average:* the summation of storm average for each hour of an overall storm
- *total average per hour:* the total average divided by the storm duration
- *track:* a sequence of storm centers or storm centroids of its hourly storms. (The storm track is important when doing trajectory-related analysis of the storm.)
- *speed:* storm speed is defined as an average distance (km) that the storm moves per hour (kmph). It can be calculated by dividing the summation of distances between each pair of consecutive storm centers (or centroids) by storm duration.
- *boundary:* an MBR covering the entire overall storm.

Chapter 4

Converting Raw Data to Conceptual Data

In the previous chapter, we identified and formally defined meaningful rainstorm concepts. These were developed in consultation with hydrology experts from the Civil Engineering Department. In this chapter, we present a *storm identification system* to extract the three rainstorm information, as well as their characteristics and properties, from the raw data. Two approaches were developed: CUAHSI-based [62] and MapReduce-based [68][76]. Both share the same goal, which is to convert the raw data into meaningful storm conceptual data. The first approach focuses on converting raw rainfall data that follows the CUAHSI standard, which is based on the standard database schema called *CUAHSI ODM* [45][47][58] to store hydrological data in a relational database. Both input and output are processed and stored in a relational database. Although there are benefits from this approach, including easy integration with CUAHSI APIs [44] and user-friendly analysis/mining through SQL [56][57], the main disadvantage we found is the slow performance, because of the overhead associated with using a relational database for the input raw data. In addition, the algorithms developed were recursive depth-first search, which also contributed to the poor performance. As a result, the second approach was proposed with the focus on drastically improved performance for the data conversion algorithms by utilizing parallel processing on a cluster of nodes using the map-reduce framework. The CUAHSI-based approach is described in Section 4.1 whereas the MapReduce-based approach is described in Section 4.2. We proposed a custom database schema for storing the conceptual storm output data in Section 4.3. The resulting storm data was verified in Section 4.4. Finally, related work is discussed in Section 4.5.

4.1 CUAHSI-based Approach

The goals of this approach are to: 1) convert raw rainfall data into meaningful storm concepts (which can capture storm-centric characteristics) and 2) to follow the CUAHSI standard, which we will discuss in the next subsection.

4.1.1 CUAHSI-related Background

CUAHSI (Consortium of Universities for the Advancement of Hydrologic Science, Inc.) [47] is a well-known research organization conducting research in the water science-related area since 2001, supported by National Science Foundation (NSF). Hydrologic observation data is gathered from various organizations and kept in various formats. To eliminate the ambiguities in sharing and interpreting hydrological information, *CUAHSI ODM* [45] was proposed in 2008 [58]. *CUAHSI ODM* provides a standard database schema to store hydrological data in a relational database. There are a total of 29 tables in the standard. Only five main tables will be briefly discussed (Sources, Sites, Methods, Variables and DataValues tables) as they were used in our analysis. Figure 4-1 shows the star-schema [41, Chapter 28] of the five main tables. For more details, please refer to [45][58].

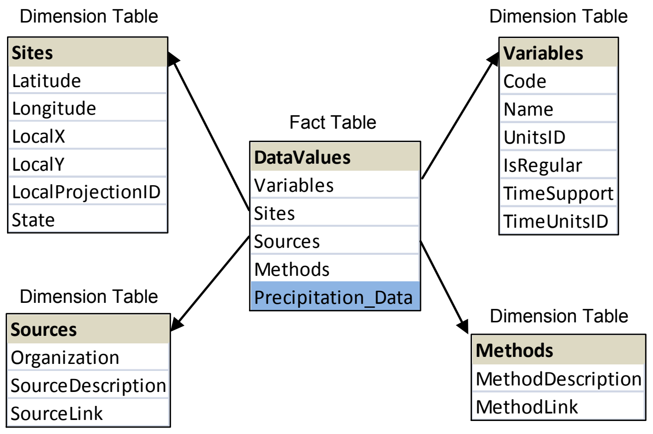


Figure 4-1 Star-Schema of 5 Main Tables of CUAHSI ODM

The Sources table stores information about where the observation data comes from. Table 4-1 shows our Sources table in the database.

Table 4-1 Sources Table in CUAHSI ODM Database

ID	Organization	SourceDescription	SourceLink
1	NOAA's National Weather Service West Gulf River Forecast Center	Files containing MPE data from NWS-WGRFC	http://www.srh.noaa.gov/wgrfc/

The Methods table (see Table 4-2) describes how the observation is collected. A brief explanation of the method along with its external link is also provided in this table.

Table 4-2 Methods Table in CUAHSI ODM Database

ID	MethodDescription	MethodLink
1	The precipitation data are multi-sensor (radar, satellite, and rain gauge).	http://www.srh.noaa.gov/rfcshare/precip_about_hourly.php

The Sites table stores site information. The site information includes SiteID, Longitude, Latitude, LocalX, LocalY, LocalProjectionID (spatial reference system for LocalX and LocalY, such as HRAP [49], which we discussed in Chapter 3, Section 3.2), State, etc. Table 4-3 shows selected columns of our Sites table in the database.

Table 4-3 Selected Columns of Sites Table in CUAHSI ODM Database

ID	Latitude	Longitude	LocalX	LocalY	LocalProjectionID	State
339072	31.0444	-97.9782	573	200	227	Texas
339073	31.0402	-97.9379	574	200	227	Texas
339074	31.0359	-97.8976	575	200	227	Texas

The next table is the Variables table. The information about observations is stored in this table. Each variable description is stored in one row, and represents different observation types and properties. The property information includes how frequent the observation is recorded (instantaneous or consistent) and what unit is used for the observation values. That is, for example, hourly precipitation observation and 15-minute interval precipitation observation are considered different variables due to their properties even though they both are the same precipitation observation types. We have one variable (one row) as demonstrated in Table 4-4 which describes hourly precipitation data.

Table 4-4 Selected Columns of Variables Table in CUAHSI ODM Database

ID	Code	Name	UnitsID	IsRegular	TimeSupport	TimeUnitsID
1	MPE	Precipitation	49	1	1	103

The last table is the DataValues table, which stores the actual rainfall data. This table stores numerical observation values for each site and variable as well as the method used and the source where they are from. Table 4-5 shows some samples of what DataValues table entries look like. The first row of the table states that we have no rain (precipitation value = 0) at site location 88814 from 12pm to 1 pm on October 1, 2011. As we can see that regardless of whether or not we have rain, the precipitation value is inserted into the table. As a result, the database grows rapidly and is sparse.

Table 4-5 Examples of DataValues Table Entries with Selected Columns

ID	DataValue	DateTimeUTC	SiteID	VariableID	MethodID	SourceID
1	0	2011-10-01 13:00	88814	1	1	1
2	0	2011-10-01 13:00	88815	1	1	1
3	0	2011-10-01 13:00	88816	1	1	1

4.1.2 Storm Identification Algorithms

In this approach, the storm identification system is divided into four main components: 1) local storm identification, 2) location proximity creator, 3) hourly storm identification, and 4) overall storm identification. Figure 4-2 shows the architecture of the CUAHSI-based storm identification system.

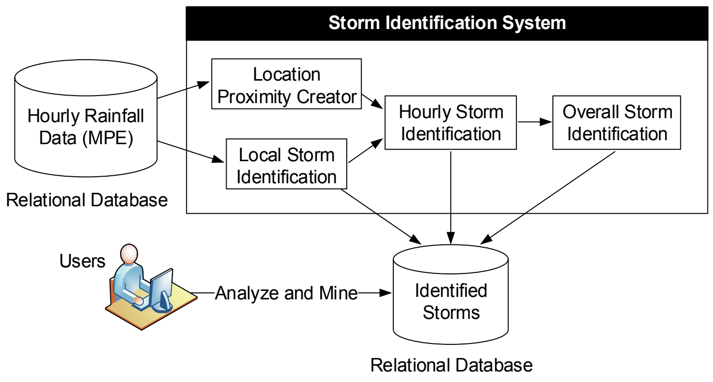

Storm Identification System
Hourly Rainfall Data (MPE)
Relational Database
Location Proximity Creator
Local Storm Identification
Hourly Storm Identification
Overall Storm Identification
Users
Analyze and Mine
Identified Storms
Relational Database

Figure 4-2 Data Flow Diagram of Storm Identification Modules

4.1.2.1 Local Storm Identification

This module separates rainfall events at any given site location (local storms) using *h*-hour inter-event time as storm separators. The input for this module is the relational rainfall data from the CUAHSI ODM DataValues table as well as inter-event time (*h* hours). In our experiment, only sites in Texas are considered. Since the area of Texas is very large, there is a significant climatic difference in its various regions. As a result, USGS (U.S. Geological Survey) divides Texas into 10 different regions based on their climatic and geographic characteristics and proposed a map, called *Texas Climatic Regions* [39], as shown in Figure 4-3. To be consistent with USGS, we analyzed each region separately. We also used threads [61] to improve algorithm performance. For each region, sites are equally partitioned into *t* different disjoint subsets. Each subset is then assigned to one thread. The threads run concurrently and then the results are merged to form LocalStormHours table. In our case, *t* is set to 4, assuming that each thread occupies each of 4 cores of our computer configuration.

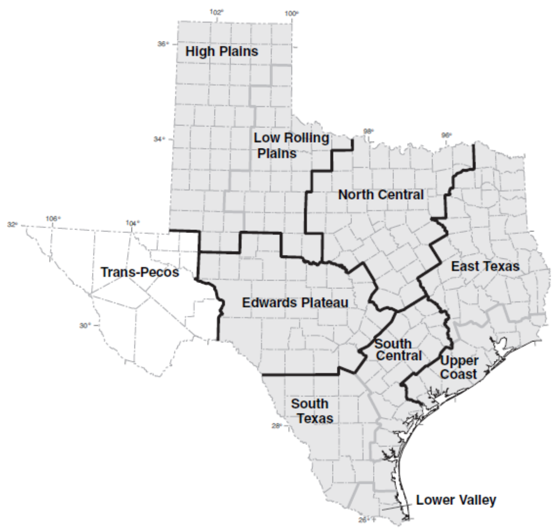


Figure 4-3 Texas Climatic Regions [39]

To separate rainfall events for a particular site, a parameter *h*, called *inter-event-count*, is maintained to keep track of the number of consecutive zero precipitation (ordered by date and time). We use *h*=6 hours inter-event time, as suggested by Huff [36][37]. In some situations or applications, a different inter-event time is needed and this can be achieved by changing the parameter in our algorithm to other values. The identified local storms are then stored in the LocalStormHours table, which we will discuss in Section 4.3. Algorithm 1 highlights how the local storm identification works. The time complexity for the algorithm is $O(\frac{s\times(n+n\log n)}{c})$, where *s* is the number of sites in a subset, *c* is the number of cores, and *n* is the number of records per site, assuming $n\log n$ is the cost for sorting by using ORDER BY in database query.

4.1.2.2 Location Proximity Creator

This module creates the LocationProximity table containing neighboring sites information for each site. The neighboring sites information is required for hourly storm identification but does not exist in any table of the original ODM standard.

Algorithm1. CUAHSI-based Local Storm Identification

```
    Input:
        - Rainfall data of a region ($D$)
        - Inter-event time ($h$)
        - Number of threads ($t=4$)
    Output:
        - Local storms stored in LocalStormHours table
1:  partition sites of region ($D$) into $t$ subsets ($S$)
2:  assign each subset to a thread
3:  concurrently,
4:      threads process their own subsets of sites $S_i$, $i = 1, 2, \dots, t$
5:      for each site $x$ in $S_i$ do
6:          $r \leftarrow$ extract records of site $x$ and order by time
7:          for each record $r_j$ in $r$ do
8:              if inter-event-count < $h$ then
9:                  include $r_j$.precipitation as part of local storm $k$
10:             else
11:                 start new local storm $k$++
12:             end if
13:         end for
14:     end for
15: merge results from each thread into LocalStormHours table
```

The input of this module is site information from ODM Sites table [43][45][58]. The output is stored in the LocationProximity table. Figure 4-4 shows neighboring sites of site 355879. The neighboring information was calculated for each site using the HRAP coordinate information [49] labeled as LocalX and LocalY in the ODM Sites table. A site $s(x,y)$ will have eight neighboring sites: $s_N(x,y+1)$, $s_S(x,y-1)$, $s_E(x+1,y)$, $s_W(x-1,y)$, $s_{NE}(x+1,y+1)$, $s_{NW}(x-1,y+1)$, $s_{SE}(x+1,y-1)$, and $s_{SW}(x-1,y-1)$ where (x,y) is a HRAP coordinate of site s. The algorithm is relatively simple and so omitted here. The time complexity is $O(n)$ where n is the number of sites. This table, however, is not needed in the map-reduce algorithm, because the neighboring sites information can actually be derived from site id presented in the raw data text files, as we describe in equations (1), (2), and (3) in Chapter 3, Section 3.2.

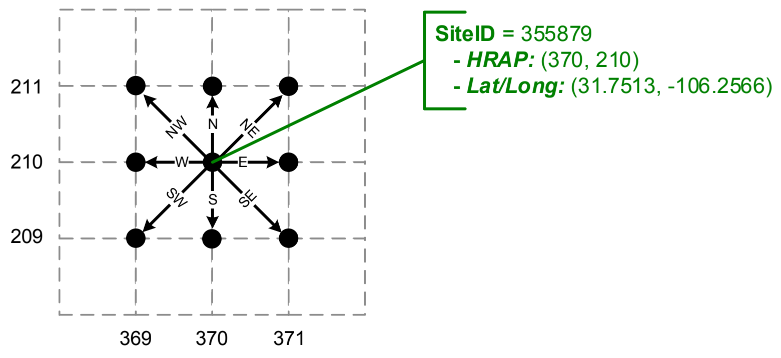


Figure 4-4 Neighboring Sites of Site Location 355879

4.1.2.3 Hourly Storm Identification

This module identifies hourly storms by finding neighboring sites that have precipitation during the same hour. Since all non-zero precipitation was already extracted and stored in LocalStormHours table, we then directly use LocalStormHours table instead of ODM DataValues table as an input to increase the performance of the module. Another input is LocationProximity table. The output is stored in the HourlyStormSites table. The algorithm (see Algorithm 2) is based on recursion and depth-first search. It checks for each hour to identify how many hourly storms there are, and the sites they cover. Similar to the local storm identification, each region is executed separately. However, we did not apply threading to this module since the amount of data to be processed is substantially less compared to the one of local storm identification module, because most of the zero-precipitation data has been removed. The time complexity for the worst case scenario is $O(h \times s^2 \times 8^s)$, where h is the number of hours in local storm data and s is the maximum number of non-zero precipitation sites in an hour.

Algorithm2. CUAHSI-based Hourly Storm Identification

```
    Input:
        - Local storm data ($L$)
        - Location proximity data ($P$)
    Output:
        - Hourly storms stored in HourlyStormSites table
1:  for each hour $h$ in $L$ do
2:      $b \leftarrow$ extract all records of hour $h$
3:      for each site $s$ in $b$ do
4:          if $s$.precipitation $\neq 0$ then
5:              identified as hourly storm $i$
6:              depthFirstSearch($s, i, b$)
7:              start new hourly storm $i$++
8:          end if
9:      end for
10: end for
11: depthFirstSearch($s, i, b$)
12:     candidates set $c \leftarrow$ expandNode($s, b, P$)
13:     for each candidate $c_j$ in $c$ do
14:         if $c_j$.precipitation $\neq 0$ then
15:             identified as part of hourly storm $i$
16:             depthFirstSearch($c_j, i, b$)
17:         end if
18:     end for
```

4.1.2.4 Overall Storm Identification

This module identifies all overall storms (which consist of hourly storms that are sharing some common site(s) (*spatial-window s*) within the specified *grouping-window g* hour(s)). The module takes hourly storm data from HourlyStormSites table as an input. An output is stored in the OverallStormHourlyStorms table, which indicates overall storms and their corresponding hourly storms. The algorithm (see Algorithm 3) uses recursion and depth-first search similar to the hourly storm identification algorithm. Instead of checking neighboring sites, it checks if successive hourly storms are sharing some common site(s) (spatial-window *s*) within the grouping-window *g* (in hours). In our analysis, grouping-window is 1 hour and spatial-window is 1 site. That is, if consecutive hourly storms are within 1 hour difference and sharing at least 1 common site, they will be considered as part of the same overall storm. Note that we did not apply threading to the algorithm as the process needs to be done sequentially and identify overall storms one at a time. The time complexity for the worst case scenario is $O(n^2 \times m^h)$, where n is the number of hourly storms, m is the maximum number of qualified (both grouping- and spatial- windows) hourly storms for an hourly storm, and h is the maximum number of hours for an overall storm.

Algorithm3. CUAHSI-based Overall Storm Identification

```
    Input:
        - Hourly storm data ($R$)
        - Grouping-window ($g=1$)
        - Spatial-window ($s=1$)
    Output:
        - Overall storms stored in OverallStormHourlyStorms table
1:  $a$ ← extract all hourly storm ids from $R$
2:  for each hourly storm id $d$ in $a$ do
3:      identified as overall storm $i$
4:      depthFirstSearch($d$, $i$, $a$)
5:      start new overall storm $i$++
6:  end for
7:  depthFirstSearch($d$, $i$, $a$)
8:      candidate set $c$ ← findCandidates($d, g, s$)
9:      for each candidate $c_j$ in $c$ do
10:         identified as part of overall storm $i$
11:         depthFirstSearch($c_j$, $i$, $a$)
12:     end for
```

4.1.3 Experimental Results

The data used for our preliminary experiment in this approach is 1.25 years (October, 2010 – December, 2011) of relational data from ODM DataValues table and covers only Texas. The data contains 394,505,690 records of historical hourly precipitation data covering 37,413 sites. The experiment was performed on a single server. The server runs on Microsoft® Windows Server® 2008 Enterprise operating system with 2.83 GHz Intel® Xeon® quad-core processors, 20 GB of RAM, 500 GB of local disk, and 10 TB of external disk. The experimental results for each region are listed in Table 4-6.

The experimental result (Table 4-6) shows the significant reduction in size of the storm data compared to the size of the raw data. The number of storm records is less than 1% of the number of raw data records. The experimental result also indicates that East Texas has the most storm data whereas Trans-Pecos has the least storm data even though Trans-Pecos has more raw data compared to East Texas. This is consistent with the fact that Trans-Pecos is the driest region and East Texas is one of the wettest regions in Texas [59][60].

Table 4-6 Experimental Results of CUAHSI-Based Approach

Regions	Number of Raw Data	Number of Identified Storms			Number of Storm Data	Reduction in Raw Data Size
		Local Storms	Hourly Storms	Overall Storms		
1. East Texas	48,953,130	325,504	21,983	4,632	352,119	0.72%
2. Edwards Plateau	73,415,532	257,859	20,136	4,191	282,186	0.38%
3. High Plains	31,711,927	97,327	8,334	2,165	107,826	0.34%
4. Low Rolling Plains	24,965,521	89,814	6,199	1,487	97,500	0.39%
5. North Central	59,082,957	299,082	17,303	3,463	319,848	0.54%
6. South Central	31,102,334	120,083	11,654	3,224	134,961	0.43%
7. South Texas	26,091,999	97,580	10,067	2,867	110,514	0.42%
8. Lower Valley	11,182,285	41,820	4,314	1,228	47,362	0.42%
9. Trans-Pecos	65,136,216	151,453	11,843	3,155	166,451	0.26%
10. Upper Coast	22,863,789	137,843	14,043	3,255	155,141	0.68%
TOTAL	**394,505,690**	**1,618,365**	**125,876**	**29,667**	**1,773,908**	**0.45%**

In summary, in this approach, the raw rainfall data is first converted and inserted into CUAHSI ODM database. The programs were designed and developed mainly based on the fact that the raw data is stored in relational databases. The storm identification then processes the raw data from the relational database. The local storm identification uses selection and sorting features of SQL to complete the process. The hourly storm identification takes location proximity and local storms as inputs and identifies hourly storms at each hour. Finally, overall storm identification combines consecutive hourly storms that meet grouping- and spatial- windows requirements to create the overall storms and store them in a relational table. Both hourly and overall storm identification use the concept of graph search using depth-first search (DFS), to complete the process.

All the processing is done sequentially in a single machine. The advantages of this approach include: 1) it supports both location- and storm- specific analyses, 2) it is consistent with the CUAHSI ODM standard (additional relations storing the extracted output conceptual data can be viewed as an enhancement to the CUAHSI ODM), 3) the resulting storm data size is much smaller compared to the original size of the raw data, and 4) it fully supports SQL and other relational database functionalities so easy analysis and mining can be done.

However, the major disadvantage of this approach is the slow performance due to several reasons. First, we used big raw rainfall data that are stored in a relational database as an input, which requires a very long time to retrieve the data to process. Second, the system is implemented based on graph search using DFS and recursion, which involve multiple scans of the same data. Third, the system is run on a single server without applying any distributed computing technology. Finally, we did not fully utilize the structure of the raw rainfall data format in the original text files. In the next approach (MapReduce-based) [68][76], we aim to improve the system performance.

4.2 MapReduce-based Approach

In this approach, every component of the storm identification algorithms are re-designed and performance is significantly improved by utilizing the original structure and format of raw rainfall data and applying the efficient, well-known distributing computing technology (namely MapReduce [66] and Hadoop® [67]) to speed up the performance. In this approach, instead of using data in the relational database as an input, we use the original raw rainfall data text files. We then applied distributed computing technology, map-reduce, to every component of the storm identification process in order to maximize the performance by parallelism. All components are re-designed to best suit map-reduce characteristics as well as the structure of the input data. MapReduce is a programming paradigm developed by Google in 2004 [66] and now is becoming a new standard for distributed computing.

As in the first approach, we load the identified storms (output data) into a relational database at the end so that analysis and mining can easily be done. The relational databases are used only for storing the final results of the identified storms; not during processing. The database schema is discussed in Section 4.3.

4.2.1 Storm Identification Algorithms

In this approach, the storm identification system consists of three components instead of four as we take advantage of the known grid structure for neighboring information. The overview of this approach is illustrated in Figure 4-5.

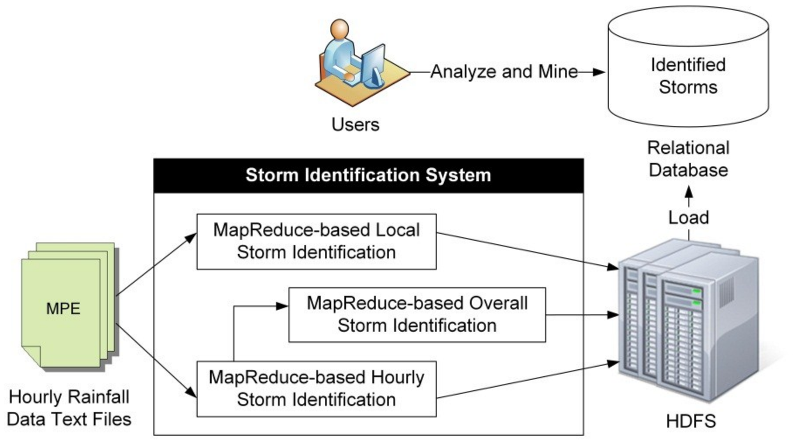

Users
Analyze and Mine
Identified Storms
Relational Database
Load
Storm Identification System
MapReduce-based Local Storm Identification
MapReduce-based Overall Storm Identification
MapReduce-based Hourly Storm Identification
MPE
Hourly Rainfall Data Text Files
HDFS

Figure 4-5 Overview of MapReduce-based Storm Identification System

4.2.1.1 MapReduce for Local Storm Identification (MR-LSI)

The previous implementation of local storm identification [62] required the selection of data from the relational database and then sorting them. The computation is done based on the selected sorted data and the result is inserted back to the database. The selection, sorting, and insertion required substantial execution time, making it impractical to analyze large datasets.

This new local storm identification algorithm [68] utilizes map-reduce, and uses the raw rainfall data text files as input. Each raw rainfall data text file contains the precipitation value of all the sites for a particular hour and hence, for the analysis of local storm, we need to group all the precipitation values by site and order them by time. Once all the values for a site are grouped together and ordered, then we can find all the local storms that occurred at that site. Thus, the local storm analysis contains two steps: 1) grouping precipitation values by site and ordering them by time and 2) finding the local storms and their characteristics for a site from the grouped values. In the map-reduce framework, there are three main phases: 1) map phase, 2) sorting and shuffling phase, and 3) reduce phase. The first two phases of map-reduce are used to perform the first step of our local storm identification and the reduce phase is used to find the local storms as well as their characteristics at the particular site.

The pseudo code for the implementation for local storm analysis in the map-reduce framework is shown in Algorithm 4. Each of the map tasks takes one raw rainfall file and processes it line by line emitting the site and time together as the key and time and precipitation value together as the value. We take advantage of the key-comparator class and grouping-comparator class of map-reduce to group the data on the basis of site id and then sort them by time. The reducer gets a site id as a key and list of precipitation values sorted by time. This list is processed sequentially to identify all the local storms and their characteristics at that particular site. We calculate the time complexity of the MR-LSI algorithm by determining the summation of maximum amounts of time spent by one map task, one sorting and shuffling task, and one reduce task. Thus, the time complexity is $O(n) + O(n \times m) + O(m) = O(n \times m)$, where n is the number of sites presented in the file and m is the number of hours (records) per site.

Algorithm4. MapReduce-based Local Storm Identification

```
    Input:
        - Text file-format rainfall data
        - Inter-event time ($h$=6 hours)
    Output:
        - Local storms data in text file format
1:  class MAPPER
2:  function MAP(key object, value line)
3:          key ← (line.siteId, line.time)
4:          value ← (line.precipValue, line.time)
5:          Emit(key, value)
6:  class REDUCER
7:  function REDUCE(key siteId, [val1, val2, …])
8:          timeList ← null
9:          //timeList.size = $h$ + 2; timeList[0:1] represents start and end time, timeList[2:7] is used to keep track $h$
10:         precipRec ← null
11:         interEventTime ← 0
12:         lsId ← 1
13:         timeList.Add(firstNonZeroPrecip.GetTime()) //for start time
14:         timeList.Add(firstNonZeroPrecip.GetTime()) //for end time
15:         precipRec.Add(firstNonZeroPrecip.GetPrecipValue())
16:         for all val ∈ values [val1, val2, …] do
17:                 precipRec.Add(val.GetPrecipValue())
18:                 if (val.GetPrecipValue() = 0) then
19:                         timeList.Add(val.GetTime())
20:                         interEventTime++
21:                 else
22:                         tempTime ← timeList[0] //keep original start time
23:                         Clear(timeList)
24:                         reset interEventTime
25:                         timeList.Add(tempTime)
26:                         timeList.Add(val.GetTime()) //get the new end time
27:                 end if
28:                 if interEventTime ≥ $h$ then
29:                         startTime ← timeList[0]
30:                         endTime ← timeList[1]
31:                         value.Set(startTime, endTime, precipRec, totalRainfall, duration, intensity)
32:                         Emit(siteId, lsId, value)
33:                         Clear(timeList)
34:                         Clear(precipRec)
35:                         lsId++
36:                 end if
37:         end for
```

In our experiment, we analyze the raw rainfall data one year at a time. Storms that start in one year and end in the next may exist. Therefore, we perform post-processing steps to combine these storms, and these local storms are assigned to the year where they started. These post-processing steps are done by using SQL. However, they can also be done in other methods such as external scripts. Algorithm5 highlights how the post-processing steps work.

Algorithm5. Local Storm Merging

```
    Input:
        - Relational tables: LocalStormHours and LocalStorms
        - Inter-event time: h (6 hours)
    Output:
        - Updates on tables: LocalStormHours and LocalStorms
1:  for each (current) year (except the last year: 2012) do
2:      for each local storm cls in current year that ends between 18:00 (6pm) and 23:00 (11pm) do
3:      //during the last 6 (h) hours of a year, the end time of cls is determined by the last non-zero precip. hour
4:          s ← cls.siteid
5:          e ← cls.endtime
6:          if there exists a local storm nls at site s, which starts by e+h in the next year then
7:              cls.Merge(nls) //updates LocalStormHours and LocalStorms tables accordingly
8:          end if
9:      end for
10: end for
```

4.2.1.2 MapReduce for Hourly Storm Identification (MR-HSI)

In the previous approach [62], we assume that any non-zero precipitation site can be part of the hourly storm, meaning it can start at one site and stop at a very farther site as long as there are some connections among them. As a result, we implemented DFS to keep track of every possible site and perform site node revisiting when needed. This, however, led to a high time complexity problem.

In the new approach [68], the program is designed specifically to take full advantage of the original raw rainfall data text file structure. Since the grid (HRAP) is known and we know exactly which site is a neighbor of which (as described in equations (1), (2), and (3) in Chapter 3), only those candidate neighboring sites need to be checked. Unlike the previous approach which uses DFS to keep track of nodes, we use linked lists and append them together as we scan when necessary. Moreover, since the data in each text file is stored in row major order, we scan each grid row once. An overview of the hourly storm identification process is shown in Figure 4-6.

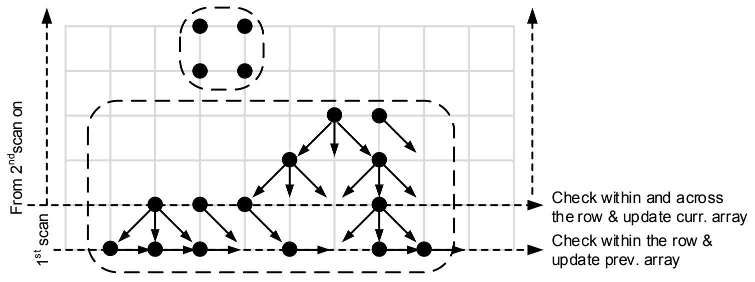

From 2nd scan on
1st scan
Check within and across the row & update curr. array
Check within the row & update prev. array

Figure 4-6 MapReduce-based Hourly Storm Identification

The program starts from the very bottom grid row to the top by calling map function for each line in the text file. It begins to identify hourly storms as soon as it reads in the data in order to minimize the number of checks. The data are then kept in two arrays called *previous* and *current arrays*, which are two-dimensional arrays and contain site ids and hourly storm ids. The current array always does the identification based on the previous array. There are two main parts of the program. The first part (line: 7-17) is executed only once for the very bottom row in a grid whereas another part (line: 18-27) is executed for the rest. The first part identifies hourly storms within the same row whereas the other part identifies hourly storms within and across the rows simultaneously. At the end of each row scan, the hourly storms so far are identified and are kept in an array of linked lists called *hourly storms list*, in which index of array indicates hourly storm id and linked list contains a set of adjacent non-zero precipitation sites of the hourly storm. When the last row is reached, the final hourly storms are produced and already kept in the hourly storms list.

Since the raw rainfall data files are independent from each other, and each file records hourly precipitation for an individual hour, map-reduce can easily be applied. Each hourly file is sent to a different mapper node for the identification of hourly storms.

Algorithm6. MapReduce-based Hourly Storm Identification

```
     Input:
        - Text file-format rainfall data
     Output:
        - Hourly storms data in text file format
1:  class MAPPER
2:  function SETUP()
3:      prev.InitializeArray(), curr.InitializeArray()
4:      hourlyStorms.InitializeArrayOfLinkedList()
5:      id ← 0
6:  function MAP(key object, value r)
7:      if r ∈ first bottom grid sites then
8:          if r.precip = 0 then
9:              prev[r.site].hsId ← 0 //no hourly storm
10:         else
11:             if r.site = first site or r.leftNeighborPrecip = 0 then
12:                 prev[r.site].hsId ← id++
13:                 temp ← CreateLinkedList(r.site)
14:                 hourlyStorms.AddLinkedList(temp)
15:             else
16:                 prev[r.site].hsId ← id
17:                 hourlyStorms.GetLinkList(id).Add(r.site)
18:     else if r ∈ next above grid sites then
19:         if r.precip ≠ 0 then
20:             if r.site = first site or r.leftNeighborPrecip = 0 then
21:                 CheckPrevious(r, id, prev, curr, 1)
22:             else
23:                 CheckPrevious(r, id, prev, curr, 0)
24:         else
25:             curr[r.site].hsId ← 0 //no hourly storm
26:     else
27:         prev ← curr
28: function CLOSE()
29:     Emit(hourlyStorms)
30: function CHECKPREVIOUS(r, id, prev, curr, flag)
31:     if flag = 1 then
32:         if hsIds of all 3 neighbors of r in prev. array = 0 then
33:             curr[r.site].hsId ← id++
34:             temp ← CreateLinkedList(r.site)
35:             hourlyStorms.AddLinkedList(temp)
36:         else
37:             minId ← MinHsId(r.all3Neighbors in prev. array)
38:             curr[r.site].hsId ← minId
39:             hourlyStorms.GetLinkedList(minId).Add(r.site)
40:             UpdateHsId(r.neighbors, minId)
41:             minId ← 0 //reset minId
42:     else
43:         curr[r.site].hsId ← id
44:         hourlyStorms.GetLinkedList(id).Add(r.site)
45:         if hsId of r's southeast neighbor in prev ≠ id then
46:             UpdateHsId(r.southeastNeighbor, id)
```

At the closing of mapper, all hourly storms identified within the hour will be written back to disk. Currently, no reducer is needed because there is no need to group the data or sort them in any order. The raw files, by themselves, are already grouped and sorted by site id in a row-major order as mentioned in Chapter 3, Section 3.2. The algorithm for hourly storm identification is shown in Algorithm 6. The algorithm has a linear time complexity ($O(n)$), where n is the number of lines (which also equals to the number of sites) in the text file.

4.2.1.3 MapReduce for Overall Storm Identification (MR-OSI)

In the previous approach [62], we use graph traversal (DFS) to identify overall storms. It uses backtracking according to the DFS concept. The two major disadvantages of this approach are that: 1) the next identification process could not start until the current identification is finished (sequential processing), and 2) the program interacts with large amounts of data residing in a relational database causing large overhead even when retrieving hourly storm data.

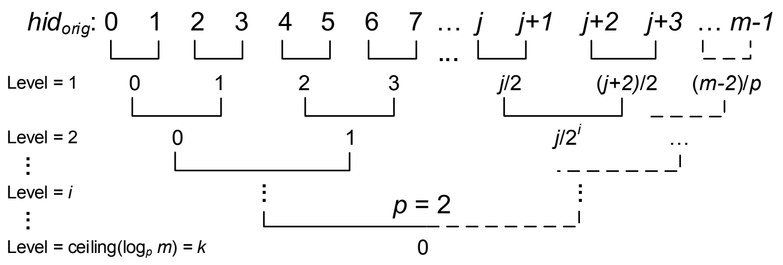
In the new approach [76], we no longer interact with the data stored in a relational database. Our input is from map-reduce-based hourly storm identification algorithm as hourly storm text files, which have the following format:

```
(hsid, time, <list of hourly storm characteristics>,
    <list of pairs (site : precipitation value)>)
```

The new algorithms are designed to work in parallel to utilize multiple machines by using map-reduce framework. Instead of the recursion used in DFS that identifies overall storms sequentially one by one, we develop an *iterative MapReduce* approach that divides the identification process into several iterations. In each iteration, partial overall storms are identified concurrently on multiple machines. The length of partial overall storm starts from 1 and keep increasing as the iteration level goes up. The flow of the algorithms could be thought as bottom up that builds overall storms from the smallest unit (1-hour duration) and expands until it is pruned (written to the output files) or ended at the last iteration. There are three phases in our new algorithm: mapper, partitioner, and reducer. All phases are executed iteratively. The number of iterations depends on the size of input file, which is the total number of hours (*m*) in our case.

In the mapper phase, each map task takes one hourly storm text file as an input, calculates *original hour id* (*hid*$_{orig}$), and emits it along with all of its hourly storms as key-value pairs. We process the data for one year at a time. The amount of data to process, however, can be changed to other sizes such as 2 or 5 years. The original hour id starts from 0, for the first hour of input files, until the last hour, which in our case, it is either 8759, for non-leap year, or 8783, for leap year. After the first iteration, the original hour ids are converted to *current hour ids* (*hid*$_{curr}$) directing hourly storms to the designated reducer. The key is hour id and the value is content of an hourly storm.

In the partitioner phase, each partition task ensures that a set of hour ids data go to the same reducer. The rules of hour ids grouping are that: 1) each group of hours must have *p* hour id members, where *p* is a range partition (*p* is set to 2 in our case), 2) hours in each group must be consecutive and the first group always starts from zero, and 3) hourly storms data with the same $\left\lfloor \frac{hid}{p} \right\rfloor$ will go to the same reducer. Figure 4-7 describes hour ids grouping conditions.



hid_{orig}: 0 1 2 3 4 5 6 7 … j j+1 j+2 j+3 … m-1
Level = 1
0 1 2 3 j/2 (j+2)/2 (m-2)/p
Level = 2
0 1 $j/2^i$ …
Level = i
p = 2
Level = ceiling($\log_p m$) = k
0

Figure 4-7 Hour IDs Grouping Conditions

In the last phase, each reduce task is the one who actually combines satisfying hourly storms from two hours into an overall storm. In each iteration, a set of hourly storms from two consecutive hours are grouped together and checked for overlapping in parallel. The results of each group (reducer) are “partial” overall storms (*ps*) for that particular iteration level. A partial overall storm can be considered as temporary overall storm, waiting to be finalized, for a sub period of the entire storm duration.

As you can see in Figure 4-7, the merging process is not finished within the first iteration. Moreover, the next iteration of each group will not start until its relevant previous iterations are finished. This makes sure that computation of different iteration levels will not happen. An output from the previous reducer is an input to another reducer in the next iteration. In our case, *p* = 2, *g* = 1, *s* = 1. When moving to the next iteration, the number of groups is decreased in half (factor of *p*). The results of the previous iteration will be checked for overlapping with the relevant consecutive group. The computation is relatively complex depending on the number of current candidate members from both groups. As a result, we implement the following three strategies: *global pruning*, *on-the-fly comparison*, and *local pruning*, based on the following lemmas.

Lemma 1. (Global pruning) Given *grouping-window* of *g* and *partition range* of *p*, the pruning can be applied starting at the end of *iteration level i*, where $i > \log_p 2g$, $i = 1, 2, \ldots, k$ and *k* is the last iteration, calculated by $\lceil \log_p number\ of\ hours\ (m) \rceil$.

The first prune step (global pruning) is based on Lemma 1. Figure 4-8 shows an example of how Lemma 1 can be applied. As you can see, at the end of iteration level *i*, where $i > \log_p 2g$, there will be partial overall storms starting as early as original hour id $(j \times p^i)$ and ending by original hour id $(j \times p^i) + p^i - 1$, where *j* is the *current hour id* and p^i is the number of participating hours. By partial overall storms, we mean intermediate overall storms that are still not finalized and need to check for overlapping with other corresponding partial overall storms unless: 1) reaching the last iteration *k* or 2) candidate set is empty, whichever comes first. Any partial overall storm that starts and ends within the range $[(j \times p^i) + g, (j \times p^i) + p^i - 1 - g]$ can be pruned and hence finalized as “final” overall storm since they will never get merged with any other partial overall storms. In our case, $g = 1$ and $p = 2$. Thus, the prune step can be applied at the end of iteration level 2 $(2 > \log_2 2(1))$. At the end of iteration level 2, at the current hour id 1, for example, any partial overall storm starting from original hour id 5 $(1(2^2) + 1)$ and ending by 6 $((1(2^2) + 2^2) - 1 - 1)$ will never get combined with the consecutive group of current hour id even though they may share *s* common sites in the same level because it exceeds grouping-window limit on both sides.

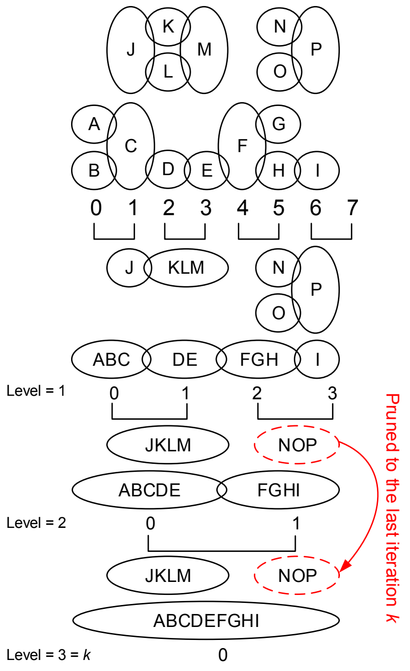


Figure 4-8 Global Pruning Concept

Lemma 2. (On-the-fly comparison) If we: 1) compare (check for overlapping) a partial overall storm ps_1 of one group hour id to every other group candidate (ps_0) one at a time in a given order, 2) replace the overlapped candidate ps_0 with the merged candidate ($ps_{01} = ps_0 \cup ps_1$), and 3) continue the checking process with ps_1, and repeat 2) whenever new overlapping is found, then every pair of candidates will require overlap checking at most once. The number of comparisons will be $O(n \times m)$, where n and m are the numbers of partial overall storms from the two groups.

Lemma 2 ensures the minimum number of comparisons (eliminating excessive/redundant comparisons) by: 1) fixing the order of candidates while checking, 2) utilizing the fact that a newly merged candidate will never overlap with previously-checked candidates otherwise they would have merged earlier (without considering overlapping within the same group because it is impossible), and 3) using merging to reduce the number of comparisons and at the same time to guarantee future overlapping discovery. To better understand this lemma, consider Figure 4-9. Suppose we have a set of partial overall storm candidates from two groups of hour ids as seen in Figure 4-9 a). As you can see, when you check B and Z the first time, they are not together. However, once B is checked with Y and C and combined, it is possible now that the new merged candidate (BYC) can overlap with Z. Therefore, the comparison process needs to start over from the beginning every time the new merged candidate is found, which is BYC in this example, to make sure all overall storms are correctly identified. This leads to a very high computation. The total number of comparisons is $\sum_{i=1}^{k} \binom{f(i)}{2}$, where *f(i)* is the number of candidates at iteration level *i* and $f(1) = n + m$, where n and m are the numbers of hourly storms of group *j* and *j+1*, respectively.

On the other hand, if we apply lemma 2 as seen in Figure 4-9 b), the re-checking will no longer be required since we will not need to start over the checking process every time a new candidate is added.

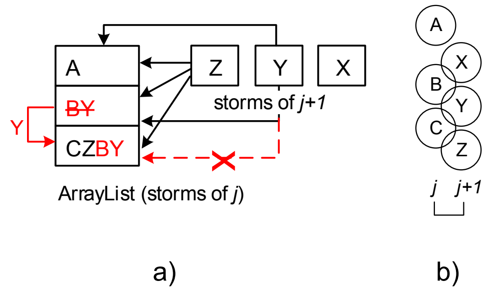


Figure 4-9 Example of How Lemma 2 can be Applied

The overlapping discovery can be guaranteed because we store one group of storms in an array list and another group of storms are being compared to it one at a time on-the-fly and merged, if necessary. So, the comparison process is thorough and complete. This on-the-fly comparison helps reduce the number of comparisons by identifying overall storm as it reads in the partial overall storms concurrently. For example, in Figure 4-9 b), once overlapping between Y and B is found, B is replaced by the new merged candidate BY and we do not need to re-check BY with previously checked candidates (A) again but rather continue the checking process using Y to the rest of the candidates in order. The order of storms in the array list and merging process we used will eventually make sure the checking is complete. That is, CZ, previously merged candidate, will later be checked and merged with BY to CZBY as the same overall storm.

Lemma 3. (Local pruning) Given a pair of partial overall storms, ps_0 and ps_1, of hour ids *j* and *j+1*, respectively, F_1 and L_1 are starting time and ending time of ps_0 whereas F_2 and L_2 are starting time and ending time of ps_1. Any pair of storms, with $F_2 - L_1 > g$, where *g* is a grouping-window, will never get merged and hence can be pruned to the next iteration.

Lemma 3 is based on the fact that any consecutive partial overall storms with grouping-window greater than *g* will never be combined and hence do not need to be checked. This lemma is similar to the first lemma except that the first lemma prunes partial overall storms to be finalized and hence no longer kept in the candidate set whereas this lemma prunes partial overall storms to the next step meaning it still can be merged with other storms at other iterations.

As mentioned earlier, there are two stop conditions: 1) it reaches the last iteration *k* or 2) there is no member in the candidate set. In our case, $k = \lceil \log_2 8760 \rceil = 14$. At the last iteration level, if the candidate set is not empty, every partial overall storm will be grouped into one big group with current hour id 0 and finalized. In order to make our algorithm more precise and efficient, we defined an object class, called Ostorm, which represents a partial overall storm unless it is in the last iteration. In such case, the overall storms are final. We assume that every overall storm is partial unless they are in the last iteration. In each iteration, each partial overall storm is either: 1) combined with one or more partial overall storms into a bigger partial overall storm if overlapping exists, or 2) converted into another partial overall storm itself for the next iteration. The operations of Ostorm class are as follows:

- IsOverlap: finds whether the current storm is overlapping with another storm.
- IsComparable: checks whether the current storm is comparable with another storm. This is based on the concept in Lemma 3 (local pruning).
- MergeStorm and CompareWithinOrder: merges and compares the given storm with other storms in the fashion as described in Lemma 2 (on-the-fly comparison).
- IsFinal: checks whether an overall storm is final. A final overall storm is one that cannot grow further. This is implemented as part of the prune step mentioned in Lemma 1 (global pruning).

The algorithm for overall storm identification in map-reduce (MR-OSI) is shown in Algorithm 7. The time complexity of MR-OSI algorithm for the worst case scenario is

$O(k)(mapper) + O(k)(partitioner) + O(n + mn + n)(reducer) = O(2k + 2n + mn)$, where *n, m* are the maximum numbers of hourly storms in the first and second hours, respectively, and *k* is the maximum number between *n* and *m*.

The post-processing steps are also required (as we run the raw data one year at a time) to combine an overall storm that starts in one year and ends in the next: 1) between current year and previous year and 2) between current year and following year. For 1), the range of original hour ids from previous year to current year to consider are $[prev.last\_hid_{orig} - g + 1, curr.first\_hid_{orig} + g - 1]$. For 2), the range of original hour ids from current year to following year to consider are $[curr.last\_hid_{orig} - g + 1, next.first\_hid_{orig} + g - 1]$. If we start with the last (or first) year, we only have to compare the current year with the previous (or next) year.

4.2.2 Experimental Results

In the previous approach [62], the experiment was performed on the rainfall dataset, resided in a relational database, using a single server. The server runs on Microsoft® Windows Server® 2008 Enterprise operating system with 2.83 GHz Intel® Xeon® quad-core processors, 20 GB of RAM, 500 GB of local disk, and 10 TB of external disk. The raw data contains 15 months of data from October 2010 to December 2011 covering 37,413 sites in Texas.

In the map-reduce approach [68][76], we use the same dataset (from October 2010 to December 2011) that is in the original text file format rather than relational format, which contains many more sites (165,750 sites) covering Texas and some surrounding areas. These results are used to compare the performance of map-reduce with the original recursive algorithm.

Algorithm7. MapReduce-based Overall Storm Identification

```
    Input:
        - Hourly storm data text files
    Output:
        - Overall storm data text file
1:  class MAPPER
2:  function MAP(key object, value line)
3:      if iteration i = 1 then
4:          //read in the hourly storm inputs record by record.
5:          $hid_{orig}$ ← line.CalHourId() //hour number id in a year, starting from 0 to the last hour of input files
6:          key ← $hid_{orig}$
7:          value ← list of values of Ostorm attributes
8:          Emit(key, value)
9:      else
10:         //read in previously-identified partial overall storms from iteration i-1.
11:         key ← first value in the record, $hid_{curr}$
12:         value ← the remaining values, Ostorm attributes
13:         Emit(key, value)
14: class PARTITIONER
15: function PARTITION(key hid, value Ostorm)
16:     for each record do //hourly or partial overall storm record
17:         hid = floor(hid/2)
18:         Emit(hid, Ostorm)
19:     end for
20: class REDUCER
21: function REDUCE(key hid, [Ostorm1, Ostorm2, …]
22:     //storms come in order based on their hids.
23:     for each key do
24:         for each os ∈ [Ostorm1, Ostorm2, …] do
25:             //assign each os to an array list, aList, in order (based on hid).
26:             if key remains unchanged then
27:                 aList.add(os)
28:             else
29:                 //all hourly or partial overall storms with lower hid are now stored in aList.
30:                 for each storm cos ∈ aList do
31:                     if cos.IsComparable(os) and
32:                       cos.IsOverlap(os) then
33:                         mos ← cos.MergeStorm(os)
34:                         aList.Add(mos, cos.pos) //replace current storm with the merged storm (mos)
35:                         CompareWithinOrder(cos.pos)
36:                         //compare cos with other storms present after its position in aList.
37:                     else
38:                         if IsFinal(os) then //Lemma 1
39:                             MarkAsFinal(os)
40:                 end for
41:         end for
42:     end for
43:     for each storm cos ∈ aList do
44:         if IsFinal(cos) then //Lemma 1 (pruning at the end)
45:             MarkAsFinal(cos)
46:     end for
```

Each text file is for all sites during a single hour and is zipped into one gunzip file. The experiment was done by using a Hadoop® cluster [67] of 1 frontend server and 18 worker nodes. Each worker node contains 3.2 GHz Intel® Xeon® quad-core processors, 4 GB of RAM and 1.5 TB of local disk allocated to HDFS. The server has the same specification but with 3 TB of local disk. The cluster is set up by using ROCKS Cluster 6.3 OS and then installing Hadoop® 1.0.3 on every node.

The comparison between the time taken by the previous implementations and the new map-reduce implementations is shown in Table 4-7. Please also note that the processing time does not include the time taken to load the data into HDFS/SQL. The experiment of the new approach gives the same results for Texas region as the previous approach but is executed significantly faster. The new approach allows programs to be executed distributedly on multiple machines and hence the efficiency of the storm analysis is increased. For local storm (LS) identification, the time improved to 2.43 hours, compared to 53.44 hours in the previous approach. For hourly storms (HS), the map-reduce took 0.93 hours, compared to 6.78 hours in the previous method (DFS). For overall storms (OS), the previous approach took 8.62 hours whereas the map-reduce approach took only 0.67 hours. These show that the new approach is more efficient since it took less time while processing a larger number of sites. Moreover, since we are using map-reduce, the performance can be improved further by increasing the number of nodes.

Notice that in the map-reduce approach we no longer divide Texas into different regions to do analysis due to our focus of this approach, which is to improve the system performance. Therefore, we process the entire grid of raw data together without dividing it. However, we can always divide and analyze each region separately if we want to.

Table 4-7 Comparison of Processing Time between Two Approaches

Regions / Number of Raw Data Records	**Number of Sites**	**Processing Time (in hours)**					
		Previous Apporach *CUAHSI*			**New Approach** *MapReduce*		
		LS	**HS**	**OS**	**LS**	**HS**	**OS**
1. East Texas (48,953,130)	4,643	8.67	1.44	1.24	**<---- 2.43 hours for all sites (including 37,413 sites in Texas) ---->**	**<------------- 0.93 hour for all sites ------------->**	**<------------- 0.67 hour for all sites ------------->**
2. Edwards Plateau (73,415,532)	6,962	8.72	1.23	2.25			
3. High Plains (31,711,927)	3,008	4.5	0.32	0.57			
4. Low Rolling Plains (24,965,521)	2,368	3.35	0.28	0.27			
5. North Central (59,082,957)	5,604	8.66	1.17	1.64			
6. South Central (31,102,334)	2,949	4.28	0.67	0.42			
7. South Texas (31,949,386)	2,933	3.97	0.48	0.60			
8. Lower Valley (5,324,898)	601	0.55	0.07	0.08			
9. Trans-Pecos (65,136,216)	6,177	6.86	0.55	1.16			
10. Upper Coast (22,863,789)	2,168	3.88	0.57	0.39			
TOTAL	**37,413**	**53.44**	**6.78**	**8.62**			

Next, we extend our experiment to the entire set of the raw rainfall data (1997-2012). The experiment is performed on the raw data one year at a time. After processing each year, the post-processing steps are conducted on local storms and overall storms to combine storms that start in one year and end in the next. For hourly storms, the post-processing steps are not required since each raw file is hourly independent and covers all the sites. At any given year, local storm and hourly storm identification processes can be executed concurrently whereas overall storm identification must be executed after the hourly storm identification process.

The standard procedure for local and hourly storm identifications is as follows:

- Un-tar the raw rainfall data, which took about 6 minutes.

- Upload the rainfall data to HDFS™ [67] (about 23 minutes).
- Execute the Java code (MR-LSI/MR-HSI) for computing the local/hourly storms (approximately 1 hour and 50 minutes for MR-LSI and 38 minutes for MR-HSI).

The overall storm identification algorithm used was iterative; hence the output of one iteration is the input for the next iteration. To find the overall storms from the hourly storms, the following standard procedure is performed:

- Upload the hourly storm data to HDFS, which took about 25 minutes.
- Execute the Java code (MR-OSI) for computing the overall storms (about 40 minutes).

The experimental statistics for the entire 16 years of raw rainfall data are listed in Table 4-8 and 4-9. Table 4-8 shows the number of identified storms (local storms, hourly storms, and overall storms) for each year. Table 4-9 shows the processing time in each iteration of MR-OSI program. As you can see, the processing time in each iteration decreases significantly. This is because of the implementations of Lemmas 1 and 3, which prune some storms from being processed and compared in the later iterations. In addition, after iteration 9, the time taken to process in the later iterations becomes almost constant. This is because after iteration 9, the number of partial overall storms that were merged to others becomes very minimal (less than 700 storms, on average).

4.3 Custom Database Schema for Conceptual Storm Outputs

We use a relational database to store the final storm outputs. The database must be designed in such a way that the expressivity and usability features of SQL can be fully utilized in the analysis tasks. SQL and relational databases are proven tools in performing analysis [56][57]. To develop an EER for the storm outputs, we take into account the formalizations of storm concepts as well as the characteristics of the raw rainfall data.

Table 4-8 Number of Storm Records in Each Component for Each Year

Year	Num Raw MPE	LS	HS	OS	Num Storm	Reduction in Raw
	Records	Num Storms	Num Storms	Num Storms	Records	Data Size (%)
1997	731,786,250	3,944,877	298,561	150,886	4,394,324	0.60
1998	1,451,804,250	6,372,104	455,575	235,409	7,063,088	0.49
1999	1,450,644,000	5,842,579	434,440	218,516	6,495,535	0.45
2000	1,453,627,500	6,138,978	439,557	225,029	6,803,564	0.47
2001	1,451,804,250	6,663,672	496,213	248,550	7,408,435	0.51
2002	1,451,804,250	6,827,462	448,670	196,531	7,472,663	0.51
2003	1,451,804,250	7,606,046	441,303	196,330	8,243,679	0.57
2004	1,455,782,250	12,526,769	545,125	237,457	13,309,351	0.91
2005	1,451,804,250	10,169,983	479,560	210,777	10,860,320	0.75
2006	1,450,478,250	10,354,175	519,978	227,517	11,101,670	0.77
2007	1,448,489,250	12,819,729	643,383	282,021	13,745,133	0.95
2008	1,455,782,250	10,371,608	544,036	248,001	11,163,645	0.77
2009	1,451,638,500	10,958,887	547,164	248,294	11,754,345	0.81
2010	1,451,141,250	10,108,909	546,926	247,449	10,903,284	0.75
2011	1,451,804,250	7,143,676	382,009	183,112	7,708,797	0.53
2012	1,455,782,250	9,024,720	481,920	225,075	9,731,715	0.67
TOTAL	**22,515,977,250**	**136,874,174**	**7,704,420**	**3,580,954**	**148,159,548**	**0.66**
AVERAGE	**1,407,248,578**	**8,554,636**	**481,526**	**223,810**	**9,259,972**	**0.66**

Table 4-9 MR-OSI Execution Time in Each Iteration

Year	Iteration Level (processing time in minutes)														TOTAL
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
1997	13.09	2.00	2.00	1.38	1.19	1.13	0.97	0.84	0.86	0.67	0.67	0.65	0.61	0.61	**26.67**
1998	24.15	2.49	1.78	1.44	1.23	1.14	1.04	0.93	0.74	0.68	0.69	0.73	0.64	0.68	**38.36**
1999	24.52	2.53	1.84	1.54	1.23	1.19	1.07	0.94	0.83	0.74	0.82	0.68	0.71	0.67	**39.31**
2000	24.32	2.23	1.65	1.38	1.25	1.08	0.99	0.83	0.73	0.64	0.68	0.69	0.68	0.70	**37.85**
2001	24.39	2.68	1.69	1.54	1.29	1.14	1.02	0.83	0.79	0.68	0.73	0.66	0.67	0.71	**38.82**
2002	25.75	2.68	1.68	1.44	1.23	1.15	0.98	0.89	0.78	0.69	0.73	0.69	0.66	0.66	**40.01**
2003	25.28	2.35	1.78	1.53	1.29	1.18	1.15	0.79	0.78	0.68	0.69	0.68	0.66	0.67	**39.51**
2004	30.01	2.73	2.01	1.50	1.27	1.13	1.03	0.90	0.78	0.69	0.68	0.67	0.66	0.66	**44.72**
2005	25.82	2.33	1.70	1.35	1.26	1.44	1.03	0.88	0.85	0.68	0.68	0.61	0.66	0.66	**39.95**
2006	27.23	3.06	1.93	1.59	1.33	1.35	1.07	0.87	0.80	0.67	0.73	0.68	0.66	0.66	**42.63**
2007	28.11	3.14	2.22	1.65	1.43	1.35	1.13	0.93	0.79	0.68	0.84	0.74	0.68	0.74	**44.43**
2008	27.46	2.88	2.09	1.63	1.39	1.33	1.13	0.94	0.79	0.68	0.73	0.70	0.67	0.68	**43.10**
2009	27.48	3.09	2.05	1.55	1.38	1.30	1.24	0.88	0.79	0.68	0.67	0.73	0.66	0.72	**43.22**
2010	29.73	3.18	1.81	1.50	1.26	1.25	1.08	0.94	0.81	0.68	0.67	0.67	0.67	0.67	**44.92**
2011	25.00	2.18	1.86	1.39	1.17	1.18	1.03	0.90	0.72	0.67	0.73	0.66	0.61	0.61	**38.71**
2012	25.82	2.64	1.91	1.42	1.33	1.20	1.28	0.91	0.73	0.68	0.69	0.73	0.65	0.66	**40.65**
TOTAL	**408.16**	**42.19**	**30.00**	**23.83**	**20.53**	**19.54**	**17.24**	**14.20**	**12.57**	**10.89**	**11.43**	**10.97**	**10.55**	**10.76**	**642.86**
AVERAGE	**25.51**	**2.64**	**1.88**	**1.49**	**1.28**	**1.22**	**1.08**	**0.89**	**0.79**	**0.68**	**0.71**	**0.69**	**0.66**	**0.67**	**40.18**

The developed EER is shown in Figure 4-10. Note that the set of hourly storms related to an overall storm are ordered by hour, because they form a sequence in the formalized ontology. We then map the EER in Figure 4-10 to the database schema using methodology/steps described in [41]. The database schema consists of 8 tables, as shown in Figure 4-11. Note that the table OverallStormTracks is derived from the relationships among the entity types Overall Storm, Hourly Storm, Centers, and Time Point in Figure 4-10.

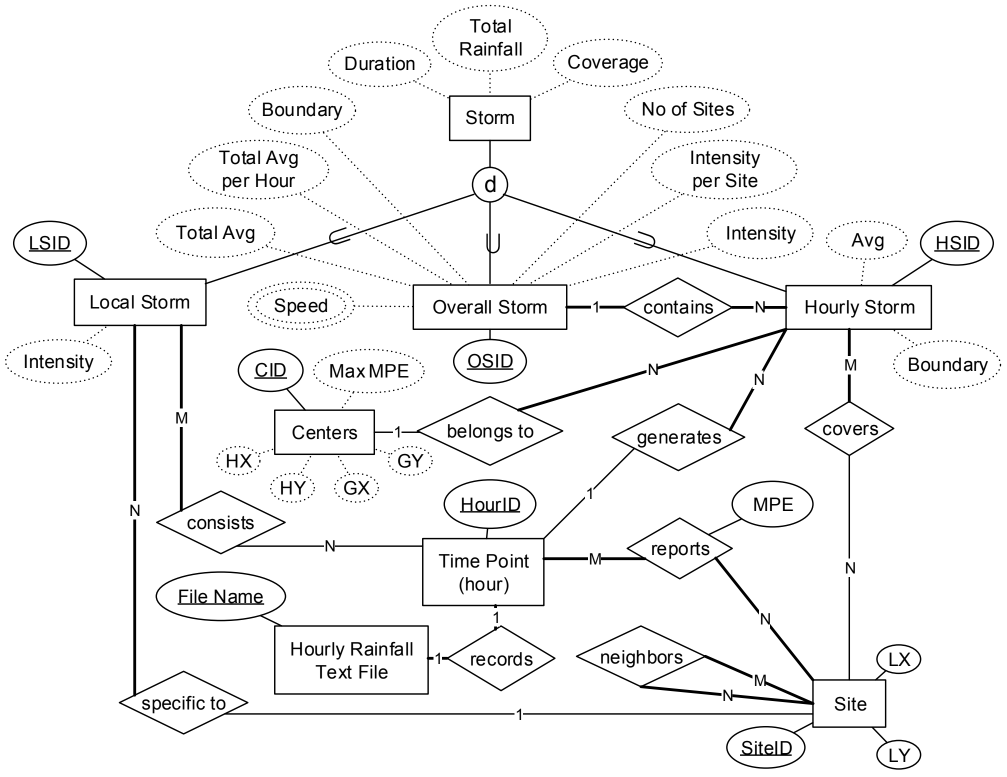


Figure 4-10 EER Diagram for Storing Conceptual Storm Data

LocalStormHours table stores local storms information for each hour of every site. The information includes local storm id, date/time, and precipitation depth (in inches) of the storm for a particular site.

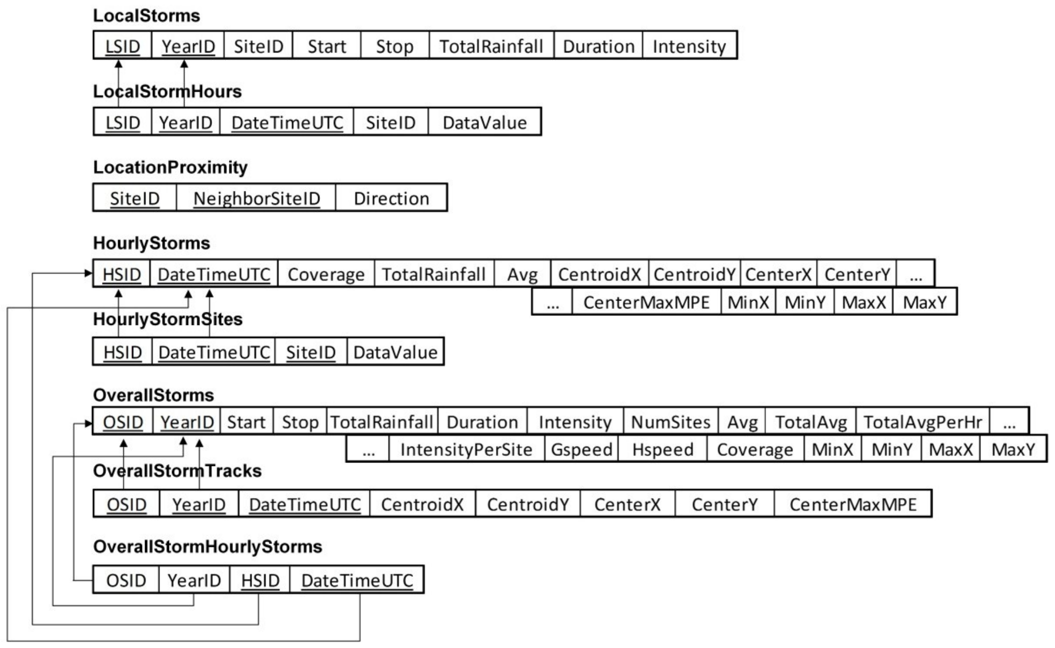

LocalStorms
LSID | YearID | SiteID | Start | Stop | TotalRainfall | Duration | Intensity
LocalStormHours
LSID | YearID | DateTimeUTC | SiteID | DataValue
LocationProximity
SiteID | NeighborSiteID | Direction
HourlyStorms
HSID | DateTimeUTC | Coverage | TotalRainfall | Avg | CentroidX | CentroidY | CenterX | CenterY | ...
... | CenterMaxMPE | MinX | MinY | MaxX | MaxY
HourlyStormSites
HSID | DateTimeUTC | SiteID | DataValue
OverallStorms
OSID | YearID | Start | Stop | TotalRainfall | Duration | Intensity | NumSites | Avg | TotalAvg | TotalAvgPerHr | ...
... | IntensityPerSite | Gspeed | Hspeed | Coverage | MinX | MinY | MaxX | MaxY
OverallStormTracks
OSID | YearID | DateTimeUTC | CentroidX | CentroidY | CenterX | CenterY | CenterMaxMPE
OverallStormHourlyStorms
OSID | YearID | HSID | DateTimeUTC

Figure 4-11 Relational Database Schema for Storing Conceptual Storm Outputs

Its characteristics are summarized into LocalStorms table. Each local storm is uniquely identified by (YearID, LSID) because local storms are identified each year at a time and so the same LSID can exist in different years. HourlyStormSites table stores precipitation value for each site of an hourly storm. The hourly storm characteristics are summarized into HourlyStorms table. Each hourly storm is uniquely identified by (DatetimeUTC, HSID) as hourly storms are identified using one hourly text file at a time so the same HSID can exist in different hourly text file. OverallStormHourlyStorms table stores information of all hourly storms combined into an overall storm. The primary key for this table is (DateTimeUTC, HSID) because an hourly storm can belong to only one overall storm. Technically, this table could be further mapped to HourlyStorms by having (YearID, OSID) as foreign key to OverallStorms. However, we mapped it into a separate table because the hourly storm statistics table (HourlyStorms) are calculated at the end. Having separated OverallStormHourlyStorms table, we do not need to wait until hourly storm identification is concluded in order to identify overall storms. Hourly storm identification and overall storm identification can run concurrently. OverallStormTracks table contains track information of the overall storms for each hour. Storm track information consisting of hydrology-based storm tracks (point with maximum precipitation of a particular hourly storm) and geometry-based storm tracks (centroid of an hourly storm), are the only overall storm characteristic created in a separate table. The remaining characteristics such as speed and total rainfall, are summarized into OverallStorms table. All of the storm characteristics were calculated either during the identification process or post-processing steps by using SQL or additional scripts. Queries SQL1-5 describe how storm statistics are calculated using SQL in the post-processing steps.

SQL1. Calculate storm statistics for local storms

```
SELECT      LSID,YearID,SiteID,
            DATEADD(hh, -1, MIN(DateTimeUTC)) AS Start,
            MAX(DateTimeUTC) AS Stop,
            SUM(DataValue) AS TotalRainfall,
            COUNT(*) AS Duration,
            TotalRainfall/Duration AS Intensity
INTO        LocalStorms table
FROM        LocalStormHours table
GROUP BY    YearID,LSID
```

SQL2. Calculate partial storm statistics (exclude storm centers) for hourly storms

```
SELECT      HSID,DateTimeUTC,
            COUNT(*) AS Coverage,
            SUM(DataValue) AS TotalRainfall,
            TotalRainfall/Coverage AS Avg,
            //HRAP(x,y) below are calculated based on equations described in Chapter 3, Section 3.2.
            SUM(290+((SiteID-15599) MOD 1701))/Coverage AS CentroidX,
            SUM(10+((SiteID-15599) DIV 1701))/Coverage AS CentroidY,
            MAX(DataValue) AS CenterMaxMPE,
            MIN(290+((SiteID-15599) MOD 1701)) AS MinX,
            MIN(10+((SiteID-15599) DIV 1701)) AS MinY,
            MAX(290+((SiteID-15599) MOD 1701)) AS MaxX,
            MAX(10+((SiteID-15599) DIV 1701)) AS MaxY
INTO        PartialHourlyStorms table
FROM        HourlyStormSites table
GROUP BY    DateTimeUTC,HSID
```

SQL3. Calculate storm centers for hourly storms (complete hourly storm statistics table)

```
SELECT    P.HSID,P.DateTimeUTC,P.Coverage,P.TotalRainfall,P.Avg,P.CentroidX,P.CentroidY,
          (SELECT   SUM(290+((SiteID-15599) MOD 1701))/COUNT(*)
          FROM      HourlyStormSites table H
          WHERE     H.DateTimeUTC=P.DateTimeUTC AND H.HSID=P.HSID AND
                    H.DataValue=P.CenterMaxMPE) AS CenterX,
          (SELECT   SUM(10+((SiteID-15599) DIV 1701))/COUNT(*)
          FROM      HourlyStormSites table H
          WHERE     H.DateTimeUTC=P.DateTimeUTC AND H.HSID=P.HSID AND
                    H.DataValue=P.CenterMaxMPE) AS CenterY,
          P.CenterMaxMPE,P.MinX,P.MinY,P.MaxX,P.MaxY
INTO      HourlyStorms table
FROM      PartialHourlyStorms table P
```

SQL4. Calculate overall storms statistics except total average, total average per hour, storm speed, and storm track

```
SELECT    M.OSID,
          M.YearID,
          DATEADD(hh, -1, MIN(T.DateTimeUTC)) AS Start,
          MAX(T.DateTimeUTC) AS Stop,
          DATEDIFF(hh, Start, Stop) AS Duration,
          TotalRainfall/Duration AS Intensity,
          COUNT(T.SiteID) AS NumSites,
          TotalRainfall/NumSites AS Avg,
          Intensity/NumSites AS IntensityPerSite,
          SUM(T.DataValue) AS TotalRainfall,
          COUNT(DISTINCT(T.SiteID)) AS Coverage,
          MIN(290+((T.SiteID-15599) MOD 1701)) AS MinX,
          MIN(10+((T.SiteID-15599) DIV 1701)) AS MinY,
          MAX(290+((T.SiteID-15599) MOD 1701)) AS MaxX,
          MAX(10+((T.SiteID-15599) DIV 1701)) AS MaxY
INTO      PartialOverallStorms table
FROM      OverallStormHourlyStorms table M JOIN HourlyStormSites table T
          ON M.DateTimeUTC=T.DateTimeUTC AND M.HSID=T.HSID
GROUP BY  M.YearID,M.OSID
```

SQL5. Calculate storm statistics: total average and total average per hour, for overall storms

```
SELECT    M.OSID,M.YearID,M.Start,M.Stop,M.Duration,M.Intensity,M.NumSites,M.Avg,
          (SELECT   SUM(H.Avg)
          FROM      HourlyStorms table H
          WHERE     H.DateTimeUTC=M.DateTimeUTC AND H.HSID=M.HSID) AS TotalAvg,
          TotalAvg/M.Duration AS TotalAvgPerHr,
          M.IntensityPerSite,M.TotalRainfall,M.Coverage,M.MinX,M.MinY,M.MaxX,M.MaxY
INTO      OverallStorms table
FROM      PartialOverallStorms table M
```

In Figure 4-12, we repeat Figure 4-11 but highlight with shading the attributes that represent storm characteristics derived through the queries SQL1 through SQL5. Storm tracks and storm speeds are the only two characteristics that were calculated separately by additional scripts. Note that new storm characteristics can also be added as new attributes to the table by using SQL without significant changes or re-running the identification programs.

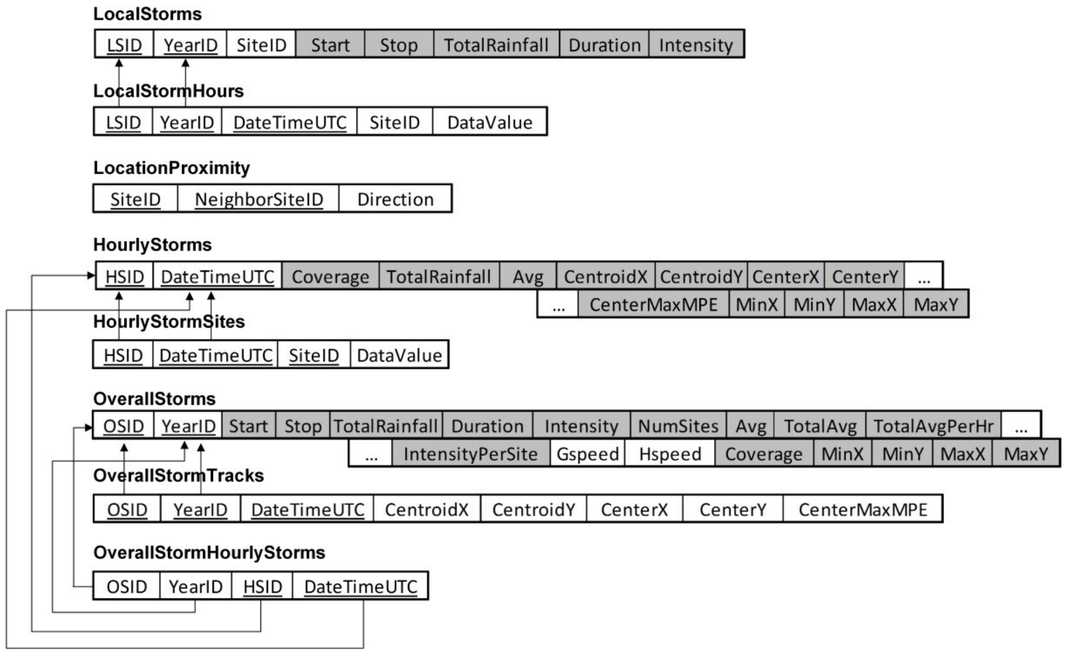

LocalStorms
LSID | YearID | SiteID | Start | Stop | TotalRainfall | Duration | Intensity
LocalStormHours
LSID | YearID | DateTimeUTC | SiteID | DataValue
LocationProximity
SiteID | NeighborSiteID | Direction
HourlyStorms
HSID | DateTimeUTC | Coverage | TotalRainfall | Avg | CentroidX | CentroidY | CenterX | CenterY | ...
... | CenterMaxMPE | MinX | MinY | MaxX | MaxY
HourlyStormSites
HSID | DateTimeUTC | SiteID | DataValue
OverallStorms
OSID | YearID | Start | Stop | TotalRainfall | Duration | Intensity | NumSites | Avg | TotalAvg | TotalAvgPerHr | ...
... | IntensityPerSite | Gspeed | Hspeed | Coverage | MinX | MinY | MaxX | MaxY
OverallStormTracks
OSID | YearID | DateTimeUTC | CentroidX | CentroidY | CenterX | CenterY | CenterMaxMPE
OverallStormHourlyStorms
OSID | YearID | HSID | DateTimeUTC

Figure 4-12 SQL-Derived Storm Characteristics

4.4 Storm Output Verification

In this section, we verify the relational storm outputs by performing a sequence of queries. This verification process complements the unit (module) testing by thoroughly checking the accuracy of each of the resulting storm records. (In fact, thanks to this verification step, we were able to solve a mysterious error (not found during the unit testing), which turns out to be about daylight saving-related functionality in the program.) We divide the verification process into three subsections, each of which targets a particular type of the storms. The verifying queries are designed such that if the storm outputs are correct, an empty set will be returned.

4.4.1 Local Storm Verification

For local storm verification, we first verify that no local storm with 6 (*h*) or more consecutive zero precipitation exists in the LocalStormHours table. The corresponding SQL is shown in query SQL6. For each zero-precipitation record (L_{i-1}) of a local storm, the immediate next hour record (L_i) is checked if it is equal to zero. If it is and repeats for 6 times (i.e., i = 2, 3,…, h) in a row, there are 6 consecutive zeroes in the LocalStormHours table, which means the local storms are incorrectly identified.

SQL6. Checking if 6 (*h*) consecutive zero precipitation exists in the LocalStormHours table

```
SELECT    *
FROM      LocalStormHours L1
WHERE     L1.DataValue=0
          (AND EXISTS   (SELECT   *
                         FROM     LocalStormHours Li
                         WHERE    Li.DataValue=0 AND Li.Time-Li-1.Time=1 AND Li.Time>Li-1.Time AND
                                  Li.YearID=Li-1.YearID AND Li.LSID=Li-1.LSID))*
```

Next, we check if there exist *overlapped* or *connected local storms*. Overlapped local storms refer to two different local storms from the same site location sharing the same time. Connected local storms refer to two consecutive local storms from the same site location, which are within less than 6 hours period. We check for overlapped local storms by query SQL7 and for connected local storms by query SQL8.

SQL7. Checking for overlapped local storms

```
SELECT     DateTimeUTC,SiteID,COUNT(*)
FROM       LocalStormHours
GROUP BY   DateTimeUTC,SiteID
HAVING     COUNT(*)>1
```

SQL8. Checking for connected local storms

```
SELECT    *
FROM      LocalStorms L1
WHERE     EXISTS   (SELECT   *
                    FROM     LocalStorms L2
                    WHERE    L2.Start > L1.Stop AND L2.Start-L1.Stop < 6 AND
                             L2.SiteID=L1.SiteID)
```

Finally, we verify that records of each local storm (in LocalStormHours table) are consecutive. However, there is no direct function in SQL to check for consecutiveness of rows in the table. Our assumption is that if the number of hour rows of a local storm equals to $MAX(hours) - MIN(hours) + 1$ AND the number of distinct hours, we conclude that hour rows of the local storm (in the LocalStormHours table) are consecutive. Query SQL9 shows how consecutiveness of local storm records are verified. We also prove our assumption by contradiction as follows:

Claim. $COUNT(hours) = COUNT(DISTINCT\ hours) = MAX(hours) - MIN(hours) + 1$

$\rightarrow Hour\ rows\ (in\ LocalStormHours\ table)\ are\ consecutive.$

Proof. Suppose that the number of hour rows of a local storm equals to $MAX(hours) - MIN(hours) + 1$ AND the number of distinct hours, but the hour rows of the local storm in the LocalStormHours table are NOT consecutive.

Let: COUNT(hours) = COUNT(DISTINCT hours) (a)

COUNT(hours) = $MAX(hours) - MIN(hours) + 1$ (b)

COUNT(DISTINCT hours) = $MAX(hours) - MIN(hours) + 1$ (c)

Hour rows of the local storm are NOT consecutive. (d)

From (d), there are three cases of rows not being consecutive: 1) duplicate hours only, 2) gap exists only, and 3) both duplicate hours and gap exists. In case 1), COUNT(hours) will always be greater than COUNT(DISTINCT hours), which contradicts (a). In addition, COUNT(hours) will always be greater than $MAX(hours) - MIN(hours) + 1$ (contradicts (b)). In case 2), COUNT(hours) will always equal to COUNT(DISTINCT hours) as no duplicate hour exists. $MAX(hours) - MIN(hours) + 1$ will always be greater than these two COUNTs, which contradicts (b) and (c). In the last case 3), since duplicate hour(s) exists, COUNT(hours) will always be greater than COUNT(DISTINCT hours), which contradicts (a). In addition, $MAX(hours) - MIN(hours) + 1$ will always be greater than COUNT(DISTINCT hours) as gap exists, which contradicts (c). Q.E.D.

SQL9. Checking for consecutiveness of local storm records

```
SELECT   *
FROM     (SELECT     YearID,LSID,COUNT(DateTime) AS NumHrs
          FROM       LocalStormHours
          GROUP BY   YearID,LSID) S
WHERE    S.NumHrs ≠  (SELECT    (MAX(L2.DateTime) - MIN(L2.DateTime))+1
                      FROM      LocalstormHours L2
                      WHERE     S.YearID=L2.YearID AND S.LSID=L2.LSID)
          OR
          S.NumHrs ≠ (SELECT    COUNT(DISTINCT L3.DateTime)
                      FROM      LocalStormHours L3
                      WHERE     S.YearID=L3.YearID AND S.LSID=L3.LSID)
```

4.4.2 Hourly Storm Verification

To verify hourly storm outputs, the following sequence of queries is performed. By definition, there is no zero precipitation in the hourly storm concept. So, query SQL10 ensures that no zero precipitation exists in the HourlyStormSites table. Next, we verify that all sites of an hourly storm are connected (connectivity property). In other words, each site must have at least one neighboring site that is also in the set. We utilized the fact that HRAP coordinate can be derived from SiteID and can be used to identify neighboring relationship. Thus, we created a stored function called isConnected(site1, site2), which will return 1 (TRUE) if two sites are neighbors and 0 (FALSE) if not. Stored function *isConnected* is described in SQL11. Query SQL12 shows how neighboring relationship can be verified by SQL. If the result of SQL12 is empty, the verification is successful.

Query SQL12, however, does not guarantee the following properties: 1) there are no two different hourly storms in the same hour sharing the same site(s), 2) there are no duplicate site(s) in the same hourly storms, and 3) no neighboring relationship exists between any pair of hourly storms during the same hour (maximality property), because by definition, there is no connected site between different hourly storms in the same hour;

otherwise, they would be combined together in the first place. (Note that the second property (checking for duplicate site in the same hourly storms) will always hold as it is enforced by the primary key constraint (DateTime,HSID,SiteID) on the HourlyStormSites table.) To check if the first property holds, query SQL13 is performed. Query SQL14 is used to check for the third property.

SQL10. Checking if zero precipitation exists in the HourlyStormSites table

```
SELECT      *
FROM        HourlyStormSites
WHERE       DataValue=0
```

SQL11. Stored Function "isConnected"

```
CREATE FUNCTION isConnected(@S1,@S2)
RETURN BIT AS
BEGIN
            SET @S1x = 290+((@S1-15599)%1701)
            SET @S1y = floor(10+((@S1-15599)/1701))
            SET @S2x = 290+((@S2-15599)%1701)
            SET @S2y = floor(10+((@S2-15599)/1701))
            IF ((@S2x=S1x-1) OR (@S2x=S1x) OR (@S2x=S1x+1)) AND
              ((@S2y=S1y-1) OR (@S2y=S1y) OR (@S2y=S1y+1))
              SET @Ans=1
            RETURN @Ans
END
```

SQL12. Checking if all sites of an hourly storm are connected

```
SELECT      *
FROM        HourlyStorms hs1
WHERE       NOT EXISTS  (SELECT     *
                        FROM        HourlyStormSites hs2
                        WHERE       hs1.DateTime=hs2.DateTime AND hs1.HSID=hs2.HSID AND
                                    EXISTS  (SELECT     *
                                            FROM        HourlyStormSites hs3
                                            WHERE       hs2.DateTime=hs3.DateTime AND
                                                        hs2.HSID=hs3.HSID AND
                                                        hs2.SiteID≠hs3.SiteID AND
                                                        isConnected(hs3.SiteID,hs2.SiteID)=1))
```

SQL13. Checking for duplicate site(s) between 2 different hourly storms in the same hour

```
SELECT      DateTime,SiteID,COUNT(*)
FROM        HourlyStormSites
GROUP BY    DateTime,SiteID
HAVING      COUNT(*)>1
```

SQL14. Checking for connected site(s) between 2 different hourly storms in the same hour

```
SELECT     *
FROM       HourlyStormSites hs1
WHERE      EXISTS   (SELECT    *
                     FROM      HourlyStormSites hs2
                     WHERE     hs1.DateTime=hs2.DateTime AND hs1.HSID < hs2.HSID AND
                               isConnected(hs1.SiteID,hs2.SiteID)=1)
```

Note that in query SQL14 hs1.HSID < hs2.HSID helps in reducing the number of checks as the same pair will not be compared twice. If the results of the above queries (SQL10, 12, 13, and 14) are all empty set, we can ensure that the hourly storm data are correctly identified.

4.4.3 Overall Storm Verification

In overall storm verification, we first make sure that all hourly storms are processed and each of them belongs to only one overall storm. These two steps can be checked by using queries SQL15 and SQL16, respectively.

SQL15. Checking if all hourly storms were processed

```
SELECT     COUNT(DISTINCT DateTime,HSID)
FROM       HourlyStormSites
=
SELECT     COUNT(DISTINCT DateTime,HSID)
FROM       OverallStormHourlyStorms
```

SQL16. Checking if an hourly storm belongs to only one overall storm

```
SELECT     oshs.DateTime,oshs.HSID,COUNT(*)
FROM       OverallStormHourlyStorms oshs
GROUP BY   oshs.DateTime,oshs.HSID
HAVING     COUNT(*)>1
```

Next, we verify the grouping-window (*g*) and spatial-window (*s*) requirements. In our case, g = 1 and s = 1. That is, for each hour of an overall storm, there must be at least one common site shared between two consecutive hourly storms. These two requirements can be verified by SQL as described in query SQL17 and SQL18. Query SQL17 lists all sites for each hour of each overall storm and filters out 1-hour overall storm from consideration. Query SQL18 uses the resulting table from query SQL17 to check for both requirements. In query SQL18, an overall storm is considered valid only if each of its hours (except the last hour) has the next hour (*g*=1) with at least one site (*s*=1) in common.

Finally, we verify the maximality property of overall storms. Any pair of consecutive hourly storms that has some site(s) in common should be combined to the same overall storm. Query SQL19 checks if there exist two consecutive hourly storms sharing at least one site but end up in different overall storms.

Note that the storm statistics are not required in the verification process since they are derived from SQL as described in SQL1, 2, 3, 4, and 5.

SQL17. List all sites for each hour of each overall storm and filter out 1-hour overall storms

```
SELECT     T1.YearID,T1.OSID,T1.DateTime,T1.SiteID
FROM       (SELECT     M1.YearID,M1.OSID,S1.DateTime,S1.SiteID
           FROM        OverallStormHourlyStorms M1 JOIN HourlyStormSites S1 ON
                       M1.HSID=S1.HSID AND M1.DateTime=S1.DateTime) T1
           JOIN
           (SELECT     M2.YearID,M2.OSID
           FROM        OverallStorms M2
           WHERE       M2.Duration > 1) T2
           //WHERE-clause specifies that only overall storms with 2 or more durations are considered.
           ON T1.YearID=T2.YearID AND T1.OSID=T2.OSID
```

SQL18. Verify if the spatial-window (*s*) and grouping-window (*g*) hold

```
SELECT     T1.YearID,T1.OSID //Invalid overall storms are reported.
FROM       (SELECT     DISTINCT YearID,OSID,DateTime
           FROM        <SQL17 resulting table>) T1
WHERE      NOT EXISTS  (SELECT     *
                       FROM        <SQL17 resulting table> T2
                       WHERE       T1.YearID=T2.YearID AND T1.OSID=T2.OSID AND
                                   T1.DateTime=T2.DateTime AND
                                   EXISTS     (SELECT     *
                                              FROM        <SQL17 resulting table> T3
                                              WHERE       T2.YearID=T3.YearID AND T2.OSID=T3.OSID AND
                                                          T3.DateTime-T2.DateTime=1 AND //Check for g
                                                          T2.SiteID=T3.SiteID AND //Check for s
                                                          T3.DateTime>T2.DateTime))
//For each correct overall storm, one row is reported for the last hour time.
//Having this GROUP BY- and HAVING- clauses, an empty set is returned if all overall storms are correct.
GROUP BY   T1.YearID,T1.OSID
HAVING     COUNT(*)>1
```

SQL19. Verify maximality property of overall storms

```
SELECT   *
FROM     OverallStormHourlyStorms M1 JOIN HourlyStormSites S1
         ON M1.HSID=S1.HSID AND M1.DateTime=S1.DateTime
WHERE    EXISTS  (SELECT   *
                  FROM     OverallStormHourlyStorms M2 JOIN HourlyStormSites S2
                           ON M2.HSID=S2.HSID AND M2.DateTime=S2.DateTime
                  WHERE    (M1.HSID≠M2.HSID OR M1.DateTime≠M2.DateTime) AND //Diff. hourly storms
                           S2.DateTime-S1.DateTime=1 AND //Within 1 hour different
                           S1.SiteID=S2.SiteID AND //Share at least 1 site
                           (M1.YearID≠M2.YearID OR M1.OSID≠M2.OSID) AND //But end up in diff. os
                           S2.DateTime>S1.DateTime=1 //Eliminate redundant comparisons)
```

4.5 Related Work

There are three main parts of related work: 1) storm characteristics analysis, and 2) map-reduce framework for spatial data computing, and 3) iterative map-reduce.

4.5.1 Storm Characteristics Analysis

Several studies suggest that storm characteristics analysis can be done in various ways, such as through its statistical properties, depth-duration frequency (DDF [50]), or focusing on its extreme precipitation values. Asquith [36] studies storm statistical characteristics including the mean (average) of storm inter-event time, storm depth, and storm duration by analyzing hourly precipitation data retrieved from National Weather Service (NWS) [42]. The data contains 155 million values covering 774 sites in Eastern New Mexico, Texas, and Oklahoma. The storm characteristics results are used to help in designing and creating a new runoff control structure. The outputs are in two formats: maps and tables. [36]'s raw data is stored in file and folder format which raises the difficulty in combining all data across an enormous number of folders and processing them together. Consequently, a huge manual effort is needed to do the analysis. In addition, its analysis has been location-specific (site-specific and regional-specific). So, the storm-specific information is lacking from the work.

In [37][38], Asquith and Roussel study storm characteristics through its Depth-Duration Frequency (DDF [50]) property. [37] presents a procedure to develop a DDF at any location in Texas for the following 14 storm durations: 15, 30, and 60 minutes; 1, 2, 3, 6, 12, and 24 hours; and 1, 2, 3, 5, and 7 days with recurrence intervals ranging from 2 to 500 years. DDF is an estimated depth of the storm given its duration and frequency (recurrence time). It is very important when creating an efficient control structure such as storm drains or parking lots. It is also used to design efficient river flow and flood prediction models. As a result, it has to be very accurate. To calculate DDF for a storm duration and frequency at any location, we need three storm depths (in inches) retrieved from three maps (location, scale, and shape parameter maps) for that storm duration and a storm intensity (in inches per hour) retrieved from precipitation intensity-duration curve of that storm frequency. Then, plug all values into the equation given in the paper [37] and the result is an estimated storm depth for that particular storm.

[38] is an extension of [37]. However, it does not require users to do the calculation themselves. It provides pre-computed DDF maps, which are ready to use. The set of storm durations and storm frequencies, however, are different from [37]. The storm durations only include 15, and 30 minutes; 1, 2, 3, 6, and 12 hours; and 1, 2, 3, 5, and 7 days and the storm frequencies only include 2, 5, 10, 25, 50, 100, 250, and 500 years. One of the key tasks of [37][38] is to create location, scale, and shape parameter maps used in the approach. To create such maps, this work uses storm data from National Climatic Data Center (NCDC) [46]. However, only location-specific storm data (by county) is provided by NCDC. So, generating these required maps will be limited to location-specific storm data. In addition, even though NCDC stores storm data in a database, CUAHSI ODM was not mentioned as its database schema. As a result, incorporating our storm data (storm-specific) into these two works may enhance their analytic capabilities.

Lanning-Rush [39] studies storm characteristics by focusing on its extreme precipitation (EP) values. The extreme precipitation depth refers to one that exceeds 100-year or greater storm depth. Unlike [36] that considers all storms, only extreme storms were taken into account in this work. Unlike [37][38] that the inputs are storm duration, frequency, and location, it only takes storm duration and area as inputs. The goal of this work is to create the extreme precipitation curve which can be used to estimate extreme precipitation depth for a particular storm duration and area. The EP curves are developed from 24 extreme storms out of 213 notable storms. They select storm durations to include 1, 2, 3, 4, 5, and 6 days and the areas include High Plains, Low Rolling Plains, North Central, Edwards Plateau, South Central, South Texas, East Texas, Upper Coast, and Lower Valley in Texas. Trans-Pecos area, however, was excluded due to the lack of its storm data.

4.5.2 MapReduce Framework for Spatial Data Computing

Since map-reduce has become the de-facto framework for the data-intensive applications, it is now being used for big data related to geography, sciences, humanities, statistics, etc. There has been previous work for spatial data analysis in map-reduce. Lu [88] uses map-reduce framework for analyzing and visualizing big spatial-temporal data. Complex climate datasets are used in their case study. Cary [63] shows the construction of R-Tree index from spatial data in map-reduce. It uses the mappers to partition the data and then every partition is sent to a different reducer which in turn build the R-Tree index on the input. Google used the map-reduce framework to study road alignments by combining satellite and vector data [64]. The work focused more on the complexity of the problem than the implementation in map-reduce. Hadoop was also used to build octrees for later use in earthquake simulations at a large scale [65]. Octrees were built in the bottom up fashion in their approach. Mappers were used to first generate the leaf nodes and then reductions were performed to merge two homogeneous leaf nodes into a sub tree. This was done in iterations to build the final sub tree.

4.5.3 Iterative MapReduce

Most data analysis and mining algorithms are iterative in nature and hence require repetitive map-reduce jobs. An example of iterative map-reduce applications is breadth-first search (BFS) [90]. Kondekar [89] developed a parallel breadth-first heuristic search (PBFHS) algorithm for solving N-Puzzle problem based on iterative map-reduce. Lin [91] mentions that map-reduce framework can be used to efficiently solve complex search problems (such as PageRank), which are iterative and require high computational capabilites. An efficient implementation of iterative graph search algorithms is developed on the map-reduce framework [91]. Twister [92] and HaLoop [93] are developed on top of hadoop for supporting iterative computations in the map-reduce framework. Twister provides long-running map and reduce tasks with a cacheable distributed memory, which is used to prevent retrieving the same data multiple times from the disks. HaLoop caches the loop-invariant data structures and hence the loading and shuffling cost is reduced in the subsequent iterations. There has also been recent work in the development of incremental processing systems such as Incoop [94] and Google’s Percolator [95]. Incoop is a hadoop-based incremental processing system whereas Google’s Percolator is an incremental processing system based on BigTable [78].

Chapter 5

Analysis and Mining of Conceptual Data

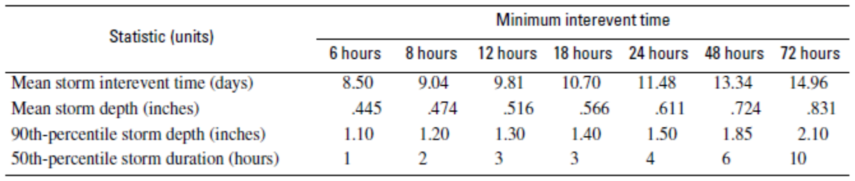
After the conversion algorithms are executed on the raw rainfall data in the previous chapter, we now have the conceptual storm data stored in a relational database. The raw data contains 16 years of historical rainfall data from 1997 to 2012. The size of the conceptual relational storm outputs are significantly reduced (< 1%) when compared to the size of the raw rainfall data as the superfluous parts are removed and the raw rainfall data is summarized/converted into meaningful rainstorm concepts. The analysis and mining tasks can then be easily conducted on the conceptual storm data either directly using SQL or by extracting the conceptual storm data from the relational database. In this chapter, we show some examples of how analysis and mining tasks can be performed on the conceptual storm data. We divide the analysis and mining tasks into two groups: 1) traditional hydrology analysis and 2) more general storm analysis and mining. Traditional hydrology analysis is discussed in Section 5.1. More flexible/robust storm analysis and mining is discussed in Section 5.2.

5.1 Traditional Hydrology Analysis

Most traditional rainfall analysis is based on location, meaning each site or region (set of sites) is considered separately when analyzing a storm. The goal is to investigate characteristics of storms at a particular location. These analyzed characteristics will then be used in creating an efficient/cost-effective hydraulic control structure (e.g., storm drain (to route localized runoff) and parking lot design for effective draining), and designing river flow or flooding prediction models [80]. The traditional rainfall analysis can be divided into three categories [36][37][38][39][80][99]: 1) storm statistical properties, 2) relationships between/among characteristics of storms, and 3) focusing on extreme precipitation values of storms.

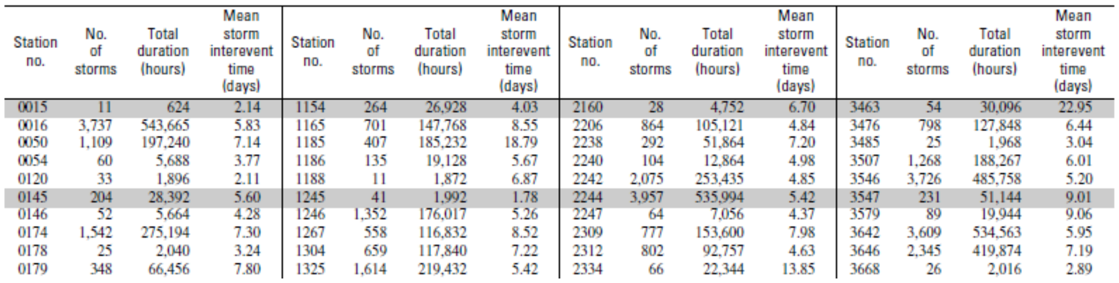
5.1.1 Storm Statistical Analysis

In storm statistical properties analysis, each characteristic of storms is analyzed separately for its statistical properties. The storm characteristics include inter-event time, total rainfall, and duration. There are six main statistics studied: mean (average) inter-event time between storms, mean total rainfall at a particular location, number of storms during the study period, total duration of all local storms, distribution of total rainfall values over the various storms, and distribution of storm durations [36]. Each statistical property is analyzed for a particular inter-event time. The considered set of inter-event times consists of 6, 8, 12, 18, 24, 48, and 72 hours. Figure 5-1 shows some examples of storm statistical properties analysis.



Statistic (units)	Minimum interevent time						
	6 hours	8 hours	12 hours	18 hours	24 hours	48 hours	72 hours
Mean storm interevent time (days)	8.50	9.04	9.81	10.70	11.48	13.34	14.96
Mean storm depth (inches)	.445	.474	.516	.566	.611	.724	.831
90th-percentile storm depth (inches)	1.10	1.20	1.30	1.40	1.50	1.85	2.10
50th-percentile storm duration (hours)	1	2	3	3	4	6	10

a) Selected Storm Characteristics for Single Site by Different Inter-Event Times



Station no.	No. of storms	Total duration (hours)	Mean storm interevent time (days)	Station no.	No. of storms	Total duration (hours)	Mean storm interevent time (days)	Station no.	No. of storms	Total duration (hours)	Mean storm interevent time (days)	Station no.	No. of storms	Total duration (hours)	Mean storm interevent time (days)
0015	11	624	2.14	1154	264	26,928	4.03	2160	28	4,752	6.70	3463	54	30,096	22.95
0016	3,737	543,665	5.83	1165	701	147,768	8.55	2206	864	105,121	4.84	3476	798	127,848	6.44
0050	1,109	197,240	7.14	1185	407	185,232	18.79	2238	292	51,864	7.20	3485	25	1,968	3.04
0054	60	5,688	3.77	1186	135	19,128	5.67	2240	104	12,864	4.98	3507	1,268	188,267	6.01
0120	33	1,896	2.11	1188	11	1,872	6.87	2242	2,075	253,435	4.85	3546	3,726	485,758	5.20
0145	204	28,392	5.60	1245	41	1,992	1.78	2244	3,957	535,994	5.42	3547	231	51,144	9.01
0146	52	5,664	4.28	1246	1,352	176,017	5.26	2247	64	7,056	4.37	3579	89	19,944	9.06
0174	1,542	275,194	7.30	1267	558	116,832	8.52	2309	777	153,600	7.98	3642	3,609	534,563	5.95
0178	25	2,040	3.24	1304	659	117,840	7.22	2312	802	92,757	4.63	3646	2,345	419,874	7.19
0179	348	66,456	7.80	1325	1,614	219,432	5.42	2334	66	22,344	13.85	3668	26	2,016	2.89

b) Selected Storm Characteristics Defined by 6-Hour Inter-Event Time

Figure 5-1 Examples of Storm Statistical Properties Analysis [36]

Since storm total rainfall and duration are pre-calculated and stored in the statistics table (i.e., LocalStorms table), their statistical properties can easily be calculated using SQL. The total number of storms can be calculated by counting number of rows in the LocalStorms table as each row in the table represents one local storm. Query SQL20 describes how averaged total rainfall, number of storms, and total duration can be done. Query SQL21 shows how distribution of storm characteristic can be determined. Inter-event time characteristic, however, was not pre-calculated but can easily be calculated using SQL as shown in query SQL22. Because ids of local storms are numbered sequentially (i.e., *i*, *i+1*, *i+2*, ..., where *i* is the first id of local storm at a given site location), the inter-event time between two consecutive local storms can be determined by L2.Start – L1.Stop, where L1 is the current local storm and L2 is the next local storm at the same site.

SQL20. Determine mean total rainfall, number of storms, total duration

```
SELECT     (AVG(TotalRainfall)|COUNT(*)|SUM(Duration))
FROM       LocalStorms
[WHERE     (SiteID = <site>|SiteID IN <region>)]
```

SQL21. Determine distribution of storm characteristic (e.g., total rainfall, duration)

```
SELECT     (TotalRainfall|Duration), COUNT(*)
FROM       LocalStorms
[WHERE     (SiteID = <site>|SiteID IN <region>)]
GROUP BY   (TotalRainfall|Duration)
ORDER BY   (TotalRainfall|Duration) (ASC|DESC)
```

SQL22. Determine mean storm inter-event time

```
SELECT     AVG(L2.Start–L1.Stop)
FROM       LocalStorms L1 JOIN LocalStorms L2 ON L1.YearID = L2.YearID AND L1.LSID = L2.LSID-1
[WHERE     (L1.SiteID = <site>|L1.SiteID IN <region>)]
```

In our analysis, we use *h* = 6 hours as the inter-event time. To determine the statistical properties for other inter-event times (*h* = 8, 12, 16, …), we just need to change the inter-event-count parameter in the local storm identification program and re-run it.

5.1.2 Correlation Analysis among Strom Characteristics

In this type of analysis, each pair or group of storm characteristics are analyzed together. There are five categories [80][37][38][96]: depth-duration, intensity-duration, depth-duration-frequency, intensity-duration-frequency, and depth-area-duration.

5.1.2.1 Depth-Duration

In depth-duration analysis, two storm characteristics are analyzed together: cumulative depth and duration. Cumulative depth monotonically increases as duration increases, because additional rainfall is added at each hour. The goal is to generate a *cumulative depth diagram* (a graph plot between cumulative depths (Y) and its associated time point in storm duration (X)) for each local storm for a given location. The location here can be a single site or a region (set of sites). Figure 5-2 is an example of cumulative depth diagram of a local storm at site id, 288096. Since cumulative depth is not pre-calculated in the local storm statistics table, we use a nested query in the SELECT-clause to calculate it. For a single site location, the cumulative depths can be calculated by using the query SQL23. Then, to create a cumulative diagram for a local storm, we just need to save the query SQL23 results in a separate table and query from it by specifying the local storm id (YearID, LSID).

SQL23. Calculate cumulative depths of local storms for a single site

```
SELECT      L1.YearID,L1.LSID,L1.Time,
            (SELECT     SUM(L2.DataValue)
            FROM        LocalStormHours L2
            WHERE       L2.YearID=L1.YearID AND L2.LSID=L1.LSID AND
                        L2.Time ≤ L1.Time) AS CDepth //cumulative depth
FROM        LocalStormHours L1
WHERE       L1.SiteID = <site>
ORDER BY    L1.YearID,L1.LSID,L1.Time ASC
```

To create a cumulative depth diagram for a region (e.g., Tarrant county, Texas), two methods are used: *arithmetic mean method* and *Thiessen polygons method* [96]. In our analysis, we use arithmetic mean method. Local storms occurring at the same time (same start and end times) within a region are considered together when creating a cumulative depth diagram. In the arithmetic mean method, a cumulative depth diagram is created based on the averaged cumulative depths of the local storms.

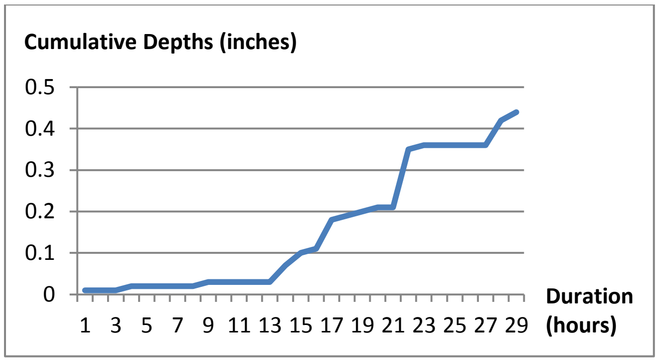


Figure 5-2 Cumulative Depth Diagram of Local Storm at Site Location 288096

Queries SQL24, 25, and 26 describe how a cumulative depth diagram for a given region can be calculated by using SQL. First, we extract a list of different time periods that local storms (within the region) occur at the same time (see query SQL24). (We can also specify duration condition (such as Duration = 10) as depth-duration analysis normally considers each duration at a time.) Second, we compute all related cumulative depths for every site in the region in query SQL25. Finally, we calculate a representative cumulative depth diagram for the time period for the region in query SQL26.

SQL24. Extract all time periods of local storms for a region

```
SELECT      DISTINCT Start,Stop
FROM        LocalStorms
WHERE       SiteID IN <region> [AND Duration = <d durations>]
```

SQL25. Compute all cumulative depths for all sites in the region

```
SELECT      L1.YearID,L1.SiteID,L1.LSID,L1.Time,
            (SELECT     SUM(L2.DataValue)   FROM    LocalStormHours L2
            WHERE       L2.YearID=L1.YearID AND L2.LSID=L1.LSID AND L2.Time ≤ L1.Time) AS CDepth
FROM        LocalStormHours L1
WHERE       L1.SiteID IN <region>
ORDER BY    L1.YearID,L1.SiteID,L1.LSID,L1.Time ASC
```

SQL26. Calculate avg. cumulative depths for the specified time period from SQL24 using table from SQL25

```
SELECT      A.Time,A.AVG(CDepth) AS ACDepth //averaged cumulative depth
 FROM       <resulting table from SQL25> A
 WHERE      EXISTS      (SELECT  *          FROM    LocalStorms L
                        WHERE    A.YearID=L.YearID AND A.LSID=L.LSID AND
                                 L.Start=<Start from SQL24> AND L.Stop=<Stop from SQL24>)
 GROUP BY   A.Time
```

In hydrology, cumulative depth and duration can be approximated by:

$$P = at^n \qquad (4)$$

where P is precipitation (mm), t is duration, a is coefficient, and n is exponent where $0 < n < 1$ [80]. This equation is limited to the location, where it is derived from. To determine a and n, all cumulative depth diagrams for a given location (i.e., site or region) are taken into account.

5.1.2.2 Intensity-Duration

In this analysis, storm intensity and duration are analyzed together. The storm intensity can increase or decrease during the storm duration, usually decreases at the end of the duration. To do analysis, a *hyetograph* is created for each local storm for a given location [96]. *Hyetograph* is a graph plot between storm intensity (Y) and duration (X) [96]. Figure 5-3 is an example of hyetograph of a local storm at site location 288096. Since our raw data is reported hourly and independently (i.e., each storm intensity for an hour is reported individually), storm intensity for each hour in storm duration can easily be extracted. Query SQL27 shows how hyetograph can be created for a single site.

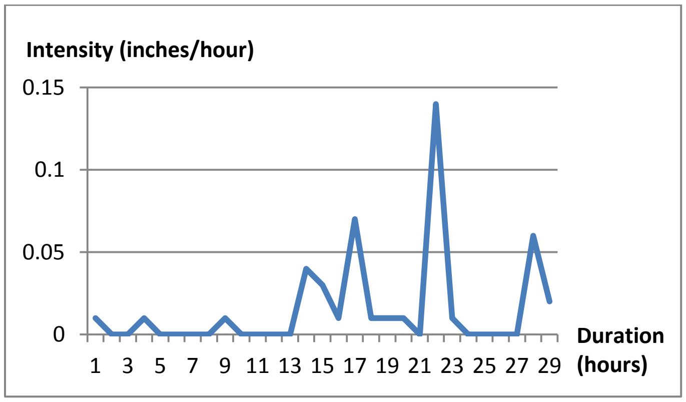


Figure 5-3 Hyetograph of Local Storm at Site Location 288096

SQL27. Extract all storm intensities of local storms for a single site

```
SELECT      YearID,LSID,Time,DataValue
FROM        LocalStormHours
WHERE       SiteID = <site>
ORDER BY    YearID,LSID,Time ASC
```

Similar to depth-duration analysis, to create a hyetograph for a region, local storms occurring at the same time within the region are considered together. That is, a hyetograph is created based on the averaged intensity of these local storms across the sites. Queries SQL24 and 28 describe how intensities of local storms for a given region can be calculated.

SQL28. Calculate averaged intensities for a region given the specified time period from SQL24

```
SELECT      A.Time,A.AVG(DataValue) AS AIntensity //averaged intensity
 FROM       LocalStormHours A
 WHERE      A.SiteID IN <region> AND
            EXISTS      (SELECT  *
                         FROM    LocalStorms L
                         WHERE   A.YearID=L.YearID AND A.LSID=L.LSID AND
                                 L.Start = <Start from SQL24> AND L.Stop = <Stop from SQL24>)
 GROUP BY   A.Time
```

Since storm intensity and cumulative depth are related, the equation (4) can be converted to relationship between storm intensity and duration by dividing both sides of the equation by time *t* as follows [80]:

$$\frac{P}{t} = \frac{at^n}{t}$$

$$i_0 = at^{n-1} \tag{5}$$

where i_o is storm intensity. As a result, to determine *a* and *n*, we can either use hyetographs or cumulative depth diagrams.

5.1.2.3 Depth-Duration-Frequency (DDF)

In this analysis, three storm characteristics are analyzed: cumulative depth, duration, and frequency. As mentioned, storm duration are pre-calculated and stored in the LocalStorms table. However, storm cumulative depth and frequency are not. Storm cumulative depth can easily be calculated as described in query SQL23. Storm frequency is a count of storms with the same total rainfall and duration for a given site. This characteristic can also be calculated by using SQL as shown in query SQL29.

SQL29. Storm frequencies for a given site, ordered by duration and total rainfall

```
SELECT     Duration,TotalRainfall,COUNT(*)
FROM       LocalStorms
WHERE      SiteID = <site>
GROUP BY   Duration,TotalRainfall
ORDER BY   Duration,TotalRainfall ASC
```

The goal of this analysis is to create DDF curves for a given location. Figure 5-4 shows an example of DDF curves for a single site. To create DDF curves for a site, one of the main steps is to calculate annual maximum total rainfall for each year and storm duration. For example, a set of selected storm durations = {2-, 4-, 8-, 12-, 24-, 36-, 72-hour durations} and we have 16 years of storm data. So, we need to calculate 16 x 7 = 112 annual maximum total rainfall values. This calculation can easily be done by a single query as described in query SQL30. Then, by using the results from both queries SQL29 and 30 and hydrology methods, a final set of DDF curves is determined and plotted [80][96].

SQL30. Find annual maximum total rainfall values for a site

```
SELECT     YearID,Duration,MAX(TotalRainfall)
FROM       LocalStorms
WHERE      SiteID = <site>
GROUP BY   YearID,Duration
ORDER BY   YearID,Duration ASC
```

To create DDF curves for a region, DDF curves for each site in the region are created. Then, by applying hydrology analysis (e.g., frequency analysis, Weibull's plotting formula [80]), the final DDF curves for a region is identified.

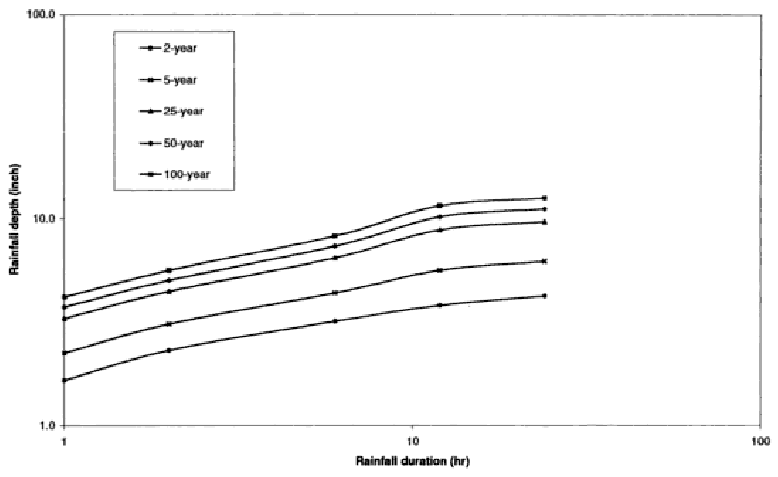


Figure 5-4 Example of DDF Curves for Single Site [80]

5.1.2.4 Intensity-Duration-Frequency (IDF)

In this analysis, three characteristics of storms are analyzed: intensity, duration, and frequency. DDF and IDF are related. To create an IDF curve, the steps are similar to the ones of DDF except that annual maximum intensity (as opposed to annual maximum total rainfall value) for each combination of a year and a storm duration is retrieved. The corresponding SQL is shown in query SQL31.

SQL31. Find annual maximum intensities for a site

```
SELECT     YearID,Duration,MAX(Intensity)
FROM       LocalStorms
WHERE      SiteID = <site>
GROUP BY   YearID,Duration
ORDER BY   YearID,Duration ASC
```

5.1.2.5 Depth-Area-Duration (DAD)

This analysis only applies to local storms that cover multiple sites (region). In particular, local storms that cover a large area are analyzed. The DAD analysis helps determine the distribution rate of the large storm. Three storm characteristics are analyzed: cumulative depth, area, and duration. An area of large local storm refers to a set of sites having non-zero precipitations in the same hours. The goal is to create DAD curves for a large local storm. Figure 5-5 shows an example of DAD curves of a storm. There are two methods to create DAD curves: *mass-curve* and *incremental isohyetal* [80]. We chose mass-curve method to create DAD curves. Two of the main steps in creating DAD curves are to: 1) compute cumulative rainfall values (depths) for each site covered by a storm and 2) calculate summation and mean rainfall values for each time point across sites of the storm. Figure 5-6 shows some examples of the results from these steps. The remaining steps are to use hydrology methods to determine the final DAD curves [80]. The two main steps can be done by using queries SQL32 and 33, respectively.

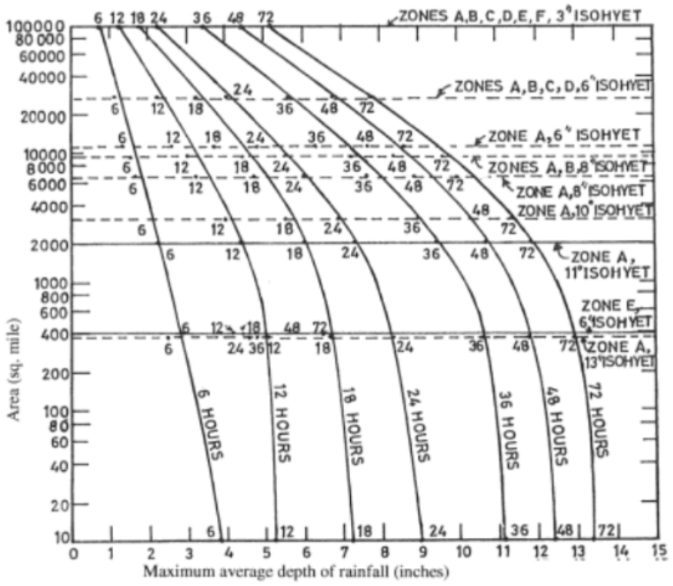


Figure 5-5 Example of DAD Curves of Large Storm [80]

5.1.3 Analysis of Extreme Rainfall Events

In this type of analysis, only extreme storms are considered [39]. An extreme storm refers to a large local storm (cover many sites) with intensity more than 150 mm/hr [80]. The enveloped curve for extreme storm can be mathematically described as shown in equation (6).

SQL32. Compute mass-curve table for a large *n*-hour local storm

```
SELECT    L1.YearID,L1.LSID,L1.SiteID,L1.Time,
          (SELECT  SUM(L2.DataValue)
          FROM     LocalStormHours L2
          WHERE    L2.YearID=L1.YearID AND L2.SiteID=L1.SiteID AND L2.LSID=L1.LSID
                   AND L2.Time ≤ L1.Time) AS CDepth
FROM      LocalStormHours L1
WHERE     EXISTS   (SELECT     *
                   FROM        LocalStorms L3
                   WHERE       L3.YearID=L1.YearID AND L3.SiteID=L1.SiteID AND L3.LSID=L1.LSID
                               AND L3.YearID=<year> AND L3.SiteID IN <region> AND L3.Duration=<n>
                               AND L3.Start=<time1> AND L3.Stop=<time2>)
ORDER BY  L1.YearID,L1.LSID,L1.SiteID,L1.Time ASC
```

SQL33. Compute summation and mean rainfall values for each time point of the storm

```
SELECT    L1.Time,SUM(L1.DataValue),AVG(L1.DataValue)
FROM      LocalStormHours L1
WHERE     EXISTS   (SELECT     *
                   FROM        LocalStorms L2
                   WHERE       L2.YearID=L1.YearID AND L2.SiteID=L1.SiteID AND L2.LSID=L1.LSID
                               AND L2.YearID=<year> AND L2.SiteID IN <region> AND L2.Duration=<n>
                               AND L2.Start=<time1> AND L2.Stop=<time2>)
GROUP BY  L1.Time ASC
```

$$P = 4.8685t^{0.4888} \tag{6}$$

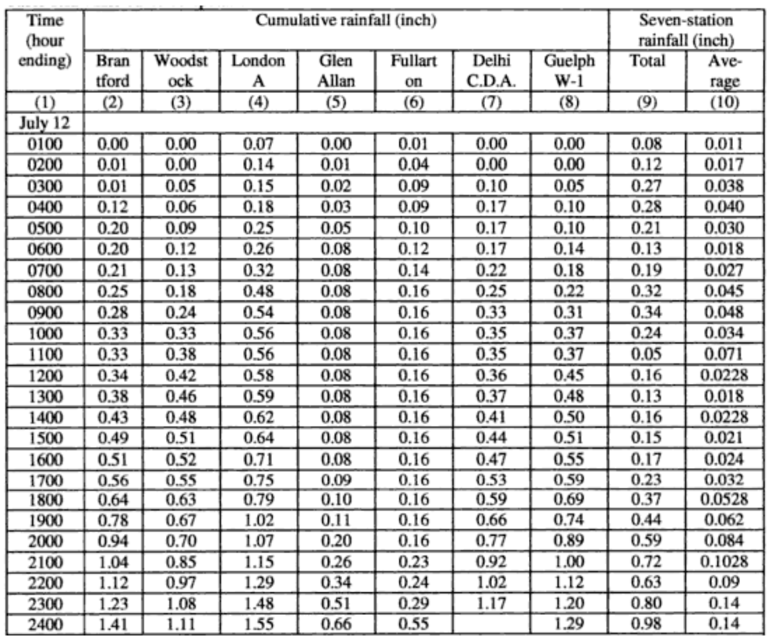
The equation is derived from the world's greatest observed rainfall events [97]. All analysis mentioned previously can also be applied to extreme rainfall events. Query SQL34 show an example of how extreme rainfall events can be analyzed.

SQL34. Counting the number of extreme rainfall events occurred at site id 355478 in the past 10 years

```
SELECT    COUNT(*)
FROM      LocalStorms
WHERE     SiteID=355478 AND Start BETWEEN 2004 AND 2014 AND Intensity > 150 mm/hr
```

5.2 More General Storm Analysis and Mining

In this section, we show how more flexible and robust analysis and mining can be done on our storm data. We divide this section into two subsections: storm analysis and storm mining.



Time (hour ending)	Cumulative rainfall (inch)							Seven-station rainfall (inch)	
	Bran tford	Woodst ock	London A	Glen Allan	Fullart on	Delhi C.D.A.	Guelph W-1	Total	Ave-rage
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
July 12									
0100	0.00	0.00	0.07	0.00	0.01	0.00	0.00	0.08	0.011
0200	0.01	0.00	0.14	0.01	0.04	0.00	0.00	0.12	0.017
0300	0.01	0.05	0.15	0.02	0.09	0.10	0.05	0.27	0.038
0400	0.12	0.06	0.18	0.03	0.09	0.17	0.10	0.28	0.040
0500	0.20	0.09	0.25	0.05	0.10	0.17	0.10	0.21	0.030
0600	0.20	0.12	0.26	0.08	0.12	0.17	0.14	0.13	0.018
0700	0.21	0.13	0.32	0.08	0.14	0.22	0.18	0.19	0.027
0800	0.25	0.18	0.48	0.08	0.16	0.25	0.22	0.32	0.045
0900	0.28	0.24	0.54	0.08	0.16	0.33	0.31	0.34	0.048
1000	0.33	0.33	0.56	0.08	0.16	0.35	0.37	0.24	0.034
1100	0.33	0.38	0.56	0.08	0.16	0.35	0.37	0.05	0.071
1200	0.34	0.42	0.58	0.08	0.16	0.36	0.45	0.16	0.0228
1300	0.38	0.46	0.59	0.08	0.16	0.37	0.48	0.13	0.018
1400	0.43	0.48	0.62	0.08	0.16	0.41	0.50	0.16	0.0228
1500	0.49	0.51	0.64	0.08	0.16	0.44	0.51	0.15	0.021
1600	0.51	0.52	0.71	0.08	0.16	0.47	0.55	0.17	0.024
1700	0.56	0.55	0.75	0.09	0.16	0.53	0.59	0.23	0.032
1800	0.64	0.63	0.79	0.10	0.16	0.59	0.69	0.37	0.0528
1900	0.78	0.67	1.02	0.11	0.16	0.66	0.74	0.44	0.062
2000	0.94	0.70	1.07	0.20	0.16	0.77	0.89	0.59	0.084
2100	1.04	0.85	1.15	0.26	0.23	0.92	1.00	0.72	0.1028
2200	1.12	0.97	1.29	0.34	0.24	1.02	1.12	0.63	0.09
2300	1.23	1.08	1.48	0.51	0.29	1.17	1.20	0.80	0.14
2400	1.41	1.11	1.55	0.66	0.55		1.29	0.98	0.14

a) Mass Curve Computations of Large Storm in Western Ontario, Canada in 1964 [80]

Observation time (hour ending)	Cumulative depths (inches) for each site location (SiteID)							7-Sites MPE	
	15879	17580	17581	17583	17584	19291	20993	Total	Average
6/29/11 17:00	0.01	0.01	0.06	0.02	0.01	0.01	0.01	0.13	0.02
6/29/11 18:00	0.01	0.02	0.07	0.03	0.04	0.02	0.01	0.07	0.01
6/29/11 19:00	0.01	0.02	0.07	0.03	0.04	0.06	0.02	0.05	0.01
6/29/11 20:00	0.38	0.5	0.62	0.29	0.12	0.07	0.02	1.75	0.25
6/29/11 21:00	0.94	0.95	0.86	0.29	0.12	0.07	0.02	1.25	0.18
6/29/11 22:00	0.94	0.95	0.86	0.29	0.12	0.07	0.02	0.00	0.00
6/29/11 23:00	0.94	0.95	0.86	0.29	0.12	0.07	0.02	0.00	0.00
6/30/11 00:00	0.94	0.95	0.86	0.29	0.12	0.07	0.02	0.00	0.00
6/30/11 01:00	0.94	0.95	0.86	0.29	0.12	0.17	0.14	0.22	0.03
6/30/11 02:00	0.94	0.95	0.86	0.4	0.3	0.56	0.44	0.98	0.14
6/30/11 03:00	0.99	0.96	0.9	0.46	0.35	0.56	0.44	0.21	0.03
6/30/11 04:00	0.99	0.96	0.9	0.46	0.35	0.56	0.44	0.00	0.00

b) Mass-Curve Computations of Local Large Storm from Our Storm Data

Figure 5-6 Some Examples of Mass-Curve Computations

5.2.1 Storm Analysis

More general storm analysis can be done in both location-specific (through LocalStorms and LocalStormHours tables) and storm-specific (i.e., the remaining tables). All types of storms (local, hourly, and overall storms) can be analyzed either directly by utilizing calculated storm characteristics and SQL features (e.g., built-in statistical functions, complex query, stored procedure, UDF, etc.) or indirectly by extracting the conceptual storm data from a relational database to be used by other methods.

For simple analysis, a generic SQL pattern is shown in SQL35. Examples of analysis that use this SQL pattern include calculating maximum total rainfall for each storm duration for a particular site (location-based) (see query SQL36) and determining the distribution of storm durations for the entire 16 years of overall storm data (see query SQL37).

SQL35. Simple storm analysis pattern

```
SELECT     (<grouping attribute>,)* (<statistical function>(<storm characteristic>))+
FROM       (LocalStorms|HourlyStorms|OverallStorms)
[WHERE     <filtering criteria>]
[GROUP BY  <grouping criteria>]
```

SQL36. Calculating maximum total rainfall for each storm duration for a single site

```
SELECT     Duration,MAX(TotalRainfall)
FROM       LocalStorms
WHERE      SiteID = <site>
GROUP BY   Duration
```

SQL37. Calculate distribution of storm durations for the entire 16 years of overall storm data

```
SELECT     Duration,COUNT(*)
FROM       LocalStorms
GROUP BY   Duration
```

More complex storm analysis can also be done such as calculating an averaged intensity for each site covered by the longest heavy rainstorm [101] (location-specific) and counting number of overall storms passing North Central region of Texas (storm-specific). These two examples are described in queries SQL 38 and 39, respectively.

SQL38. Calculating an averaged intensity for each site covered by the longest heavy rainstorm

```
//Assuming that all sites covered by the rainstorm have the same start time and end time
SELECT     L1.SiteID,AVG(L1.DataValue)
FROM       LocalStormHours L1
WHERE      EXISTS   (SELECT    *
                     FROM      LocalStorms L2
                     WHERE     L1.YearID=L2.YearID AND L1.LSID=L2.LSID AND L2.Intensity≥0.3 AND
                               L2.Duration = (SELECT     MAX(Duration)
                                              FROM       LocalStorms))
GROUP BY   L1.SiteID
```

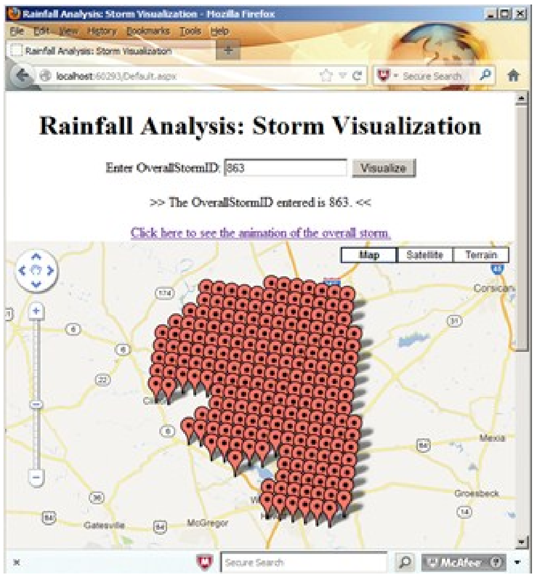
SQL39. Counting number of overall storms passing North Central region of Texas last year

```
SELECT     COUNT(DISTINCT O.OSID)
FROM       OverallStormHourlyStorms O
WHERE      O.YearID=2013 AND
           EXISTS   (SELECT    *
                     FROM      HourlyStormSites H
                     WHERE     O.Time=H.Time AND O.HSID=H.HSID AND
                               H.SiteID IN <list of sites in North Central region>)
```

We can also extract the conceptual storm data from a relational database and analyze it by using other methods. As an example, we extracted the relational storm data and analyze it through visualization [62][100]. We developed a prototype for visualization, called StormVisualization [62]. The StormVisualization tool is implemented in C#, Javascript, HTML5, Google API [48] and ASP.NET. The tool illustrates how an overall storm are formed and moved over time. Algorithm 8 highlights how storm visualization works. Figure 5-7 shows the very first screenshot of the Storm Visualization, which projects the overall storm (ID:863) in year 2011 onto the map. After triggering by a user, the animation of overall storm (863) is shown in Figure 5-8. The projection of each hourly storm of the overall storm (863) is shown hour by hour starting at 4/26/2011, 20:00 (Figure 5-8 (a)). The number in parentheses indicates the number of hourly storms involved in that hour.

Algorithm8. StormVisualization

```
    - Project an overall storm on the map
1:  //Extract all lat/long coordinates of the specified overall storm from the relational storm data
2:  y = SELECT (290+((SiteID-15599)MOD 1701)) AS HRAPX, (10+((SiteID-15599)DIV 1701)) AS HRAPY
3:      FROM   ( SELECT    DISTINCT(S.SiteID)
4:               FROM      HourlyStormSites S
5:               WHERE     EXISTS    (SELECT  *
6:                                    FROM    OverallStormHourlyStorms O
7:                                    WHERE   O.DateTimeUTC=S.DateTimeUTC AND O.HSID=S.HSID AND
8:                                            O.YearID=<year> AND O.OSID = <osid>)
9:  Project_on_the_map(ToLatLong(y))
10: //Project overall storm on the map by hour
11: d = SELECT     O.Start, O.Stop
12:     FROM       OverallStorms O
13:     WHERE      O.YearID = <year> AND O.OSID = <osid>
14: for i = d.Start to d.Stop do
15:     p = SELECT (290+((SiteID-15599)MOD 1701)) AS HRAPX, (10+((SiteID-15599)DIV 1701)) AS HRAPY
16:         FROM   //Extract all sites of an overall storm for a particular hour
17:                ( SELECT  DISTINCT(S.SiteID)
18:                 FROM     HourlyStormSites S
19:                 WHERE    S.HSID IN
20:                          ( SELECT  DISTINCT(M.HSID)
21:                           FROM     OverallStormHourlyStorms M
22:                           WHERE    M.DateTimeUTC=S.DateTimeUTC AND M.HSID=S.HSID AND
23:                                    M.YearID=<year> AND M.OSID = <osid> )
24:                          AND S.DateTimeUTC = i)
25:     Project_on_the_map(ToLatLong(p))
26: end for
```


Rainfall Analysis: Storm Visualization

Enter OverallStormID: 863

>> The OverallStormID entered is 863. <<

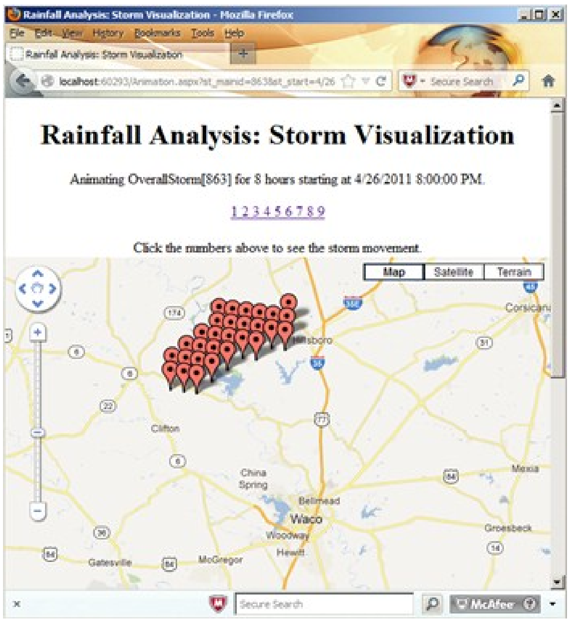
Click here to see the animation of the overall storm.

Rainfall Analysis: Storm Visualization - Mozilla Firefox
localhost:60293/Default.aspx
Visualize
Map
Satellite
Terrain
Corsicana
Mexia
Groesbeck
Gatesville
McGregor

Figure 5-7 Screenshot of Overall Storm ID 863

5.2.2 Storm Mining

More efficient storm mining can also be done on our storm data in both location-based and storm-based approaches. In the location-based approach, temporal data mining techniques can be applied as local storm data can be considered as time series data for a given location (single site or region). There are seven major data mining tasks for temporal data [104]: 1) indexing, 2) clustering, 3) classification, 4) prediction, 5) summarization, 6) anomaly detection, and 7) segmentation. On the other hand, for the storm-specific approach, since our overall storm outputs are spatio-temporal data, we can apply spatio-temporal data mining to it. Three main areas of spatio-temporal data mining include [82]: 1) spatio-temporal forecasting and prediction, 2) spatio-temporal clustering, and 3) spatio-temporal visualization. Similar to storm analysis, the storm mining tasks can be done directly on the relational database by using database mining techniques [72], or indirectly by extracting the overall storm data to be used by other methods. We are currently working on some of these spatio-temporal data mining approaches, but the results are part of our future work.


Rainfall Analysis: Storm Visualization - Mozilla Firefox
File Edit View History Bookmarks Tools Help
localhost:60293/Animation.aspx?st_mainid=863&st_start=4/26
Map
Satellite
Terrain
Clifton
China Spring
Bellmead
Waco
Woodway
Hewitt
Gatesville
McGregor
Mexia
Groesbeck
Corsicana
Hillsboro

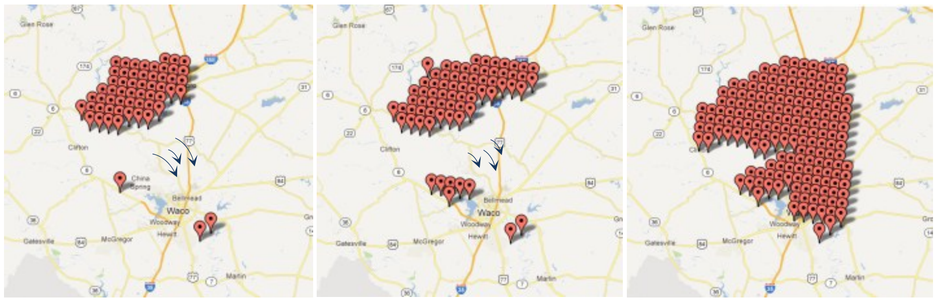
Rainfall Analysis: Storm Visualization

Animating OverallStorm[863] for 8 hours starting at 4/26/2011 8:00:00 PM.

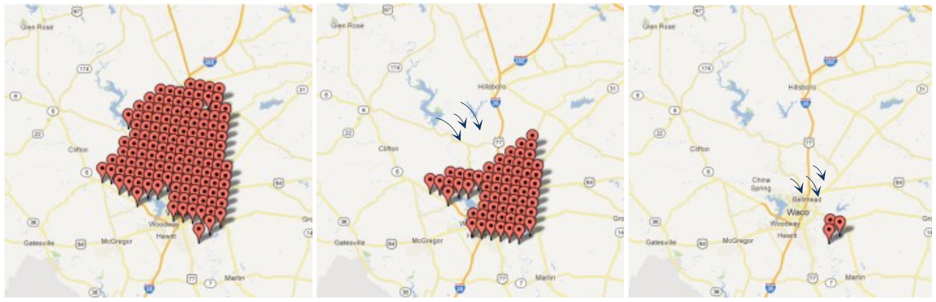
1 2 3 4 5 6 7 8 9

Click the numbers above to see the storm movement.

a) 4/26/2011, 20:00 (1)



b) 4/26/2011, 22:00 (3) c) 4/26/2011, 23:00 (3) d) 4/27/2011, 01:00 (1)



e) 4/27/2011, 02:00 (1) f) 4/27/2011, 03:00 (1) g) 4/27/2011, 04:00 (1)

Figure 5-8 Visualization of Overall Storm (ID: 863) by its Hourly Storms

Chapter 6

Framework for Conceptual Analysis and Mining of Big Data Using Ontologies and EER

In the previous chapters (Chapter 3, 4, and 5), the techniques and methodology that we use are for big raw rainfall data. The goal is to allow big raw rainfall data to be easily analyzed and mined through a relational database. These techniques and methodology can also be adapted to different types of hydrological data such as soil moisture, water level, etc., as well as other types of big data in other application domains. Therefore, in this chapter, we propose a more generalized framework for analyzing and mining big data in any given domain. The framework allows big data in a particular domain to be conceptually analyzed and mined by using ontologies and EER. We discuss the background and motivation in Section 6.1. The framework description is discussed in Section 6.2

6.1 Background and Motivation

Enormous amounts of data are rapidly generated every day in almost every application domain. In any given domain, big data contains potential hidden meaningful concepts as well as superfluous data that are not of interest to the domain experts. As a result, dealing with big data solely by applying a set of distributed computing technologies such as MapReduce [66], BSP (Bulk Synchronous Parallel) [70], and Spark [71], and/or distributed storage systems namely NoSQL databases [73] may not be an efficient way to discover the knowledge hidden in the data. To enable analysis, the big data need to be pre-processed so that the superfluous parts are removed (also known as a “cleaning” of the raw data) and the meaningful domain-specific knowledge is extracted.

Ontology, a specification of conceptualization [105], has practically been used in knowledge modeling as it allows domain-specific knowledge to be formalized and reasoned about in a logical way. ER and EER models and diagrams are excellent tools to communicate concepts, and can also be easily converted to relational tables. We use ontologies and EER to represent the conceptual knowledge in the data. The formalized concepts are developed based on consulting with domain experts in the area of knowledge covered by the raw data

The advantages of our framework include the capture of domain-specific conceptual knowledge (which is significantly smaller in size, compared to the raw data but substantially more meaningful to the domain experts), better system performance by applying distributed computing technologies to clean and convert the raw data, and more robust and user-friendly analysis by storing the extracted conceptual knowledge in a relational database.

6.2 Framework Description

An overview of our framework is illustrated in Figure 6-1. There are four main processes:

1. developing and formalizing domain-specific concepts into an ontology with the assistance of domain experts
2. translating the domain-specific ontology to EER and mapping the EER concepts to relational tables
3. designing and implementing mapping algorithms in a distributed framework to convert the big raw data to the conceptual data
4. performing analysis and mining on the conceptual relational data

We discuss each of these in the following subsections.

6.2.1 Developing and Formalizing Domain-Specific Concepts into an Ontology with the Assistance of Domain Experts

The first step is to study a particular domain, where the big raw data comes from, come up with the domain-specific concepts, and formalize them into domain-specific ontology concepts. This step requires literature review in the application domain and working with domain experts to determine the concepts that are important to them, and how their research is currently conducted using traditional data processing methods/technologies. The pros and cons of the methods/technologies they use are determined, and potential improvements that utilize the available big data analysis tools such as MapReduce [66] and Hadoop® [67], are identified.

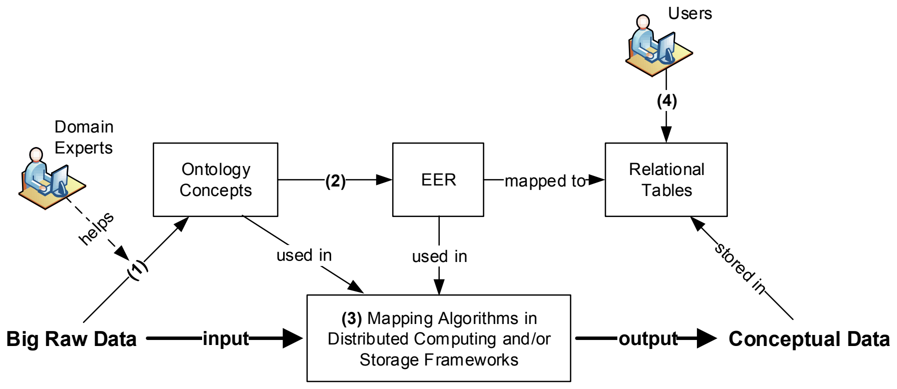


Figure 6-1 Framework for Analyzing and Mining Big Data using Ontologies and EER

The developed ontology must meet the domain experts' requirements, and capture essential concepts that they are looking for as well as other potential concepts, which may not have been previously identified because of limitations of the traditional analysis methods, but could be of benefit to them. These limitations may be due to structure of the big raw data and the lack of knowledge of domain users in distributed technologies. The developed ontology can also help domain users to understand their data better. The hidden insight and conceptual relationships can lead them to the knowledge that was not previously identified. Consequently, further complex analysis and mining can also be done. In our rainfall analysis, this process is equivalent to Chapter 3, where we formalized rainstorm concepts to capture “overall” characteristics of the storms.

6.2.2 Translating the Domain-Specific Ontology to EER and Mapping the EER Concepts to Relational Tables

In the context of big data [69], relational database and RDBMS are usually not a preferred option and are often labeled as incompatible with the needs of big data analysis and mining. On the other hand, the concepts of NoSQL databases spread rapidly and caught a lot of attention as tools for big data storage and analysis/mining in the past few years. (As of now, it was estimated that there are at least 150 different NoSQL database vendors [73].) The advantages of NoSQL databases include high availability, fast key-value access, horizontal scalability, fault-tolerance, and dynamic/semi-structured data type support. However, the disadvantages of NoSQL databases includes weak consistency, not fully supporting relational features (e.g., join, group by, order by operations) across partitions, denormalized data model, and lack of a powerful declarative query language (i.e., SQL) [73][74][75][78] that can be used for data analysis. Our framework loads the final conceptual output data into a relational database so that all RDBMS benefits can be used. To store the final conceptual outputs in a relational database, we translate the formalized domain concepts from the previous step to an EER model, which will later be mapped to relational tables. The mapping process is done by using the methodology/steps described in [41]. In the case of our rainfall data, this process is equivalent to what we did in Chapter 4, Section 4.3, where we designed an EER model based on the storm formalizations and characteristics of the raw rainfall data, and mapped it to a relational database schema for storing the conceptual storm data.

6.2.3 Designing and Implementing Mapping Algorithms to Convert the Raw Data to the Conceptual Data

To design mapping algorithms, four factors are taken into account: structure and format of the big raw data, choices of distributed computing/storage framework, domain-specific ontology, and EER model corresponding to the ontology. Understanding the structure and format of the big raw data helps in optimizing the computation, I/O, and buffer usage in the raw data-to-conceptual data mapping (or conversion) algorithms. Three aspects of big raw data are considered: data representation/interpretation, transmission of data, and data integrity. In data representation and interpretation, we examine how the raw data is represented and interpreted, how many columns/attributes there are in the raw files, what format they are based on, the particular order of the data items, and if any prior knowledge is required to interpret them. In the data transmission, we determine how data is transmitted and delivered to the storage system, how often data is reported, and if there is any regular downtime period, where the data might not arrive on time or get lost. In the integrity aspect, we check if the raw data can be updated after delivery and if so, how it is done and the possible effects on the previous values. We also determine when the data will be finalized after the initial observation time, and how data integrity is maintained.

Next, we make the decision as to which distributed technology should be used. The selected technology should take full advantage of the characteristics of the raw data and other available resources (e.g., hardware). The ontology is used to ensure the formalized domain concepts are correctly identified. Finally, the corresponding EER model is used to convert the final conceptual outputs into relational database-compatible format. Since the final conceptual data is now stored in a relational database, the verification process can also be done through SQL.

The developed mapping algorithms should be flexible enough in case minor changes are made to the raw data formats and/or other requirements (e.g., changing of data interval from hourly to 15-minute interval), so that significant changes to the algorithms will not be required. (In practice, in such cases, typically, the source provider will send a notification in advance and usually include converter tools. So, it is recommended to subscribe to the source provider newsletters so that when that happens, we will be notified.) However, if the changes are significant, which require dramatic updates, we would have a tradeoff between: 1) re-design the programs based on the new format and/or the data requirements, or 2) develop a converter tool of our own to convert the new format to the old format so that we can continue using the existing programs. In our rainfall analysis, this step is equivalent to what we described in Chapter 4, Section 4.2, where we improved the system performance by applying distributed computing technologies for converting the raw data to conceptual data.

6.2.4 Performing Analysis and Mining on the Conceptual Relational Data

After the algorithms are executed on the big raw data, we now have the conceptual data that we are interested in stored in a relational database. The size of the conceptual relational outputs are usually significantly reduced when compared to the size of the big raw data as the superfluous parts are removed and the raw data is summarized/converted into meaningful domain-specific concepts. The analysis and mining tasks can then be easily conducted by domain users on the conceptual knowledge base either directly using SQL or indirectly by extracting the conceptual data from the relational database. In our rainfall analysis, this process is equivalent to Chapter 5, where we perform analysis and mining on the conceptual storm data through a relational database.

Chapter 7

Conclusions and Future Work

In this dissertation, we work with geospatio-temporal datasets. First, we consider spatio-temporal datasets in general. We focus on the semantics of spatio-temporal data and define spatio-temporal formalization using a technique called “light-weight” for integrating multiple ontologies in the Protégé (OWL) framework. We then implement the defined spatio-temporal formalization to the actual system (e.g., Protégé) to create a spatio-temporal ontology framework that allows spatio-temporal data to be analyzed, reasoned, inferenced, and queried.

Next, we focus on a spatio-temporal dataset in a particular domain. In our case, it is a rainfall precipitation data in the hydrology domain. We first examine the structure and format of the raw rainfall data, study rainfall-related concepts, and consult with experts in the domain. We then create a rainstorm ontology formalization consisting of local storm, hourly storm, and overall storm. The formalized rainstorm concepts enable more efficient analysis and mining on the rainfall data such that not only can traditional hydrology analysis be done on the rainfall data, but storm-specific analysis that can capture “overall” characteristics of storms can also be done.

To identify the formalized rainstorm concepts from the raw rainfall data, we develop conversion/mapping algorithms called storm identification system based on two approaches: CUAHSI-based approach and MapReduce-based approach. The CUAHSI-based approach is based on CUAHSI standard, which stores and processes input and output from a relational database based on CUAHSI ODM database schema, and uses recursive depth-first search to identify storms. The MapReduce-based approach is an improved version of the CUAHSI-based approach with higher performance. All components in the storm identification systems are re-designed for full utilization of resources. That is, we process the raw data directly from the original text file format instead of relational data to eliminate the overhead problem related to relational database. Additionally, we utilize our cluster of 19 machines by applying distributed computing technology map-reduce to parallelize the storm identification process. The final storm outputs are eventually loaded to a relational database for easy analysis and mining. The verification process is also conducted on the relational storm output data using SQL to ensure that the storm data are correctly identified.

Then, we show how conceptual storm data can be used in analysis and mining. We first describe how traditional hydrology analysis (location-specific) can be done on the rainstorm data. Three categories of rainfall-related analysis are discussed: storm statistical properties, correlation among characteristics of storms, and analysis of extreme rainfall events. We also discuss more general storm analysis and mining using the conceptual storm data. The more general analysis/mining can be done in both location-specific and storm-specific modes either directly from relational storm data or indirectly by extracting the storm data from a relational database. We show some examples of these analyses, which also include an implemented visualization tool called StormVisualization to illustrate the formation and movement of a given overall storm.

Finally, we generalize our techniques and methodology used for the rainfall data into a framework for analyzing and mining big data in any given domain. The framework allows big data in a particular domain to be conceptually analyzed and mined by utilizing ontologies and EER. The framework consists of four main processes: 1) developing and formalizing domain-specific concepts into an ontology with the assistance of domain experts, 2) translating the domain-specific ontology to EER and mapping the EER concepts to relational tables, 3) designing and implementing mapping algorithms in a distributed framework to convert the big raw data to the conceptual data, 4) performing analysis and mining on the conceptual relational data.

The main contributions of our research are:

1. Developing a spatial ontology and integrating it with a temporal ontology.
2. Proposing formalized storm concepts that enable easier analysis and mining of rainfall data.
3. Developing efficient algorithms to convert raw rainfall data into meaningful storm concepts.
4. Applying analysis and mining techniques to conceptual storm data.
5. Developing a framework that can be applied to other types of big spatio-temporal raw data to reduce the data and convert it into concepts for better analysis and mining.

For future work, we can implement the storm identification system in different distributed computing frameworks, such as Sparks and BSP, and compare the performance among them. For better storm analysis and mining, we can process finer rainfall data (such as 5-minute interval rainfall data) and longer history. We can also develop other formalizations of other hydrological data related to rainfall such as wind speed, soil moisture, and temperature and integrate them with the rainstorm ontology for better storm prediction. Our partial rainstorm ontology can also be extended to complete rainfall ontology for broader and complete rainfall analysis. Finally, we can develop a methodology and tools to partially automate the domain consultation process so that more domain-specific ontologies can be developed quickly.

Appendix A

Proving Completeness of OpenGIS® SQL Spatial Relationships and Operations

In this appendix, we prove the completeness of OpenGIS SQL Spatial Relationships and Operations. Our proof first displays the complete set of 68 relationships [33] (each of which is numbered) in a graphical form as shown in Figure A-1. We then show how each relationship can be specified in OpenGIS SQL. For convenience, we define some shorthand operations that are commonly used in combinations of OpenGIS SQL. We use the shorthand operations to represent the subsequent OpenGIS SQL operations. Our notation is as follows: *P* stands for a point, *L* stands for a line, *A* stands for a polygon (area), and *G* stands for any geometry object (*P*, *L*, or *A*).

A.1 Shorthand Notations

A.1.1 Shorthand Notations for Function Operations

A.1.1.1 $Interior(L)$

Interior(L:Line) function returns a line without its ending points.

$-Interior(L) \equiv Difference(Difference(L, StartPoint(L)), EndPoint(L))$

A.1.1.2 $Interior(A)$

Interior(A:Polygon) function returns a polygon without its exterior ring.

$-Interior(A) \equiv Difference(A, ExteriorRing(A))$

A.1.2 Shorthand Notations for Boolean Operations

A.1.2.1 $IntersectsEitherEndPoint(L, G)$

IntersectsEitherEndPoint(L,G) operation returns TRUE if either endpoint of *L* intersects with *G*.

$-IntersectsEitherEndPoint(L, G) \equiv$

$Intersects(StartPoint(L), G) \vee Intersects(EndPoint(L), G)$

A.1.2.2 $IntersectsBothEndPoints(L, G)$

IntersectsBothEndPoints(L,G) operation returns TRUE if both endpoints of *L* intersect with *G*.

$-IntersectsBothEndPoints(L, G) \equiv$

$Intersects(StartPoint(L), G) \wedge Intersects(EndPoint(L), G)$

A.1.2.3 $Intersects1EndPoint(L, G)$

Intersects1EndPoints(L,G) operation returns TRUE if only one of the endpoints of *L* intersects with *G*.

$-Intersects1EndPoint(L, G) \equiv$

$[Intersects(StartPoint(L), G) \wedge Disjoint(EndPoint(L), G)] \vee$

$[Intersects(EndPoint(L), G) \wedge Disjoint(StartPoint(L), G)]$

A.1.2.4 $DisjointBothEndPoints(L, G)$

DisjointBothEndPoints(L,G) operation returns TRUE if both endpoints of L are disjoint with G.

$-DisjointBothEndPoints(L, G) \equiv Disjoint(StartPoint(L), G) \wedge Disjoint(EndPoint(L), G)$

A.1.2.5 $InsideMax(G_1, G_2)$

InsideMax(G_1,G_2) operation returns TRUE if G_1 is inside G_2 such that G_1 does not intersect the boundary of G_2.

$-InsideMax(G_1, G_2) \equiv Within(G_1, G_2) \wedge Disjoint(G_1, Boundary(G_2))$

A.1.2.6 $ContainMax(G_1, G_2)$

ContainMax(G_1,G_2) operation return TRUE if G_1 contains G_2 such that G_2 does not intersect the boundary of G_1.

$-ContainMax(G_1, G_2) \equiv Contains(G_1, G_2) \wedge Disjoint(G_2, Boundary(G_1))$

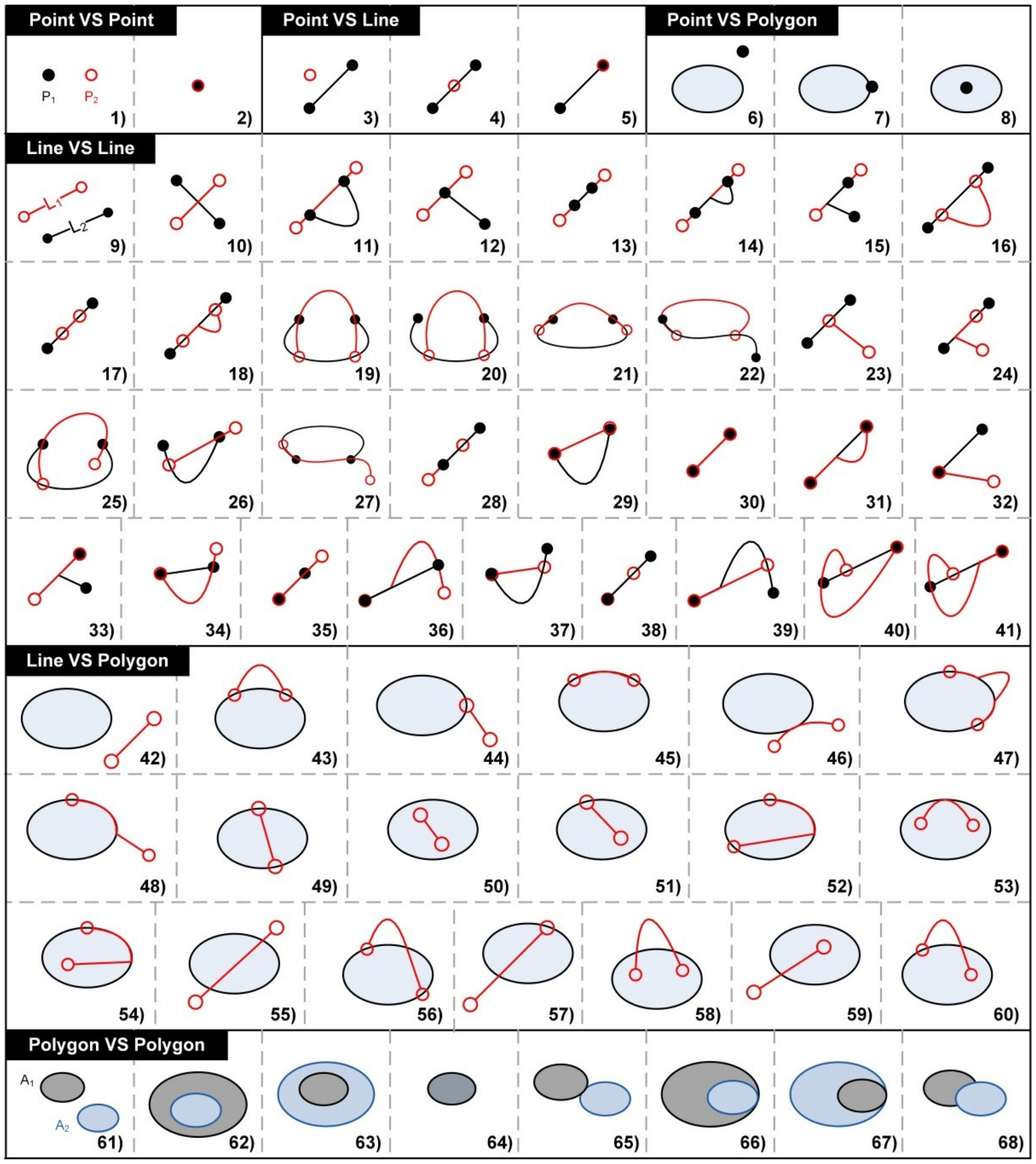


Figure A-1 List of Possible Spatial Relationships between 2-D Spatial Objects (Point, Line, and Polygon) [33]

A.1.2.7 $CoveredByMax(G_1, G_2)$

CoveredByMax(G$_1$,G$_2$) operation return TRUE if G_1 is inside G_2 such that G_1 intersects the boundary of G_2.

$-CoveredByMax(G_1, G_2) \equiv Within(G_1, G_2) \wedge Intersects(G_1, Boundary(G_2))$

A.1.2.8 $CoverMax(G_1, G_2)$

CoverMax(G$_1$,G$_2$) operation return TRUE if G_1 contains G_2 such that G_2 intersects the boundary of G_1.

$-CoverMax(G_1, G_2) \equiv Contains(G_1, G_2) \wedge Intersects(G_2, Boundary(G_1))$

A.2 Proof of Completeness

We will use the defined shorthand notations in our proof. The proof of completeness can be divided into three subsections: 1) P/P, P/L, and P/A, 2) L/L and L/A, and 3) A/A.

A.2.1 Proof of Completeness: P/P, P/L, and P/A

A.2.1.1 P/P

1) $Disjoint(P_1, P_2)$

2) $Equals(P_1, P_2)$

A.2.1.2 P/L

3) $Disjoint(P, L)$

4) $Within(P, L)$

5) $Touches(P, L)$

A.2.1.3 P/A

6) $Disjoint(P, A)$

7) $Touches(P, A)$

8) $InsideMax(P, A)$

A.2.2 Proof of Completeness: L/L and L/A

A.2.2.1 L/L

9) $Disjoint(L_1, L_2)$

10) $\big(Crosses(L_1, L_2) \vee Overlaps(L_1, L_2)\big) \wedge DisjointBothEndPoints(L_1, L_2) \wedge DisjointBothEndPoints(L_2, L_1))$

11) $IntersectsBothEndPoints(L_2, Interior(L_1)) \wedge Disjoint(L_1, Interior(L_2))$

12) $Touches(L_1, L_2) \wedge DisjointBothEndPoints(L_1, L_2) \wedge DisjointBothEndPoints\big(L_2, Boundary(L_1)\big) \wedge Intersects1EndPoint\big(L_2, Interior(L_1)\big)$

13) $InsideMax(L_2, L_1)$

14) $(Crosses(L_1, L_2) \vee Overlaps(L_1, L_2)) \wedge DisjointBothEndPoints(L_1, L_2) \wedge IntersectsBothEndPoints(L_2, Interior(L_1))$

15) $(Crosses(L_1, L_2) \vee Overlaps(L_1, L_2)) \wedge DisjointBothEndPoints(L_1, L_2) \wedge Intersects1EndPoint(L_2, Interior(L_1))$

16) $IntersectsBothEndPoints\big(L_1, Interior(L_2)\big) \wedge Disjoint(L_2, Interior(L_1))$

17) $InsideMax(L_1, L_2)$

18) $(Crosses(L_1, L_2) \vee Overlaps(L_1, L_2)) \wedge DisjointBothEndPoints(L_2, L_1) \wedge IntersectsBothEndPoints(L_1, Interior(L_2))$

19) $Touches(L_1, L_2) \wedge IntersectsBothEndPoints\big(L_1, Interior(L_2)\big) \wedge IntersectsBothEndPoints(L_2, Interior(L_1))$

20) $Touches(L_1, L_2) \wedge Intersects1EndingPoint(L_2, Interior(L_1)) \wedge IntersectsBothEndPoints\big(L_1, Interior(L_2)\big)$

21) $(Crosses(L_1, L_2) \vee Overlaps(L_1, L_2)) \wedge IntersectsBothEndPoints\big(L_2, Interior(L_1)\big) \wedge IntersectsBothEndPoints\big(L_1, Interior(L_2)\big)$

22) $\big(Crosses(L_1, L_2) \vee Overlaps(L_1, L_2)\big) \wedge IntersectsBothEndPoints\big(L_1, Interior(L_2)\big) \wedge$

$Intersects1EndPoint(L_2, Interior(L_1))$

23) $Touches(L_1, L_2) \wedge DisjointBothEndPoints(L_2, L_1) \wedge$

$DisjointBothEndPoints\big(L_1, Boundary(L_2)\big) \wedge Intersects1EndPoint\big(L_1, Interior(L_2)\big)$

24) $\big(Crosses(L_1, L_2) \vee Overlaps(L_1, L_2)\big) \wedge DisjointBothEndPoints(L_2, L_1) \wedge$

$Intersects1EndPoint\big(L_1, Interior(L_2)\big)$

25) $Touches(L_1, L_2) \wedge Intersects1EndPoint\big(L_1, Interior(L_2)\big) \wedge$

$IntersectsBothEndPoints\big(L_2, Interior(L_1)\big)$

26) $Touches(L_1, L_2) \wedge Intersects1EndPoint\big(L_1, Interior(L_2)\big) \wedge$

$Intersects1EndPoint\big(L_2, Interior(L_2)\big) \wedge DisjointBothEndPoints(L_1, Boundary(L_2))$

27) $(Crosses(L_1, L_2) \vee Overlaps(L_1, L_2)) \wedge IntersectsBothEndPoints\big(L_2, Interior(L_1)\big) \wedge$

$Intersects1EndPoint\big(L_1, Interior(L_2)\big)$

28) $(Crosses(L_1, L_2) \vee Overlaps(L_1, L_2)) \wedge Intersects1EndPoint\big(L_1, Interior(L_2)\big) \wedge$

$Intersects1EndPoint\big(L_2, Interior(L_1)\big) \wedge DisjointBothEndPoints(L_1, Boundary(L_2))$

29) $Touches(L_1, L_2) \wedge IntersectsEitherEndPoint\big(L_2, StartPoint(L_1)\big) \wedge$

$IntersectsEitherEndPoint\big(L_2, EndPoint(L_1)\big)$

30) $Equals(L_1, L_2)$

31) $(Crosses(L_1, L_2) \vee Overlaps(L_1, L_2)) \wedge IntersectsEitherEndPoint\big(L_2, StartPoint(L_1)\big) \wedge$

$IntersectsEitherEndPoint\big(L_2, EndPoint(L_1)\big)$

32) $Touches(L_1, L_2) \wedge DisjointBothEndPoints\big(L_1, Interior(L_2)\big) \wedge$

$DisjointBothEndPoints\big(L_2, Interior(L_1)\big) \wedge ((IntersectsEitherEndPoint(L_2, StartPoint(L_1)) \wedge$

$DisjointBothEndPoints(L_2, EndPoint(L_1))) \vee (IntersectsEitherEndPoint(L_2, EndPoint(L_1)) \wedge$

$DisjointBothEndPoints(L_2, StartPoint(L_1))))$

33) $\left(Crosses(L_1,L_2) \vee Overlaps(L_1,L_2)\right) \wedge DisjointBothEndPoints\left(L_1,Interior(L_2)\right) \wedge$

$DisjointBothEndPoints\left(L_2,Interior(L_1)\right) \wedge ((IntersectsEitherEndPoint(L_2,StartPoint(L_1)) \wedge$

$DisjointBothEndPoints(L_2,EndPoint(L_1))) \vee (IntersectsEitherEndPoint(L_2,EndPoint(L_1)) \wedge$

$DisjointBothEndPoints(L_2,StartPoint(L_1))))$

34) $Touches(L_1,L_2) \wedge DisjointBothEndPoints\left(L_1,Interior(L_2)\right) \wedge$

$Interesects1EndPoints\left(L_2,Interior(L_1)\right) \wedge ((IntersectsEitherEndPoint(L_2,StartPoint(L_1)) \wedge$

$DisjointBothEndPoints(L_2,EndPoint(L_1))) \vee (IntersectsEitherEndPoint(L_2,EndPoint(L_1)) \wedge$

$DisjointBothEndPoints(L_2,StartPoint(L_1))))$

35) $CoverMax(L_1,L_2)$

36) $\left(Crosses(L_1,L_2) \vee Overlaps(L_1,L_2)\right) \wedge Intersects1EndPoint\left(L_2,Interior(L_1)\right) \wedge$

$DisjointBothEndPoints\left(L_1,Interior(L_2)\right) \wedge ((IntersectsEitherEndPoint(L_2,StartPoint(L_1)) \wedge$

$DisjointBothEndPoints(L_2,EndPoint(L_1))) \vee (IntersectsEitherEndPoint(L_2,EndPoint(L_1)) \wedge$

$DisjointBothEndPoints(L_2,StartPoint(L_1))))$

37) $Touches(L_1,L_2) \wedge DisjointBothEndPoints\left(L_2,Interior(L_1)\right) \wedge$

$Interesects1EndPoint\left(L_1,Interior(L_2)\right) \wedge ((IntersectsEitherEndPoint(L_2,StartPoint(L_1)) \wedge$

$DisjointBothEndPoints(L_2,EndPoint(L_1))) \vee (IntersectsEitherEndPoint(L_2,EndPoint(L_1)) \wedge$

$DisjointBothEndPoints(L_2,StartPoint(L_1))))$

38) $CoveredByMax(L_1,L_2)$

39) $\left(Crosses(L_1,L_2) \vee Overlaps(L_1,L_2)\right) \wedge Intersects1EndPoint\left(L_1,Interior(L_2)\right) \wedge$

$DisjointBothEndPoints\left(L_2,Interior(L_1)\right) \wedge ((IntersectsEitherEndPoint(L_2,StartPoint(L_1)) \wedge$

$DisjointBothEndPoints(L_2,EndPoint(L_1))) \vee (IntersectsEitherEndPoint(L_2,EndPoint(L_1)) \wedge$

$DisjointBothEndPoints(L_2,StartPoint(L_1))))$

40) $Touches(L_1, L_2) \wedge Intersects1EndPoint(L_1, Interior(L_2)) \wedge$ $Intersects1EndPoint(L_2, Interior(L_1)) \wedge ((IntersectsEitherEndPoint(L_2, StartPoint(L_1)) \wedge$ $DisjointBothEndPoints(L_2, EndPoint(L_1))) \vee (IntersectsEitherEndPoint(L_2, EndPoint(L_1)) \wedge$ $DisjointBothEndPoints(L_2, StartPoint(L_1))))$

41) $(Crosses(L_1, L_2) \vee Overlaps(L_1, L_2)) \wedge Intersects1EndPoint(L_1, Interior(L_2)) \wedge$ $Intersects1EndPoint(L_2, Interior(L_1)) \wedge ((IntersectsEitherEndPoint(L_2, StartPoint(L_1)) \wedge$ $DisjointBothEndPoints(L_2, EndPoint(L_1))) \vee (IntersectsEitherEndPoint(L_2, EndPoint(L_1)) \wedge$ $DisjointBothEndPoints(L_2, StartPoint(L_1))))$

A.2.2.2 L/A

42) $Disjoint(L, A)$

43) $Touches(L, A) \wedge Touches(L, ExteriorRing(A)) \wedge IntersectsBothEndPoints(L, ExteriorRing(A))$

44) $Touches(L, A) \wedge Touches(L, ExteriorRing(A)) \wedge Intersects1EndPoint(L, ExteriorRing(A))$

45) $InsideMax(L, ExteriorRing(A))$

46) $Touches(L, A) \wedge DisjointBothEndPoints(L, A)$

47) $Touches(L, A) \wedge (Crosses(L, ExteriorRing(A)) \vee Overlaps(L, ExteriorRing(A))) \wedge$ $IntersectsBothEndPoints(L, ExteriorRing(A))$

48) $Touches(L, A) \wedge (Crosses(L, ExteriorRing(A)) \vee Overlaps(L, ExteriorRing(A))) \wedge$ $Intersects1EndPoints(L, ExteriorRing(A))$

49) $CoverMax(A, L) \wedge ContainMax(A, Interior(L)) \wedge IntersectsBothEndPoints(L, ExteriorRing(A))$

50) $ContainMax(A, L)$

51) $CoverMax(A, L) \wedge ContainMax(A, Interior(L)) \wedge Intersects1EndPoint(L, ExteriorRing(A))$

52) $CoverMax(A, L) \wedge (Crosses(L, ExteriorRing(A)) \vee Overlaps(L, ExteriorRing(A))) \wedge$ $IntersectsBothEndPoints(L, ExteriorRing(A))$

53) $CoverMax(A, L) \wedge IntersectsBothEndPoints(L, Interior(A))$

54) $CoverMax(A, L) \wedge (Crosses\big(L, ExteriorRing(A)\big) \vee Overlaps(L, ExteriorRing(A))) \wedge$

$Intersects1EndPoint\big(L, ExteriorRing(A)\big)$

55) $Crosses(L, A) \wedge DisjointBothEndPoints(L, A)$

56) $Crosses(L, A) \wedge IntersectsBothEndPoints\big(L, ExteriorRing(A)\big)$

57) $Crosses(L, A) \wedge Intersects1EndPoints\big(L, ExteriorRing(A)\big) \wedge$

$DisjointBothEndPoints(L, Interior(A))$

58) $Crosses(L, A) \wedge IntersectsBothEndPoints\big(L, Interior(A)\big)$

59) $Crosses(L, A) \wedge Intersects1EndPoints\big(L, Interior(A)\big) \wedge$

$DisjointBothEndPoints(L, ExteriorRing(A))$

60) $Crosses(L, A) \wedge Intersects1EndPoint\big(L, ExteriorRing(A)\big) \wedge$

$Intersects1EndPoint\big(L, Interior(A)\big)$

A.2.3 Proof of Completeness: A/A

A.2.3.1 A/A

61) $Disjoint(A_1, A_2)$

62) $InsideMax(A_2, A_1)$

63) $ContainMax(A_2, A_1)$

64) $Equals(A_1, A_2)$

65) $Touches(A_1, A_2)$

66) $CoverMax(A_1, A_2)$

67) $CoveredByMax(A_1, A_2)$

68) $Overlaps(A_1, A_2)$

References

[1] Abdelmoty, A.I., Smart P.D., El-Geresy, B.A. and Jones, C.B. 2009. Supporting Frameworks for the Geospatial Semantic Web. In *Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases*. SSTD'09. LNCS 5644, Springer, 355-372.

[2] Allen, J.F. 1983. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM 26*, 11 , 832-843.

[3] Allen, J.F. and Kautz, H.A. 1985. A Model of Naïve Temporal Resoning. *Formal Theories of the Commonsense World*. Ablex Pub, 251-268.

[4] Andronikos, T., Michalis, S. and Ioannis, P. 2009. Adding Temporal Dimension to Ontologies via OWL Reification. In *Proceedings of the 13th Panhellenic Conference on Informatics*. PCI'09. IEEE Computer Society Wachington, DC, 19-22.

[5] Baglioni, M., Masserotti, M.V., Renso, C. and Spinsanti, L. 2007. Building Geospatial Ontologies from Geographical Databases. In *Proceedings of the 2nd International Conference on GeoSpatial Semantics*. GeoS'07. LNCS 4853, Springer, 195-209.

[6] Baratis, E., Petrakis, E.G.M., Batsakis, S., Maris, N. and Papadakis, N. 2009. TOQL: Temporal Ontology Querying Language. In *Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases*. SSTD'09. LNCS 5644, Springer, 338-354.

[7] Bechhofer, S., Horrocks, I. and Patel-Schneider, P.F. 2003. Tutorial on OWL. Retrieved March 25, 2011, from: http://www.cs.man.ac.uk/~horrocks/ISWC2003/-Tutorial/.

[8] Bennacer, N., Aufaure, M., Cullot, N., Sotnykova, A. and Vangenot, C. 2004. Representing and Reasoning for Spatiotemporal Ontology Integration. *LNCS 3292*. Springer, 30-31.

[9] Bittner, T., Donnelly, M. and Smith, B. 2006. A Spatio-Temporal Ontology for Geographic Information Integration. *International Journal of Geographical Information Science 0(0)*, 1-29.

[10] Gruber, T. 2008. Ontology. *Encyclopedia of Database System*, Liu, L. and M.T. Özsu. Springer.

[11] Grutter, R. and Bauer-Messmer, B. 2007. Combining OWL with RCC for spatioterminological reasoning on environmental data. In *Proceedings of the 3rd International Workshop OWL: Experiences and Directions.* OWLED'07. CEUR-WS.org.

[12] Hess, G.N., Iochpe, C. and Castano, S. 2006. An Algorithm and Implementation for GeoOntologies Integration. In *Proceedings of the 8th Brazillian Symposium on GeoInformatics*. Advances in Geoinformatics, Springer, 129-140.

[13] Hobbs, J.R. and Pan, F. 2004. An Ontology of Time for the Semantic Web. *ACM Transactions on Asian Language Information Processing (TALIP) 3(1)*, 66-85.

[14] Hogenboom, F., Borgman, B., Frasincar, F. and Kaymak, U. 2010. Spatial Knowledge Representation on the Semantic Web. In *Proceedings of the 4th IEEE International Conference on Semantic Computing*. ICSC'10, 252-259.

[15] Horridge, M., Knublauch, H., Rector, A., Stevens, R. and Wroe, C. 2004. A Practical Guide to Building OWL Ontologies Using the Protégé-OWL Plugin and CO-ODE Tools Edition 1.0. University of Manchester.

[16] Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B. and Dean, M. 2004. SWRL: A Semantic Web Rule Language. Retrieved May 19, 2011, from World Wide Web Consortium (W3C): http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[17] Jitkajornwanich, K., Elmasri, R., Li, C. and McEnery, J. 2011. Formalization of 2-D Spatial Ontology and OWL/Protégé Realization. Technical Report. CSE-2011-6, The University of Texas at Arlington.

[18] Mark, D., Egenhofer, M., Hirtle, S. and Smith, B. 2000. Ontological Foundations for Geographic Information Science. University Consortium for Geographic Information Science (UCGIS).

[19] Milea, V., Fransincar, F. and Kaymak, U. 2009. A Temporal Web Ontology Language. Erasmus Research Institute of Management (ERIM).

[20] Noy, N.F. and McGuinness, D.L. 2001. Ontology Development 101: A Guide to Creating Your First Ontology. Study Guide. Stanford University.

[21] O'Connor, M.J. and Das, A.K. 2009. SQWRL: A Query Language for OWL. In *Proceedings of OWL: Experiences and Directions*. OWLED'09. CEUR Workshop Proceedings 529.

[22] O'Connor, M.J. and Das, A.K. 2010. A Method for Representing and Querying Temporal Information in OWL. *Communications in Computer and Information Science 127*, 97-110.

[23] Open GIS Consortium. 1999. OpenGIS Simple Features Specification For SQL. Open Geospatial Consortium, Inc.

[24] Parent, C., Spaccapietra, S. and Zimanyi, E. 1999. Spatio-Temporal Conceptual Models: Data Structures + Space + Time. In *Proceedings of the 7th ACM International Symposium on Advances in Geographic Information Systems*. GIS'99, 26 - 33.

[25] Parent, C., Spaccapietra, S. and Zimanyi, E. 2006. *Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS Approach*. Springer.

[26] Randell, D.A., Cui, Z. and Cohn, A.G. 1992. A Spatial Logic Based on Regions and Connection. In *Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning*. KR'92. Morgan Kaufmann, 165-176.

[27] Sandia National Laboratories. Jess®: The Rule Engine for the Java™ Platform. Retrieved Jan 1, 2011, from: http://www.jessrules.com/jess/index.shtml.

[28] Shekhar, S. and Chawla, S. 2003. *Spatial Databases: A Tour*. Pearson Education, New Jersey.

[29] Spaccapietra, S., Cullot, N., Parent, C. and Vangenot, C. 2004. On Spatial Ontologies. Database Laboratory, Swiss Federal Institute of Technology, Lausanne, Switzerland.

[30] Stanford Center for Biomedical Informatics Research. 2011. Protégé Project. Retrieved May 20, 2011, from Protégé: http://protege.stanford.edu.

[31] Wang, X., Zhou, X. and Lu, S. 2000. Spatiotemporal Data Modeling and Management: Survey. In *Proceedings of the 36th International Conference on Technology of Object-Oriented Languages and Systems*. IEEE Xplore, 202-211.

[32] Egenhofer, M.J. 1989. A Formal Definition of Binary Topological Relationships. In *Proceedings of the 3rd International Conference on Foundations of Data Organization and Algorithms*. FODO'89, 457-472.

[33] Egenhofer, M.J. 1991. Categorizing Binary Topological Relations between Regions, Lines, and Points in Geographic Databases. Technical Report. 90-12, Department of Surveying Engineering, University of Maine, Orono, ME.

[34] Egenhofer, M.J. and Mark, D.M. 1995. *Modeling Conceptual Neighborhoods of Topological Line-Region Relations*. *International Journal of Georgaphical Information Systems 9(5*), 555–565.

[35] Reid, D.A. and Knipping, C. 2010. *Proof in Mathematics Education: Research, Learning and Teaching*. Sense Publishers. Rotterdam, Netherlands.

[36] Asquith, W.H., Roussel, M.C., Cleveland, T.G., Fang X. and Thompson D.B. 2006. Statistical Characteristics of Storm Interevent Time, Depth, and Duration for Eastern New Mexico, Oklahoma, and Texas. Professional Paper 1725. U.S. Geological Survey (USGS).

[37] Asquith, W.H. 1998. Depth-Duration Frequency of Precipitation for Texas. Water-Resources Investigations Report 98-4044. U.S. Geological Survey (USGS).

[38] Asquith, W.H. and Roussel, M.C. 2004. Atlas of Depth-Duration Frequency of Precipitation Annual Maxima for Texas. Scientific Investigations Report 2004-5041 (TxDOT Implementation Report 5-1301-01-1). U.S. Geological Survey (USGS).

[39] Lanning-Rush, J., Asquith, W.H. and Slade R.M. 1998. Extreme Precipitation Depth for Texas, Excluding the Trans-Pecos Region. Water-Resources Investigations Report 98-4099. U.S. Geological Survey (USGS).

[40] Asquith, W. H., Thompson, D.B., Cleveland, T.G. and Fang, X. 2004. Synthesis of Rainfall and Runoff Data used for Texas Department of Transportation Research Projects 0-4193 and 0-4194. Open-File Report 2004-1035. U.S. Geological Survey (USGS).

[41] Elmasri, R. and Navathe, S. 2010. *Fundamentals of Database Systems (6th edition)*. Pearson Education, Massachusetts.

[42] National Oceanic and Atmospheric Administration (NOAA). 2011. National Weather Service River Forecast Center: West Gulf RFC (NWS-WGRFC). Retrieved December 31, 2011, from: http://www.srh.noaa.gov/wgrfc/.

[43] McEnery, J. 2011. CUAHSI HIS: NWS-WGRFC Hourly Multi-sensor Precipitation Estimates. Retrieved December 31, 2011, from: http://hiscentral.cuahsi.org/-pub_network.-aspx?n=187.

[44] Consortium of Universities for the Advancement of Hydrologic Science, Inc. (CUAHSI). 2008. HydroDesktop. Retrieved October 26, 2011, from: http://his.cuahsi.org-/hydrodesktop.html.

[45] Consortium of Universities for the Advancement of Hydrologic Science, Inc. (CUAHSI). 2008. ODM Databases. Retrieved October 26, 2011, from: http://his.cuahsi.org-/odmdatabases.html.

[46] NOAA Satellite and Information Service. 2012. National Climatic Data Center (NCDC). Retrieved March 15, 2012, from: http://www.ncdc.noaa.gov/oa/ncdc.html.

[47] Consortium of Universities for the Advancement of Hydrologic Science, Inc. (CUAHSI). 2008. Universities Allied for Water Research. Retrieved October 26, 2011, from: http://www.cuahsi.org/.

[48] Google. 2012. Google Developers: Google Maps API. Retrieved April 14, 2012, from: https://developers.google.-com/maps/.

[49] NOAA's National Weather Service. 2011. The XMRG File Format and Sample Codes to Read XMRG Files. Retrieved December 31, 2011, from: http://www.nws.noaa.gov/oh/hrl/-dmip/2/xmrgformat.html.

[50] Overeem, A., Buishand, A. and Hollemanet, I. 2008. Rainfall Depth-Duration-Frequency Curves and Their Uncertainties. *Journal of Hydrology 348 (1-2)*, 124-134.

[51] Virginia Department of Conservation and Recreation. 2012. Stormwater Management: Hydrologic Methods. Retrieved May 2, 2012, from: http://dcr.cache.vi.virginia.gov/storm-water_management/documents/Chapter_4.pdf.

[52] Asquith, W.H. 2005. Summary of Dimensionless Texas Hyetographs and Distribution of Storm Depth Developed for Texas Department of Transportation Research Project 0-4194. Report 0-4194-4. U.S. Geological Survey (USGS).

[53] Suyanto, A., O'Connell, P.E. and Metcalfeet. A.V. 1995. The Influence of Storm Characteristics and Catchment Conditions on Extreme Flood Response: A Case Study Based on the Brue River Basin, U.K. *Surveys in Geophysics 16(2)*, 201-225.

[54] Franks, B. 2012. *Taming The Big Data Tidal Wave: Finding Opportunities in Huge Data Streams with Advanced Analytics*. John Wiley & Sons, Inc., Hoboken, New Jersey.

[55] DevZone. 2011. *Big Data Bibliography*. O'Reilly Media.

[56] Linoff, G. 2007. *Data Analysis Using SQL and Excel*. Wiley Publishing, Inc., Indianapolis, Indiana.

[57] Cameron, S. 2009. *Microsoft® SQL Server® 2008 Analysis Services Step by Step*. Microsoft Press, Redmond, Washington.

[58] Horsburgh, J.S., Tarboton, D.G., Maidment, D.R. and Zaslavsky, I. 2008. A Relational Model for Environmental and Water Resources Data, Water Resources Research.

[59] George, W.B. 2012. *WEATHER: The Handbook of Texas Online.* Retrieved September 5, 2012, from: http://www.tshaonline.org/handbook/online/articles/yzw01.

[60] Frontier Associates, LLC. 2008. Texas Renewable Energy Resource Assessment. Texas State Energy Conservation Office.

[61] Freeman, A. 2010. *Pro .NET 4 Parallel Programming in C#*. Springer-Verlag, New York.

[62] Jitkajornwanich, K., Elmasri, R., Li, C. and McEnery, J. 2012. Extracting Storm-Centric Characteristics from Raw Rainfall Data for Storm Analysis and Mining. In

Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data. ACM SIGSPATIAL BIGSPATIAL'12, 91-99.

[63] Cary, A., Sun, Z., Hristidis, V. and Rishe, N. 2009. Experiences on Processing Spatial Data with MapReduce. In *Proceedings of the 21st International Conference on Scientific and Statistical Database Management*. SSDBM'09, 302-319.

[64] Wu, X., Carceroni, R., Fang, H., Zelinka, S. and Kirmse, A. 2007. Automatic Alignment of Large-Scale Aerial Rasters to Road-maps, Geographic Information Systems. In *Proceedings of the 15th ACM International Symposium on Advances in Geographic Information Systems.* ACM GIS'07.

[65] Schlosser, S.W., Ryan, M.P., Taborda, R., Lopez, J., O'Hallaron, D.R. and Bielak, J. 2008. Materialized Community Ground Models for Large-Scale Earthquake Simulation. In *Proceedings of the 2008 ACM/IEEE International Conference for High Pergormance Computing, Networking, Storage and Analysis*. SC'08.

[66] Dean, J. and Ghemawat, S. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*. OSDI'04.

[67] Lam, C. 2011. *Hadoop in Action*. Dreamtech Press, New Delhi, 2011.

[68] Jitkajornwanich, K., Gupta, U., Elmasri, R., Fegaras, L. and McEnery, J. 2013. Using MapReduce to Speed Up Storm Identification from Big Raw Rainfall Data. In *Proceedings of the 4th International Conference on Cloud Computing, GRIDs, and Virtualization.* CLOUD COMPUTING'13, 49-55.

[69] Embley, D.W. and Liddle, S.W. 2013. Big Data—Conceptual Modeling to the Rescue. In *Proceedings of the 32nd International Conference on Conceptual Modeling*. ER'13.

[70] Valiant, L.G. 2008. A Bridging Model for Multi-Core Computing. In *Proceedings of the 16th Annual European Symposium.* ESA'08.

[71] Apache. Apache Spark™. 2014. Retrieved March 26, 2014, from: http://spark.apache.org.

[72] Zou, B., Ma, X., Kemme, B., Newton, G. and Precup, D. 2006. Data Mining Using Relational Database Management Systems. In *Proceedings of the 10th Pacific-Asia Conference*. PAKDD'06.

[73] nosql-database.org. 2014. List of NOSQL Databases. Retrieved March 26, 2014, from: http://nosql-database.org.

[74] Amazon. 2014. Amazon DynamoDB. Retrieved March 26, 2014, from: http://aws.amazon.com/dynamodb.

[75] MongoDB. 2014. MongoDB. Retrieved March 26, 2014, from: http://www.mongodb.org.

[76] Jitkajornwanich, K., Gupta, U., Shanmuganathan, S.K., Elmasri, R., Fegaras, L. and McEnery, J. 2013. Complete Storm Identification Algorithms from Big Raw Rainfall Data. In *Proceedings of the 2013 IEEE International Workshop on Big Data and Science: Infrastructure and Services*.

[77] Unidata. 2014. What is the LDM? Retrieved March 26, 2014, from: https://www.unidata.ucar.edu/software/ldm/ldm-6.6.5/tutorial/whatis.html.

[78] Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R.E. 2006. Bigtable: A Distributed Storage System for Structured Data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*. OSID'06.

[79] NOAA. 2014. MPE: Multisensor Precipitation Estimate. Retrieved March 26, 2014, from: http://www.erh.noaa.gov/marfc/Maps/xmrg/index_java.html.

[80] Mishra, S.K. and Singh, V.P. 2003. *Soil Conservation Service Curve Number (SCS-CN) Methodology*. Kluwer Academic Publishers, 2003.

[81] Dahan, H., Cohen S., Rokach, L. and Maimon, O. 2014. Proactive Data Mining with Decision Trees. Springer.

[82] Cheng, T., Haworth, J., Anbaroglu, B., Tanaksaranond, G. and Wang J. 2013. Spatio-Temporal Data Mining, *Handbook of Regional Science*, Springer-Verlag Berline Heidelberg, 2013.

[83] Drummond, N. and Horridge, M. 2005. A Practical Introduction to Ontologies and OWL. Retreived March 25, 2011, from The University of Manchester: http://www.co-ode.org/resources/tutorials/intro/.

[84] Rubin, D.L., Noy, N.F. and Musen, M.A. 2007. Protégé: A Tool for Managing and Using Terminology in Radiology Applications. *Journal of Digital Imaging (0)0*, 1-13.

[85] Tauberer, J. 2005. What is RDF and what is it good for?. Retrieved February 14, 2011, from rdf:about: http://www.rdfabout.com/intro/?section=7.

[86] Grenon, P. and Smith, B. 2004. SNAP and SPAN Towards Dynamic Spatial Ontology. *Spatial Cognition and Computation*.

[87] McGuinness, D.L. and Harmelen, F. 2004. OWL Web Ontology Language Overview. W3C Recommendation.

[88] Lu, S., Li, R.M., Tjhi, W.C., Lee, K.K., Wang, L., Li, X. and Ma, D., 2011. A Framework for Cloud-Based Large-Scale Data Analytics and Visualization: Case Study on Multiscale Climate Data, In *Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science*. CloudCom'11.

[89] Kondekar, R., Gupta, A., Saluja, G., Maru, R., Rokde, A. and Deshpande, P. 2012. Iterative Mapreduce Based Heuristic Algorithm for Solving N Puzzle. In *Proceedings of the* 2012 International Conference on Computer & Information Science. ICCIS'12.

[90] Tangient. 2014. Iterative MapReduce and Counters. Retrieved April 1, 2014, from: http://hadooptutorial.wikispaces.com/Iterative+MapReduce+and+Counters.

[91] Lin, J. and Schatz, M. 2010. Design Patterns for Efficient Graph Algorithms in MapReduce. In *Proceedings of the 8th Workshop on Mining and Learning with Graphs*. MLG '10.

[92] Ekanayake, J. 2010. Twister: A Runtime for Iterative MapReduce. In *Proceedings of the 1st International Workshop on MapReduce and its Applications*. MAPREDUCE'10.

[93] Bu, Y., Howe, B., Balazinska, M. and Ernst, M.D. 2010. HaLoop: Efficient Iterative Data Processing on Large Clusters. In *Proceedings of the VLDB Endowment*, Vol. 3, No. 1.

[94] Bhatotia, P., Wieder, A., Rodrigues, R., Acar, U.A. and Pasquini, R. 2011. Incoop: MapReduce for Incremental Computations. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. SOCC'11.

[95] Peng, D. and Dabek, F. 2010. Large-scale Incremental Processing Using Distributed Transactions and Notifications. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*. OSDI'11.

[96] Akintug, B. 2014. Precipitation. Retrieved April 1, 2014, from: http://www.metu.edu.tr/~bertug/SEES503/SEES%20503%20-%202%20Precipitation.pdf.

[97] Ponce, V.M. 1989. *Engineering Hydrology: Principles and Practices*. Prentice Hall, Englewood Cliffs, New Jersey.

[98] Ester, M., Kriegel, H., Sander, J. and Xu, X. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*. KDD'96.

[99] McEnery, J. and Jitkajornwanich, K. 2012. Depth-Area-Duration Characteristics of Storm Rainfall in Texas using Multi-Sensor Precipitation Estimates. 2012 American Geophysical Union. AGU'12.

[100] McEnery, J., Shelton, G. and Jitkajornwanich, K. 2012. ArcGIS Online Map Feature for NWS Precipitation Data in the Dallas/Fort Worth Metroplex. 2012 Texas GIS Forum.

[101] American Meteorological Society. 2014. Glossary of Meteorology: Rain. Retrieved April 1, 2014, from: http://glossary.ametsoc.org/wiki/Rain.

[102] Ikenaga, B. 2014. Proof by Cases. Retrieved May 5, 2014, from Millersville University: http://www.millersville.edu/~bikenaga/math-proof/cases/cases.html.

[103] Driscoll, E.D., Palhegyi, G.E., Strecker E.W. and Shelley, P.E. 1989. Analysis of Storm Event Characteristics for Selected Rainfall Gages throughout the United States. U.S. Environmental Protection Agency.

[104] Ratanamahatana, C.A., Lin, J., Gunopulos, D., Keogh, E., Vlachos, M. and Das, G. 2010. Mining Time Series Data. *Data Mining and Knowledge Discovery Handbook*. Springer US.

[105] Gruber, T. 1995. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human-Computer Studies*, 43:5-6, Nov./Dec. 1995, 907-928.

Biographical Information

Kulsawasd Jitkajornwanich received his bachelor's degree with honors in computer science from Chulalongkorn University, Bangkok, Thailand in 2004. After graduation, he worked for two years for Bangkok Bank and Reuters (now Thomson Reuters) before receiving a Royal Thai government scholarship to pursue his graduate studies in 2006. He received his master's degree in computer science from The University of Texas at Arlington in 2009. In the same year, he started his PhD in computer science at The University of Texas at Arlington. He received his doctoral degree in computer science in 2014. His area of interest includes spatio-temporal ontology, spatio-temporal databases, GIS, map-reduce, customized data analysis and mining for rainfall precipitation data, and conceptual analysis and mining of big data. He will be working with Geo-Informatics and Space Technology Development Agency (GISTDA) in Bangkok, Thailand. GISTDA is a public organization providing services and conducting researches/projects relevant to geo-information, which could be of benefit to the public as well as is responsible for other space technologies (e.g., satellites) and geo-informatics applications. During his PhD studies, he taught a database systems 1 course (CSE 3330/5330: Database Systems and File Structures) and was a teaching assistant for a database systems 2 course (CSE 4331/5331: DBMS Models and Implementation Techniques). In addition, he worked with National Weather Service – West Gulf River Forecast Center (NWS-WGRFC) and Tarrant Regional Water District (TRWD) in preparing data for their water management decisions including flooding prediction. He also published several research papers including ACM SWIM (part of ACM SIGMOD/PODS), ACM BIGSPATIAL (part of ACM SIGSPATIAL GIS), CLOUD COMPUTING, and IEEE BIG DATA.