DESCUBRIMIENTO DE PATRONES CATEGÓRICOS SECUENCIALES EN UN ENTORNO ESPACIO-TEMPORAL

by

Leticia Irene Gómez

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

Major Subject: Ingeniería en Informática

at

INSTITUTO TECNOLÓGICO DE BUENOS AIRES

Buenos Aires, Argentina Noviembre 24, 2009

CATEGORICAL SEQUENTIAL PATTERN MINING IN A SPATIO-TEMPORAL ENVIRONMENT

by

Leticia Irene Gómez

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

Major Subject: Informatics Engineering

at

INSTITUTO TECNOLÓGICO DE BUENOS AIRES

Buenos Aires, Argentina November 24, 2009

Instituto Tecnológico de Buenos Aires

Faculty of Engineering

Department of Informatics Engineering

The undersigned hereby certify that they have examined, and recommend to the Faculty of Graduate Studies for acceptance, the thesis entitled "**Categorical Sequential Pattern Mining in a Spatio-Temporal Environment**" by **Leticia Irene Gómez** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** .

Dated: ____________________

Supervisor: ______________________________
Ph.D. Alejandro Ariel Vaisman

Examiners: ______________________________
Ph.D. Juan Ale

____
Ph.D. Silvia Gordillo

____
Ph.D. Regina Motz

Instituto Tecnológico de Buenos Aires

Faculty of Engineering

DATE: ____________

AUTHOR:	Leticia Irene Gómez
TITLE:	**Categorical Sequential Pattern Mining in a Spatio-Temporal Environment**
MAJOR SUBJECT:	Informatics Engineering
DEGREE:	Doctor of Philosophy
CONVOCATION:	November, 2009

Signature of Author

Abstract

Los Sistemas para el Soporte de Decisión (DSS) ayudan a las actividades para la toma de decisiones en las organizaciones y los negocios. Hoy en día las organizaciones requieren de DSS más sofisticados, capaces de manejar diferentes tipos de datos en forma transparente. Esos datos puede estar en formato de información espacial, producida por Sistemas de Información Geográfica (GIS), que manejan datos georeferenciados. De este modo, aparece un nuevo concepto: DSS basado en GIS para análisis de datos con respecto a información espacial.

Las herramientas para el procesamiento analítico en línea (OLAP) son cruciales para los DSS. En esta tesis primero proponemos un modelo formal, llamado Piet, que permite la integración en un único marco, de datos espaciales y OLAP. También introducimos un lenguaje formal que constituye la base para un lenguaje de consulta, estilo SQL, que soporta el modelo Piet, llamado Piet-QL. Piet-QL permite expresar complejas y poderosas consultas GIS y OLAP, extendiendo el concepto bien conocido de SOLAP (Spatial OLAP).

Los objetos móviles (MOs) equipados con dispositivos que proporcionan posiciones, producen trayectorias en la forma de tuplas (O_{id}, t, x, y) que contienen identificador de objetos e información espacio-temporal. Las trayectorias descriptas de esta forma pueden ser reescritas in términos de lugares previamente definidos. Esta forma de expresar trayectorias les agrega información semántica, permitiendo inferir interesantes patrones de movimiento que conducen al concepto de *trayectoria semántica*. En esta tesis argumentamos que la información de trayectorias también puede ser integrada con datos GIS y OLAP, generando un marco poderoso de análisis. Esta integración es llevada a cabo en términos de un lenguaje basado en expresiones regulares que permite expresar patrones secuenciales por intensiø'n. Este lenguaje, llamado RE-SPaM, puede ser utilizado dentro de un algoritmo de descubrimiento de patrones secuenciales para restringir la cantidad de secuencias a descubrirse, o como un lenguaje de consulta sobre la base de datos de trayectorias. La principal novedad en RE-SPaM es que no sólo soporta atributos en expresiones (los esfuerzos previos solamente manejaron identificadores) sino también atributos temporales, variables y funciones. Extendiendo RE-SPaM con la habilidad de incluir consultas Piet-QL en las expresiones regulares conduce a RE-SPaM^{++} el cual permite que el análisis de los objetos móviles se relacione con el entorno geométrico en el cual las trayectorias evolucionan. El caso de estudio muestra como todo este marco permite la extracción de información interesante a partir de la colección espacial, OLAP y objectos móviles. Piet-QL, RE-SPaM, y RE-SPaM^{++} han sido implementados e integrados en una herramienta, basada en la plataforma OpenJump, permitiendo tanto ediciones de consultas como visualización y análisis de los resultados.

Abstract

Decision Support Systems (DSS) support business and organizational decision-making activities. Nowadays, organizations require more sophisticated DSS, capable of handling different kinds of data in a seamless fashion. These data maybe in the form of spatial information, for example, information produced by Geographic Information Systems (GIS), that deal georeferenced data. Thus, a new concept appears: GIS-based DSS for analyzing data with respect to spatial information. On-Line Analytical processing (OLAP) tools are crucial to DSS. In this thesis we first propose a formal model, denoted Piet, that makes it possible to integrate in a single framework, spatial and OLAP data. We also introduce a formal query language, that constitutes the basis of the SQL-Like query language supporting the Piet data model, denoted Piet-QL. Piet-QL can express complex and powerful GIS and OLAP queries, extending the well-known notion of SOLAP (Spatial OLAP).

Moving objects (MOs) equipped with location-aware devices produce trajectory data in the form of a sample of (O_{id}, t, x, y)-tuples that contain object identifier and time-space information. Raw trajectories described in this way can be rewritten in terms of places previously defined. This way of expressing trajectories adds semantic information to them, allowing to infer interesting patterns of movement yielding the concept of *semantic trajectory.* In this thesis we argue that trajectory information can also be integrated with GIS and OLAP data, leading to a powerful analysis framework. This integration is achieved by means of a regular expression-based language that allows expressing sequential patterns in an intensional way. This language, denoted RE-SPaM, can be used within a sequential pattern mining algorithm to restrict the number of sequences that are discovered, or as a query language over a trajectory database. The main novelty in RE-SPaM is that it not only supports attributes in the expressions (former efforts only deal with item identifiers) but also temporal attributes, variables and functions. Expanding RE-SPaM with the ability of including Piet-QL queries in the regular expressions leads to RE-SPaM^{++}, which makes moving object analysis aware of the geometric environment where the trajectories evolve. A case study shows how all this framework can be used for extracting interesting information from a collection of spatial, OLAP, and moving object data.

Piet-QL, RE-SPaM, and RE-SPaM^{++} have been implemented and integrated into a tool, based on the Open Jump platform, allowing query editing and visualization and analysis of the results.

Acknowledgements

I want to thank God for giving me a family that taught me the importance of compromising with goals and making efforts in life. Thanks to my sister and friend Silvia for her invaluable support and unconditional love.

I would also like to thank the authorities at ITBA for encouraging me to continue studying and stimulating me during all these years. I wish to thank my colleagues at ITBA for their incredible help.

Related to my doctoral activity, I want to thank Roberto Perazzo and the members of the Doctoral Committee for their constant encouragement. Thanks to Eduardo Bonelli for sharing his experiences with me. Also thanks to Professors Joos Heintz and Marcelo Scasso for helping me during the courses I took with them. I am also grateful to Bart Kuijpers for his cordiality during my stay in Belgium.

Finally, I want to remark my special thanks to my advisor Professor Alejandro Vaisman for all the invaluable time he devoted to me, his patience and dedication. I have no doubt that without his direction and advice it would have been impossible for me to conclude this thesis.

Table of Contents

Abstract **iv**

Abstract **vi**

Acknowledgements **viii**

List of Tables **xiii**

List of Figures **xiv**

Chapter 1 Introduction **1**

1.1 OLAP and Decision Support Systems 2

1.2 Moving Object Databases (MOD) 4

1.3 Thesis Contribution 5

1.4 Thesis Organization 7

Chapter 2 Related Work **8**

2.1 GIS and OLAP Integration 8

2.1.1 The concept of SOLAP 8

2.1.2 Conceptual Modeling 9

2.1.3 Spatial Aggregation 10

2.1.4 Implementation 11

2.2 Moving Objects 11

2.2.1 Trajectory Query Languages 12

2.2.2 Trajectory Aggregation 12

2.2.3 Trajectory Pattern Detection 13

2.2.4 Semantic Trajectories 15

2.3 Summary 16

Chapter 3 Data Model 17

3.1 An Introductory Example 17

3.2 Piet Data Model 18

3.3 Comparison of our proposal with Related Work 23

3.4 Spatial Aggregation 25

3.4.1 Geometric Aggregation 25

3.4.2 Summable Queries 28

3.5 Summary 30

Chapter 4 Piet-QL Query Language 31

4.1 Language Design 31

4.2 Piet-QL by Example 32

4.3 Syntax 36

4.4 Semantics 40

4.4.1 GIS 40

4.4.2 OLAP 42

4.4.3 GIS-OLAP and OLAP-GIS 42

4.4.4 Clauses 42

4.5 Implementation Details 45

4.6 Piet-QL theoretical foundation 51

4.7 Comparison of our proposal with Related Work 54

4.8 Summary 55

Chapter 5 Moving Object Trajectories 56

5.1 Introductory Example 56

5.2 Trajectories 57

5.3 A Data Model for Compressed Trajectories 58

5.3.1 Compressing the MOFT 60

5.3.2 Computing the SM-MOFT 61

5.4 Summary 63

Chapter 6 RE-SPaM: A Data Mining Language 64

6.1 Motivation 65

6.2 Data Model 67

6.3 Sequential Expressions 70

6.4 The RE-SPaM Language 77

6.4.1 Syntax and Semantics 78

6.4.2 RE-SPaM by Example 81

6.5 RE-SPaM Evaluation 84

6.5.1 Preliminary Considerations 84

6.6 The Data Mining Process 85

6.6.1 The RE-SPaM Algorithm 86

6.6.2 Complexity 93

6.7 Comparison of our Proposal with Related Work 95

6.8 Summary 97

Chapter 7 Adding Geography to RE-SPaM 98

7.1 Introduction 98

7.1.1 RE-SPaM^{++}Motivation 99

7.2 RE-SPaM^{++} Design 100

7.2.1 Functions in RE-SPaM^{++} 101

7.3 RE-SPaM^{++}By Example 102

7.4 Implementation 104

7.5 Summary 108

Chapter 8 Implementation and Case Study 109

8.1 Data Preparation 109

8.1.1 OLAP Data 109

8.1.2 GIS Data 110

8.1.3 Trajectory Data 111

8.1.4 Preparing and Running the Simulator 112
8.1.5 Preparing the Places of Interest 114
8.2 Experiments 116
8.2.1 Goal 1 117
8.2.2 Goal 2 118
8.2.3 Goal 3 122
8.2.4 Goal 4 123
8.3 Visualizing RE-SPaM^{++} results 124
8.4 Summary 125

Chapter 9 Conclusion 127
9.0.1 Open Research Directions 128

Bibliography 130

List of Tables

4.1 Piet-QL Semantics 43

4.2 An OLAP query result set 47

6.1 Categories Schema 69

6.2 An example of category instances 69

6.3 An instance of the ToI 71

6.4 RE-SPaM Grammar 72

6.5 An instance of a ToI 78

6.6 Regular expression operators 79

6.7 The normalize ToI instance 85

7.1 Grammar for RE-SPaM^{++} 102

8.1 POIs category instance of case study 115

8.2 Category Schema for the case study 115

8.3 Max and Min Duration for PoIs 116

8.4 Executing time for experiment 1.1 117

8.5 Phases of Experiment 1.1 118

8.6 Performance of Experiment 2.2 (DB medium) 121

8.7 Performance of Experiment 2.2 (DB large) 121

8.8 Performance of Experiment 4.1 124

List of Figures

3.1 GIS Layers 18

3.2 A GIS dimension schema 20

3.3 A GIS dimension instance 22

4.1 Hierarchy mapping 32

4.2 Visualizing in OpenJump result set of GIS-OLAP query Q7 . 46

4.3 Visualizing in a browser the result set of OLAP-GIS query Q8 49

4.4 OLAP Hierarchy inferred 50

5.1 MOFT and SM-MOFT instances 57

5.2 An example of stop and moves 60

5.3 Automata for Stops and Moves calculation 63

6.1 Trajectories semantically equivalent 66

6.2 A taxonomy example 77

6.3 Automaton for query Q0 80

6.4 Automaton for query Q2 82

6.5 Automaton for query Q6 83

6.6 Steps for computing C1 89

6.7 Steps for computing C2 90

6.8 Steps for computing C3 90

6.9 Steps for computing F 92

7.1 Regions, rivers and PoIs in Belgium map 105

7.2 Automaton for RE-SPaM^{++}query Q1 106

7.3 Computing C_1: before and after pruning with automaton . . . 107

7.4 Computing C_2 : before and after pruning with automaton . . 108

8.1 The Belgium road network 113

8.2 A Raw Trajectory 114

8.3 Experiment 2.1 120

8.4 Experiment 3.1 122

8.5 Input Parameters for RE-SPaM^{++} 125

8.6 Semantically Equivalent Trajectories 126

8.7 Semantically Equivalent Trajectories in the Road Network . . 126

Chapter 1

Introduction

Geographic Information Systems (GIS) have been extensively used in various application domains, ranging from economical, ecological and demographic analysis, to city and route planning [60, 73]. The NCGIA[1] defines a GIS as a system of hardware, software and procedures to facilitate the management, manipulation, analysis, modelling, representation and display of georeferenced data to solve complex problems regarding planning and management of resources

In general, information in a GIS application is represented in several so-called thematic layers or themes containing related data that deal with one thematic topic. For example, rivers, volcanoes and regions can be organized in different layers. Information in each layer consists of purely spatial data, associated with classical alphanumeric attribute data. Usually, data are stored in a relational database. The spatial integration of these layers can be carried out by using a common coordinate system. By overlapping layers we can obtain a unified spatial view for better analysis.

Geometric objects within themes can be stored in different data structures according to different data models. Two main data models are used for the representation of the spatial part of the information within one layer, the vector model and the raster model. The choice of model typically depends on the data source from which the information is imported into the GIS. The *vector model* is mostly used in current GIS [41]. In the vector model, infinite sets of points in space are represented as finite geometric structures, or geometries, like, for example, points, polylines and polygons.

[1]The National Center for Geographic Information and Analysis is an independent research consortium dedicated to basic research and education in geographic information science and its related technologies, including geographic information systems (GIS). See http://www.ncgia.ucsb.edu.

More concretely, vector data within a layer consists in a finite number of tuples of the form (geometry, attributes) where a geometry can be a point, a polyline or a polygon. There are several possible data structures to actually store these geometries [73]. In the *raster model*, the space is sampled into pixels or cells. For each cell or pixel, the sample value of some function is computed and associated to the cell as an attribute value, e.g., a numeric value or a color. Usually, these cells are organized into zones, where the cells of a zone have the same value for some attribute(s). This model has highly efficient indexing structures and it is very well-suited to model continuous change but its disadvantages include its size and the cost of computing the zones. Queries requiring map overlay (spatial joins) are more difficult to compute in the vector model than in the raster model. On the other hand, the vector model offers a concise representation of the data, independent from the resolution. For a uniform treatment of different layers given in the vector or the raster model, in this thesis we focus on the vector model. Indeed the raster model can be considered as a special case of the vector model by considering each cell or zone as point or polygons, respectively. Besides, the attribute value associated to the cell or pixel can be regarded as an attribute in the vector model.

1.1 OLAP and Decision Support Systems

OLAP (On Line Analytical Processing) [37] comprises a set of tools and algorithms that allow efficiently querying multidimensional databases, containing large amounts of data, usually called Data Warehouses. In OLAP, data are organized as a set of *dimensions* and *fact tables*. In this multidimensional model, data can be perceived as a *data cube*, where each cell contains a measure or set of (probably aggregated) measures of interest. OLAP dimensions are further organized in hierarchies that favor the data aggregation process [10]. Several techniques and algorithms have been developed for query processing, most of them involving some kind of aggregate precomputation [32] (an idea we will use later in this paper). Three typical ways of OLAP tools implementation exist: MOLAP (standing for multidimensional OLAP), where data is stored in proprietary multidimensional structures, ROLAP (relational

OLAP), where data is stored in (object) relational databases, and HOLAP (standing for hybrid OLAP), which provides both solutions. In a ROLAP environment, data is organized as a set of dimension tables and fact tables, and we assume this organization in the remainder of this thesis. There are a number of OLAP operations that allow exploiting the dimensions and their hierarchies, thus providing an interactive data analysis environment. Warehouse databases are optimized for OLAP operations which, typically, imply data aggregation or de-aggregation along a dimension, called roll-up and drill-down, respectively. Other operations involve selecting parts of a cube (slice and dice) and re-orienting the multidimensional view of data (pivoting). In addition to the basic operations described above, OLAP tools provide a great variety of mathematical, statistical, and financial operators for computing ratios, variances, ranks, etc. It is an accepted fact that data warehouse conceptual design is still an open issue in the field [62]. Most of the data models either provide a graphical representation based on the Entity-Relationship (E/R) model or UML notations, or they just provide some formal definitions without user-oriented graphical support. Recently, Malinowsky and Zimányi [47] have proposed the MultiDim model. This model is based on the E/R model and provides an intuitive graphical notation. Also recently, Vaisman [67, 68] has introduced a methodology for requirement elicitation in Decision Support Systems, arguing that methodologies used for OLTP systems are not appropriate for OLAP.

Decision Support Systems (DSS) are information systems that support business and organizational decision-making activities. A DSS is an interactive piece of software intended to help decision makers to compile useful information from raw data, documents, personal knowledge, and/or business models to identify and solve problems and make decisions. Most DSS are based on OLAP tools like the ones described above. However, a DSS can include other kinds of software tools, like for instance, Geographic Information Systems. The seamless integration of both OLAP and GIS tools is one of the topics of this thesis, as we discuss below.

1.2 Moving Object Databases (MOD)

In the last few years, the possibility of obtaining information of moving objects through the use of Global Position Systems (GPS) or Radio Frequency IDentification (RFID) [70] has been consistently increasing given that these devices have become widely available. Moving objects carrying location-aware devices produce trajectory data in the form of a sample of (O_{id}, t, x, y)-tuples, that contain object identifier and time-space information. The analysis of these data can be of interest in many domains, like traffic analysis, road planning, or location of advertisement in city streets, among other ones. Research concerning moving objects was introduced by Wolfson *et al.* [69, 71, 72].

Typical problems in this field concern the analysis of trajectory similarity, aggregation, and pattern discovery. Detecting trajectory similarity is a complex task, because locations of different moving objects, represented by floating point coordinates, do not necessary coincide. Some approaches try to detect moving object patterns with different mining techniques. In any case, due to the huge volume of these trajectory data any form of compression facilitates the data processing. Recently, the notions of *stops* and *moves* were introduced [16, 64]. These concepts allow compressing trajectory data produced by moving objects using application-dependent places of interest. In this approach, a designer selects a set of places of interest that are relevant to her application. For instance, in a tourist application, such places can be hotels, restaurants and tourist attractions. In a traffic control application, they may be road segments, traffic lights and junctions, stored in GIS layers. If a moving object spends a sufficient amount of time in a place of interest, this place is considered a *stop* for the object's trajectory. Between stops, the trajectory is considered to have *moves*. Thus, we can replace a raw trajectory given by (O_{id}, t, x, y)-tuples by a sequence of application-relevant *stops* and *moves*, leading to the concept of *semantic trajectories*. Hidden movement patterns in these kinds of trajectories can be discovered using different variations of existing data mining techniques, like the well-known Generalized Sequential Patterns (GSP) algorithm [4, 65]. Also, some proposals allow expressing patterns intensionally by means of regular expressions [16, 20] over the items that are being mined. These techniques can take advantage of semantic information about the places of interest. Further, even more interesting patterns can be inferred if moving objects are integrated with OLAP and GIS data in a common framework. We discuss this integration in this thesis.

1.3 Thesis Contribution: Towards the Integration of GIS, OLAP, and Moving Object Data

Nowadays, organizations need sophisticated GIS-based Decision Support System to analyze their data with respect to geographic information. In this sense, OLAP and GIS vendors are increasingly integrating their products[2]. Aggregate queries are central to DSSs. Thus, classical aggregate queries (like "total sales of cars in California"), and aggregation combined with complex queries involving geometric components ("total sales in all villages crossed by the Mississippi river within a radius of 100 km around New Orleans") must be efficiently supported. Moreover, navigation of the results using typical OLAP operations like roll-up or drill-down is also required. These operations are not supported by commercial GIS in a straightforward way. *First,* GIS data models were developed with 'transactional' queries in mind. Thus, the databases storing non-spatial attributes or objects are designed to support those (non-aggregate) kinds of queries. Decision support systems need a different data model, where non-spatial data, consolidated from different sectors in an organization, is stored in a data warehouse. Here, numerical data are stored in fact tables built along several dimensions. For instance, if we are interested in the sales of certain products in stores in a given region, we may consider the sales amounts in a fact table over the dimensions Store, Time and Product. Moreover, we mentioned above that OLAP dimensions are organized into aggregation hierarchies. For example, stores can aggregate over cities which in turn can aggregate into regions and countries. Each of these aggregation levels can also hold descriptive attributes like city population, the area of a region, etc. Integration between GIS and OLAP data requires warehouse data to be linked to geographic data. For instance, a polygon representing a region

[2]See Microstrategy and MapInfo integration in http://www.microstrategy.com/, http://www.mapinfo.com/

must be associated to the region identifier in the warehouse. *Second,* system integration in commercial GIS is not an easy task. As a consequence, in current commercial systems, the GIS and OLAP worlds must be integrated in an ad-hoc fashion, in a different way each time an implementation is required.

The *first contribution of this thesis* is aimed at solving the problem of integrating GIS and OLAP data. For this, we present a formal data model, denoted Piet, that integrates GIS and OLAP in a unique framework. We also define a formal query language that supports spatial aggregation. Based on this formal language, we introduce an SQL-like query language denoted Piet-QL, that can express complex and powerful GIS and OLAP queries in a natural and concise way. Piet-QL supports four basic kinds of queries: (a) standard spatial queries; (b) standard OLAP queries; (c) spatial queries filtered with an aggregation (i.e., filtered using a data cube); (d) OLAP queries filtered with a spatial condition.

The integration mentioned above becomes more interesting when moving objects are included in the picture. We have already commented that the notion of semantic trajectories allows inferring interesting patterns of movement. Usually, this pattern analysis is performed disregarding the information about the geographic location where the moving objects evolve. As a *second contribution of this thesis* we propose a language based on regular expressions, denoted RE-SPaM, that can intensionally express sequential patterns by means of regular expressions built over constraints defined over the attributes of the places of interest visited by the trajectories under analysis. This language can express patterns of the form: "tourists first visit cheap restaurants, then visit tourist attractions repeatedly, and finish at 3-star hotels". Integration of GIS, OLAP and moving object data is achieved by means of including Piet-QL statements into the constraints that compose a RE-SPaM pattern. *This is the third contribution of this thesis.* In other words, we make moving object analysis aware of the geometric environment where the objects evolve.

As a *fourth contribution of this thesis,* we show, through a case study, how the tools defined and implemented in this thesis can be put together for decision support in the presence of the three kinds of data previously mentioned.

This thesis consolidates and expands a corpus of work previously published. In the field of OLAP-GIS integration, relevant work has been published in [17, 23]. Piet-QL was first introduced in [26]. The concept of Places of Interest and Stops and Moves has been discussed in [24]. Finally, RE-SPaM has been presented in [25].

1.4 Thesis Organization

This thesis is organized as follows: In Chapter 2 we review related work in the field of OLAP and Spatial Data Integration (also called SOLAP) and review MOD literature. In Chapter 3 we propose a formal model, called Piet, for the integration of spatial and OLAP data and introduce a formal query language based on the concept of spatial aggregation. Then, in Chapter 4 we present a query language, called Piet-QL, designed for Piet data model.

In Chapter 5 introduces the concept of semantic trajectories and how can be used for compressing trajectories and enriching them with additional information. In Chapter 6 we present the notion of trajectories *semantically equivalent* and propose a mining algorithm, called RE-SPaM, to detect sequential behavior patterns over them. Also, to reduce the number of sequences obtained during the mining process, we propose a query language based on regular expressions, which supports variables, functions and attributes. Although it can be applied to other scenarios, we will apply it to MO setting. Chapter 7 shows how Piet-QL with RE-SPaM can be integrated, resulting in the RE-SPaM^{++}language that lets us find sequential patterns considering the geographic environment in which MOs evolve. In Chapter 8 we discuss a complete case study, analyze the visualizing of the result of the mining process and present experimental results. Finally, in Chapter 9, we analyze future extensions of our work and summarize the main features of our contribution.

Chapter 2

Related Work

In this chapter we review the efforts in OLAP and Spatial Data integration and Moving Object Databases (MOD) literature. We compare the approaches discussed here against our proposal at the end of each corresponding chapter.

2.1 GIS and OLAP Integration

In the last five years, the topic of spatial OLAP and spatio-temporal OLAP has been attracting the attention of the database and GIS communities. In this section we discuss advances and limitations that have been appearing, and classify them according to different lines of research.

2.1.1 The concept of SOLAP

En 1997, Bédard introduce the concept of SOLAP (standing for Spatial OLAP) to support spatio-temporal analysis and exploration of data according to a multidimensional approach. Later, Bédard *et al.* describe the characteristics of a SOLAP user interface tool capable of exploring spatial data by drilling on maps as it is performed in OLAP with tables and charts [7], although in this proposal they do not have an underlying formal model. Related to the concept of SOLAP, Shekhar *et al.* [63] introduced MapCube, a visualization tool for spatial data cubes. MapCube is, basically, an operator that given a so-called base map, cartographic preferences and an aggregation hierarchy, produces an album of maps that can be navigated via roll-up and drill-down operations.

2.1.2 Conceptual Modeling

Stefanovic *et al.* [66] and Bédard *et al.* [6], classify spatial dimension hierarchies in: (a) non-geometric; (b) geometric to non-geometric; and (c) fully geometric. Dimensions of type (a) can be treated as any descriptive dimension. In dimensions of types (b) and (c) a geometry is associated to members of the hierarchies. Malinowski and Zimányi [46], in the *Multidim* model, extend this classification to consider that even in the absence of spatial levels, a dimension can be considered spatial. If it is represented as a spatial data type (e.g., point, region), allowing linking spatial levels through topological relationships (e.g., contains, overlaps). Thus, a spatial dimension is a dimension that contains at least one spatial hierarchy. This model is an extension of a previous conceptual model for OLAP introduced by the same authors, based on the well-known Entity-Relationship model [45]. In the models above, spatial measures are characterized in two ways, namely: (a) measures representing a geometry, which can be aggregated along the dimensions; (b) a numerical value, using a topological or metric operator. Most proposals support option (a) either as a set of coordinates [6, 8, 46, 61] or a set of pointers to geometric objects [66]. In particular, in [46], the authors define measures as attributes of an n-ary fact relationship between dimensions. Further on, the same authors present a method to transform a conceptual schema to a logical one expressed in the Object-Relational paradigm [48]. Fidalgo *et al.* [18] and da Silva *et al.* [13] introduced GeoDWFrame, a framework for spatial OLAP, which classifies dimensions as geographic and hybrid, if they represent only geographic data, or geographic and non-spatial data, respectively. Over this framework, da Silva *et al.* [14] propose GeoMDQL, a query language based on Multidimensional Expressions (MDX)[1] and OGC[2] simple features for querying spatial data cubes (see Section 2.1.4). Pourabbas introduces a conceptual model that uses binding attributes to bridge the gap between spatial databases and a data cube [58]. No implementation of the proposal is discussed. Besides, this approach relies on the

[1]MDX was first introduced as part of the OLE DB for OLAP specification in 1997 from Microsoft. While it was not an open standard, but rather a Microsoft owned specification, it was adopted by the wide range of OLAP vendors.

[2]Open Geospatial Consortium, http:www.opengeospatial.org

assumption that all the cells in the cube contain a value, which is not the usual case in practice, as the author expresses. Moreover, the approach also requires modifying the structure of the spatial data.

2.1.3 Spatial Aggregation

The notion of OLAP is closely related to data aggregation. In a spatial setting the concept of spatial aggregation arises. Vega López *et al.* [44] present a comprehensive survey on spatiotemporal aggregation that includes a section on spatial aggregation. Pedersen and Tryfona [54] propose pre-aggregation of spatial facts. They analyze that in the spatial setting traditional pre-aggregation techniques do not work due to the properties of topological relationship between 2D spatial objects. In traditional pre-aggregation techniques summarizability [43] must be satisfied, but in spatial data it cannot be guaranteed due to geometry overlapping. Therefore, they propose a method to take advantage of pre-aggregation in spatial data warehouses that consists of pre-processing spatial facts by computing their disjoint parts. These facts could be aggregated later, given that pre-aggregation works if the spatial properties of the objects are distributive over some aggregate function. This proposal ignores the geometry, and do not address forms other than polygons. Thus, queries like "Give me the total population of cities crossed by a river" are not supported. The authors do not report experimental results. Extending this model with the ability to represent partial containment hierarchies (useful for a location-based services environment), Jensen *et al.* [36] propose a multidimensional data model for mobile services, i.e., services that deliver content to users, depending on their location. Like in the previously commented proposals, this model omits considering the geometry, limiting the set of queries that can be addressed.

With a different approach, Rao *et al.* [59], and Zang *et al.* [75] combine OLAP and GIS for querying so-called spatial data warehouses, using R-trees for accessing data in fact tables. The original star-schema does not need to be modified and the data warehouse is then evaluated in the usual OLAP way. Thus, they take advantage of OLAP hierarchies for locating information in the R-tree which indexes the fact table. Here, although the measures are not spatial objects, they also ignore the geometric part. It is assumed that some fact table containing the identifiers of spatial objects exists. Moreover, these objects happen to be just points, which is quite unrealistic in a GIS environment where different types of objects appear in the different layers. Other proposals in the area of indexing spatial and spatio-temporal data warehouses [51, 52] combine indexing with pre-aggregation resulting in a structure denoted Aggregation R-tree (aRtree), an R-tree that annotates each Minimal Bounding Rectangle (MBR) with the value of the aggregate function for all the objects that are enclosed by it. This is a very efficient solution for some particular cases, specially when a query is posed over a query region whose intersection with the objects in a map must be computed on-the-fly. However, problems may appear when leaf entries partially overlap the query window. In this case, the result must be estimated, or the actual results computed using the base tables. Kuper and Scholl [41], suggest the possible contribution of constraint database techniques to GIS. Nevertheless, they did not consider spatial aggregation, nor OLAP techniques.

2.1.4 Implementation

SOLAP concepts and operators have been implemented in a commercial tool called JMAP [1]. Han *et al.* [31] use OLAP techniques for materializing selected spatial objects, and proposed a so-called Spatial Data Cube. This model only supports aggregation of such spatial objects and not aggregation of non-spatial data subject to geometric conditions. As for a query language Silva *et al.* [14] propose GeoMDQL, a query language for SOLAP environments. This proposal, however, besides lacking an underlying formal model, does not work over spatial data but over a data warehouse which includes at least one geographic dimension. Thus, the language is limited to an extension of MDX with geographic operators.

2.2 Moving Objects

The field of moving object databases (MODs) has been extensively studied in the last ten years, specially regarding data modeling and indexing. Güting and Schneider [29] provide a good reference to this large corpus of work.

2.2.1 Trajectory Query Languages

Wolfson *et al.* [72] stated a set of capabilities that a moving object database must have, and introduced the DOMINO system, which develops those features on top of existing database management systems (DBMS) [71]. As moving objects report their changes very frequently, MODs must deal with frequent updates. Thus, the proposal consists of managing dynamic attributes instead of traditional static attributes. A dynamic attribute changes its value continuously without being explicitly updated (e.g. the position of an object). The main intention is to be able to analyze current data (properties of a MO such as location and speed) to predict future positions. The proposal does not deal with pattern detection, but introduces a new query language called FTL (Future Temporal Language) which manages location uncertainty. For instance, the query "which objects may/must intersect some region within the next 5 minutes?" is supported. Although they implemented MO using an object-relational database and a GIS to allow users to interact with geographic objects within a map (e.g. draw a region from scratch), no details are provided about the experiments, number of MO analyzed, average length of trajectories, etc.

Hornsby and Egenhofer [33] introduced a framework for modeling MOs, which supports viewing objects at different granularities, depending on the sampling time interval. The idea is to model all the locations visited by an object by inferring them from discrete samples. The basic modeling element they consider is a *geospatial lifeline*, which is composed of triples of the form $< Id, location, time >$, where *Id* is the identifier of the object, *location* is given by x-y coordinates, and *time* is the timestamp of the observation. The possible positions of an object between two observations is estimated to be within two inverted halfcones that conform a *lifeline bead* (also usually called space-time prism), whose projection over the x-y plane is an ellipse. Thus, the movement recorded by samples is generalized to a coarser view.

2.2.2 Trajectory Aggregation

In trajectory scenarios the idea of aggregation has been studied and it is related to the problem of trajectory similarity. Detecting similarities among trajectories could reduce their storage and facilitate their process.

Existing work focuses on the spatial notion of similarity, sometimes borrowing from the time-series analysis field. This is the approach followed by Pelekis *et al.* [57] who introduce a framework consisting of a set of distance operators based on parameters of trajectories like speed and direction, and propose distance operators based on this. Frentzos *et al.* [19] propose an approximation method for supporting the k-most-similar-trajectory search using R-tree structures. Meratnia and de By [50] tackle the topic of aggregation of trajectories, identifying similar trajectories and merging them in a single one, by dividing the area of study into homogeneous spatial units. Papadias *et al.* [53] index historical aggregate information about moving objects. Finally, Kuijpers *et al.* [39] propose a taxonomy of aggregation queries on moving object data.

2.2.3 Trajectory Pattern Detection

Data mining studies techniques to discover interesting patterns hidden in large volumes of data. These techniques have also been applied to the field of MOD.

Clustering is a well-known mining algorithm for grouping together similar objects. Lee *et al.* [42] remark that by applying clustering to whole trajectories several patterns could not be detected. Thus, their proposal aims at discovering common sub-trajectories, and also use a partitioning strategy, proposing a partition-and-group framework for clustering trajectories. For that, they produce a trajectory partition into lines using the minimum description length (MDL) principle, and cluster those segments to detect trajectory similarities.No temporal component is analyzed during the mining process, i.e., they reduce trajectories to their spatial component (only the shape of the trajectory is analyzed).

For mining *trajectories constrained by road networks*, Brakatsoulas *et al.* [9] propose to add spatial information to trajectories of moving objects. They consider to incorporate information about the relationships between trajectories (e.g., intersect, meets, near), and between a trajectory and the GIS environment (e.g. stay/within/leave some building). They propose a mining language denoted SML (standing for Spatial Mining Language), oriented to traffic networks. The language does not take advantage of the particular characteristics that moving object data present. Also in the framework of road traffic mining, Gonzalez *et al.* [27] use a partitioning approach for obtaining interesting driving and speed patterns from large sets of traffic data. They compute frequent path-segments at the area level with a support relative to the traffic in the area (i.e., a kind of adaptative support), and propose an algorithm to automatically partition a road network and build a hierarchy of areas.

Classic data mining algorithms (in particular sequential pattern algorithms) can be applied to trajectory databases if we view the latter as a collection of *ordered sequences*. Two main approaches had been followed in the field of pattern discovery in sequences: the classic Agrawal and Srikant proposal [4], the approach of Mannila *et al.* [49]. The former is aimed at discovering inter-transactions patterns, based on previous work [2, 3] dealing with detecting intra-transactions patterns. The information to be mined is organized in transactions and the system returns the frequent sequential patterns among them. The latter, instead, considers the information to be mined as a large single sequence. The choice of the algorithm depends on the application domain. In their seminal work, Agrawal *et al.* [2, 3] propose data to be pre-processed in a way such that each customer (in a market basket analysis scenario) is associated with all her transactions ordered by time of occurrence. The idea is to find inter-transaction patterns corresponding to the same customer with a certain support. An interesting sequential pattern is one that appears in the database at least as many times as an user-specified threshold. The support of a sequence is defined as the fraction of the total number of transactions containing it. In further work, the authors extend their proposal [65] in order to support three kinds of constraints: (a) time-gap constraints; (b) taxonomies; (c) time windows. The resulting algorithm is called Generalized Sequential Patterns (GSP). Although many frequent sequential patterns could be obtained using GSP, it is likely that only a few of them could be relevant to the user. To avoid this situation, Garofalakis *et. al.* [20, 21] propose a variation of the GSP algorithm, denoted SPIRIT, where user-defined regular expressions are used to prune the information obtained. Apriori-like algorithms are based on the anti-monotony property. Although they take advantage of pruning, they generate a generally huge number of candidate frequent patterns. To improve this, pattern-growth methods have been proposed to avoid the generation of candidate sequences: FreeSpan [30] and PrefixSpan [55]. In these methods, so-called projected databases are built recursively, and these smaller databases are scanned to find locally frequent sequences, avoiding scanning the original sequence database. These methods find the full set of frequent subsequences. Then, constraint-based sequential pattern mining based on constructing projected databases have been studied [56]. Further, to avoid generating patterns that could be obtained from other ones, Yan *et al.* introduce the CloSpan algorithm [74], which reduces the number of generated patterns by mining only frequent closed subsequences, i.e., those containing no super-sequence with the same support.

It is worth to remark that none of these sequential pattern mining algorithm could be applied directly to raw trajectories, since the former are based on detecting coincidence of object's identifiers (nominal or integer), for instance, ISBN of books, employee ID. Imagine that two sequences of positions of different objects, i.e., two trajectories, could be similar enough but it is unrealistic to expect that their samples (locations) coincide.

2.2.4 Semantic Trajectories

Techniques that add semantic information to trajectory data have been recently proposed. Mouza and Rigaux [16] present a model where trajectories are represented as a sequence of *moves* (zones represented by labels or IDs). They propose a query language based on regular expressions aimed at obtaining so-called mobility patterns. We postpone the analysis of this proposal to Chapter 6, where we compare it with the language we present in this thesis.

Giannotti *et al.* [22] study trajectory pattern mining, based on Temporally Annotated Sequences (TAS), an extension of sequential patterns, where there is a temporal annotation between two nodes. In this way, 's1; 2; s2' defines a pattern that starts at s1 and after 2 seconds arrives at s2. In other words, a trajectory pattern is a set of trajectories that visit the same sequence of places with similar travel times between each one of them. They also propose three different mining methods and introduce the concept of Region of Interest (RoI) which is dynamically computed from the trajectories.

With a similar idea, Damiani *et al.* [64] introduce the concept of stops and moves, in order to enrich trajectories with semantically annotated data. Alvares *et al.* [5] presented a framework for trajectory analysis based on *stops* and *moves*. The concept of Stop differ from the RoI. The former is application-dependant, defined in advance and really relevant to a trajectory. The latter is detected dynamically. Their work proposes how to compress trajectories with the notion of stops and moves, and use SQL to ask for trajectories which satisfy some conditions. No mining algorithm is discussed to detect patterns within trajectories.

2.3 Summary

We have reviewed proposals in the field of SOLAP and Moving Object databases that are relevant to this thesis. In the following chapters, as we introduce our proposal, we compare it with the work discussed here.

Chapter 3

Data Model

In this chapter, we present a general framework to integrate OLAP and spatial data in a single framework. We introduce a formal model and query language and compare our approach with existing proposals.

3.1 An Introductory Example

We begin with an example that motivates our proposal. We selected five layers with geographic features obtained from the spatial library of the GIS Center[1], containing *cities*, *districts*, *provinces*, *regions* and *rivers* in Belgium. Cities are represented by points, rivers by polylines and the other ones as polygons, as shown in Figure 3.1. Also numerical and textual information on the geographic components exists (e.g., number of potatoes cultivated per hectare, number of agricultural employees, area), stored as usual in a GIS. There is also a data warehouse that we created for this case study (i.e., it does not correspond to a real world situation), about sales in Belgium. It has a *store dimension*, a *product dimension* and a *time dimension*. Dimensions are organized in hierarchies. In particular, the store dimension contains the following *spatial hierarchy* (represented by means of non-geometric data): Store Name → City→ District → State → Country. The measures of the cube are *Unit Sales*, *Store Cost*, *Store Sales* and *Number of Products Sold*. There are also fact data about sales. In this scenario, sales information could be analyzed in the light of geographical features. For instance, we can ask for "Unit Sales and Store Cost for stores in districts which belong to

[1]http://giscenter-sl.isu.edu

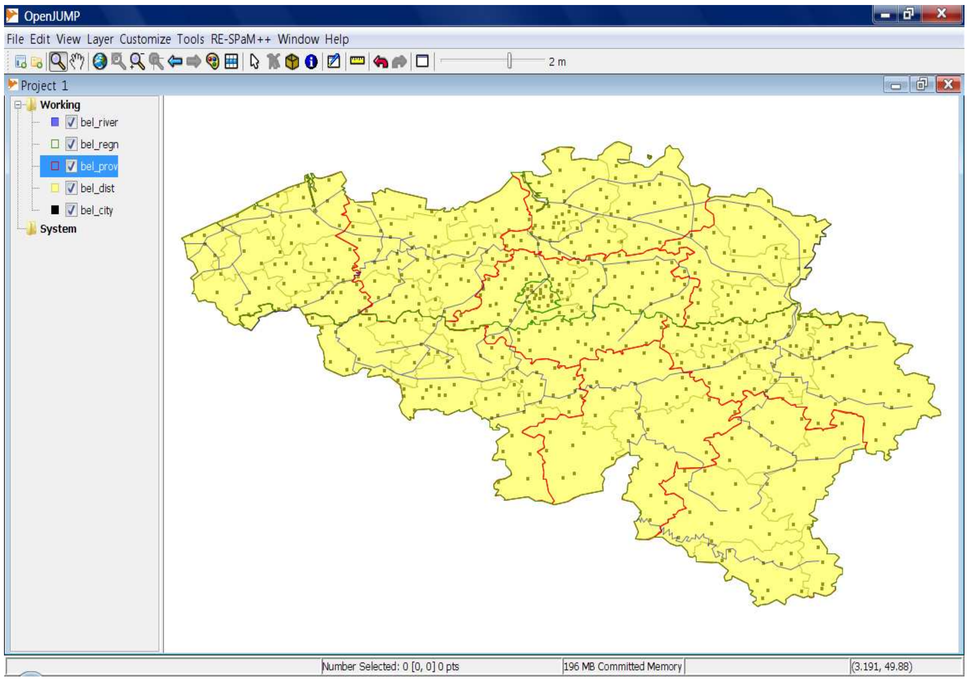


Figure 3.1: Layers in Belgium Map

provinces crossed by at least 5 rivers". In this case, information about regions could be stored in a GIS layer, instead of in the data warehouse. Notice that this query returns spatial data but the result is constrained by OLAP data.

3.2 Piet Data Model

In our proposal, denoted Piet (after Piet Mondrian, the painter whose name was adopted for the open source OLAP system that we also use in our implementation), is aimed at integrating in a single model, spatial and non-spatial information, probably produced independently from each other. We assume, without loss of generality, that non-spatial data are stored in a data warehouse following the standard OLAP notion of dimension hierarchies and fact tables [10, 35]. We denote this approach *loosely-coupled*, given that warehouse and spatial data are maintained independently from each other. The main component of the model is denoted a *GIS dimension*. A GIS dimension consists, as usual in databases, in a dimension schema that describes its structure, and dimension instances. Each dimension is composed of a set of graphs, each one describing a set of geometries in a thematic layer. Figure 3.2 depicts a GIS dimension schema, with three graphs representing three different layers, following our running example: rivers (L_r), airports (L_a), and districts (L_d), respectively. Typically, each layer contains a set of binary relations between geometries of a single kind (although the latter is not mandatory). For example, an instance of the relationship (*line,polyline*) stores the identifiers of the lines belonging to a polyline. We define three sectors, denoted the *Algebraic part*, the *Geometric part*, and the *Classical OLAP part*.

- *Algebraic part*: Is the lowest level (i.e., the level with the finest granularity) in the dimension schema, represented by a node with no incoming edges. We assume that this level, called 'point', represents points in space. Here, data in each layer are represented as infinite sets of points (x, y), finitely described by means of linear algebraic equalities and inequalities [40].

- *Geometric part*: Consists of a finite number of elements of certain geometries (represented by geometric object identifiers). This part is used for solving the geometric part of a query, for instance to find all polygons that compose the shape of a country. Each point in the *Algebraic part* corresponds to one or more elements in the *Geometric part*. Note that, for example, *a point may correspond to two adjacent polygons*, or to the *intersection of two or more roads*. Moreover, a line may correspond to more than one polygon. There is also a distinguished level, denoted 'All', with no outgoing edges.

- *OLAP part*: Consists in a set of OLAP hierarchies and fact tables in the usual OLAP sense. Dimensions can be spatial or non-spatial. In the former case, dimension levels can be associated with levels in the geometric part. For example, information about districts, stored in a data warehouse, can be associated to polygons, or information about rivers, to polylines.

Example 1. *In Figure 3.2, the schema is composed of three graphs. The graph for rivers contains edges saying that a point (x, y) in the algebraic part relates to a line identifier in the geometric part, and that the latter corresponds to a polyline identifier. Besides, the level* polygon *in layer L_d is associated with two dimension*

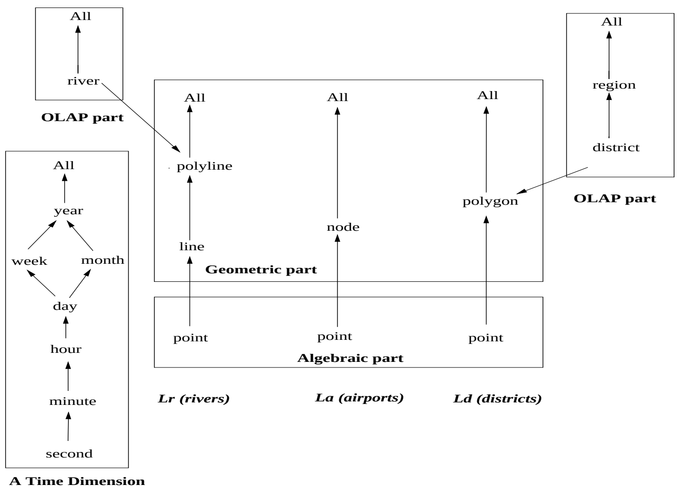


Figure 3.2: An example of a *GIS* dimension schema

levels, district and region, where district → region ("A → B" means that there is a functional dependency from level A to level B in the OLAP part [10]). Each level may have attributes associated, like population or income. Both layers are associated with the OLAP part, i.e. a geometrically-represented component is associated with a dimension level in the OLAP part. For example, the level river in the OLAP hierarchy (a spatial dimension with non-spatial levels) is associated to the layer L_r at the level of polyline *(geometric part). Notice that since dimension levels are associated to geometries, it is straightforward to associate facts stored in a data warehouse in the OLAP part in order to aggregate these facts along geometric dimensions. Finally, in the algebraic part, the relationship $\langle point, polygon\rangle$ associates infinite point sets with polygons.* □

Formally, assume there is a set of layer names **L**, and a set **G** of geometry names, which contains at least the following geometries: point, node, line, polyline, polygon and the distinguished element "All". Each geometry G of **G** has an associated domain *dom*(G). The domain of Point, *dom*(Point), for example, is the set of all pairs in $\mathbb{R}^2$. The domain of All = {all}. The domain of the elements G of **G**, except Point and All, is a set of geometry identifiers, g_{id}, i.e., g_{id} are identifiers of geometry instances, like polylines or polygons.

Definition 1. [*GIS Dimension Schema*] *A* GIS dimension schema *is a tuple* $\text{dGIS}_{sch} = \langle \mathcal{H}, \mathcal{F}_{att}, \mathcal{D} \rangle$ *where* $\mathcal{H}$ *is a finite set of graphs,* $\mathcal{F}_{att}$ *a set of functions, and* $\mathcal{D}$ *a set of OLAP dimension schemas. We define these sets below.*

Given a layer $L \in \mathbf{L}$, $\text{H}(L)$, *is a graph where: (a) there is a node for each kind of geometry* $\text{G} \in \mathbf{G}$ *in L; (b) there is an edge between two nodes* G_i *and* G_j *if* G_j *is composed by geometries of type* G_i *(i.e., the granularity of* G_j *is coarser than that of* G_i*); (c) there is a distinguished member All that has no outgoing edges; (d) there is exactly one node representing the geometry* $\langle point \rangle$ *with no incoming edges. The dimension schemas* $\text{D} \in \mathcal{D}$ *are tuples of the form* $\langle dname, Levels, \preceq \rangle$, *such that dname is the name of the dimension, Levels is a set of dimension level names, and* $\preceq$ *is a partial order between levels (see [35]). Finally,* $\mathcal{F}_{att}$ *contains* partial *functions (denoted Att) mapping attributes in OLAP dimensions to geometries in the layers.* □

Example 2. *The GIS dimension depicted in Figure 3.2 has the schema:* $\text{dGIS}_{sch} = \langle \{(\text{H}_1(L_r),\ (\text{H}_2(L_a),\ \text{H}_3(L_d)\},\ \{Att(\text{district}),\ Att(\text{river})\},\ \{Rivers, Districts\} \rangle$. *In this schema, for example,* $\text{H}_1(L_r) = (\{\text{point, line, polyline, All}\}, \{(\text{point, line}), (\text{line, polyline}), (\text{polyline, All})\})$. *For the OLAP dimensions* Rivers *and* Districts, *the Att functions in* $\mathcal{F}_{att}$ *are:* $Att(\text{district, Districts}) = (\text{polygon}, L_d)$ *(meaning that the attribute* district *maps to polygons in layer* L_d*), and* $Att(\text{river, Rivers}) = (\text{polyline}, L_r)$ *(i.e., the attribute* river *maps to polylines in the layer* L_r*).* □

Definition 2. [*GIS Dimension Instance*] *Let* $\text{dGIS}_{sch} = \langle \mathcal{H}, \mathcal{F}_{att}, \mathcal{D} \rangle$ *be a GIS dimension schema. A* GIS dimension instance *is a tuple* $\langle \text{dGIS}_{sch}, r_{alg}, r, \alpha, \mathcal{D}_{inst} \rangle$, *where: (a)* r_{alg} *is a 5-ary relation representing the rollup between the algebraic and geometric parts. Thus, it is a tuple of the form* $\langle L_i, G_j, x, y, g_{id} \rangle$, *where* L_i *is a layer name,* G_j *is the geometry in the geometric part to which point rolls up,* x, y *are the coordinates of a point, and* g_{id} *is the identifier of the object associated to* x, y *in the geometric part; (b)* r *is a 5-ary relation representing the rollup between objects in the geometric part. It is of the form* $\langle L_i, G_i, G_j, g_{id_i}, g_{id_j} \rangle$, *where* L_i *is a layer name,* G_i *and* G_j *are geometries*

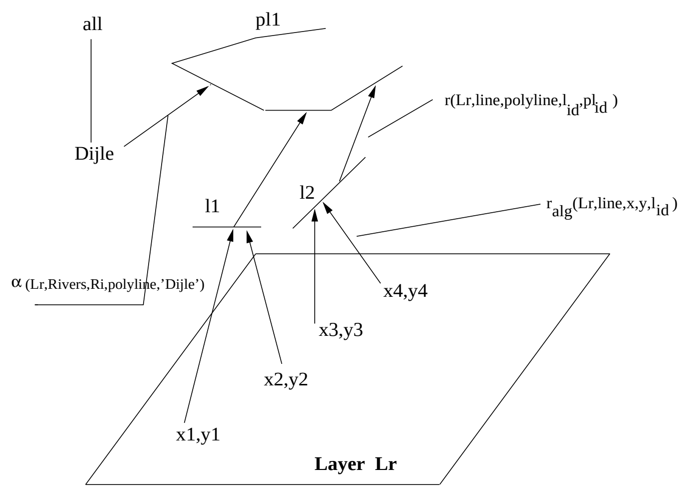


Figure 3.3: A portion of a *GIS* dimension instance in Figure 3.2.

in the geometric part such that the former rolls up to the latter (i.e., there is an edge $G_i \rightarrow G_j$ in $\mathrm{H}(L_i)$ *in* dGIS_{sch}*), and g_{id_i} and g_{id_j} are instances of these geometries. We denote r_{alg} and r* rollup *relations. The function α maps a* member *in a level denote* level *in an OLAP dimension D, to an object g_{id} in a geometry G in a layer L. This mapping corresponds to the functions in $\mathcal{F}_{att}$. Intuitively, α provides a link between a data warehouse instance and an instance of the hierarchy graph. Finally, for each dimension schema $D \in \mathcal{D}$ there is a dimension instance in $\mathcal{D}_{inst}$, composed of a set of rollup functions [35] that relate elements in the different dimension levels (these functions indicate how dimension level members are aggregated).* □

Example 3. *Figure 3.3 shows a portion of an instance of the GIS dimension schema of Figure 3.2. A member ('Dijle') of the level river in the OLAP dimension Rivers, is mapped (through the function α) to the polyline pl_1, in layer L_r. We show four points at the point level $\{(x_1, y_1), \ldots, (x_4, y_4)\}$ (recall that the points at this level are actually infinite and described by algebraic expressions). We also show the relations r_{alg} and r containing the association of points to lines and lines to polylines, respectively. For example, r_{alg} contains the tuple $\langle L_r, line, x_4, y_4, l_2\rangle$ and r contains the tuple $\langle L_r, line, polyline, l_{id}, pl_{id}\rangle$.* □

Elements in the geometric part in Definition 1 can be associated with *facts*, each fact being quantified by one or more *measures*, in the usual OLAP sense.

Definition 3. [*GIS Fact Table*] *Given a Geometry* G *in a graph* H(L) *of a GIS dimension schema, and a list* M *of measures* $(M_1, \ldots, M_k)$, *a* GIS Fact Table schema *is a tuple* $FT = (\mathrm{G}, L, \mathrm{M})$. *A* Base GIS Fact Table schema *is a tuple* $BFT = (\mathrm{p}\mathit{oint}, L, \mathrm{M})$, *that means, a fact table with the finest geometric granularity. A* GIS Fact Table instance *is a function* ft *mapping values in* $dom(G) \times \boldsymbol{L}$ *to values in* $dom(\mathrm{M_1}) \times \cdots \times dom(\mathrm{M_k})$. *A* Base GIS Fact Table instance *maps values in* $\mathbb{R}^2 \times L$ *to values in* $dom(\mathrm{M_1}) \times \cdots \times dom(\mathrm{M_k})$. □

Example 4. *Consider a fact table containing* state *populations in our running example, stored at the polygon level. The fact table schema would be* $(polyId, L_e, population)$, *where* population *is the measure. If information about, for example, temperature data, is stored at the* point *level, we would have a base fact table with schema* $(point, L_e, temperature)$, *with instances like* $(x_1, y_1, L_e, 25)$. *Note that temporal information could be also stored in these fact tables, by simply adding the* time *dimension to the fact table. This would allow to store temperature information all throughout time.* □

Basically, a GIS fact table is a standard OLAP fact table where one of the dimensions is composed of geometric objects in a layer. Classical fact tables in the OLAP part, defined in terms of the OLAP dimension schemas can also exist. For instance, instead of storing the population associated to a polygon identifier, as in Example 4, this information may reside in a data warehouse.

3.3 Comparison of our proposal with Related Work

As we remarked in Chapter 2, Bédard *et al.* [7] propose a taxonomy where they classify spatial dimension as *non-geometric*, *geometric* and *mixed.* Spatial measures are classified as *sets of pointers to geometries* and *numeric measures.* The Piet data model includes all these spatial dimensions. A *non-geometric* spatial dimension can be found when we have a spatial hierarchy represented as non spatial data. In our example we have a hierarchy in the warehouse relating the name of stores, cities, districts, states and countries. The *geometric* spatial dimension is the one given by the geometries in the GIS dimension part of the model. The mixed spatial dimension can store geometries and text. Bédard also defines two kinds of spatial measures that would be desirable to be supported: numerical and geometric measures. The former contains only numerical data and results, for instance, from the computation of spatial metrics. The latter are sets of geometries corresponding to a particular combination of dimension members from geometric or mixed spatial dimensions. We also support both kinds of measures. In the case of numerical measures, we can apply area, length or other operators to geometries in the GIS. In the case of geometric measures, applying union or intersection to aggregate geometries. We remark that we do not store the geometries in the DW as in other proposals. We store the pointers to the geometries that bind GIS with DW (we called this a loosely-coupled approach); thus, we apply those functions after de-referencing the pointers.

Stefanovic [66] and also Malinowski and Zimányi [46] propose to model the spatial components in the warehouse as a star schema [37]. In their proposal they build a data cube incorporating geometries (or pointers of geometries). These conceptual models follow what we have denoted as a *tightly-coupled* approach between the GIS and OLAP components, given that the spatial objects are included in the data warehouse. On the contrary, we follow a *loosely-coupled* approach, where GIS maps and data warehouses are maintained in a separate fashion, and bound by means of a matching function (α). We believe that this approach favors autonomy, updating and maintenance of the databases. In our proposal, it is only necessary to bind geographic elements in the maps to the existing data cube(s) that integrate organizational information, and the system is ready to receive queries.

It is worth noting that in the *tightly-coupled* approach it is unnatural to model some scenarios. This occurs when there is geographic information that we may want to include, but that is not directly related with the warehouse information. In our running example we may want to include information about volcanoes. It would be unnatural to include this information in a spatial star-schema.

In some sense, our proposal is similar to the one of Pourabbas [58], because her conceptual model binds attributes between spatial databases and a data cube. However, the model of Pourabbas requires modifying the structure of the spatial data and relies on the assumption that all the cells in the cube contain a value, which is not the usual case in practice, as the author expresses.

3.4 Spatial Aggregation

The formal model introduced in Section 3.2 allows us to define a formal query language over it making use of the elements defined in the model. This language is based on the notion of spatial aggregation. Thus, we start by defining this notion in order to give a precise definition of the kinds of queries that our proposal addresses.

3.4.1 Geometric Aggregation

Definition 4. [*Geometric Aggregation*] *Given a GIS dimension as introduced in Definitions 1 and 2, a* Geometric Aggregation *is the expression*

$$\iint_{\mathbb{R}^2} \delta_C(x,y)\; h(x,y)\; dx\, dy,$$

where $C = \{(x,y) \in \mathbb{R}^2 \mid \varphi(x,y)\}$[2]*, and δ_C is defined as follows:*

$\delta_C(x,y) = 1$ on the two-dimensional parts of C is a Dirac delta function [15] on the zero-dimensional *parts of C; and it is the product of a Dirac delta function with a combination of Heaviside step functions [34] for the* one-dimensional *parts of C. Also, φ is a first-order (FO) formula in a multi-sorted logic $\mathcal{L}$ over the reals, geometric objects, and dimension level members. The vocabulary of $\mathcal{L}$ contains the relations r, r_{alg}, and the function α, together with the binary functions $+$ and $\times$ on real numbers, the binary predicate $<$ on real numbers and the real constants 0 and 1*[3]*. Also constants for layers, dimension names, dimension level names, and geometry names*

[2] The sets C in Definition 4 are known in mathematics as *semi-algebraic sets*. In the GIS practice, only linear sets (points, polylines and polygons) are used. Therefore, it could suffice to work with addition over the reals only, leaving out multiplication.

[3] The first-order logic over the structure $(\mathbb{R}, +, \times, <, 0, 1)$ is well-known as the first-order logic with polynomial constraints over the reals. This logic is well-studied as a data model and query language in the field of constraint databases [40].

may appear in $\mathcal{L}$[4]*. Atomic formulas in $\mathcal{L}$ are combined with the standard logical operators $\wedge$, $\vee$ and $\neg$ and existential and universal quantifiers over real variables, variables for geometric objects identifiers, and variables for dimension level members. Finally, h is an integrable function constructed from elements of $\{1, ft\}$ (ft stands for fact table), using arithmetic operations.*

Note that this definition gives the basic construct for geometric aggregation queries. More involved queries can be written as combinations of this construct (e.g., "total number of airports per square kilometer" would require dividing the geometric aggregation that computes the number of airports in the query region, by the aggregation computing the area of such region)[5]. □

Example 5. *The following queries refer to Example 1. The layers containing rivers and districts are labeled L_r and L_d, respectively. The population for each coordinate in L_d is stored in a* base fact table *ft_{pop} (we assume it is stored in some finite way, i.e., using polynomial equations over the real numbers). In what follows, we abbreviate* Point*,* Polygon *and* PolyLine *by* Pt*,* Pg *and* Pl *respectively. Also,* Di *and* Ri *stand for the attributes* district *and* river*, respectively. Furthermore, all constants are capitalized, to distinguish them from variables in our expressions. Finally, in the queries below, the Dirac delta function is such that $\delta_C(x, y) = 1$, inside the region C, and $\delta_C(x, y) = 0$, outside this region.*

- ***Q_1: Total population of districts within 100km from* Antwerpen*.***

$$Q_1 \equiv \int\int_{C_1} ft_{pop}(x, y)\ dx\, dy,$$

where C_1 is defined by the expression:

[4]We may also quantify over layer variables, dimension level variables, etc., but we have chosen not to do this, for the sake of clarity.

[5]For the language $\mathcal{L}$, as usual in relational database theory, we assume set semantics, while in Chapter 4, set, bag, and mixed bag-set semantics [12] are supported through the SQL-like query language Piet-QL.

$$C_1 = \{(x,y) \in \mathbb{R}^2 \mid (\exists x')(\exists y')(\exists x'')(\exists y'')(\exists pg_1)(\exists pg_2)(\exists d)$$
$$(\alpha(L_d, \text{Districts}, \text{Di}, \text{Pg}, \textit{Antwerpen}) = pg_1 \ \wedge \ r_{alg}(L_d, \text{Pg}, x', y', pg_1) \ \wedge$$
$$\alpha(L_d, \text{Districts}, \text{Di}, \text{Pg}, d) = pg_2 \ \wedge \ r_{alg}(L_d, \text{Pg}, x'', y'', pg_2) \ \wedge$$
$$pg_2 \neq pg_1 \ \wedge \ ((x''-x')^2 + (y''-y')^2 \leq 100^2) \ \wedge r_{alg}(L_d, \text{Pg}, x, y, pg_2))\}.$$

Here, $\alpha(L_d, \text{Districts}, \text{Di}, \text{Pg}, \ Antwerpen)$ maps the district of Antwerpen (an instance of the level Di in dimension Districts), to a polygon pg_1 in layer L_d. The third and fourth lines find the districts within 100 Km of Antwerpen, and the relation $r_{alg}(L_d, \text{Pg}, x, y, pg_2)$, with the mapping between the points and the polygons that satisfy the condition. We are interested in the points that belong to pg_2.

- ***Q_2: Total population of the districts crossed by the* Dijle *river*.**

$$Q_2 \equiv \iint_{C_2} ft_{pop}(x,y)\ dx\, dy,$$

$$C_2 = \{(x,y) \in \mathbb{R}^2 \mid (\exists x')(\exists y')(\exists pg_1)(\exists d)$$
$$(r_{alg}(L_d, \text{Pl}, x', y', \alpha(L_r, \text{Rivers}, \text{Ri}, \text{Pl}, \textit{Dijle})) \ \wedge$$
$$\alpha(L_d, \text{Districts}, \text{Di}, \text{Pg}, d) = pg_1 \ \wedge \ r_{alg}(L_d, \text{Pg}, x', y', pg_1) \ \wedge$$
$$r_{alg}(L_d, \text{Pg}, x, y, pg_1))\}. \quad \square$$

This notion of *geometric aggregation* characterizes a wide range of aggregate queries over regions defined as semi-algebraic sets. Although general enough to express queries, some of them can be hard to compute in a real-world GIS environment because they involve computing an integral over a certain area. Thus, we identify the class of *summable* queries, which can be efficiently evaluated replacing this integral with a sum of functions of geometric objects.

3.4.2 Summable Queries

We identify a subclass of geometric aggregate queries that simplifies the computation of the integral of the functions $h(x, y)$ of Definition 4. In Example 5, the sets C_1 and C_2 return a finite set of polygons, representing districts. If the function ft_{pop} is constant for each district, it suffices to compute ft_{pop} once for each polygon, and then multiply this value by the area of the polygon. Summing up the products would yield the correct result, without the need of integrating ft_{pop} over the area C_1 or C_2. This is exactly the subclass of queries we want to propose, those that can be rewritten as sums of functions of geometric objects returned by condition 'C'. We denote these queries *summable.*

Definition 5. [*Summable Query*] *A geometric aggregation query* $Q = \int\int_{\mathbb{R}^2} \delta_C(x, y)\ h(x, y)\ dx\, dy$ *is* summable *if and only if:*

1. $C = \bigcup_{g \in G} ext(g)$*, where* G *is a set of geometric objects, and* $ext(g)$ *means the* geometric extension *of* g*, that is, the subset of* $\mathbb{R}^2$ *that* g *occupies (e.g, a polygon or a polyline, as a subset of* $\mathbb{R}^2$*).*

2. *There exists* h'*, constructed using* $\{1, f_t\}$ *and arithmetic operators, such that*

$$Q = \sum_{g \in S} h'(g),$$

with $h'(g) = \int\int_{\mathbb{R}^2} \delta_{ext(g)}(x, y)\ h(x, y)\ dx\, dy.$ □

Working with less accurate functions for this type of queries means that the Base GIS fact table instances of Definition 3 will be defined as mappings from values in $dom(\mathrm{G}) \times \mathbf{L}$ to values in $dom(\mathrm{M_1}) \times \cdots \times dom(\mathrm{M_k})$ (where the elements in $dom(\mathrm{G})$ are the geometric objects $g \in \mathrm{G}$ such that $r_{alg}(L, G, x, y, g)$), instead of mappings from values in $\mathbb{R}^2 \times \mathbf{L}$ to values in $dom(\mathrm{M_1}) \times \cdots \times dom(\mathrm{M_k})$.

Note that Definition 5 implies that the three summarizability conditions defined by Lenz *et al.* [43] must be preserved. The first two conditions (disjointness of categorical attributes, and completeness, respectively) are satisfied by our definition of a GIS dimension. The third condition requires function $h'(g)$ to be summarizable over g. The functions we use (typically sums, averages and maximum/minimum) satisfy the third condition in [43]. Since the integration region is replaced by geometric identifiers, set semantics expresses correctly the most usual summable queries of interest[6].

Example 6. *Let us reconsider query Q_2 from Example 5. The function ft_{pop} (i.e., the fact table) now maps elements of dom*(Polygon) *to populations. Note that C'_2 returns a finite set of polygons, indicated by their ids (denoted g_{id}).*

- ***Q_2: Total population of the districts crossed by the*** **Dijle** ***river***.

$$Q_2 \equiv \sum_{\mathrm{g_{id}} \in C'_2} ft_{pop}(\mathrm{g_{id}}).$$

$$\begin{aligned} C'_2 = \{g_{id} \mid & (\exists x)(\ \exists y)(\exists d) \\ & (r_{alg}(L_r, \mathrm{Pl}, x, y, \alpha(L_r, \mathrm{Rivers}, \mathrm{Ri}, \mathrm{Pl}, \mathit{Dijle})) \ \wedge \\ & \ \alpha(L_d, \mathrm{Districts}, \mathrm{Di}, \mathrm{Pg}, d) = g_{id} \ \wedge r_{alg}(L_d, \mathrm{Pg}, x, y, g_{id}))\}. \end{aligned}$$

□

Queries aggregating over zero or one-dimensional regions (such as queries requiring counting the number of occurrences of some phenomena) can also be summable. For example, counting the number of airports over a certain region, can be expressed as $Q \equiv \sum_{\mathrm{g_{id}} \in C} 1$.

Summable queries are useful in practice because, most of the time, we do not have information about parts of an object, like, for instance, the population of a part of a district. On the contrary, populations are often given by totals per district or province, etc. In this case, we may divide the district, for example, in a set of sub-polygons such that each sub-polygon represents a neighborhood. Thus, queries asking for information on such neighborhoods become summable.

[6]This can be extended to support bag semantics [28, 38], at the expense of increasing the presentation formal overload, and we chose to avoid this. Moreover, queries requiring bag semantics, like "Total number of rivers in Antwerpen and Ghent" (where, if a river crosses both district must be counted twice), can be written using combinations of the basic construct (splitting the query and adding the results), as we commented in Section 3.4.1.

3.5 Summary

In this chapter we introduced a formal model that integrates GIS and OLAP applications in an elegant way, and formalized the notion of *spatial aggregation* that characterizes a wide range of aggregate queries over regions defined as semi-algebraic sets. We also identified a class of queries, denoted summable, which can be evaluated efficiently without accessing the algebraic part of the GIS dimensions. This formal language constitutes the formal basic for a more user-friendly language, Piet-QL, that we introduce next.

Chapter 4

Piet-QL : A Query Language for Piet

We now present a query language based on the formal one introduced in Chapter 3. We provide the syntax and semantics of the language and a comprehensive set of examples that illustrate its use.

4.1 Language Design

The language was designed along the lines of the syntax of the OpenGIS[1] and MDX standard operators, in order to be familiar to users of GIS and OLAP worlds. Besides, the operators and functions proposed by OpenGIS are supported. The idea underlying the choice of combining the two languages was to preserve the GIS and OLAP standards as much as possible. Typically, four kinds of queries need to be supported in an OLAP and GIS integrated scenario:

- GIS queries filtered by geometric conditions.
- GIS queries filtered by OLAP conditions.
- OLAP queries filtered by geometric conditions.
- OLAP queries filtered by OLAP conditions.

[1]The Open Geospatial Consortium,Inc, http://www.opengeospatial.org/standards/sfs

In fact, the first and last ones were introduced to give completeness to the language, in the sense that the language can be also able to support standard OLAP and GIS queries.

4.2 Piet-QL by Example

We first provide a comprehensive set of examples to illustrate the capabilities of Piet-QL using our running example. Note that in this example we have a non-spatial hierarchy in the OLAP part, containing only non-spatial elements. On the other hand, the maps in the GIS part define mixed hierarchies, that we need to adapt in order to obtain consisten answers to the queries. In this case, the name of the levels do not coincide, and there are levels in one hierarchy that cannot match any level in the other one. Figure 4.1 depicts the hierarchy mapping. Thus, in the queries, 'state' will have the same meaning as 'province' (actually, Belgium is divided in provinces).

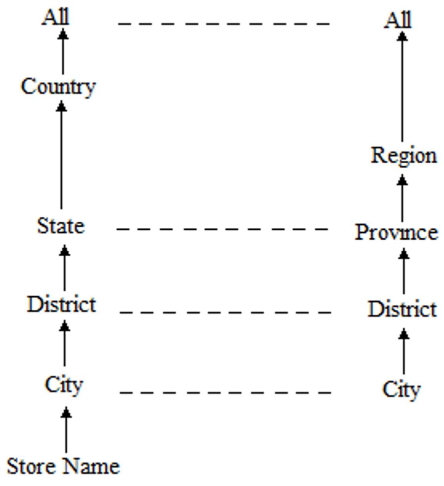


Figure 4.1: The store dimension in OLAP (left). The mixed spatial dimension in GIS (right)

Example 7. *Q1 (OLAP-GIS): "Unit Sales, Store Cost and Store Sales of stores in districts crossed by the Dijle river, in 2007", this query reads in Piet-QL:*

```
SELECT CUBE [Measures].[Unit Sales], [Measures].[Store Cost],
            [Measures].[Store Sales] ON COLUMNS
FROM [Sales]
WHERE [Store].[All Stores] IN(
  SELECT GIS bel_prov
  FROM bel_prov, bel_river
  WHERE ST_INTERSECTS(bel_river.the_geom,bel_prov.the_geom)
        AND bel_river.name='Dijle'
 )
SLICE [Time].[2007]
```

Here, the keyword ***CUBE*** *indicates that an OLAP query follows (and cube elements will be returned). Analogously, the keyword* ***GIS*** *indicates that a spatial query follows (and spatial elements will be returned) The geometric part corresponds to the inner subquery and returns provinces crossed by the Dijle river. The OLAP part returns the information of sales constrained by stores in these provinces, by using the* ***IN*** *predicate.* □

Example 8. *Q2 (OLAP-GIS): "Unit Sales of stores in cities which belong to provinces crossed by rivers. The analysis is constrained to single customers who purchased something during 2007". The query reads:*

```
SELECT CUBE [Measures].[Unit Sales] ON ROWS,
            [Product].[All Products] ON COLUMNS
FROM [sales]
WHERE [Store].[All Stores].[Store City] IN(
        SELECT GIS bel_city
        FROM bel_city, bel_prov AS bp, bel_river AS lr
        WHERE Intersects(bp, lr) AND Contains(bp,bel_city)
       )
SLICE ([Marital Status].[All Marital Status].[S], [Time].[2007])
```

This query is similar to Q1, but here we use a slice to filter specific conditions not only about the year of the sales but also about features of the customer dimension (although the information of this dimension is not displayed in the result). □

Example 9. *Q3 (Pure OLAP): "Store Sales and Store Cost for products and promotion media offered by stores in provinces (states) with sales greater than 5000". This query reads:*

```
SELECT CUBE [Measures].[Store cost],
            [Measures].[Store sales] ON COLUMNS,
            [Promotion Media].[All Media],
            [Product].[All Products] ON ROWS
FROM [Sales]
WHERE [Store].[All Stores] IN(
      SELECT CUBE filter([Store].[All Stores].[BELGIUM].Children,
                         [Measures].[Unit Sales]>5000)
      FROM [SALES] )
```

Here, all the parts are solved by the OLAP engine. The inner subquery, via the MDX expression [BELGIUM].Children, finds provinces (since in the store hierarchy provinces aggregate over countries) with stores with units sales greater than 5000. □

Example 10. *Q4 (GIS-OLAP): "Names and geometries of cities of Belgium in the Nijvel district such that their stores had sales during 2007". The query reads:*

```
SELECT GIS bel_city.name, bel_city.the_geom
FROM bel_city
WHERE bel_city IN(
    SELECT CUBE filter([Store].[Store District].[Nijvel].Children,
                       [Measures].[Unit Sales]>0)
    FROM [Sales]
    slice [Time].[2007]  )
```

The OLAP part corresponds to the inner subquery, which returns cities in the Nijvel district with stores that sold something during 2007. Remark that this subquery returns cities, i.e. identifiers. □

Example 11. *Q5 (GIS-OLAP): "Names and geometries of cities in provinces crossed by the river Dijle. Restrict these cities to the ones with sales greater than 5000 and that belong to provinces with stores that sold more than 100000 units"*

```
SELECT GIS lc1.name, lc1.the_geom
FROM bel_city AS lc1, bel_prov AS lp2, bel_river AS lr2
WHERE contains(lp2, lc1) AND
  intersects(lp2, lr2) AND lr2.name='Dijle'
  AND lc1 IN(
            SELECT CUBE
              filter([Store].[Store City].Members,
                     [Measures].[Unit Sales]>5000)
            FROM [Sales])
      AND lp2 IN(
            SELECT CUBE
              filter([Store].[Store State].Members,[Measures].
                     [Unit Sales]>100000)
            FROM [Sales])
```

Here we have two OLAP inner queries. The former about cities, the latter about provinces. The GIS outer query uses both of these lists in its `WHERE` *clause.* □

Example 12. *Q6 (Pure GIS): "Names and geometries of cities in provinces crossed by a river"*

```
SELECT GIS bel_city.name, bel_city.the_geom
FROM bel_city
WHERE ( bel_city IN(
    SELECT GIS bel_city
    FROM bel_city, bel_prov, bel_river
    WHERE INTERSECTS(bel_prov.the_geom, bel_river.the_geom ) AND
          CONTAINS(bel_prov.the_geom,bel_city.the_geom) ))
```

This is a typical spatial query. Although it could be expressed in only one query, we use the `IN` *predicate to illustrate Piet-QL features.* □

4.3 Syntax

Listing 4.1 depicts the syntax of Piet-QL in Backus-Naur Form.

The language uses the *basic data types* Number (real, integer) and String. Geometric data, like Point, Linestring and Polygon, Multi-Point (collections of Points), Multi-Linestring (collections of Linestrings) and Multi-Polygon (collections of Polygons) are expressed via strings in OpenGIS. For example, ‘Point(10 20)’, ‘Line(5 10, 16 2)’ and ‘Polygon((10 20, 30 30, 50 20, 10 20))’ are valid geometries in Piet-QL. We define the sets:

- Layers: a set of thematic layers in GIS.
- Geometries: a set of points, lines, polygons that belong to a GIS layer.
- GISIDs: a set of the identifiers of the elements in Geometries.
- Attributes: a set of attributes of a geometric object in a layer.
- Cubes: a set of OLAP data cubes.
- Members: a set composed of all members in a data cube, according to the MDX specification. For instance, in the path Argentina.BuenosAires each element is a member in the hierarchy of a geographic dimension going from country to province.

- Levels: a set composed of all levels in a data cube, according to the MDX specification. For instance, city and country are levels in the hierarchy of a geographic dimension.
- Dimensions: a set composed of all dimensions in a cube.
- Measures: a set composed of all measures in a cube.
- GISOLAP Tuples: a set of tuples that associate a geometric object with a dimension member. This means that there is a tuple (m, i) such as m ∈ Members and i ∈ GISIDs.

The syntax shown in Listing 4.1 uses the following *terminal symbols*:

- LITERAL$: a *basic data type.*
- GIS_ATTR$: references an attribute in the set Attributes, associated to a geometry of a layer. For instance, layer_city.name references the name of each geometry in a city layer.
- GIS_LAYER$: references a GIS layer in the set Layers.
- GIS_FN$: one of the following GIS functions:
  - Buffer or ST_Buffer[2]

    Syntax:
    Buffer: Geometries $\times$ Number $\rightarrow$ Geometries
    ST_Buffer: Geometries $\times$ Number $\rightarrow$ Geometries

  - Area or ST_Area

    Syntax:
    Area: Geometries $\rightarrow$ Number
    ST_Area: Geometries $\rightarrow$ Number

[2]Most of the functions and predicates in GIS have two similar notations. This is because there exist two specifications for spatial operators: OpenGIS and SQL/MM Part 3. Their differences are not substantial, but after several years, OpenGIS adopted SQL/MM operators.

- Length or ST_Length

  Syntax:

  Length: Geometries → Number

  ST_Length: Geometries → Number

- Intersection or ST_Intersection

  Syntax:

  Intersection: Geometries × Geometries → Geometries

  ST_Intersection: Geometries × Geometries → Geometries

• GIS_PREDICATE$: one of the following GIS predicate:

- Intersects or ST_Intersects

  Syntax:

  Intersects: Geometries × Geometries → Boolean

  ST_Intersects: Geometries × Geometries → Boolean

- Contains or ST_Contains

  Syntax:

  Contains: Geometries × Geometries → Boolean

  ST_Contains: Geometries × Geometries → Boolean

- Within or ST_Within

  Syntax:

  Within: Geometries × Geometries → Boolean

  ST_Within: Geometries × Geometries → Boolean

- Crosses or ST_Crosses

  Syntax:

  Crosses: Geometries × Geometries → Boolean

  ST_Crosses: Geometries × Geometries → Boolean

- Overlaps or ST_Overlaps

  Syntax:

  Overlaps: Geometries × Geometries → Boolean

  ST_Overlaps: Geometries × Geometries → Boolean

- Touches or ST_Touches

  Syntax:

  Touches: Geometries × Geometries → Boolean

  ST_Touches: Geometries × Geometries → Boolean

• OLAP_FN$: At this time we support the following filtering function:

- Filter

  Syntax:

  Filter: $2^{Members}$ × Expressions → $2^{Members}$

  Where 'Expressions' is a set of comparison predicates with infix notation.

  * Comparison predicates, one of =, >, <, <=, >=, <>

    Syntax:

    operator: Measures × Number → Boolean

• OLAP_CUBE$: references an OLAP data cube in Cubes.

• OLAP_MEMBER$: references a member in the set Members.

• OLAP_LEVEL$: references a level in the set Levels.

- General predicate:
    - IN

Syntax:
IN: GISIDs $\times$ 2^{GISIDS} $\rightarrow$ Boolean
IN: GISIDs $\times$ $2^{Members}$ $\rightarrow$ Boolean
IN: Members $\times$ 2^{GISIDS} $\rightarrow$ Boolean
IN: Members $\times$ $2^{Members}$ $\rightarrow$ Boolean

4.4 Semantics

4.4.1 GIS

OpenGIS predicates follow the so-called calculus-based method (CBM)[11]. The CBM allows to model the topological relationships between two 2-D spatial objects (point/-multipoint, line/multiline, polygon/multipolygon). To be more precise, let us introduce some concepts that allows to define those topological relationships:

- A point set S in $\mathbb{R}^2$ is *open* if for each of its points p there exists an $\epsilon \in \mathbb{R}$, $\epsilon > 0$, such that the disk with radius ϵ and center p is contained in S. S is *closed* if $\mathbb{R}^2$ - S is open.

- A *spatial object* is a non-empty set of points.

- The *interior* of a spatial object is the largest open set contained in it. The interior of a geometry g is called I(g).

- The *closure* of an spatial object is the intersection of all closed sets that contain it.

- The *boundary* of a spatial object is the difference between its closure and its interior.

- The *exterior* of a spatial object (with respect to the embedding space $\mathbb{R}^2$, also so-called *complement*) is the difference between $\mathbb{R}^2$ and its closure.

- The dimensionality of a geometry g is called d(g). For points/multipoints, lines/multilines and polygons/multipolygons the dimensionality is 0, 1 and 2, respectively.

We remark that the *boundary of a line* is composed by its starting and ending points (could be more than one), the other points are their interior. Besides, a point has interior but not boundary.

Now, we are ready to define the semantics of the Piet-QL predicates as follows:

- Intersects(g1, g2) or ST_Intersect(g1, g2): Returns TRUE if $g1 \cap g2 \neq \phi$. Otherwise, returns FALSE.

- Within(g1, g2) or ST_Within(g1, g2): Returns TRUE if

  $g1 \cap g2 = g1 \wedge I(g1) \cap I(g2) \neq \phi$. Otherwise, returns FALSE.

- Contains(g1, g2) or ST_Contains(g1, g2): Returns TRUE iff Within(g2, g1).

- Overlaps(g1, g2) or ST_Overlaps(g1, g2): Returns TRUE if

  $dim(I(g1)) = dim(I(g2)) = dim(I(g1) \cap I(g2)) \wedge g1 \cap g2 \neq g1 \wedge g1 \cap g2 \neq g2$. Otherwise, returns FALSE. Remark that if $dim(I(g1)) \neq dim(I(g2))$, returns FALSE.

- Crosses(g1, g2) or ST_Crosses(g1, g2): Returns TRUE if

  $((dim(g1) \neq 2 \ \vee \ dim(g2) \neq 2) \ \wedge (dim(g1) \neq 0 \ \vee \ dim(g2) \neq 0)) \ \wedge$

  $dim(I(g1) \cap I(g2)) < max(dim(I(g1)), dim(I(g2))) \wedge g1 \cap g2 \neq g1 \wedge g1 \cap g2 \neq g2$. Otherwise, returns FALSE. Remark that if both geometries are polygon or both are points, returns FALSE.

- Touches(g1, g2) or ST_Touches(g1, g2): Returns TRUE if

  $d(g1) \neq 0 \wedge d(g2) \neq 0 \wedge Int(g1) \cap Int(g2) = \phi \wedge g1 \cap g2 \neq \phi$. Otherwise, return FALSE. Remark that if both geometries are points, returns FALSE.

The semantics of the functions are defined as follows:

- Buffer(g, n) or ST_Buffer(g, n). Returns a geometry containing all points whose distance from g is less than or equal to the distance expressed by n.
- Area(g) or ST_Area(g): Returns the area of the geometry g if d(g) = 2. Otherwise, returns 0.
- Length(g) or ST_Length(g): Returns the area of the geometry g if d(g) = 1. Otherwise, returns 0.
- Intersection(g1, g2) or ST_Intersection(g1, g2): Returns the set $g1 \cap g2$.

4.4.2 OLAP

We manage only one function, called filter. Recall that this expression is of the form: filter(m, e) The semantics is that filter(m, e) returns a new set composed by those elements in m which verify the MDX logical expression e. For example, filter([Store].[Store City].Members, [Measures].[Unit Sales]>5000), returns those members in stores in cities with sales greater than 5000. The semantics of the MDX logical expression has the usual first order logic meaning.

4.4.3 GIS-OLAP and OLAP-GIS

The predicate `IN` materializes the filter between GIS-OLAP or OLAP-GIS queries. Recall that this infix predicate is of the form: id `IN` $(setofIDs)$, where id could be a spatial object identifier or an OLAP member. It has the following meaning: the predicate returns TRUE if the 'id' belongs to the set of identifiers. These identifiers, depending on the type of query, can be of spatial or non-spatial type. In other words, a `IN` S returns TRUE if (a, x) is a *gisolap tuple*, and $x \in S$.

4.4.4 Clauses

We summarize Piet-QL semantics in Table 4.1. Given that we have defined four basic forms of a query, in the first three columns on the left we indicate which component the clause in the fourth column applies to. For example, the third line in the table indicates that the `SELECT CUBE` statement applies to a sub-query in a query of GIS-OLAP type, and the fifth column explains the semantics of the OLAP sub-query.

Applies to query			Clause	Semantics
GIS	OLAP	SUB		
SELECT CLAUSE				
NO	YES	NO	SELECT CUBE <olap-list> ON ROWS [<olap-list> ON COLUMNS <olap-from-clause>	Selects the members and/or levels of the cube in <olap-from-clause>. Also indicates the axes in which the results will be returned (rows or columns). Each element selected must belong to the cube mentioned in <olap-from-clause>
YES	NO	NO	SELECT GIS <gis-list> <gis-from-clause>	Selects the attributes of the GIS layers defined in <gis-from-clause>. Each element selected must belong to some layer in <gis-from-clause>
NO	NO	YES	SELECT CUBE OLAP_MEMBERS$ <olap-from-clause>	Selects the members whose ids will be used in the IN predicate of which subquery is an argument.
NO	NO	YES	SELECT GIS GIS_LAYER$ <gis-from-clause>	Selects a layer from <gis-from-clause>. The identifiers in this layer are used for evaluating the IN predicate associated to the subquery
FROM CLAUSE				
NO	YES	YES	... FROM OLAP_CUBE$	Defines the cube over which the query will be executed. Only one cube is allowed.
YES	NO	YES	... FROM <layer-list>	Defines the GIS layers that will participate in the query. The semantics is SQL-like, i.e., the cartesian product of the geometric elements in the layers mentioned in <layer-list>
WHERE CLAUSE				
NO	YES	YES	... WHERE <olap-filter>	Selects the tuples that verify the <olap-filter> predicate.
YES	NO	YES	... WHERE <gis-filter>	Filters the results of the cartesian product of the elements in <layer-list> allowing the tuples that verify the predicate <gis-filter>
SLICE CLAUSE				
NO	YES	YES	... SLICE <olap-list>	Takes the portion of the cube corresponding to <olap-list>. All the members in the <olap-list> must belong to different dimensions.

Table 4.1: Piet-QL Semantics

Listing 4.1: Piet-QL Syntax

```
<query> ::= SELECT GIS <gis-heading> | SELECT GIS Distinct( <gis-heading> ) |
    SELECT CUBE <olap-heading>

<gis-heading> ::= <gis-list> <gis-from-clause> [ <gis-where-clause> ] [ <group
    -by-clause> ]
<gis-list> ::= GIS_ATTR$ [, <gis-list> ] |  GIS_FN$ [, <gis-list> ]
<gis-from_clause> ::= FROM <layer_list>
<layer-list> ::= GIS_LAYER$ [, <layer-list> ]
<gis-where-clause> ::= WHERE <gis-filter>
<gis-filter> ::= <gis-predicate> AND <gis-filter> | <gis-predicate> OR <gis-
    filter> | [NOT] ( <gis-filter> )
<gis-predicate> ::= GIS_ATTR$ IN ( <single-result-subquery> ) | GIS_PRED$ |
    GIS_FN$ = LITERAL$ | GIS_ATTR$ = LITERAL$
<group-by-clause> ::= <gis-list>

<olap-heading> ::= <olap-list> ON ROWS [ <olap-list> ON COLUMNS ] <olap-from-
    clause> [ <olap-where-clause> ] [ <olap-slice-clause> ] | <olap-list> ON
    COLUMNS [ <olap-list> ON ROWS ] <olap-from_clause> [ <olap-where-clause> ]
     [ <olap-slice-clause> ]
<olap-list> ::= OLAP_MEMBER$ [, <olap_list> ] |  OLAP_LEVEL$ [, <olap-list> ]
    | OLAP_FN$ [, <gis-list> ]
<olap-from-clause> ::= FROM OLAP_CUBE$
<olap-where-clause> ::= WHERE <olap-filter>
<olap-filter> ::= <olap-predicate> [ OR <olap-filter> ] | [NOT] ( <olap-filter
    > )
<olap-slice-clause> ::= SLICE <olap-list>
<olap-predicate> ::= OLAP_MEMBER$ IN ( <single-result-subquery> )   |
    OLAP_LEVEL$ IN ( <single-result-subquery> )  | OLAP_FN$ IN ( <single-
    result-subquery> )

<single-result-subquery> ::= SELECT GIS GIS_LAYER$  <gis-from-clause> [ <gis-
    where-clause> ] |  SELECT CUBE OLAP_MEMBER$ <olap-from-clause> [ <olap-
    where-clause> ] [ <olap-slice-clause> ] |
SELECT CUBE OLAP_LEVEL$ <olap-from-clause> [ <olap-where-clause> ] [ <olap-
    slice-clause> ] |
SELECT CUBE OLAP_FN$ <olap-from-clause> [ <olap-where-clause> ] [ <olap-slice-
    clause> ] |
```

4.5 Implementation Details

In our implementation, data is stored in the PostgreSQL[3] RDBMS. To solve the geometric part of a query related to the geometric section of the data model, we use Postgis[4], a plugin that adds support for geographic objects to the PostgreSQL. Actually, it enhances PostgreSQL by allowing to use it as a backend spatial database for geographic information systems. PostGIS follows the OpenGIS "Simple Features Specification for SQL" and has been certified as compliant with the "Types and Functions" profile. As our framework was developed in Java[5], we use the JTS Topology Suite[6] library, a wrapper for manipulating 2-dimensional linear geometries and also conforms to the Simple Features Specification for SQL published by the Open GIS Consortium. To solve the OLAP part of a query related to the OLAP section of the data model, we use Mondrian[7], a ROLAP server written in Java. Finally, our client is based on OpenJUMP[8], a Java-based GIS software.

When a Piet-QL query is submitted a parser decomposes it into the GIS and OLAP parts, and each one of them is executed in its corresponding engine, the former in Postgres/Postgis and the latter on Mondrian Server. The query is split into GIS and OLAP parts using `IN` predicate. The inner query is executed first, and the intermediate result set obtained is used. This result set is used to rewrite the external query. In order to clarify the process we next show two queries. The first one is a GIS query filtered by OLAP one. The latter is the opposite case (i.e., an OLAP query filtered with a GIS query).

Example 13. *We begin with the GIS-OLAP query Q7: "Names and geometries of districts crossed by the river Nete, only for those districts which contain at least one city with stores where sales have been greater than 5000".*

This query is expressed in Piet-QL as shown in Listing 4.2

[3]http://www.postgresql.org.

[4]http://www.postgis.org.

[5]http://java.sun.com.

[6]http://www.vividsolutions.com/jts.

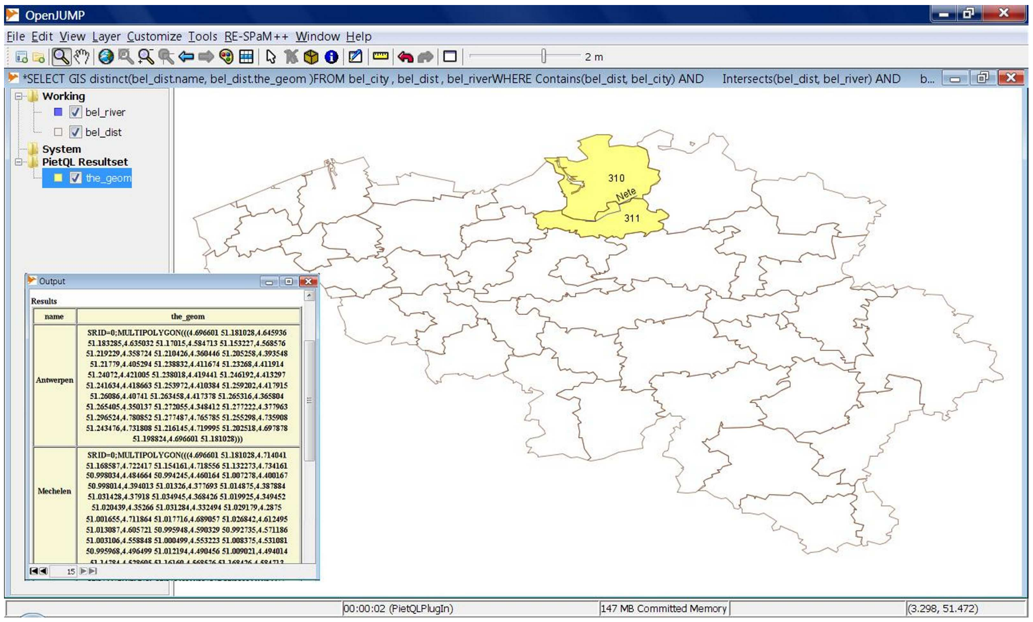
[7] http://mondrian.pentaho.org.

[8] http://www.openjump.org.

Listing 4.2: GIS-OLAP Piet-QL query

```
1 SELECT GIS distinct(bel_dist.name, bel_dist.the_geom )
2 FROM bel_city , bel_dist , bel_river
3 WHERE Contains(bel_dist , bel_city) AND Intersects(bel_dist , bel_river) AND
4         bel_river.name= 'Nete' AND bel_city IN (
5       SELECT CUBE filter([Store].[Store City].Members,
6       [Measures].[Unit Sales]>5000) FROM [Sales] )
```

Since the outer query is a GIS one, its result could be visualized in a map. In fact, there are 40 of the 43 districts in Belgium which are crossed by rivers. But only three of them contain a city with sales greater than 5000 (in our running example), namely, Antwerpen, Mechelen and Turnhout. Only the first two districts are crossed by the Nete river as shown in Figure4.2.



*SELECT GIS distinct(bel_dist.name, bel_dist.the_geom)FROM bel_city , bel_dist , bel_riverWHERE Contains(bel_dist, bel_city) AND Intersects(bel_dist, bel_river) AND b...

Results

name	the_geom
Antwerpen	SRID=0;MULTIPOLYGON(((4.696601 51.181028,4.645936 51.183285,4.635032 51.17015,4.584713 51.153227,4.568576 51.219229,4.358724 51.210426,4.360446 51.205258,4.393548 51.21779,4.405294 51.238832,4.411674 51.23268,4.411914 51.24072,4.421005 51.238018,4.419441 51.246192,4.413297 51.241634,4.418663 51.253972,4.410384 51.259202,4.417915 51.26086,4.40741 51.263458,4.417378 51.265316,4.365804 51.265405,4.350137 51.272055,4.348412 51.277222,4.377963 51.296524,4.780852 51.277487,4.765785 51.255298,4.735908 51.243476,4.731808 51.216145,4.719995 51.202518,4.697878 51.198824,4.696601 51.181028)))
Mechelen	SRID=0;MULTIPOLYGON(((4.696601 51.181028,4.714041 51.168587,4.722417 51.154161,4.718556 51.132273,4.734161 50.998034,4.484664 50.994245,4.460164 51.007278,4.400167 50.998014,4.394013 51.01326,4.377693 51.014875,4.387884 51.031428,4.37918 51.034945,4.368426 51.019925,4.349452 51.020439,4.35266 51.031284,4.332494 51.029179,4.2875 51.001655,4.711864 51.017716,4.689057 51.026842,4.612495 51.013087,4.605721 50.995948,4.590329 50.992735,4.571186 51.003106,4.558848 51.000499,4.553223 51.008375,4.531081 50.995968,4.496499 51.012194,4.490456 51.009021,4.494014

Figure 4.2: Visualizing in OpenJump the result of GIS-OLAP query Q7 in Example 13

The parser fist identifies two subqueries using the `IN` *predicate: the subquery from lines 7 to 8 in Listing 4.2 is the inner one and it is solved first. Since the expression begins with* ***`SELECT CUBE`****, it is an OLAP subquery, thus, it is translated to MDX as* *follows:*

```
SELECT filter([Store].[Store City].Members,
       [Measures].[Unit Sales]>5000) ON ROWS
FROM [Sales]
```

and is executed on the Mondrian server. The result set obtained is shown in Table 4.2 and corresponds to cities with stores with sales greater than 5000.

Cities	Antwerpen	Essen	Lier	Ravels
Sales	49,113	25,635	75,764	116,261

Table 4.2: Cities with stores that have sales greater than 5000

The `IN` predicate needs the GIS identifiers of those cities. Thus, the GIS-OLAP metadata should be queried. As we explained in Chapter3, the GIS layer city has a binding with OLAP data, it is stored in the GIS-OLAP metadata. We only show the portion of interest of these metadata in Listing 4.3

Listing 4.3: Portion of GIS and GIS-OLAP Metadata

```
<?xml version="1.0"?>
<GISSchema name="FoodMart"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="Piet.xsd">
    ...
    <!-- City layer -->
    <Layer name="bel_city" table="bel_city" primaryKey="PIET_ID"
        geometry="the_geom" descriptionField="name">
        ...
        <OLAPRelation table="gis_olap_cities"
            gisId="gisid" olapId="olapid"
            olapDimensionName="Store" olapLevelName="Store City">
            <OlapTable name="store" id="city_id"
                hierarchyNameField="store_city"/>
        </OLAPRelation>
    </Layer>
    ...
</GISSchema>
```

In line 10 of Listing4.3, we can see that the name of the mapping table is gis_olap_cities, and the attributes are denoted gisid *and* olapId *for GIS and OLAP, respectively. Thus, joining* gis_olap_cities *and the Store dimensional table* Store*, we can find the ids of those cities on the GIS side. In our example these values are 111, 243, 412 and 528. Now, we can rewrite the GIS subquery with these ids, obtaining an SQL expression that can be run in the PostgreSQL database:*

```
SELECT distinct bel_dist.name, bel_dist.the_geom
FROM bel_city , bel_dist , bel_river
WHERE Contains(bel_dist.the_geom, bel_city.the_geom) AND
      Intersects(bel_dist.the_geom, bel_river.the_geom) AND
      bel_river.name= 'Nete' AND
      bel_city.piet_id IN (111, 242, 412, 528)
```

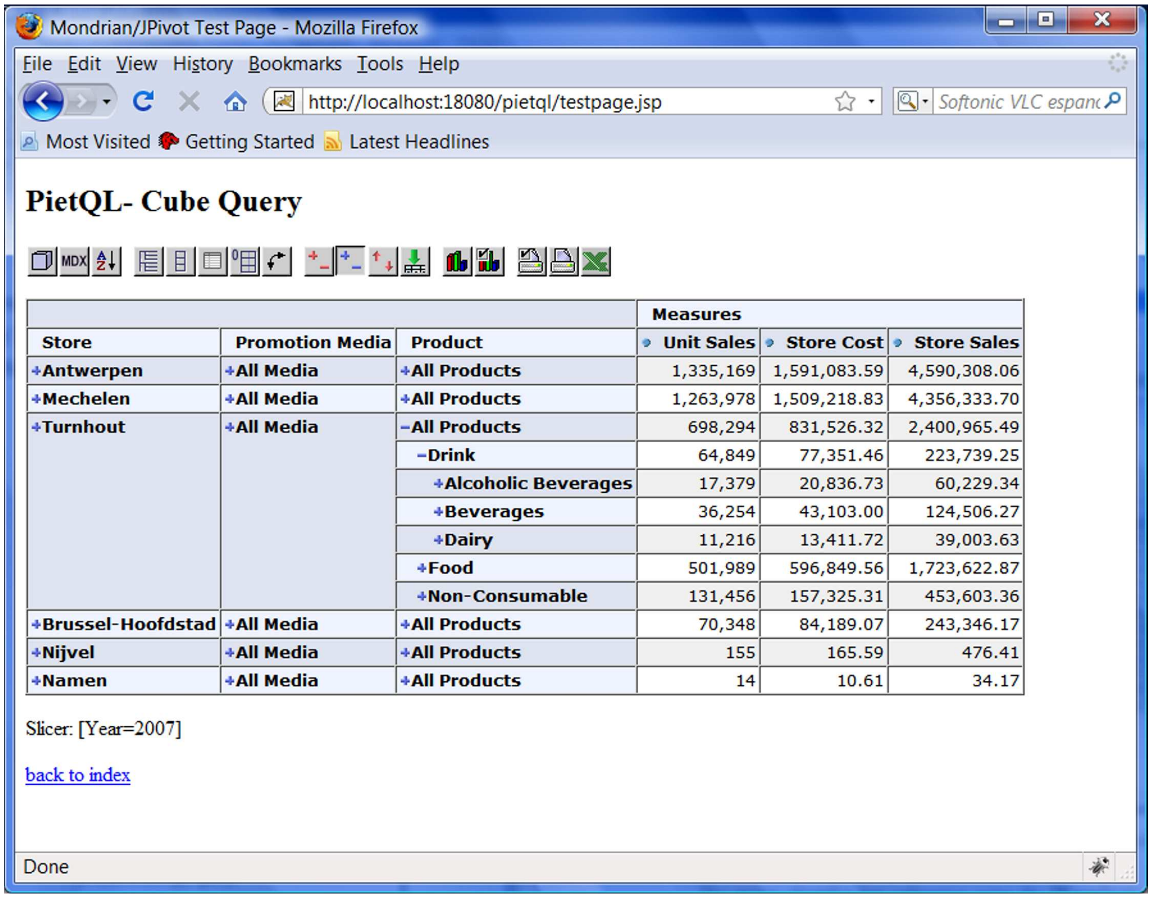
□

Example 14. *The analysis of a OLAP-GIS query is a bit more involved, because the MDX expression must be rewritten, but there is no such* `IN` *predicate in the MDX language. Suppose we submit the query Q8: "Unit Sales, Store Cost and Store Sales for the products and promotion media offered by stores in districts crossed by rivers, in 2007". This query is expressed in Piet-QL as shown in Listing 4.4*

Listing 4.4: OLAP-GIS Piet-QL query

```
SELECT CUBE [Measures].[Unit Sales], [Measures].[Store Cost],
[Measures].[Store Sales] ON COLUMNS,
[Promotion Media].[All Media], Product.[All Products] ON ROWS
FROM [Sales]
WHERE [Store].[All Stores] IN(
        SELECT GIS bel_dist
        FROM bel_dist , bel_river
        WHERE intersects(bel_river , bel_dist))
SLICE [Time].[2007]
```

The outer query is an OLAP one. Thus, its result could be browsed, rolled up, sliced, etc. The result is show in Figure4.3.


Mondrian/JPivot Test Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:18080/pietql/testpage.jsp
Softonic VLC espanc
Most Visited Getting Started Latest Headlines
MDX
Done

PietQL- Cube Query

			Measures		
Store	Promotion Media	Product	Unit Sales	Store Cost	Store Sales
+Antwerpen	+All Media	+All Products	1,335,169	1,591,083.59	4,590,308.06
+Mechelen	+All Media	+All Products	1,263,978	1,509,218.83	4,356,333.70
+Turnhout	+All Media	-All Products	698,294	831,526.32	2,400,965.49
		-Drink	64,849	77,351.46	223,739.25
		+Alcoholic Beverages	17,379	20,836.73	60,229.34
		+Beverages	36,254	43,103.00	124,506.27
		+Dairy	11,216	13,411.72	39,003.63
		+Food	501,989	596,849.56	1,723,622.87
		+Non-Consumable	131,456	157,325.31	453,603.36
+Brussel-Hoofdstad	+All Media	+All Products	70,348	84,189.07	243,346.17
+Nijvel	+All Media	+All Products	155	165.59	476.41
+Namen	+All Media	+All Products	14	10.61	34.17

Slicer: [Year=2007]

back to index

Figure 4.3: Visualizing in a web browser the result of the OLAP-GIS query Q8 of Example 14

The parser identifies two subqueries using the ***IN*** *predicate: the subquery from lines 6 to 8 in Listing 4.4 is the inner one and it is solved first. Since the expression begins with* ***SELECT GIS****, i.e. it is a GIS subquery. First, it is translated into SQL:*

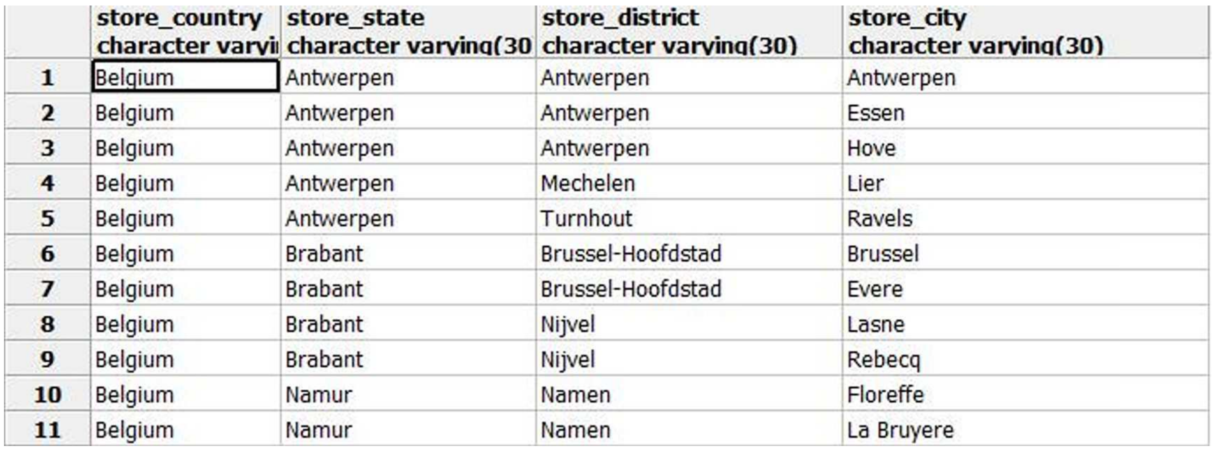
```
SELECT bel_dist.piet_id

FROM bel_dist, bel_river

WHERE intersects(bel_river.the_geom,bel_dist.the_geom)
```

and can be executed in PostgreSQL database. The result set contains the 40 districts of Belgium crossed by rivers. Analogously to the previous example, we query *the GIS-OLAP metadata to find out the name of the table which contains the binding between OLAP ids and GIS ids. In this case the table is gis_olap_districts. Thus, joining the* gis_olap_districts *and the table where the district layer is stored* (bel_dist), *we can find the ids of the districts on the OLAP side. Then, we must* build the hierarchy *in the OLAP store dimension using the OLAP metadata. Figure 4.4 shows the Store dimension table containing this information, that is used to build the following MDX expressions:*

```
[Belgium].[Antwerpen].[Antwerpen],
[Belgium].[Antwerpen].[Mechelen],
[Belgium].[Antwerpen].[Turnhout],
[Belgium].[Brabant].[Brussel-Hoofdstad],
[Belgium].[Bravant].[Nijvel],
[Belgium].[Namue].[Namen]
```



	store_country character varyi	store_state character varying(30	store_district character varying(30)	store_city character varying(30)
1	Belgium	Antwerpen	Antwerpen	Antwerpen
2	Belgium	Antwerpen	Antwerpen	Essen
3	Belgium	Antwerpen	Antwerpen	Hove
4	Belgium	Antwerpen	Mechelen	Lier
5	Belgium	Antwerpen	Turnhout	Ravels
6	Belgium	Brabant	Brussel-Hoofdstad	Brussel
7	Belgium	Brabant	Brussel-Hoofdstad	Evere
8	Belgium	Brabant	Nijvel	Lasne
9	Belgium	Brabant	Nijvel	Rebecq
10	Belgium	Namur	Namen	Floreffe
11	Belgium	Namur	Namen	La Bruyere

Figure 4.4: The members and levels used to build the hierarchy

Finally, the original OLAP query is rewritten as a crossjoin of these hierarchies and run on the Mondrian OLAP server:

```
SELECT {[Measures].[Unit Sales],[Measures].[Store Cost],
[Measures].[Store Sales]} ON COLUMNS,
Crossjoin( Hierarchize(
{[Store].[All Stores].[Belgium].[Antwerpen].[Antwerpen],
[Store].[All Stores].[Belgium].[Namur].[Namen],
[Store].[All Stores].[Belgium].[Brabant].[Nijvel],
[Store].[All Stores].[Belgium].[Brabant].[Brussel-Hoofdstad],
[Store].[All Stores].[Belgium].[Antwerpen].[Mechelen],
[Store].[All Stores].[Belgium].[Antwerpen].[Turnhout]} ),
{([Promotion Media].[All Media],[Product].[All Products])} ) ON ROWS
FROM [Sales]
WHERE [Time].[2007]
```

□

4.6 Piet-QL theoretical foundation

In Section 3.4 we defined a formal FO-language based on the notion of geometric aggregation, and introduced the concept of *summable queries*. This language, although powerful, is clearly not appropriate for an end user, although it provides the theoretical basis for a more practical query language along the lines of the well-known SQL. Let us consider the following summable query "Total population of the districts crossed by the *Dijle* river", where the function ft_{pop} (i.e., the fact table) maps elements of *dom*(Polygon) to populations. It reads in our formal language:

$$\sum_{\mathrm{pg_{id}} \in C} ft_{pop}(\mathrm{pg_{id}}).$$

$$\begin{aligned} C = \{pg_{id} \mid (\exists x)(\ \exists y)(\exists pl_{id})(\exists d) \\ (\alpha(L_r, \mathrm{Rivers}, \mathrm{Ri}, \mathrm{Pl}, \mathrm{Dijle}) = pl_{id}\ \wedge r_{alg}(L_r, \mathrm{Pl}, x, y, pl_{id})\ \wedge \\ r_{alg}(L_d, \mathrm{Pg}, x, y, pg_{id})\ \wedge \alpha(L_d, \mathrm{Districts}, \mathrm{Di}, \mathrm{Pg}, d) = pg_{id})\}. \end{aligned}$$

Implementing a query language using the operators defined by OpenGIS we can simplify this expression. For this, we can get rid of the algebraic part of the formal data model (r_{alg}), replacing it by conditions over geometries as we show below. We make a substantial change to the original language with the introduction of variables that range over the *extensions of the geometric objects.* These variables can also be existentially or universally quantified, and provide a natural link between theory and implementation, since the extensions can be mapped to data of the types supported by standard GIS languages. We add the following terms to the language:

- a 3-ary relation, r_{ext} representing that a geometry extent corresponds to a geometry identifier in the geometric part. It is of the form $\langle L_i, g_{id}, g_{ext}\rangle$, where L_i is a layer name, g_{id} is an instance of a geometry and g_{ext} is an extent of a geometry.
- the OpenGIS predicates treated as relations.

In the example above, the sub-expression

$$r_{alg}(L_r, \mathrm{Pl}, x, y, pl_{id}) \ \wedge r_{alg}(L_d, \mathrm{Pg}, x, y, pg_{id}).$$

can be replaced by

$$r_{ext}(L_r, pl_{id}, pl_{ext}) \ \wedge r_{ext}(L_g, pg_{id}, pg_{ext}) \ \wedge intersects(pl_{ext}, pg_{ext}).$$

Finally, the original summable query is rewritten by means of OpenGIS operators and the new r_{ext} relation, as follows:

$$\sum_{\mathrm{pg_{id}} \in C'_2} ft_{pop}(\mathrm{pg_{id}}).$$

$$C'_2 = \{pg_{id} \mid (\exists pl_{ext})(\ \exists pg_{ext})(\exists pl_{id})(\exists d)$$
$$(\alpha(L_r, \mathrm{Rivers}, \mathrm{Ri}, \mathrm{Pl}, \mathrm{Dijle}) = pl_{id} \ \wedge r_{ext}(L_r, pl_{id}, pl_{ext}) \ \wedge r_{ext}(L_d, pg_{id}, pg_{ext}) \ \wedge$$
$$intersects(pl_{ext}, pg_{ext}) \ \wedge \alpha(L_d, \mathrm{Districts}, \mathrm{Di}, \mathrm{Pg}, d) = pg_{id})\}.$$

This simplified model provides the formal basis of the spatial aggregation of Piet-QL.

Example 15. *Consider the Piet-QL query*
Q9 (OLAP-GIS): "Store Cost of stores in provinces crossed by rivers", reads in Piet-QL:

```
SELECT CUBE [Measures].[Store Cost] ON COLUMNS
FROM [Sales]
WHERE [Store].[All Stores] IN(
  SELECT GIS bel_prov
  FROM bel_prov, bel_river
  WHERE ST_INTERSECTS(bel_river.the_geom,bel_prov.the_geom)
 )
```

This query contains an aggregation over the measure "store cost". This aggregation must be calculated over the stores in provinces crossed by rivers. The geometric part is expressed in the simplified formal language by building the set C with the extents of provinces crossed by rivers. In the simplified formal language it reads:

$$\sum_{\mathrm{pg_{id}} \in C} ft_{cost}(\mathrm{pg_{id}}).$$

$$\begin{aligned} C = \{pg_{id} \mid (\exists pl_{ext})(\ \exists pg_{ext})(\exists pl_{id})(\exists d) \\ (r_{ext}(L_r, pl_{id}, pl_{ext})\ \wedge r_{ext}(L_p, pg_{id}, pg_{ext})\ \wedge \\ intersects(pl_{ext}, pg_{ext})\ \wedge \alpha(L_p, \mathrm{State}, \mathrm{Si}, \mathrm{Pg}, d) = pg_{id})\}. \end{aligned}$$

□

4.7 Comparison of our proposal with Related Work

Shekhar *et al.* [63] introduced MapCube, a visualization tool for spatial data cubes. MapCube is an operator built for spatial data warehouses. By taking a base map, cartographic preferences and an aggregation hierarchy, MapCube produces an album of maps to browse results of aggregation via roll-up and drill-down operations. In than sense, it is more aligned with the idea of maps navigation. Our Piet-QL implementation allows displaying geometries in maps when the spatial component is returned by a query but cannot be drilled-down or rolled-up *á la SOLAP*, as, for example, in the map cube operator commented above. Although navigation with (map) cubes are necessary when interactive queries are posed by analysts, our query language was designed with the idea of integrating spatial and OLAP information with moving objects scenarios. That is the reason why we prefer to focus on how to use the result set answer by the engine as part of a more powerful query language instead of adding navigation features.

GeoMDQL [14] proposal consists in defining a so-called Geographic Data Warehouse (GDW) and a Geographical Data Cube. The GDW stores geographic objects, instead of other tightly-coupled approaches that store in the data warehouse pointers to geographic objects. The query language of GeoMDQL is completely based on MDX. This MDX extension is designed to deal with geographic objects. Instead, we propose a mixed query language with the goal of using SQL for the GIS query part. Similarly to our approach, they show the query result in tables and if possible in a map. The information on tables can be browsed with typical OLAP operators (as our Piet-QL Web implementation [9]), but the information visualized on maps does not provide this interaction (unlike our Piet-QL stand-alone OpenJump implementation, that provides it). A very important difference is that GeoMDQL cannot deal with spatial measures. Piet-QL supports spatial measures, as we can manage spatial aggregation in GIS-OLAP and Pure GIS queries. Related with this, GeoMDQL cannot run pure GIS queries.

[9]http://piet.exp.dc.uba.ar/pietql

4.8 Summary

We presented a query language, denoted Piet-QL, supporting the Piet data model, which can express complex integrated GIS and OLAP queries in a natural an concise way. Its spatial aggregation part is based on the FO language proposed in Chapter 3. Piet-QL supports the operators proposed by the Open Geospatial Consortium for SQL. In addition, it incorporates the necessary syntax to integrate OLAP operations through the OLAP standard MDX. We gave detail of Piet-QL syntax and semantics and provide an extensive set of queries that show the power of the language and its main features, and show how the queries are translated to the underlying database languages.

Chapter 5

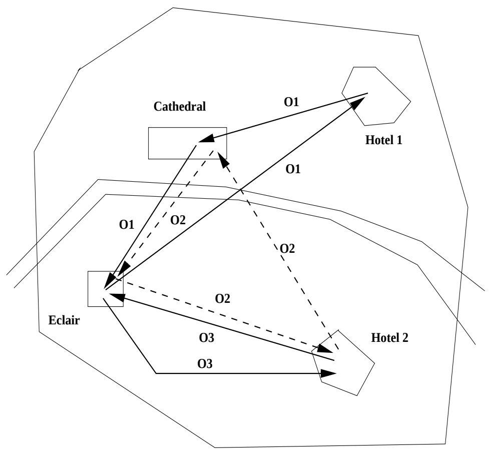
Moving Object Trajectories

Moving objects carrying location-aware devices produce trajectory data in the form of a sample of (O_{id}, t, x, y)-tuples, that contain object identifier and time-space information. In this Chapter we introduce a formal model for integrating moving object data with the Piet data model (Chapter 3). We then study the representation of trajectory data produced by moving objects using the notion of semantic trajectories and discuss its consequences.

5.1 Introductory Example

We motivate our work with the following example. Figure 5.1 (left) shows a simplified map of Belgium, containing two hotels, denoted Hotel 1 and Hotel 2 (H1 and H2 from here on), the Cathedral of Our Lady and the Eclair Bruxelles Sport Club. We consider three moving objects: O1, O2 and O3. Object O1 goes from H1 to the Cathedral, the Eclair, spends just a few minutes there, and returns to the hotel. Object O2 goes from H2 to the Cathedral, the Eclair (spending a couple of hours visiting each place), and returns to the hotel. Object O3 leaves H2 to the Eclair, visits the place, and returns to H2. Figure 5.1 (center) shows part of a table containing the raw trajectories (i.e., expressed for each object as $\langle t, x, y \rangle >$ tuples). All points of the same trajectory are temporally ordered and stored together (i.e., the raw trajectories table is sorted by O_{id} and t). In what follows, we use the object identifier as the trajectory identifier, unless specified.

Many useful applications arise in this scenario. For instance, a GIS user may be



O_{id}	t	x	y
O_1	1	x_1	y_1
O_1	2	x_2	y_2
O_1	3	x_3	y_3
O_1	4	x_4	y_4
...	...	...	...
O_2	5	x_5	y_5
O_2	6	x_6	y_6
O_2	7	x_7	y_7
...	...	...	...
O_3	4	x_5	y_5
O_3	5	x_8	y_8
O_3	6	x_9	y_9
...	...	...	...

O_{id}	g_{id}	t_s	t_f
O_1	H_1	1	10
O_1	C	20	30
O_1	H_1	100	140
O_2	H_2	5	20
O_2	C	25	40
O_2	E	50	80
O_2	H_2	120	140
O_3	H_2	4	10
O_3	E	15	40
O_3	H_2	60	140

Figure 5.1: Running example (left), its moving object fact table (center), and its compressed fact table (right)

interested in finding out trajectory information, like "number of persons going from H1 to the Cathedral of Our Lady and then to the Eclair Sport Club (stopping to visit both places) in the same day". An analyst may also want to discover hidden information using data mining techniques. For instance, she would like to identify interesting patterns in the trajectory data using association rule mining. She may also want to verify a certain pattern, like "people do not visit two castles in the same day". Complex queries that aggregate non-spatial information and also involve GIS and moving object data must also be addressed. For instance, "total sales in stores located on the left bank of the Rupel river, such that people visit them before going to the Castle of Veves in the same day".

To address these problems, we need a common framework that can integrate moving objects, GIS data and non-spatial data. In this chapter we show that the Piet data model can be naturally extended to provide this capability.

5.2 Trajectories

We define a fact table denoted *Moving Object Fact Table* (MOFT), that stores a collection of moving objects spatiotemporal locations. Ordering this locations over time, leads to the concept of *trajectory*. Let us first define this concept formally.

Definition 6. [*Trajectory*] *A* trajectory *is a list of time-space points* $\langle(t_0, x_0, y_0), (t_1, x_1, y_1), ..., (t_N, x_N, y_N)\rangle$, *where* $t_i, x_i, y_i \in \mathbb{R}$ *for* $i = 0, ..., N$ *and* $t_0 < t_1 < \cdots < t_N$. *We call the interval* $[t_0, t_N]$ *the* time domain *of the trajectory.* □

For the sake of finite representability, we may assume that the time-space points (t_i, x_i, y_i) have rational coordinates. A MOFT (see the table in the center of Figure 5.1) contains a finite number of identified trajectories.

Definition 7. [*Moving Object Fact Table*] *Given a finite set* $\mathcal{T}$ *of trajectories, a* Moving Object Fact Table *(MOFT) for* $\mathcal{T}$ *is a relation with schema* $< Oid, T, X, Y >$, *where* Oid *is the identifier of the moving object,* T *represents time instants, and* X *and* Y *represent the spatial coordinates of the objects. An instance* $\mathcal{M}$ *of the above schema contains a finite number of tuples of the form* (O_{id}, t, x, y), *that represent the position* (x, y) *of the object* O_{id} *at instant* t *for the trajectories in* $\mathcal{T}$. □

5.3 A Data Model for Compressed Trajectories

In practice, the MOFTs can contain huge amounts of data. For instance, suppose that a GPS takes observations of daily movements of one thousand people, every ten seconds, during one month. This gives a MOFT of 1000×360× 24× 30 = 259,200,000 records. In this scenario, querying raw trajectory data may become extremely expensive. Moreover, note that a MOFT only provides the position of objects at a given instant. Sometimes we are not interested in such level of detail, but we look for more aggregated information instead. For example, we may want to know how many people go from a hotel to a sport club on weekdays. Or we can even want to perform data mining tasks like inferring trajectory patterns that are hidden in the MOFT. These tasks require *semantic information*, not present in the MOFT. In the best case, obtaining this information from that table will be expensive, because it would imply a join between this table an the spatial data. As a solution, we propose to use the notion of *stops and moves* in order to obtain a more concise MOFT, that can represent a trajectory in terms of places of interest for a particular application, characterized as *stops*. This table cannot replace the whole information provided by the MOFT, but allows to quickly obtain information of interest without accessing the complete data set.

We first define the notion of "place of interest of an application" and then formalize the concept of *stops* and *moves*. We base ourselves on the definition by Alvarez *et al* [5].

Definition 8. [*Place of Interest*] *A* place of interest (PoI) *C is a tuple (R_C, Δ_C), where R_C is a (topologically closed) polygon, polyline or point in $\mathbb{R}^2$ and Δ_C is a strictly positive real number. The set R_C is called the* geometry *of C and Δ_C is called its* minimum duration. *The* places of interest of an application *$\mathcal{P}_{\mathcal{A}}$ is a finite collection of PoIs with mutually disjoint geometries.* □

Definition 9. [*Stops and Moves of a Trajectory*] *Let $T = \langle(t_0, x_0, y_0), (t_1, x_1, y_1), ..., (t_n, x_n, y_n)\rangle$ be a trajectory. Also, $\mathcal{P}_{\mathcal{A}} = \{C_1 = (R_{C_1}, \Delta_{C_1}), ..., C_N = (R_{C_N}, \Delta_{C_N})\}$.*

A stop *of T* with respect to *$\mathcal{P}_{\mathcal{A}}$ is a maximal contiguous sub-trajectory $\langle(t_i, x_i, y_i), (t_{i+1}, x_{i+1}, y_{i+1}), ..., (t_{i+\ell}, x_{i+\ell}, y_{i+\ell})\rangle$ of T such that for some $k \in \{1, ..., N\}$ the following holds: (a) $(x_{i+j}, y_{i+j}) \in R_{C_k}$ for $j = 0, 1, ..., \ell$; (b) $t_{i+\ell} - t_i > \Delta_{C_k}$.*

A move *of T* with respect to *$\mathcal{P}_{\mathcal{A}}$ is: (a) a maximal contiguous subtrajectory of T in between two temporally consecutive stops of T; (b) maximal contiguous subtrajectory of T in between the starting point of T and the first stop of T; (c) a maximal contiguous subtrajectory of T in between the last stop of T and ending point of T; (d) the trajectory T itself, if T has no stops.* □

Figure 5.2 illustrates these concepts. Here, there are four places of interest with geometries $R_{C_1}, R_{C_2}, R_{C_3}$ and R_{C_4}. The trajectory T is depicted by linearly interpolating between its sample points, to indicate their order. Let us imagine that T is run through from left to right. If the three sample points in R_{C_1} are temporally far enough apart (longer than Δ_{C_1}), they form a stop. Imagine that further on, only the two sample points in R_{C_4} are temporally far enough apart to form a stop. Then we have two stops in this example and three moves.

We remark that our definition of stops and moves of a trajectory is arbitrary and can be modified in many ways. For example, if we would work with linear interpolation of trajectory samples, rather than with samples, we see in Figure 5.2, that the trajectory briefly leaves R_{C_1} (not in a sample point, but in the interpolation). We could incorporate a tolerance for this kind of small exits from PoIs in the definition, if we define stops and moves in terms of continuous trajectories, rather than on terms of samples. Finally, in what follows we assume that samples are taken at regular and relatively short intervals. The following property is straightforward.

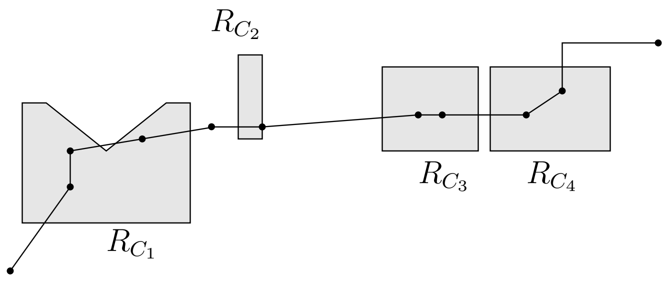


Figure 5.2: An example of a trajectory with two stops and three moves.

Proposition 1. *There is an algorithm that returns, for any input $(\mathcal{P}_{\mathcal{A}}, T)$ with $\mathcal{P}_{\mathcal{A}}$ the places of interest of an application, and T a trajectory $\langle(t_0, x_0, y_0), (t_1, x_1, y_1), ..., (t_n, x_n, y_n)\rangle$, the stops of T with respect to $\mathcal{P}_{\mathcal{A}}$. This algorithm works in time $\mathcal{O}(n \cdot p)$, where p is the complexity of answering the point-query [60].* □

Proof. We provide the algorithm in Section 5.3.2. □

5.3.1 Compressing the MOFT

In this section, we describe how we go from MOFTs to application-dependent compressed MOFTs, where (O_{id}, t_i, x_i, y_i) tuples are replaced by $(O_{id}, g_{id}, t_s, t_f)$ tuples. In the latter, O_{id} is a moving object identifier, g_{id} is an identifier of the geometry of a place of interest and t_s and t_f are two instants that encode the time interval $[t_s, t_f]$ of a stop. The idea is to replace the MOFT (containing the raw trajectories) by other table that represents the same trajectory more concisely by listing its stops and the time intervals spent in them. In this sense, the concise MOFT, which we denote SM-MOFT (standing for Stops and Moves-MOFT), behaves like a summarized materialized view of the MOFT. The SM-MOFT contain the object identifier, the identifier of the geometries representing the Stops, and the interval $[t_s,\ t_f]$ of the stop duration. Notice that we do not need to store the information about the moves, which remains implicit, because we know that between two stops there could only be a move. Also, if trajectory passes through a PoI, but remains there an insufficient amount of time for considering the place a trajectory stop, the stop is not recorded in the SM-MOFT.

Definition 10. [*$\mathcal{S}M$-MOFT*] *Let $\mathcal{P}_{\mathcal{A}} = \{C_1 = (R_{C_1},\ \Delta_{C_1}), ..., C_N = (R_{C_N}, \Delta_{C_N})\}$ be the PoIs of an application, and let $\mathcal{M}$ be a MOFT. The $\mathcal{S}M$-*MOFT *$\mathcal{M}^{sm}$* of *$\mathcal{M}$* with respect to *$\mathcal{P}_{\mathcal{A}}$ consist of the tuples $(O_{id}, g_{id}, t_s, t_f)$ such that (a) O_{id} is the identifier of a trajectory in $\mathcal{M}$[1]; (b) g_{id} is the identifier of the geometry of a PoI $C_k = (R_{C_k}, \Delta_{C_k})$ of $\mathcal{P}_{\mathcal{A}}$ such that the trajectory with identifier O_{id} in $\mathcal{M}$ has a stop in this PoI during the time interval $[t_s, t_f]$. This interval is called the* stop interval *of this stop.* □

Figure 5.1 (right) shows the $\mathcal{S}M$-MOFT for our running example.

5.3.2 Computing the SM-MOFT

We first give details of the computation of the SM-MOFT from the MOFT containing the raw trajectories. We process the MOFT one trajectory at a time. A cursor is placed at the first tuple of the trajectory, and only two points need to be in main memory at the same time. We use the automaton shown in Figure 5.3 to detect the sequence of PoIs that can become a *stop* in a trajectory. The transitions in this automaton can be either a *readPoint()* action, or the empty string λ. There are four states in the automaton: *StartTrajectory*, *EndTrajetory*, *InsidePOI* and *OutsidePOI*.

StartTrajectory: This is the initial state. If the first point in the trajectory belongs to a PoI, the transition is to the *InsidePOI* state (we have recognized the beginning of a PoI). If not, the transition is to the OutsidePOI state.

InsidePOI: This state can be reached from any state, except EndTrajectory. Different situations must be analyzed:

[1]We could also use a trajectory identifier other than the object's id, if we want to analyze several trajectories of an object in different days. We use this approach in Section 2.1.4)

- The previous states were *OutsidePOI* or *StartTrajectory*. In the first case, the *previous* point must belong to a move. In the latter, we are at the start of trajectory. The *current* point corresponds to a PoI, which is a candidate to become a stop (we call this a **candidate stop**). This time instant of the PoI becomes the initial time of the interval of this potential stop.

- The previous state was *InsidePOI*: if two consecutive points (the *previous* and the *current* ones) are both *inside the same PoI*, then the action will be: read the next input (i.e., move to the next point). Otherwise, we have reached the boundary of the PoI, and we are entering another one; thus, before reading the next input, we need to compute the *duration* of the interval in order to check if the sub-trajectory inside the PoI was actually a stop. If we are using *trajectory sampling*, the timestamp of the *previous* point is the ending time of the stop interval. The timestamp of the *current* point is used as the starting time of the interval of the new PoI the object is entering. If we are using *linear interpolation*, we build a line between both points and calculate the intersection between this line and the PoI (and, of course, the corresponding time instant).

OutsidePOI: this intermediate state can be reached from any state, except *EndTrajectory*. Again, different situations must be analyzed:

- The previous states were *OutsidePOI* or *StartTrajectory*. In the first case, the *previous* point must belong to a move. In the latter we are at the start of the trajectory. The algorithm reads the next input point.

- The previous state was *InsidePOI*: the automaton has detected that the object has left a candidate stop, and proceeds as explained above, computing the duration of the candidate stop top define if the object is still within a move, or if it has found a *stop*.

EndTrajectory: the last state when the cursor has consumed all the tuples in the MOFT.

To give an idea of practical results, in our case study, starting from a MOFT containing 30,808,296 tuples, we obtained an SM-MOFT with 105,684 tuples (i.e., 0.343% of the original size).

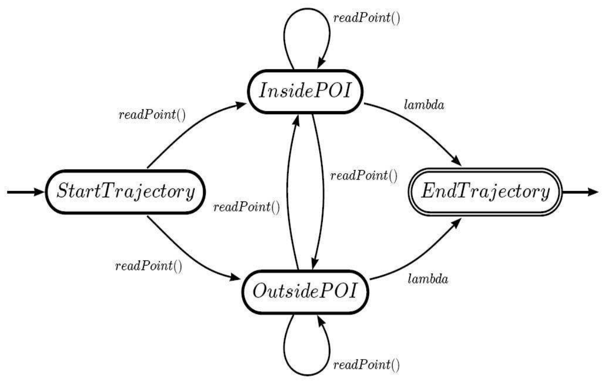


Figure 5.3: Automata for Stops and Moves calculation

5.4 Summary

We have proposed to rewrite a raw trajectory using the concept of *stops* and *moves*. The benefits of the approach are twofold. On the one hand, it leads to compressed trajectories, a desirable feature in MO databases. On the other hand, since those PoIs are application-dependant, they can contain additional interesting information which enriches the physical locations of those places and expands the scope on the analysis.

Chapter 6

RE-SPaM: A Data Mining Language for Semantic Trajectories

In this chapter we show how we can apply sequential pattern mining algorithms to semantic trajectories, in order to obtain interesting information from a collection of trajectory samples. These algorithms are based on the apriori principle, first applied in the field of Association Rule Mining [3]. A problem with the usual apriori-based algorithms is that they return *all* frequent sequences present in a database, although in general only a few ones are interesting from a user's point of view. Thus, post-processing tasks are required in order to discard uninteresting sequences. In scenarios dealing with a large number of patterns this strategy could be tedious and costly. To avoid this drawback, languages based on regular expressions (REs) were proposed to restrict frequent sequences to the ones that satisfy user-specified constraints. Proposals like SPIRIT [20, 21] are aimed at pruning uninteresting sequences using regular expressions to express these constraints. However, when mining huge volumes of data and we need to express complex constraints, existing approaches do not suffice for an effective pruning phase. For instance, in all of these languages, REs are applied over items, which limits their applicability in complex real-world situations. In this chapter we present a more powerful language, based on regular expressions, denoted RE-SPaM (standing for Regular Expression over Sequential Pattern Mining), where the basic elements are constraints defined over the (temporal and non-temporal) attributes of the items to be mined. Expressions in this language may include attributes, functions over attributes, and variables. We specify the syntax and semantics of RE-SPaM, and present a comprehensive set of examples to illustrate its expressive power. We also study in detail how the expressions can be used to prune the resulting sequences in the mining process. In addition, we introduce techniques that allow pruning sequences in the early stages of the process, reducing the need of accessing the database, making use of the categorization of the attributes that compose the items, and of the automaton that accepts the language generated by the RE. Note that although in this thesis we focus on trajectory databases, our mining approach is general enough for being applied to other settings.

6.1 Motivation

In Chapter 5 we defined the concept of semantic trajectories, which are produced replacing a sequence of <t, x, y>-tuples by a sequence of *stops* and *moves*, defined from the collection of PoIs. To introduce the problem, let us consider a set of PoIs corresponding to restaurants and banks. Assume that restaurants are characterized by their specialities, i.e., French and Italian food. Figure 6.1 depicts a simplified view of those PoIs. Banks are represented by orange circles, Italian restaurants by purple circles and French restaurants by green ones. There are also three compressed trajectories, let us call them t1, t2 and t3, respectively. Trajectories t1 and t3 visit exactly the same sequence of PoIs. Trajectory t2 visits the same bank, then an Italian restaurant (different than the one visited by t1 and t3), the same French restaurant than t1 and t3, and finishes at a bank (again, not the same bank than in the other trajectories). Trajectories t1 and t2 present the same movement pattern (they visit exactly the same places), but t3 may appear to follow a different one. However, for many applications, we may consider all the Italian restaurants as equivalent with respect to the attribute *type of food*. Analogously, if we consider that all banks offer basically the same service, we can say that t1, t2 and t3 are *semantically equivalent* (i.e., equivalent with respect to some property or set of properties).

Traditional approaches only use implicitly the notion of equality of places, i.e., they consider that two trajectories are similar if they visit the same places. We believe that when dealing with *semantic trajectories* the notion of equality of places

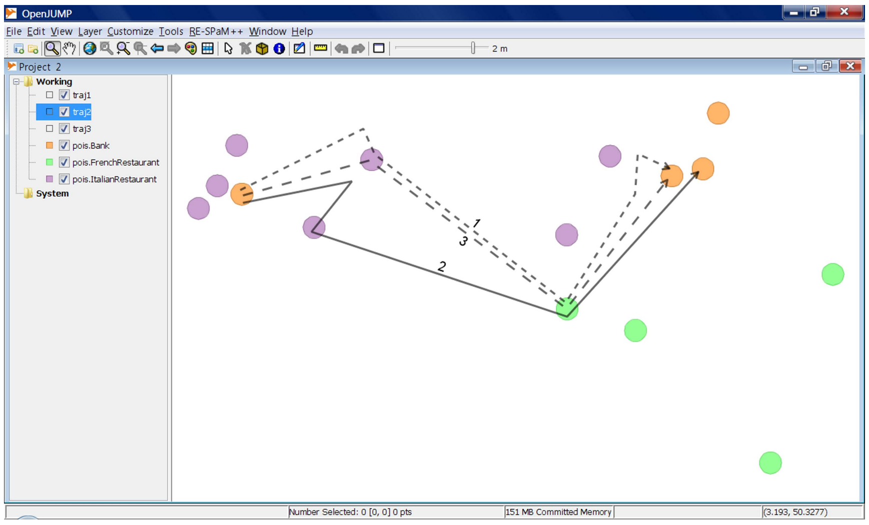


Figure 6.1: Trajectories semantically equivalent

must be relaxed, meaning that the concept of *semantic equivalence of places* must be considered. Intuitively, two places are *semantically equivalent* if both of them are characterized by the same value of some relevant/s attribute/s. In our example above, the two Italian restaurants were considered *semantically equivalent* because they serve the same speciality. Other attributes could be considered, for instance their prices (expensive, cheap), the rule of accepting or not pets, etc. Notice that the notion of *equality of places* implies the notion of *equivalence of places*, but the latter lets us detect new behavioral patterns impossible to discover using the comparison of places in a strict sense. With this in mind, we present a mining algorithm where the output can be restricted to the patterns that satisfy a set of constraints expressed by means of regular expressions over the attributes of the PoIs. This language also supports variables and functions, as we explain below.

Throughout the chapter we refer to a tourist application in Belgium. The items to be mined are sequences of *stops* composed of the PoIs visited by tourists, the time spent by each moving object at each *stop*, and the attributes of the PoIs. Each item can be classified as belonging to a category described by a set of attributes. In our example we have four categories: hotels, restaurants, castles and zoos, with different attributes and number of occurrences. In this scenario our ultimate goal is to discover frequent sequential patterns for moving objects (although the approach could be used in any application domain), restricting these patterns to the ones that satisfy a set of constraints, specified by means of regular expressions over the attributes of the objects. These constraints are of the form “trajectories that first visit cheap restaurants, then go to a 3-star hotel, and finish at the first restaurant”.

6.2 Data Model

Our sequential pattern algorithm is based on the ideas of the Generalized Sequential Patterns algorithm (GSP) [65], but restricts the patterns to be obtained via a regular expression. Regular expressions in RE-SPaM can contain constants, attributes, and variables in a way that substantially expands the expressiveness of the constraints considered in previous proposals. We also allow *functions over attributes.* These functions can be defined, for instance, in a relational database, a multidimensional database (in the form of a rollup function [10]), or as a Web Service. We also remark that, in the work of *Srikant et al.* [65], only items (IDs) are allowed to participate on hierarchies. We extend this idea allowing any kind of attribute (including temporal ones) in such hierarchies. We show that supporting attribute, variables, and categorization, implies not merely an extension of previous proposals, but introduces new theoretical and practical problems, providing a powerful language for sequential pattern mining, relevant to many real-world applications, in particular MO scenarios.

Remark 1. *The incorporation of functions in the RE-SPaM regular expression lets us bind* stops *and* moves *with the Piet data model, materializing the integration of the three worlds mentioned in Section 1.3. This will become clear in the next chapter.*

Before introducing the language we present a formal model and then define the regular language supporting it. As usual in databases, we work with the notion of a *schema* and its associated *instances.* We have a set of attribute names $\mathbf{A}$, and a set of identifier names $\mathbf{I}$. Each attribute $attr \in \mathbf{A}$ is associated with a set of values in $dom(attr)$, and each identifier $ID \in \mathbf{I}$ is associated with a set of values in $dom(ID)$.

Definition 11. [*Category Schema*] *A* category schema *S is a pair (ID, A), where $ID \in \boldsymbol{I}$ is a distinguished attribute denoted* identifier, *and $A = \{attr | attr \in \boldsymbol{A}\}$. Without loss of generality, and for simplicity, in what follows we consider the set A ordered. Thus, S has the form $(ID, attr_1, ..., attr_n)$.* □

Definition 12. [*Category Occurrence*] *Given a category schema S, a category occurrence for S is the pair $(\langle ID, id\rangle, P)$, where ID is the ID attribute of Definition 11 above, $id \in dom(ID)$, and P is the set of pairs $[(attr_1, v_1), ..., (attr_n, v_n)]$, where: (a) $attr_i = A(i)$ (remember that A is considered ordered); (b) $v_i \in dom(attr_i), \forall i, i = 1..n$; (c) All the occurrences of the same category have the same set of attributes; (d) ID is unique for a category occurrence, meaning that no two occurrences of the same category can have the same value for ID (see below).* □

Remark 2. *In what follows, for clarity reasons, we assume that $attr_0$ stands for ID. Thus, a category occurrence is the set of pairs $[(attr_0, v_0), (attr_1, v_1), ..., (attr_n, v_n)]$.* □

Definition 13. [*Category Instance*] *A set of occurrences of the same category is denoted a* category instance. *Also, given set of category instances (see Table 6.2), we extend the fourth condition in Definition 12 to hold for the whole set: ID is unique for a set of category instances, meaning that no two occurrences of categories in the set can have the same value for ID.* □

Example 16. *The schemas of the four categories in our running example are shown in Table 6.1. The corresponding set of category instances is shown in Table 6.2 (for example, the category* hotels *has two occurrences).* □

Adding a time interval to a category occurrence, produces an **item**. The time interval of an item is described by its initial and final instants, and denoted [ts, tf]. Definition 14 spells the above out.

Category	Schema
hotels	$[\mathbf{ID}, categoryName, geom, star]$
restaurants	$[\mathbf{ID}, categoryName, geom, price, typeOfFood]$
castles	$[\mathbf{ID}, categoryName, geom, name]$
zoos	$[\mathbf{ID}, categoryName, geom, price]$

Table 6.1: Schema of the categories in the running example

Category	Instance
hotels (2 occurrences)	$[(ID, H1), (categoryName, hotel), (geom, pol1), (star, 3)]$ $[(ID, H2), (categoryName, hotel), (geom, pol2), (star, 5)]$
restaurants (3 occurrences)	$[\ (ID, R1),\ (categoryName, restaurant),\ (geom, pol3),\ (price, cheap),\ (typeOfFood, French)\ ]$ $[\ (ID, R2),\ (categoryName, restaurant),\ (geom, pol4),\ (price, expensive),\ (typeOfFood, French)\ ]$ $[\ (ID, R3),\ (categoryName, restaurant),\ (geom, pol5),\ (price, cheap),\ (typeOfFood, Italian)\ ]$
castles (1 occurrence)	$[(ID, B), (categoryName, castle), (geom, pol6), (name, Belfort\ \ Castle)]$
zoos (1 occurrence)	$[(ID, Z), (categoryName, zoo), (geom, pol7), (price, cheap)]$

Table 6.2: Set of instances for the categories in Table 6.1

Definition 14. [*Item*] *Let S be a category schema, and $O(S)$ a category occurrence of the form $[(ID, v), (attr_1, v_1),\ldots,(attr_n, v_n)]$. An* item *$I$ associated with $O(S)$ is the set of pairs: $[(ts, v_{ts}),\ (tf, v_{tf}),\ (ID, v),\ (attr_1, v_1), (attr_n, v_n)]$, where ts and t_f are temporal attributes corresponding to the beginning and ending of the time interval of the occurrence, and v_{ts} and v_{tf} are actual values for these attributes.* □

Remark 3. *In the sequel, we decompose the temporal attributes ts and tf into date and time parts of the form ts_date, ts_time, tf_date and tf_time, respectively. This allows, for example, to talk easily about the different parts of the day, and an implementation can make use of the many features provided by DBMSs to handle temporal data types. Nevertheless, it must be clear that we can indistinctly use both forms of referring to temporal attributes.* □

Definition 15. [*Itemset*] *An* itemset $(i_1, i_2, ...i_n)$ *is a non-empty set of items, where* $n \geq 1$, *and* $\forall\ i_k, k = 1..n$, *the* ts, tf *values are the same. The starting time and ending time value of an itemset* g *is denoted* $v_{ts}(g)$ *and* $v_{tf}(g)$, *respectively.* □

Remark 4. *In the moving object setting, since each moving object can be in only one place at each moment, all itemsets belonging to the same OID contain exactly one item.* □

Definition 16. [*Strict Time Interval Overlap*] *Let* $I_1 = [ts_1,\ tf_1]$ *and* $I_2 = [ts_2,\ tf_2]$, *be two time intervals. We say that there is a* Strict Time Interval Overlap *between them, if* $I_1 \neq I_2$ *and neither* tf_1 *precedes* ts_2 *nor* tf_2 *precedes* ts_1. □

Definition 17. [*Table of Items*] *Given a finite set of items* $\mathcal{I}$, *the schema of a* Table Of Items *(ToI) for* $\mathcal{I}$ *is the pair* $T = (OID, Items)$. *An instance of* T *is a finite set of tuples of the form* $\langle O_j, i_k \rangle$ *where* $i_k \in \mathcal{I}$ *is an item associated with the object* O_j. *Given* $\langle O_j, i_k \rangle$ *and* $\langle O_j, i_m \rangle$, *two tuples corresponding to the same object, then, either both time intervals of* i_k *and* i_m *are the same, or there is no strict time interval overlap between them.* □

Example 17. *Table 6.3 shows an instance of a ToI corresponding to the category instances of Table 6.2. Note that the first two items for* OID $= O_2$ *have the same ID because they correspond to the same category occurrence:* $[(categoryName, zoo), (ID, Z), (geom, pol7), (price, cheap)]$. *For the attribute* geom, *we assume that* $pol7$ *stores the geometric extension of* Z. □

6.3 Sequential Expressions

We begin with a simple language, based on paths over constraints, that we use later to elaborate the concept of support of regular expressions. This language is the cornerstone of our theory. In short, we define a language that expresses paths of constraints (denoted sequential expressions), and define the support of these paths. First, we define the syntax of this language.

OID	Items
O_1	([(ts_date,04/08/2008), (ts_time,14:05), (tf_date,04/08/2008), (tf_time,14:33), (ID,R2),(categoryName,restaurant), (geom,pol4), (price,expensive), (typeOfFood,French)])
	([(ts_date,04/08/2008), (ts_time,15:10), (tf_date,04/08/2008), (tf_time,16:05), (ID,B), (categoryName,castle), (geom,pol6), (name,BelfortCastle)])
	([(ts_date,04/08/2008), (ts_time,17:30), (tf_date,04/08/2008), (tf_time,18:48), (ID,R3),(categoryName,restaurant), (geom,pol5), (price,cheap), (typeOfFood,Italian)])
	([(ts_date,08/08/2008), (ts_time,06:22), (tf_date,08/08/2008), (tf_time,07:05), (ID,R1), (categoryName,restaurant), (geom,pol3), (price,cheap), (typeOfFood,French)])
	([(ts_date,08/08/2008), (ts_time,10:00), (tf_date,08/08/2008), (tf_time,13:00), (ID,B), (categoryName,castle), (geom,pol6), (name,BelfortCastle)])
	([(ts_date,08/08/2008), (ts_time,17:10), (tf_date,08/08/2008), (tf_time,18:17), (ID,R1), (categoryName,restaurant), (geom,pol3), (price,cheap), (typeOfFood,French)])
O_2	([(ts_date,03/08/2008), (ts_time,11:00), (tf_date,03/08/2008), (tf_time,11:15), (ID,Z),(categoryName,zoo),(geom,pol7), (price,cheap)])
	([(ts_date,08/08/2008), (ts_time,18:30), (tf_date,08/08/2008), (tf_time,21:00), (ID,Z),(categoryName,zoo),(geom,pol7), (price,cheap)])
	([(ts_date,19/08/2008), (ts_time,09:00), (tf_date,19/08/2008), (tf_time,10:20), (ID,R1), (categoryName,restaurant), (geom,pol3), (price,cheap), (typeOfFood,French)])
	([(ts_date,19/08/2008), (ts_time,17:00), (tf_date,19/08/2008), (tf_time,18:12), (ID,R2), (categoryName,restaurant), (geom,pol4), (price,expensive), (typeOfFood,French)])

Table 6.3: An instance of the ToI

Definition 18. [*Terms*] *There exist no* term *other than the following ones: (1) Constants: a literal enclosed by simple quotes. For example, '3' for the integer three, '12/10/2007' for a date. (2) Attributes of two types: (a) non-temporal, i.e., attributes which are elements in the category schema (e.g., categoryName, ID, geom, price); (b) temporal attributes, i.e., attributes which identify temporal occurrences of an item. They are denoted ts_date, tf_date, ts_time and tf_time. (3) Variables: a literal that begins with the '@' symbol. For example, @x, @Y1, etc. (4) Functions of n arguments: An expression fn(attribute, 'ct1', 'ct2', ... , 'ct_{n-1}'), $n \geq 1$, is a function where the* first *parameter is an attribute and all the other ones are constants.* □

Definition 19. [*Formula*] *Let C, V, A and F be a set of constants, variables, attributes and functions, respectively. The expression term1 = term2 is a formula, where term1 $\in A \cup F$, term2 $\in C \cup V$, and '=' is the equality symbol. Moreover, if $\mathcal{F}_1$ and $\mathcal{F}_2$ are formulas, $\mathcal{F}_1 \wedge \mathcal{F}_2$ is also a formula.* □

Definition 20. [*Constraint and Formula of a Constraint*] *A* constraint *is a formula (See Definition 19) enclosed in squared brackets or the empty constraint, denoted as '[ ]'. If C is not an empty constraint we denote* $\mathcal{F}(\mathcal{C})$ *the* formula of *C.* □

Definitions from 18 through 20 are summarized in Table 6.4.

R1	CONSTRAINT ← [CONDITION]
R2	CONDITION ← λ
R2	CONDITION ← EQ
R2	CONDITION ← EQ ∧ CONDITION
R3	EQ ← attr = 'constant'
R3	EQ ← attr = @vble
R3	EQ ← functionName(attr, ...) = 'constant'
R3	EQ ← functionName(attr, ...) = @vble

Table 6.4: Grammar for constraints

Now, we can define what a sequential expression is.

Definition 21. [*Sequential Expression*] *A* sequential expression *(SE) of length n is an ordered list of n sub-expressions* $c_1.c_2.c_3...c_n$*, where each* c_i *is a constraint,* $\forall i, i = 1..n$ □

Example 18. *The sequential expression of length two* $[].[price = \text{'}cheap\text{'}]$ *is composed of two constraints. The first one is an empty constraint. The second one expresses the equality condition.* □

Constraints can include functions over attributes. In our running example we use functions over OLAP hierarchies. These functions have the form *rollup*(*attribute*, '*level_i*', '*dimension_k*'). The meaning is that in a dimension called *dimension_k*, where *attribute* is the bottom level, a member of this level rolls up to a member of the level '*level_i*'. That is, *dom*(*level_i*) is the range of the rollup function. We can take advantage of the fact that our model supports a family of functions defined above to integrate an OLAP and GIS environment in this setting. Other (application-dependant) functions can be defined ad-hoc. In Section 6.4, we give an example of the use of these rollup functions in RE-SPaM.

Now we can formalize the semantics of SE.

Definition 22. [*Satisfability of a Constraint*] *Given a constraint C and an Item I, we say that I* satisfies *C if one of the following conditions hold:*

- *if C is the empty constraint '[]'.*
- *if $\mathcal{F}(C)$ is an atom of the form attr = 'ct' where attr is an attribute in any of the I, 'ct' is a constant in dom(attr), and the instantiation of attr with its value in I, equals 'ct'.*
- *if $\mathcal{F}(C)$ is an atom of the form attr = @x where attr is an attribute in any of the I, @x is a variable in dom(attr).*
- *if $\mathcal{F}(C)$ is an atom of the form f_n(attr, 'ct1', 'ct2', ..., 'ct_{n-1}') = 'ct', where attr is an attribute in any of I, 'ct' is a constant in dom(attr), and the instantiation of attr in f_n with its value in I, makes the equality* true.
- *if $\mathcal{F}(C)$ is an atom of the form f_n(attr, 'ct1', 'ct2', ..., 'ct_{n-1}') = @x, where attr is an attribute in any of I, @x is a variable in dom(attr).*
- *if $\mathcal{F}(C)$ is a formula of the form $\mathcal{F}_1 \wedge \mathcal{F}_2$, and $\mathcal{F}_1$ and $\mathcal{F}_2$ are satisfied by E.*

$\square$

Example 19. *The sequential expression SE=[].[price = 'cheap'] includes two constraints. The first one is an empty constraint, satisfied by all the items in an instance of a ToI. The second one expresses the equality condition. In our running example it is satisfied by the items with category occurrence identifier Z, R1 or R3.* $\square$

Definition 23. [*Bounded Variables of a Constraint in an Item*] *Given an item I and a constraint $C = [a_1 \wedge a_2 \wedge ...a_n]$, with $n \geq 1$, let us denote V the set of all variables in a_i, $\forall i, i = 1..n$. If $@x \in V$ then the set BV(@x, C, I) is composed of values v where $\forall i = 1..n$:*

- *if a_i is an atom of the form attr = @x then v is the result of all possible instantiations of attr with its corresponding value in I.*
- *if a_i is an atom of the form f_n(attr, 'ct1', 'ct2', ..., 'ct$_{n-1}$') = @x, then v is the result of all possible instantiations of attr in f_n with its value in I.*

$\square$

Example 20. *Given the constraint C=[geom = @x $\wedge$ price = @w] and the item I= ([(ts_date, 03/08/2008), (ts_time, 11:00), (tf_date, 03/08/2008), (tf_time, 11:15), (ID, Z), ($geom, pol7$), ($price, cheap$)]). Then, BV(@x, C, I)={pol7} , and BV(@w, C, I)={cheap}.* $\square$

Example 21. *Given the constraint C=[geom = @x $\wedge$ price = @x] and the item I= ([(ts_date, 03/08/2008), (ts_time, 11:00), (tf_date, 03/08/2008), (tf_time, 11:15), (ID, Z), ($geom, pol7$), ($price, cheap$)]). We have two instantiations of the same variable B(@x, C, I)=pol7 and B(@x, C, I)=cheap. Thus, BV(@x, C, I)={pol7, cheap}.*

$\square$

Definition 24. [*Contiguous List*] *Given a ToI instance with tuples of the form $\langle O_j, i_k \rangle$, let us denote $Itemset(O_j)$ the set of itemsets associated with O_j. We say that $CL(O_j)$= $\langle g_1, g_2, ..., g_n \rangle$ is a contiguous list for O_j if the following conditions hold:*

- $\forall h \in Itemset(O_j)$ *and* $h \neq g_i \ \ \forall i, \ i = 1..n$, $v_{ts}(h) < v_{ts}(g_1)$ *or* $v_{ts}(g_n) < v_{ts}(h)$
- $\forall i, i = 1..n \ \ \forall j, j = 1..n \ v_{ts}(g_i) < v_{ts}(g_j)$
- $\forall i, i = 1..n \ g_i \in Itemset(O_j)$

$\square$

Definition 25. [*Satisfability of a Sequential Expression*] *Given a ToI instance and a contiguous list $CL(O_j)$=$\langle g_1, g_2, ..., g_n \rangle$ with n≥1, a sequential expression SE=$c_1.c_2....c_n$ is satisfied by $CL(O_j)$ if the following conditions hold:*

- *an item in g_j satisfies c_j (Definition 22), $\forall\ j, j = 1..n$.*

- *$\forall p, p = 1..n$; $\forall q, q = 1..n$; if @x appears in c_p and c_q, $c_p \neq c_q$, then exists an item i_{g_p} in g_p, an item i_{g_q} in g_q such that $BV(@x,\ c_p,\ i_{g_p}) = BV(@x,\ c_q,\ i_{g_q})$ and $BV(@x,\ c_p,\ i_{g_p})$ and $BV(@x,\ c_q,\ i_{g_q})$ are singletons*[1].

Note that a contiguous list $CL(O_j)$ satisfies SE of length n if there are n items with different starting time (each one belonging to the different n itemsets). Each of these items are composed, by definition, by a temporal part and a category occurrence. Removing the temporal part, we obtain a list composed of only the parts corresponding to the category occurrence of these n items. We denote this list LCat(O_j, SE). □

Definition 26. [*Matching of a Sequential Expression*] *Given a sequential expression SE and a ToI instance with tuples of the form $\langle O_j, i_k\rangle$, we say that an object O_j* matches *SE, if there exists at least one $CL(O_j)$ that satisfies SE.* □

Example 22. *For the SE of Example 19 and the ToI of Table 6.3, object O_1 matches SE, using the second and third items in Table 6.3, call them g_1 and g_2, respectively. Also object O_2 matches SE, using the seventh and eight items in the mentioned table, call them g_3 and g_4, respectively. We can see that both objects O_1 and O_2 match SE using different items, illustrating the idea of* semantically equivalent *trajectories previously introduced.* □

Now we are ready to give a precise definition of the notion of *Support* of a sequential expression.

Definition 27. [*Support of a Sequential Expression*] *Given a sequential expression SE and a ToI instance with tuples of the form $\langle O_j, i_k\rangle$. The support of SE is the fraction of the different objects O_j in the ToI, that match SE. We denote the support of SE as Supp(SE) and the set of objects that match SE as ObjMatches(SE).* □

[1]This condition implies that all occurrences of the same variable must have the same value when instantiated.

Given a set of category occurrences M, a sequential expression SE, a ToI instance $\mathcal{T}$, and a parameter called minsup $\in [0, 1] \subseteq \mathbb{R}$, the *frequent patterns* in $\mathcal{T}$ restricted by SE, are defined as the *sequences of category occurrences* in LCat(O_j, SE), $\forall O_j \in$ ObjMatches(SE), such that Supp(SE) $\geq$ms.

Example 23. *In the example 22 the support of the SE is 100%. Moreover, the sequences identified by IDs* $\{B \quad R3\}$ *and* $\{Z \quad Z\}$ *are discovered whatever the minimum support be.* □

Example 24. *The support of SE=[name='BelfortCastle'].[typeOfFood='French'* $\wedge$ *price='cheap'] is 50%, and the sequence identified by IDs* $\{B \quad R1\}$ *is discovered.* □

As a final example, we illustrate definitions 15 through 27 through an example outside the MOD domain. To be more general, we choose an scenario where itemsets are of size > 1. Example 25 below is adapted from the classical data mining literature [65], and shows that our approach is general enough to capture other domains.

Example 25. *Consider the taxonomy shown in Figure 6.2, and a corresponding ToI instance in Table 6.5. The taxonomy can be seen as two rollup functions where, for example, rollup(F)='Tolkien', and rollup(I)='Clarke*[2]*. There are two different objects in the ToI, namely* $C1$ *and* $C2$*. We show below that the sequential expression SE=[ID='F'].[rollup(ID)='Tolkien'] has a support of 100% using the concepts explained above. We also have a contiguous list* $CL(C1)=\langle g_2, g_3\rangle$ *where itemset* g_2 *is composed of the second item of Table 6.5, and* g_3 *is composed of third and fourth items.*

We first analyze if $CL(C1)$ *satisfies SE. First,* g_2 *is composed of one item:* $g_2 = ([(ts\_date, 10/10/99), \ (ts\_time, 00{:}02), \ (tf\_date, 10/10/99), (tf\_time, 00{:}02), (categoryName, Book), \ (ID, F)])$ *and the constraint* $[ID = \text{'}F'\text{'}]$ *is satisfied by the only item composing* g_2*, since the instantiation of ID with the value in the item equals 'F'.*

[2]We use a simplified notation for the rollup function, since we do not deal here with dimensions and dimension levels, but simple taxonomies.

Second, g_3 is composed of two items: g_3= ([(ts_date, 10/10/99), (ts_time, 00:15), (tf_date, 10/10/99), (tf_time, 00:15), (categoryName, Book), (ID, J)], [(ts_date, 10/10/99), (ts_time, 00:15), (tf_date, 10/10/99), (tf_time, 00:15), (categoryName, Book), (ID, H)]) and the constraint [$rollup(ID)$ = 'Tolkien'] is satisfied by the second item of g_3, since the instantiation of ID in the function rollup with its value in this item (i.e., rollup(H)) equals 'Tolkien'.

Thus, $CL(C1)$ satisfies SE. We can conclude that the object $C1$ *matches SE and* contributes to the support *of the sequential expression SE. With a similar analysis, we can show that $C2$ also contributes to the support of SE. Then, we conclude that the support of the sequential expression SE is 100%.* □

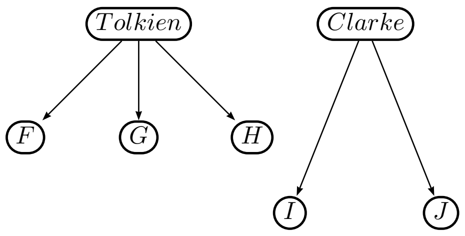


Figure 6.2: A taxonomy for Example 25

6.4 The RE-SPaM Language

We now introduce a language, denoted RE-SPaM, based on regular expressions where instead of atomic items (as in previous proposals), the atoms are constraints expressed as formulas over attributes of the complex items defined in Section 6.2. We formalize the language in the remainder of this section.

Intuitively a sequential expression lets us restrict the number of sequences we can obtain from a ToI. Moreover, if we consider an alphabet over constraints, a sequential expression is a simple regular expression which supports concatenation over constraints. If we have different sequential expressions, each one accepting a regular language we can build a deterministic finite automaton (DFA) that accepts the union of those languages. However, this raises the question of how to define the

ID	Items
C1	[(ts_date,10/10/99), (ts_time,00:01), (tf_date,10/10/99), (tf_time,00:01), (categoryName,Book), (ID,**I**)]
C1	[(ts_date,10/10/99), (ts_time,00:02), (tf_date,10/10/99), (tf_time,00:02), (categoryName,Book), (ID,**F**)]
C1	[(ts_date,10/10/99), ((ts_time,00:15), (tf_date,10/10/99), (tf_time,00:15), (categoryName,Book), (ID,**J**)] [(ts_date,10/10/99), ((ts_time,00:15), (tf_date,10/10/99), (tf_time,00:15), (categoryName,Book), (ID,**H**)]
C2	[(ts_date,10/10/99), ((ts_time,00:01), (tf_date,10/10/99), (tf_time,00:01), (categoryName,Book), (ID,**F**)] [(ts_date,10/10/99), ((ts_time,00:01), (tf_date,10/10/99), (tf_time,00:01), (categoryName,Book), (ID,**I**)]
C2	[(ts_date,10/10/99), ((ts_time,00:20), (tf_date,10/10/99), (tf_time,00:20), (categoryName,Book), (ID,**G**)]
C2	[(ts_date,10/10/99), ((ts_time,00:50), (tf_date,10/10/99), (tf_time,00:50), (categoryName,Book), ((ID,**J**]

Table 6.5: Instance of a ToI containing two objects (C1, C2)

support of the regular expression RE accepted by this automaton. We discuss the issue in this section.

6.4.1 Syntax and Semantics

Definition 28. [*RE over constraints*] *A* regular expression *over the constraints of Definition 20, is an expression generated by the grammar*

$$E \longleftarrow C \;/\; E|E \;/\; E? \;/\; E^* \;/\; E^+ \;/\; E.E \;/\; E \;/\; \epsilon$$

where C is a constraint, and ϵ represents the empty expression. The meaning of each operator is shown in Table 6.6. The precedence is the usual one. □

Remark 5. *Given a regular expression $\mathcal{R}$ we can always build the Deterministic Finite Automation (DFA) $\mathcal{A}_{\mathcal{R}}$ that accepts $\mathcal{R}$. The set of words accepted by $\mathcal{A}_{\mathcal{R}}$ is composed of constraints (recall that our alphabet is composed of constraints, i.e., the words accepted by $\mathcal{A}_{\mathcal{R}}$ are* sequential expressions*).* □

Symbol	Meaning
|	Disjunction. For example, C | D expresses 'C' or 'D'.
.	Concatenation. For example, C.D expresses that constraint 'D' immediately follows constraint 'C'.
*	Zero or more occurrences. For example, C^* expresses that constraint "C" holds zero or more times.
+	One or more occurrences. For example, C^+ expresses that constraint "C" holds one or more times.
?	Zero or one occurrence. For example, C? expresses that 'C' is optional.

Table 6.6: Regular Expression operators

Definition 29. [*Matching of a RE*] *Consider a regular expression $\mathcal{R}$ generated by the grammar of Table 6.6, the DFA $\mathcal{A}_{\mathcal{R}}$ that accepts $\mathcal{R}$, and $W(\mathcal{A}_{\mathcal{R}})$ the set of words accepted by $\mathcal{A}_{\mathcal{R}}$. There is also a ToI instance with tuples of the form $\langle O_j, i_k \rangle$. We say that O_j* matches *RE, if there exists at least one contiguous list $CL(O_j)$ that* matches *a word $w \in W(\mathcal{A}_{\mathcal{R}})$. We denote LCat($O_j$, RE) the ordered list obtained removing the temporal part from the items in $CL(O_j)$ that match the RE (i.e., the list containing only the category occurrences of the items in such contiguous list).* □

Now we can express the support of RE.

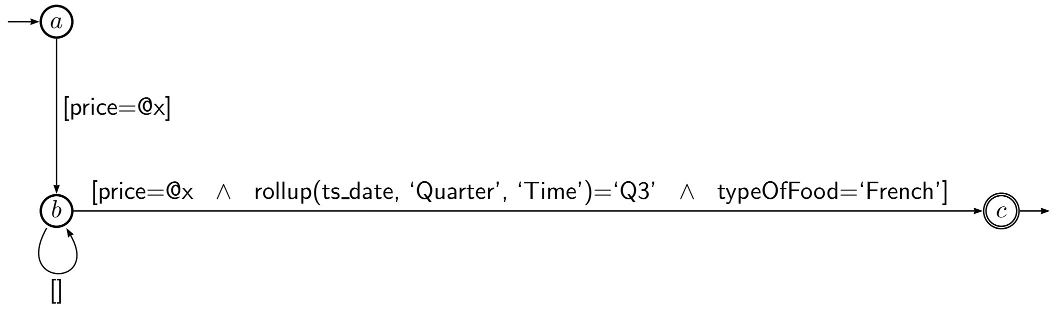
Definition 30. [*Support of a RE*] *Given a regular expression $\mathcal{R}$ and a ToI instance with tuples of the form $\langle O_j, i_k \rangle$, the support of RE is the fraction of the different objects O_j in the ToI, associated with a contiguous list $CL(O_j)$ that matches $\mathcal{R}$. We denote Supp(RE) the support of RE, and ObjMatches(RE) the set of objects that math RE.* □

Given a set of category occurrences M, a sequential expression SE, a ToI instance $\mathcal{T}$, and minsup $\in$ [0,1] $\subseteq \mathbb{R}$, the *frequent patterns* in $\mathcal{T}$ restricted by RE are the *sequences of category occurrences* in LCat(O_j, RE), $\forall O_j \in$ ObjMatches(RE), such that Supp(RE) $\geq$ms.

Example 26. *Consider the RE-SPaM expression:*

$$\mathcal{R} = [price = @x].[] * .$$
$$[price = @x \ \wedge \ rollup(ts\_date, \text{‘Quarter’}, \text{‘Time’}) = \text{‘Q3’} \wedge \ typeOfFood = \text{‘French’}]$$

Figure 6.3 shows the DFA that accepts the language generated by $\mathcal{R}$.



a
[price=@x]
b
[price=@x ∧ rollup(ts_date, ‘Quarter’, ‘Time’)=‘Q3’ ∧ typeOfFood=‘French’]
c
[]

Figure 6.3: Automaton for Q0

We want to check if the trajectory of object O_1 in Table 6.3 matches $\mathcal{R}$. Consider the contiguous list $CL(O_1)$ composed of the third and fourth itemsets, denoted as g_1 and g_2, respectively:

g_1= *([(ts_date,04/08/2008), (ts_time,17:30), (tf_date,04/08/2008), (tf_time,18:48), (ID,R3), (categoryName,restaurant),(geom,pol5),(price,cheap),(typeOfFood,Italian)])*

g_2= *([(ts_date,08/08/2008), (ts_time,06:22), (tf_date,08/08/2008), (tf_time,07:05), (ID,R1), (categoryName,restaurant),(geom,pol3),(price,cheap),(typeOfFood,French)])*

and the SE of length 2 accepted by $\mathcal{A}_{\mathcal{R}}$:

$$SE = [price = @x].$$
$$[price = @x \ \wedge \ rollup(ts\_date, quarter, Time) = \text{‘Q3’} \wedge \ typeOfFood = \text{‘French’}]$$

For simplicity we call c_1 the first constraint and c_2 the second one. The first (and only) item in g_1 (call it i_1) satisfies c_1, since c_1 contains the attribute price and a variable. Moreover, $BV(@x,\ c_1,\ i_1)$=$\{cheap\}$ since the value of the price attribute in *this item is cheap. Analogously, the first (and only) item in g_2 (call it i_2), satisfies c_2, since c_2 contains a conjunction composed of: (a) an equality condition between an attribute and a variable; (b) an equality between a function and a constant such as the instantiation of the attribute ts_date in the rollup function with its value in i_2 yields 'Q3'; (c) an equality condition between an attribute and a constant, and the instantiation of the attribute typeOfFood with its value in i_2 yields 'French', and coincides with the constant in c_2. Also, $BV(@x,\ c_2,\ i_2)=\{cheap\}$. We can see that $BV(@x,\ c_1,\ i_1)= BV(@x,\ c_2,\ i_2)$ and both BVs are singletons. We have found a contiguous list CL(O_1) that satisfies SE, and we can conclude that* CL(O_1) matches SE. *Summarizing, a contiguous list CL(O_1) composed by the items identified by R3 and R1 was found. Analogously, object O_2* matches SE *by means of a contiguous list of length 2 composed of its second and third itemsets. In this case, a sequence composed by the items identified by Z and R1 is found.*

Note that, even the contiguous list $\{R3\ R1\}$ and $\{Z\ R1\}$ are not exactly the same, they can be considered semantically equivalent with respect to the attribute price, *i.e., both of them are associated with a 'cheap' price.* □

6.4.2 RE-SPaM by Example

In this section through a set of queries, we give the reader the intuition of what RE-SPaM can express and how it differs from and substantially improves other proposals. For example, existing efforts force the user to enumerate the IDs of the items to express disjunctions like $(A|B|C|D)^*$. In practice, when the number of items is large, this solution would not be applicable. RE-SPaM allows writing concise expressions using the semantic information available. Expressions can be built with attributes, functions, constants and variables. We now present examples of the different kinds of expressions supported by RE-SPaM, in our running example.

Constraints without Variables.

Q1: "Trajectories of tourists who visit hotel H1, then optionally stop at restaurant

R3 and the Zoo, and either end at H1 or visiting the Belfort Castle".

[ID='H1'].([ID='R3'])*.([ID='Z'])*.([ID='B']|[ID='H1'])

Note that Q1 uses only ID attributes in all its subexpressions.

Q2: "Trajectories that visit hotel H1, then, optionally visit different places, and finish at the Belfort Castle Tower or going back to H1".

[ID='H1'].[]*.([ID='B']|[ID='H1'])

The use of the empty constraint allows avoiding the enumeration of all the items. If an expression includes an empty constraint, during the mining process it is instantiated with all the IDs of the category instances. Figure 6.4 shows the DFA that accepts the language generated by this expression.

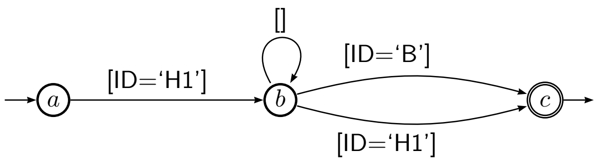


Figure 6.4: Automaton for Q2

Q3: "Trajectories of tourists who visit hotel H1 and then a cheap place or a place serving French food".

[ID='H1'].([price='cheap']|[typeOfFood='French'])

Q3 contains a subexpression with no ID. The disjunction is evaluated as follows. Places with cheap prices are R1, R3 and Z, and places that serve French food are R1 and R2. During sequential pattern mining we compute the items which satisfy these conditions, without the need of explicitly enumerating all the possibilities (note that the expression is equivalent to

[ID='H1']([ID='R1'] | [ID='R3'] | [ID='Z'] | [ID='R2']).

Q4: "Trajectories that visited hotel H1 and then some cheap place, on 10/10/2007".

[ID='H1'].([ts_date='10/10/2007' $\wedge$ price='cheap'])

Q4 contains a subexpression with a *temporal attribute*, which characterizes the occurrences of items in the database of sequences (i.e., the ToI).

Q5: "Trajectories that visit a cheap place during the third quarter of any year".

[rollup(ts_date, 'quarter', 'Time')='Q3' $\wedge$ price='cheap']

Q5 contains a *rollup function* over time. Like in the previous query, during mining, by accessing the ToI we compute the items that satisfy the temporal constraint.

Constraints with Variables.

Q6: "Trajectories that start at a place characterized by price (i.e., *price* is an attribute of the item representing this kind of place), then stop either at the zoo or the Belfort Castle, and end up going to a place that serves French food, and has the same price range as the initial stop".

[price=@x].([ID='Z'] | [ID='B']).[typeOfFood='French' ∧ price=@x]

Figure 6.5 shows the DFA that accepts the language generated by the expression. In our running example, 'cheap' and 'expensive' are the only possible values for prices; thus, the only valid combinations are: cheap-cheap and expensive-expensive. Sequences in objects of ToI such as {H1 Z R1} and {Z B R2} do not satisfy the query. The first one because hotel H1 is not characterized by price, the second one because Z has cheap prices but R2 is an expensive restaurant. The sequence {Z Z R3} satisfies the expression. In our implementation, variables are bound to items during the mining process as we explain later.

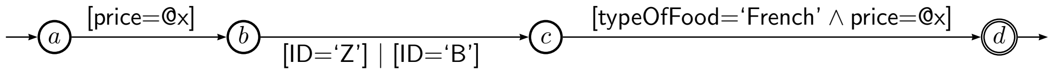


Figure 6.5: Automaton for Q6

Q7: "Trajectories that stopped at two places (the second one having cheap prices), at the same part of the day (e.g., both of them during the morning) on October 10th, 2008".

[rollup(ts_time, 'range', 'Time')=@z ∧ ts_date='10/10/2008'].

[rollup(ts_time, 'range', 'Time')=@z ∧ ts_date='10/10/2008' ∧ price='cheap']

Here, Q7 uses variables that the system binds (during the mining process), to the result of applying a function over temporal attributes.

Metadata Constraints.

Although, in general, variables are used to express matching conditions between different subexpressions, they can also be used to constraint items according to their structure. We call these kinds of expressions, metadata constraints.

Q8: "Trajectories that visited a place characterized by price".

[price=@Z]

The semantics here is: a sequence is in the result if it contains an item with an attribute *price* in it. As a more involved example, the constraint [price=@x]$^+$ is verified by sequences of one or more item (not necessary the same ones), all of them with the same price. In our running example, Z, R1, R2 and R3 are the items that satisfy this constraint. Let us analyze another example.

Q9: "Trajectories that visited a place characterized by price, and finished on October 10th, 2006".

[price=@x].[ts_date='10/10/2006']

In our running example, only items Z, R1, R2 and R3 satisfy the first constraint.

6.5 RE-SPaM Evaluation

6.5.1 Preliminary Considerations

We explained in Section 6.2 that the ToI is composed of itemsets such that items are associated with an object identifier OID. In the MO setting, at every time instant each OID can only be in exactly only one place, meaning that *itemsets are of length one*. However, the algorithm we present here can be applied to itemsets of any length. We work with the category instances depicted in Table 6.2. Temporal information associated with item occurrences is stored in the ToI (Table 6.3), which in our implementation is decomposed (i.e., normalized) as follows: we have a table with schema (OID, ts_date, ts_time, tf_date, tf_time, ID), where ID is the identifier of the corresponding category instance. All other attributes of the ToI are stored in a different structure, and retrieved via the ID. Moreover, we implemented two alternative approaches: (a) information about category instances is stored in an XML file;

(b) information about category instances is stored in a relational table, accessing the data through a join operation. Table 6.7 shows the normalized ToI instance for our running example.

OID	Items
O_1	([(ts_date,04/08/2008), (ts_time,14:05), (tf_date,04/08/2008), (tf_time,14:33), (ID,R2)])
	([(ts_date,04/08/2008), (ts_time,15:10), (tf_date,04/08/2008), (tf_time,16:05), (ID,B)])
	([(ts_date,04/08/2008), (ts_time,17:30), (tf_date,04/08/2008), (tf_time,18:48), (ID,R3)])
	([(ts_date,08/08/2008), (ts_time,06:22), (tf_date,08/08/2008), (tf_time,07:05), (ID,R1)])
	([(ts_date,08/08/2008), (ts_time,10:00), (tf_date,08/08/2008), (tf_time,13:00), (ID,B)])
	([(ts_date,08/08/2008), (ts_time,17:10), (tf_date,08/08/2008), (tf_time,18:17), (ID,R1)])
O_2	([(ts_date,03/08/2008), (ts_time,11:00), (tf_date,03/08/2008), (tf_time,11:15), (ID,Z)])
	([(ts_date,08/08/2008), (ts_time,18:30), (tf_date,08/08/2008), (tf_time,21:00), (ID,Z)])
	([(ts_date,19/08/2008), (ts_time,09:00), (tf_date,19/08/2008), (tf_time,10:20), (ID,R1)])
	([(ts_date,19/08/2008), (ts_time,17:00), (tf_date,19/08/2008), (tf_time,18:12), (ID,R2)])

Table 6.7: Normalized ToI instance

6.6 The Data Mining Process

Typically, in GSP-based algorithms, frequent sequences with an user-specified minimum support are computed in incremental phases. At each intermediate step k, the following occurs: (1) A temporary set C_k is built using the previous set C_{k-1}. Its elements are candidate sequences of length k. (2) Each element in C_k which contains at least one sub-sequence with support less than the minimum is discarded due to anti-monotony property (C_{k-1} is analyzed). (3) The database is accessed in order to analyze support, and each element in C_k with at least minimum support is added to the set F of frequent sequences. (4) When an empty C_k set is obtained, F contains the frequent sequences with minimum support.

We already explained that in RE-SPaM, a constraint is expressed as a regular expression $\mathcal{R}$; to evaluate if a sequence satisfies it, we build a DFA, denoted $\mathcal{A}_\mathcal{R}$ which accepts the language generated by $\mathcal{R}$. For example, Figure 6.5 shows the DFA that accepts the language generated by the regular expression in Q6 in Section 6.4.2 (we use this automaton later). The idea of using $\mathcal{A}_\mathcal{R}$ for pruning C_k before querying the database was first proposed in the SPIRIT algorithm. There, instead of using the original constraint C, a relaxed constraint C' (not necessarily anti-monotonic) is used during the mining process. When C' is not anti-monotonic, the second phase above is replaced with a strategy consisting in pruning the sequences in C_k which contain at least one subsequence which *satisfies* C' and does not have minimum support. In the last phase, F is analyzed to obtain the frequent sequences that *satisfy* C, i.e., a strict C verification is carried out. In what follows, we use a relaxed constraint C' that accepts the sequences (denoted *legal*) which correspond to a path in $\mathcal{A}_{\mathcal{R}}$. Informally, if the automaton accepts only two words "abbd" and "acabd", then the sequence "cab" is not pruned because it is a substring of the second one, but the sequence "cbd" is pruned because it is not a substring of any of these words.

We identify three phases when building C_k: (i) *C_k population*: C_k is populated using the information previously obtained. (ii) *C_k pruning by $\mathcal{A}_{\mathcal{R}}$*: C_k is pruned using the automaton and perhaps some extra information. If a candidate sequence does not satisfy the relaxed constraint C' it is discarded at this moment. (iii) *C_k pruning by the ToI instance*: C_k is pruned using the ToI instance, as we explain later, and added to a set F of frequent candidate sequences. Finally, F is pruned using the original constraint C.

6.6.1 The RE-SPaM Algorithm

Using $\mathcal{A}_{\mathcal{R}}$ During the candidate generation step, sequences that do not satisfy the constraints are pruned, given that they do not contribute to the result. We discuss here how RE-SPaM uses the automaton to prune candidate sequences while generating intermediate C_k's, *without accessing the ToI.* In step *k*, once C_k has been populated with candidate sequences of length k, we use $\mathcal{A}_{\mathcal{R}}$ for pruning ($\mathcal{A}_{\mathcal{R}}$ is stored in main memory). Using $\mathcal{A}_{\mathcal{R}}$ we check which candidate sequences satisfy the relaxed constraint C'. In this step, the algorithm finds out whether or not the conditions in the edges of paths of length k in $\mathcal{A}_{\mathcal{R}}$ are satisfied by candidate sequences in C_k. Note that the edges of the automaton are labeled *with constraints.* This is a relevant difference with existing approaches (where edges are labeled with IDs).

Let p=$\{e_1, e_2, ..., e_k\}$ be a path of length k in $\mathcal{A}_{\mathcal{R}}$, and let cs = {ID='ID$_1$', ID='ID$_2$', ..., ID='ID$_k$'} be a candidate sequence in C$_k$. We use $\mathcal{A}_{\mathcal{R}}$ to determine whether or not each one of the items identified by $\mathrm{ID_j}$, j $\in$ 1..k, satisfies the constraint that labels the edge e_j. Thus, we need to find the values of the attributes that characterize the item identified by $\mathrm{ID_j}$. Note that our intention is to prune C_k *without* accessing the ToI; thus, the only sources of information we can use are the automaton and the *category instances* (this table is also likely to be stored in main memory), and the only kinds of attributes that can be analyzed at this point are the ones in categories. The analysis of *temporal attributes* is postponed to a later stage. In summary, during this step we only use the automaton and category instances to *verify* the sub-conditions that do not involve temporal attributes, and postpone the evaluation of the conditions over temporal attributes.

Example 27. *In query Q6, the sub-conditions to evaluate are four: price=@x, ID='Z', ID='B' and typeOfFood='French'. As none of them involves temporal attributes, all of them can be analyzed using $\mathcal{A}_{\mathcal{R}}$ and category instances.*

However, in the case of query Q7, the sub-conditions are three: rollup(ts_time, 'range', 'time dimension')=@z, ts_date='10/10/2008' and price='cheap'. The only sub-condition that can be analyzed using $\mathcal{A}_{\mathcal{R}}$ and category instances is price='cheap'.

□

Handling Variables We have said that, in general, variables are used to match different constraints within the same expression. However, in the intermediate phases, when building set C_k, only sub-paths of length k are considered. If the same variable is used in both extremes of a path of length k, sets C_j with j<k are not useful for checking if these variables coincide.

Another question that arises is: which is the best strategy when a variable appears only once in an expression? We study two possibilities regarding the moment when the *verification phase* for non-temporal attributes can take place: *early evaluation* and *late* or *postponed evaluation.* In the former, the system determines if a sub-condition with no temporal attributes belonging to an edge of the automaton is verified by an item when building C_k, and before querying the ToI instance. In the latter, the verification occurs when the algorithm enters its final phase, i.e., when it must prune the set F using the original (not relaxed) constraint C. Thus, verification is postponed until the final phase and constraints are not checked while building intermediate C_k's. Obviously, late or early evaluation only affects performance, not the the final result. Deciding which strategy is better can be somehow tricky, as the following example shows. In the remainder, except when noted, we assume the following: *early evaluation* for conditions that involve non-temporal attributes and *constants*, and *late evaluation* for conditions that involve non-temporal attributes and *variables.*

Example 28. *Expression Q6 looks for a matching in the prices at the beginning and end of a path. The set C_1 is composed of sequences of* one item *that satisfy the constraints* $[price = @x]$, $[ID = \text{'}Z\text{'}]$, $[ID = \text{'}B\text{'}]$ *and* $[typeOfFood = \text{'French'} \wedge price = @x]$. *Also, C_2 is composed of sequences of* two items *that satisfy the constraints* $[price = @x].[ID = \text{'}Z\text{'}]$, $[price = @x].[ID = \text{'}B\text{'}]$, $[ID = \text{'}Z\text{'}].[typeOfFood = \text{'French'} \wedge price = @x]$ *and* $[ID = \text{'}B\text{'}].[typeOfFood = \text{'French'} \wedge price = @x]$. *Therefore, the binding of the variable @x is not used in these two phases (we are interested in matching both extremes). Only during C_3 this is relevant, because both bindings can be compared, and perhaps some candidate sequences could be pruned. For example, the sequence {Z Z R2} does not satisfy this constraint because Z is cheap and R2 is expensive.* □

Using the ToI We now discuss the use of the ToI for pruning candidate sequences with support less than the minimum. Assume we have computed C_k, which now contains the candidate sequences that satisfy the subexpression of length k. We continue with the analysis of Q6, and we now want to find out the sequences that satisfy the constraint, with a support of 100%. At a first glance, in Table 6.7 there is one sequence generated from OID= O_1, which satisfies the constraint: {R1, B, R1}. With a similar analysis, there exists only one sequence generated from OID= O_2, which satisfies Q6: {Z, Z, R1}. Since we are interested in categorical mining, not just in counting strict occurrences of items, we have to modify the way of counting support. Although none of the two transactions in Table 6.7 contains the same sequence, both satisfy Q6. Here, Z and R1 are semantically equivalent with respect to Q6 because both have cheap price associated with them. This is the reason why the algorithm cannot discard a candidate sequence (although it has support less than the minimum), if it is supported by at least one transaction (see Remark 6 below). Figure 6.6 depicts the three steps for computing C_1. Note that $\mathcal{A}_{\mathcal{R}}$ prunes H1 and H2 in step2, because they *do not match* any of the edges of the automaton.

Remark 6. *If we had followed the SPIRIT strategy, C_1 would have only contained R1 and R2. Items Z, B, and R3, would have been pruned, because their support is less than the minimum (each of them are in only one transactions). Moreover, no sequential pattern with this support would have been found, given that the regular expression requires that the trajectory stops at Z or B.* □

IDs
H1
H2
R1
R2
R3
B
Z

IDs
R1
R2
R3
B
Z

IDs
R1
R2
R3
B
Z

Figure 6.6: Computing C1: Step 1 (left), Step 2 (Legal) (center), Step 3 (right)

In the third step of the computing of C_k, the ToI is scanned for pruning the candidate sequences not present in any transaction. After this, C_k is added to the temporary set F. Figures 6.7 and 6.8 show how C_2 and C_3 are computed. Given that C_4 is empty, the algorithm enters its final phase, i.e., the *strict verification* of the sequences in F.

The *final phase* uses all sequences in the temporary set F and proceeds as follows. First, it uses the automaton to prune all sequences which are *not accepted.* Notice that here we are using the automaton for *acceptance verification* and not for *legal* verification. Also note that using the automaton to find sequential patterns with minimum support does not suffice, i.e., the ToI *must be scanned.* This scan has different goals: for verification of the conditions that have been postponed (for example, expressions which involve variables or temporal constraints), and for calculation of minimum support. Until this phase we only know that the sequences in F are present in some transaction. Now, we have to check which sequences in the set F have enough

IDs	
R1	R1
R1	R2
R1	R3
R1	B
R1	Z
R2	R1
...	...
R3	R1
R3	R2
...	...
B	R1
...	...
B	Z
Z	R1
...	...
Z	Z

IDs	
R1	B
R1	Z
R2	B
R2	Z
R3	B
R3	Z
B	R1
B	R2
Z	R1
Z	R2
Z	B
Z	Z

IDs	
R1	B
R2	B
B	R1
Z	R1
Z	Z

Figure 6.7: Computing C2: Step 1 (left), Step 2 (Legal) (center), Step 3 (right)

IDs		
R1	B	R1
R2	B	R1
B	R1	B
Z	R1	B
Z	Z	R1
Z	Z	Z

IDs		
R1	B	R1
R2	B	R1
Z	Z	R1

IDs		
R1	B	R1
Z	Z	R1

Figure 6.8: Computing C3: Step 1 (left), Step 2 (Legal) (center), Step 3 (right)

support. Recall that we consider all sequences in F equivalent with respect to the regular expression under analysis. In our example, OID=O_1 supports {R1 B R1} and OID=O_2 supports {Z Z R1}. Thus, all of these sequences verify the original expression, yielding a support of 100%. Figure 6.9 shows the set F before and after automaton verification, and set F after the ToI is scanned. As our expression does not involve temporal attributes, the ToI scan does not change the sequences in the set F. However, this scan is necessary to compute the support.

Algorithm Details Algorithm 1 sketches the procedure for mining sequential patterns using the approach described above.

Once the user defines the regular expression RE and the minimum support, the DFA automaton is built. The relaxation constraint C' is also defined. The algorithm proceeds in incremental phases until the final condition holds, which depends on the relaxation choice. Garofalakis *et al* [20, 21] proposed four variations of the algorithm, namely *Naïve*, *Legal*, *Valid WRT*, *Regular*. For example, in the *Legal* algorithm (which we have used in the discussion above), the final state is reached when no legal

Algorithm 1 RE-SPaM Algorithm

```
01. minSupport := ReadSupport()
02. query := ReadQuery()
03. $\mathcal{A}_{\mathcal{R}}$ := BuildAutomaton(query)
04. C' := define C relaxation
05. //Incremental phases
06. k := 1
07. REPEAT
08.         // Add sequences of length k which verify C' to set $F$
09.         // Populate $C_k$
10.         $C_k$ := { $c_k$ | $c_k$ is a candidate sequence of length k }
11.         // Update $C_k$ using $\mathcal{A}_{\mathcal{R}}$ and category instances
12.         $tmp$ := { $c_k$ | $c_k$ is not verified by any path of length k in $\mathcal{A}_{\mathcal{R}}$ }
13.
14.         $C_k$ := $C_k$ - $tmp$
15.         // Update $C_k$ using ToI instances
16.         $tmp$ := { $c_k$ | $c_k$ is not verified by any sequence of any OID }
17.
18.         $C_k$ := $C_k$ - $tmp$
19.         // Update $F$
20.         $F := F \cup C_k$
21. UNTIL FinalConditionHolds or MinSupport($F$) < minSupport
22. // Final phase
23. // Eliminate from $F$ sequences that do not satisfy constraint C
24. $F$ := VerifyOriginalConstraint($F$, minSupport)
25. ListSequences($F$)
```

sequences of length k with respect the start state of the automaton can be generated.

The main loop is the core of the algorithm. Line 10 corresponds to the generation of candidate sequences. For example, in the *Legal* and *Valid WRT* algorithms this

IDs		
R1		
R2		
R3		
B		
Z		
R1	B	
R2	B	
B	R1	
Z	R1	
Z	Z	
R1	B	R1
Z	Z	R1

IDs		
R1	B	R1
Z	Z	R1

IDs		
R1	B	R1
Z	Z	R1

Figure 6.9: Set F: Initial (left); Accepted by automaton (center); After the ToI scan (right)

step corresponds to the generation of C_k using C_{k-1}. However, adopting the *Valid* variation would require using F *and* the automaton. Steps 12-14 analyze each of the candidate sequences $c_i \in C_k$. The idea consists in detecting *all the paths* in the automaton, which in fact represent sequences of conditions satisfied by c_i. To do this we need the information stored in the automaton and the category instances. If c_i does not verify any path of length k, it is pruned. Steps 16-18 scan the ToI instance to compute support. If we are using *early evaluation* for the temporal attributes, here is where the ToI is used to validate temporal conditions. For each OID the algorithm calculates the $c_i \in C_k$ supported by consecutive sequences. Given that we are introducing semantic information into the mining process and we consider *equivalent sequences* to be interchangeable, we add all the sequences c_i supported by at least one OID to the set F, and prune the ones not supported by any OID. While the support of the set F is equal or greater than the minimum, the algorithm continues. In the final phase (Step 24-25), the algorithm repeats the sequence performed inside the loop, but using C instead of C'.

Either using early or late evaluation, we need to bind a variable to a value. In RE-SPaM, there is no limit on the number of variables that can be defined. Thus, we use a hashing structure to store and check if a variable has already been bound. This structure is built for each one of the candidate sequences, i.e., the bindings cannot be shared between sequences. For each item in a candidate sequence we analyze the variables involved. For each variable that appears, it may happen that: (a) it is the first time that this variable is instantiated; thus, the variable and the value are hashed in a structure of variable bindings; (b) the variable has been already bound to a value; thus, the new binding is compared with the previous one. For an efficient support count, we also use hashing structures to store sequences and the OIDs which support them.

Example 29. *For query [price = @x]+, suppose we obtain a candidate sequence* $\{R1 \quad R2\}$*. Due to the item identified by R1, @x is stored in a hash table with its corresponding value ('cheap'). Later, analyzing the price associated with the item R2, we obtain the value 'expensive', which does not match the previous one. Thus, the candidate sequence does not verify the expression.* □

6.6.2 Complexity

We now analyze the complexity of Algorithm 1. Each step of the algorithm is composed of three phases. Each phase generates candidate sequences of the same length (i.e., at each step k, candidate sequences of length k are computed in three phases). The elements in C_k (line 10, first phase) are computed using the sequences in the set C_{k-1}. We compute C_k by means of a self-join between the sequences in C_{k-1}, and discard the ones such that their suffixes and prefixes of length $k-2$ do not match. For example, when joining the sequences 'ABC' and 'BCD', we generate 'ABCD' (the suffix and prefix 'BC' is the same for both sequences). In the worst case, this operation has complexity $\mathcal{O}(|C_{k-1}|^2)$, where $|C_{k-1}|$ is the number of candidate sequences in C_{k-1}.

The second phase (see lines 12, 13 and 14) consists in using C_k and pruning it with the automaton generated from the constraints used in the query (i.e., the automaton is used to verify if a candidate sequence of IDs in C_k satisfies some path of length k). If $|\mathcal{A}_\mathcal{R}|$ is the number of states of the automaton, then, in the worst case the algorithm performs $|\mathcal{A}_\mathcal{R}|^{(k+1)}$ comparisons to detect if a k-sequence of IDs in C_k satisfies an expression in a path of length k in the automaton. Since $|\mathcal{A}_\mathcal{R}|$ is the number of nodes of the graph representing the automaton, the number of paths of length k could be at most $|\mathcal{A}_\mathcal{R}|^{(k+1)}$, because the automaton accepts loops. Checking which sequences in set C_k (generated in the previous phase) satisfy the constraints in the automaton, takes, in the worst case $\mathcal{O}(|C_k| * |\mathcal{A}_{\mathcal{R}}|^{(k+1)})$. All these sequences of candidate C_k are organized in a hash table, for the following phase.

The last phase of step k consists in pruning using the ToI, thus, accessing the database (lines 16, 17 and 18). A database scan must be performed in order to build contiguous lists (see Definition 24) of length k. Let us denote $|Items|$ the number of items in the ToI. In the worst case, all these items belong to the same object and the number of consecutive lists (each one of these lists is composed of IDs.) of length k that are generated is given by $|Items| - k + 1$. This is an upper bound. If these items were distributed among different objects (transactions) the number of lists would become considerably smaller. The candidate sequences in C_k are organized in a hash table; thus, checking if a list of IDs belonging to the ToI matches a list of IDs in C_k could be done in $\mathcal{O}(1)$. Depending on the query, this phase requires more than a simple matching between IDs in the list and IDs in some element in C_k. Recall that we do not evaluate only *coincidence* here. For example, if a sequence in C_k is 'ABA' and there is a list 'ABA' in the database, we may initially think that this sequence satisfies the query. However, let us consider the $[ts = @x \wedge price = \text{'cheap'}].[tf = @x \wedge food = \text{'Italian'}]$. Although 'AB' is a candidate sequence in C2 (because A verifies the price='cheap' and B verifies food='Italian'), we must postpone the evaluation of ts and tf to the database scan stage. We cannot do this with the automaton because *temporal* attributes are not part of category occurrences. Thus, the database scan is not only for evaluating support. Then, in the worst case, this phase can be done in $\mathcal{O}(|Items| - k + 1)$. These three phases are preformed repeatedly until no more candidate sequences are generated or the candidate sequences in C_k are all pruned accessing the ToI, because there does not exist any useful list in the database of length k.

The last part of the algorithm (line 24) is analogous to the former ones, except for the first phase (generation of C_k). Phases two and three use all the candidate sequences not pruned in previous steps, to detect if they are recognized by the automaton and have the required support.

6.7 Comparison of our Proposal with Related Work

Gianotti *et al.* [22] introduced *t-patterns* for mining sequential patterns on regions of interest. A t-pattern is of the form Railway Station $\xrightarrow{1h10min}$ Castle Square $\xrightarrow{2h15min}$ Museum'. These patterns are basically defined by extension. On the contrary, regular expressions allow us to define more complex patterns in an intensional fashion.

Mouza and Rigaux [16] propose a query language based on regular expressions. Our language differs considerably from theirs because they work with IDs or names. Moreover, they do not relate trajectories with *the GIS environment and/or non-spatial data*. On the contrary, the query language we propose uses functions, which bind MO information with attributes of the PoIs and other external information. In this chapter our examples were restricted to rollup functions over temporal attributes, but other possibilities will be discussed in Chapter 7, showing how we support even more powerful expressions. The regular expressions proposed by Mouza and Rigaux talks about zones represented by some label (a constant) or a variable (@x). In this language, each occurrence of a variable in the pattern is instantiated with the same value. The units of time spent by the moving object inside some zone are expressed with the symbol + (undetermined time) or via the temporal constraint boundaries {min, max}. The query "objects that started in zone A, visited another zone and five minutes later came back to A", is expressed in this language as: 'A,7.@X$^+$A,12'. Variables *can only be associated with places* (represented by labels or IDs) visited by objects. Thus, the language cannot deal with *time constraints* or categories. On the contrary, our approach allows variables associated with any attribute of an item. For example, we can express "trajectories that visit two places with the same price" with the query [price=@x].[price=@x] where the variable @x is instantiated with prices.

Giannotti *et al.* also introduced the concept of Region of Interest (RoI). Although with similar goals, our work clearly differs from [22] in several ways. First, we work with *stops* and *moves* instead of pre-defined regions of interest. This allows to identify which of the RoIs are really relevant to a trajectory. We also use these *stops* and *moves* to "encode" or compress a trajectory, which, in many practical situations is enough to identify interesting sequences very efficiently. Second, in [22] the authors focus on computing the RoIs dynamically from the trajectories. On the contrary, in our approach the user defines the places of interest of an application in advance, and from them the *stops* and *moves* are computed prior to performing trajectory mining. Finally, our work approach, allows integration between trajectories and background geographic data, an issue mentioned albeit not addressed in [22].

Brakatsoulas *et al.* [9] enrich trajectories with the relationship between trajectories and between a trajectory and the GIS environment (via topological functions like *intersect*, *meets*), but their approach requires all information on moving objects to be processed. In our proposal, on the contrary, we use semantic information to reduce, whenever possible, the amount of data to be considered.

We remark that all of these proposals cannot detect if trajectories are *semantically equivalent*, they only work with the classical notion of similarity.

Now, regarding the sequential pattern mining algorithm we presented in this chapter, we would like to point out several issues. In the work of *Srikant et al.* [65] only items (IDs) can participate in hierarchies. We extended this idea allowing any kind of attribute (including temporal ones), and showed that supporting attribute, variables, and categorization yields a much powerful language for sequential pattern mining.

In SPIRIT [20, 21], regular expressions are used to restrict the sequences produced by the mining algorithm. However, SPIRIT deals only with the IDs of items. We explained above that using attributes lets us avoid item enumeration. But, besides *writing concise queries*, our proposal supports the binding of variables to any attribute (or function over an attribute) during the mining process. As far as we are concerned, this is the first proposal in this sense, in sequential pattern discovery.

The original idea of counting for sequences support has been maintained, although regular expressions have been introduced to constrain the number of sequences to be obtained. As a simple example, the expression $(A|B).C$ is satisfied by sequences like A.C or B.C. Even though the semantics of this RE suggests that both of them are equally interesting to the user, if neither of them verifies a minimum support (although altogether they do), they would not be retrieved. Suppose that a database contains two transactions, the first one with two itemsets $<(A)(C)>$ and the second one with two itemsets $<(B)(C)>$. In SPIRIT, no sequences expressed by the RE $(A|B).C$ with minimum support 75% would be retrieved. This is because during the mining process the itemsets of length 1 are composed of the items A, B and C, although only C is maintained as the other ones are considered separately and they do not have minimum support (i.e., A has 50% support and B has 50% support). We count support in a different way because we consider both, A.C and B.C *equivalent* under the RE. Thus, our algorithm returns these two sequences as the ones that satisfy the RE and have minimum support.

6.8 Summary

In this chapter we proposed an Apriori-like mining algorithm. Although it is general enough to be applied to different settings, we focused our discussion on the MO setting. To reduce the number of sequences obtained by the mining process we defined a language denoted RE-SPaM. Then, we studied RE-SPaM syntaxis and semantics, and provided a comprehensive set of examples. This powerful language differs form other proposals because it support attributes, variables, categorization and functions. In particular the introduction of functions allow to relate attributes (temporal or not temporal) of the *stop* in a trajectory with the environment in which the trajectory evolves. We also provided an in-depth discussion of different implementation issues.

Chapter 7

Adding Geography to RE-SPaM

In this Chapter we extend RE-SPaM in order to bind trajectories to the geometric environment where they evolve. This is implemented by allowing a geometric condition to be included in RE-SPaM constraints. We call the resulting language RE-SPaM^{++}. This feature allows to fulfill our original goal, namely integrating OLAP, GIS, and moving object data.

7.1 Introduction

In Chapter 6 we presented RE-SPaM, a language that can express sequential patterns by means of regular expressions over constraints defined in terms of the attributes of the items to be analyzed, where an item is a tuple composed of an object identifier, a time interval, and a PoI. These expressions can be used during the sequential pattern mining process to prune sequences that, although satisfying minimum support requirements are not of interest to the user. RE-SPaM supports functions that are the key to bind *stops* with information not stored in the PoIs (like, for instance, OLAP data). We have already shown in Chapter 3 of this thesis that GIS and OLAP integration can be achieved by means of the Piet data model and the Piet-QL query language. We now integrate Piet-QL into RE-SPaM and show how this allows to achieve GIS, OLAP and Trajectory integration. We call the resulting language RE-SPaM^{++}.

7.1.1 RE-SPaM^{++}Motivation

RE-SPaM expressions can use three constructs: variables, attributes (spatial and non-spatial) and functions. Attributes let us talk about the characteristics of the *stops*. Functions allow to compare the value of some attribute of the *stop* with external information. When binding trajectories to GIS, it appears intuitive to take advantage of functions. For example, we may want to express within a constraint if the geometry of the *stop* is contained by some region. Other OpenGIS operators could be used. The next example illustrates this.

Example 30. *Using the running example introduced in the previous chapter we can express "Trajectories that start at a place characterized by price (i.e., a place such that* price *is an attribute of the item representing this kind of place), then stop either at the zoo or the Belfort Castle, and end up at a place that serves French food, has the same price range than the initial stop, and is close the Dijle river". We assume that here 'near' implies that it is at 0.05 of distance units in the coordinate reference system. The geometry of Dijle river is expressed in Postgis by the literal 'MULTILINESTRING ((4.714209 50.931933, 4.69394 50.974907))'. We could write this query in RE-SPaM as follows:*

$$[price = @x].([ID = `Z'] | [ID = `B']).[typeOfFood = `French' \wedge price = @x \wedge near(geom, `MULTILINESTRING((4.714209 \quad 50.931933, 4.69394 \quad 50.974907))', 0.05) = `true']$$

The function near *follows the RE-SPaM syntax, i.e., its first argument is an attribute (the geometry of the PoI) and the other ones are literals. We show an implementation (in Java-like pseudo-code) in Listing 7.1, where the function getValue returns the number contained in the literal and st_distance is the OpenGIS operator.*

Listing 7.1: Function Implementation

```
String near(Geometry geometry, String pl, String distance)
{
    if (st_distance(geometry, pl) < distance.getValue())
        return 'true';

    return 'false';
}
```

This method implies knowing the string expression of a geometry in advance (in this case the geometry of Dijle river). It would be more powerful if, instead of implementing all GIS operators, we take advantage of Piet-QL language introduced in Chapter 4. This integration appears to be straightforward. The approach we follow consists in adding a new kind of functions, with the ability capable to iterate over Piet-QL result sets, and make this result set available at the moment the RE expression is evaluated. We explain this in next section.

7.2 RE-SPaM^{++} Design

RE-SPaM^{++} is based on Piet-QL and RE-SPaM. Recall that the former lets us express complex queries by supporting four kinds of queries: GIS-OLAP, OLAP-GIS, pure GIS, and pure OLAP queries. In RE-SPaM^{++} we use only a subset of Piet-QL, namely the queries that return spatial objects (GIS-OLAP and Pure GIS queries). This allows to make the spatial component of a Piet-QL query (the one we can obtain in the SELECT clause) *available* to be used by a RE-SPaM function. In general, any query that exposes a spatial component (non-geographic, geographic, mixed) is useful for our purpose of integrating geographic features for trajectory pattern using mining RE-SPaM. We explain this integration next.

7.2.1 Functions in RE-SPaM^{++}

Piet-QL queries return a collection of over tuples (technically, a set of literals). Our idea is based on defining a cursor over this set, and make this cursor available when the RE expression is evaluated. To implement this, we just add a `WITH` statement to the Piet-QL `SELECT` clause. This statement defines an alias over the cursor. Since, functions in RE-SPaM cannot deal with cursors, we must incorporate new kinds of functions whose syntax consists in a first parameter which corresponds to an *attribute of a category occurrence* (for example, *geom*) or a *temporal attribute* (ts, tf, or their subparts). The second parameter is of the form `a.b`, where the semantics is that `b` is the name of an attribute, and `a` is the name of a cursor associated to the `WITH` clause. The function returns a literal. We give the flavor of the language by means of a first example query, using the running example that we followed in previous chapters.

Q1: "Trajectories that stop at a place which belongs to a region that contain a river, and whose next stop is a zoo or a castle, finishing there."

```
WITH TABLE regRiver(the_geom) AS
SELECT GIS DISTINCT(bel_regn.the_geom)
FROM bel_regn, bel_river
WHERE contains(bel_regn.the_geom, bel_river.the_geom);

[containedBy(geom, regRiver.the_geom)='true'].
([categoryName=Zoo']|[categoryName='Castle'])
```

The Piet-QL part returns a set of geometric objects (polylines) representing regions containing rivers, in the cursor `regRiver(the_geom)`. In the RE-SPaM part of the query, the first constraint checks if the PoI is contained in one of the regions in the set. In other words, when an item in the ToI is being evaluated, the corresponding PoI geometry (represented by the attribute `geom`) is compared against each one of the geometric elements in the cursor.

The syntax of RE-SPaM^{++} is shown in Table 7.1. In this table, *GisOlapPietQL-Query* and *RE-SPaMQuery* denote the syntax of Piet-QL queries of type GIS-GIS and GIS-OLAP and RE-SPaM queries defined in Chapters 4 and 6, respectively.

R1	RE-SPaM^{++}Query $\leftarrow$ listOfWiths RE-SPaMQuery
R2	listOfWiths $\leftarrow$ λ
R2	listOfWiths $\leftarrow$ aWith listOfWiths
R3	RE-SPaM^{++}aWith $\leftarrow$ WITH TABLE tableName(listOfAttrs) AS GisOlapPietQLQuery ;
R4	listOfAttrs $\leftarrow$ attr
R4	listOfAttrs $\leftarrow$ attr, listOfAttrs

Table 7.1: Grammar for RE-SPaM^{++}

7.3 RE-SPaM^{++}By Example

We now give some examples to illustrate the use of RE-SPaM^{++} queries.

Q2: "Trajectories that stop at a place with cheap prices, which is very close to a district located in a region crossed by a river, and finish at the Belfort Castle".

```
WITH TABLE district(the_geom) AS
SELECT GIS DISTINCT(bel_dist.the_geom)
FROM bel_dist, bel_regn, bel_river
WHERE intersects(bel_regn.the_geom, bel_river.the_geom) and
contains(bel_regn.the_geom, bel_dist.the_geom);

[price='cheap'  ∧  short_distance(geom, district.the_geom)='true'].
[name='Belfort Castle']
```

This example shows how the Piet-QL part of the query is used to link the trajectories of the MOs to the geographic space where they evolve. Here, the Piet-QL query returns districts (i.e., polygons) in a map. At evaluation time, each geometry of the PoI where a trajectory stops is compared with the geometry of each district in the cursor, to check if the PoI being visited is close to it.

Q3: "Trajectories that visit a place with cheap prices, and then stop at the zoo (finishing there), such that *both stops* are either located in regions crossed by a river (although not necessarily the same region), or in regions not crossed by rivers".

```
WITH TABLE regCrByRiver(the_geom) AS
SELECT GIS DISTINCT(bel_regn.the_geom)
FROM bel_regn, bel_river
WHERE intersects(bel_regn.the_geom, bel_river.the_geom);
```

[price='cheap' $\wedge$ *containedBy*(geom,regCrByRiver.the_geom)= @x].
[*containedBy*(geom, regCrByRiver.the_geom)= @x $\wedge$
categoryName='Zoo'].

The variable @x is of boolean type. At evaluation time, the variable is bound to 'true' or 'false', and the two constraints are evaluated with this value. In this example, the two constraints in the RE-SPaM^{++} expression include the spatial function *containedBy*.

Q4: We now discuss a query that does not refer to the categories defined in Table 6.2, but helps to illustrate other features of the language. We ask for "trajectories that visit a city in the district of Nijvel in 2007 with stores selling more than 10000 units, and finish in a city in a province crossed by Albertkanal river, with sales greater than 5000 units. In between, other cities can be visited".

```
WITH TABLE city1(name) AS
SELECT GIS bel_city.name
FROM bel_city
WHERE bel_city IN(
   SELECT CUBE   filter([Store].[Store District].
   [Nijvel].Children,[Measures].[Unit Sales]>10000)
   FROM [Sales]
   SLICE [Time].[2007]);
```

```
WITH TABLE city2(name) AS
SELECT GIS lc1.name
FROM bel_city AS lc1, bel_prov AS lp2, bel_river AS lr2
WHERE Contains(lp2, lc1) AND Intersects(lp2, lr2)
AND lr2.name='Albertkanal'
AND lc1 IN (
   SELECT CUBE filter([Store].[Store City].Members,
   [Measures].[Unit Sales]>5000)
   FROM [Sales] );
```

[*belong*(name,city1.name)='true'].[]*.[*belong*(name,city2.name)='true']

Cities here are represented by points in a map. The query requires two `WITH` clauses and a function *belong*, used for matching the city names (the names of the PoI and the spatial object returned). This example also shows that the functions defined ad-hoc are not necessarily geometric ones. The first parameter of the function is the name of the PoI, and the second one is a collection of names of the spatial objects in a layer containing the cities. As a consequence, the same layer is used for the PoIs and the geometric environment where trajectories occur, opposite to the previous examples where PoIs and the geometric environment where in different layers.

7.4 Implementation

We now analyze implementation details of the mining algorithm, and show the implications of adding spatial objects support to RE-SPaM. Strictly speaking, no changes are needed to be introduced to the mining algorithm presented in Chapter 6. However, since the RE-SPaM mining algorithm proceeds in incremental phases, and in each phase a portion of the candidate sequence is evaluated to decide if it satisfies some constraint labeling the edges of the automaton, we must guarantee that the new expressions introduced in RE-SPaM^{++} can be efficiently evaluated. To accomplish this objective we borrow ideas from dynamic programming techniques, where once a function is evaluated with some parameters *its result is stored in a cache* avoiding recalculation. In our setting, there are two types of caching that can be exploited, which we denote *macro* and *micro-cache*, respectively. The former stores the result of a function of type $fn(`value', tableName.attribute)$ (that is, functions returning sets). The latter stores the result of a function of the form $fn('value', 'literal', ...)$. We illustrate these ideas by means of an example.

Consider query Q1 from Section 7.2, and the category occurrences in Table 6.2. Let us recall that the query asks for trajectories that pass through a place which belongs to regions that contain a river and finish at a zoo or a castle. The Piet-QL part of the query returns regions in the cursor denoted `regRiver.the_geom`. To clarify the example, we must first give some geographic information about Belgium. There are three regions: Vlaams Gewest in the north, Brussel-Hoofdstad (the small region within the former one) and the Wallonne (in the southern part of the country). The three regions are shown in Figure 7.1. Brussel-Hoofdstad is *crossed by* the Kanaal van Charleroi river, but this river is not *contained* by the region. Vlaams Gewest *contains* the Ieperlee river and the Wallonne region *contains* the Ourthe Occle river. Thus, both regions are in the cursor after the Piet-QL query is evaluated. For clarity, in Figure 7.1 the names of the rivers were omitted, but we can see that the large northern and southern regions contain rivers, while the small region is crossed by a river.

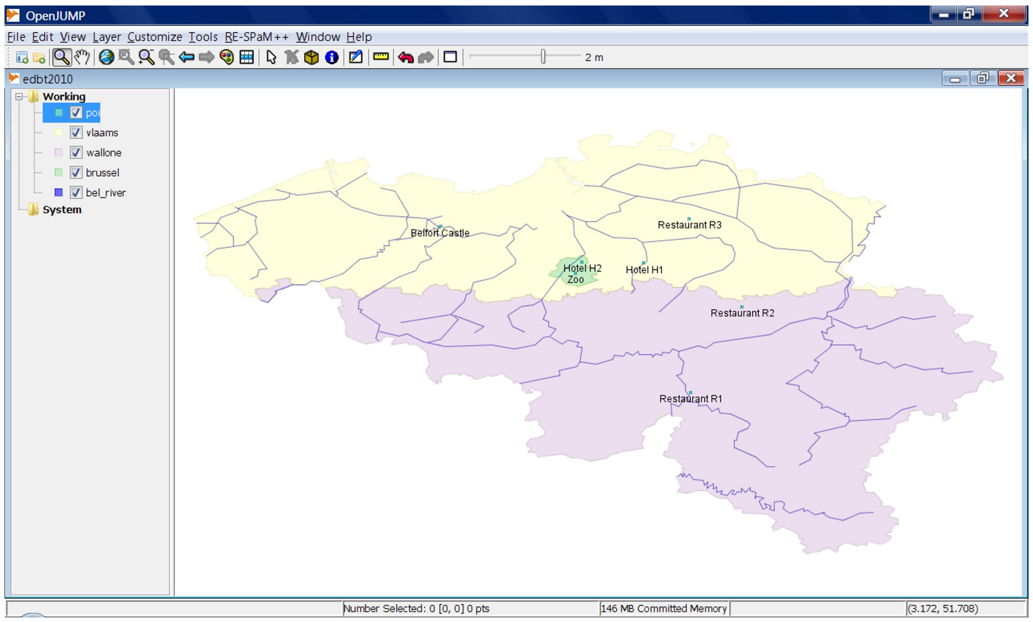


Figure 7.1: Regions, rivers and PoIs in Belgium map.

The algorithm starts by building C_1 with all the category occurrences. Then, it uses the automaton for pruning: if a candidate sequence does not satisfy any path in the automaton, it is pruned. We have three paths of length 1 in the automaton (see Figure 7.2).

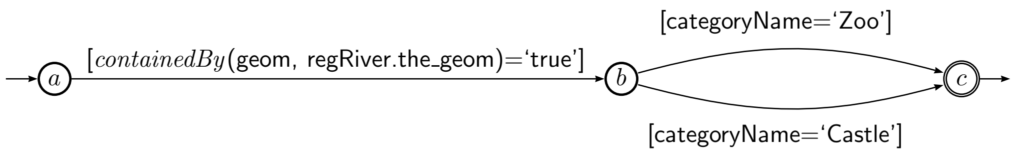


Figure 7.2: Automaton for Q1

The constraint `[categoryName = 'Zoo']` is satisfied by the the zoo occurrence of Table 6.2. The constraint `[categoryName='Castle']` is satisfied by the Belford Castle (denoted B). The constraint, [*containedBy*`(geom, regRiver.the_geom)='true']` must be evaluated for the category occurrences of hotels and restaurants, to check which PoIs are in regions crossed by rivers (returned by the Piet-QL part of RE-SPaM^{++}). Given that the constraint contains a function, the cache is used for this evaluation. The engine first looks up in the *macro-cache* (implemented as a hash table) for the value associated with the key *containedBy*('*pol*1',*regRiver.the_geom*). No value is retrieved in this case, and the function starts browsing the cursor. The first tuple in the cursor is $\langle$'*Vlaams Gewest*'$\rangle$. Instead of evaluating the function, the *micro-cache* is now queried for the value associated with the key *containedBy*('*pol*1', '*Vlaams Gewest*'). We can see in Figure 7.1 that '*pol*1' (the geometry of Hotel H1) is contained in the Vlaams Gewest region. Then, the association between *containedBy*('*pol*1', '*Vlaams Gewest*') and 'true' is stored in the *micro-cache*. Since the value returned by the function is 'true' there is no need to continue browsing the cursor. Moreover, *the macro-cache is also updated*, associating the key *containedBy*('*pol*1', *regRiver.the_geom*) with the value 'true' (see below how this is used in the next step). Next we evaluate *containedBy*('*pol*2', *regRiver.the_geom*). Again, nothing is retrieved from the *macro-cache*, therefore we must browse the cursor. The engine looks up in the *micro-cache* for a value associated with *containedBy*('*pol*2', '*Vlaams Gewest*'). Since nothing is retrieved, the function must be evaluated, returning the value 'false', and this association is stored in the *micro-cache*. Similarly, the *micro* *and macro-cache* are updated with the associations *containedBy*('*pol*2', '$Wallonne$') and 'false', and *containedBy*('*pol*2', *regRiver*.the_geom) and 'false', respectively. Finally, the candidate sequence H2 is discarded since it can not satisfy any path of length one in the automaton. Analogously, the system finds out whether or not 'pol3', 'pol4' and 'pol5' are contained in some tuple of the cursor regRiver.the_geom by taking advantage of the *macro and micro cache*. Figure 7.3 shows the initial and final states of C_1. In the final phase of the first step (the step with $k = 1$), C_1 is analyzed against the ToI to check for minimum support. For the second step, let us suppose that all candidate sequences of C_1 are maintained (i.e., all candidates will be checked for minimum support). Thus, the system populates the set C_2 in the second step by joining C_1 with itself. Similarly to the first step, using the automaton the engine determines which candidate sequences of length two are satisfied by some path of length two. Many candidate sequences are discarded here, for instance, those combinations which do not end at zoos or castles. Again, it becomes important to optimize the evaluation of functions. For this, we use now the *macro-cache*. When deciding if the candidate sequence of length 2 $\{H1 \quad Z\}$ is accepted by some path of length two in the automaton, the function *containedBy*('*pol*1', *regRiver.the_geom*) must be evaluated. But now, this key and its associated value *can be obtained from the macro-cache*, since its value was calculated in the previous step and the cache was updated accordingly. The process continues until a step k such that no sequences of length k exist in the ToI. In Figure 7.4 we show the state of C_2 before and after pruning.

IDs
B
Z
H1
H2
R1
R2
R3

IDs
B
Z
H1
R1
R2
R3

Figure 7.3: Computing C_1: before (left) and after (right) pruning with automaton

We now analyze the impact in the performance of RE-SPaM^{++} as a consequence of adding the new kinds of functions. Taking advantage of the cache strategy, Piet-QL queries are evaluated only once, i.e. during the first step of the mining process; the

IDs	
B	B
B	Z
B	H1
B	R1
B	R2
B	R3
...	...
Z	B
...	...
H1	B
H1	Z
...	...
R3	R3

IDs	
B	B
B	Z
H1	B
H1	Z
R1	B
R1	Z
R2	B
R2	Z
R3	B
R3	Z

Figure 7.4: Computing C_2 : before (left: 36 tuples), and after (right: 10 tuples) pruning with automaton

subsequent evaluations of the same function (with the same parameters) is reduced to a lookup in the *macro o micro-cache*. Thus, the overhead is introduced only at the beginning of the process.

7.5 Summary

In this chapter we studied the problem of finding trajectory sequential patterns taking into account the geographic environment in which the trajectories occur, and provided a solution. This solution builds over the work discussed in previous chapters of this thesis.

Chapter 8

Implementation and Case Study

In this chapter we present a complete case study based on a collection of maps of Belgium, a data warehouse, and a set of trajectories obtained through simulation. We discuss the data preparation for the spatial, OLAP, moving object data, and run different experiments, reporting and analyzing the results. Then, we show how RE-SPaM^{++} is used in a visualization tool to help to understand the patterns obtained.

8.1 Data Preparation

Our case study comprises the behavior pattern detection of MOs over the map of Belgium, in light of SOLAP. It is composed of OLAP data, GIS data and MO database as we describe in the following subsections.

8.1.1 OLAP Data

We analyze the fact sales of different stores distributed around Belgium country. Sales are stored in a data warehouse (no geometries are stored in such DW). The measures of the cube are *Unit Sales*, *Store Cost*, *Store Sales* and *Number of Products Sold*. The dimensions that have been identified are *Store*, *Customer*, *Product*, *Promotion Media*, and *Time*. We have mentioned in Section 3.1 that this DW does not correspond to a real-world situation.

- The *Store Dimension* is a non-geometric spatial one, according to Bédard classification. Its levels conform the following hierarchy: *Store Name* → *City* →

District → *State* → *Country.* Each level could have additional descriptive attributes. For instance, the level *Store Name* has attributes like Street Address and Store Manager.

- The *Customer Dimension* is also a non-geometric spatial dimension and its levels conform the following hierarchy: *Name* → *City* → *State* → *Country.* Its bottom level also has the following attributes: Gender, Marital Status, Education and Yearly Income.

- The *Product Dimension* is a non-spatial one, with hierarchy: *Product Name* → *Brand Name* → *Product Subcategory* → *Product Category* → *Product Department* → *Product Family.*

- The *Promotion Media Dimension* is non-spatial and it has only one level: *Media Type.*

- The *Time Dimension* contains the following level hierarchy: *Month* → *Quarter* → *Year* → *Time.*

We implemented the data cube using a ROLAP approach. The fact table contains 2,010,337 tuples. The store, customer, product, promotion media and time dimensions contain 60, 10,281, 1,560, 1,864 and 2,192 tuples, respectively. The total size of the data warehouse is 190MB.

8.1.2 GIS Data

We describe the layers in the Belgium map. The geographical information of Belgium is available and published over the Web by the GIS Center. There are layers for rivers, cities, districts, provinces and regions. Cities are represented by points, rivers as polylines and the others layers as polygons. According to Bédard taxonomy, we have a mixed spatial dimension, conformed by cities, districts, provinces and regions. Recall that a mixed dimension contains geometries and non-spatial information. For example, provinces have information in the GIS about population and number of agricultural employees, regions have the number of olives cultivated per hectare. The number of tuples in the tables corresponding to rivers, cities, districts, provinces and regions are 39, 583, 43, 9 and 3, respectively. In Section 4.2 (Figure 4.1), we explained how we match the geometric and non-geometric hierarchies.

8.1.3 Trajectory Data

As we have explained in Chapter 7, RE-SPaM^{++} works over semantic trajectories. Therefore, preparing the SM-MOFT table from the raw data requires some data pre-processing that we explain next.

Real vs. Synthetic Trajectories We first describe the data acquisition phases. We consider two possibilities: working with real-world trajectories or synthetic ones. Real-world trajectories are difficult to acquire due privacy issues. It is well-known that it is not enough to encode the identity of MOs to protect privacy. Depending on the number of trajectories in a pattern, the identity could be revealed by a pattern. Thus, most MO trajectories are not publicly available. There are some exceptions: some research projects have published real-world raw trajectories. The INFATI Project[1] is one of them. Its purpose was to investigate driver responses to alerts issued by a device installed in the car. In the case of Belgium there are not public data available. Thus, we decided to work with simulated data. We worked with the *Network-based Generator of Moving Objects*[2], which by using the information of a network in a proprietary graph format generates samples of raw trajectories. The simulator uses parameters such as maximum speed of MO, maximum capacity of connections, external objects that affect the movement of an object, influence of external objects and the interval time between samples. We simulated movement of different kinds of MO, like pedestrians, cars, bikes.

[1]http://timecenter.cs.aau.dk/software.htm
[2]http://www.fhoow.de/institute/iapg/personen/brinkhoff/generator

8.1.4 Preparing and Running the Simulator

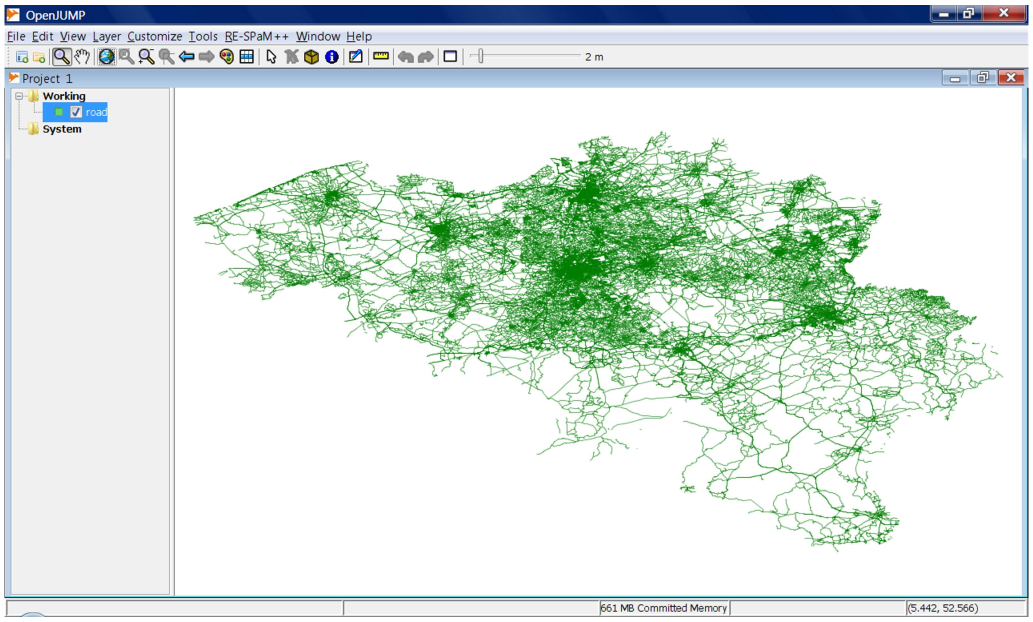
We downloaded a graph with the Belgium road network from the MapCruzin.com website[3] and transformed this format to the one expected by the Network-based Generator. The process generated a network with 756,498 nodes and 1,542,051 edges. We remark that the network is a digraph, i.e., some of these roads are double-way. Figure 8.1 depicts the network we used in our experiments, at two levels of detail.

The simulator can be controlled by several parameters. We used the road network, the maximum time duration, the number of initial MOs, the maximum number of MOs generated at each instant, the maximal range of speed of the MOs (e.g., low, medium, high), and name of the file that will store the synthetic data. Each row in the target file stores the following fields: (a) a string that indicates ‘newpoint’, ‘point’ and ‘disappearpoint’ for indicating that this row corresponds to a new, existing or disappearing point (MO); (b) a point ID (i.e., a MO identifier); (c) a sequence number for a specific point ID; (d) the type of the MO (for instance, pedestrian, cyclist, motor cyclist, car), (e) (x, y) coordinates, (f) an integer that represent an instant and (g) speed. The information is order by instant in ascendent order, i.e. the information of MOs is interleaved. For our experiments we needed to generate real timestamp information.

We ran the generator several times and processed all the simulated trajectories that belong to the same file as if they had been generated on the same day. In all these experiments we used a maximum simulation time duration between 400 and 600 iterations (we then transform these iterations in actual time intervals), which corresponds approximately to one day-long trajectories (we assume that MO reported positions each 30 seconds). The maximum speed for the MOs ranged from slow to fast objects.

We generated a total of 2204 raw trajectories. The minimum, average and maximum lengths of those trajectories (i.e., number of positions) were 1, 157, 1593, respectively. Figure 8.2 depicts a raw trajectory generated by the simulator. The trajectory starts at the bottom-center of the figure and follows the north-west direction. When

[3] MapCruzin is an independent firm specializing in innovative GIS projects, environmental and social and demographic research, website development and hosting. http://www.mapcruzin.com/free-belgium-arcgis-maps-shapefiles.htm



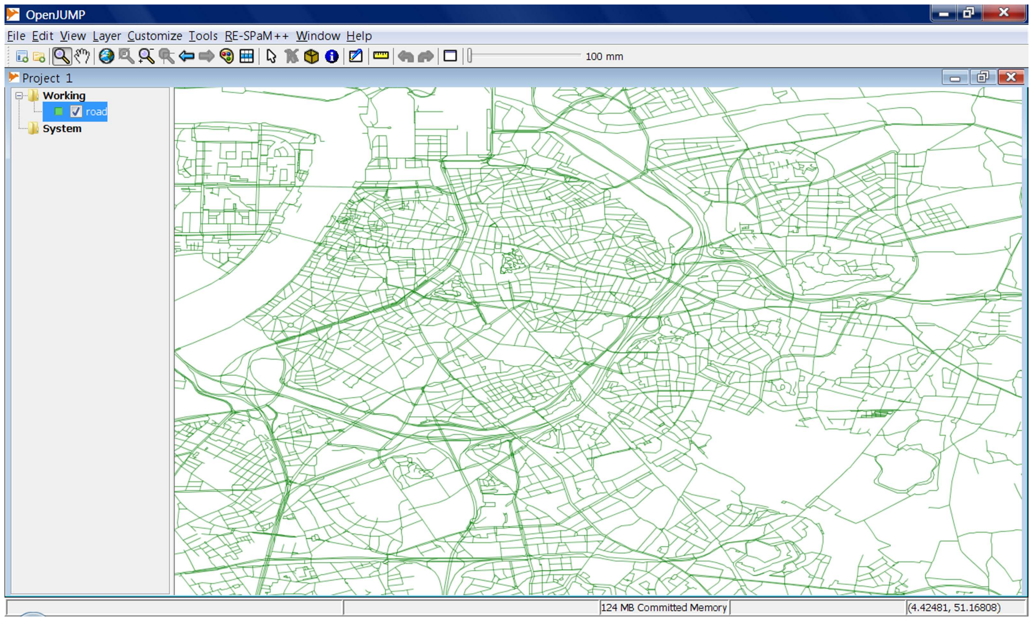


Figure 8.1: The Belgium road network at different levels of detail

it arrives at the top, it takes the same path (double-way) and follows the south-east direction (bottom-right). The same portion of the path is used twice, so double points can be visualized. All points belong to the road network.

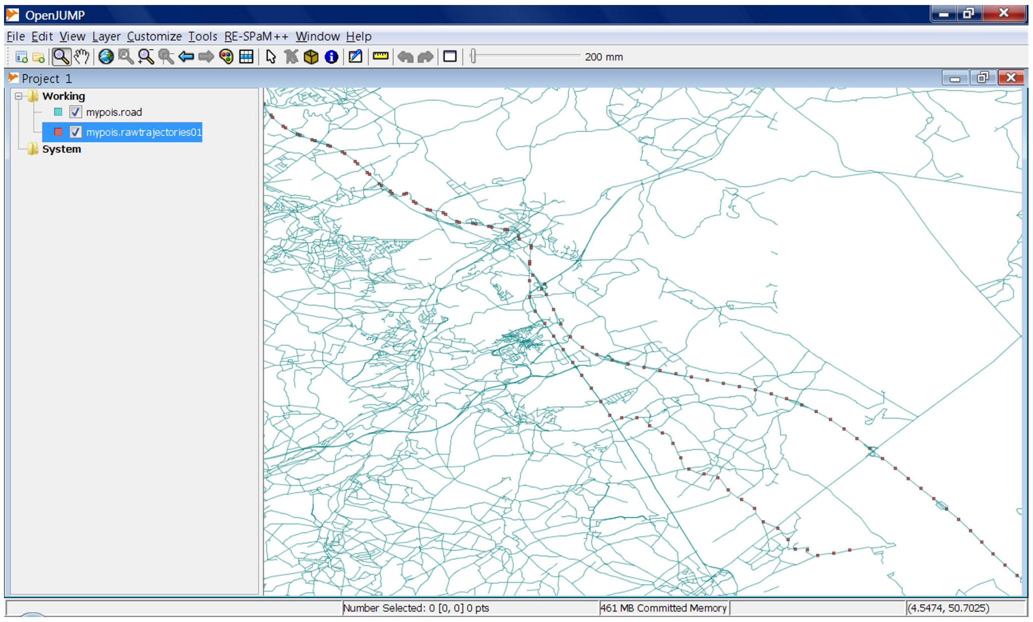


Figure 8.2: A Raw Trajectory

8.1.5 Preparing the Places of Interest

Information about PoIs in Belgium was obtained from the *PoI Plaza* Website[4]. We grouped these data in six different categories: stores (supermarkets and department stores), sport clubs (basketball, football, golf and tennis clubs), gastronomy (pubs and restaurants), banks, hotels, tourist attractions (abbeys and castles). In total, 1230 PoIs were downloaded. We summarize them in Table 8.1. The files were in Google Earth KML format[5] and we converted them to ESRI Shapefile format[6] files before importing them in PostgreSQL database.

The attributes related to each category instance were obtained, in general, from these files. We added other information not available: *price* (for gastronomy, sport clubs and tourist attractions), and *star*, indicating the hotel category. The values for these attributes were either inferred or generated randomly. For example, in the case of 'price' for sports clubs, we proceeded as follows: golf and tennis were considered expensive, and basketball and football, cheap. See the category schema in Table 8.2.

[4]http://poiplaza.com

[5]KML (standing for Keyhole Markup Language) is an XML-based language for expressing geographic annotation and visualization on three-dimensional Earth browsers

[6]This file format is a geospatial vector data format for GIS

Category	Category Instance (number of occurrences)
Bank	Fortis(109)
Gastronomy	Breweries(91), Pizza Hut(40), Pasta Vapiano(1)
Hotel	Different Names(373)
Sport Club	Basketball(102), Football(49), Golf(61), Tennis(40)
Store	Carrefour(85), Delhaize(111), Ikea(6)
TouristAttraction	Abbeys(40), Castles(122)

Table 8.1: POIs category instance of case study

Category	Schema
Bank	$[\mathbf{ID}, categoryName, geom, name]$
Gastronomy	$[\mathbf{ID}, categoryName, geom, speciality, name, town, street, phone, price]$
Hotel	$[\mathbf{ID}, categoryName, geom, name, phone, star]$
Sport Club	$[\mathbf{ID}, categoryName, geom, game, name, town, price]$
Store	$[\mathbf{ID}, categoryName, geom, name, town]$
TouristAttraction	$[\mathbf{ID}, categoryName, geom, subtype, name, town, phone, price]$

Table 8.2: Category Schema for the case study

Finally, we generated the SM-MOFT from the MOFT. The computation requires that the raw trajectories contain the reported spatiotemporal positions. However, the simulator generates continuous movement, but cannot simulate a stop in the trajectory. Thus, again, some additional data pre-processing was needed to simulate these stops. Table 8.3 shows the maximum and minimum stop durations for each category. Using these times, trajectory stops were randomly generated, and two datasets were produced, denoted the large and medium-sized ones. The former contains 1230 PoIs, from a MOFT of 252590 tuples that yielded an SM-MOFT of 4484 tuples. The latter contains 800 PoIs, with a compression yielding an SM-MOFT of 2851 tuples.

Category	Minimum Duration (min)	Maximum Duration (min)
Bank	10	60
Gastronomy	30	120
Hotel	30	480
Sport Club	30	180
Store	30	120
TouristAttraction	30	120

Table 8.3: Max and Min Duration for PoIs

8.2 Experiments

We ran our tests on an Intel Core 2 Duo CPU, at a clock speed of 2.6 GHz, equipped with 4GB RAM and running over Windows Vista Operating System. All the data (PoI and ToI) was stored in PostgreSQL 8.2.3 database. The application framework was developed as a java plug-in for OpenJump. Algorithms have been implemented in Java 1.6.

Many factors could affect execution time of the mining process using RE-SPaM^{++}: among others, we can mention the number of trajectories to be mined, the minimum support, the length of the query, the kind of a Piet-QL query, the number of intermediate steps, the number of PoIs, the attributes that these PoIs share. It is not trivial to isolate one of them and maintain the others fixed. We have designed different experiments to get an insight of the algorithm, and compare the theoretical complexity computation presented in Chapter 6 against real-world implementation results. We remark than in our RE-SPaM^{++} implementation the minimum support works as a threshold. We defined the support of a RE rather than a SE, i.e., if the RE has enough support all the sequences which satisfy this support will be retrieved, even if the sequences, individually, do not satisfy the support, implementing the concept of *semantic equivalence*. Thus, given a minimum support, we obtain a collection of sequences, or no sequence at all, making it irrelevant to vary the minimum support value to vary the number of sequences to be retrieved. That is the reason why during all of our experiments we chose the same (low) value for the minimum support, 0.01.

8.2.1 Goal 1

RE-SPaM^{++} uses an automaton for pruning candidate sequence before scanning the ToI. Our first goal is to confirm the impact of this pruning phase.

Experiment 1.1 We formulated several queries and computed, in each step, the number of candidate sequences generated in phases 0 and 1, i.e. before and after the use of the automaton. We used the following queries, which return different numbers of frequent sequences of different lengths that influence the results.

Q1: $[categoryName = \text{`}\texttt{gastronomy}\text{'}].[price = \text{`}\texttt{cheap}\text{'} \wedge game = \text{`}\texttt{basket}\text{'}]$
Q2: $[speciality = \text{`}\texttt{beer}\text{'}].[game = \text{`}\texttt{basket}\text{'}]$
Q3: $[categoryName = \text{`}\texttt{gastronomy}\text{'}].[game = \text{`}\texttt{basket}\text{'}]^{+}$
Q4: $[categoryName = \text{`}\texttt{gastronomy}\text{'}].[game = \text{`}\texttt{basket}\text{'}]^{+}.[star = \text{`}\texttt{3}\text{'}]$

Q1, Q2, Q3, Q4 produce 9, 9, 11, and 0 sequences, respectively. However, the performance of Q4 is approximately three times slower than Q1 and dos not produce sequences. This makes clear that the number of sequences obtained is not a good performance indicator. The results are shown in Table 8.4.

Query	Execution Time (sec)	Number of frequent sequences obtained
Q1	10.55	9
Q2	16.39	9
Q3	27.38	11
Q4	31.19	0

Table 8.4: Executing time for experiment 1.1

In fact, the performance is associated with the *number of intermediate steps* that the mining process uses to generate the frequent sequences, and with the number of candidate sequences that the algorithm manages throughout all the steps. We remark that the third phase of step k scans the ToI (i.e., requiring accessing the disk) for building contiguous lists and compares them with the candidate sequences in C_k. Thus, if the number of candidate sequences in C_k could be reduced, the total execution time would be lower. Table 8.5 shows for each query its intermediate steps (excluding the final phase), and the number of candidate sequences generated during Phase 0 (before using the automaton) and during Phase 1 (after the use of the automaton), i.e., the size of C_k in each phase. As a measure of the automaton effectiveness, we show in the last column the percentage of reduction of the size of each set C_k. The use of the automaton substantially reduces the number of candidate sequences in intermediate steps, which is relevant since this steps precede the scanning of the ToI.

Query	Automaton Effectiveness		
	Phase 0: $\|C_k\|$	Phase 1: $\|C_k\|$	Reduction (%)
Q1 (three intermediate steps)	1230	234	81 %
	13225	3306	75 %
	0	0	0 %
Q2 (three intermediate steps)	1230	193	84 %
	11881	2958	75 %
Q3 (five intermediate steps)	1230	234	81 %
	13225	6670	50 %
	44	44	0 %
	6	6	0 %
	1	1	0 %
Q4 (five intermediate steps)	1230	363	70 %
	20164	8236	59 %
	58	58	0 %
	6	6	0 %
	1	1	0 %

Table 8.5: Phases of Experiment 1.1

8.2.2 Goal 2

One of the novelties of our algorithm consists in the support of variables. Thus, our second goal is to estimate its impact.

Experiment 2.1 We analyzed the performance of the four algorithm variations mentioned in Chapter 6, namely legal-early, legal-late, valid-early and valid-late. We selected four queries, containing two variables that must be matched, and in each query the distance between these two variables ranges from 1 to 27 (the largest trajectory in the MO database contains 27 *stops*). Our ultimate goal is to measure the performance of late and early binding of variables. The query used are:

Q5: `[price=@x].[price=@x]`

Q6: `[price=@x].[subtype='castle'].[price=@x]`

Q7: `[price=@x].[subtype='castle'].[].[price=@x]`

Q8: `[price=@x].[subtype='castle'].[]+.[price=@x]`

The difference between *early* and *late* evaluation is the instant when the set of bounded variables BV is calculated (see Definition 23). *Legal* and *Valid* algorithms differ in the relaxed constraint they use for pruning using the automaton. In the *Legal* algorithm: (a) A sequence of length N is legal if some path of length N in the automaton satisfies it. (b) The candidate sequence generation, as we explained in Chapter 6, is obtained by a self join, i.e., to generate candidate sequences of length k, the join between previous candidate sequences C_{k-1} and C_{k-1} is performed. As for the *Valid* algorithm: (a) A sequence of length k is valid if some path of length k that finishes in a final state in the automation satisfies it. (b) The candidate sequence generation differs from the Legal algorithm. In step k it tries to extend the candidate sequence moving one step backwards, i.e., it does not use a join.

At first sight, the *Valid* algorithm could appear to be the best one. But this depends on several factors. Assume that we have N PoIs, and at the end of step k-1 the number of candidate sequences generated is P. Using the *Legal algorithm* we obtain, at Phase 0 of Step k, a set containing NxN candidate sequences. Instead, using the *Valid* algorithm, at Phase 0 of Step k we would obtain a set of PxN candidate sequences. Which is better depends on the sizes of P and N. If the number of PoIs largely exceeds the number of candidate sequences, i.e., P >> N, then the *Legal* algorithm would be the best, otherwise the *Valid* algorithm would do a better job. We used 800 PoIs (i.e., the medium-sized dataset) and the number of sequences pruned by the ToI in a previous step is an average of 8 times smaller (100 elements). Thus, in our experiments, all executions with the *Valid* were extremely bad with respect to the *Legal* variation. Further, we run the first three queries with the four algorithms, but the *Valid* algorithm for Q4 took 3 hours and 31 minutes. Thus, we discarded the use of *Valid algorithm* for the rest of the experiments. For details see Figure 8.3. We ran four algorithm combinations: Early and Late variable evaluation, using the Legal and Valid approaches.

We conclude that for the *Legal* approach, there was not a substantial difference between using early or late evaluation for queries that have a distance of variables less than 4. Binding variables in early stages prunes intermediate tuples, but more overhead is introduced. However, when the distance is larger, late evaluation performs better. In the case of *Valid algorithm* late evaluation outperforms early evaluation.

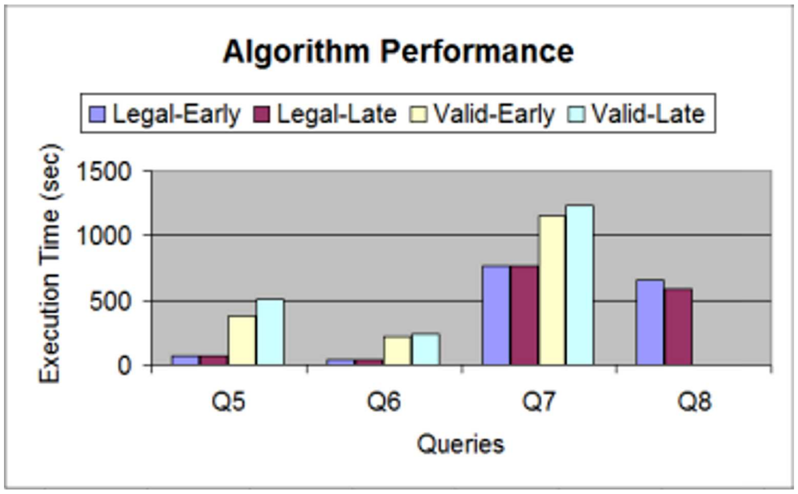


Figure 8.3: Experiment 2.1

Experiment 2.2 The number of different values that can take a variable could affect its performance (recall that an equality condition over attributes other than IDs is equivalent to a disjunction over the IDs). The queries we propose contain two variables that must be matched. The queries were:

Q9: `[categoryName=@x].[categoryName=@x]`

Q10: `[star=@x].[star=@x]`

Q11: `[price=@x].[price=@x]`

Q12: `[game=@x].[game=@x]`

Q13: `[subtype=@x].[subtype=@x]`

We used the algorithm with the *legal-late evaluation* alternative. The attributes we used were: categoryName, star, price, game and subtype. We wanted to analyze to what extent the number of different occurrences (i.e., possible values) for the attributes that appear in the ToI affect the performance. Thus, we grouped the queries according to these values as follows. First, using the medium-sized dataset (with 800 PoIs), the number of different occurrences for the attribute *categoryName* are six ('bank', 'gastronomy', 'hotel', 'sportclub', 'star', 'touristattraction'); for *price* and *star* there are three possible values the attribute can take (‘free’, ‘cheap’, and ‘expensive’ for the former, and ‘3’, ‘4’, ‘5’ for the latter); for *game* and *subtype*, there are two (‘basket’, ‘football’ and ‘abbey’, ‘castle’, respectively). The queries that use categoryName form a group (Q9) as the bounded value set BV for the constraint has 6 elements. Analogously, the queries that use price and star form a second group (Q10 and Q11) since BV has 3 elements. Finally those queries that use game and subtype form a third group (Q12, Q13) since BV has 2 elements.

The results are shown in Table 8.6.

Size of BV	6 (Q9)	3 (Q10,Q11)	2 (Q12,Q13)
Time (sec)	539.18	40.58	12.45

Table 8.6: Performance of Experiment 2.2 (DB medium)

Running the same experiment with the original dataset (1230 PoIs), the number of occurrences that appear in the items of the ToI are different. The number of possible values for attribute game was 4 (basket, football, golf, tennis). The other ones were the same. The queries that use categoryName form a group (Q9) since BV has 6 elements. The queries that use game form a second group (Q12). The other groups are built for the queries that use price and star (Q10 and Q11), and the queries that use subtype (Q13). The results are shown in Table 8.7.

Size of BV	6 (Q9)	4 (Q12)	3 (Q10)	2 (Q13)
Time (sec)	1800	22.67	145.48	13.46

Table 8.7: Performance of Experiment 2.2 (DB large)

Note that when using the large dataset, query Q12 (with *four* different possible instantiations for the variable) performs much better than Q10 (with *three* different possible instantiations for the variable). Then, we have not conclusive evidence on the influence of the size of the set BV over the performance, in particular when this value is relatively small. The next experiment is aimed at looking for other variable that can influence performance: the number of PoIs in the dataset.

8.2.3 Goal 3

We wanted to measure the relevance of the number of PoIs that satisfy some condition, considering it appears reasonable that the higher this number,the higher the amount of memory that must be used, decreasing the algorithm performance.

Experiment 3.1 We formulate the following experiment. We ran queries that contain constraints that were satisfied by different number of PoIs. In this case we ran the queries with large dataset containing 1230 PoIs. Here, the number of PoIs that match the condition game='basket', speciality='Beer', game='Golf', speciality='pizza' and speciality='pasta' are 102, 91, 61, 40 and 1, respectively. These are the queries we ran. In each query we always used the same condition:

Q14: `[game='basket'].[game='basket']`
Q15: `[speciality='beer'].[speciality='beer']`
Q16: `[game='golf'].[game='golf']`
Q17: `[speciality='Pizza'].[speciality='Pizza']`
Q18: `[speciality='Pasta'].[speciality='Pasta']`

The results are shown in Figure 8.4

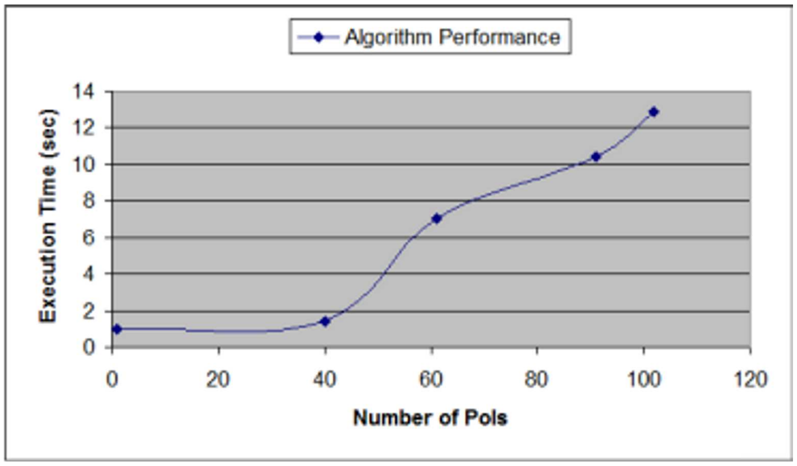


Figure 8.4: Experiment 3.1

We can see that as the number of PoIs that match a condition increases, the performance decreases, confirming our hypothesis.

8.2.4 Goal 4

Now we move to RE-SPaM^{++} queries that include Piet-QL queries. These new capabilities would not affect performance substantially due to the use of *macro and micro-cache* strategies. We verify this through the following experiment.

Experiment 4.1 We first ran queries that use a constraint of the form $[c1 \wedge c2]$, where c1 is the invocation to an extended function in RE-SPaM^{++} (the ones that browse a cursor over a collection of geometric identifiers returned by a Piet-QL query), and c2 is an equality expression. In our running example we use the extended function *containedBy*. After this, we run the query removing $c1$, that is, the query is of the form $[c2]$. The experiment is aimed at verifying the negative impact over the performance that invoking the extended function in the condition $c1$ may produce, against the positive impact of this condition because it may reduce the number of candidate sequences in the intermediate steps of the algorithm. The queries are:

Q19:

```
WITH TABLE distRiver(the_geom) AS
SELECT GIS DISTINCT(bel_dist.the_geom)
FROM bel_dist, bel_river
WHERE crosses(bel_dist.the_geom, bel_river.the_geom);
[containedBy(geom, distRiver.the_geom)='true' ∧ price='cheap'].
[containedBy(geom, distRiver.the_geom)='true' ∧ game='basket']
```

Q19-short:

```
[price='cheap'].[categoryName= 'basket']
```

Q20:

```
WITH TABLE distRiver(geom) AS
SELECT GIS DISTINCT(bel_dist.the_geom)
FROM bel_dist, bel_river
WHERE crosses(bel_dist.the_geom, bel_river.the_geom);
[containedBy(geom, distRiver.geom)='true' ∧ price='cheap'].
[containedBy(geom, distRiver.geom)='true' ∧ price='cheap']
```

Q20-short:

`[price='cheap'].[speciality='cheap']`

The results are shown in Table 8.8.

Query	Q19	Q19-short	Q20	Q20-short
Time (sec)	151.11	30.87	17.33	61.86
Number of sequences	40	41	20	99

Table 8.8: Performance of Experiment 4.1

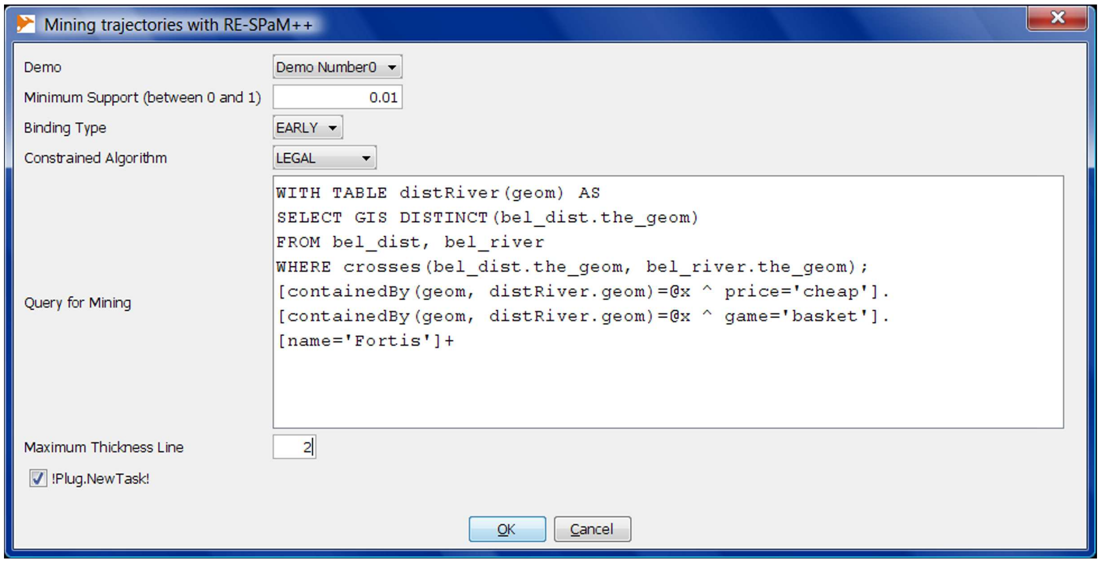
We can see that Q19-short is five times faster than Q19: in this case the overhead of evaluating a geometric function over the 1230 PoIs -again, the large dataset- (Phase 0 in Step 1) has more impact than the reduction in the intermediate steps, given that only one extra sequence was pruned due to the geometric condition. Instead, Q20 is approximately four times faster than Q20-short, since the former returns less sequences. It is worth noting that the use of geometric functions introduces and overhead at the beginning of the process because the geometric condition must be evaluated (i.e., the Piet-QL query).

8.3 Visualizing RE-SPaM^{++} results

In the MO setting, the result of applying a RE-SPaM^{++} query are trajectories. RE-SPaM^{++} discovers *semantically similar trajectories*, which, in a sense, is a way to perform trajectory aggregation. Visualizing these aggregated trajectories can help the data analyst to draw conclusions about the dataset at hand. We implemented a tools that allows to visualize semantically equivalent trajectories returned by an RE-SPaM^{++} query in the OpenJump framework. In our implementation, the trajectories that pass trough the same PoIs in the same order are visualized only once, in the same layer and the line that links these PoIs is drawn with a thickness proportional to a maximum thickness value, proposed by the user. This thickness is proportional to the number of trajectories in the path. Those trajectories that are *semantically equivalent* but do not pass though the same PoIs are drawn in different layers. Figure 8.5 shows the execution in our tool of the the following query:

```
WITH TABLE distRiver(geom) AS
SELECT GIS DISTINCT(bel_dist.the_geom)
FROM bel_dist, bel_river
WHERE crosses(bel_dist.the_geom, bel_river.the_geom);
[containedBy(geom, distRiver.geom)=@x ∧ price=‘cheap’].
[containedBy(geom, distRiver.geom)=@x ∧ game=‘basket’].
[name=’Fortis’]+
```

Figures 8.6 and Figure 8.7 depict the result of the query, visualizing it at different levels of detail.



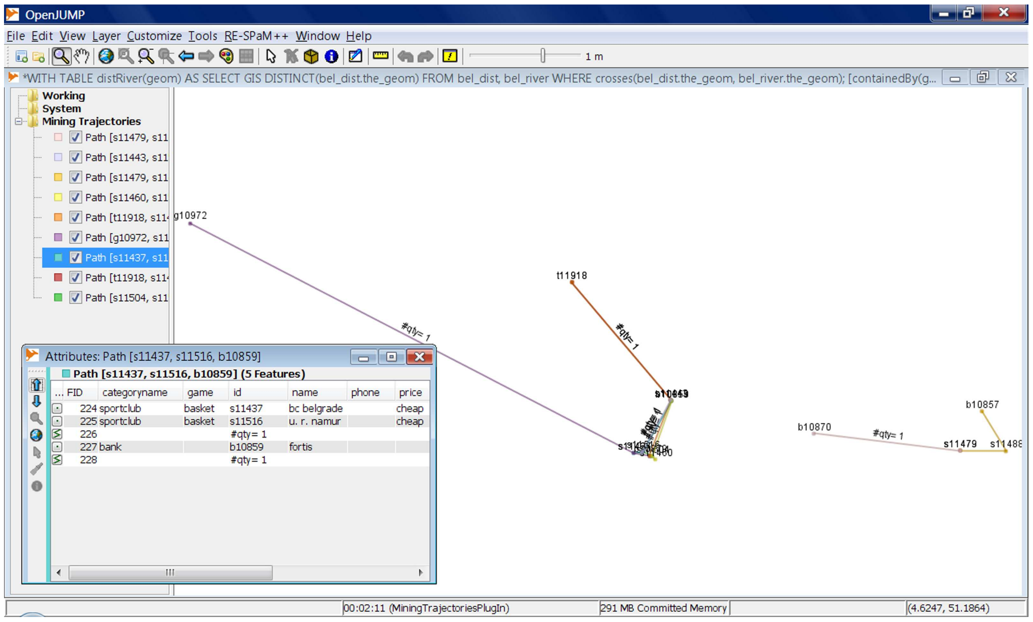
Demo	Demo Number0
Minimum Support (between 0 and 1)	0.01
Binding Type	EARLY
Constrained Algorithm	LEGAL
Query for Mining	WITH TABLE distRiver(geom) AS SELECT GIS DISTINCT(bel_dist.the_geom) FROM bel_dist, bel_river WHERE crosses(bel_dist.the_geom, bel_river.the_geom); [containedBy(geom, distRiver.geom)=@x ^ price='cheap']. [containedBy(geom, distRiver.geom)=@x ^ game='basket']. [name='Fortis']+
Maximum Thickness Line	2

☑ !Plug.NewTask!

Figure 8.5: Input Parameters for RE-SPaM^{++}

8.4 Summary

We have presented a case study were we show how our proposal can be applied, integrating, in a single framework, GIS data (in this case, real-world maps of Belgium), OLAP data (adapted from an external data warehouse), and moving object data (synthetic data). In addition, we performed experiments that provide insight into the parameters and conditions that affect the performance of the query language RE-SPaM^{++} that integrates these three worlds. Finally, we presented a visualization tool



*WITH TABLE distRiver(geom) AS SELECT GIS DISTINCT(bel_dist.the_geom) FROM bel_dist, bel_river WHERE crosses(bel_dist.the_geom, bel_river.the_geom); [containedBy(g...

Attributes: Path [s11437, s11516, b10859]

Path [s11437, s11516, b10859] (5 Features)

...	FID	categoryname	game	id	name	phone	price
	224	sportclub	basket	s11437	bc belgrade		cheap
	225	sportclub	basket	s11516	u. r. namur		cheap
	226			#qty= 1			
	227	bank		b10859	fortis		
	228			#qty= 1			

00:02:11 (MiningTrajectoriesPlugIn) | 291 MB Committed Memory | (4.6247, 51.1864)

Figure 8.6: Semantically Equivalent Trajectories

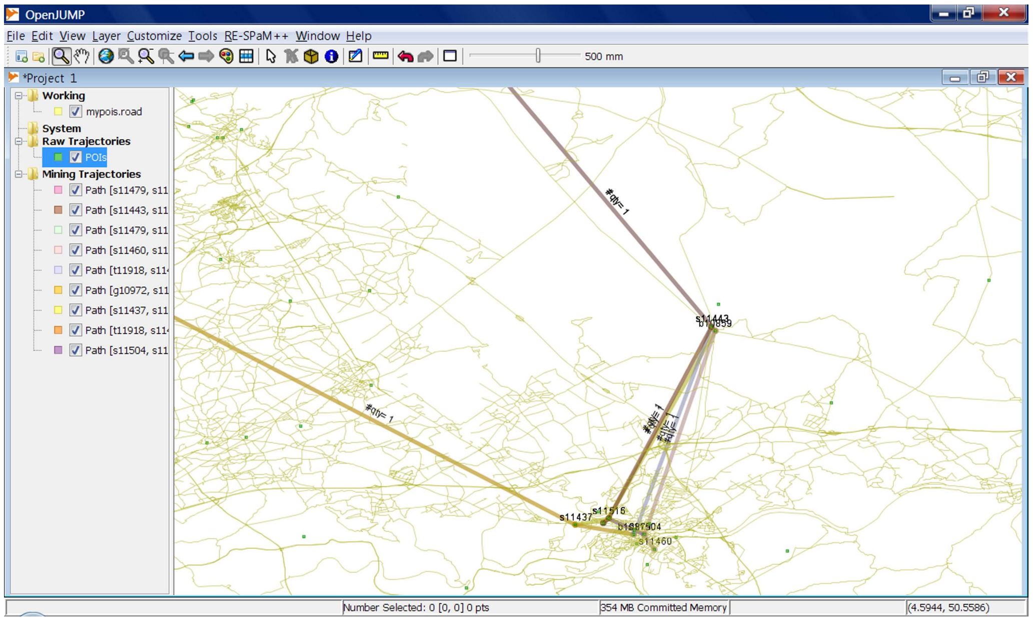


Figure 8.7: Semantically Equivalent Trajectories in the Road Network

allowing to run RE-SPaM^{++} queries and display the results.

Chapter 9

Conclusion

In this thesis we have shown that MO data can be effectively integrated with GIS and OLAP data. This integration is achieved through a framework based on a formal data model that we have denoted Piet. A formal query language for spatial aggregation and an SQL-like language, Piet-QL, based on the OGC and MDX standards supports OLAP and GIS integration. To integrate MO data into this framework, we first showed that trajectories can be semantically enriched with information about the geography where they develop. We discuss how mining techniques can identify *semantically equivalent* trajectories, contributing to the detection of patterns that cannot be discovered with traditional approaches. We then introduced a sequential pattern mining algorithm and a regular expression query language (called RE-SPaM) to restrict the number of sequences to be discovered. This proposal differs from previous ones in many ways: first, it formalizes all the concepts related to the mining process by means of a data model; second, it supports categorical attributes, and the use of functions and variables; third, expresses patterns as regular expressions instead of sequences (i.e., patterns are expressed by intension rather than by extension); finally, it introduces the idea of computing the support of a regular expression. Finally, we showed how Piet-QL queries can be included in RE-SPaM expressions, leading to a more powerful language called RE-SPaM^{++}, that implements the integration of MO into de GIS-OLAP framework.

We show how our proposal can be used for data analysis by means of a case study based on a collection of maps of Belgium. For this, we produced a collection of synthetic trajectories and created a data warehouse. Over this setting, we performed a set of experiments aimed at showing that our tools can be used in a real world situation with good results.

We have implemented the Piet-QL language, the RE-SPaM mining algorithm and the RE-SPaM^{++} language. The results obtained in Piet-QL language are visualized in a plug-in that was developed for the OpenJump platform. Queries returning spatial data are shown in map layers. The results of the queries returning OLAP data are displayed in a chart for browsing. For RE-SPaM^{++}, the semantic trajectories that satisfy a certain RE are visualized in layers of OpenJump.

9.0.1 Open Research Directions

In traditional sequential pattern discovery the support of a sequence is computed as the number of data-sequences satisfying a pattern with respect to the total number of data-sequences in the database. In our approach it is the fraction of number of trajectories satisfying a RE over the total number of trajectories. If the attributes of the items are frequently updated, this could lead to incorrect or at least incomplete conclusions. For instance, in the examples that we have discussed in Chapter 8, we assumed that the schema of categories and also the values associated to category occurrences are invariant over time. The pattern discovery algorithm assumes that no changes have occurred in these categories. In light of this, for some situations it can be important to revise the classic notion of support in sequential pattern mining and consider the notion of *Temporal Support of a RE*, that accounts for the commented category updates.

Another open research direction comes from the area of so-called *continuous fields*, which describe the distribution of physical phenomena that change continuously in time and/or space. Examples of such phenomena are temperature, pressure, or land elevation. Besides physical geography, continuous fields, like land use and population density, are used in human geography as an aid in spatial decision making process. Although some work has been done to support querying fields in GIS, the area of spatial multidimensional analysis of continuous data is still almost unexplored. Integrating spatiotemporal continuity within multidimensional structures poses numerous challenges. Further, existing multidimensional structures and models dealing with discrete data, are not adequate for the analysis of continuous phenomena. Multidimensional models and associated query languages are thus needed, to support continuous data. The work presented in this thesis could be used as a starting point for integrating OLAP and continuous field data.

Bibliography

[1] *Jmap spatial olap*, White paper, KHEOPS Technologies, 2005, http://www.kheops-tech.com/en/jmap/solap.jsp.

[2] R. Agrawal, T. Imieliński, and A. Swami, *Mining association rules between sets of items in large databases*, SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data (New York, NY, USA), ACM, 1993, pp. 207–216.

[3] R. Agrawal and R. Srikant, *Fast algorithms for mining association rules in large databases*, VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1994, pp. 487–499.

[4] R. Agrawal and R. Srikant, *Mining sequential patterns*, ICDE, 1995, pp. 3–14.

[5] L. O. Alvares, V. Bogorny, B. Kuijpers, J. A. Fernandes de Macêdo, B. Moelans, and A. A. Vaisman, *A model for enriching trajectories with semantic geographical information*, GIS, 2007, p. 22.

[6] Y. Bédard, T. Merrett, and J. Han, *Fundamentals of spatial data warehousing for geographic knowledge discovery*, Geographic data mining and Knowledge Discovery (GKD), Taylor & Francis, 2001, pp. 53–73.

[7] Y. Bédard, S. Rivest, and M. J. Proulx, *Spatial on-line analytical processing (solap): Concepts, architectures and solutions from a geomatics engineering perspective*, ch. 13, pp. 298–319, IRM Press (Idea Group), 2007.

[8] S. Bimonte, A. Tchounikine, and M. Miquel, *Towards a spatial multidimensional model*, DOLAP, 2005, pp. 39–46.

[9] S. Brakatsoulas, D. Pfoser, and N. Tryfona, *Modeling, storing, and mining moving object databases*, IDEAS, 2004, pp. 68–77.

[10] L. Cabibbo and R. Torlone, *Querying multidimensional databases*, DBPL, 1997, pp. 319–335.

[11] E. Clementini and P. Di Felice, *A model for representing topological relationships between complex geometric features in spatial databases*, Inf. Sci. **90** (1996), no. 1-4, 121–136.

[12] S. Cohen, *Equivalence of queries combining set and bag-set semantics*, PODS, 2006, pp. 70–79.

[13] J da Silva, V. C. Times, and A. C. Salgado, *An open source and web based framework for geographic and multidimensional processing*, SAC, 2006, pp. 63–67.

[14] J. da Silva, A. S. Castro Vera, A. Grisi de Oliveira, R. do Nascimento Fidalgo, A. C. Salgado, and V. Cesário Times, *Querying geographical data warehouses with geomdql*, SBBD, 2007, pp. 223–237.

[15] P. A. M. Dirac, *Principles of quantum mechanics*, Oxford University Press, 1958.

[16] C. du Mouza and P. Rigaux, *Mobility patterns*, GeoInformatica **9** (2005), no. 4, 297–319.

[17] A. Escribano, L. I. Gómez, B. Kuijpers, and A. A. Vaisman, *Piet: a gis-olap implementation*, DOLAP, 2007, pp. 73–80.

[18] R. Fidalgo, V. C. Times, J. da Silva, and Fernando da Fonseca de Souza, *Geodwframe: A framework for guiding the design of geographical dimensional schemas*, DaWaK, 2004, pp. 26–37.

[19] E. Frentzos, K. Gratsias, and Y. Theodoridis, *Index-based most similar trajectory search*, ICDE, 2007, pp. 816–825.

[20] M. Garofalakis, R. Rastogi, and K. Shim, *Mining sequential patterns with regular expression constraints*, IEEE Trans. on Knowl. and Data Eng. **14** (2002), no. 3, 530–552.

[21] M. N. Garofalakis, R. Rastogi, and K. Shim, *Spirit: Sequential pattern mining with regular expression constraints*, VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1999, pp. 223–234.

[22] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi, *Trajectory pattern mining*, KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (New York, NY, USA), ACM, 2007, pp. 330–339.

[23] L. I. Gómez, S. Haesevoets, B. Kuijpers, and A. A. Vaisman, *Spatial aggregation: Data model and implementation*, Inf. Syst. **34** (2009), no. 6, 551–576.

[24] L. I. Gómez, B. Kuijpers, and A. A. Vaisman, *Querying and mining trajectory databases using places of interest*, New Trends in Data Warehousing and Data Analysis, vol. 3, Springer US, 2009, pp. 1–26.

[25] L. I. Gómez and A. A. Vaisman, *Efficient constraint evaluation in categorical sequential pattern mining for trajectory databases*, EDBT, 2009, pp. 541–552.

[26] L. I. Gómez, A. A. Vaisman, and S. Zich, *Piet-ql: a query language for gis-olap integration*, GIS '08: Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems (New York, NY, USA), ACM, 2008, pp. 1–10.

[27] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. P. Sondag, *Adaptive fastest path computation on a road network: A traffic mining approach*, VLDB, 2007, pp. 794–805.

[28] S. Grumbach and T. Milo, *Towards tractable algebras for bags*, J. Comput. Syst. Sci. **52** (1996), no. 3, 570–588.

[29] R. H. Güting and M. Schneider, *Moving objects databases*, Morgan Kaufmann, 2005.

[30] J. Han and M. Kamber, *Data mining: Concepts and techniques*, Morgan Kaufmann Publishers, 2001.

[31] J. Han, N. Stefanovic, and K. Koperski, *Selective materialization: An efficient method for spatial data cube construction*, PAKDD, 1998, pp. 144–158.

[32] V. Harinarayan, A. Rajaraman, and J. D. Ullman, *Implementing data cubes efficiently*, SIGMOD Conference, 1996, pp. 205–216.

[33] K. Hornsby and M. J. Egenhofer, *Modeling moving objects over multiple granularities*, Ann. Math. Artif. Intell. **36** (2002), no. 1-2, 177–194.

[34] R.F. Hoskins, *Generalised functions*, Ellis Horwood Series: Mathematics and its applications, John Wiley & Sons, 1979.

[35] C. A. Hurtado, A. O. Mendelzon, and A. A. Vaisman, *Maintaining data cubes under dimension updates*, ICDE, 1999, pp. 346–355.

[36] C. S. Jensen, A. Kligys, T. B. Pedersen, and I. Timko, *Multidimensional data modeling for location-based services*, VLDB J. **13** (2004), no. 1, 1–21.

[37] R. Kimball, *The data warehouse toolkit: Practical techniques for building dimensional data warehouses*, John Wiley, 1996.

[38] A. C. Klug, *Equivalence of relational algebra and relational calculus query languages having aggregate functions*, J. ACM **29** (1982), no. 3, 699–717.

[39] B. Kuijpers and A. A. Vaisman, *A data model for moving objects supporting aggregation*, ICDE Workshops, STDM, 2007, pp. 546–554.

[40] G. M. Kuper, L. Libkin, and J. Paredaens (eds.), *Constraint databases*, Springer, 2000.

[41] G. M. Kuper and M. Scholl, *Geographic information systems*, Constraint Databases, 2000, pp. 175–198.

[42] J. Lee, J. Han, and K. Whang, *Trajectory clustering: a partition-and-group framework*, SIGMOD Conference, 2007, pp. 593–604.

[43] H. Lenz and A. Shoshani, *Summarizability in olap and statistical data bases*, SSDBM, 1997, pp. 132–143.

[44] I. Vega López, R. T. Snodgrass, and B. Moon, *Spatiotemporal aggregate computation: a survey*, IEEE Trans. Knowl. Data Eng. **17** (2005), no. 2, 271–286.

[45] E. Malinowski and E. Zimányi, *Olap hierarchies: A conceptual perspective*, CAiSE, 2004, pp. 477–491.

[46] E. Malinowski and E. Zimányi, *Representing spatiality in a conceptual multidimensional model*, GIS, 2004, pp. 11–12.

[47] E. Malinowski and E. Zimányi, *Hierarchies in a multidimensional model: From conceptual modeling to logical representation*, Data Knowl. Eng. **59** (2006), no. 2, 348–377.

[48] E. Malinowski and E. Zimányi, *Logical representation of a conceptual model for spatial data warehouses*, GeoInformatica **11** (2007), no. 4, 431–457.

[49] H. Mannila, H. Toivonen, and A. I. Verkamo, *Discovering frequent episodes in sequences*, KDD, 1995, pp. 210–215.

[50] N. Meratnia and R. A. de By, *Aggregation and comparison of trajectories*, ACM-GIS, 2002, pp. 49–54.

[51] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao, *Efficient olap operations in spatial data warehouses*, SSTD, 2001, pp. 443–459.

[52] D. Papadias, Y. Tao, P. Kalnis, and J. Zhang, *Indexing spatio-temporal data warehouses*, ICDE, 2002, pp. 166–175.

[53] D. Papadias, Y. Tao, J. Zhang, N. Mamoulis, Q. Shen, and J. Sun, *Indexing and retrieval of historical aggregate information about moving objects*, IEEE Data Eng. Bull. **25** (2002), no. 2, 10–17.

[54] T. B. Pedersen and N. Tryfona, *Pre-aggregation in spatial data warehouses*, SSTD, 2001, pp. 460–480.

[55] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and Mei-Chun Hsu, *Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth*, ICDE '01: Proceedings of the 17th International Conference on Data Engineering (Washington, DC, USA), IEEE Computer Society, 2001, p. 215.

[56] J. Pei, J. Han, and W. Wang, *Constraint-based sequential pattern mining: the pattern-growth methods*, J. Intell. Inf. Syst. **28** (2007), no. 2, 133–160.

[57] N. Pelekis, I. Kopanakis, G. Marketos, I. Ntoutsi, G. L. Andrienko, and Y. Theodoridis, *Similarity search in trajectory databases*, TIME, 2007, pp. 129–140.

[58] E. Pourabbas, *Cooperation with geographic databases*, Multidimensional Databases, 2003, pp. 393–432.

[59] F. Rao, L. Zhang, X. Yu, Y. Li, and Y. Chen, *Spatial hierarchy and olap-favored search in spatial data warehouse*, DOLAP, 2003, pp. 48–55.

[60] P. Rigaux, M. Scholl, and A. Voisard, *Spatial databases*, Morgan Kaufmann Publishers Inc., 2002.

[61] S. Rivest, Y. Bédard, and P. Marchand, *Toward better support for spatial decision making: Defining the characteristics of spatial, on-line analytical processing*, 2001.

[62] S. Rizzi and M. Golfarelli, *Date warehouse design*, ICEIS, 2000, pp. IS 39–42.

[63] S. Shekhar, C. Lu, X. Tan, S. Chawla, and R. Vatsavai, *Map cube: A visualization tool for spatial data warehouses*, Geographic data mining and Knowledge Discovery (GKD) (H. Miller and J. Han, eds.), Taylor & Francis, 2001, pp. 74–109.

[64] S. Spaccapietra, C. Parent, M. L. Damiani, J. A. Fernandes de Macedo, F. Porto, and C. Vangenot, *A conceptual view of trajectories*, Technical Report, 2007.

[65] R. Srikant and R. Agrawal, *Mining sequential patterns: Generalizations and performance improvements*, EDBT, 1996, pp. 3–17.

[66] N. Stefanovic, J. Han, and K. Koperski, *Object-based selective materialization for efficient implementation of spatial data cubes*, IEEE Trans. Knowl. Data Eng. **12** (2000), no. 6, 938–958.

[67] A. A. Vaisman, *Data quality-based requirements elicitation for decision support systems - in data warehousing and olap: Challenges and solutions*, IDEA Group, 2006.

[68] A. A. Vaisman, *Requirements elicitation for decision support systems: A data quality approach*, ICEIS (3), 2006, pp. 316–321.

[69] M. Vazirgiannis and O. Wolfson, *A spatiotemporal model and language for moving objects on road networks*, SSTD, 2001, pp. 20–35.

[70] G. Wang. and P. Liu, *Temporal management of rfid data*, VLDB '05: Proceedings of the 31st international conference on Very large data bases, VLDB Endowment, 2005, pp. 1128–1139.

[71] O. Wolfson, A. P. Sistla, B.Xu, J. Zhou, and S. Chamberlain, *Domino: Databases for moving objects tracking*, SIGMOD Conference, 1999, pp. 547–549.

[72] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang, *Moving objects databases: Issues and solutions*, SSDBM, 1998, pp. 111–122.

[73] M. F. Worboys, *Gis: A computing perspective*, Taylor & Francis, 1995.

[74] X. Yan, J. Han, and R. Afshar, *Clospan: Mining closed sequential patterns in large databases*, SDM, 2003.

[75] L. Zhang, Y. Li, F. Rao, X. Yu, Y. Chen, and D. Liu, *An approach to enabling spatial olap by aggregating on spatial hierarchy*, DaWaK, 2003, pp. 35–44.