

DESIGN AND IMPLEMENTATION OF SPATIOTEMPORAL DATABASES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

AZİZ SÖZER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

JULY 2010

Approval of the thesis:

DESIGN AND IMPLEMENTATION OF SPATIOTEMPORAL DATABASES

submitted by **AZİZ SÖZER** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof.Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences**

Prof.Dr. Adnan Yazıcı
Head of Department, **Computer Engineering**

Prof.Dr. Adnan Yazıcı
Supervisor, **Computer Engineering Dept., METU**

Assoc.Prof.Dr. Halit Oğuztüzün
Co-Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. İsmail Hakkı Toroslu
Computer Engineering Dept. METU

Prof. Dr. Adnan Yazıcı
Computer Engineering Dept. METU

Assoc.Prof.Dr. Dr. Ahmet Coşar
Computer Engineering Dept. METU

Asst. Prof. Dr. Pınar Şenkul
Computer Engineering Dept. METU

Asst. Prof. Dr. İbrahim Körpeoğlu
Computer Engineering Dept. Bilkent University

Date: **02/07/2010**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Aziz Sözer

Signature :

ABSTRACT

DESIGN AND IMPLEMENTATION OF SPATIOTEMPORAL DATABASES

Sözer, Aziz

Ph.D., Department of Computer Engineering

Supervisor : Prof.Dr. Adnan Yazıcı

Co-supervisor : Assoc.Prof.Dr. Halit Oğuztüzün

July, 2010, 163 pages

Modeling spatiotemporal data, in particular fuzzy and complex spatial objects representing geographic entities and relations, is a topic of great importance in geographic information systems, computer vision, environmental data management systems, etc. Because of complex requirements, it is challenging to design a database for spatiotemporal data and its features and to effectively query them. This thesis presents a new approach for modeling, indexing and querying the spatiotemporal data of fuzzy spatial and complex objects and/or spatial relations. As a case study, we model and implement a meteorological application in an intelligent database architecture, which combines an object-oriented database with a knowledge base.

Keywords: Spatiotemporal Data, Object-Oriented Database, Knowledge Base, Fuzzy Objects, Meteorological Database Application

ÖZ

UZAM-ZAMANSAL VERİTABANLARININ TASARIM VE GERÇEKLEŞTİRİMİ

Sözer, Aziz

Doktora, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi : Prof.Dr. Adnan Yazıcı

Ortak Tez Yöneticisi : Doç.Dr. Halit Oğuztüzün

Temmuz, 2010, 163 sayfa

Uzam-zamansal verileri, özellikle coğrafi oluşum ve ilişkileri temsil eden bulanık ve karmaşık uzamsal nesneleri modellemek coğrafi bilgi sistemleri, bilgisayarlı görme, çevresel veri yönetim sistemleri, vb. için çok önemli bir konudur. Karmaşık gereksinimler nedeniyle, uzam-zamansal veriler ve özellikleri için veritabanı tasarlamak ve etkin şekilde sorgulamak zordur. Bu tez çalışması, bulanık uzamsal ve karmaşık nesnelerin ve/veya uzamsal ilişkilerin uzam-zamansal verilerini modelleme, endeksleme ve sorgulama için yeni bir yaklaşım sunmaktadır. Örnek çalışma olarak, uzam-zamansal nesnelerin modellenmesi ve sorgulanması için nesneye dayalı veritabanı ile bilgi tabanını akıllı bir veritabanı mimarisinde birleştiren meteorolojik veritabanı uygulaması gerçekleştirilmiştir.

Anahtar Kelimeler: Uzam-Zamansal Veri, Nesneye Dayalı Veritabanı, Bilgi Tabanı, Bulanık Mantık, Meteorolojik Veritabanı Uygulaması

To my parents, Türkan–Mustafa Sözer

and

Zeynep Selek, Salih Taşbaş

ACKNOWLEDGEMENTS

I would like to thank my supervisor Prof.Dr. Adnan Yazıcı and co-supervisor Assoc.Prof.Dr. Halit Oğuztüzün for their encouragement, advice and guidance throughout this study. Their continuous support and insight made this thesis possible.

I would like to thank to my thesis jury members Prof. Dr. İsmail Hakkı Toroslu, Assoc.Prof.Dr. Ahmet Coşar, Asst.Prof.Dr. Pınar Şenkul and Asst.Prof.Dr. İbrahim Körpeoğlu for their valuable comments and guidance.

I would like to thank to Levent Yalçın, Sevcin Yalçın from the Turkish State Meteorological Service and Cihan Şahin from European Center for Medium-Range Weather Forecasts, who are weather forecasters and city planning experts and helped me to have meteorological data, mapping and continuous support.

I would like to thank to my colleagues in Central Bank of Turkey for motivating me for the success of my PhD. study. I would like to thank Burçin Bostan Körpeoğlu for valuable comments and suggestions throughout my study.

At last, but not the least, I would like to thank to my family for their patience, love and belief in me. I am especially grateful to my beloved wife Serapcan and sister Filiz, who provided everything I needed. Without their patience, continuous assistance and long-time friendship, I would not have had the strength to complete this work.

TABLE OF CONTENTS

ABSTRACT.....	iv
ÖZ	v
ACKNOWLEDGEMENTS.....	vii
TABLE OF CONTENTS	viii
LIST OF TABLES	xi
LIST OF FIGURES.....	xii
CHAPTER	
1. INTRODUCTION	1
1.1. Scope of the Study	1
1.2. Summary of Contributions.....	5
1.3. Organization of the Thesis	6
2. BACKGROUND AND RELATED WORK	7
2.1. Spatial Objects.....	7
2.2. Spatial Data Types.....	10
2.3. Spatial Relations.....	11
2.3.1. Fuzzy Topological Relations	12
2.3.2. Topological Relations Between Complex Regions.....	14
2.4. Temporal Aspects of Spatiotemporal Databases.....	16
2.5. Spatial Indexing.....	17
2.6. Related Work.....	20
2.6.1. Modeling Spatiotemporal Data	20
2.6.2. Querying Spatiotemporal Data.....	22
2.6.3. Spatial Index Structures.....	24
3. A GENERIC MODEL FOR SPATIOTEMPORAL MODELING	26
3.1. The Fuzzy Object-Oriented Database (FOOD) Model.....	26
3.2. The Generic Model	28

3.2.1. C-Logic Notation.....	28
3.2.2. Alloy Notation.....	31
3.3. The Object Model.....	45
3.4. Coupling The Fuzzy Database With a Fuzzy Knowledge Base.....	50
4. THE ARCHITECTURE OF THE SPATIOTEMPORAL DATABASE APPLICATION	52
5. QUERY PROCESSING	55
5.1. Fuzzy Non-Spatial Query.....	56
5.2. Complex Spatial Query.....	57
5.3. Fuzzy Spatiotemporal Query.....	62
5.4. Nested Rule Query.....	66
5.5. Fuzzy Spatiotemporal Query.....	67
6. FUZZY SPATIAL/ASPATIAL INDEXING	71
6.1. Enhanced R*-Tree	71
6.2. Building the Enhanced R*-Tree	74
6.3. The Visualization of the Enhanced R*-Tree.....	76
6.4. Querying the Enhanced R*-Tree	81
6.4.1. Crisp Aspatial Queries.....	81
6.4.2. Crisp Spatial Queries.....	84
6.4.3. Fuzzy Spatial/Aspatial Queries.....	84
7. IMPLEMENTATION AND PERFORMANCE EVALUATION	87
7.1. Implementation.....	87
7.2. An Object Oriented Database, Db4O	88
7.3. The Rule Engine, Jess.....	88
7.4. Crisp Queries.....	90
7.4.1. Point Query	91
7.4.2. Range Query	94
7.4.3. Circle Query.....	97
7.4.4. Ring Query.....	100
7.4.5. K th Nearest Neighbor (KNN).....	102
7.5. Fuzzy Queries.....	106
7.5.1. Fuzzy Spatial Relations Query.....	106

7.5.2. Fuzzy Spatiotemporal Query	110
7.6. Fuzzy Semantic Queries.....	113
7.6.1. Fuzzy Semantic Query 1 (Extreme Conditions)	113
7.6.2. Fuzzy Semantic Query 2 (Trajectory of Objects)	117
7.6.3. Fuzzy Semantic Query 3 (k Highest Measurements).....	120
7.6.4. Fuzzy Semantic Query 4 (Agricultural Risky Zones)	122
7.6.5. Fuzzy Semantic Query 5.....	124
7.7. Experimental Evaluation.....	128
7.7.1. The Scalability of the Application	128
7.7.2. The Enhanced R*-Tree	131
7.7.3. The Effect of Third Dimension.....	135
8. CONCLUSIONS	141
REFERENCES	143
APPENDICES	
A. SAMPLE METEOROLOGICAL MAPS	149
B. THE OBJECT MODEL SPECIFICATIONS IN ALLOY	151
CURRICULUM VITAE	162

LIST OF TABLES

TABLES

Table 1 The similarity matrix for temperature attribute.....	27
Table 2 Sample records in database	56
Table 3 Similarity matrix of cloudiness attribute	56
Table 4 Objects in the FOOD	59
Table 5 Computing a fuzzy topological relation for a wavy region and ferry lines .	65
Table 6 Computing a fuzzy topological relation for a windy region and ferry lines	65
Table 7 The overall fuzzy relation degrees	66
Table 8 The text file structure for meteorological data	77

LIST OF FIGURES

FIGURES

Figure 1	A weather chart showing wave heights on the Mediterranean Sea	8
Figure 2	Temperature mapping on 31.12.2007	9
Figure 3	Humidity mapping on 31.12.2007	9
Figure 4	Spatial data types	10
Figure 5	Visualization of a simple fuzzy region	12
Figure 6	Examples of topological relations between fuzzy regions	14
Figure 7	Examples of the relations (a) <i>disjoint</i> and (b) <i>inside</i>	15
Figure 8	An example of R-tree	19
Figure 9	A fuzzy spatiotemporal model	46
Figure 10	Meteorological application model	48
Figure 11	The architecture of the spatiotemporal database application	53
Figure 12	Query evaluation algorithm	54
Figure 13	Maximum temperature regions (a) and Meteorological events (b) on 01.01.2008	58
Figure 14	Complex topological predicate evaluation algorithm	60
Figure 15	Wave height (a) and wind speed (b) over Marmara Sea	62
Figure 16	Fuzzy topological predicate evaluation algorithm	64
Figure 17	Meteorological objects on 01.01.2008 (a) and 02.01.2008 (b)	67
Figure 18	The algorithm to evaluate area, speed and direction change	68
Figure 19	Rain object movement on 01.01.2008 (a) and 02.01.2008 (b)	69
Figure 20	The structure of Enhanced R*-tree	72
Figure 21	Enhanced R*-tree insertion algorithm	74
Figure 22	Enhanced R*-tree visualization	78
Figure 23	Enhanced R*-tree nodes	79
Figure 24	Crisp Aspatial Query Algorithm in Enhanced R*-tree	82

Figure 25	Crisp Spatial Query Algorithm in Enhanced R*-tree	83
Figure 26	Fuzzy Spatial/Aspatial Query Algorithm in Enhanced R*-tree	85
Figure 27	Database visualization by Object Manager tool	89
Figure 28	An example of rules	90
Figure 29	Point query input screen	91
Figure 30	Point query algorithm	92
Figure 31	Point query results	93
Figure 32	Range query input screen	94
Figure 33	Range query algorithm	95
Figure 34	Range query result	96
Figure 35	Circle query input screen	97
Figure 36	Circle query algorithm	98
Figure 37	Circle query result	99
Figure 38	Ring query input screen	100
Figure 39	Ring query algorithm	101
Figure 40	Ring query result	102
Figure 41	KNN query input screen	103
Figure 42	KNN query algorithm	104
Figure 43	KNN query result	105
Figure 44	Fuzzy Spatial Relations Query input screen	107
Figure 45	Fuzzy Spatial Relations Query algorithm	108
Figure 46	Fuzzy Spatial Relations Query result	109
Figure 47	Fuzzy Spatiotemporal query input screen	110
Figure 48	Fuzzy Spatiotemporal query algorithm	111
Figure 49	Fuzzy Spatiotemporal query result	112
Figure 50	Fuzzy Semantic query 1 (Extreme conditions) input screen	114
Figure 51	Fuzzy Semantic query 1 (Extreme conditions) algorithm	114
Figure 52	The degree of extremeness function in FKB	115
Figure 53	Fuzzy Semantic query 1 (Extreme conditions) result	117
Figure 54	Fuzzy Semantic query 2 (trajectory of objects) input screen	118
Figure 55	Fuzzy Semantic query 2 (trajectory of objects) algorithm	118
Figure 56	Fuzzy Semantic query 2 (trajectory of objects) results	119

Figure 57	Fuzzy Semantic query 3 (k Highest Measurements) input screen.....	120
Figure 58	Fuzzy Semantic query 3 (k Highest Measurements) algorithm	121
Figure 59	Fuzzy Semantic query 3 (k Highest Measurements) result.....	122
Figure 60	Fuzzy Semantic query 4 (Agricultural risky zones) input screen.....	123
Figure 61	Fuzzy Semantic query 4 (Agricultural risky zones) algorithm	123
Figure 62	Fuzzy Semantic query 4 (Agricultural risky zones) result.....	124
Figure 63	Fuzzy Semantic query 5 input screen	125
Figure 64	Fuzzy Semantic query 5 algorithm	125
Figure 65	Fuzzy Semantic query 5 result.....	126
Figure 66	Altitude vs. meteorological parameters variations (a) Humidity (b) Pressure (c) Wind speed	127
Figure 67	The scalability of the system by Crisp Spatial Queries	129
Figure 68	The scalability of the system by Fuzzy Spatial Queries	130
Figure 69	The scalability of the system by Fuzzy Semantic Queries.....	130
Figure 70	The primary index build time	131
Figure 71	The secondary index build time.....	132
Figure 72	Fuzzy Spatiotemporal Query run with R*-tree and Enhanced R*-tree	133
Figure 73	Semantic Query (trajectory) run with R*-tree and Enhanced R*-tree.	134
Figure 74	Semantic Query performance of R*-tree and Enhanced R*-tree	135
Figure 75	Number of inner nodes in Enhanced R*-Tree	136
Figure 76	Number of data nodes in Enhanced R*-Tree	136
Figure 77	The nodes of three dimensional Enhanced R*-Tree	137
Figure 78	Primary index building time for Enhanced R*-Tree	137
Figure 79	Secondary index building time for Enhanced R*-Tree.....	138
Figure 80	Range Query performance of Enhanced R*-Tree (1)	138
Figure 81	Range Query performance of Enhanced R*-Tree (2)	139
Figure 82	Fuzzy Semantic Query performance of Enhanced R*-Tree (1)	139
Figure 83	Fuzzy Semantic Query performance of Enhanced R*-Tree (2)	140
Figure 84	Cloudiness mapping on 30.12.2007.....	149
Figure 85	Pressure mapping on 30.12.2007.....	149
Figure 86	Wind strength mapping on 30.12.2007.....	150
Figure 87	Sunshine duration mapping on 30.12.2007	150

CHAPTER 1

INTRODUCTION

1.1 Scope of the Study

Space and time are inherent notions in spatiotemporal applications. These applications (e.g. traffic control, environmental, meteorological, etc.) include spatial and temporal data and variations. For example, a moving car in traffic changes position over time. The borders of a salty lake move back and forth because of seasonal evaporation and rainfall. The size and speed of a meteorological storm may change during its lifetime. Hence spatiotemporal databases are required to deal with both spatial and temporal phenomena.

Modeling spatiotemporal data is difficult because of such spatial variations and advanced data structures and techniques are needed [20, 31, 44]. In modeling spatiotemporal data, two approaches have been widely used: field-based and object-based modeling [33]. The field-based approach assumes the real world to have attributes which are varying over space as a continuous function. For example, contour lines on a map represent points of a constant value (e.g. pressure, temperature, velocity, density) within a volume of space. On the other hand, the object-based approach distinguishes fully definable disjunctive objects. That is, the database for the application stores a map that consists of a collection of identifiable objects, which refer to the partitions and fragments of information space. For example, the temperature regions (e.g. cold, warm, etc.), and the rainy or foggy

areas on a weather map are reflected with their unique attributes (e.g. borders, position, direction, etc.) in the database.

Uncertainty and fuzziness are also features of most spatiotemporal applications. Spatial and temporal information and various relationships often involve uncertainty and fuzziness. For example, in describing a windy and wavy region, the region's boundary is inherently fuzzy. In the case of estimating a moving weather object, the need to determine its position at a certain time, or its time of arrival at a certain location, gives rise to fuzzy estimations. The most common reasons for considering various types of uncertainty in spatiotemporal applications are as follows:

- Some spatial information is imprecise or fuzzy. The locations of objects, spatial relationships and various geometric and topological properties usually involve uncertainty [47].
- Many natural phenomena have fuzzy boundaries due to the transitional nature of variation in the phenomenon (e.g. a river's changing line because of floods and drought) [6, 15, 35].
- To obtain precise data is difficult and unnecessary most of the time, and we may only be able to give a range of values in which the exact numbers would lie. For instance, we may need the number of "cloudy" or "partly cloudy" days for some region in a period. In this request, the user specifies cloudiness criteria in linguistic terms instead of giving numeric degrees of cloudiness (e.g. 4/8 or 6/8) [3].

There have been several efforts aimed at using fuzzy set theory for modeling spatial objects and their properties [40, 41, 42, 44, 48, 59]. Schneider *et. al.* [40, 41, 42] represent fuzzy spatial objects and relationships as well as complex crisp objects and relationships by using fuzzy techniques. Tang *et. al.* [48] propose basic fuzzy spatial object types based on a fuzzy topology. A fuzzy cell complex is defined for fuzzy points, lines and regions. Zhan *et.al.* [59] describe how to find the resultant regions from the topological overlay of two simple polygons with indeterminate boundaries. The effect of three typical overlay operations – intersection, difference, and union –

on resultant regions is also discussed. Tao *et. al.* [49] study range query on multidimensional uncertain data using a “probabilistically constrained rectangle”.

Temporality has also been studied by some researchers [34, 37, 62]. In its simplest form, time is considered as an attribute of spatial objects in [37]. A simple time stamping approach is adequate to obtain the states of objects at certain times. However, to identify individual changes in objects, event-based approaches are developed in [34]. In [62] temporal uncertainty and fuzzy timing are introduced in a model that combines temporality and fuzziness. In this model the notions of fuzzy time stamping, enabling time, occurrence time and delays are defined.

There are also efforts to combine spatial and temporal properties into one modeling framework using an object-oriented modeling approach [16, 17, 50]. Tenets of object-orientation, such as classes and instances, attributes and abstract data types, operations and methods, classification and encapsulation, aggregation, information hiding, inheritance, polymorphism and dynamic binding are very useful for modeling and manipulating spatiotemporal data. Worboys [55] introduces the concept of the spatiotemporal object and defined a spatiotemporal object as a unified object with both spatial and temporal extents, also called a simplex. A finite set of such spatiotemporal simplexes are then defined to form a spatiotemporal complex on the basis of which query algebra is developed.

The object oriented modeling approach is also used to support fuzzy data. This lead to development of fuzzy object-oriented modeling techniques for imperfect information requirements of various complex applications. Gyseghem *et. al.* [18] propose an object-oriented model that represents uncertainty and fuzzy information. In that work, fuzzy information is presented by fuzzy sets and uncertainty by means of generalized fuzzy sets. Bordogna *et. al.* [4] define a graph based fuzzy object-oriented data model that permits attributes to take linguistic values. The association between an object instance and instance properties are modeled through a fuzzy reference relation. Lee *et. al.* [26] propose a new approach to object-oriented modeling based on fuzzy logic to formulate fuzzy classes, fuzzy rules to describe the

relationship between attributes, the membership function of a fuzzy class based on both static and dynamic properties, and uncertain fuzzy associations between classes. Marin *et. al.* [29] present a set of operators to compare objects in a fuzzy setting. Among them is a generalized resemblance degree between two fuzzy sets of imprecise objects and to compare complex fuzzy objects. Yazici and George [56] study a similarity based fuzzy object-oriented data model in which impreciseness at the data level contributes to uncertainty in the class-object and class-subclass hierarchy. In this thesis we introduce some extensions to that model for spatiotemporal objects.

There also exist some other studies extending conceptual models for modeling fuzzy information. For example, Geo-ER [19] is an extension of the entity-relationship (ER) model that provides a set of concepts specific to the spatial application domain, and attempts to capture spatial peculiarities at the conceptual level of geographic database design. Yazici *et. al.* [58] use unified modeling language (UML) [2], providing extensions to handle spatial and temporal objects. In their work, some new special entity sets, relationships, and constructs were introduced for modeling spatial objects.

In knowledge intensive applications, support for deduction is an important requirement. In a spatiotemporal application, relations between objects can be very complex. Consider, for example, a ship crossing the sea. In some parts the sea line may be restricted for travel due to wave and wind conditions. How can we record this information and make the deduction that the sea line is restricted? The spatiotemporal data can be stored in databases but naturally there are relations including some rules as well. Instead of storing all relations in a database, a knowledge base that is capable of representing knowledge and making deductions is preferable and very helpful for retrieving the status of the sea line. Hence, the interaction and/or integration of database and knowledge base technologies are important requirements for the development of knowledge intensive applications. This is reflected in the continuing research into the development of deductive object-oriented models since the late 1980s [9, 27].

1.2 Summary of Contributions

In this study, we present a new approach to model and query real world spatiotemporal objects, in particular meteorological phenomena. The main contributions of this thesis and our working layout can be summarized as follows:

- A generic model is introduced. Spatiotemporal objects and relations are incorporated into the model. The types of the objects and relations can be classified as follows:
 - Complex crisp spatial objects,
 - Fuzzy spatial objects,
 - Crisp/fuzzy spatial relations (e.g. topological, directional and metric) between spatial objects.

Then an application specific model is combined with the generic model. The geographic objects (i.e. city, sea, line, etc.) and meteorological objects (i.e. temperature, wind, waves, etc.) are included in a three dimensional space.

- The model including generic and specific parts and fuzzy spatiotemporal querying mechanism are presented logically by C-logic [52] and conceptually by extended UML [46]. The model is also specified formally using Alloy and verified by Alloy analyzer [21].
- Following modeling efforts, an architecture is designed by utilizing the Intelligent Fuzzy Object-Oriented Database (IFOOD) [25] including components which are:
 - an object oriented database,
 - a knowledge base,
 - a querying interface and fuzzy spatial.
- The architecture is implemented as a prototype application. Using meteorological data some crisp and fuzzy queries are implemented to verify the application.

- The queries are enhanced and diversified by adapting a spatial index structure (R*-tree) [22, 45]. The adapted R*-tree (Enhanced R*-tree) supports fuzzy spatiotemporal queries.
- Finally, the fuzzy semantic queries are run with real meteorological data and the efficiency and scalability of the application is evaluated.

1.3 Organization of the Thesis

In the following chapter, we give some background information on concepts related to fuzzy spatiotemporal database modeling, including spatial and temporal fuzziness as well as relationships between fuzzy and complex objects. Then a comprehensive related work summary is presented in the same chapter. In Chapter 3, we describe how to develop a generic model for spatiotemporal database applications. We use a meteorological database application to illustrate our approach. Chapter 4 gives details about the architectural design of the system. In Chapter 5, we present queries from the application domain, and discuss crucial details of their processing. The development of a fuzzy index structure (Enhanced R*-tree) are explained in Chapter 6. Chapter 7 diversifies proof-of-concept queries in Chapter 5 by adding fuzzy index structure and real data. The implementation details of crisp, fuzzy and semantic queries are presented. We also evaluate the scalability and performance of the application components. Finally, we present our conclusions and point out possible future studies, in the last chapter.

CHAPTER 2

BACKGROUND AND RELATED WORK

In order to support our modeling and querying aspects, basic spatial and temporal concepts are discussed in this chapter. The spatial objects in geographic information systems especially in meteorological maps are presented in Section 2.1. Basic definitions of fuzzy spatial data types are given in Section 2.2. We describe spatial relations in general and topological relations between complex regions and fuzzy regions in Section 2.3. The temporal requirements of a spatiotemporal application are presented in Section 2.4. These are followed by a summary of index requirements and R-tree/R*-tree descriptions in Section 2.5. Finally, we give a summary of the related work in literature and our contributions in Section 2.6.

2.1 Spatial Objects

In many areas of geographic information systems, natural objects (e.g. mountains, rivers, aridity areas, population distribution areas and meteorological phenomena like foggy regions, wavy sea regions, etc.) and man-made objects (e.g. cadastral divisions, administrative borders of the cities, roads and bridges etc.) are modeled, stored and queried. The objects are defined with spatial (e.g. geometric shape, location, boundary length, diameter etc.) and/or descriptive (e.g. name, origin etc.) attributes [28].

In Figure 1, wave heights over the Mediterranean Sea are illustrated on a weather map. According to the map, the wave heights have varying characteristics, which are most dense in south-west of Italy and clear on the Eastern Mediterranean. The borders of the density regions are indeterminate since the height characteristic changes somewhat gradually.

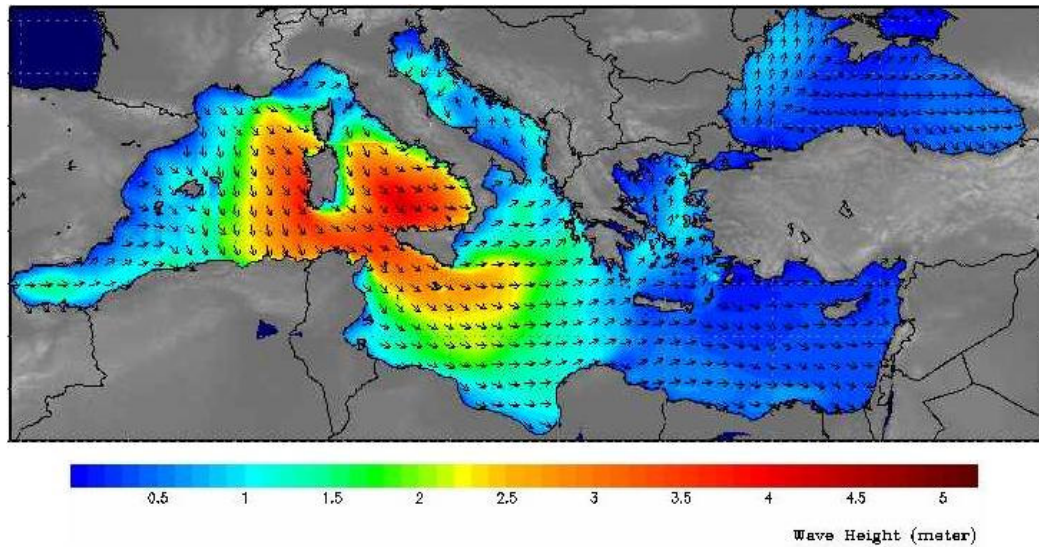


Figure 1: A weather chart showing wave heights on the Mediterranean Sea

In Figure 2, the temperature values of Turkey on 31.12.2007 are depicted on a map using GIS software [14]. The software groups the similar values together with close tones of colors. So the cold and warm areas and the gradual changes are visualized on the map.

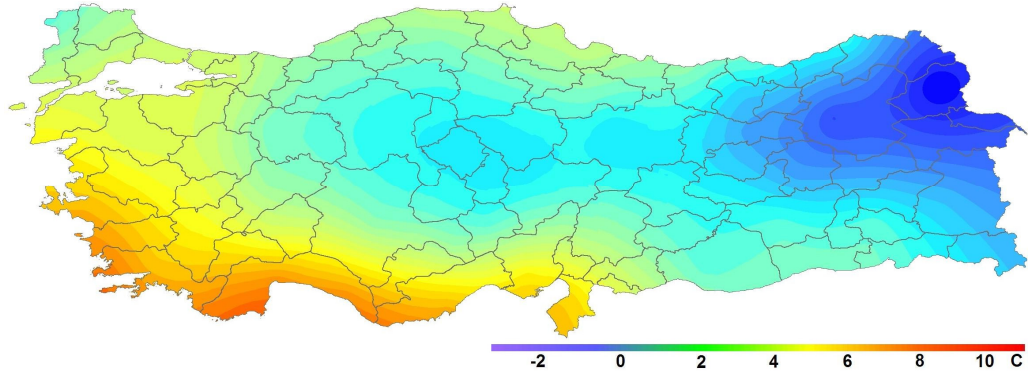


Figure 2: Temperature mapping on 31.12.2007

Figure 3 is a similar map produced for the humidity measurements of Turkey at the same date. The brown colors indicate the relatively dry regions whereas greenish colors indicate the humid regions. Similar maps produced for different meteorological parameters are presented in Appendix A.

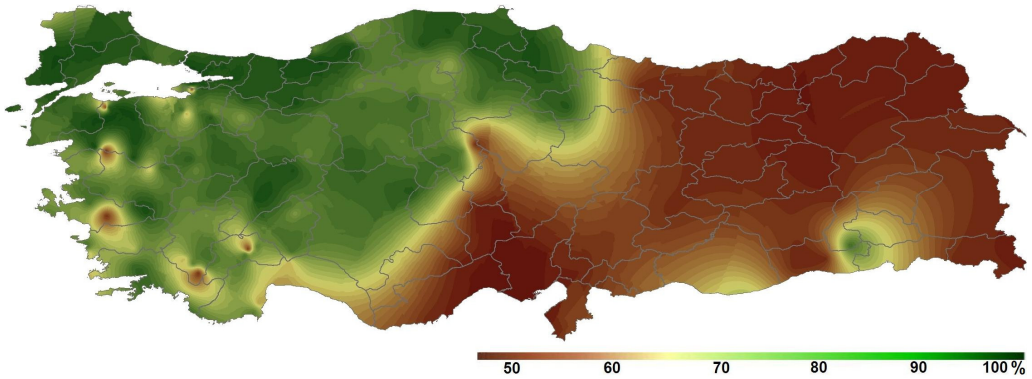


Figure 3: Humidity mapping on 31.12.2007

In a geographic space like these maps, the objects with imprecise or vague spatial attributes could be referred to as *fuzzy spatial objects* and the ones with precise or exact attributes (e.g. country borders) could be referred to as *crisp spatial objects* [40]. We define for fuzzy spatial objects, namely fuzzy points, lines and regions in the next section.

2.2 Spatial Data Types

A *fuzzy point* is a point for which an exact position is not known but possible positions are known within a certain area. In Figure 4-(a) the expected position of such a point is shown by a black dot and the possible positions are shown by grey dots. For instance, a ship waiting in the queue for crossing “Istanbul Bosphorus” is supposed to be found at a certain point but may drift from that position from time to time (e.g. move to the grey parts).

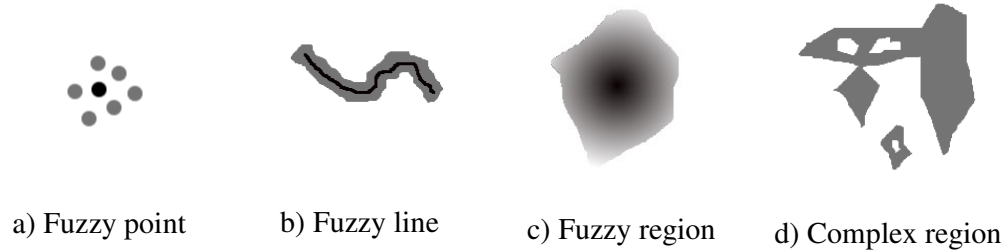


Figure 4: Spatial data types

A *fuzzy line* is a line, the exact shape, position or length of which is not known, but what is known is which area the line must reside in. In Figure 4-(b) the center line

shows the normal shape of a river. The actual river line can change position and shape due to floods or droughts (hence the grey area).

A *fuzzy region* is a region with indeterminate boundaries. It has three parts: (1) the core (indicated by the dark part) (2) the indeterminate boundary (grey part) and (3) the exterior (the outer parts of indeterminate boundary) [59]. In Figure 4-(c) a typical fuzzy region is depicted and might be used to express the gradual change over a spatial domain for a given attribute (e.g. wave height).

Finally, a *complex region* is a set of regions, possibly with holes and multiple components (see, Figure 4-(d)) [40]. Foggy regions with clear patches, for example, can be represented as complex regions.

2.3 Spatial Relations

Spatial relationships can be one of the three kinds, namely, topological (e.g., *overlap*, *inside*, *covers*, etc.), directional (e.g., *North (N)*, *South East (SE)*, etc.) and metric (e.g., "*5 km away from*") relationships [7].

Topological relations describe spatial relationships of objects in space. A model for analyzing binary topological relations, known as the *9-intersection model*, has been proposed in the literature [13]. The 9-intersection model is based on the intersection between the parts (*interior*, *boundary*, *exterior*) of the regions involved. The intersections of the parts are analyzed with 3x3 matrices (total $2^9=512$ matrices). The model distinguishes eight meaningful (*disjoint*, *meet*, *overlap*, *equal*, *contains*, *inside*, *covers* and *covered by*) relations for crisp regions. Later, this model was generalized for fuzzy regions [32, 48, 59] and complex regions [11, 41]. These generalizations are presented in the next sections.

2.3.1 Fuzzy Topological Relations

The topological relations between fuzzy regions are inevitably fuzzy because of the indeterminate boundaries of the regions involved. Suppose that “A” is a set of attributes under consideration, and that a region is a fuzzy subset defined in two dimensional space R^2 over “A”. The membership function of the fuzzy region can be defined as $\mu : X \times Y \times A \rightarrow [0,1]$, where X and Y are the sets of coordinates defining the region. Each point (x, y) within the region is assigned a membership value for an attribute a in A .

A fuzzy region is illustrated in Figure 5 with *the core, the indeterminate boundary, the exterior* and α -cut levels. The indeterminate boundary of the fuzzy region is an aggregation of regions whose boundary is defined by α -cut levels, that is all points with membership values equal to α . So, an α -cut level region is defined as follows:

$$R_\alpha = \{(x, y, a) \mid \mu_R(x, y, a) \geq \alpha\} (0 < \alpha < 1) \quad (1)$$

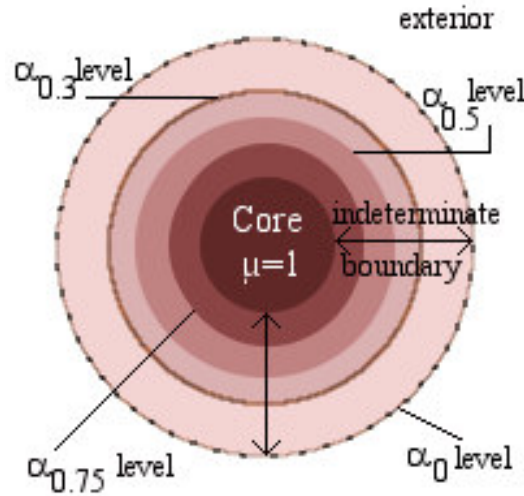


Figure 5: Visualization of a simple fuzzy region

The degree of the fuzzy relation is measured by aggregating the α -cut level of fuzzy regions. The *basic probability assignment* $m(R_{\alpha_i})$, which can be interpreted as the probability that R_{α_i} is the true representative of R , is defined as in [12, 41, 59]:

$$m(R_{\alpha_i}) = \alpha_i - \alpha_{i+1} \quad (2)$$

for $1 \leq i \leq n$ for some $n \in N$ with $1 = \alpha_1 > \alpha_2 > \dots > \alpha_n > \alpha_{n+1} = 0$

It is clear that $\sum_{i=1}^n m(R_{\alpha_i}) = 1$.

Let $\tau(R_{\alpha_i}, S_{\alpha_j})$ indicate the existence of a topological relation between two α -cut level regions of fuzzy regions R and S (e.g. 0 or 1). Then the degree of a topological relation between R and S can be determined by the following equation:

$$\tau(R, S) = \sum_{i=1}^n \sum_{j=1}^m m(R_{\alpha_i}) m(S_{\alpha_j}) \tau(R_{\alpha_i}, S_{\alpha_j}) \quad (3)$$

Because $\tau(R_{\alpha_i}, S_{\alpha_j})$ has a value of either 0 or 1 and $\sum_{i=1}^n m(R_{\alpha_i}) = \sum_{j=1}^m m(S_{\alpha_j}) = 1$, $\tau(R, S)$ should be in $[0, 1]$.

Here we present the formulation for the overlap relation as an example but the remaining topological relations which are illustrated in Figure 6 can be formulated in a similar manner:

$$\tau(R, S) = \sum_{i=1}^n \sum_{j=1}^m m(R_{\alpha_i}) m(S_{\alpha_j}) \tau_{\text{overlap}}(R_{\alpha_i}, S_{\alpha_j}) \quad (4)$$

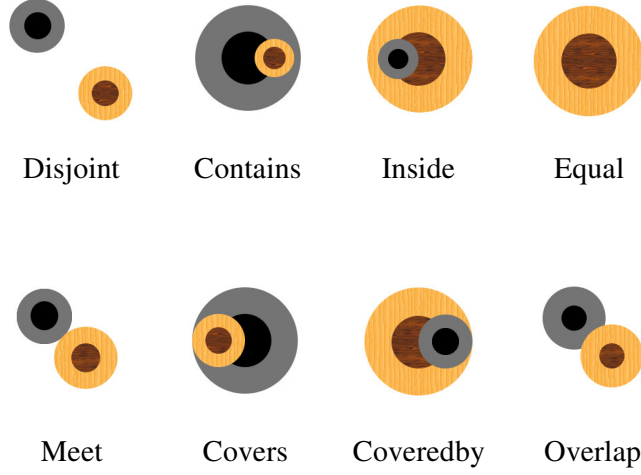


Figure 6: Examples of topological relations between fuzzy regions.

2.3.2 Topological Relations Between Complex Regions

A complex region is the union of simple regions (SR) including, possibly, holes. Let F and G be two *simple regions with holes*, that is

$$F_{SR} = F_0 - \bigcup_{i=1}^n F_i \text{ and}$$

$$G_{SR} = G_0 - \bigcup_{j=1}^m G_j, \quad (5)$$

where F_0 and G_0 are bases and F_i and G_j are the holes of F and G respectively. Then, two regions are disjoint if F_0 and G_0 are disjoint or one region is inside of another region's hole. More precisely,

$$disjoint_{SR}(F, G) = disjoint(F_0, G_0) \vee \left(\exists 1 \leq i \leq n : inside(G_0, F_i) \right) \vee \left(\exists 1 \leq j \leq m : inside(F_0, G_j) \right) \quad (6)$$

, where

$$inside_{SR}(F, G) = inside(F_0, G_0) \wedge \left(\forall 1 \leq j \leq m : disjoint(F_0, G_j) \vee \left(inside(G_j, F_0) \wedge \exists 1 \leq i \leq n : inside(G_j, F_i) \right) \right) \quad (7)$$

F is considered to be inside G if F_0 is inside G_0 and if each hole G_j of G is either disjoint from F_0 or inside a hole of F_i . Exemplary regions with holes and their relations are illustrated in Figure 7.

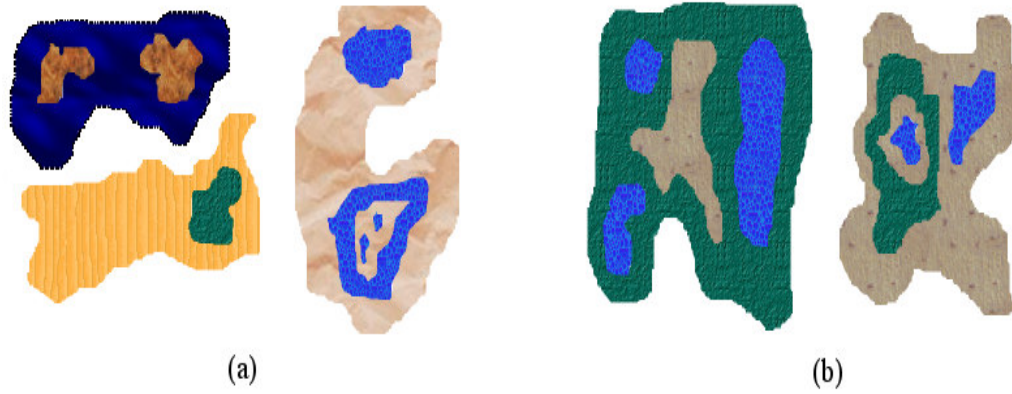


Figure 7: Examples of the relations (a) *disjoint* and (b) *inside*

Other topological predicates for simple regions, possibly with holes, are defined in the same vein. Based on these definitions, topological predicates for complex regions are defined as follows:

Let

$$F_{CR} = \bigcup_{i=1}^n F_i \text{ and } G_{CR} = \bigcup_{j=1}^m G_j$$

be two *complex regions* (CR), where F_i and G_j are simple regions with holes. Then the topological relations are defined as follows:

$$disjoint_{CR}(F, G) = \forall 1 \leq i \leq n \forall 1 \leq j \leq m : disjoint_{SR}(F_i, G_j) \quad (8)$$

$$\begin{aligned} meet_{CR}(F, G) = & \neg disjoint_{CR}(F, G) \wedge \\ & (\forall 1 \leq i \leq n \forall 1 \leq j \leq m : (disjoint \mid meet)_{SR}(F_i, G_j)) \\ inside_{CR}(F, G) = & \forall 1 \leq i \leq n \exists 1 \leq j \leq m : inside_{SR}(F_i, G_j) \end{aligned} \quad (9)$$

$$contains_{CR}(F, G) = inside_{CR}(G, F) \quad (10)$$

$$equal_{CR}(F, G) = \forall 1 \leq i \leq n : equal_{SR}(F_i, G_i) \quad (11)$$

$$\begin{aligned} coveredBy_{CR}(F, G) = & ((inside \mid equal)_{CR}(F, G)) \wedge \\ & (\forall 1 \leq i \leq n \exists 1 \leq j \leq m : (inside \mid coveredBy \mid equal)_{SR}(F_i, G_j)) \end{aligned} \quad (12)$$

$$covers_{CR}(F, G) = coveredBy_{CR}(G, F) \quad (13)$$

$$overlap_{CR}(F, G) = \neg(disjoint \mid meet \mid inside \mid contains \mid equal \mid coveredBy \mid covers)_{CR}(F, G) \quad (14)$$

2.4 Temporal Aspects of Spatiotemporal Databases

Temporal aspects have been the focus of attention in the literature, and applications often require that time information to be stored in the database. Information about

objects' attributes and relationships among objects are valid *when* the object exists temporally. For example, windy regions exist over the sea within a time interval and the ships which have to cross these regions are planned to start and finish their journeys at certain times. The windy regions and the ship routes will be expected to relate to each other in certain ways in this interval. Temporal information is generally stored in databases in two forms:

- the *valid time* is the time when the information about an object or relationship holds in the modeled reality. For example the valid times of a ferry route in the Marmara Sea is 08:30, 12:00 and 17:00 daily.
- the *transaction time* of a database entry is the time when the entry becomes a part of the current state of the database. The time when the ferry lines' times are stored in the database is the transaction time of the entry.

Individual time values are termed *chronons* and many applications also have *duration*, which can be captured by using time *intervals*, where a time interval $[t_{\text{begin}}, t_{\text{end}}]$ is defined as a set of consecutive chronons. We call t_{begin} and t_{end} the start and the end chronon of the interval, respectively.

2.5 Spatial Indexing

The design of a spatiotemporal database should meet the unique requirements of spatiotemporal data. In this section, we discuss a number of requirements at the physical level.

A spatial object has a complex structure. It may be composed of numerous points, line segments and polygons with holes and vague parts. It is not usually possible to store such collections in a single relational table. Spatial objects also have dynamic properties. The attributes like shape, position, etc. may change by the time. Data structures used in this context should support this dynamic behavior. The complex

structure and the dynamic behaviors result in large databases comparatively. Finally, the spatial operators are generally more expensive than the standard relational operators. The spatial search operations require special support at the physical level because of the existence of spatial attributes as well as non-spatial attributes.

Spatial index structures are designed to support such operations at physical level. Based on the properties of spatiotemporal data, spatial index structures should be dynamic for changing attributes, scalable for database growth, and should support broad range of operations. There are a number of index structures which support spatial indexing [45]. R-tree index family is widely studied [22] and we adapt one of them, R*-tree for this study.

An R-Tree is an index structure for spatial data. At the leaf node of R-Tree, an index record refers to the spatial data. The index record is an n-dimensional rectangle and it is the bounding rectangle of the spatial data indexed. This rectangle is also known as minimal bounding rectangle, MBR. Non-leaf nodes contain entries (I, childnode-pointer) where I is the MBR bounding all the rectangles in the lower nodes' entries. Childnode-pointer is the pointer to a lower node in the R-Tree.

Figure 8 shows an example of an R-tree. In the figure, leaf nodes (LN_1 to LN_7) are the enclosing rectangles of original spatial data objects in the data space. Inner nodes (IN_1 to IN_3), are MBRs of corresponding leaf nodes, and they are stored in the root of this R-tree.

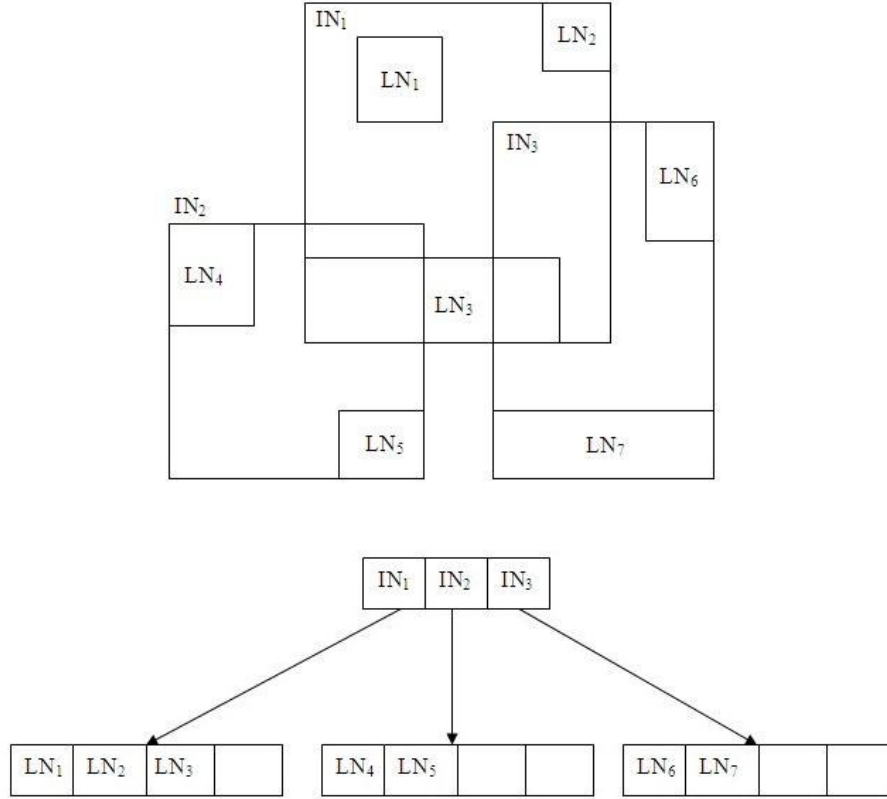


Figure 8: An example of R-tree

R*-tree is basically different from R-tree in the insertion phase. The design of the R*-tree introduces a policy called *forced reinsert*: If a node overflows, it is not split right away but firstly p entries are removed from the node and reinserted into the tree. The parameter p may vary; but it is suggested that p to be about 30% of the maximal number of entries per page [1].

Another difference between R-tree and R*-tree is the node splitting policy. While R-tree algorithms try to minimize the area that is covered by the bucket regions, the R*-tree algorithms also take into account minimum overlap between bucket regions, minimum region perimeters and maximum storage utilization.

2.6 Related Work

A recent literature survey about the topic is presented in this section. The works are classified under the modeling, querying and indexing sections.

2.6.1 Modeling Spatiotemporal Data

In this section a number of works about modeling spatiotemporal objects are checked and compared to our work. The fuzzy object modeling, the past and future states of moving objects, the conceptual, logical and physical modeling are the main topics of our survey.

In [24] finite number of crisp regions where each region is associated with a membership value indicating the degree of belonging, forms so called a plateau region. Thus, a fuzzy region in Figure 5 is approximated by n crisp regions. So an implementation effort can benefit from well known crisp region algebra. The authors define formal plateau regions and operations. In our work in the prototype system we implemented complex regions with holes and fuzzy regions. The fuzzy region which is used to represent wavy and windy regions over the sea is a kind of plateau region mentioned in [24] with some differences. In that work plateau regions are presented like complex spatial objects with multiple parts each having different fuzzy degrees whereas in our implementation fuzzy regions of windy areas are more crisped shape of the simple fuzzy region. So our fuzzy regions have a core region and from core to outside some non uniform crisp rings cover the core with decreasing memberships.

Like plateau regions, vague spatial object and topological predicates [32], fuzzy spatial data types [40], complex crisp and simple fuzzy regions [41] and complex spatial objects and topological predicates [42] are formalized in a number of work

by Schneider and co-workers. We also handled fuzzy and complex spatial objects and the formalism presented in these papers is quite useful for our modeling efforts.

The past and future states of moving objects are modeled in [36]. The formal definitions of moving objects with respect to their past and future movements are provided. In our study, we used past meteorological data and hence for moving objects. In the ferry lines example, the restrictions for the line are queried. In weather forecasting, the future states of meteorological objects are predicted. So if we use future data our model can be used to query for the future topological relations (e.g. the future states of the ferry lines will be restricted can be queried).

A survey on multidimensional modeling discusses the issues about the phases of modeling [38]. The modeling phases are described at conceptual, logical and physical level. The conceptual modeling aims an implementation independent and expressive schema. In literature, conceptual modeling has been searched from two perspective, multidimensional and Extraction-Transformation-Loading (ETL) modeling. The multidimensional modeling approaches use extensions to Entity Relationships model, UML and ad-hoc models. The authors state that ETL is less mature then multidimensional modeling. The logical modeling takes place after conceptual modeling and creates a logical schema. Finally physical design phase concerns the issues specifically related to the implementation such as *indexing*. In our research effort we follow the conceptual, logical and physical modeling order. The conceptual model is based on extended UML for spatiotemporal data. The logical design use C-logic and Alloy for a logical schema of the model. Finally at the physical level we used an object oriented database supported by a spatial index structure.

A recent work on modeling and querying vague spatial objects [63] uses shapelets which is an image decomposition technique developed in astronomy. Shapelet is as set of functions that includes a Gaussian function and higher order terms composed of the products of Gaussian function with a set of polynomials. The shapelets approach is optimized especially for smoothly varying fuzzy spatial objects. The

arithmetic operations (add/subtract two objects), topological operations (overlap, etc.) and metric operations are defined formally for shapelets. An extension including shapelet class is integrated into PostgreSQL as a library. In addition, existing R-tree support in PostgreSQL is used to index shapelets. There are a couple of differences in our approach and this approach. We use vague spatial objects which are formalized by fuzzy set theory and complex crisp objects whereas in this work shapelets are used to model spatial objects especially for smoothly varying objects. The vague spatial objects and complex crisp objects are very suitable for modeling meteorological objects. Both work use a database (PostgreSQL and db4o) and a spatial index structure (R-tree and R*-tree). The R-tree and R*-tree use bounding boxes as a standard indexing mechanism. In addition to that, we adapt R*-tree for fuzzy spatial and aspatial indexing for fuzzy and semantic queries. In querying spatial data, in addition to topological, metric relations we implement also fuzzy semantic queries which may require deduction defined in fuzzy knowledge base.

Our modeling effort follows conceptual, logical and physical modeling of spatiotemporal data. We use fuzzy set theory in modeling fuzzy data. The definitions for the fuzzy objects and complex crisp objects which have foundations in literature [42] are suitable for meteorological objects. So our modeling efforts are parallel with the works above and we present a complete architecture including modeling at three levels supported by knowledge base and fuzzy semantic querying.

2.6.2 Querying Spatiotemporal Data

In this section we briefly give an overview of recent works from literature which is related to querying methods. The location based queries, nearest neighbor and distance searches are mainly the subjects of these works. Also fuzzy inputs and processing are required to process the queries.

The location of reference objects and the target instances in a given range are the subject location based spatial queries (LBSP) [5]. LBSQ take a reference point and find instance objects in a distance or a range. The distance may be specified between minimum and maximum values. The range may be a geometrical shape like circle. The reference point and the instance locations can be uncertain also. The range, circle, ring, distance range, fuzzy topological (inside, overlap, etc) queries are part of our work.

In [43], an intelligent querying tool, TreeSap is presented. The tool focuses on qualitative fuzzy input when querying spatial data. The distance relationships between objects can be stated ambiguously such as *near*, *close*, *very close*, etc. TreeSap converts this qualitative input to a numerical form and presents the results of the query which is in numerical form in a simple and intuitive manner.

In [23], modeling and querying uncertain location information from free text which is obtained from newspaper or event reporting sources is studied. Uncertain locations such as near (Building A) and event types such as ‘traffic accident’ are extracted from text and mapped onto probability density functions (pdf). They also analyze several types of spatial queries such as range search.

ESSE [61] system allows user to query the environmental data archives in human linguistic terms. These terms are mapped into query language by fuzzy logic. Fuzzy states of spatiotemporal data sources are specified as logical expressions (AND, OR, NOT) applied to a set of linguistic terms (*Large*, *Small*, etc.) and numeric predicates (*Less than*, *equal*, etc.). The transformation from one state to another state by the time is also defined formally.

These works in the previous paragraphs are focused on a specific part of spatial querying. The location, distance, topological relations are implemented in our work as well as other fuzzy semantic queries. We use meteorological objects and data sets although it can be implemented for other data sets. We also adapt index structure and knowledge base for efficient fuzzy semantic querying.

2.6.3 Spatial Index Structures

The indexing approaches in recent literature are discussed in this section. The usage of R-tree index family and using separate or single index structures and fuzzy query supporting issues are the main topics of these works.

Using separate index structures for different dimensions are discussed in [30]. In the survey for geographic information retrieval systems two separate index structures for text and geographical scopes are planned to use. We used single index structure for spatial and non-spatial indexing. The additional indexing is added at the data and intermediate nodes. The advantages of our approach are firstly it is dynamic so any number of attributes of objects at the leaf level can be indexed and secondly during a search only one index is loaded and all indexing criteria can be found in the same level and structure.

Another work also uses two separate index structures for video clips. The variants of R-tree and R*-tree are adapted for a two phased retrieval algorithm in content based multimedia system [60]. While first tree is built on the spatial objects in the scenes of a video, the second tree stores the spatial relations between the objects in the first tree.

The relative spatial orientation and distance relation are indexed for icons which are a collection of labeled point features in [10]. In this approach each pairing of two icons is represented by a single point and all pairs with same separation and relative orientation map to same point. Given a spatial relationship or range, the database search is performed by using R-theta index which is a variant of R-tree. In our approach the spatial relations and orientation are calculated dynamically by fuzzy spatial processor and/or fuzzy knowledge base together.

The U-tree is built on the multidimensional uncertain data for range search [49]. Like R*-tree's MBRs, a U-tree uses *probabilistic constrained rectangles (pcr)*, where $pcr(0)=MBR$ and $pcr(0.6)$ is a smaller rectangle and so on. As the probability

of pcr increases to 1, the bounding rectangle gets smaller. A query returns all objects that appear in search region with at least a certain or higher probability. This approach gives probability values to rectangles so in the core part of the rectangles the object is found most probably. In our approach we use crisp rectangles to index objects. Nevertheless the spatial (direction, relation, etc) and aspatial fuzziness are supported as we mentioned.

The indexing approaches in these works are mainly differentiating from our work in some couple of ways. In summary, we use single indexing structure which may be more costly during the built up but more efficient while querying. We put index on each attribute of the objects including temporal data. The spatial relations are not indexed but calculated dynamically. Since the meteorological objects change position, shape etc. continuously we believe that our approach is more convenient. Finally in our work, any fuzzy attribute at the leaf level is indexed in the intermediate nodes and it is dynamic which is independent of the number of attributes.

CHAPTER 3

A GENERIC MODEL FOR SPATIOTEMPORAL MODELING

In this chapter, the components of the generic spatiotemporal model, namely, the fuzzy object oriented database (FOOD) [57] and the fuzzy knowledge base (FKB) [25], are presented.

3.1 The Fuzzy Object-Oriented Database (FOOD) Model

The Fuzzy Object Oriented model supports multivalued attributes and fuzzy domains are defined for these attributes. The domain of an attribute is the set of all possible values that the attribute can take. For example, the fuzzy domain for a “temperature” attribute of a meteorological observation can be defined as:

$$Domain_{temperature} = \{hot, warm, moderate, cool, cold\} \quad (15)$$

That is, the temperature attribute can have some combination of these values from the domain such as $\{hot, warm\}$, $\{warm\}$, $\{cool, cold, moderate\}$. The similarity matrix in Table 1 shows the similarity of each element with other elements in the domain.

Table 1: The similarity matrix for temperature attribute

Temperature	hot	warm	moderate	cool	cold
hot	1.0	0.6	0.4	0	0
warm	0.6	1.0	0.8	0.2	0
moderate	0.4	0.8	1.0	0.6	0.4
cool	0	0.2	0.6	1.0	0.8
cold	0	0	0.4	0.8	1.0

The matrix indicates that *cool* and *cold* temperatures are similar with a degree of 0.8. In a case where the temperature value is estimated and given a threshold value of 0.8, multiple values {cool, cold} can be associated, which gives us a fuzzy representation for temperature value. Note that the values of the similarity relations can be defined either by domain experts or computed using various methods existing in the literature [53].

In FOOD, attributes can take values within a range and in general, $range \subseteq domain$. The range of an attribute a_i of a class C is represented by the notation $rng_c(a_i)$, where $a_i \in \{a_1, a_2, \dots, a_n\}$, the attributes of class C . For example, the range of the temperature attribute of a class for a “fog” object can be defined as a subset of the temperature domain:

$$rng_{fog}(temperature) = \{moderate, cool, cold\} \quad (16)$$

Another type of fuzziness in FOOD takes place between classes and objects. That is, while some objects are full members of a fuzzy class, some other objects may belong to the class partially. The objects may still be considered as instances of this class but with a degree of membership in $[0, 1]$. A formal range definition indicating the ideal values for a fuzzy attribute is given in the class definition. However, an attribute of an object can take any value from the related domain. Then, the degree

of membership of an object to its class is computed by using the similarities between the attribute values and the range values, and the relevance of fuzzy attributes. The relevance is given by the weight of the fuzzy attribute in determining the boundary of a fuzzy class. Thus, the degree of membership of an object O_j to a class C is determined by the formula:

$$\mu_C(o_j) = \sum_{i=1}^n INC(rng_C(a_i)/o_j(a_i)) \times RLV(a_i, C) / \sum_{i=1}^n RLV(a_i, C) \quad (17)$$

where $INC(rng_C(a_i)/o_j(a_i))$ is the inclusion value that is taking into account the semantics of attributes and $RLV(a_i, C)$ is the relevance of attribute a_i to the class C , as given in the class definition by the class designer. All attributes, therefore, affect membership degrees in proportion to their relevance values. For the details of the FOOD model, including examples of the computation of inclusion values, the reader is referred to [57].

3.2 The Generic Model

In this section, the types, operations and predicates for a generic spatiotemporal model are specified. C-Logic and Alloy notations are used for the formal definitions of the model. Then in the next chapter implementation details are presented.

3.2.1 C-Logic notation

We use C-Logic, which allows direct transformation of the specification into first order formulas [54]. C-logic also allows class and subclass specification independently, which facilitates the update of objects' subparts. This specification can then be easily implemented in an object oriented programming language.

In C-logic a class is specified as a collection of atomic properties. For example, a “Point” is a spatial data type having a membership value to exist at a location (x, y).

$$Point := [\mu \Rightarrow [float], x \Rightarrow [float], y \Rightarrow [float]], \text{ where } 0 \leq \mu \leq 1.0. \quad (18)$$

For example, an object of type *Point* can be defined as $p := Point[\mu \Rightarrow 1.0, x \Rightarrow 3.0, y \Rightarrow 5.5]$, which indicates that the point p is located at (3.0, 5.5) with membership value $\mu = 1.0$. A more general specification for Point class is given below:

$$Point := [\mu \Rightarrow \{\mu_1, \dots, \mu_n\}, x \Rightarrow \{x_1, \dots, x_n\}, y \Rightarrow \{y_1, \dots, y_n\}] \quad (19)$$

where $0 \leq \mu \leq 1.0$ and $(x, y) \in R^2$.

The mapping \Rightarrow can be understood as either “containing as a subset”, if it is followed by a collection of terms, or “containing an element”, if it is followed by a single term. In the former case, terms with the same index are associated with each other.

An object is defined as an instance of a class within an interval of time and specified by a predicate *is_instance*:

$$is_instance(Object, Class, T), \text{ where } T_{begin} \leq T \leq T_{end}$$

In a spatiotemporal model, the spatial portions of objects are described with “Geometry”, “Point”, “Line” and “Region” classes. The “Point”, “Line” and “Region” classes are associated with each other through aggregation relation. For example, a line can be described by an aggregation of points and similarly a region can be represented by an aggregation of lines, and so on. A special form of aggregation is the “whole/parts” relation between “Geometry” and part classes,

namely, “Point”, “Line” and “Region”. “Geometry” is formed by the combination of more than one spatial type and expressed in “Geometry” class definition as follows:

$$\begin{aligned}
 \text{Geometry} &:= [parts \Rightarrow \{P_i, L_j, R_k\}, size \Rightarrow [float], MBR \Rightarrow [Reg_1], center \Rightarrow [P_1]], \\
 \text{where} \quad &T_{begin} \leq T \leq T_{end}, is_instance(P_i, Point, T), is_instance(L_j, Line, T), \\
 &is_instance(R_k, Region, T), is_instance(Reg_1, Region, T), \\
 &is_instance(P_1, Point, T) \quad i, j, k \in N, \text{ and at least one of } i, j \text{ and } k > 0
 \end{aligned} \tag{20}$$

The other attributes of the “Geometry” class are “Minimum Bounding Rectangles (MBR)” [8] for locating and accessing objects in space, “center” for the central position of the object and “size” for the volume that an object occupies.

The “STObject” class is associated with the “Geometry” class from which detailed spatial information is extracted, such as *geometries* and possible *holes* for regions as well as the *position* and *trajectory* of an object. The “STObject” class, having geometric and temporal attributes is defined below. Note that an “STObject” can have at least one simple geometry with possible holes.

$$\begin{aligned}
 \text{STObject} &:= \left[\begin{aligned} &geometry \Rightarrow \{Geo_i\}, holes \Rightarrow \{Hole_j\}, position \Rightarrow [Pos], \\ &trajectory \Rightarrow \{Traj_k\}, times \Rightarrow \{T_{begin}, \dots, T_{end}\} \end{aligned} \right], \text{ where} \\
 &is_instance(T, DateTime), is_instance(Geo_i, Geometry, T), \\
 &is_instance(Hole_j, Geometry, T), is_instance(Pos, Point, T), \\
 &is_instance(Traj_k, Point, T), \\
 &T_{begin} \leq T \leq T_{end}, i, j, k \in N \text{ and } i \geq 1, j \geq 0, k \geq 1
 \end{aligned} \tag{21}$$

A fuzzy spatial relation describes the relative positions of two fuzzy spatial objects. The degree of relation can be computed by using the definitions in Section 2.3 and the “Fuzzy Topological Relation algorithm” presented in Figure 16. A formal definition is given here:

$$\begin{aligned}
& fuzzy_spatial_relation(STObj_1, STObj_2, T) := \\
& \left[\begin{array}{l} \mu \Rightarrow \{\mu_1, \dots, \mu_n\}, \\ rel_Type \Rightarrow \{disjoint, meet, inside, equal, contains, covers, coveredBy, overlap\}, \\ T \Rightarrow \{T_{begin}, \dots, T_{end}\} \end{array} \right] \\
& , where \quad T_{begin} \leq T \leq T_{end}, is_instance(STObj_1, STObject, T), \\
& is_instance(STObj_2, STObject, T)
\end{aligned} \tag{22}$$

3.2.2 Alloy notation

After the model is defined formally in C-logic, Alloy analyzer is used. Alloy is a formal object oriented specification language and its tool can be used for specifying properties about objects and validating them [21]. The model is first abstracted in Alloy language and then the analyzer verifies it.

In Alloy there are two kinds of specification elements:

- Signatures: define new types and contains a set of objects. The objects can be related by the relations, which are fields of the objects.
- Facts, functions, predicates: define constraints and true statements.

Next we define our model in Alloy as follows:

module systems/STModel

open alloy/models/util/ordering[LineSegment] as ordL

open alloy/models/util/ordering[Time] as ordT

open alloy/models/util/ordering[Fuzzy] as ordF

The *open* statements are used to access the predefined *ordering* module. Ordering module gets an argument and creates a linear ordering over it. The module presents some functions, such as which element is first in the ordering (*first*), or whether a

given element precedes another (*next*), and some predicates such as comparisons (*gte*, *lt*, *eq*, etc.) .

```
//abstract fuzzy class. The implementation consists of definition of fuzzy number
// which gives degree of fuzziness between 0 and 1
abstract sig Fuzzy{}
```

```
//The Time class includes defnition of time in YYYYMMDD hh:mm
sig Time{}
// a temporal class includes temporal class and a temporal entity has
// beginning time and end time
sig Temporal{
  beginTime,endTime:Time
}
```

```
// Beginnig time should be less than or equal to end time
fact TemporalFact{
  all T:Temporal\ ordT/le[T.beginTime,T.endTime]
}
```

```
//a temporal object exists in a temporal interval
pred isInstance(o:Temporal,bt,et:Time){
  ordT/gte[o.beginTime,bt] and ordT/le[o.endTime,et]
}
```

Alloy allows the definition of abstract classes. This is similar to object oriented abstract classes. The detailed definitions of *Fuzzy* and *Time* classes are left to implementation. The *Temporal* class definition which is extended by temporal classes is followed by a fact definition. *Facts* are constraints which are assumed to be always true. In the model, *TemporalFact* asserts that the *beginTime* should be less than or equal than the *endTime*. Next, we define a predicate which implies that a

temporal object should exist in a temporal interval. Predicates in Alloy are simply named constraints.

```
// Spatialbase has common entries for spatial classess  
// membership: is a fuzzy number and shows the degree of  
// spatial object's belonging to a particular spatial class  
// size: for fuzzy spatial object size is also fuzzy.  
abstract sig SpatialBase extends Temporal{  
membership:Fuzzy, //fuzzy membership  
size:Fuzzy  
}
```

```
// Coordinate defines an x, y location in the space.  
// x, y may be float numbers  
sig Coordinate{}
```

```
// Point is the basic spatial element and can be part of line segments.  
sig Point extends SpatialBase {  
location:Coordinate  
}
```

```
// a line segment is aggregated by a set of points,  
// It has a beginning and ending defined by points.  
sig LineSegment extends SpatialBase{  
sourceEnd:Point,  
targetEnd:Point  
}
```

```
// a region is aggregated by a set of line segments  
sig Region extends SpatialBase{  
linesegs:set LineSegment  
}
```

```

// at least 3 line segments form a region
fact RegionConst{
all r:Region | #r.linesegs >= 3
}

```

The *SpatialBase* is a super class of spatial classes such as *Point*, *LineSegment* and *Region*. Then some facts are defined which presents natural truths about geometry. For example a line segment has two points and at least three line segments form a region

```

// a geometry is formed by a set of points and/or linesegments
// and/or regions
sig Geometry extends Temporal{
points: set Point,
linesegs: set LineSegment,
regions: set Region
}

```

```

// a geometry should have at least one of the parts.
// not all of the parts can be empty sets.
// this fact does not allow empty geometry
fact GeometryFact{
all g:Geometry |
not (#g.points=0 and #g.linesegs=0 and #g.regions=0)
}

```

```

// if a geometry exists in some temporal interval so that
// its parts should exist in the same interval
fact GeometryConst{
all g:Geometry |
isInstance[g,g.beginTime,g.endTime] =>
( isInstance[g.points,g.beginTime,g.endTime] and

```



```

isInstance[g.lineSegs,g.beginTime,g.endTime] and
isInstance[g.regions,g.beginTime,g.endTime])
}

```

The GeometryConst fact states that if there exists a geometric entity in some temporal interval, so do all its parts in the same interval.

```

// A spatiotemporal object definition
sig STObject extends Fuzzy{
  geometry:some Geometry, // an STObject has one or more Geometry
  holes:set Geometry, // an STObject may have holes
  trajectory:some Point, //trajectory is a non-empty set of points
  spatialRelation:set RelationType // An STObject may have spatial
                                // relation(s) with other STObjects
}

```

The STObject has fields such that *geometry* and *holes* define the geometry of the spatiotemporal object. The *trajectory* show the path that object follows. The *spatialRelation* field holds the object's topological relation with other *STObjects*. These relations are formally defined in Chapter 2. An Alloy specification is given here:

```

// A spatial relation exists in some temporal interval
// including two STObjects and fuzzydegree that shows
// the degree of the relation
abstract sig RelationType extends Temporal{
  F,G: one STObject, //two STObjects F and G
  fuzzyDegree:Fuzzy //the degree of the relation which is fuzzy
}

```

```

// These are the possible types of spatial relations.
// Each one of them is a relation between two STObjects and have a degree

```

```

one sig Disjoint,Meet, Inside,Equal, Contains,Covers,CoveredBy,Overlap extends
RelationType{
rel:F->G->Fuzzy
}

```

This definition is an enumeration of RelationType class. Each relation is a triple from one object to another and to the *Fuzzy* class because the degree of relation can be fuzzy. The predicates below define these relations by giving some constraints about object geometries and/or holes.

// the following predicates give definitions for the spatial relations

```

pred disjointCR(R:RelationType){
no(R.F.geometry & R.G.geometry) and no(R.G.geometry & R.F.holes) and
no(R.F.geometry & R.G.holes) and no(R.F.holes & R.G.holes)
}

```

```

pred insideCR(R:RelationType){
some (R.F.geometry & R.G.geometry) and (no(R.F.geometry & R.G.holes) or
((R.G.holes in R.F.geometry) and (R.G.holes in R.F.holes)))
}

```

```

pred insideCR2(R:RelationType){
some (R.F.geometry & R.G.geometry) and (no(R.F.geometry & R.G.holes) or
((R.F.holes in R.G.geometry) and (R.F.holes in R.G.holes)))
}

```

```

pred meetCR(R:RelationType){
one (R.F.geometry & R.G.geometry) and not disjointCR[R] and
not insideCR[R] not insideCR2[R] and not equalCR[R]
}

```

```

pred containsCR(R:RelationType){
  insideCR2[R]
}

```

```

pred equalCR(R:RelationType){
  (R.F.geometry=R.G.geometry) and (R.F.holes=R.G.holes)
}

```

```

pred coversCR(R:RelationType){
  insideCR2[R] and meetCR[R] and #(F.geometry & G.geometry) = 1
}

```

```

pred coversCR2(R:RelationType){
  insideCR[R] and meetCR[R] and #(F.geometry & G.geometry) = 1
}

```

```

pred coveredbyCR(R:RelationType){
  coversCR2[R]
}

```

```

pred overlapCR(R:RelationType, fuz:Fuzzy){
  not (disjointCR[R] or meetCR[R] or insideCR[R] or containsCR[R] or
  equalCR[R] or coveredbyCR[R] or coversCR[R]) and ordF/gte[R.fuzzyDegree,fuz]
}

```

Some assertions and consistency checks are also necessary in order to show that the definitions are correct:

```

// Spatial relation asserts
assert disjointT{ //if disjoint not any other relation
all R:RelationType, fDegree:Fuzzy | disjointCR[R] =>
  not (overlapCR[R,fDegree] or meetCR[R] or insideCR[R]
  or containsCR[R] or equalCR[R] or coveredbyCR[R] or

```

```

        coversCR[R]) or
        R.G.geometry in R.F.holes //a geometry may be inside the hole
    }

check disjointT

assert meeT{
all R:RelationType, fDegree:Fuzzy | meetCR[R] =>
    not disjointCR[R] and not insideCR[R] and      not equalCR[R] and
    not coversCR[R] and not overlapCR[R,fDegree]
}

check meeT

assert insidE{
all R:RelationType, fDegree:Fuzzy | insideCR[R] =>
    not (disjointCR[R] or meetCR[R] or coversCR[R] or
    overlapCR[R,fDegree]) or (R.F.geometry in R.G.holes)
}

check insidE

assert coverS{
all R:RelationType, fDegree:Fuzzy | coversCR[R] =>
    not (disjointCR[R] or overlapCR[R,fDegree]) and
    insideCR2[R] and (G.holes in F.geometry)
}

check coverS

assert equalL{
all R:RelationType, fDegree:Fuzzy | equalCR[R] =>

```

```

    not (disjointCR[R] or overlapCR[R,fDegree]) and
    (R.G.geometry in R.F.geometry) and (R.F.holes in R.G.holes)
}

```

check equal

The full list of spatial assertions together with complete model in Alloy can be found in Appendix B. So far the generic classes and related constraints, facts and assertions are defined. The application specific classes extend the generic classes.

```

//application specific classes
//-----
// City may have some routes crossing and have some weather object
sig City extends STObject{
  route:set Route,
  weather:some MetObject
}

sig Route{
  parts: some LineSegment, //route has at least one LineSegment or more

  //route may be one type or a mixed type. e.g. maritimeRoute or
  // territoriolRoute + MaritimeRoute
  rType:some RouteType,

  // some parts may be clear some parts may be restricted so it has at least
  // one status but may have more than one
  rStatus:some RouteStatus,

  // route crosses at least one City
  cities:some City,

```

```

// a set of vehicles use the route
vehicles:set Vehicle
}

//two consecutive line segments over a route should have one common point
//one's targetEnd equals other's sourceEnd
assert routeFact{
  all R:Route, ls1,ls2:R.parts|
    (ls1!=ls2 and ordL/eq[ordL/next[ls1],ls2])=>ls1.targetEnd= ls2.sourceEnd
}
check routeFact

// route type can ben maritime, territorial or aerial
abstract sig RouteType{}
one sig MaritimeRoute,TerritorialRoute, AerialRoute extends RouteType{}

// route may be clear, wavy (for maritime route) or restricted (for all types)
abstract sig RouteStatus{}
one sig Clear,Wavy,Restricted extends RouteStatus{}

sig Vehicle{
  type:VehicleType,
  status:VStatusType,
  route:set Route
}

//route-vehicle relation at specific time
sig Journey extends Temporal{
  route: Route,
  vehicle: Vehicle,
}

```

```

// 1-a journey has only one vehicle and one route
// 2-A vehicle is used on only one journey during the journey
assert JourneyFact{
  all j1,j2:Journey, t:Temporal\
    isInstance[t,j1.beginTime,j1.endTime] and
    isInstance[t,j2.beginTime,j2.endTime]=>
    one j1.vehicle and one j1.route and one j2.vehicle and one j2.route and
    j1.vehicle != j2.vehicle

}

check JourneyFact

sig Voyage extends Journey{} //ship journey
abstract sig VehicleType{}
one sig Ship, Bus, Train, Plane extends VehicleType{}

// a voyage has a MaritimeRoute and the vehicle running should be Ship
fact voyageFact{
  all vyg:Voyage\
    vyg.route.rType=MaritimeRoute and vyg.vehicle.type=Ship
}

// A vehicle may be on time, delayed or canceled
abstract sig VStatusType{}
one sig OnTime,Delayed,Canceled extends VStatusType{}

//if a journey's route is restricted vehicle is delayed or canceled
fact statusFact{
  all j:Journey\
    j.route.rStatus=Restricted=>
    j.vehicle.status=Delayed or j.vehicle.status=Canceled
}

```

```

// a meteorological object is a spatiotemporal object
sig MetObject extends STObject{
  object:MeteorType,
  degree:MeteorObjectDegree
}

// the types of meteorological objects are enumerated here
abstract sig MeteorType{}
one sig Temperature, Pressure, Visibility, Wind, Wave, Cloude, Precipitation
extends MeteorType{}

// The strength of Meteorological object i
abstract sig MeteorObjectDegree extends Fuzzy{}

// Here a sample is given for visibility
one sig Visible, Misty, Foggy extends MeteorObjectDegree{}

// Another sample for precipitation
one sig Drizzle, Rainy, Snowy, Thunderstorm extends MeteorObjectDegree{}

// a meteorological measurement in a City includes a number of meteorological
objects
sig Measurement extends Temporal{
  metobj:some MetObject,
  city:lone City
}

// two spatiotemporal objects have overlap degrees
abstract sig OverlapDegree extends Fuzzy{}

//fuzzy overlapdegrees are enumerated
one sig Less, Moderate,High extends OverlapDegree{}

```



```

//if city and meteorological object overlaps than the object is in the city's weather
fact weatherFact{
  some M:MetObject, C:City, R:Overlap|
    (R.F=M and R.G=C and overlapCR[R,High]) => M in C.weather
}

// As an example : if a city's weather has visibility and precipitation and
// their degrees are strong enough, then the route is restricted
fact routeStatusFact{
  some M:MetObject, C:City, route:Route|
    (
      (M.object=Visibility and M.degree=Foggy) or
      (M.object =Precipitation and (M.degree=Snowy or
      M.degree=Thunderstorm)) and
      M in C.weather and C in route.cities)=>route.rStatus=Restricted
}

```

We finally run all check commands in Alloy and have no inconsistency:

Executing "Check disjoint"

*Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
 0 vars. 0 primary vars. 0 clauses. 140ms.
 No counterexample found. Assertion may be valid. 0ms.*

Executing "Check meeT"

*Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
 0 vars. 0 primary vars. 0 clauses. 31ms.
 No counterexample found. Assertion may be valid. 0ms.*

Executing "Check inside"

*Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
 0 vars. 0 primary vars. 0 clauses. 31ms.
 No counterexample found. Assertion may be valid. 0ms.*

Executing "Check coverS"

Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20

0 vars. 0 primary vars. 0 clauses. 16ms.

No counterexample found. Assertion may be valid. 0ms.

Executing "Check equal"

Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20

0 vars. 0 primary vars. 0 clauses. 15ms.

No counterexample found. Assertion may be valid. 0ms.

Executing "Check routeFact"

Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20

0 vars. 0 primary vars. 0 clauses. 16ms.

No counterexample found. Assertion may be valid. 0ms.

Executing "Check JourneyFact"

Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20

0 vars. 0 primary vars. 0 clauses. 16ms.

No counterexample found. Assertion may be valid. 0ms.

7 commands were executed. The results are:

#1: No counterexample found. disjoint may be valid.

#2: No counterexample found. meeT may be valid.

#3: No counterexample found. insidE may be valid.

#4: No counterexample found. coverS may be valid.

#5: No counterexample found. equal may be valid.

#6: No counterexample found. routeFact may be valid.

#7: No counterexample found. JourneyFact may be valid.

These definitions and consistency checks in Alloy show that the model is sound and can be transformed into a programming language. In the next section we present the model in a UML notation.

3.3 The Object Model

In Section 3.1 the object model is formalized using C-logic and verified using Alloy. In this section this modeling approaches are visualized using UML. The object model in Figure 9 consists of the objects and relationships between objects for the generic part of our model. The second part in Figure 10 consists of meteorological application specific objects and relations.

The “Temporal” class is extended by classes having temporal data. The temporal class is made up of the “Time” class including beginning/end times and the type of the time record (*valid* or *transaction*). The temporal dependency of a class is shown by “*T*” on the upper right-hand side of the entities and the spatiotemporal dependency by “*ST*”. This is one of the extensions to UML that we used for the specific requirements of the fuzzy spatiotemporal application [58].

The *Fuzzy* class is an abstract class and provides range definitions, relevance values and class-object membership values for other inheriting classes. The fuzzy constructor, indicated by the tag *U* to the left-hand side of the name of the class, is used to indicate the existence of class attributes having fuzzy values, such as the degree of a spatial relation.

The *SpatialBase* is a super class for spatial classes. The *membership* attribute in spatial classes stores a membership value to describe a proximity to a certain fixed space. So, spatial objects may belong to a class fully (i.e. with a degree of 1) or partially (i.e. with a membership degree between 0 and 1). As an extension to UML, a fuzzy class constructor, indicated by a double-square placed on the upper-left hand side of the spatial class, explicitly represents the fuzzy instances. The spatial classes (*Point*, *LineSegment* and *Region*) have an aggregation relation in between which is shown by a diamond symbol. The *Point* class is defined with a *Coordinate* which includes x and y values in R^2 and z as a third dimension represents the altitude value.

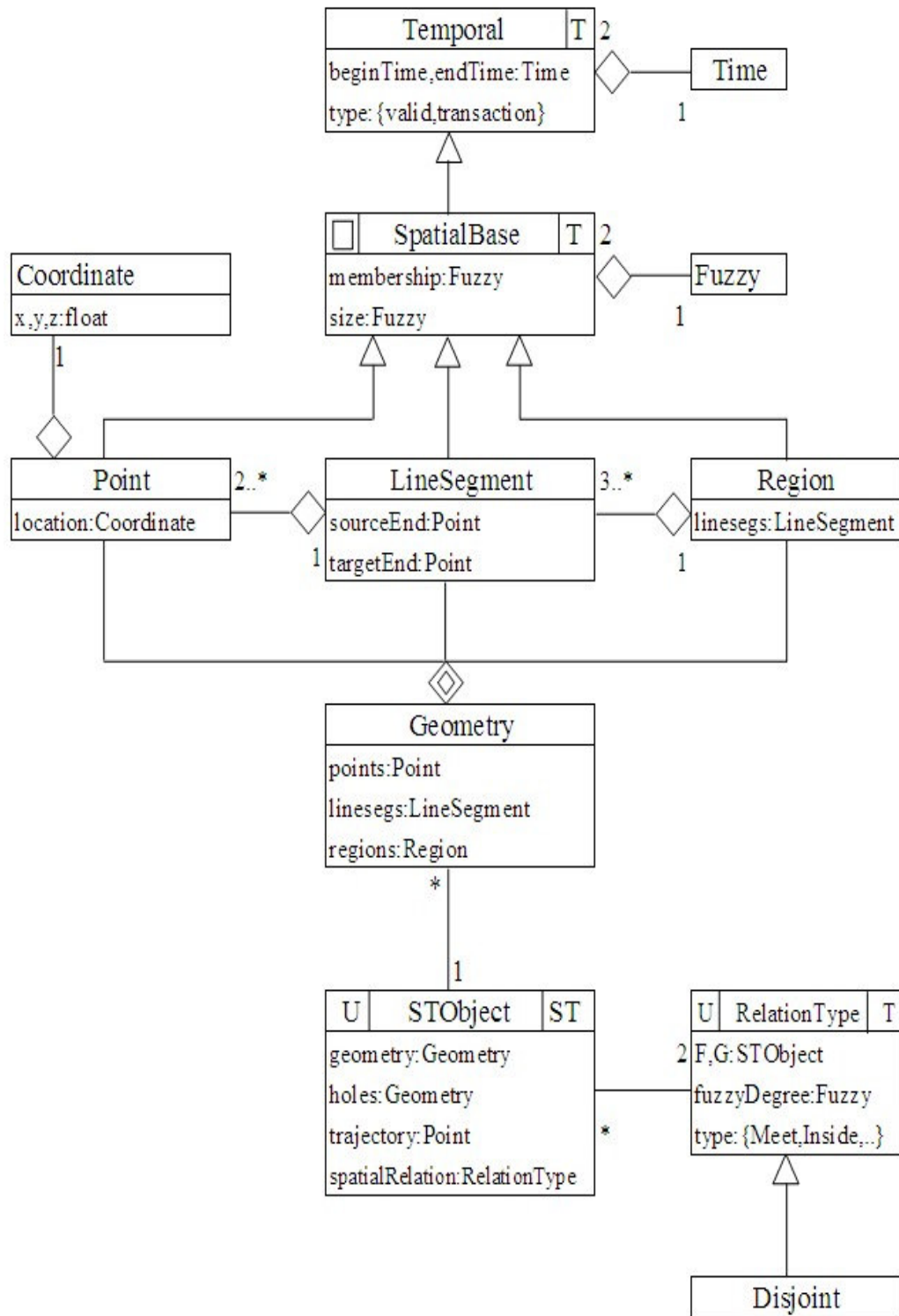


Figure 9: A fuzzy spatiotemporal model

Point, LineSegment and Region are parts of *Geometry* and a *whole-part* relation exists with Geometry, in which the whole is aggregated by different kinds of parts. The whole/part relation is indicated by a double diamond symbol. The STObject can have *some* geometry with a *set* of holes, follow a trajectory and have a *set* of relations with other STObjects. A spatial relation between two STObjects can be enumerated by eight topological relations (i.e. Disjoint, Meet, Inside ... etc.) defined in Section 2.3.2.

Under the generic model, meteorological application classes (e.g. *MetObject*, *Route*, *Measurement*, *Vehicle* and *City*) exist inheriting the STObject and Temporal and LineSegment classes. The model is shown in Figure 10.

According to application model the classes and some relations or constraints between them are defined. For example, a *MetObject* is aggregated by numerous *Measurement* observed by meteorological stations in the cities. A *City* may be on the way of some *Routes* used by *Vehicles*. A *Route* is formed by consecutive line segments. There may be different route types such as *Maritime*, *Territorial*, *Aerial*, etc. A route crossing the multiple cities may be also used by multiple vehicles. A vehicle using a route in some temporal interval forms a *Journey*. A journey refers to a route which may be a real one like a river, a railway, or a virtual route like the route of a ferry (a *Voyage*) or a plane.

This modeling approach should satisfy most practical requirements for spatiotemporal applications. For example, the changes of spatial and temporal attributes are captured by collecting all the related *Geometry* and *Temporal* objects. The relations between the objects are calculated dynamically.

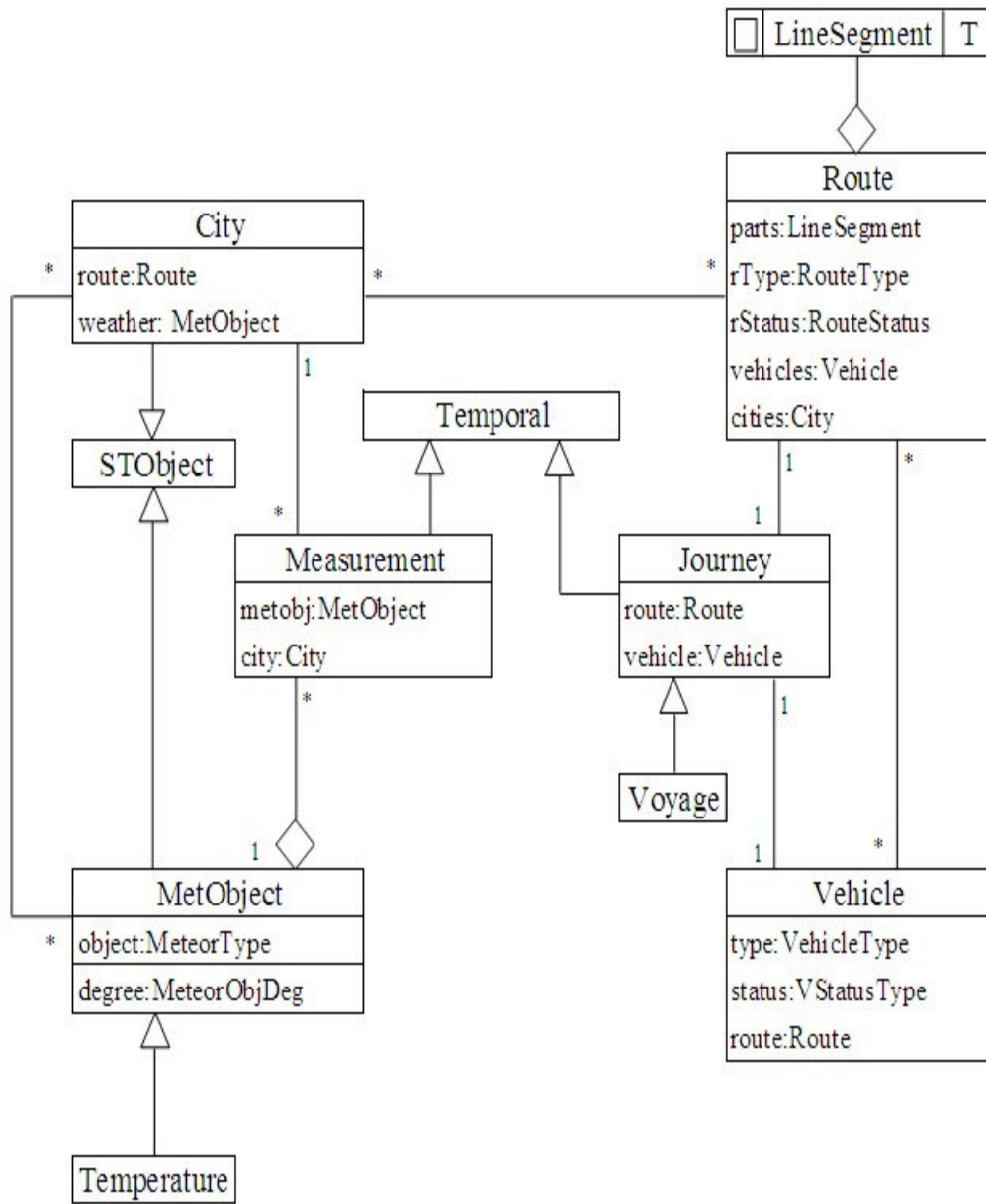


Figure 10: Meteorological application model

The generic model and the object model using are finalized with example class definitions in Java. We show a few typical class definitions since the other classes can be created similarly.

```

public class Geometry{
    Point p; //point reference
    LineSegment l; //line reference
    Region r; //polygon reference
    Temporal temporal; //temporal reference
    Fuzzy size;
}

public abstract class STObject extends Fuzzy{
    Geometry gset []; //geometries
    Geometry hset []; //holes
    int ngset, nhset; //number of geometries and holes
    Point tset []; //trajectory
}

public class City extends STObject{
    String name;
    int population;
    HashSet<Measurement> measurements; //a set of measurements
    Route route;
}

```

The *Geometry* class contains the references for the parts of the geometry (i.e. Point, LineSegment and Region). The *geometries*, *holes*, *position* and *time* are some fields of the *STObject*. Finally, the *City* definition which is a spatiotemporal object is presented. The *HashSet<Measurement>* represents a set of elements (unordered and not duplicated) of type *Measurement*.

3.4 Coupling the Fuzzy Database with a Fuzzy Knowledge Base

In order to achieve an intelligent application, a knowledge base (KB) is integrated to the object-oriented database. We utilize the *Intelligent Fuzzy Object Oriented Database (IFOOD)* [25], which provides flexible and powerful querying mechanisms for complex data and knowledge with uncertainty in both database and knowledge base.

The knowledge base (KB) used in the IFOOD architecture includes rules and intelligent objects having fuzzy attributes. In addition, it features a fuzzy inference method used for deduction of fuzzy conclusions. It gets the rules and facts/objects as input, tries to satisfy rules by comparing them with facts, and produces a conclusion from the satisfied rules.

The IFOOD language is an object-oriented database language extended with declarative rules to define predicates. We illustrate this with an example: sea traffic is prohibited in the Istanbul Strait due to conditions of wind, visibility, and waves, etc. In the knowledge base, the combination of rules and the objects attributes fire the *maritime lines are prohibited* conclusion. The fuzzy rules are defined using linguistic values as follows:

if city.visibility is *badsight* **or** underAverage
and city.wind is *windy* or *gust*
then city.route is *restricted*.

The rule given below exemplifies the fuzzy *if-then* rules utilized in the IFOOD language formally.

```
defrule X.Status([prohibited],Y,Z,threshold) > Route(X), City(Y),  
Y.visibility([badsight,underAverage],threshold), Y.wind([windy,gust],threshold),
```


where $o.a(v, [threshold_{rng}(a)])$ is an object term, where o is an object ID , a is an object attribute, v is an attribute value, and $threshold_{rng}(a)$ is the threshold level defined for the attribute a . For more details of the IFOOD inference engine and language the reader is referred to [25].

CHAPTER 4

THE ARCHITECTURE OF THE SPATIOTEMPORAL DATABASE APPLICATION

The architecture of the proposed environment for spatiotemporal data modeling is illustrated in Figure 11. The FOOD system acts as a database server for data management and the FKB system acts as a knowledge server for knowledge management. Additionally, the fuzzy spatial predicates are determined by the fuzzy spatial processor (FSP). The communication and interaction between the database system, the knowledge base system and the fuzzy spatial processor is performed by the bridge interface (BI). At the higher level, there is a single user interface that provides a unified environment for both data and knowledge management and allows users the capability of query processing independently from the physical structure of the architecture.

Fuzzy processors are used to handle uncertainty at both the object-oriented database component and the knowledge base component of the system. At the user interface level, users are able to define objects and rules having uncertain properties and to query the system with uncertain conditions. The definitions of uncertain types, similarity relations, and membership functions are stored in the object-oriented database.

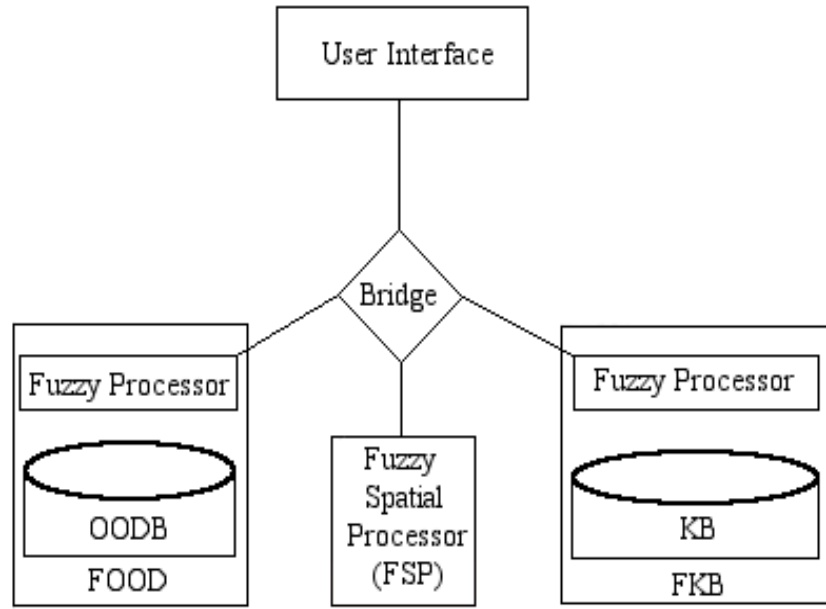


Figure 11: The architecture of the spatiotemporal database application

The *FKB* system processes rules taking fuzzy objects as input. We provide the required facilities in the FKB system to access the definitions in the FOOD system. For example, if the FKB system needs the similarity of two fuzzy terms of a special domain, it gets this value via the fuzzy processor from the FOOD system.

The *FSP* module processes topological predicates between complex spatial objects possibly with holes and fuzzy spatial objects. BI forwards the user request to FKB if the query includes a topological predicate. FSP requests the spatial objects from FOOD and finds the predicates and the degree of membership of the relation.

The *BI* component plays a coordinating role in query processing. It gets user queries, analyzes them, sends requests to the database and/or to the knowledge base, retrieves the results, and sends them up to the user interface. The algorithm, with implementation steps, is as follows:

Query Evaluation Algorithm:

Input: Query supplied by the user

Output: Retrieved objects

Get and Parse(query);

if query is nonspatial-query **then**

if query is crisp-query **then**

 Send-to-OODB(crisp-query);

else

 Send-to-OODB(fuzzy-query);

end if

if query includes knowledge-base predicate(s) **then**

 Transfer-to-knowledge-base(satisfying-objects);

 Start-inference-engine-evaluation;

 Return(result);

end if

 Get(satisfying-objects);

else

 Send-to-OODB(spatial-query);

if query includes knowledge-base predicate(s)(rule) **then**

 Transfer-to-knowledge-base(satisfying-objects);

 Transfer-to-FSP(satisfying-objects);

 Apply fuzzy spatial and/or complex spatial algorithm;

 Start-inference-engine-evaluation;

 Return(result);

else

 Transfer-to-FSP (satisfying-objects);

 Apply fuzzy spatial algorithm in Figure 16 and/or complex spatial
 algorithm in Figure 14;

end if

end if

Submit-to-user (selected satisfying-objects);

Figure 12: Query evaluation algorithm

CHAPTER 5

QUERY PROCESSING

In this chapter query processing mechanism is tested with various types of queries. The following procedures are applied to resolve the query according to its type:

- *The basic query (crisp and non-spatial)*: This type of query asks for crisp data that does not have a spatial dimension. The BI sends the parsed query expression directly to OODB. The objects that meet the query condition are sent back to the BI.
- *The fuzzy non-spatial query*: This type of query asks for data that is fuzzy but non-spatial and the BI, FKB, and OODB components are employed. The objects retrieved by the BI are sent to the FKB component to check whether they meet the fuzzy conditions. How these objects are checked is illustrated in Section 5.1. Objects satisfying the conditions are sent back to the BI.
- *The complex spatial query*: Complex spatial objects and their relationships are queried in this type of query. The BI, OODB and the FSP components are employed to fetch query results. The user asks for the objects that have topological relations (described in Section 2.2 and 2.4) with the objects under inquiry. Section 5.2 illustrates this type of query.
- *The fuzzy spatiotemporal query*: In this type query, the user asks for the objects that meet the conditions of the predefined rules within a specified time interval. The rules can be evaluated by an examination of topological relations between fuzzy regions and fuzzy objects. The fuzzy spatiotemporal queries are illustrated in Section 5.3-5.5.

5.1 Fuzzy Non-Spatial Query

The objects used in the example are listed in Table 2, and the similarity relation of “*cloud*” is included in Table 2 and Table 3. The similarity relation of “*temperature*” is already presented in Section 3.1.

Table 2: Sample records in database

ID	Object	Name	Temperature	Cloudiness	Visibility	DateTime
C1	City	Istanbul	Cool	Cloudy	Bad sight	01.01.2008
C2	City	Edirne	Moderate	Partly cloudy	Under average	01.01.2008
C3	City	Izmit	Cold	Cloudy,closed	Average	01.01.2008

Table 3: Similarity matrix of cloudiness attribute

Cloud	Clear	Partly	Cloudy	Closed
Clear	1.0	0.6	0	0
Partly cloudy	0.6	1.0	0.6	0.4
Cloudy	0	0.6	1.0	0.8
Closed	0	0.4	0.8	1.0

Query: Retrieve the cool and partly cloudy cities on 01.01.2008.

This query is formulated as follows:

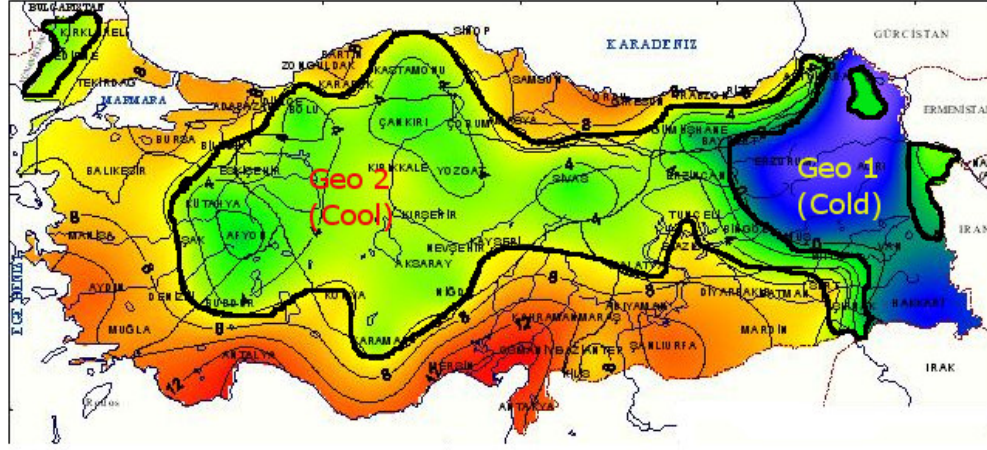
```
select X.cityname
from city(X)
where X.temperature([cool],0.6) and X.cloud([cloudy],0.8),
X.validtime(01.01.2008);
```

The query is evaluated as follows:

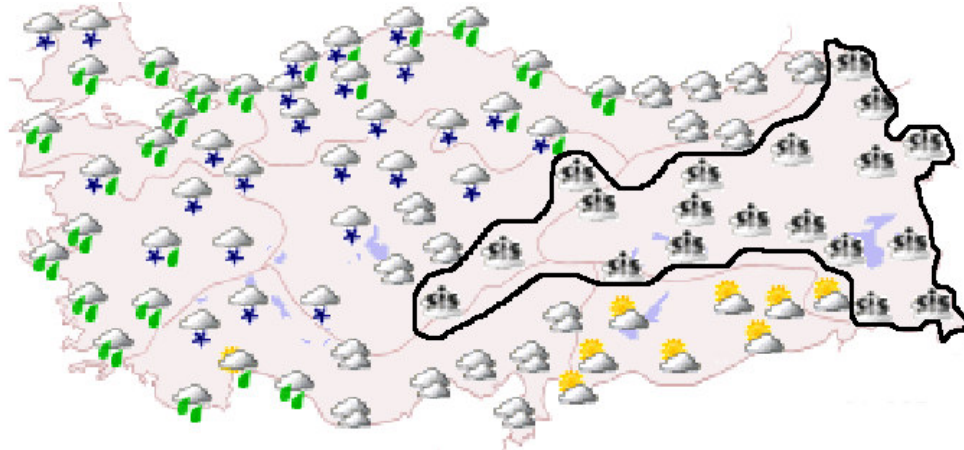
- i. The first predicate to evaluate in this query is $X.temperature([cool],0.6)$.
 - C1.temperature is cool, and $\mu_{Similarity}(cool,cool)=1.0$. Therefore C1 satisfies the temperature predicate.
 - C2.temperature is moderate, and $\mu_{Similarity}(cool,moderate)=0.6$. Therefore C2 satisfies it.
 - C3.temperature is cold, and $\mu_{Similarity}(cool,cold)=0.8$. Therefore C3 satisfies it.
- ii. Then, the predicate $X.cloud([cloudy],0.8)$ is evaluated.
 - C1.cloud is cloudy, and $\mu_{Similarity}(cloudy,cloudy)=1.0$. Therefore C1 satisfies the cloud predicate.
 - C2.cloud is partly cloudy, and $\mu_{Similarity}(cloudy,partly\ cloudy)=0.6$. Therefore C2 does not satisfy it.
 - C3.cloud is cloudy or closed with $max\{\mu_{Similarity}(cloudy,cloudy),\mu_{Similarity}(cloudy,closed)\}=max\{1.0,0.8\}=1.0$. Therefore C3 satisfies it.
- iii. As a result, the objects C1 and C3 satisfy the fuzzy query conditions

5.2 Complex Spatial Query

Figure 13 shows the maximum temperature regions (a) and the meteorological events (b) as mapped by the Turkish Meteorological Office on 01.01.2008.



(a)



(b)

Figure 13: Maximum temperature regions (a) and meteorological events (b) on 01.01.2008.

The temperature regions are shown in different colors (e.g. cold parts by dark blue, cool parts by green, moderate parts by orange and warm parts by red). Temperature regions are visualized as complex spatial objects since they have multiple components possibly with holes. The expected meteorological events are depicted

with symbols and colors, e.g. rain (green drops), snow (blue stars), grey clouds, black foggy areas and yellow patchy areas.

The spatial objects representing temperature regions and meteorological objects in Figure 13 are inserted in the database as shown in Table 4. The temperature regions, which are classified by their degrees (e.g. cool, cold, etc.), have different geometries (e.g. Geo₁, Geo₂, etc.). According to the figure, Cold (dark blue) region has one simple region (Geo₁) and a hole (Hole₁). The cool regions (green) have four simple regions forming Geo₂, and none of them has a hole.

Table 4: Objects in the FOOD

Object	ObjType	Degree	Geometries	Holes	Valid Time
Met Object	temperature	{ <u>cold</u> , <i>cool</i> , moderate, warm}	{ <u>Geo₁</u> , Geo ₂ , Geo ₃ , Geo ₄ }	{ <u>Hole₁</u> , Null, Null, Null}	01.01.2008
Met Object	fog	{foggy}	{Geo ₅ }	{Null}	01.01.2008
Met Object	snow	{heavy, rainy}	{Geo ₆ , Geo ₇ }	{Null, Null}	01.01.2008
Met Object	rain	{shower}	{Geo ₈ }	{Null}	01.01.2008
Met Object	cloud	{cloudy, partly cloudy}	{Geo ₉ , Geo ₁₀ }	{Null, Null}	01.01.2008

Query: Retrieve the cold and foggy regions and the relation on 01.01.2008.

This query is formulated as follows:

```
select spatial_relation(X.geometry,Y.geometry)
from MetObject(X), MetObject(Y)
where X.ObjType([temperature]) and Y.ObjType([fog]) and
X.degree([cold],0.8) and Y.degree([foggy],0.8)
and X.validtime(01.01.08) and Y.validtime(01.01.08);
```

In this query, the temperature objects having the attribute value *cold*, and the fog objects having the *foggy* degree are fetched from FOOD to BI. The user supplies a threshold value 0.8 for temperature degree, so “cool” regions are also fetched since $\mu_s([cold], [cold]) = 1.0$ and $\mu_s([cold], [cool]) = 0.8$. The simple topological relation algorithm is applied for components with holes of complex regions. After finding simple topological predicates, the complex topological relation algorithm is applied to determine the final topological predicate. Note that we show only the “disjoint” case in the algorithm in Figure 14 since it occupies much space and the other cases are handled similarly, as explained in Section 2.4:

Complex Topological Relation algorithm:

Input: Simple regions of two complex regions

Output: Sequence of topological predicates that hold between each pair of simple regions

1. $STR \leftarrow \emptyset$ //Simple Topological Relation
2. **for** each simple region of complex regions(F, G)
 - $F_0 \leftarrow \{\text{Base geometry of } F\}$
 - $G_0 \leftarrow \{\text{Base geometry of } G\}$

Figure 14: Complex topological predicate evaluation algorithm

```

 $S \leftarrow \emptyset$  //S is a set of Simple topological relations
if spatial_relation( $F_0, G_0$ ) = "disjoint" then
    simple_topological_relation( $STR$ )  $\leftarrow$  "disjoint";
else
    for  $i \leftarrow 1$  to  $n$  //For each hole of F
        if spatial_relation( $G_0, F_i$ ) = "inside" then
            simple_topological_relation( $STR$ )  $\leftarrow$  "disjoint";
        end if
    end for
    for  $j \leftarrow 1$  to  $m$  //For each hole of G
        if spatial_relation( $F_0, G_j$ ) = "inside" then
            simple_topological_relation( $STR$ )  $\leftarrow$  "disjoint";
        end if
    end for
end if
    // If not disjoint do related calculations for other topological predicates
     $S \leftarrow S \cup \{STR\}$ ;
end for

3. for each  $STR$  in  $S$ 
    if all  $STR$  are pairwise disjoint then
        complex_topological_relation( $CTR$ )  $\leftarrow$  "Disjoint";
    end if
    // If not disjoint do similar calculations for other complex topological
    predicates
    
$$CTR \leftarrow \left\{ \begin{array}{l} Disjoint \mid Meet \mid Inside \mid Equal \mid \\ Covers \mid Coveredby \mid Contains \mid Overlap \end{array} \right\}$$

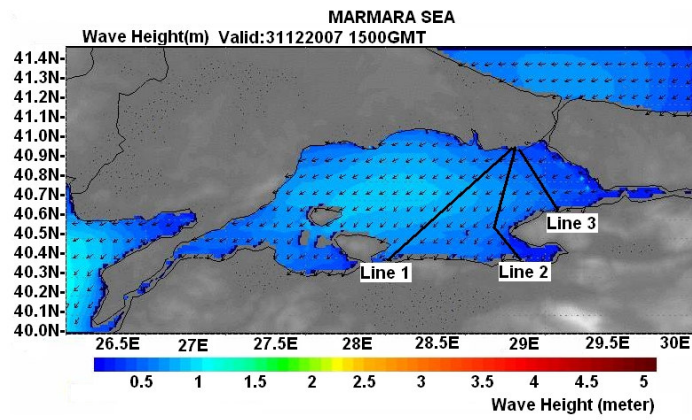
end for

```

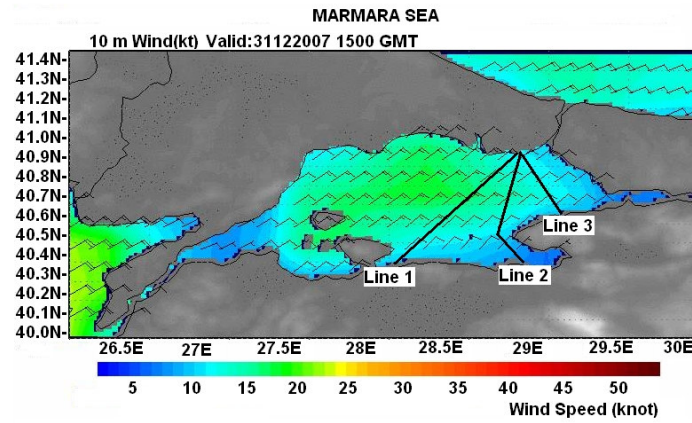
Figure 14: Complex topological predicate evaluation algorithm (cont'd)

5.3 Fuzzy Spatiotemporal Query

In this example, fuzzy spatial relations are queried. In Figure 15, wave height (a) and wind speed (b) for “Marmara Sea” are illustrated on 31.12.2007 15:00 Greenwich Mean Time (GMT) (between 40.0-41.4 North latitudes and 26-30 East longitudes).



(a)



(b)

Figure 15: Wave height (a) and wind speed (b) over Marmara Sea.

Both meteorological events are represented as fuzzy spatial objects, in which the central parts have the highest waves and strongest winds while the coastal areas have lower waves and calmer wind conditions. The three lines, Line₁, Line₂ and Line₃ represent ferry routes between the ports.

Query: Retrieve the sea lines restricted for transportation due to wind and wave conditions on 31.12. 2007.

This query is formulated as follows:

```
select X
from Geo_line(X), MetObject(Y), MetObject(Z)
where X.LineType([SeaLine]) and Y.ObjType([Wave]) and
Z.ObjType([Wind]) and X.status([restricted], Y, Z, threshold_value),
X.validtime (31.12.2007);
```

In the query, the sea lines restricted for transportation are requested. In this case, *X.status([restricted],Y,Z,threshold)* is a rule defined in the FKB as follows:

```
defrule X.Status([restricted],Y,Z,threshold) >
Geo_Line(X), MetObject(Y), MetObject(Z),
overlap(X.geometry,Y.geometry,threshold),
overlap(X.geometry,Z.geometry,threshold);
```

The threshold value supplied by the user gives a limit for the restriction of the sea line. Required objects (sea wind and wave height geometries) are fetched from the OODB, and FSP calculates the fuzzy spatial relation (overlap in this case) between the *fuzzy regions* wind and wave, and *crisp* ferry lines using the *fuzzy topological relation algorithm* in Figure 16:

Fuzzy Topological Relation algorithm (region vs. line):

Input: Two fuzzy object geometries

Output: The overlap degree of two objects

1. $FuzzyRelation \leftarrow 0$
2. **for** each α – cut level region $R_{\alpha i} - R_{\alpha i+1}$
 - if** the line overlaps with $R_{\alpha i} - R_{\alpha i+1}$ **then**
 $FuzzyRelation \leftarrow FuzzyRelation + m(region_{\alpha i}) \times m(line)$
 - end if**
3. **Return** $FuzzyRelation$

Figure 16: Fuzzy topological predicate evaluation algorithm

According to the meteorological forecast, the sea area is divided into five α – cut levels ($i=5$) and the ferry lines overlap some of them (see Figure 15); the calculation details are presented in Table 5 and Table 6.

The results of the fuzzy spatial relation calculations are supplied to FKB for inference. In FKB, a rule may be composed of more than one condition. Each condition in a rule may have its own matching degree. Therefore, we compute an overall matching degree. Here, we use the “min” operator for combining the degree of matching of conjunction (AND) conditions and the “max” operator for combining the degree of matching of disjunction (OR) conditions [47].

For example, considering the rule given for “restricted sea line” above, each term is matched with a matching degree, as shown in Table 5 and Table 6, and the overall

matching degree is calculated as in Table 7. According to the overall restriction degrees in the third column, given the threshold value 0.7, “Line₁” and “Line₂” will be restricted.

Table 5: Computing a fuzzy topological relation for a wavy region and ferry lines

α – cut levels of wavy region	$\tau_{overlap}$	$m(region_{\alpha_i}) \times$ $m(line_1)$	$\tau_{overlap}$	$m(region_{\alpha_i}) \times$ $m(line_2)$	$\tau_{overlap}$	$m(region_{\alpha_i}) \times$ $m(line_3)$
1.0 - 0.75	1	0.25	0	0.00	0	0.00
0.75 - 0.50	1	0.25	1	0.25	0	0.00
0.50 - 0.30	1	0.20	1	0.20	0	0.00
0.30 - 0.0	1	0.30	1	0.30	1	0.30
$\tau_{overlap}(R, L)$		1.0		0.75		0.30

Table 6: Computing a fuzzy topological relation for a windy region and ferry lines

α – cut levels of windy region	$\tau_{overlap}$	$m(region_{\alpha_i}) \times$ $m(line_1)$	$\tau_{overlap}$	$m(region_{\alpha_i}) \times$ $m(line_2)$	$\tau_{overlap}$	$m(region_{\alpha_i}) \times$ $m(line_3)$
1.0 - 0.65	1	0.35	1	0.35	0	0.00
0.65 - 0.30	1	0.35	1	0.35	0	0.00
0.30 - 0.20	1	0.10	1	0.10	1	0.10
0.20 - 0.0	0	0.00	1	0.20	1	0.20
$\tau_{overlap}(R, L)$		0.80		1.00		0.30

Table 7: The overall fuzzy relation degrees

	Degree of overlap of lines with wavy	Degree of overlap of lines with windy	$\mu_{\text{overall}} = \text{Min}(\text{overlap}_{\text{wavy}}, \text{overlap}_{\text{windy}})$
Line 1	1.00	0.80	0.80
Line 2	0.75	1.00	0.75
Line 3	0.30	0.30	0.30

5.4 Nested Rule Query

Due to the restrictions in the transportation lines, the vehicles' trips are "cancelled" or "delayed". Following the example in Section 5.3, we find the "delayed" vehicles for the same date.

***Query:** Retrieve delayed ferries on 31.12.2007.*

This query is formulated as follows:

```
select X
from Vehicle(X), MetObject(Y), MetObject(Z), Geo_Line(L)
where X.Type([Ferry]) and X.GetLine()=L and
X.status([delayed],Y,Z,L,threshold_value) and X.validtime (31.12.2007);
```

The rule is defined in FKB as follows:

```
defrule X.Status([delayed],Y,Z,L,threshold) > Vehicle(X), MetObject(Y),
MetObject(Z), Geo_Line(L), L.Status([restricted],Y,Z,threshold);
```

This rule is a nested rule as it fires another rule (*Y.status[restricted],threshold*), as described in Section 5.3.

5.5 Fuzzy Spatiotemporal Query

In Figure 17, the meteorological objects for an interval are presented. According to the figures the rainy areas (green drops) move to the east while decreasing in effective size. On the other hand the cloudy areas move to the west while increasing in effective size.

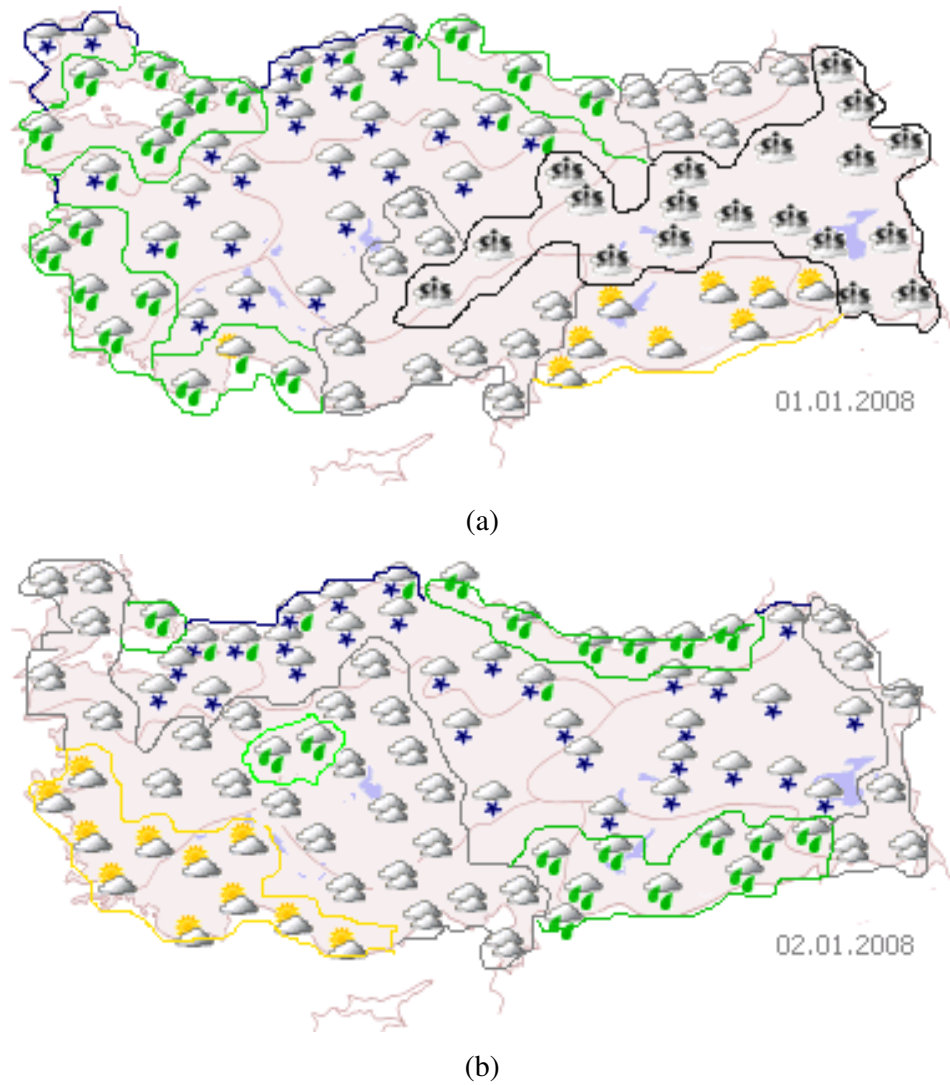


Figure 17: Meteorological objects on 01.01.2008 (a) and 02.01.2008 (b)

Query: Retrieve the area, direction and speed changes of meteorological events in terms of position and effect area between 01.01.2008 and 02.01.2008 .

In the query, the directional, positional and areal changes of the objects are queried and the algorithm in Figure 18 is applied.

Fuzzy Spatiotemporal Query algorithm:

Input: A pair of geometries for a time interval

Output: Area, direction and speed change

1. **for** each chronon_i (i=0 to n)
 - for** each object in the chronon
 - TotalArea_i \leftarrow 0
 - for** each component of the object's geometry
 - TotalArea_i \leftarrow TotalArea_i + component's area (CA)
 - end for**
 - for** each component of the object's geometry
 - Get the minimum bounding rectangle(MBR)
 - Get the center (x,y) of the MBR
 - Object center (X,Y)_i \leftarrow Weighted average of (x,y)'s
 - end for**
 - end for**
2. **for** each pair of center points (X,Y)_i
 - Distance_i \leftarrow Sqrt((Y_i-Y_{i-1})² + (X_i-X_{i-1})²)
 - Speed_i \leftarrow Distance_i / (choronon_i - choronon_{i-1})
 - Direction_i \leftarrow (Y_i-Y_{i-1}) / (X_i-X_{i-1})
 - AreaChange_i \leftarrow TotalArea_i - TotalArea_{i-1}
 - end for**

Figure 18: The algorithm to evaluate area, speed and direction change

In order to illustrate how the algorithm works, the rain object's spatial attribute changes are shown in Figure 19. This complex object is made of three parts on the first day (on the left) and four parts on the second day (on the right). The MBRs, central points for each part and the areas are depicted in the figure. A central point for the whole object on both days is calculated, using a weighted average considering the proportional area of each part:

$$CenterX = \sum_{i=1}^n A_i \times x_i$$

$$CenterY = \sum_{i=1}^n A_i \times y_i, \text{ where} \quad (23)$$

$$0 < A_i \leq 1 \text{ and } \sum_{i=1}^n A_i = 1 \text{ and } A_i = Area_i / Total \text{ area}$$

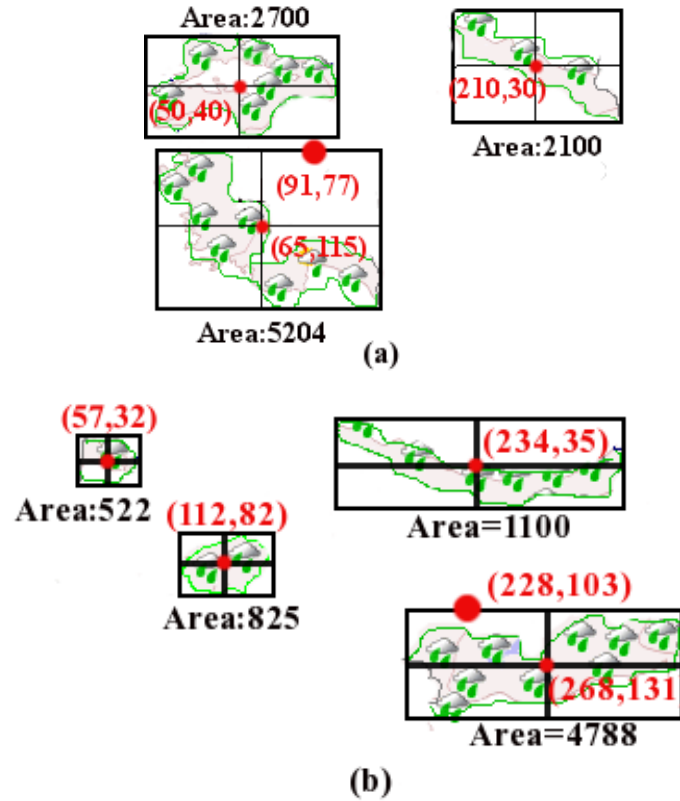


Figure 19: Rain object movement on 01.01.2008 (a) and 02.01.2008 (b)

The approximated centre points for the complex object (e.g. at coordinates of (91, 77) and (228,103)) are used to calculate the directional change and the speed of the object. The summation of the areas of each part shows the change (growth or decrease) in the object's effect area.

CHAPTER 6

FUZZY SPATIAL/ASPATIAL INDEXING

In this chapter, the adaption of an index structure from R*-tree for fuzzy spatial and aspatial data is explained. This new tree is called *Enhanced R*-tree* throughout our work. We present the logical structure of enhanced R*-tree and visualize it with meteorological data. Enhanced R*-tree is very flexible so that any data in the leaves can be indexed and the indexed attributes are fuzzified in upper levels.

6.1 Enhanced R*-Tree

R*-tree is a variant of R-tree family that uses rectangles to organize spatial data. While the directory nodes hold the organizing rectangles, the leaf nodes hold the data itself. R*-tree introduces a *forced reinsert* policy which means that whenever a node overflows it is not split right away but firstly p entries are removed and reinserted into the tree.

The structure of Enhanced R*-tree is depicted in Figure 20. It basically shows three parts in the tree: the root of the tree, the directory nodes and the data nodes. The fields in the figure can be summarized as follows:

R-tree*: This node is the header of the whole Enhanced R*-tree and includes some statistical data about the tree.

num_of_data: Number of stored data

num_of_dnodes: Number of data nodes

num_of_inodes: Number of directory nodes

root_is_data: Shows if the root is a data node

root_ptr: Pointer to the root node

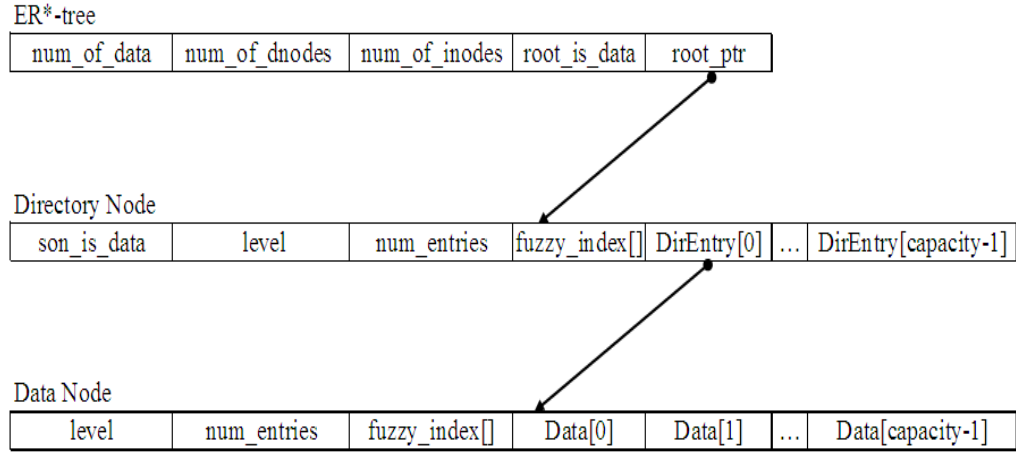


Figure 20: The structure of Enhanced R*-tree

Directory Node: Directory node implements the intermediate nodes in the tree. This is where the organizing rectangles (MBR) are stored. The members of a directory node are as follows:

son_is_data: Shows whether the son is a data node

level: the level of the directory node in the tree

num_entries: number of directory entries in the directory node

fuzzy_index[]: An array of fuzzy indexing values for the underlying nodes. Assume that there are n attributes of each data record. Then i^{th} attribute is stored in the following indices:

Low value of the attribute: $\text{fuzzy_index}[2*i-2]$

High value of the attribute: $\text{fuzzy_index}[2*i-1]$, where $1 \leq i \leq n$

For example in our application eight attributes are stored in the following indices:

0-1: Direction of station's location in 360°.

2-3: Altitude of the stations low and high value

4-5: Temperature low and high value

6-7: Humidity low and high value

8-9: Pressure low and high value

10-11: Precipitation low and high value

12-13: Wind Direction low and high value

14-15: Wind Speed low and high value

DirEntry []: Array of directory entries which has a pointer to another directory or data node. Directory entries also hold a rectangle which covers all rectangles under this directory node. The rectangle indexes the underlying nodes spatially. The capacity of the directory node is calculated with the size of Enhanced R*-tree block size divided by the size of each directory entry.

Data Node: Data node implements the leaf nodes in the tree. This is where the data objects are stored. The members of the data node are as follows:

level: the level of the data node in the tree

num_entries: number of data objects in the data node

fuzzy_index[]: An array of fuzzy indexing values for the object's attribute values. These attributes are same ones which are explained in directory node. The fuzzy index of data node holds the low and high range values for the data objects' attributes whereas in the directory nodes these are the range values for the whole nodes underlying. As the level of the directory nodes increase the range gets bigger and bigger. In the root directory node the fuzzy index stores the full range values for the whole tree.

Data []: Array of Data objects which has spatial location and other meteorological attribute values. The attributes in Data object are stored in the following indices:

0-3: Location – $\{(x_1, y_1)-(x_2, y_2)\}$

4: Altitude

5: Date Time

6: Station Number

7: Temperature

8: Humidity

9: Pressure

10: Precipitation

11: Wind Direction

12: Wind Speed

6.2 Building the Enhanced R*-Tree

An enhanced R*-tree is built in two steps: First the primary index is built based on minimum bounding rectangles and then secondary index is built based on the attributes of the objects. The creation algorithm shows how the primary and secondary indexes are built in Figure 21.

Algorithm for Enhanced R-tree Creation*

Input: Input data file

Output: Enhanced R-tree*

While Not EOF(input data file)

Read a line

Parse MBR and other attributes

Figure 21: Enhanced R*-tree insertion algorithm


```

Create Data Object
Insert into tree recursively starting from the root pointer
    Get corresponding son
    Insert into son
        If data node capacity is full
            Calculate the center of the node
            Sort the entries by the distance to the center
            Copy the nearest %70 entries to new node
            Reinsert the last %30 entries
        Else if reinsert is applied then
            Split the node
        End if
    If Split happens in the son then
        Create a new entry to hold the new son
        Insert this entry to directory node
    End if
    If directory node splits then
        Split the directory node
    End if
End While
Build secondary index
    For each entry in the node
        Get son
        If the son is directory node then
            Build secondary index for the son

```

Figure 21: Enhanced R*-tree insertion algorithm (cont'd)

```

Else
  For each attribute of Data objects
    If the attribute is less than minimum range in secondary index
      Set the minimum of index range
    End if
    If the attribute is bigger than max range in secondary index
      Set the maximum of index range
    End if
  End For
End if
Set the minimum value of the corresponding attribute in secondary index
Set the max value of the corresponding attribute in secondary index
End For

```

Figure 21: Enhanced R*-tree insertion algorithm (cont'd)

6.3 The Visualization of Enhanced R*-Tree

The structure of the Enhanced R*-tree is visualized with meteorological data. Turkey's meteorological measurements data between 30.12.2007 00:00 and 01.01.2008 21:00 in Table8 are read from a text file and inserted into the tree. The text file has the following fields:

F0: Longitude of the meteorological station
 F1: Latitude of the meteorological station
 F2: Altitude of the meteorological station
 F3: Date Time (yyyymmddhh)

F4: Station Number, an international unique code given to each meteorological station. The first two digits are for the country code and the rest is the station number.

F5: Temperature in Celsius

F6: Humidity as percentage

F7: Pressure in milibar

F8: Precipitation Hour

F9: Wind direction in 360° scale

F10: Wind Speed in knots

Table 8: The text file structure for meteorological data

F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
41,11	31,48	248	2007123000	17018	-4,70	97	NULL	1	0	0
41,38	32,20	189	2007123000	17020	1,80	94	1029	1	0	0
41,27	31,48	118	2007123000	17022	3,30	72	1029	1	140	15

In Figure 22 the Enhanced R*-tree is depicted. There are three levels in the tree. The smallest rectangles at level 3 show the data nodes. At level 2 and level 1 the directory nodes group smallest rectangles. The logical structure of the Enhanced R*-tree is seen in Figure 23. The details of the tree in general are presented in the header of the tree:

- *Number of data:1059*
- *Number of data nodes:116*
- *Number of internal (or directory) nodes:7*
- *Pointer to the first node*

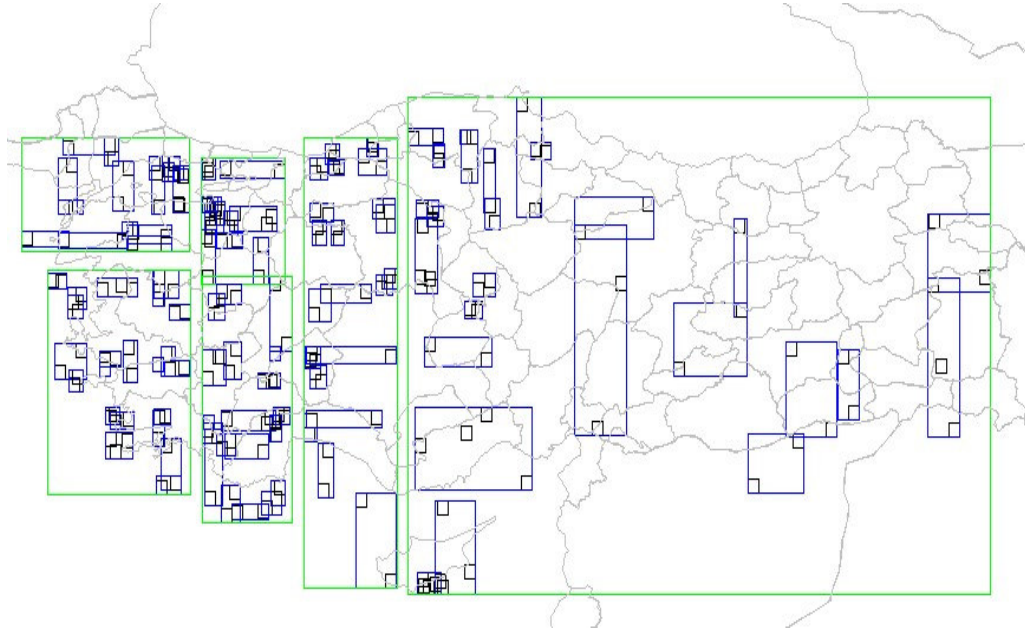


Figure 22: Enhanced R*-tree visualization

For the first directory node, the details are presented as follows:

- the kind of son node (i.e. data or directory node): false
- the level of the directory node:2
- number of entries in the node:6
- a fuzzy index array which shows the range values for spatial and aspatial attributes in all data nodes which can be accessed through this directory node. The values in the array indicate the range values in the following order:
 - lowest value for orientation of the meteorological stations under this node in 360°: 2°
 - highest value for orientation of the meteorological stations under this node in 360°: 355°
 - lowest altitude value of the meteorological stations under this node:
 - highest altitude value of the meteorological stations under this node:

R*-tree

num_of_data	num_of_dnodes	num_of_inodes	root_ptr
1059	116	7	•

Directory Node	
son_is_data	false
level	2
num_entries	6
fuzzy_index[]	{2,355,0,1822,-16.7,17,16,99,990,1053,1,2,0,360,0,51}
DirEntry[0]	(100,243)-(265,339)
DirEntry[1]	(265,339)-(419,455)
DirEntry[2]	(252,324)-(278,360)
DirEntry[3]	(427,921)-(239,559)
DirEntry[4]	(122,244)-(350,495)
DirEntry[5]	(253,330)-(354,513)

Directory Node	
son_is_data	true
level	1
num_entries	19
fuzzy_index[]	{172,210,0,350,-9.8,17,16,90,990,1030,1,2,0,360,5,36}
DirEntry[0]	
...	
DirEntry[18]	(311,324)-(485,502)

Data Node	
level	0
num_entries	9
fuzzy_index[]	{208,209,20,100, 3.5,16,18,85,1021,1022,1,2,0,350,5,15}
Data[0]	
...	
Data[8]	(314,324)-(485,495)
	{2007-12-30 00:00, 52, 17302,6.8,36,1022,1,0,5}

Figure 23: Enhanced R*-tree nodes

- minimum temperature value measured by the stations under this node: -16.7
- maximum temperature value measured by the stations under this node: 17
- minimum humidity value measured by the stations under this node: 16
- maximum humidity value measured by the stations under this node: 99
- minimum pressure value measured by the stations under this node: 990
- maximum pressure value measured by the stations under this node: 1030
- The shortest duration of precipitation in hour: 1
- The longest duration of precipitation in hour: 2
- The minimum angle of wind direction in 360° scale: 0°
- The maximum angle of wind direction in 360° scale: 360°
- The calmest wind speed value in knots : 0
- The strongest wind speed value in knots : 51
- An array of directory entries under this node, each of which specifies a bounding rectangle and a pointer to the underlying node whether directory or data node. The bounding rectangle is the minimum one which can cover all objects underlying.

For some data node the details are presented as follows:

- the level of the data node : 0
- number of entries in the node : 9
- a fuzzy index array which shows the range values for spatial and aspatial attributes in the data nodes. The order of the values in the array is the same as directory node but here it shows the range values for the data node,
- An array of data objects with some attributes. Data[8] is given as an example in Figure 23 with the following attributes:
 - Bounding rectangle borders

- Altitude
- Valid time
- Station number
- Temperature
- Humidity
- Pressure
- Precipitation duration in hour
- Wind direction
- Wind speed

6.4 Querying the Enhanced R*-Tree

In this section the working of Enhanced R*-tree is shown for various kinds of query types. The query types can be spatial, aspatial which may include fuzzy or crisp data and a combination of these.

6.4.1 Crisp Aspatial Queries

Crisp and aspatial input parameter is used to fetch the object by means of Enhanced R*-tree. For example:

Retrieve the measurements which have 5 °C temperatures.

An algorithm is presented to show the working of Enhanced R*-tree in Figure 24. The nodes of the Enhanced R*-tree is searched for the input parameter. In our example it is the temperature attribute and 5 °C. Whether the searched node is directory or data node the fuzzy index is checked for the aspatial attributes. The indices values are four for low temperature and five for the high temperature. So *fuzzy_index[4]* and *fuzzy_index[5]* are checked and if the input is between those ranges than the node satisfies. If the node is directory node each sub-tree under the node is searched recursively, otherwise this should be a data node. In that case the

data entries are checked and this time we search for the exactly matching entries. If any data entry satisfies the object is retrieved.

Crisp Aspatial Query Algorithm for Enhanced R*-tree:

Input: Enhanced R*-tree, aspatial parameter value

Output: The objects having exact aspatial value

For each Enhanced R*-tree node

 Check the low and high value for input parameter in the fuzzy index

If the input parameter is between the low and high value

If the node is a directory node

for each directory entry in R*-tree directory node

 Search the nodes pointed by directory entry

end for

else if the node is a data node

for each Data entry in R*-tree data node

if the Data entry's related attribute equals input parameter

 Get the object

end if

end for

end if

end if

end for

Figure 24: Crisp Aspatial Query Algorithm in Enhanced R*-tree

Crisp Spatial Query Algorithm for Enhanced R*-tree:

Input: Enhanced R*-tree, spatial parameter value

Output: The objects having exact spatial value

For each Enhanced R*-tree node

If the input parameter is related to the MBR of the tree &&

the input rectangle intersects (inside, overlap) the MBR of the node

If the node is a directory node

for each directory entry in R*-tree directory node

Search the nodes pointed by directory entry

end for

else if the node is a data node

for each Data entry in R*-tree data node

if the Data entry's rectangle inside the input parameter

Get the object

end if

end for

end if

else if the input parameter is related to the fuzzy index &&

the input parameter is between the low and high value

//fuzzy index[0-1] for direction and fuzzy_index[2-3] for altitude

***Search sub-tree for the directory nodes or get the objects as in the if part
of this else if block***

end if

end for

Figure 25: Crisp Spatial Query Algorithm in Enhanced R*-tree

6.4.2 Crisp Spatial Queries

In this type of query, crisp spatial input parameter is used to find the objects. For example:

- Fetch objects in given $(x_1, y_1)-(x_2, y_2)$ rectangular range*
- Fetch objects at 270° degree of orientation*
- Fetch object at 1500 meters of altitude*

An algorithm is presented to show the working of Enhanced R*-tree in Figure 25. The primary indexing attribute in the Enhanced R*-tree is the MBR of the nodes. So if the input is a range specified by a rectangle then the MBR of the nodes are used. But if the input parameter is related to other spatial parameters such as orientation or altitude then the secondary fuzzy index is searched (i.e. *fuzzy_index[0-1]* for orientation and *fuzzy_index[2-3]* for altitude). If the node is directory node each sub-tree under the node is searched recursively, otherwise the data entries are checked and if any data entry satisfies the object is retrieved.

6.4.3 Fuzzy Spatial/Aspatial Queries

Fuzzy spatial or aspatial parameters are input of this query type. Some examples are given here:

- Retrieve the measurements which have warmer than 5°C temperature,*
- Get the N - NW oriented objects,*
- Find the cities higher than 1000 meters.*

An algorithm is presented to show the working of Enhanced R*-tree for three fuzzy spatial and/or aspatial criteria in Figure 26.

Fuzzy Spatial/Aspatial Query Algorithm for Enhanced R*-tree:

Input: Enhanced R*-tree, fuzzy spatial/aspatial parameters combination
(temperature, orientation ,altitude)

Output: The objects satisfying all three criteria

For each Enhanced R*-tree node

Check the low, high value for input parameter₁ (>5 °C)in the fuzzy index

Check the input parameter₂, orientation (N-NW range) values of fuzzy index

*Check the input parameter₃, altitude low, high value (>1000 m) values of
fuzzy index*

If all criteria is between the ranges

If the node is a directory node

for each directory entry in R*-tree directory node

Search the nodes pointed by directory entry

end for

else if the node is a data node

for each Data entry in R*-tree data node

if the Data entry's related attributes satisfies input parameters

Get the object

end if

end for

end if

end if

end for

Figure 26: Fuzzy Spatial/Aspatial Query Algorithm in Enhanced R*-tree

The input parameters are related to fuzzy index. *fuzzy_index[0-1]* for orientation and *fuzzy_index[2-3]* for altitude and *fuzzy_index[4-5]* for temperature are checked for directory nodes and data nodes. If the node is directory node each sub-tree under the node is searched recursively, otherwise the data entries are checked and if any data entry satisfies the object is retrieved.

CHAPTER 7

IMPLEMENTATION AND PERFORMANCE EVALUATION

In Chapter 5, we implemented the proof-of-concept type queries and verified that the application runs smoothly. It is an integrated environment that the objects are stored and fetched from an object oriented database whereas the knowledge base applies some rules whenever necessary and the user interface runs as a coordinator.

After the proof-of-concept type work, we believe that it is necessary to validate the application with real data. In meteorology application there is excessive spatial data so also an index structure adaption would be useful in querying. In the previous work [45] several spatial index structures have been adapted and compared. In this work, we adapt one of them, R*-tree into the spatiotemporal application and scalability of the application is tested as the number of records grows.

7.1 Implementation

The application is developed in Java using NetBeans IDE 6.5. The other components of the implementation environment are

- db4o 6.4 [51] for object database, which is an open source database engine,
- jess.jar [39], a rule engine for java platform
- Enhanced R*-tree [22], a spatial index structure

The application runs on a notebook computer with Intel Core Duo CPU T9400, 2.53 GHz, 4 GB RAM.

7.2 An Object Oriented Database, Db4O

Db4O is an open source object database that enables to store and retrieve any application object by predefined database libraries. For example, an object is firstly created in Java and then stored in the database with *set* command:

```
db.set(<savedObject>), where  
db = Db4o.openFile(<databaseName>);
```

The stored objects are fetched from database by the *get* command:

```
db.get(<getObject>);
```

The objects in Db4O are visualized by *ObjectManager* tool. The objects can be inquired and the whole object hierarchy can be seen. In Figure 27, the stored objects like ST_Object with attributes can be seen on the left part of the screen. The other parts of the tool are the upper part for querying and the middle part where the results of the queries and some statistics can be seen.

7.3 The Rule Engine, Jess

Jess is a rule engine developed in Java language. By using the knowledge supplied in the form of declarative rules, Jess is able to inference some results. It's scripting language allows to access to Java's APIs so one can create Java objects, call Java methods and implement Java interfaces. An example for the declarative rules is given in Figure 28.

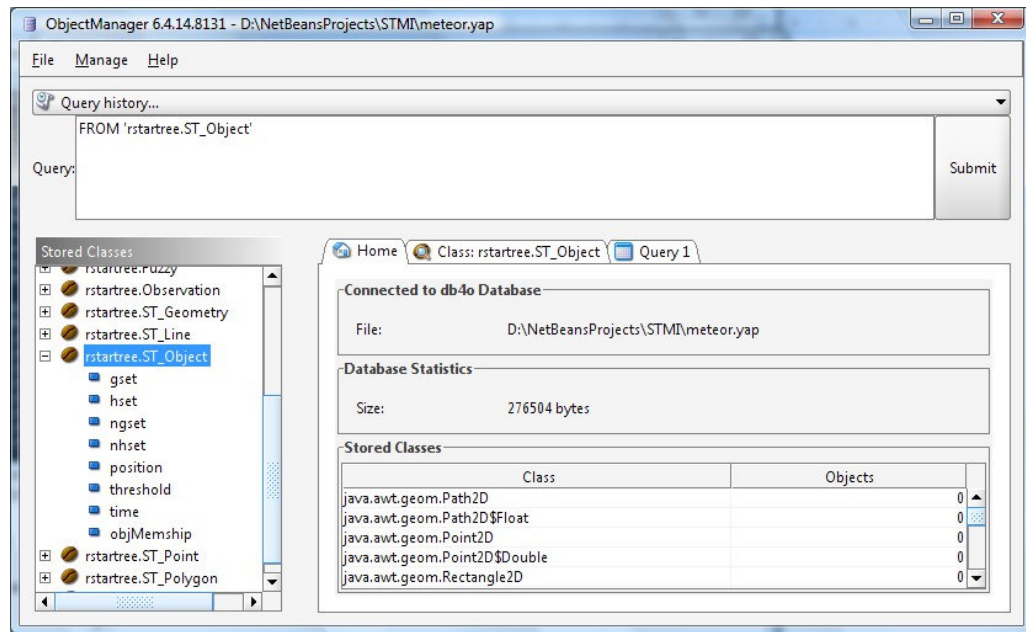


Figure 27: Database visualization by ObjectManager tool

The rule defines the geographic line's status. FSP module which is a Java module is called from FKB. So it is defined at the beginning. In the section before the double arrow the prerequisites are written. So any *GeoLine* object which is put into the queue of the FKB should satisfy some constraints. *LineType* should not be null and be *SeaLine*. It should have an attribute as *threshold*. The *obj* refers to the object as the final parameter.

The right side of the double arrow is applied to the objects which satisfy the prerequisites. The geographic line's topological relation with *wave* and *wind* objects is checked. If any of them is above threshold value then the line status is set to "*Restricted*" otherwise to "*Clear*".

```

defglobal ?*fp* = (new Fsp))
(defrule geolinesstatus

?p1 <- (GeoLine
  (lineType ?lT&:(and (neq ?lT nil) (eq ?lT "SeaLine")))
  (threshold ?th)
  (OBJECT ?obj))
=>
(bind ?result (call ?*fp* FuzzyRelation ?obj "wave" "wavy"))
(bind ?result2 (call ?*fp* FuzzyRelation ?obj "wind" "windy"))
(bind ?minresult (min ?result ?result2))
(if (> ?minresult ?th) then
  (call ?obj setlineStatus "restricted")
)
(if (< ?minresult ?th) then
  (call ?obj setlineStatus "clear")
)
(call ?obj setOverlap ?minresult)
)

```

Figure 28: An example of rules

7.4 Crisp Queries

The crisp queries are basic spatial queries. They do not include fuzzy or semantic input nor require knowledge base processing but they are used by fuzzy/semantic queries.

In the next sections, the crisp queries are described by the details of input screens, implementation algorithms and the output screens. The crisp queries which are supported by the application are as follows:

- Point query
- Range query
- Circle query
- Ring query
- K^{th} Nearest Neighbor (kNN) query

7.4.1 Point Query

Point query fetches all objects at a specific point which is an input data by the user in the form of (x, y) coordinate. In the example, the user asks all objects located at $(x=220, y=440)$ coordinate. The user interface gets the input coordinates and the bridge as a coordinator applies the *point query algorithm* in Figure 30.

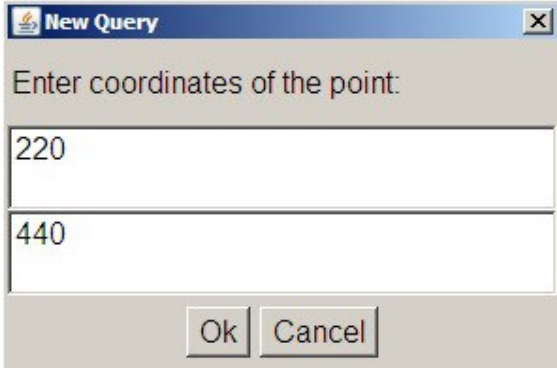


Figure 29: Point query input screen

Point Query Algorithm:

Input: R^* -tree, a point data located at (x, y)

Output: The objects which are located at the input location

1. Create a point from input coordinates, $P \leftarrow (X, Y)$
2. Create an empty result list res
3. Search R^* -tree nodes
 - If the node is a directory node*
 - for each directory entry in R^* -tree directory node*
 - if the point P is inside of the directory node DN*
 - search sub-tree of DN*
 - end if*
 - end for*
 - else if the node is a data node*
 - for each entry in R^* -tree data node*
 - if the point P is inside the data objects' points*
 - call fetch utility of object database*
 - insert into res*
 - end if*
 - end for*
 - end if*
4. call $drawPoint$
5. display output objects

Figure 30: Point query algorithm

Firstly a reference point, P is created at (x, y) . Then, an empty list *res* for holding the resulting objects is created. The search subroutine of the R*-tree is called with these two parameters. The directory nodes are checked for whether the bounding rectangles contain the reference point. If so the sub tree which is covered by the directory entry is searched recursively. The algorithm reaches to a data node if the point is inside the directory entries. The data object coordinates and reference point are checked one by one and the matching objects are fetched from object database by the fetch utility to append to the result list. The objects in the result list are mapped in Figure 31.

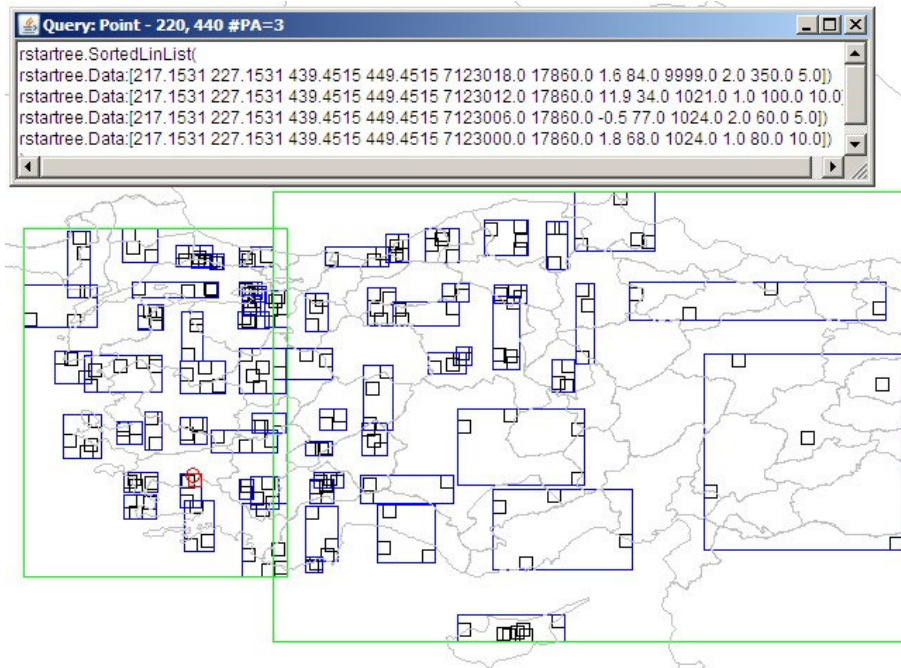


Figure 31: Point query results.

In Figure 31 the reference point is mapped by the red colored circle. The data objects at leaf level which intersects the reference point is also shown at the same

point in red. The spatial representation is detailed in a separate window above. The attributes of the objects in the result list are presented. The details are coordinates of the object, date time, station number, and meteorological parameters.

7.4.2 Range Query

The range query searches for the objects in a spatial range between (x_1, y_1) and (x_2, y_2) which is an input data.

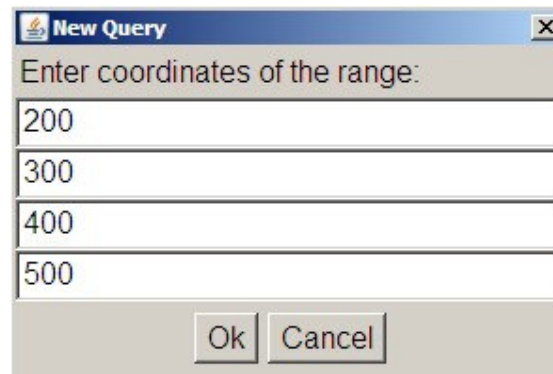
The image shows a software window titled "New Query" with a close button in the top right corner. Below the title bar, the text "Enter coordinates of the range:" is displayed. There are four text input fields stacked vertically. The first field contains the number "200", the second contains "300", the third contains "400", and the fourth contains "500". At the bottom of the window, there are two buttons: "Ok" and "Cancel".

Figure 32: Range query input screen

In Figure 32, the user enters a range between $(x_1=200, y_1=300)$ and $(x_2=400, y_2=500)$. The UI gets the input coordinates and the bridge fetches the spatial objects from OODB by using the following algorithm:

Range Query algorithm:

Input: R^* -tree, a spatial range defined by a rectangle (x_1, y_1) and (x_2, y_2)

Output: The objects which are covered by the range

1. Create a rectangle from input coordinates, $MBR \leftarrow (x_1, y_1, x_2, y_2)$
2. Create an empty result list res
3. Search objects in the input range
 If the node is a directory node
 for each entry in R^* -tree directory node
 if the spatial relation between directory entry bounds and MBR is
 $INSIDE$ or $OVERLAP$
 search sub-tree of directory entry
 end if
 end for
 else if the node is a data node
 for each entry in R^* -tree data node
 if the object's geometry and MBR intersects
 call fetch utility of object database
 insert into res
 end if
 end for
 end if
4. call $drawRange$
5. display output objects

Figure 33: Range query algorithm

In the first step of the range query algorithm a rectangle is created from the input data. Then, an empty list *res* for holding the resulting objects is created. The range search subroutine of the R*-tree is called. The directory nodes are checked for whether the bounding rectangles inside of or overlap with the input range. If so the sub tree which is covered by the directory entry is searched recursively. If the algorithm reaches a data node the data object coordinates and input range are checked for overlap or inside relation. The matching objects' details are fetched from object database by the fetch utility and appended to the result list. The objects in the result list are mapped in Figure 34.

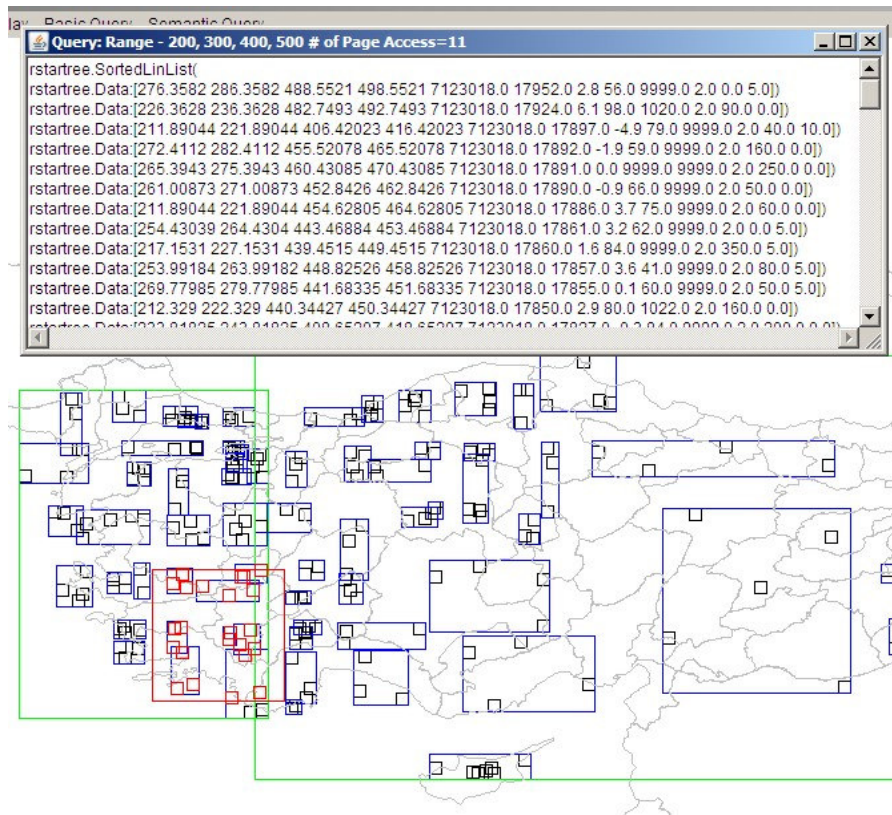


Figure 34: Range query result

The input range is drawn as a red colored bigger rectangle in Figure 33. The objects inside the input range at leaf level are also shown by red color. The object details are presented in a separate window above the screen. The details are coordinates of the object, date time, station number, and meteorological parameters.

7.4.3 Circle Query

A circle is defined with two parameters, the centre and the radius. The parameters are entered in UI by the input screen in Figure 35:

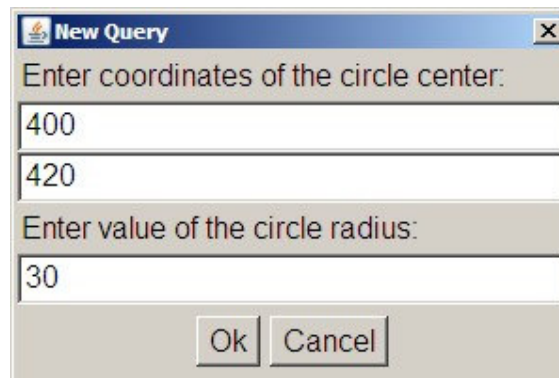


Figure 35: Circle query input screen

The user enters a circle center at $(x=400, y=420)$ with radius 30. The UI gets the input and the bridge fetches the spatial objects which reside inside the reference circle from OODB by using the following algorithm:

Circle Query algorithm:

Input: *R*-tree, circle parameters center and radius*

Output: *The objects which reside inside the circle*

1. Create a circle from input coordinates and the radius, $\text{Circle} \leftarrow (x, y, r)$

2. Create an empty result list *res*

3. Search objects in the circular area

If the node is a directory node

for each directory entry in R-tree directory node*

if directory entry bounds and circle area intersects

search sub-tree of directory entry

end if

end for

else if the node is a data node

for each entry in R-tree data node*

if the object's geometry and Circle intersects

call fetch utility of object database

insert into res

end if

end for

end if

4. call *drawCircle*

5. display output objects

Figure 36: Circle query algorithm

Circle query algorithm creates a reference circle from the center coordinates and radius. The circular area search subroutine of the R*-tree is called with circle and output list *res* parameters. The directory nodes are checked for whether the bounding rectangles intersect with the circle. If so the sub tree which is covered by the directory entry is searched recursively. If the algorithm reaches a data node the data objects which reside in the circular area are fetched from database and appended to the result list. The objects in the result list are mapped in Figure 37.

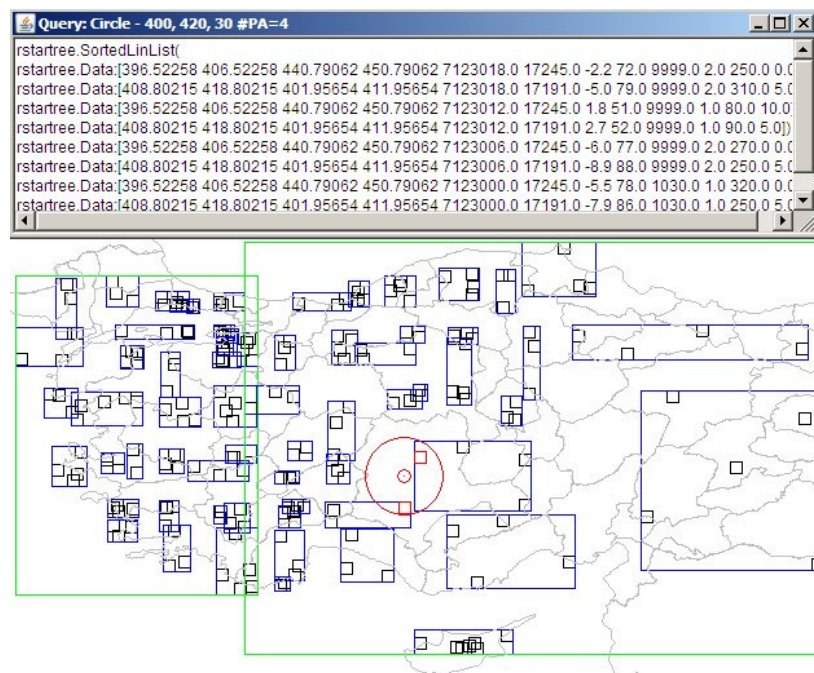
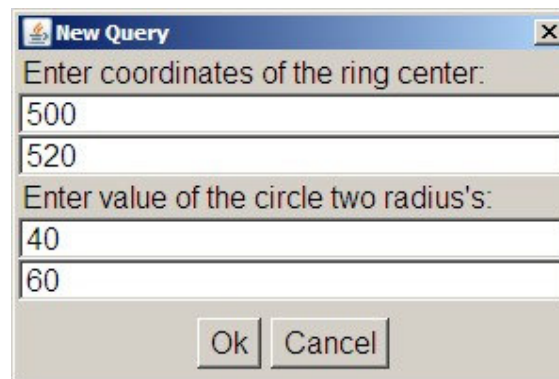


Figure 37: Circle query result

In the output screen the input circle and its radius can be seen. The objects inside the circle at leaf level of R*-tree are also visualized by red color. The object details are presented in a separate window above the screen. The details are coordinates of the object, date time, station number, and meteorological parameters.

7.4.4 Ring Query

A ring is the difference of two circles with the same center point but different radiuses. The ring query deals with the objects which reside inside the ring. The ring area is obtained by subtracting small circle from bigger circle. The query parameters are entered by UI as in Figure 38:



The image shows a 'New Query' dialog box with a blue title bar. It contains two sections for input. The first section is labeled 'Enter coordinates of the ring center:' and has two input fields; the first contains '500' and the second contains '520'. The second section is labeled 'Enter value of the circle two radius's:' and has two input fields; the first contains '40' and the second contains '60'. At the bottom of the dialog are two buttons: 'Ok' and 'Cancel'.

Figure 38: Ring query input screen

The ring in this query example is defined with center at $(x=500, y=520)$ and the inner circle radius ($r_1=40$) and outer circle radius ($r_2=60$). The UI gets the input and the bridge fetches the spatial objects which reside inside the ring area by using the Ring Query Algorithm in Figure 39.

In Figure 39, ring query algorithm creates two reference circles with r_1 and r_2 and the same center point. The ring is obtained as the difference of two circles. The ring area search subroutine of the R*-tree searches from upper nodes to the leaf nodes. If the directory nodes intersect with the ring area search continues to the bottom of the node until a data node is reached. If the algorithm reaches to a data node and the

Ring Query algorithm:

Input: R^* -tree, ring parameters: center, radius₁ and radius₂

Output: The objects in the ring area

1. Create $Circle_1 \leftarrow (x, y, r_1)$
2. Create $Circle_2 \leftarrow (x, y, r_2)$
3. Set $Ring \leftarrow Circle_2 - Circle_1$ //Difference operation
4. Create an empty result list res
5. Search objects in the ring area via R^* -tree nodes
 - if** the node is a directory node
 - for** each directory entry in R^* -tree directory node
 - if** directory entry bounce and the Ring intersects
 - search sub-tree of directory entry
 - end if**
 - end for**
 - else if** the node is a data node
 - for** each entry in R^* -tree data node
 - if** the object's geometry and ring intersects
 - call fetch utility of object database
 - insert into res
 - end if**
 - end for**
 - end if**
6. call $drawRing$
7. display results in the main map and detail window

Figure 39: Ring query algorithm

data objects reside in the ring area, they are appended to the result list. The objects in the result list are mapped in Figure 40.

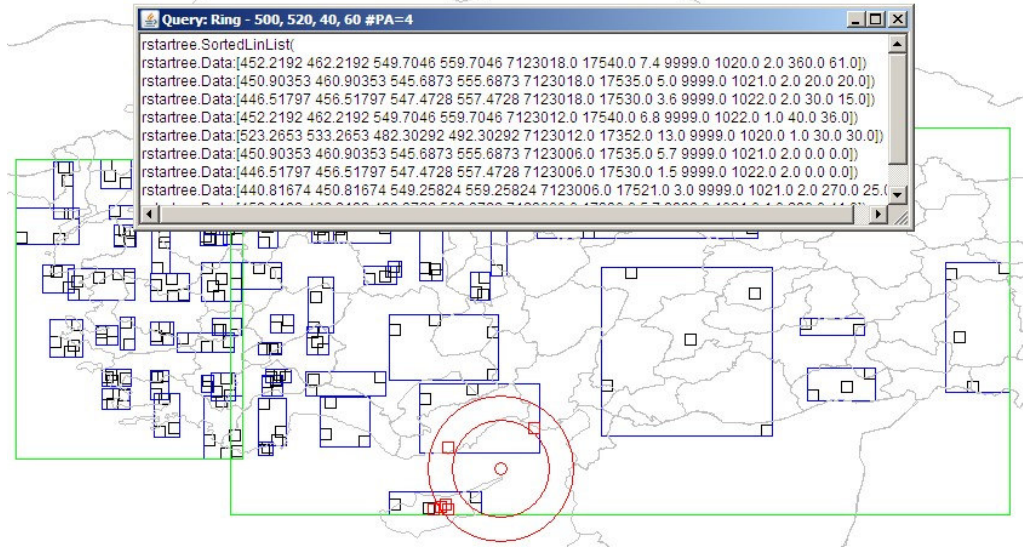


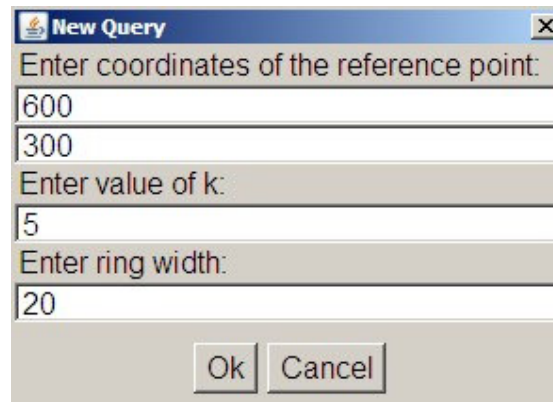
Figure 40: Ring query result

In the output screen the ring area and the objects inside are visualized by red color. The object details are presented in a separate window above the screen. The details are coordinates of the object, date time, station number, and meteorological parameters.

7.4.5 Kth Nearest Neighbor (KNN)

KNN query finds first nearest k entries to a reference object. In our implementation we use the ring query to find the kth nearest neighbor of an object and the center of the ring is accepted as the reference point. The algorithm starts with a circle and the

ring is the area between the center and the circle. In every loop of the algorithm matching objects are fetched and appended to the output list. In the new run a ring is created by increasing the circles radius so the outer circle of the previous run becomes the inner circle of the previous run and a new outer circle is created. The loops continue until k object is found. The parameters are entered in UI by the following screen in Figure 41:



The image shows a 'New Query' dialog box with a blue title bar and a close button. It contains four input fields with labels in red text: 'Enter coordinates of the reference point:', 'Enter value of k:', and 'Enter ring width:'. The first field has two sub-inputs with values '600' and '300'. The second field has the value '5'. The third field has the value '20'. At the bottom are 'Ok' and 'Cancel' buttons.

Field Label	Value
Enter coordinates of the reference point:	600 300
Enter value of k:	5
Enter ring width:	20

Figure 41: KNN query input screen

The center of the rings is $(x=600, y=300)$, ring width is 20 and 5 nearest neighbor are required. KNN query algorithm is described in Figure 42.

KNN query algorithm runs similar to the ring query algorithm. In every loop the ring is searched. At the end of the loop $circle_1$ is replaced with $circle_2$ and a new $circle_2$ is created. The loop continues until k object is found.

KNN Query algorithm:

Input: R^* -tree, reference point coordinates (x, y) , number of nearest neighbors k , ring width

Output: The k nearest objects to the reference point

1. Set $r_1 \leftarrow 0$
2. Create a circle from input coordinates and the radius, $\text{Circle}_1 \leftarrow (x, y, r_1)$
3. Create an empty result list res
4. Search objects in the ring area via R^* -tree nodes

Loop until k objects are found

Set $r_2 \leftarrow r_1 + \text{ring width}$

Create a circle $\text{Circle}_2 \leftarrow (x, y, r_2)$

Set $\text{Ring} \leftarrow \text{Circle}_2 - \text{Circle}_1$

If the node is a directory node

for each directory entry in R^* -tree directory node

if the entry's bounce and Ring intersects

search sub-tree of directory entry

end if

end for

else if the node is a data node

for each entry in R^* -tree data node

if the data geometry and Ring intersects

call fetch utility of object database

insert into res

end if

end for

Figure 42: KNN query algorithm

end if

Set $r_1 \leftarrow r_2$

Set $\text{Circle}_1 \leftarrow (x, y, r_1)$

end for

5. *call drawRings*

6. *display results in the main map and detail window*

Figure 42: KNN query algorithm (cont'd)

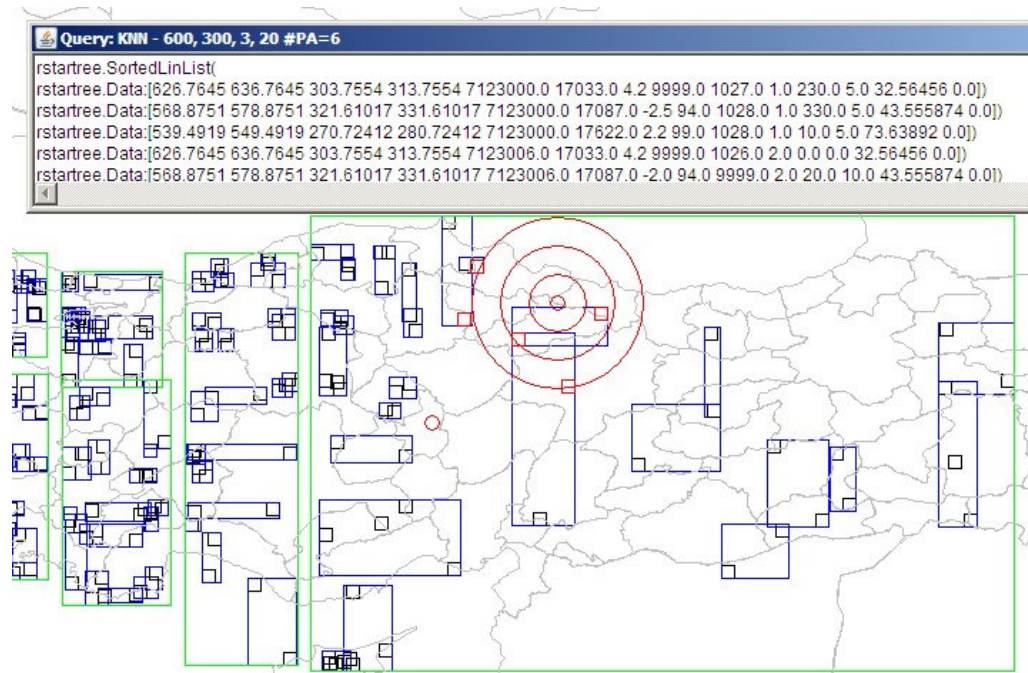


Figure 43: KNN query result

The result of the KNN query is shown in Figure 43. The innermost circles are increased by some width and in every loop new nearest neighbors are found. In the

output screen the ring area and the objects inside are visualized by red color. The object details are presented in a separate window above the screen.

7.5 Fuzzy Queries

Fuzzy queries in general are more complex queries than crisp queries. The complexity arises since the queries include fuzzy input and also they may require some intelligence which is achieved by some rule processing in FKB. The crisp query algorithms can be used for fuzzy semantic queries.

In the next sections, the fuzzy semantic queries are described by the details of input screens, implementation algorithms and the output screens. The fuzzy queries implemented in the application are:

- Fuzzy Spatial Relations Query
- Fuzzy Spatiotemporal Query

7.5.1 Fuzzy Spatial Relations Query

Topological, directional and distance relations are combined in fuzzy spatial relations query. A union of constraints for these relations is supplied by the user. The constraints may be fuzzified also. For instance, if an object is not in the exact North direction of the reference object but to some fuzzy degree we also accept that object in the result. Similarly, we can find the degree of a topological relation so that for instance 0.8 overlapping objects with the reference object may be fetched as the result of the query. The parameters are entered in UI by the screen in Figure 44.

According to the input screen, the reference object is located at ($x_1=450$, $y_1=517$) and ($x_2=350$, $y_2=389$). The distance constraint is given as a range between 0 and

New Query

Reference Object

450

517

350

389

Distance Range

0

200

Fuzzy Direction

0.7

☒ E

☐ NE

☐ N

☐ NW

☐ W

☐ SW

☐ S

☒ SE

Fuzzy Topology

1.0

☒ Disjoint

☐ Meet

☐ Equal

☐ Inside

☐ Covers

☐ Contains

☐ CoveredBy

☐ Overlap

Ok Cancel

Figure 44: Fuzzy Spatial Relations Query input screen

200. The direction constraint can be a union of 8 directions (i.e. N, NE, S, etc) with possible fuzzy degree. In the example user asks for 0.7 degree E, SE objects. As a last constraint, a union of 8 topological relations (i.e. Overlap, Inside, Disjoint etc.) can be selected with possible fuzzy degree. In the example user asks for disjoint objects compared to reference object. Getting input parameters the algorithm in Figure 45 is applied.

Fuzzy Spatial Relations Algorithm:

Input: R^* -tree, reference object $\{(x_1, y_1) - (x_2, y_2)\}$, minimum distance, maximum distance, fuzzy direction degree, direction values, fuzzy topology degree, relations

Output: The objects that satisfy the constraints

1. Create a rectangle from input coordinate for the reference object,

$Set\ Rect \leftarrow (x_1, y_1, x_2, y_2)$

2. Set direction from direction checkboxes

3. Set topology from topology checkboxes

4. Create an empty result list res

5. Call the R^* -tree search subroutine

If the node is a directory node

for each entry in R^* -tree directory node

if the directory entry bound is in distance range && direction constraint satisfies fuzzy index && topology constraint satisfies

search sub-tree of directory entry

end if

end for

else if the node is a data node

for each entry in R^* -tree data node

if object lies in distance range && direction constraint satisfies fuzzy index && topology constraint satisfies

call fetch utility of object database

insert into res

end if

end for

end if

6. Display results in the main map and detail window

Figure 45: Fuzzy Spatial Relations query algorithm

Fuzzy Spatial Relations algorithm starts with creating a reference object. Since the user may enter more than one direction, a binary number is created and related bits are set to 1 for selected directions. Similarly a topology bitmap is created from selected topological relations. The distance range, topology and direction bitmaps with fuzzy degrees are input to search subroutine. The search subroutine searches the directory nodes, where the bounding rectangles and the reference object are compared, and the data nodes, where objects and the reference object are compared. The objects which satisfy all constraints are appended to the result list. The objects in the result list are mapped in Figure 46.

In the output screen the reference object's geometry can be seen as the big rectangle. The data objects which are in 0-200 range, disjoint and E/SE direction are displayed. Notice that solution includes not only exact East or South East objects but also some North East or South objects. This is because of the fuzzy degree of direction constraints.

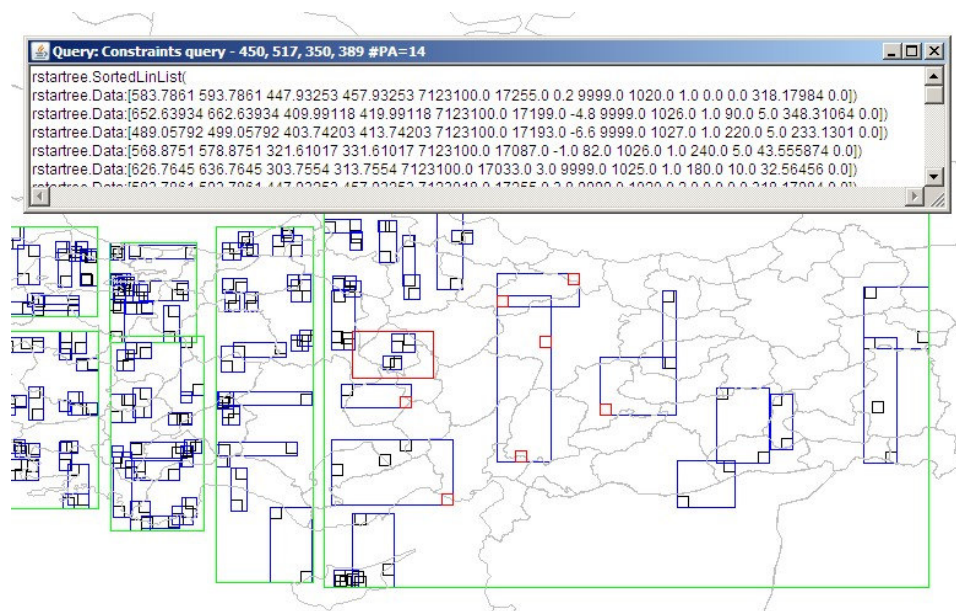
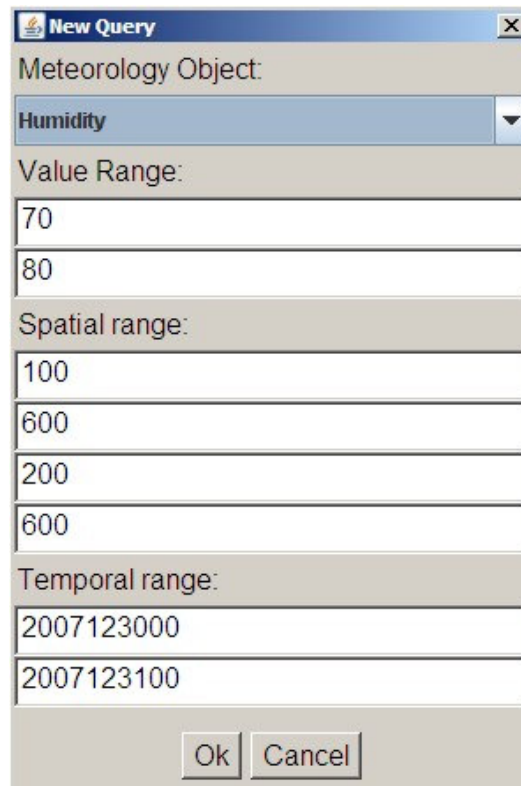


Figure 46: Fuzzy Spatial Relations query result

7.5.2 Fuzzy Spatiotemporal Query

An object can change its position and shape during a temporal interval. For example, a cold weather area moves, strengthen or weaken (i.e. becomes warmer or cooler). Fuzzy spatiotemporal query finds this kind of spatial changes in temporal interval.

The object type is selected as *Humidity* in the input screen in Figure 47. The values are entered as a range so the humidity values within the %70 and %80 values measured by the meteorological stations will be queried. The spatial range specifies the search area whereas the temporal range specifies the temporal interval (i.e. between 30.12.2007 00:00 and 31.12.2007 00:00).



New Query	
Meteorology Object:	
Humidity	
Value Range:	
70	
80	
Spatial range:	
100	
600	
200	
600	
Temporal range:	
2007123000	
2007123100	
Ok	Cancel

Figure 47: Fuzzy Spatiotemporal Query input screen

Fuzzy Spatiotemporal Query algorithm:

Input: R*-tree, Object Type, Value range, spatial range and temporal range

Output: The object's path and the values at each position

1. Create an empty result list res
2. **If** the node is a directory node
 - for** each directory entry in R*-tree directory node
 - if** directory entry bounce and spatial range intersects &&
value range is in the directory node's fuzzy index range
search sub-tree of directory entries
 - end if**
 - end for**
 - else if** the node is a data node
 - for** each entry in R*-tree data node
 - if** the object's geometry and spatial range intersects &&
value range is in the data node's fuzzy index range &&
time is in temporal interval
call fetch utility of object database
insert into res
 - end if**
 - end for**
 - end if**
3. Calculate and display trajectory
 - for** each unique temporal chronon in the result list
 - Find average value of the meteorological parameter (i.e. humidity)
 - Find center point of the objects
 - end for**

Figure 48: Fuzzy Spatiotemporal Query algorithm

4. *Display results in the main map and detail window*
 - Display matching objects*
 - Display each temporal chronon, position and the value*
 - Draw trajectory between positions*

Figure 48: Fuzzy Spatiotemporal Query algorithm (cont'd)

Fuzzy spatiotemporal query finds the movement of objects in a spatial and temporal range. The value range constraint is another constraint. The three constraints are checked in directory and data nodes. The objects satisfying the three constraints are appended to the output list. The algorithm calculates a trajectory in step three. For each unique temporal instance an average of values of objects from output list is obtained. Then these instances are connected to each other to show the trajectory of the selected object. The trajectory and output list are mapped in Figure 49.

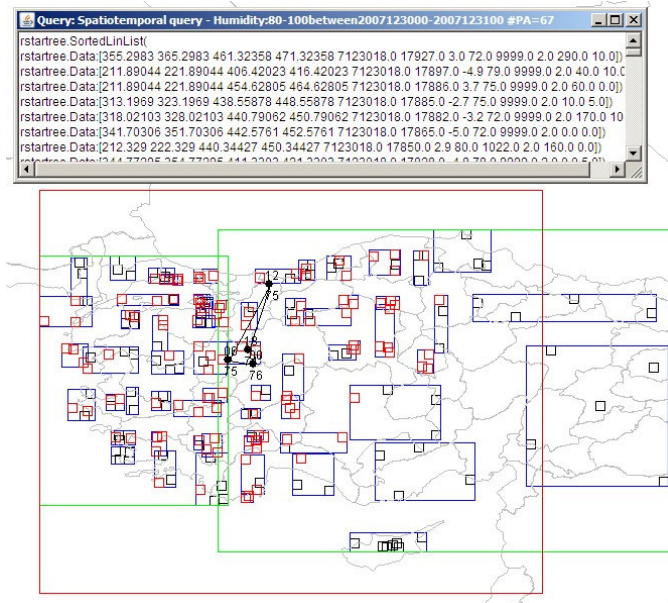


Figure 49: Fuzzy Spatiotemporal query result

In Figure 49, the average values of the objects and the temporal value are displayed as the black dots. The black lines connect the instances. In addition the red squares show the data nodes satisfying the constraints. The bigger red square shows the spatial range. And lastly the details of the objects are displayed in the upper window.

7.6 Fuzzy Semantic Queries

Fuzzy semantic queries are more complex queries than the crisp and fuzzy queries. In these queries fuzzy query algorithms, FKB processing and some semantic processing are combined.

In the next sections, the fuzzy semantic queries are described by the details of input screens, implementation algorithms and the output screens. The fuzzy semantic queries implemented in our application are:

- Fuzzy Semantic Query 1 (Extreme Conditions)
- Fuzzy Semantic Query 2 (Trajectory of Objects)
- Fuzzy Semantic Query 3 (k Highest Measurements)
- Fuzzy Semantic Query 4 (Agricultural Risky Zones)
- Fuzzy Semantic Query 5 (Altitude vs. Meteorological Parameters)

7.6.1 Fuzzy Semantic Query 1 (Extreme Conditions)

Semantic queries find semantic properties and/or behaviors of spatiotemporal objects. For example meteorological warnings are very important to urban or rural life, like floods, extreme temperatures, etc. These types of queries may be more complicated and require some intelligence so that a knowledge base component is used.

In the first semantic query, the extreme meteorological conditions are searched. The extremeness conditions are defined in FKB by the attributes of the meteorological values.

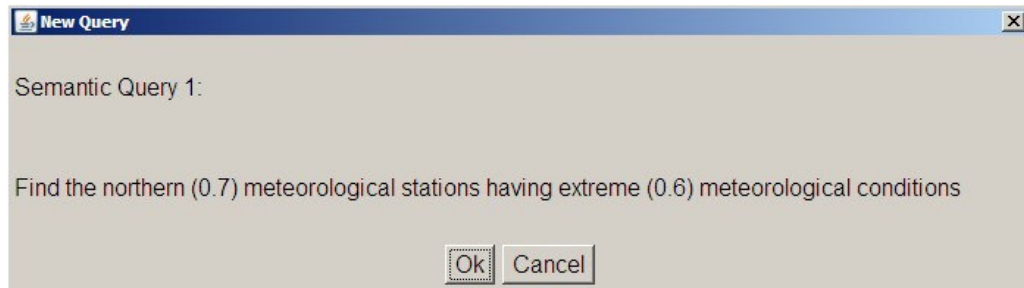


Figure 50: Fuzzy Semantic Query 1 (Extreme conditions) input screen

The query input parameters are displayed in Figure 50. The parameters are specified as fuzzy spatial constraint and fuzzy semantic constraint. The fuzzy direction (0.7 North) as the fuzzy spatial constraint and extremeness (0.6 extreme) as the fuzzy semantic constraint are input parameters of the algorithm in Figure 51.

Fuzzy Semantic Query 1 (Extreme Conditions) algorithm:

Input: *R*-tree, fuzzy spatial and semantic condition*

Output: *The objects which satisfy both conditions*

1. *Create an empty result list res*
2. *Apply fuzzy spatial relations algorithm for direction.*

Figure 51: Fuzzy Semantic query 1 (Extreme conditions) algorithm

3. *Send result list to FKB queue*
4. *Run FKB engine*
 - If the object is eligible for the rule*
 - Calculate extremeness degree of conditions using temperature, wind speed etc.*
 - Set object's attribute for extremeness*
 - end if*
5. *Get objects from output queue of FKB*
6. *Display results in the main map and detail window*

Figure 51: Fuzzy Semantic query 1 (Extreme conditions) algorithm (cont'd)

The fuzzy direction parameter is already implemented in Fuzzy Spatial Relations Query. The same algorithm is applied to filter the objects for the first constraint. Then selected objects are put into the input queue of FKB to further filter for the knowledge base predicate. In FKB, one of the semantic rules is invoked according to the query type. The degree of the extremeness is calculated by a predefined function in Figure 52.

```
(deffunction fuzzyM (?a)
  (bind ?i 1)
  (bind ?FR 0)
  (foreach ?e ?a
```

Figure 52: The degree of extremeness function in FKB

```

    (if (and (eq ?i 7) (< ?e 50) (> ?e -30)) then
      ;temperature
      (if (< ?e 0) then
        (bind ?FR (+ ?FR 0.4))
      )
    )
    (if (and (eq ?i 8) (< ?e 100) (> ?e 0)) then
      ;humidity
      (bind ?FR (+ ?FR (* (/ ?e 100) 0.2)))
    )
    (if (and (eq ?i 12) (< ?e 50) (> ?e 0)) then
      ;wind speed
      (bind ?FR (+ ?FR (* (/ ?e 40) 0.4)))
    )
    (bind ?i (+ ?i 1))
  )
  (return ?FR)
)

```

Figure 52: The degree of extremeness function in FKB (cont'd)

FuzzyM function gets an array of meteorological parameters as input. It then calculates an overall degree of extremeness by weighting each attribute such as *temperature*, *humidity* and *wind speed*. The returned fuzzy value *?FR* is set as an attribute of the object. So when the objects are returned back to *bridge* the semantic degree can be used for output. The returned objects having *?FR* greater than 0.6 are displayed as output in Figure 53.

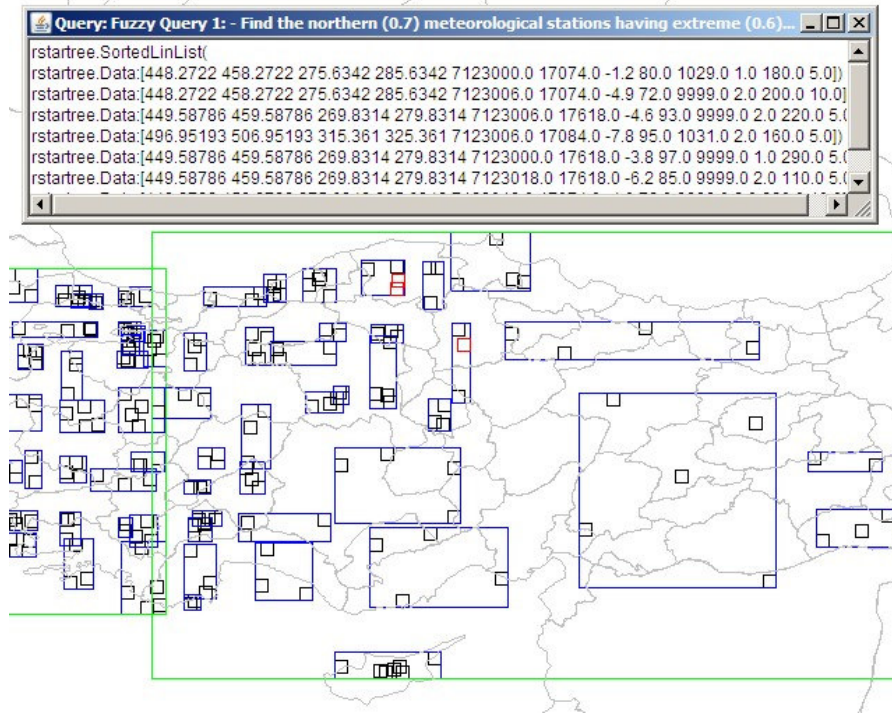


Figure 53: Fuzzy Semantic query 1 (Extreme conditions) result

In Figure 53, three stations which measured extreme meteorological conditions are displayed as the red squares. The measurements are displayed in the upper window.

7.6.2 Fuzzy Semantic Query 2 (Trajectory of Objects)

The trajectory of a moving object is the subject of the fuzzy semantic query 2. In fuzzy spatiotemporal query the value ranges for meteorological measurements defined the object. In this query the object (cold weather) is defined semantically in FKB. The input screen is shown in Figure 54.

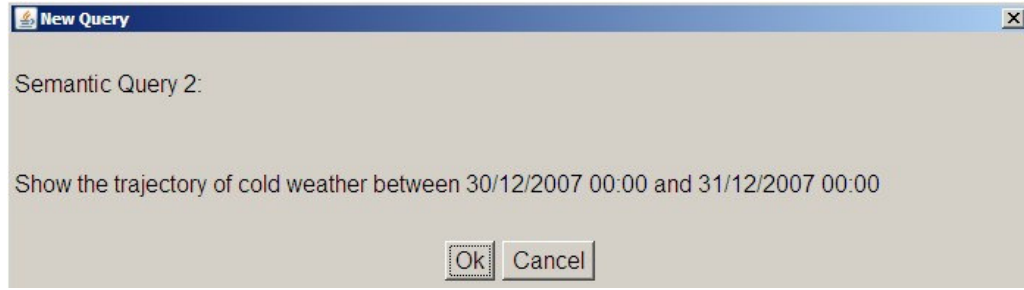


Figure 54: Fuzzy Semantic query 2 (trajectory of objects) input screen

The cold weather trajectory between 30.12.2007 00:00 and 31.12.2007 00:00 is queried. The algorithm in Figure 55 is applied for temperature range (< 5). Cold weather is not just the low temperature but wind and humidity also affect the cold feeling. So after fuzzy spatiotemporal algorithm result is obtained some more refinement is needed. FKB finds cold weather measurements by temperature, temperature and wind speed, temperature and humidity. The final result from FKB is displayed in Figure 56.

Fuzzy Semantic query 2 (trajectory of objects) algorithms:

Input: *R*-tree, the spatiotemporal object type, temporal interval*

Output: *The trajectory of the object*

1. *Set temporal (30.12.2007 00:00 and 31.12.2007 00:00) and temperature interval (< 5)*
2. *Create an empty result list res*
3. *Apply fuzzy spatiotemporal algorithm for temperature*
4. *Send result list to FKB queue*

Figure 55: Fuzzy Semantic query 2 (trajectory of objects) algorithm

5. *Run FKB engine*

Fire the coldweather rule

If (*temperature* < 0) ***or***

(*temperature* < 5 && *wind speed* > 20) ***or***

(*temperature* < 5 && *humidity* > 80) ***then***

Tag measurement as cold weather

end if

6. *Get objects from output queue of FKB*

7. *Display trajectory in the main map and results in detail window*

Figure 55: Fuzzy Semantic query 2 (trajectory of objects) algorithm (cont'd)

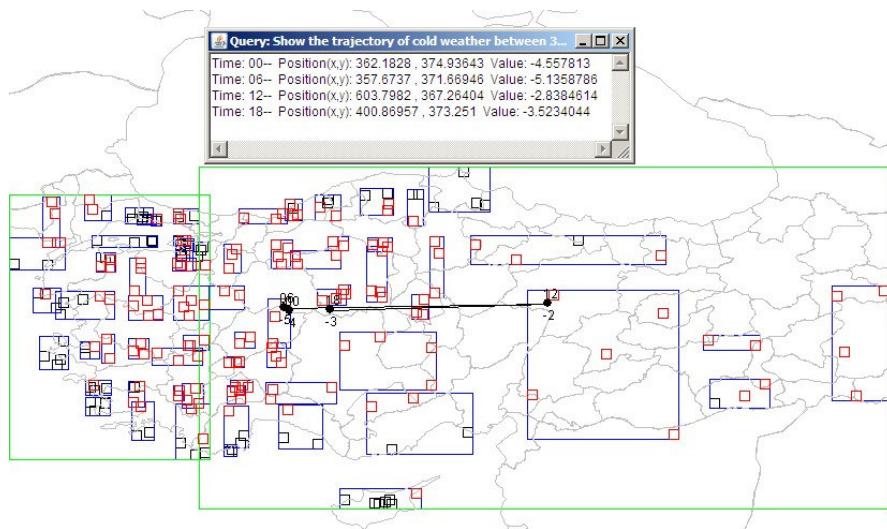


Figure 56: Fuzzy Semantic query 2 (trajectory of objects) results

In Figure 56, the cold weather data on data nodes are displayed with red squares. The average values of the temperature measurements and the temporal value are displayed over and below the black dots. The black lines between the dots show the trajectory of the cold weather. In the upper window each temporal instance is displayed in detail such as temporal instance, position and the average value of the temperatures.

7.6.3 Fuzzy Semantic Query 3 (k Highest Measurements)

The fuzzy semantic query 3 finds the k highest measurement of a parameter from a reference point. This might be helpful especially planning a trip. The meteorological parameter can be flexible but in this example we concern with the temperature. The query is defined in input screen (see Figure 57).

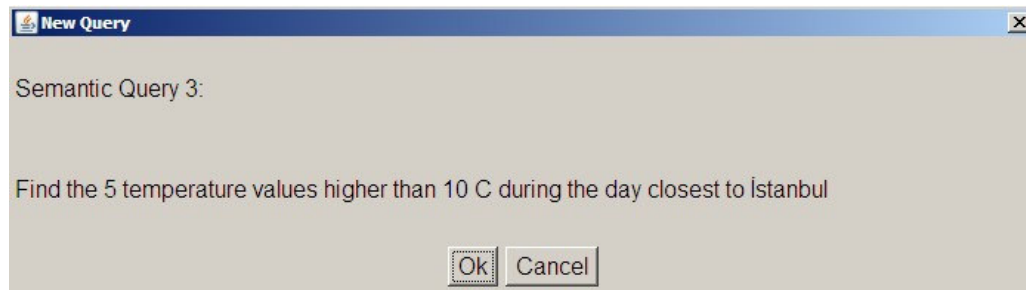


Figure 57: Fuzzy Semantic query 3 (k Highest Measurements) input screen

The input specifies İstanbul city as the reference point and inquires “5 temperature values higher than 10 C during the day and closest to İstanbul city”.

Fuzzy Semantic query 3 (k Highest Measurements) algorithm:

Input: *R*-tree, temporal range, constraint value (10 C), reference city (İstanbul)*

Output: *The objects which satisfy the constraints*

1. *Set value(>10) , temporal (during the day->06:00 && <18:00) and spatial range*
2. *Create an empty result list res*
3. *Apply fuzzy spatiotemporal algorithm*
4. *Set p_ist ← point for İstanbul*
5. ***for*** *each entry in res*
 - find distance of objects to p_ist*
 - Add to 5 closest list****end for***
6. *Display results in the main map and detail window*

Figure 58: Fuzzy Semantic query 3 (k Highest Measurements) algorithm

The fuzzy semantic query 3 runs fuzzy spatiotemporal query algorithm for temperature range (> 10) and temporal interval (06:00 and 18:00) as the daytime. The semantic part which calculates the k highest value comes in step 4. A reference point for İstanbul city is created and the distance function is applied in the loop for 5 nearest neighbor. The results are displayed in Figure 59.

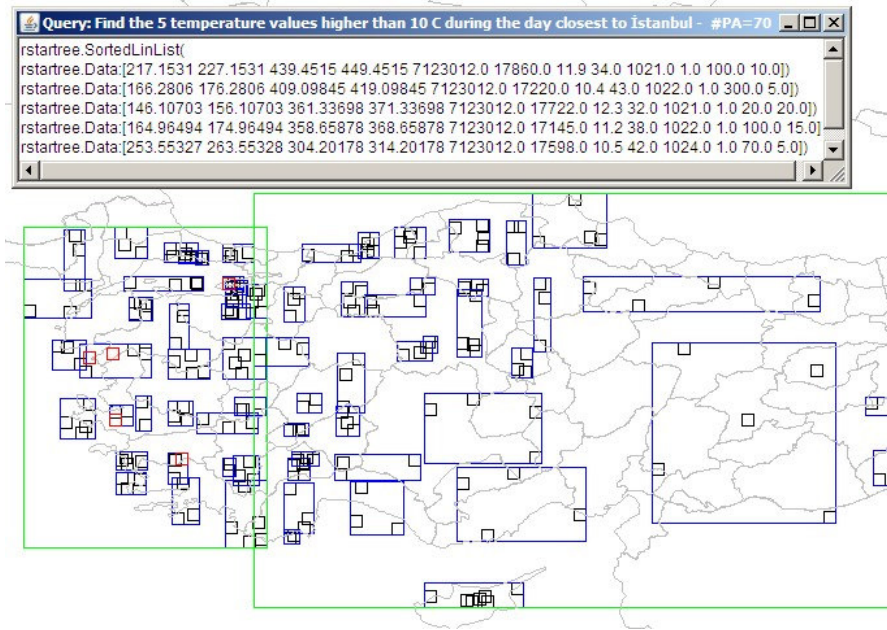


Figure 59: Fuzzy Semantic query 3 (k Highest Measurements) result

In Figure 59, five stations which measured higher than 10 C and closest to İstanbul are displayed as the red squares. The measurements details are displayed in the upper window.

7.6.4 Fuzzy Semantic Query 4 (Agricultural Risky Zones)

Fuzzy Semantic query 4 inquires an important meteorological and agricultural parameter that is frosty zones. The query finds the frosty zones with several severity degrees. Input screen with details of query is displayed in Figure 60.

According to the input parameters, the algorithm in Figure 61 will find the frosty risky zones during the night.

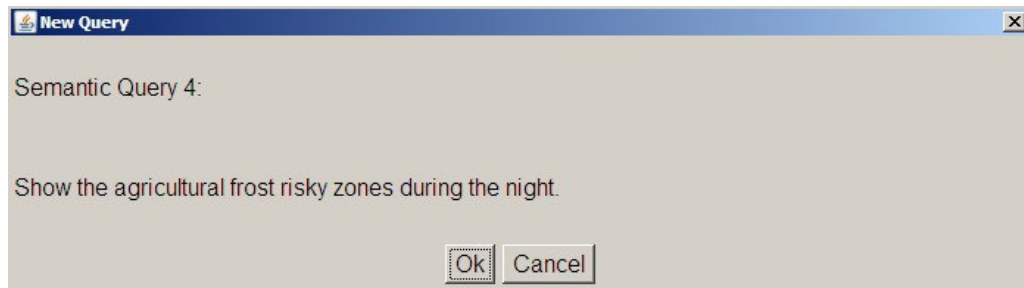


Figure 60: Fuzzy Semantic query 4 (Agricultural risky zones) input screen

Fuzzy Semantic query 4 (Agricultural risky zones) algorithm:

Input: Temperature range for frost, temporal range ($>00:00$ and $<06:00$), spatial range

Output: The frost levels of meteorological stations are displayed

1. Create an empty list res
2. Apply spatiotemporal algorithm with input parameters
3. Put the results to the input queue of FKB
4. Apply the frost rule

Classify the frost level of each record

Set object's attribute with frost level

3. Display result

Display each level with different gray scale

Display legend

Display result window

Figure 61: Fuzzy Semantic query 4 (Agricultural risky zones) algorithm

The fuzzy semantic query 4 runs fuzzy spatiotemporal query algorithm for temperature range (< 00) and temporal interval ($00:00$ and $06:00$) as the nighttime. The semantic part which calculates the degree of frosty zones is performed in FKB. FKB executes the frost rule and classifies each frost value. The results are displayed in Figure 62.

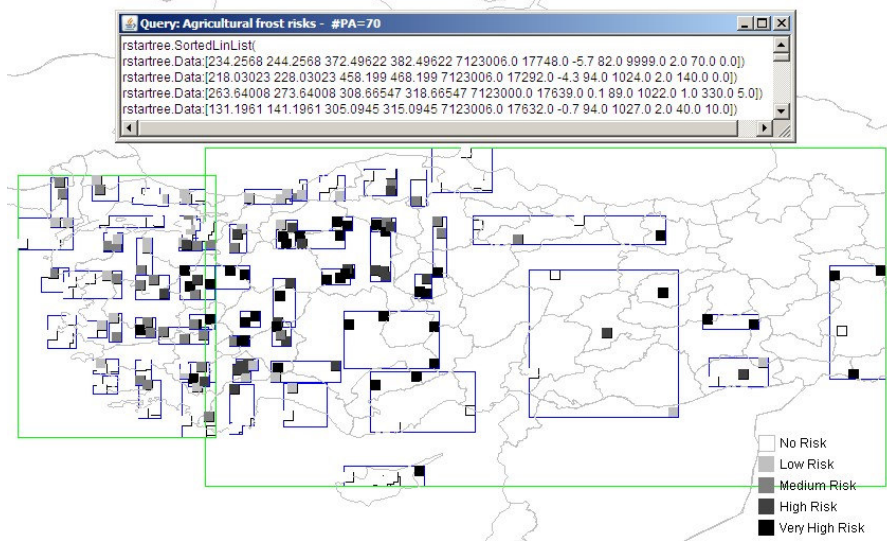


Figure 62: Fuzzy Semantic Query 4 (Agricultural risky zones) result

The frosty zones are displayed with different grayscale colors according to the frost levels (from No Risk to Very High Risk) in Figure 62.

7.6.5 Fuzzy Semantic Query 5

In this query, the relation between meteorological parameters and the altitude is queried. The query finds whether parameters increase or decrease in a given

temporal and spatial interval as the altitude changes. Input screen with details of query is displayed in Figure 63.

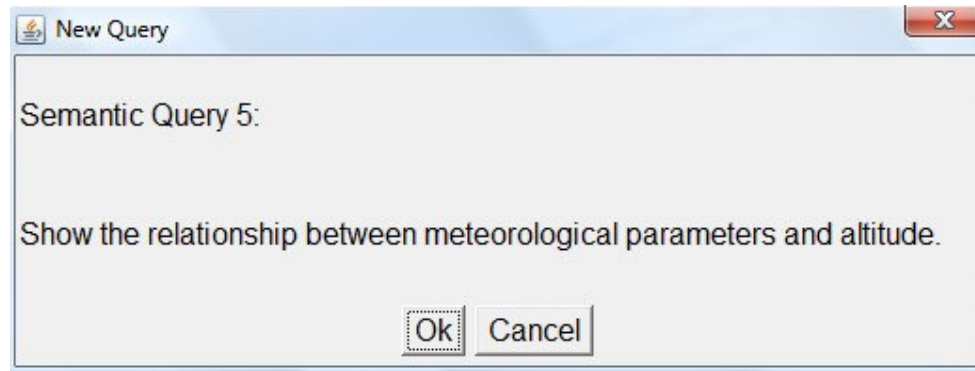


Figure 63: Fuzzy Semantic query 5 input screen

The algorithm in Figure 64 will find the relationship between the meteorological parameters (i.e. temperature, humidity, pressure, wind speed) and altitude.

Fuzzy Semantic query 5 algorithm:

Input: An R^* -tree with altitude values and meteorological parameters loaded, temporal range, spatial range

Output: The average value of meteorological parameter for each altitude threshold and the average difference between levels of altitude are displayed

1. Create an empty list *res*
2. Apply spatiotemporal algorithm with spatial, temporal ranges given

Figure 64: Fuzzy Semantic Query 5 algorithm

3. Find the average value of parameter for each threshold value (i.e. for each 300 meters

4. Display result

For each altitude value

Display altitude

Display average value of parameter

Display average difference

End for

Figure 64: Fuzzy Semantic Query 5 algorithm (cont'd)

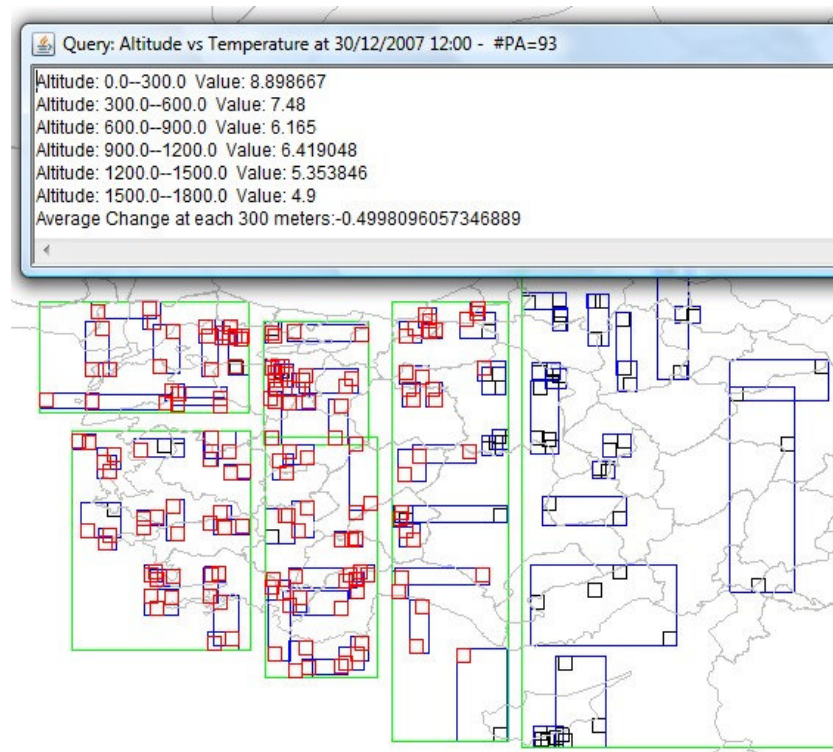
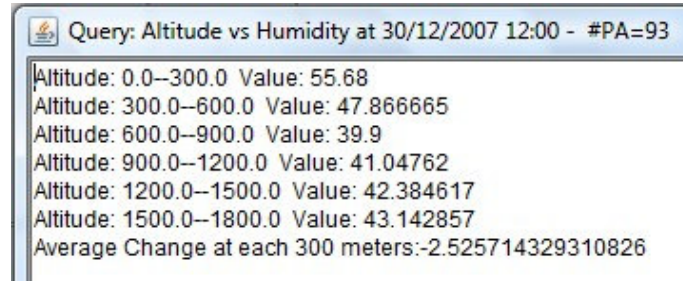
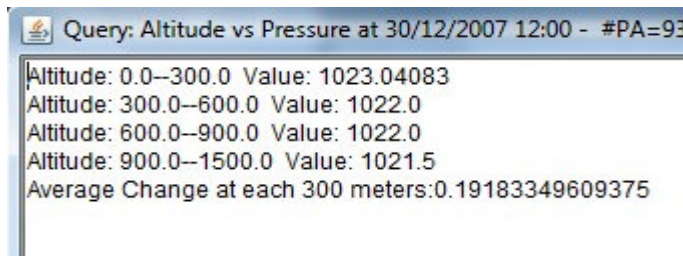


Figure 65: Fuzzy Semantic Query 5 result

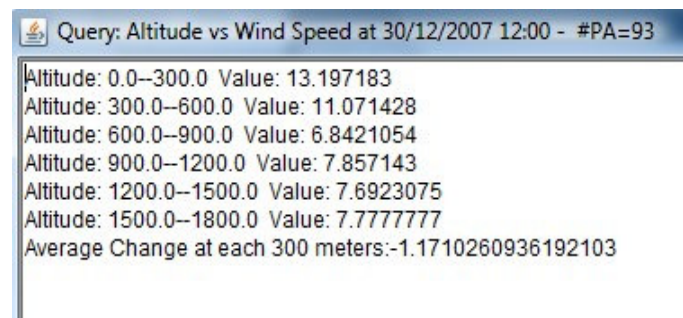
The fuzzy semantic query 5 runs fuzzy spatiotemporal query algorithm for instance for a temperature range between -30 and +30 °C and in a spatial range (100,200)-(400,600) to cover western part of the country and finally on 30.12.2007 12:00 time. The results are displayed in Figure 65.



(a)



(b)



(c)

Figure 66: Altitude vs. meteorological parameters variations (a)Humidity (b)Pressure (c)Wind speed.

The red squares in the window shows the measurements for the given spatial, temporal and value range. The second window above shows the results. The results indicate that temperature decreases slightly at each 300 meters on the average 0.5 °C. The other parameter variations are shown in Figure 66.

The result of the fuzzy semantic query 5 indicates that for the input parameters we specified:

- Temperature decreases on the average 0.5 °C,
- humidity decreases % 7-8 at every 300 meters up to 900 meters then steady
- Pressure doesn't change
- wind speed is around 13 knots up to 600 meters and then decreases to 7 knots

7.7 Experimental Evaluation

In this section, we experimentally study the effectiveness of the architecture components. First, we observe the scalability of the system. The crisp queries and fuzzy semantic queries are run with real meteorological data. The number of records is as many as 80.000 records. Next we show the effect of using Enhanced R*-tree by comparing R*-tree. Since we adapted a secondary fuzzy index some fuzzy semantic queries are run for the performance evaluation. Finally, the effect of third dimension on the tree is evaluated. The application is run on a laptop with Windows Vista operating system, 4G RAM, JDK 1.6 and Net Beans 6.5.1.

7.7.1 The Scalability of the Application

The scalability of the system is tested with crisp, fuzzy and semantic queries. Firstly the crisp spatial queries (point, range, circle, kNN and ring) are tested with the data that belongs to fifteen days between 30.12.2007 00:00 and 15.01.2008 12:00. In

Figure 67, the number of node access and number of records are depicted for each type of query as well as the average.

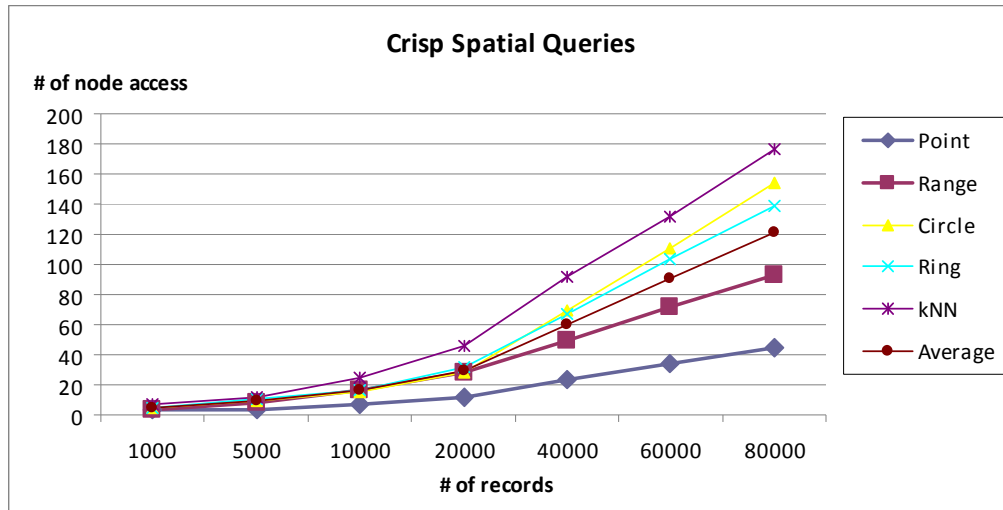


Figure 67: The scalability of the system by Crisp Spatial Queries

The queries perform quite similar to each other. The performance of each query is good even for the high number of records. So the system is scalable for the crisp spatial queries.

Next fuzzy spatial queries are tested. Fuzzy Spatial Relation (FSR) and Fuzzy Spatiotemporal (FST) queries are tested with the same data set. In Figure 68, the number of node access and number of records are depicted for each type of query as well as the average.

Both queries and the average are close and application runs smoothly for even high number of records. Finally, fuzzy semantic queries are tested. In Figure 69, the

number of node access and number of records are depicted for each type of semantic query (SQ1 to SQ5) as well as the average. The details of the semantic queries are already presented in Section 7.6.

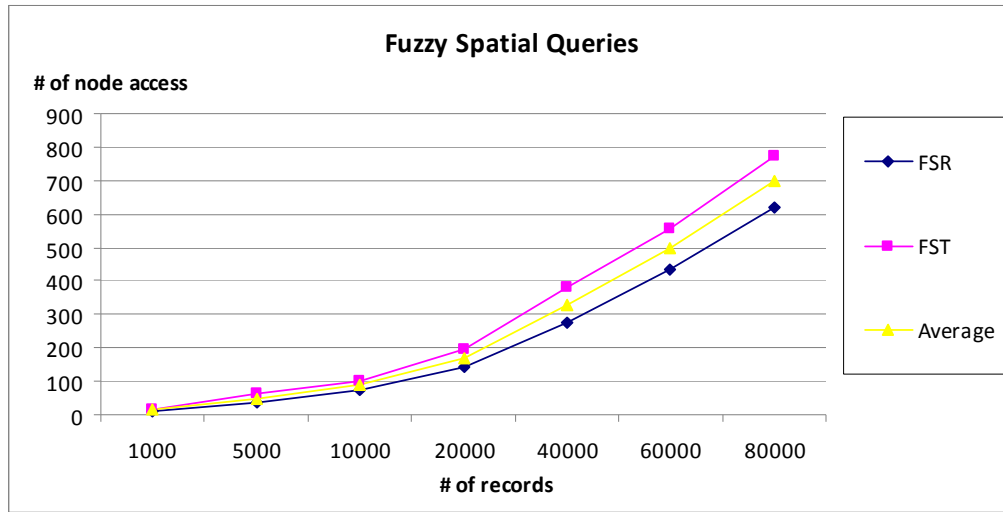


Figure 68: The scalability of the system by Fuzzy Spatial Queries

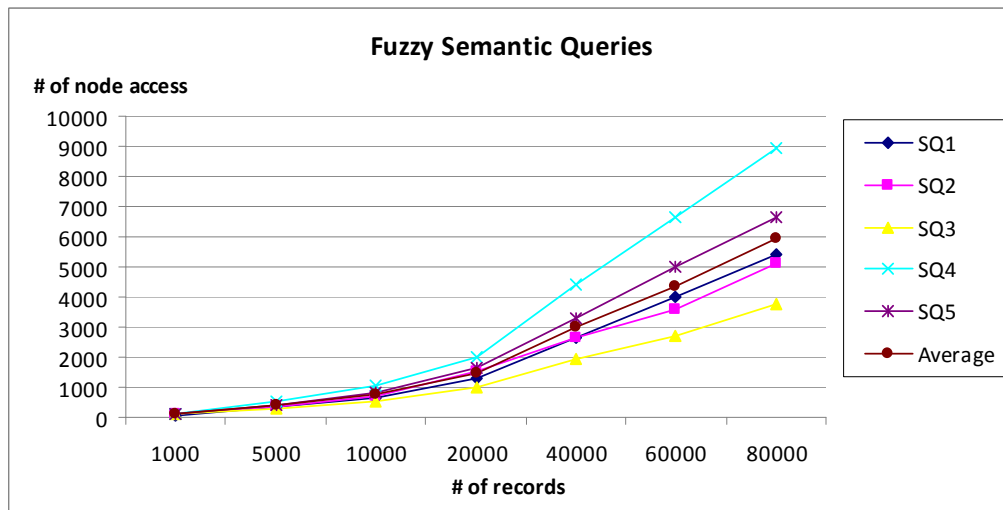


Figure 69: The scalability of the system by Fuzzy Semantic Queries

The SQ4 (Agricultural Risk Zones) has more node access, because it checks every record and performs full scan. But the other queries are close to the average. In the next section the Enhanced R*-tree and R*-tree are tested and the effect of adaptation is shown.

7.7.2 The Enhanced R*-tree

In our implementation the R*-tree is enhanced to support fuzzy semantic spatiotemporal queries. The basic form of R*-tree uses rectangles to organize spatial data. In Enhanced R*-tree we adapt a secondary index which includes the range values for all attributes at the leaf nodes. Before comparing performance of queries using Enhanced R*-tree and basic R*-tree let's check the building cost of R*-tree and enhanced R*-tree. In Figure 70 and Figure 71 the elapsed times of building times for R*-tree and Enhanced R*-tree are depicted.

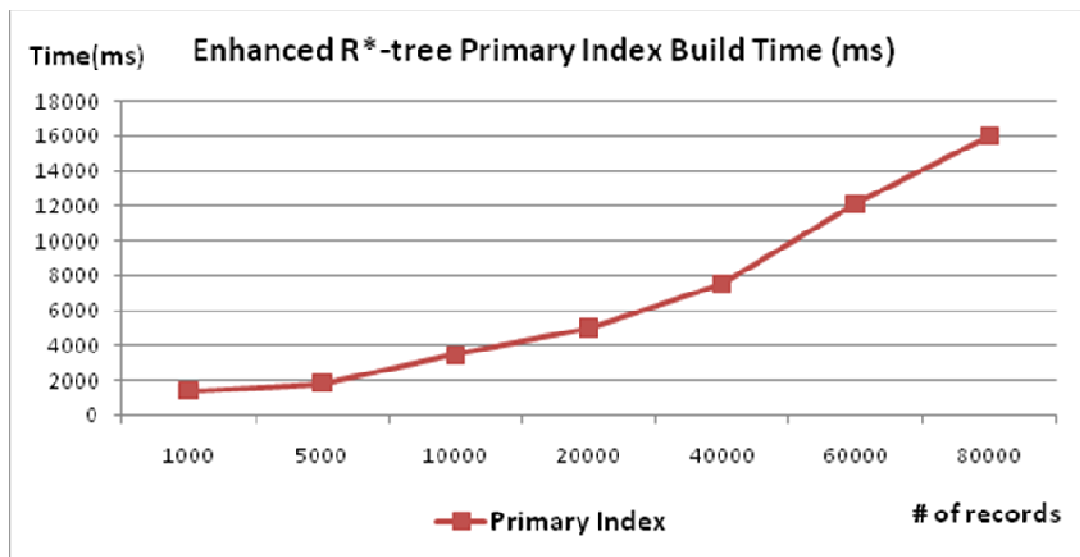


Figure 70: The primary index build time

In Figure 71 the cost of building the Enhanced R*-tree, which uses MBRs as the primary index is shown. The cost of insertion increases linearly. In the following figure the cost of adapting a secondary index is shown:

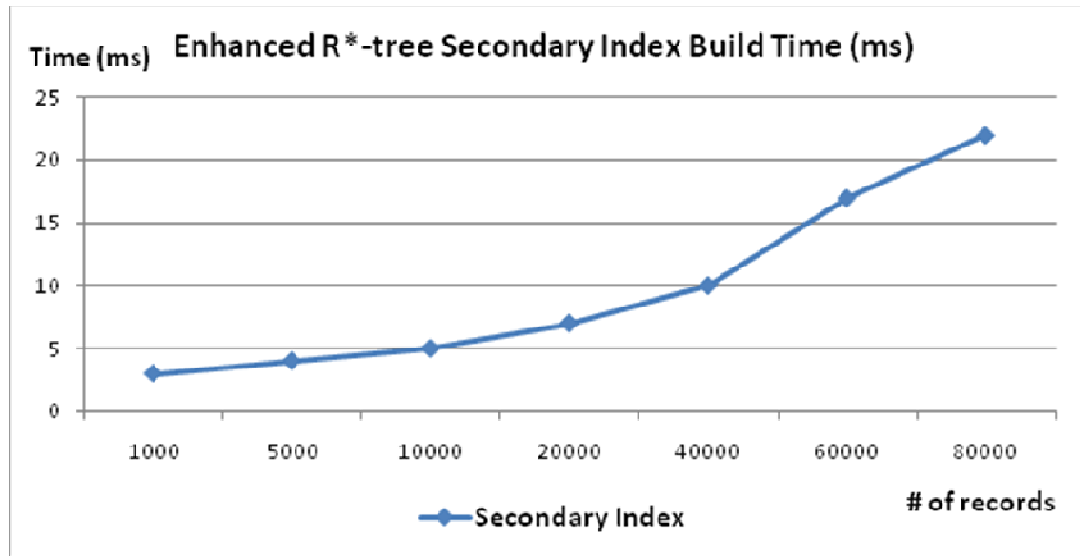


Figure 71: The secondary index build time

The cost of building a secondary index in addition to the primary one seems very negligible. While the main index takes 2 to 16 seconds to build, secondary index built takes only 25 milliseconds for 80.000 records. This is because the secondary index is built on already organized tree so no split or reinsert occurs and one full scan is enough and no file I/O is required.

In performance work, we run some of the queries by using classical R*-tree and Enhanced R*-tree. The meteorological data is obtained from Meteorology Service for the dates between 30.12.2007 00:00 and 15.01.2008 12:00. The data is partitioned into bulks (i.e. 1.000, 5.000, 10.000, 20.000, 40.000, 60.000 and 80.000).

In each run number of nodes accessed is measured. In the first query, the fuzzy spatiotemporal query which is described in Section 7.5.2 is run with the meteorological data sets. The input parameters are selected as follows: temperature measurements between 10-15 °C, spatial range (100,200)-(600,600) and temporal range 30.12.2007 00:00 to 31.12.2007 00:00. The results are depicted as a graphic in Figure 72. The number of node access is close up to 10.000 records but the gap increases as the number of records increases. The Enhanced R*-tree performs better than R*-tree in general.

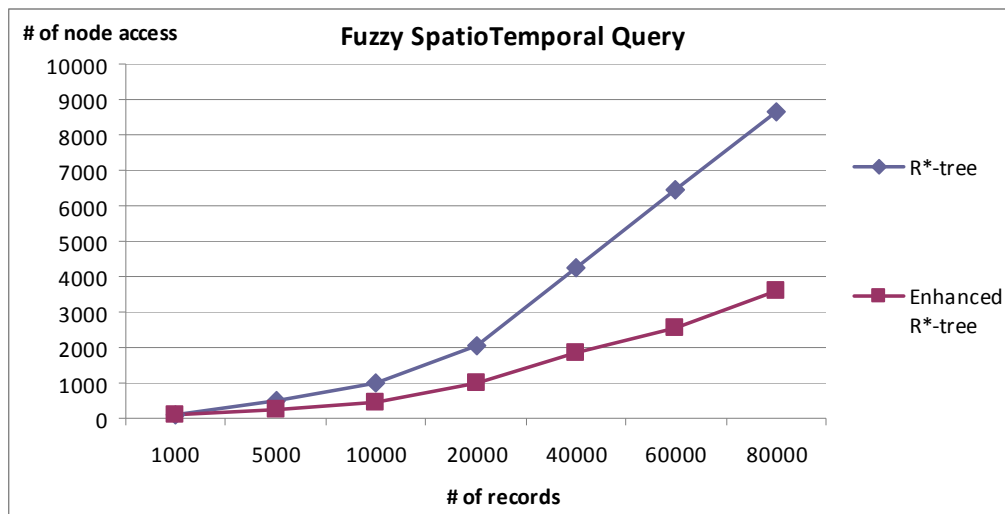


Figure 72: Fuzzy Spatiotemporal Query run with R*-tree and Enhanced R*-tree

Next we run semantic query 2 which finds trajectory of objects that the details of the query algorithm is shown in Section 7.6.2. The query runs for the trajectory of humidity values between % 40-42 in the spatial range (0,0)-(1024,800) and in temporal range 30.12.2007 00:00 to 31.12.2007 00:00. The results are depicted as a graphic in Figure 73.

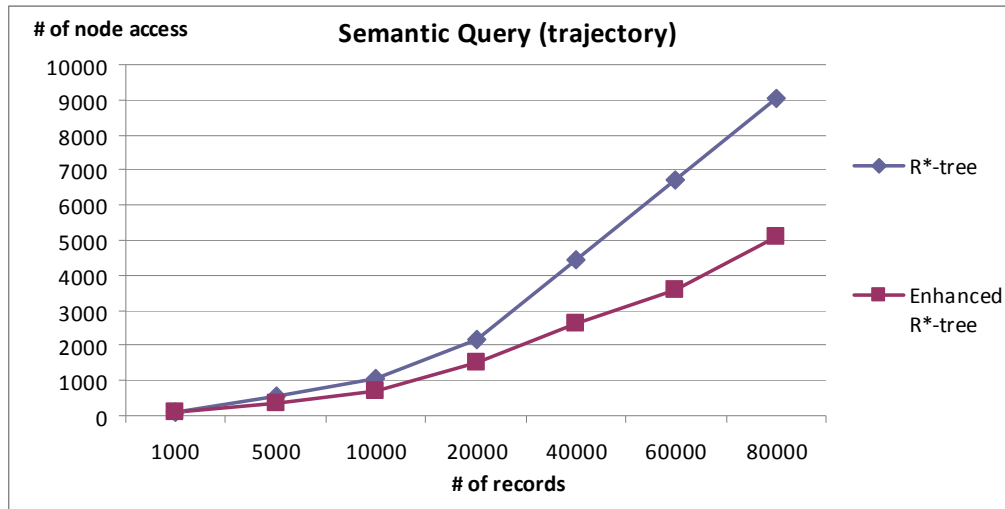


Figure 73: Semantic Query (trajectory) run with R*-tree and Enhanced R*-tree

Both index structures perform quite close up to 20.000 records. The Enhanced R*-tree runs better after 20.000 records. On the average the Enhanced R*-tree performs better than R*-tree.

Finally we run Fuzzy Semantic Query 3 (k-highest measurements which is described in Section 7.6.3. The query finds five (k=5) highest temperature measurements close to Istanbul. The resulting graphic obtained by the number of node access are shown in Figure 74.

The graphic shows similar figures as the previous runs. So it is verified that the Enhanced R*-tree runs better for fuzzy and semantic queries especially for the number of records higher than 10.000-20.000.

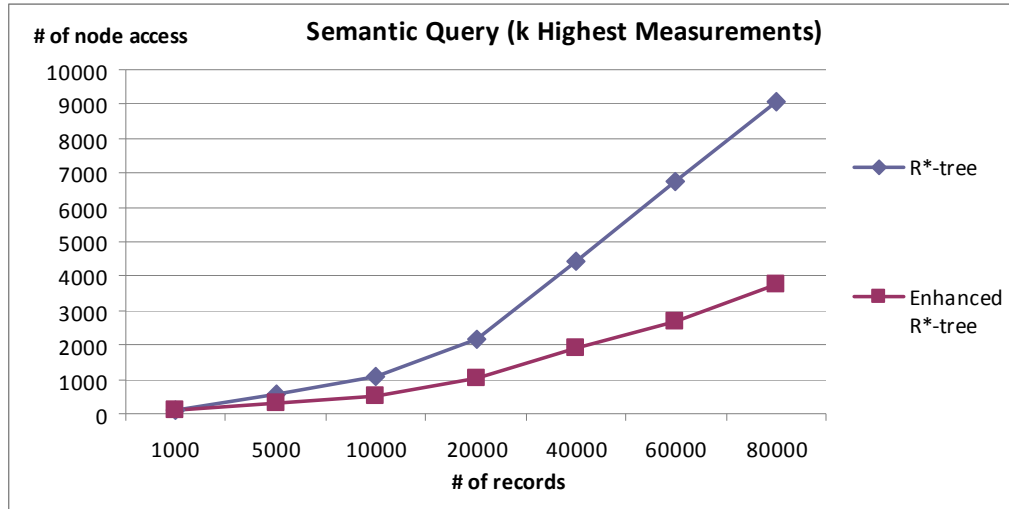


Figure 74: Semantic Query performance of R*-tree and Enhanced R*-tree

7.7.3 The Effect of Third Dimension

The altitude data forms the third dimension in the application. Altitude data is firstly included in the fuzzy index structure. So it is kind of other meteorological attributes like temperature, pressure, etc. Then it is used as the primary organizing attribute in the Enhanced R*-tree. We think that it may be implemented either as secondary or primary index. Since our data and query types mostly belong to ground we included in the secondary index. But we evaluate the effect of altitude usage as a third dimension as a primary index on tree building and query run times as well.

The major effect of altitude data as the primary indexing attribute is the increasing number of inner nodes. This is reflected in Figure 75. Since insertion algorithm considers a third dimension more rectangles are needed to group the data. The number data nodes are slightly affected by third dimension (see Figure 76). In Figure 77, the increase in the inner nodes is visualized.

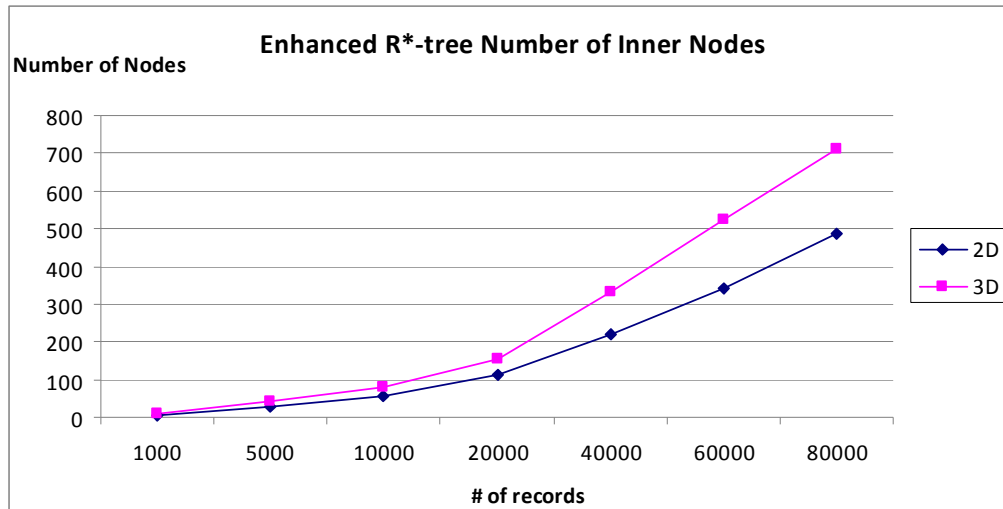


Figure 75: Number of inner nodes in Enhanced R*-Tree

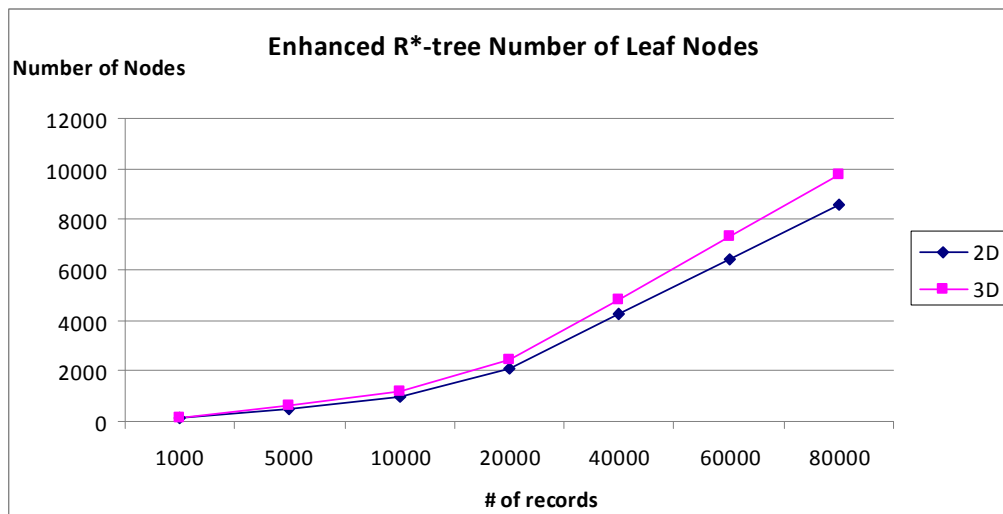


Figure 76: Number of data nodes in Enhanced R*-Tree

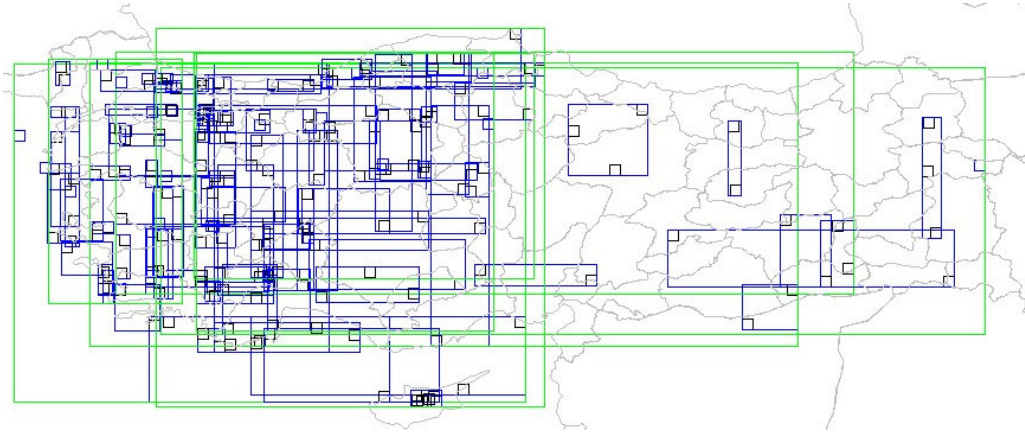


Figure 77: The nodes of three dimensional Enhanced R*-Tree

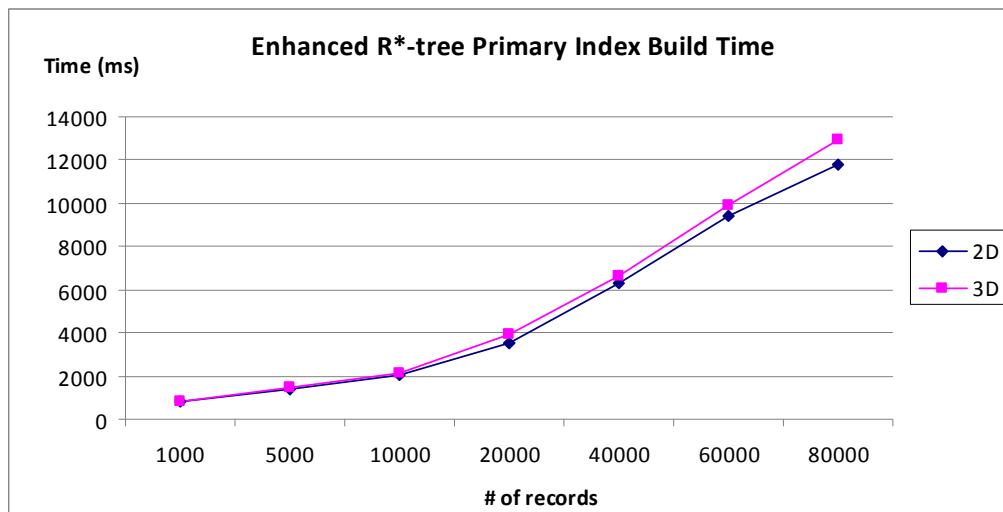


Figure 78: Primary index building time in Enhanced R*-Tree

Although third dimension increases the inner nodes, it doesn't affect the building time of Enhanced R*-Tree. While primary index building times are almost same for two dimensional (2D) and three dimensional (3D) versions, secondary index building times are negligible (see Figure 78 and Figure 79).

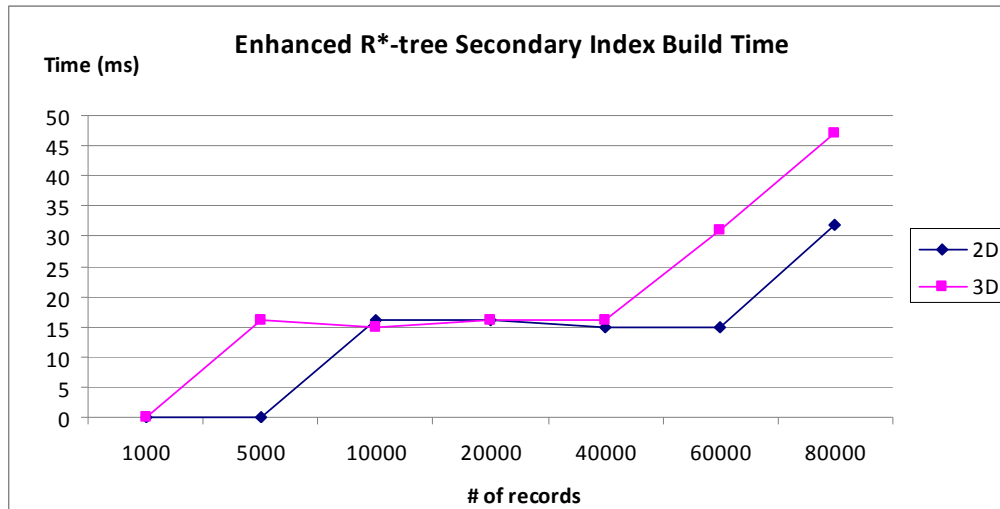


Figure 79: Secondary index building time in Enhanced R*-Tree

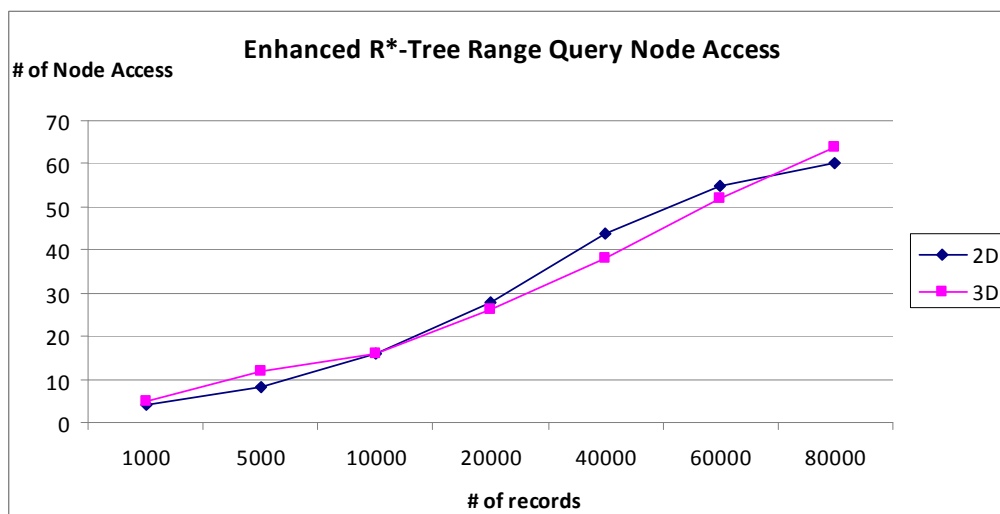


Figure 80: Range Query performance of Enhanced R*-Tree (1)

Two and three dimensional Enhanced R*-Trees are tested with crisp and fuzz semantic queries. In Figure 80 and 81 number of node accesses and execution times are shown for range query. In the three dimensional range query an altitude range is also input by the user.

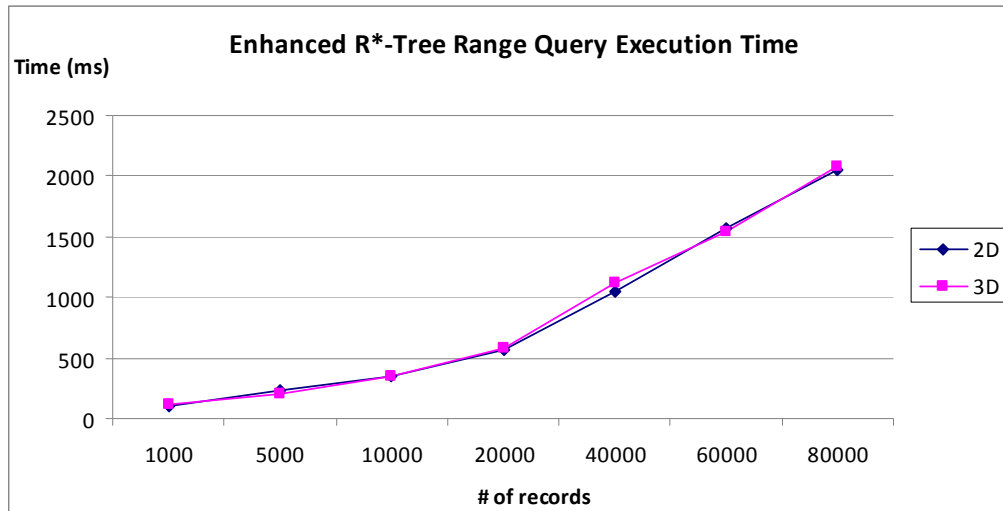


Figure 81: Range Query performance of Enhanced R*-Tree (2)

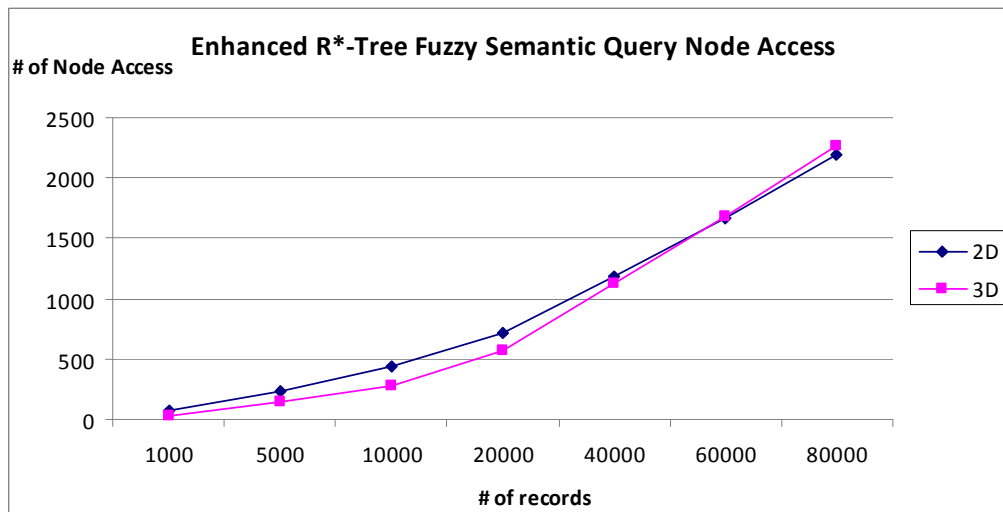


Figure 82: Fuzzy Semantic Query performance of Enhanced R*-Tree (1)

The performance of Enhanced R*-Tree nearly same for range query. In Figure 82 and Figure 83 the results of Fuzzy Semantic Query is shown. In this query, the meteorological parameter temperature change by altitude is measured.

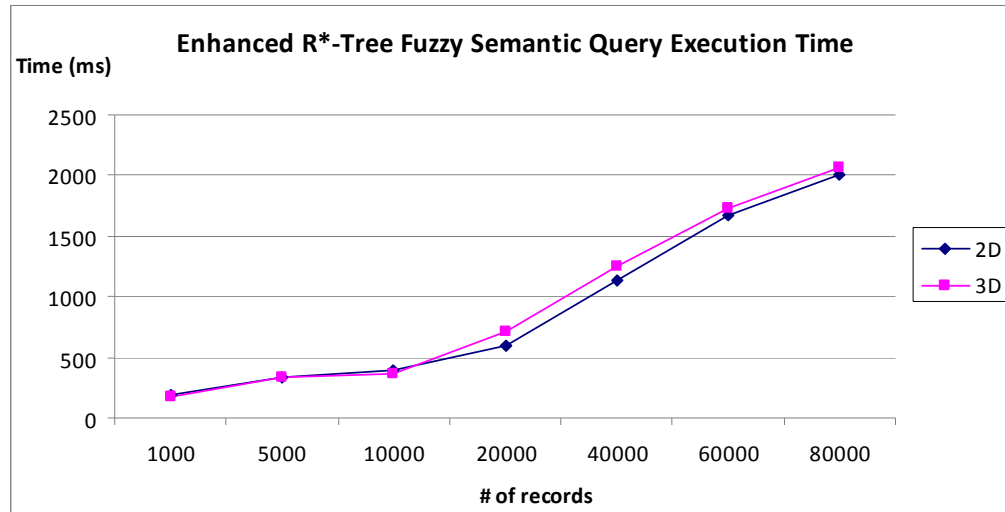


Figure 83: Fuzzy Semantic Query performance of Enhanced R*-Tree (2)

The performance of three dimensional Enhanced R*-Tree is slightly worse than two dimensional Enhanced R*-tree. As a result, number of inner nodes increase by using the altitude as the primary indexing attribute. This affects index building and query execution times so that three dimensional tree executes worse than two dimensional tree. But in terms of number of node access, two dimensional tree performs worse than three dimensional tree.

CHAPTER 8

CONCLUSIONS

In this study we have introduced a generic spatiotemporal data model and a querying mechanism for spatiotemporal databases. We presented our method, designed to handle uncertainty in spatiotemporal database applications. We used an application, involving meteorological objects with some spatial and temporal attributes, as an example. The proposed mechanism has been implemented as a proof-of-concept prototype.

In the scope of this work, spatial objects, relations including temporality are incorporated into a generic model. Based on the generic model meteorological phenomena and geographic data are modeled as spatiotemporal objects. These objects can move and evolve in time. In addition, the meteorological and geographic man made objects may have spatial relations. The model and fuzzy spatiotemporal querying mechanisms are presented formally. The crucial decision was to integrate the model with a fuzzy knowledge base allowing a fuzzy deduction and querying capability to handle complex data and knowledge. As a result, we are able to handle spatiotemporal queries (position, spatial properties and spatial relationships).

We also adapted an index structure for efficient querying and verified that with performance runs using three dimensional data. Since our queries mostly deal with ground information we used the third dimension (altitude) in the secondary index. For the queries related with the atmospheric data it could be better to use third dimension as the primary index.

Spatiotemporal data modeling and querying require further research. The model and the method presented in this thesis should be applied to other fields, such as wireless sensor networks and multimedia, to gain more insight into fuzzy spatiotemporal modeling and querying.

REFERENCES

- [1] Beckmann N. and Seeger B., “A revised R*-tree in comparison with related index structures”, Proceedings of the 35th SIGMOD International Conference on Management of Data, Rhode island USA, pages 799-812, 2009.
- [2] Booch G., Rumbaugh J. and Jacobson I., “The Unified Modeling Language User Guide”, The (2nd Edition) (Addison-Wesley Object Technology Series), 2005.
- [3] Bordogna G., Chiesa S. and Geneletti D., “Linguistic modelling of imperfect spatial information as a basis for simplifying spatial analysis”, Information Sciences, 176, 4 (Feb. 2006) pp.366-389, 2006.
- [4] Bordogna G., Leporati A., Lucarella D. and Pasi G., “The fuzzy object oriented database”, Recent Issues on Fuzzy Databases, Springer-Verlag, 2000.
- [5] Bordogna G., Pasi G. and Psaila G., “Evaluating Uncertain Location-Based Spatial Queries”, SAC’08, March 16-20, 2008, Fortaleza, Cear , Brazil.
- [6] Cheng T., Molenaar M. and Lin H., “Formalizing fuzzy objects from uncertain classification results”, International Journal of Geographical Information Science, vol. 15, no. 1, pp.27-42, 2001.
- [7] Claramunt C. and Theriault M., “Fuzzy semantics for direction relations between composite regions”, Information Sciences, 160, pp.73-90, 2004.
- [8] Cobb M.A. and Petry F.E., “Modeling spatial relationships within a fuzzy framework”, Journal of the American Society For Information Science 49(3):pp.253-266, 1998.
- [9] Colomb R.M., “Deductive Databases and Their Applications”, Taylor & Francis, 1998.
- [10] Cranston B. C. and Samet H., “Efficient Position-Independent Iconic Search Using An R-Theta Index”, *ACM-GIS’06*, November 10–11, 2006, Arlington, Virginia, USA

- [11] Du S., Qin Q., Wang Q. and Ma H., “Evaluating structural and topological consistency of complex regions with broad boundaries in multi-resolution spatial databases”, *Information Sciences*, 178, pp.52-68, 2008.
- [12] Dubois D. and Jaulent M.C., “A general approach to parameter evaluation in fuzzy digital pictures”, *Pattern Recognition Letters*, pp.251–259, 1987.
- [13] Egenhofer M., Clementini E. and Felice P., “Topological relations between regions with holes”, *International Journal of Geographical Information Systems* 8 (2), pp.129-144, 1994.
- [14] ESRI Products overview, http://www.esri.com/products/index.html#desktop_gis_panel, April 2010.
- [15] Fisher P., Arnot C., Wadsworth R. and J.J. Wellens, “Detecting change in vague interpretations of landscapes”, *Ecological Informatics* 1, pp.163-178, 2006.
- [16] Frihida A., Marceau D.J. and Theriault M., “Spatiotemporal object-oriented data model for disaggregate travel behavior”, *Transactions in GIS*, 6(3), pp.277-294, 2002.
- [17] Griffiths T., Fernandes A., Paton N., and Barr R., “The TRIPOD spatio-temporal data model”, *Data and Knowledge Engineering*, 49(1), pp.23-65, 2003.
- [18] Gyseghem N.V. and Caluwe R.D., “Imprecision and uncertainty in the UFO database model”, *Journal of the American Society for Information Science*, Volume 49, Issue 3 , pp.236 – 252, 1999.
- [19] Hadzilacos T. and Tryfona N., “An extended entity-relationship model for geographic applications”, *SIGMOD Record*, Vol. 26, No. 3, 1997 .
- [20] Iwerks G.S., Samet H. and Smith K.P., “Maintenance on K-nn and spatial join queries on continuously moving points”, *ACM Transactions on Database Systems*, Vol. 31, No.2, pp.485-536, 2006.
- [21] Jackson D., “Software abstractions: logic, language, and analysis”, The MIT Press, ISBN:0262101149, 2006.
- [22] Joomla, “R-tree Portal - Home”, <http://www.rtreeportal.org/>, Last visited (28/05/2010).

- [23] Kalashnikov D. V., Ma Y., Mehrotra S., Hariharan R. and Butts C., “Modeling and Querying Uncertain Spatial Information for Situational Awareness Applications”, *ACM GIS’06*, November 10–11, 2006, Arlington, Virginia, USA
- [24] Kanjilal V., Liu H. and Schneider M., “Plateau Regions: An Implementation Concept for Fuzzy Regions in Spatial Databases and GIS”, *IPMU, LNAI 6178*, 624–633, 2010.
- [25] Koyuncu M. and Yazici A., “IFOOD: An Intelligent Fuzzy Object-Oriented Database Architecture”, *IEEE Trans. on Knowledge and Data Engineering*, pp.1137–1154, 2003.
- [26] Lee J., Xue N.L., Hsu K.H., Yang S.J., “Modeling imprecise requirements with fuzzy objects”, *Information Sciences*, 118, pp.101–119, 1999.
- [27] Liu M., “Deductive Database Languages: Problems and Solutions”, *ACM Computing Surveys*, vol. 31, no. 1, pp.27–62, 1999.
- [28] Lu C.T., Kou Y., Zhao J. and Chen L., “Detecting and tracking regional outliers in meteorological data”, *Information Sciences*, 177, pp.1609–1632, 2007.
- [29] Marin N., Medina J.M., Pons O. and Vila M.A., “Object resemblance in a fuzzy object-oriented context”, *Proceedings of 2002 IEEE International Conference on Fuzzy Systems, Honolulu (EEUU)*, 2002.
- [30] Martins B., Silva M. J. and Andrade L., “Indexing and Ranking in GeoIR Systems”, *GIR’05*, November 4, 2005, Bremen, Germany.
- [31] Papadias D., Tao Y., Mouratidis K. and Hui C.K., “Aggregate nearest neighbor queries in spatial databases”, *ACM Transactions on Database Systems*, Vol.30, No. 2, pp. 529–576, June 2005.
- [32] Pauly A., and Schneider, M., “Topological predicates between vague spatial objects”, In *9th Int. Symp. On Spatial and Temporal Databases (SSDT)*, *Lecture Notes in Computer Science (LNCS)*, pp.418–432. Springer, 2005.
- [33] Pelekis N., Theodoulidis B., Kopanakis I., and Theodoridis Y., “Literature review of spatio-temporal database models”, *The Knowledge Engineering Review*, Vol.19:3, pp.235–274, 2004.
- [34] Peuquet D., “Making space for time: Issues in Space-Time Data Representation”, *GeoInformatica* 5(1), pp.11–32, 2001.

- [35] Plewe B., "The nature of uncertainty in historical geographic information", *Transaction in GIS*, pp.431-456, 2002.
- [36] Praing R. and Schneider M., Modeling Historical and Future Movements of Spatio-Temporal Objects in Moving Objects Databases, *CIKM'07*, November 6–8, 2007, Lisboa, Portugal.
- [37] Renolen A., "Temporal Maps and Temporal Geographical Information Systems (Review of Research)", Department of Surveying and Mapping, The Norwegian Institute of Technology, 1997.
- [38] Rizzi S., Abello A., Lechtenbörger J. and Trujillo J., "Research in Data Warehouse Modeling and Design:Dead or Alive?", *DOLAP'06*, November 10, 2006, Arlington, Virginia, USA.
- [39] Sandia National Laboratories, "Jess, the Rule Engine for the Java Platform", <http://www.jessrules.com/>, Last visited (28/05/2010).
- [40] Schneider M., "Uncertainty Management for Spatial Data in Databases: Fuzzy Spatial Data Types", in 6th Int. Symp. on Advances in Spatial Databases, LNCS 1651, pp.330–351. Springer-Verlag, 1999.
- [41] Schneider M., "A Design of Topological Predicates for Complex Crisp and Fuzzy Regions", 20th International Conference on Conceptual Modeling, Yokohama, Japan, November 27-30, 2001.
- [42] Schneider M. and Behr T. "Topological Relationships Between Complex Spatial Objects", *ACM Transactions on Database Systems*, Vol.31, No. 1, pp. 39-81, March 2006.
- [43] Schults C. P. L., Guesgen H. W. and Amor R., "Computer-Human Interaction Issues when Integrating Qualitative Spatial Reasoning into Geographic Information Systems", *Chinz'06, July 6–7, 2006, Christchurch, New Zealand*
- [44] Shekhar S., Chawla S., Ravada S., Fetterer A., Yuan L. and Chang-Tien L., "Spatial Databases – Accomplishments and Research Needs", *IEEE Trans. On Knowledge and Data Engineering*, Vol. 11, No.1, pp.45-55, 1999.
- [45] Sozer A. and Yazici A., "Design and implementation of index structures for fuzzy spatial databases", *International Journal of Intelligent Systems*, Vol.22, Issue 7, pp. 805-826, 2007.

- [46] Sozer A., Yazici A., Oguztuzun H. and Tas O., “Modeling and Querying Fuzzy Spatiotemporal Databases”, *Information Sciences* 178, pp. 3665-3682, 2008.
- [47] Stell J.G., “Part and complement: fundamental concepts in spatial relations”, *Annals of Artificial Intelligence and Mathematics*, 41, pp.1-18, 2004.
- [48] Tang X., Fang Y. and Kainz W., “Fuzzy topological relations between fuzzy spatial objects”, *FSKD 2006, LNAI 4223*, pp.324-333, Springer Verlag, 2006.
- [49] Tao, Y., Xiao, X., and Cheng, R., “Range search on multidimensional uncertain data”. *ACM Trans. Datab. Syst.* 32, 3, Article 15, 54 pages, 2007.
- [50] Tryfona N. and Jensen S.J., “Using abstractions for spatio-temporal conceptual modeling”, *Proceedings of ACM SAC, Como Italy*, 2000.
- [51] Versant Corp., “db4o :: Java & .NET Object Database – Open Source Object Database, Open Source Persistence, Oodb”, <http://www.db4o.com/>, Last visited (28/05/2010).
- [52] Vidal C. and Rodriguez A., “A logical approach for modeling spatio-temporal objects and events”, *LNCS 3770*, pp.218-227, Springer-Verlag, 2005.
- [53] Warren T., Zhang L.Z. and Mount C., “Similarity Measures for Retrieval in Case-based Reasoning Systems”, *Applied Artificial Intelligence*, vol. 12,no. 4, pp.267-288, 1998.
- [54] Weidong C. and Warren D., “C-logic of complex objects”, *ACM SIGACTSIGMOD-SIGART Symp. Principles of Database Systems*, pp.369–378, 1989.
- [55] Worboys M.F., “A unified model for spatial and temporal information”, *The Computer Journal*, Vol. 37, No.1, 1994.
- [56] Yazici A. and George R., “Fuzzy Database Modeling”, Heidelberg/New York:Physica-Verlag, 1998.
- [57] Yazici A., George R. and Aksoy D., “Design and Implementation Issues in the Fuzzy Object-Oriented Data (FOOD) Model”, *Information Sciences*, Vol. 108/4, pp.241-260, 1998.

- [58] Yazici A., Zhu Q. and Sun N., “Semantic data modeling of spatiotemporal database applications”, *International Journal of Intelligent Systems*, Vol. 16, pp.881-904, 2001.
- [59] Zhan F.B. and Lin H., “Overlay of two simple polygons with indeterminate boundaries”, *Transactions in GIS*, 7(1), pp.67-81, 2003.
- [60] Zhang J., Pan H. and Yuan Z., “A Novel Spatial Index for Case based Geographic Retrieval”, *ICIS 2009*, November 24-26, 2009 Seoul, Korea.
- [61] Zhizhin M., Kihn E., Lyutsarev V., Berezin S., Poyda A. Mishin D., Medvedev D. And Voitsekhovsky D., “Environmental Scenario Search and Visualization”, *ACMGIS'07*, November 7-9, 2007, Seattle, WA.
- [62] Zhou Y. and Murata T., “Petri net model with fuzzy timing and fuzzy-metric temporal logic”, *International Journal of Intelligent Systems*, volume 14, issue 8, pp.719-745, 1999.
- [63] Zinn D., Bosch J. and Gertz M., “Modeling and Querying Vague Spatial Objects Using Shapelets”, *VLDB '07*, September 23-28, 2007, Vienna, Austria.

APPENDIX A

SAMPLE METEOROLOGICAL MAPS



Figure 84: Cloudiness mapping on 30.12.2007

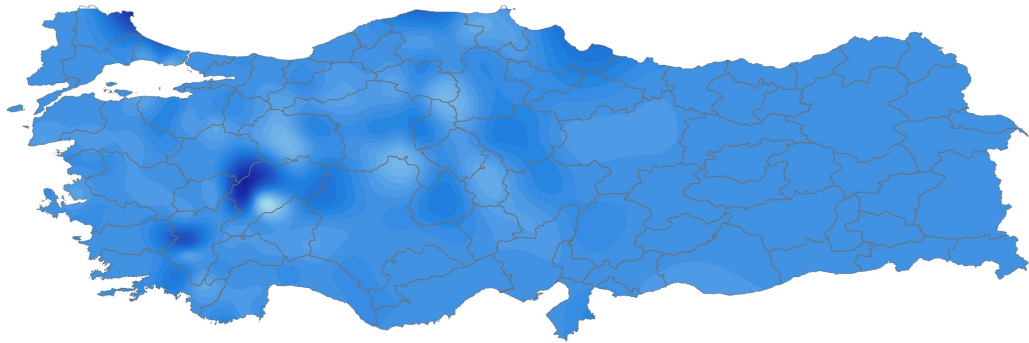


Figure 85: Pressure mapping on 30.12.2007

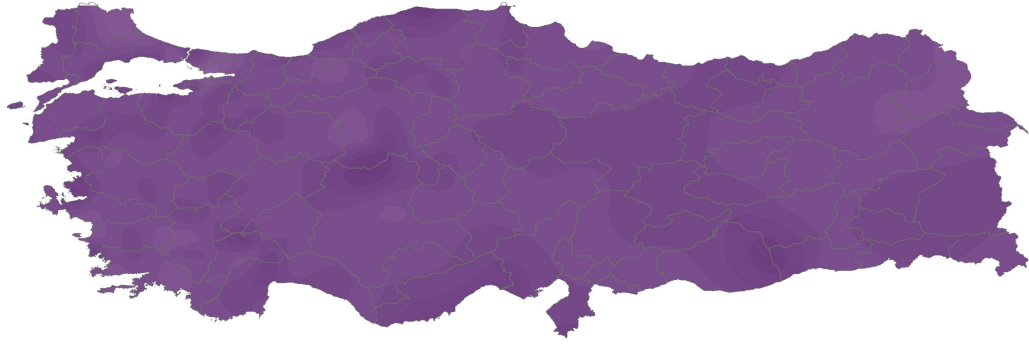


Figure 86: Wind strength mapping on 30.12.2007

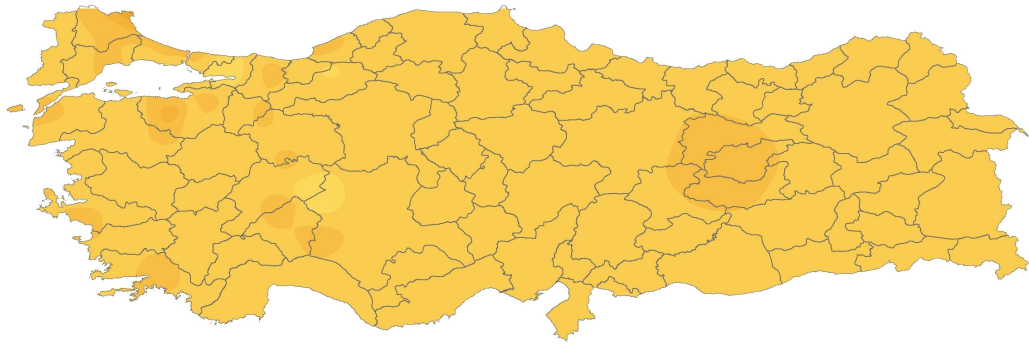


Figure 87: Sunshine duration mapping on 30.12.2007

APPENDIX B

THE OBJECT MODEL SPECIFICATIONS IN ALLOY

module systems/STModel

open alloy/models/util/ordering[LineSegment] as ordL

open alloy/models/util/ordering[Time] as ordT

open alloy/models/util/ordering[Fuzzy] as ordF

//abstract fuzzy class. The implementation consists of definition of fuzzy number

// which gives degree of fuzziness between 0 and 1

abstract sig Fuzzy{}

//The Time class includes definiton of time in YYYYMMDD hh:mm

sig Time{}

// a temporal class includes temporal class and a temporal entity has

// beginning time and end time

sig Temporal{

beginTime,endTime:Time

}

// Beginning time should be less than or equal to end time

fact TemporalFact{

all T:Temporal\ ordT/le[T.beginTime,T.endTime]

}

```

//a temporal object exists in a temporal interval
pred isInstance(o:Temporal,bt,et:Time){
ordT/gte[o.beginTime,bt] and ordT/lte[o.endTime,et]
}

// Spatialbase has common entries for spatial classess
// membership: is a fuzzy number and shows the degree of
// spatial object's belonging to a particular spatial class
// size: for fuzzy spatial object size is also fuzzy.
abstract sig SpatialBase extends Temporal{
membership:Fuzzy, //fuzzy membership
size:Fuzzy
}

// Coordinate defines an x, y location in the space.
// x, y may be float numbers
sig Coordinate{}

// Point is the basic spatial element and can be part of line segments.
sig Point extends SpatialBase {
location:Coordinate
}

// a line segment is aggregated by a set of points,
// It has a beginning and ending defined by points.
sig LineSegment extends SpatialBase{
sourceEnd:Point,
targetEnd:Point
}

// a region is aggregated by a set of line segments
sig Region extends SpatialBase{
linesegs:set LineSegment
}

```

```

// at least 3 line segments form a region
fact RegionConst{
  all r:Region | #r.linesegs >= 3
}

// a geometry is formed by a set of points and/or linesegments
// and/or regions
sig Geometry extends Temporal{
  points: set Point,
  linesegs: set LineSegment,
  regions: set Region
}

// a geometry should have at least one of the parts.
// not all of the parts can be empty sets.
// this fact does not allow empty geometry
fact GeometryFact{
  all g:Geometry |
    not (#g.points=0 and #g.linesegs=0 and #g.regions=0)
}

// if a geometry exists in some temporal interval so that
// its parts should exist in the same interval
fact GeometryConst{
  all g:Geometry |
    isInstance[g,g.beginTime,g.endTime] =>
      ( isInstance[g.points,g.beginTime,g.endTime] and
        isInstance[g.linesegs,g.beginTime,g.endTime] and
        isInstance[g.regions,g.beginTime,g.endTime])
}

```

```

// A spatiotemporal object definition
sig STObject extends Fuzzy{
  geometry:some Geometry, // an STObject has one or more Geometry
  holes:set Geometry, // an STObject may have holes
  trajectory:some Point, //trajectory is a non-empty set of points
  spatialRelation:set RelationType // An STObject may have spatial
                                // relation(s) with other STObjects
}

```

```

// A spatial relation exists in some temporal interval
// including two STObjects and fuzzydegree that shows
// the degree of the relation
abstract sig RelationType extends Temporal{
  F,G: one STObject, //two STObjects F and G
  fuzzyDegree:Fuzzy //the degree of the relation which is fuzzy
}

```

```

// These are the possible types of spatial relations.
// Each one of them is a relation between two STObjects and have a degree
one sig Disjoint,Meet, Inside,Equal, Contains,Covers,CoveredBy,Overlap extends
RelationType{
  rel:F->G->Fuzzy
}

```

```

// the following predicates give definitions for the spatial relations
pred disjointCR(R:RelationType){
  no(R.F.geometry & R.G.geometry) and no(R.G.geometry & R.F.holes) and
  no(R.F.geometry & R.G.holes) and no(R.F.holes & R.G.holes)
}

```



```

pred insideCR(R:RelationType){
  some (R.F.geometry & R.G.geometry) and (no(R.F.geometry & R.G.holes) or
  ((R.G.holes in R.F.geometry) and (R.G.holes in R.F.holes)))
}

```

```

pred insideCR2(R:RelationType){
  some (R.F.geometry & R.G.geometry) and (no(R.F.geometry & R.G.holes) or
  ((R.F.holes in R.G.geometry) and (R.F.holes in R.G.holes)))
}

```

```

pred meetCR(R:RelationType){
  one (R.F.geometry & R.G.geometry) and not disjointCR[R] and
  not insideCR[R] not insideCR2[R] and not equalCR[R]
}

```

```

pred containsCR(R:RelationType){
  insideCR2[R]
}

```

```

pred equalCR(R:RelationType){
  (R.F.geometry=R.G.geometry) and (R.F.holes=R.G.holes)
}

```

```

pred coversCR(R:RelationType){
  insideCR2[R] and meetCR[R] and #(F.geometry & G.geometry) = 1
}

```

```

pred coversCR2(R:RelationType){
  insideCR[R] and meetCR[R] and #(F.geometry & G.geometry) = 1
}

```

```

pred coveredbyCR(R:RelationType){
  coversCR2[R]
}

pred overlapCR(R:RelationType, fuz:Fuzzy){
  not (disjointCR[R] or meetCR[R] or insideCR[R] or containsCR[R] or
  equalCR[R] or coveredbyCR[R] or coversCR[R]) and ordF/gte[R.fuzzyDegree,fuz]
}

// Spatial relation asserts
assert disjointT{ //if disjoint not any other relation
all R:RelationType, fDegree:Fuzzy | disjointCR[R] =>
  not (overlapCR[R,fDegree] or meetCR[R] or insideCR[R]
  or containsCR[R] or equalCR[R] or coveredbyCR[R] or
  coversCR[R]) or
  R.G.geometry in R.F.holes //a geometry may be inside the hole
}

check disjointT

assert meeT{
all R:RelationType, fDegree:Fuzzy | meetCR[R] =>
  not disjointCR[R] and not insideCR[R] and not equalCR[R] and
  not coversCR[R] and not overlapCR[R,fDegree]
}

check meeT

assert insideE{
all R:RelationType, fDegree:Fuzzy | insideCR[R] =>
  not (disjointCR[R] or meetCR[R] or coversCR[R] or
  overlapCR[R,fDegree]) or (R.F.geometry in R.G.holes)
}

```

check insideE

assert coverS{

all R:RelationType, fDegree:Fuzzy | coversCR[R]=>

not (disjointCR[R] or overlapCR[R,fDegree]) and

insideCR2[R] and (G.holes in F.geometry)

}

check coverS

assert equalL{

all R:RelationType, fDegree:Fuzzy | equalCR[R]=>

not (disjointCR[R] or overlapCR[R,fDegree]) and

(R.G.geometry in R.F.geometry) and (R.F.holes in R.G.holes)

}

check equalL

//application specific classes

//-----

// City may have some routes crossing and have some weather object

sig City extends STObject{

route:set Route,

weather:some MetObject

}

sig Route{

parts: some LineSegment, //route has at least one LineSegment or more

//route may be one type or a mixed type. e.g. maritimeRoute or

// territoriolRoute + MaritimeRoute

rType:some RouteType,

```

// some parts may be clear some parts may be restricted so it has at least
// one status but may have more than one
rStatus:some RouteStatus,

// route crosses at least one City
cities:some City,

// a set of vehicles use the route
vehicles:set Vehicle
}

//two consecutive line segments over a route should have one common point
//one's targetEnd equals other's sourceEnd
assert routeFact{
  all R:Route, ls1,ls2:R.parts|
    (ls1!=ls2 and ordL/eq[ordL/next[ls1],ls2])=>ls1.targetEnd= ls2.sourceEnd
}
check routeFact

// route type can ben maritime, territorial or aerial
abstract sig RouteType{}
one sig MaritimeRoute,TerritorialRoute, AerialRoute extends RouteType{}

// route may be clear, wavy (for maritime route) or restricted (for all types)
abstract sig RouteStatus{}
one sig Clear,Wavy,Restricted extends RouteStatus{}

sig Vehicle{
  type:VehicleType,
  status:VStatusType,
  route:set Route
}

```

```

//route-vehicle relation at specific time
sig Journey extends Temporal{
  route: Route,
  vehicle: Vehicle,
}

// 1-a journey has only one vehicle and one route
// 2-A vehicle is used on only one journey during the journey
assert JourneyFact{
  all j1,j2:Journey, t:Temporal{
    isInstance[t,j1.beginTime,j1.endTime] and
    isInstance[t,j2.beginTime,j2.endTime]=>
    one j1.vehicle and one j1.route and one j2.vehicle and one j2.route and
    j1.vehicle != j2.vehicle

  }
  check JourneyFact
}

sig Voyage extends Journey{} //ship journey

abstract sig VehicleType{}
one sig Ship, Bus, Train, Plane extends VehicleType{}

// a voyage has a MaritimeRoute and the vehicle running should be Ship
fact voyageFact{
  all vyg:Voyage{
    vyg.route.rType=MaritimeRoute and vyg.vehicle.type=Ship
  }
}

// A vehicle may be on time, delayed or canceled
abstract sig VStatusType{}
one sig OnTime,Delayed,Canceled extends VStatusType{}

```

```

//if a journey's route is restricted vehicle is delayed or canceled
fact statusFact{
  all j:Journey|
    j.route.rStatus=Restricted=>
    j.vehicle.status=Delayed or j.vehicle.status=Canceled
}

// a meteorological object is a spatiotemporal object
sig MetObject extends STObject{
  object:MeteorType,
  degree:MeteorObjectDegree
}

// the types of meteorological objects are enumerated here
abstract sig MeteorType{}
one sig Temperature, Pressure, Visibility, Wind, Wave, Cloude, Precipitation
extends MeteorType{}

// The strength of Meteorological object i
abstract sig MeteorObjectDegree extends Fuzzy{}

// Here a sample is given for visibility
one sig Visible, Misty, Foggy extends MeteorObjectDegree{}

// Another sample for precipitation
one sig Drizzle, Rainy, Snowy, Thunderstorm extends MeteorObjectDegree{}

/ a meteorological measurement in a City includes a number of meteorological
objects
sig Measurement extends Temporal{
  metobj:some MetObject,
  city:lone City
}

```

```

// two spatiotemporal objects have overlap degrees
abstract sig OverlapDegree extends Fuzzy{}

//fuzzy overlapdegrees are enumerated
one sig Less, Moderate,High extends OverlapDegree{}

//if city and meteorological object overlaps than the object is in the city's weather
fact weatherFact{
  some M:MetObject, C:City, R:Overlap|
    (R.F=M and R.G=C and overlapCR[R,High]) => M in C.weather
}

// As an example : if a city's weather has visibility and precipitation and
// their degrees are strong enough, then the route is restricted
fact routeStatusFact{
  some M:MetObject, C:City, route:Route|
    (
      (M.object=Visibility and M.degree=Foggy) or
      (M.object =Precipitation and (M.degree=Snowy or
M.degree=Thunderstorm)) and
      M in C.weather and C in route.cities)=>route.rStatus=Restricted
}

```

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Sözer, Aziz

Nationality: Turkish (TC)

Date and Place of Birth: 20.09.1973, Acıpayam

Marital Status: Married

Phone: +90 312 507 59 90

Fax: +90 312 507 60 18

Email: e070364@ceng.metu.edu.tr

EDUCATION

Degree	Institution	Year of Graduation
MS	METU Computer Engineering	2001
BS	METU Computer Engineering	1995

WORK EXPERIENCE

Year	Place	Enrollment
1997-present	T.C Merkez Bankası	Expert
1995-1997	Arçelik A.Ş.	Analyst
1991-1995	Devlet Meteoroloji İşleri Gen. Md.	Forecaster

PUBLICATIONS

1. Aziz Sözer, Adnan Yazıcı, Halit Oğuztüzün and Fred Petry, "Querying Fuzzy Spatiotemporal Databases: Implementation Issues,", Uncertainty Approaches for Spatial Data Modeling and Processing: a Decision Support

Perspective (Edited Book) Edited by J. Kacprzyk, F. Petry, A. Yazıcı,
Springer Verlag, Germany (2009)

2. Aziz Sözer, Adnan Yazıcı, Halit Oğuztüzün and Osman Taş, “Modeling and querying fuzzy spatiotemporal Databases”, *Information Sciences*. 178(19): 3665-3682 (2008).
3. Aziz Sözer, Adnan Yazıcı, “Design and implementation of index structures for fuzzy spatial Databases”, *International Journal of Intelligent Systems* 22(7): 805–826 (2007)
4. Sozer, A. and A. Yazıcı “ Index Structures for Flexible Querying in Fuzzy Spatial Databases”, *IEEE Conference on Fuzzy Systems*, Budapest, 2004.
5. Sozer, A. ve A. Yazıcı, “Access Structures for Fuzzy Spatial Queries” *NAFIPS-2002*, pp: 383-388, New Orleans, USA.