Modeling, Querying and Indexing Moving Objects with Sensors on Road Networks

Iulian Sandu Popa

No. d'ordre :

THESE

De

Doctorat de l'Université de Versailles Saint-Quentin-en-Yvelines

Présentée et soutenue publiquement par

Iulian SANDU POPA

le 1er décembre 2009

Pour obtenir le grade de

Docteur en Informatique de l'Université de Versailles Saint-Quentin-en-Yvelines

Titre de la thèse :

Modélisation, interrogation et indexation de données de capteurs à localisation mobile dans un réseau routier

JURY

M. Ralf H. GUTING	Université Fern, Hagen	Rapporteur
M. Michel SCHOLL	CNAM, Paris	Rapporteur
M. Jacques EHRLICH	LIVIC, INRETS-LCPC	Examinateur
Mme Karine ZEITOUNI	Université de Versailles	Codirectrice de thèse
M. Georges GARDARIN	Université de Versailles	Codirecteur de thèse

Ph. D. Thesis

Modeling, Querying and Indexing Moving Objects with Sensors on Road Networks

Iulian Sandu Popa

University of Versailles Saint-Quentin

Table of Contents

Index of Figures

Index of Tables

Chapter I: Introduction

I.1 Context

Humans, like all things in the universe are in constant motion. Transport (i.e., the movement of people and goods from one location to another) is an element that can not be dissociated from society. With the constant progress in technique and modes of transportation, the movements are more and more rapid and expanding continuously. This fast development puts an enormous pressure on the infrastructure and on the environment. Many problems concerning the transport safety, the transport planning and the transport impact on the environment, require to be dealt with. Hence, there is an increased need today for appropriate observation tools, as well as for tools for planning, analyzing and decision making.

In this context, several geolocation tools emerged such as GPS (Global Positioning System) or Galileo (improved European equivalent of GPS that should be operational by 2013), or systems for monitoring the infrastructure and the traffic. These systems produce geolocalized data flows that offer global or local views on the mobile users of a transportation mode and on the infrastructure.

The geolocation systems have developed considerably with the introduction of the *GPS system*. GPS consists in three parts: a number of satellites (i.e., between 24 and 32) orbiting the Earth, several control stations on Earth and the GPS receivers of the users. Since it became fully operational in 1995, the GPS has been widely adopted in numerous application domains. Most transportation systems use GPS to provide navigation for ground, maritime and aviation operations. Also, the GPS is employed by many other applications such as tracking, surveillance and map-making. Moreover, the GPS serves as a tool for scientific studies, e.g., earthquakes. The precise reference time that it offers is used as a time synchronization source, e.g., for cellular network protocols.

These geolocation systems integrate more and more the *Intelligent Transportation Systems* (ITS). ITS regroup different technologies, whose main objective is to improve the processes of observation, decision and planning in the transportation domain. Such systems have known a strong development since the early 90s. Their flourishing is associated to the development of *embedded sensors* and *embedded computer systems* into vehicles and into the infrastructure of transportation networks. Many types of applications are based on the ITS technologies such as (real-time) traffic management, incident detection in transportation networks, route planning, individual behavior analysis, transport planning at different scales, safety.

Along with the development of GPS and ITS systems, the area of *Geographic Information Systems* (GIS) has also flourished in the last two decades. A geographic information system, or geographical information system, captures, stores, analyzes, manages, and presents data that is linked to location. In a more generic sense, GIS applications are tools that allow users to create interactive queries (user created searches), analyze spatial information, edit data, maps, and present the results of all these operations. In simplest terms, GIS is the merging of graphic map entities and databases. Appeared in the early 60s, the GIS systems largely became available in the late 90s thanks to the development of computer

graphics and spatial databases. Nowadays, their scope is expanding to spatio-temporal data encouraged by the development of above mentioned GPS and ITS systems.

Combining geolocalized data, intelligent transportation systems and geographic information systems opens the way for novel applications in the transportation domain. In this context, large amounts of spatio-temporal data are produced every day. The state of the art databases today offer limited capabilities for managing this type of data. Therefore, finding appropriate data models and algorithms remains an important issue to solve for the database research community.

I.2 Issues Related to Managing Moving Objects with Sensors

Let us consider an example of application in the transportation safety area. In most cases, it was shown that driver's failing is the main cause of accidents. Thus, Advanced Driving Assistance Systems (ADAS) are introduced to help the driver and prevent its failing. But, to be efficient and well accepted, ADAS must fit as well as possible with the situation in order to make it safer and conform to the driver's behavior and expectations. Therefore, it is important to have a good knowledge of driver behavior in normal or pre-crash situations. Up to now, except for data from driving simulator (distorted sometimes by the simulation environment), we have very few information that could permit to understand how drivers act in normal condition or to assess the impact of an ADAS by comparing between driving with and without the system.

Since a few years, the concept of "naturalistic driving" [32], that consists in collecting a large amount of data from a large number of drivers over an extended period of time in natural situation, makes possible to lead in-depth behavior and epidemiological analysis. Naturalistic driving becomes economically possible because modern vehicles are natively equipped with sensors, avoiding extensive and expensive instrumentation. Thus, vehicles need only to be equipped with a data logger which mainly records data from the car multiplexed sensor bus (CAN). Besides the tools required for proper data collection the question of data management must also be addressed. Actually, the database is an important component of the whole data-chain because it will contain a huge amount of data that will be quite unworkable if badly designed. Moreover, data from naturalistic driving experiences have three important specific characteristics: (i) it concerns moving objects, (ii) it is generally doubly referenced (over the time and over the location), and (iii) the location itself could be map-matched on road networks provided with a topology. These characteristics are generic and remain valid for any type of application related to mobile sensors.

However, the state of the art database management systems (DBMSs) fail to handle such complex data and their processing. This leads us to develop an extended DBMS to fulfill the requirements of such applications. The research in the area of spatio-temporal databases started nearly two decades ago, when handling data that are related to both spatial and temporal changes, was acknowledged to be an unavoidable task. However, dealing with this problem was not a straightforward work. This effort was preceded by another decade of research in both temporal [1] and spatial [2] databases, which founded the basis for spatio-temporal database models.

One known data model in spatio-temporal databases is the *moving object* data model [4], which gave rise to the field of *moving objects databases* [3]. The moving objects data model

has the ability to capture the continuous movement and/or changes of moving entities. If only the position in space of an object is relevant, then *moving point* is a basic abstraction. If also the extent is of interest, then the *moving region* abstraction captures moving as well growing or shrinking regions. The model includes a language for querying about the past states of the moving objects. Implementation of spatio-temporal data models exist only at prototype level (e.g., SECONDO [28], HERMES [26]).

However, the moving object data model considers only the location of the moving object and can not be generalized to measures ranging along a spatio-temporal trajectory (as it is the case of the considered application type). Moreover, although these measures initially correspond to a temporal data stream, their variation is more dependent on their location on the transportation network than time: for example, the variation of speed is usually constrained by the geometry of the road and the speed limit. Also, the temporal analysis of data of different trajectories is irrelevant because on the one hand they are asynchronous and on the other hand, this comparison makes sense only if these paths overlap in space. Therefore, we must capture the spatial variability of these measures and allow its manipulation through the data model and query language. Besides, the data model must take into account the fact that, in transportation, moving objects (e.g., vehicles, trains) move in spatially embedded networks (e.g., roads, highways, even airlines have fixed routes).

In addition to an appropriate data model and query language for mobile sensor data, the optimization aspects need also be addressed. There are many possible directions for optimizing a database system such as efficient algorithms for the operations over the data types, appropriate access methods, translations rules for extensions of the optimizer, and so on.

I.3 Thesis Objectives and Contributions

The conventional database management systems were conceived for relative simple business applications. For example, the data types available for attribute types in the relational model are simple: basically integers, floating-point numbers, or short text strings. One goal of the database research in the last two decades has been to widen the scope, so that as much as possible any kind of data used by any application can be managed within a DBMS, described by a high-level data model and accessed by a powerful query language. The objective of this thesis is to design and to develop a database system that is capable of managing the data produced by mobile sensors.

Most (commercial or open source) database engines today offer extending capabilities to meet the needs of some application domain. These extendible DBMS architectures can be used to integrate a new data model into a DBMS. A DBMS extension should comprise several components such as the following [3]: new data types, operations that encapsulate the data types, appropriate access methods, cost functions for all methods (to be used by the query optimizer), extensions of the optimizer (e.g., in the form of translation rules), user interface extensions to handle presentation of data types.

The contributions of this thesis are:

- *Design of a data model and a query language for moving sensor data*. We propose a new data model and a language that handle mobile location sensor data, as an extension of the moving object data model proposed in [4], [5]. To this end, we

introduce the concept of measure profile to capture the measure variability in space, along with specific operations that permit to analyze the data. We also describe their implementation using the object-relational paradigm.

- *Design of an access method for in-network trajectories*. Complementary to modeling, we address some optimization aspects. We propose a new access method (that we name PARINET) to efficiently retrieve the trajectories of objects moving in networks. PARINET is designed for historical data and relies on the distribution of the data over the network as for historical data, the data distribution is known in advance. Because the network can be modeled using graphs, the partitioning of the trajectory data is based on graph partitioning theory and can be tuned for a given query load. PARINET can easily be integrated in any RDBMS, which is an essential asset particularly for industrial or commercial applications.

- *Development of a spatio-temporal DBMS extension*. Based on the above proposals, we develop a spatio-temporal DBMS prototype called CALM. CALM is implemented as an Oracle DBMS extension. We create new data types, new operations and new index types by using the extensibility capabilities of Oracle 11g data server.

- *Development of a graphical user interface for visualizing mobile sensor data*. We implement a graphical user interface to handle the presentation of data types in CALM. To this end, we use GeoServer [111] coupled with GoogleMaps [112].

I.4 Thesis Plan

This document is organized in seven chapters. Figure I.1 schematizes the organization of the thesis and the relationships between the chapters. The current (first) chapter introduces the subject of the thesis. The next two chapters are concerned with the spatio-temporal data modeling, i.e., the second chapter presents the related work on spatio-temporal data modeling and the third chapter introduces our proposed model for mobile sensor data. The following two chapters deal with spatio-temporal indexing, i.e., in the forth chapter we present the related work on spatio-temporal access methods, whereas in the fifth chapter we introduce a new access method for trajectories of objects moving in networks. The sixth chapter evaluates the prototype CALM. We conclude and give the directions for future work in the seventh chapter. In the rest of this section we detail the content of the main chapters of the thesis, i.e., from Chapter II to Chapter VI.

We begin the Chapter II with a short presentation of some base concepts on spatial and temporal databases. Then, we introduce the needs for modeling moving objects. We also present some of the related works for modeling current and past moving objects. We detail the state-of-the-art model and query language for moving objects trajectories (for non-constrained and constrained kinds of movement), along with existing prototypes of moving objects database systems.

In Chapter III, we propose a new data model and a language that handle mobile location sensor data. We first motivate our approach by presenting a real application and a query scenario. Then, we describe the proposed model, i.e., the new type and the corresponding operations, and demonstrate its use by expressing some query examples. Finally, we discuss several aspects of the implementation, e.g., database system architecture, data type representation, handling aggregate functions and operators.

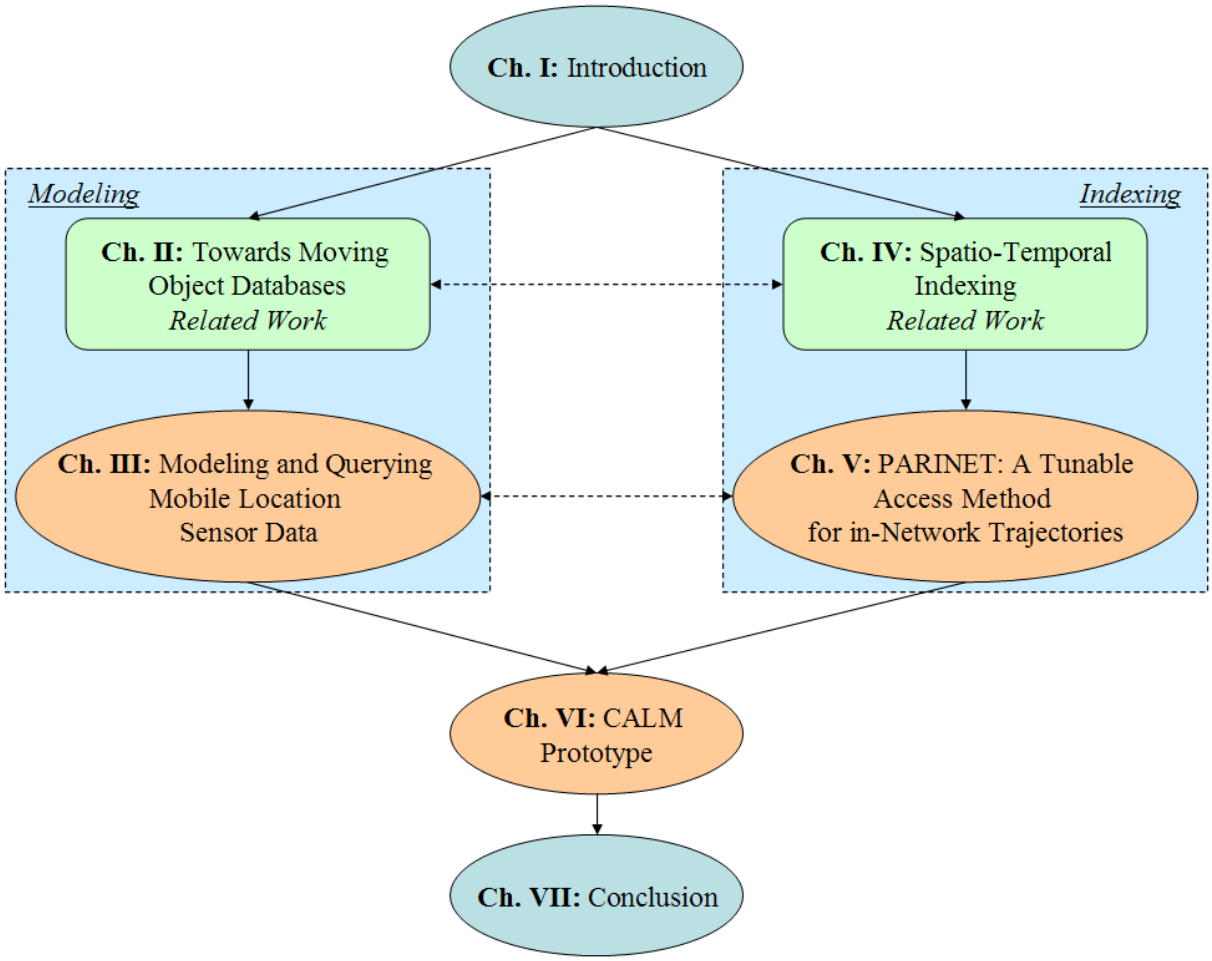


Figure I.1: Organization of the thesis

In Chapter IV we present some of the related work on spatial, temporal and spatio-temporal indexing. We begin with an overview of a few spatial access methods. Then, we focus on the interval indexing techniques. Spatial and interval indexing are very important in a spatio-temporal DBMS for efficient processing of spatial and temporal operators. We continue with a few access methods for indexing past and current and near-future movement. Finally, we present in detail the work concerning the indexing of the past trajectories of moving objects in networks, since this is directly related with our proposed method in the fifth chapter.

In Chapter V we propose PARINET, a new access method to efficiently retrieve the trajectories of objects moving in networks. PARINET is designed for historical data and relies on the distribution of the data over the network as for historical data, the data distribution is known in advance. We first introduce the context for PARINET by defining the network model, the data model and the query types. Then, we describe the proposed access method along with the cost model and the tuning process. We conclude the chapter with the experimental evaluation of PARINET.

In Chapter VI we present some practical aspects of the prototype CALM, which is a spatio-temporal DBMS (STDMS) extension implementing the proposed data model and index. We detail the upstream data processing (i.e., before integrating the data into CALM) and the downstream data processing (i.e., post-processing of the results obtained in CALM)

from the spatio-temporal database point of view. In the upstream, an important work is required before integrating the data in the STDBMS. In the downstream, tools that can help analyzing the data are needed, such as a graphical interface. We also evaluate our prototype by using a naturalistic driving data set and several query scenarios.

Chapter II: Towards Moving Objects Databases

II.1 Introduction

Space and time represent the fundamentals of the physical world. All physical entities are characterized by a location and an extent in (three-dimensional) space. Moreover, the location and the extent can vary in time. This applies to all kinds of objects such as people, animals, vehicles, trains, planes, but also forests, deserts, lakes, cities, or countries.

An impressive number of application domains that are based on the objects' spatio-temporal information have been flourishing in the last two decades. Among these application domains we enumerate: transport, Intelligent Transportation Systems (ITS), environmental studies, Geographic Information Systems (GIS), meteorology, mobile computing, location-based services, logistics. All these types of applications require manipulating spatial, temporal and spatio-temporal data, which brings the need for a database system that is capable of managing spatio-temporal data.

However, the classical database management systems (DBMSs) can handle only basic data types. The early DBMSs were conceived for relatively simple applications that could be modeled by base data types such as real or integer numbers and strings of characters of limited size. A constant in the database research field has been to extend the scope of a database system in order to keep the pace with the technical progress and accommodate new types of applications. Modern applications require storing pictures, video, music, or maps into a DBMS. Moreover, one would then need to interrogate this kind of data and obtain the results as fast as possible. Hence, an appropriate query language and optimizations must also be provided within the DBMS.

In the case of spatio-temporal data, nearly two decades of research has led to a field that is known as *moving objects databases* (MOD). Besides, this effort was preceded by another decade of research in the fields of temporal and spatial databases, which constitute the grounds for MOD. In this chapter we present the related work in this area.

There are three main objectives that need to be considered when designing a moving objects database [3]. These objectives indicate the limitations of a classical DBMS that we have to surpass in order to obtain spatio-temporal DBMS:

- We should be able to represent and query geometries in the database. This is not possible within classical DBMSs in straightforward manner (except for points that could be stored as several base types indicating the spatial coordinates).

- We should be able to keep track of the evolution of geometries in time. Again, this is not possible since conventional DBMSs only keep a single state (snapshot) of the data. All changes (i.e. inserts, deletes, updates) lead to a new state and the last one is normally lost.

- Applications dealing with moving objects can consider either the past (history) and/or the present and the future of the data. Therefore, a moving objects database should be able to handle at least one of the two cases, i.e., store past spatio-temporal data or keep track of present moving objects and anticipate on their future positions.

We shortly address the first two limitations in Section II.2. We address the last one in more detail in Section II.3.

The rest of this chapter is organized as follows. In Section II.2 we shortly present some base concepts on spatial and temporal databases. Section II.3 introduces the needs for modeling moving objects. We also present some of the related works for modeling current and past moving objects. Section II.4 details the state-of-the-art model and query language for moving objects trajectories (for non-constrained and constrained kinds of movement), along with existing prototypes of MOD systems. We conclude the chapter in Section II.6.

II.2 Spatial and Temporal Databases

Spatio-temporal databases find their origins in spatial and in temporal databases. In this section, we shortly describe some of the essential features of spatial database systems and temporal database systems with an emphasis on the first. We will only insist on the aspects that are directly related to the work we cover in this thesis.

Thus, regarding the spatial databases part, we present the fundamental spatial entities found in most spatial models. Then, we discuss some spatial classes of operations and give some query examples. Finally, we present the spatial database extension of a state-of-the-art object-relational DBMS that we used for implementing our prototype. Notice that the classical spatial indexing techniques are discussed in Chapter IV.

For the temporal databases part, we will cover the basic concepts and definitions of the time domain (i.e., its structure and dimensionality) together with the standard temporal data types defined in the ODMG Standard [6]. Some temporal index techniques for time intervals are discussed in Chapter IV of this thesis. Other aspects such as temporal data models, temporal query languages or implementation issues are not covered here.

II.2.1 Spatial Databases

As indicated in the introduction, the goal of any database extension is to accommodate new types of data and to permit querying these data. Therefore, spatial databases offer data types for storing geometries and specific operations to handle geometries. These types and operations are integrated in the DBMS' SQL-like query language together with the existing (conventional) data types and operations. We discuss more on the data types, operations and their integration in spatial databases in the following subsections. In addition to the data model and the query language, the optimization aspects need to be studied [3]. Spatial operations may require complex geometrical computations. Moreover, spatial databases tend to be very large. Hence, effective multidimensional access methods have to be implemented in the DBMS to accelerate spatial searches. Finally, although complementary to a certain degree to the spatial DBMS, the visualization of spatial data types as maps is also an important component.

One of the main beneficiaries of spatial databases was represented by GIS applications. This domain that arose in the early 60s, began flourishing only in the early 90s with the birth

of spatial databases. However, besides GIS applications there are many other application domains that can benefit from a spatial database such as 3D modeling, VLSI design or molecular biology. Spatial databases permit building an application directly on top of the DBMS, thus providing full support for such applications. Many commercial database vendors provide today spatial extensions to their database server (e.g., Oracle, IBM DB2, SQL Server, Informix). Moreover, spatial extension can now be found even in open source databases (e.g., PostgreSQL, MySQL).

II.2.1.1 Modeling Spatial Concepts

A spatial database should permit the representation of any entity found in the geographic space, e.g., highways, railroads, water pipelines, cities, boundaries of countries, hospitals, gas stations, rivers, forests, lakes, and so forth. All these diverse entities are modeled either as *single objects* or *spatially related collections of objects*.

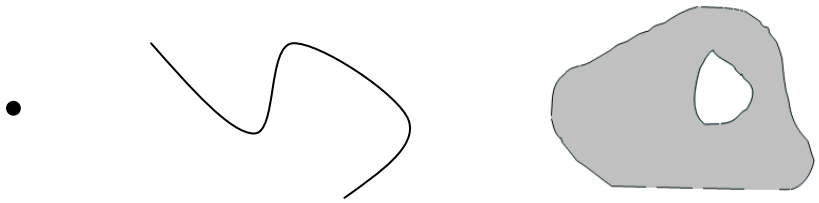


Figure II.1: The three basic abstractions: point, line and region

There are three basic abstractions for representing the entities in the geographic space [3], i.e., point, line and region. These abstractions are usually considered in the two-dimensional space (see Figure II.1). A *point* is used to describe an entity for which the spatial extension has no importance in the given context. Examples of *points* are capitals on the world map, or gas stations, hospitals or similar points of interest on a road network. A *line* represents a connection between two points. Typical examples of entities that are represented as lines are rivers, water pipes or paths that traverse a forest. A region is used to depict an entity that has a 2D spatial extent. Examples of such spatial objects are lakes, forests, the boundaries of a city, countries.

Among the collections of objects that are spatially related, there are two main examples, i.e., partitions and networks. A *partition* is a collection of non-overlapping regions, where each element can have a common border with adjacent elements. For example, the administrative regions in France form a partition (see Figure II.2). A *network* is the equivalent of a spatially embedded graph. It consists of several connected lines and nodes (see Figure II.3). The networks are use to represent transportation networks (e.g., roads, highways, railroads), water supply pipes, rivers, and so on. Other examples of spatially related collection are nested partitions. These are similar to partitions (e.g., a country partitioned into regions partitioned into departments, etc.).

Figure II.2: A partition

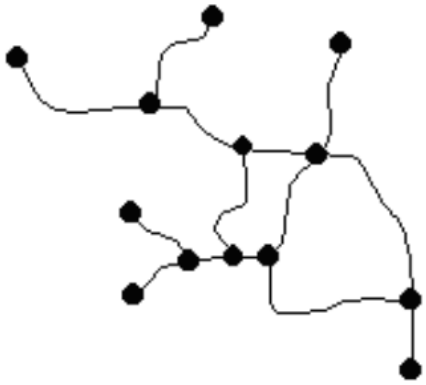


Figure II.3: A network

II.2.1.2 Spatial Operations

These fundamental abstractions are defined as data structures in the DBMS. Then, a collection of operations over the data types is defined to operate the spatial data. The data types together with the set of operations form an algebra. A reference example for the spatial database domain is the ROSE algebra [25]. More recently the ISO 19107:2003 standard came out [7]. It specifies conceptual schemas for describing the spatial characteristics of geographic features, and a set of spatial operations consistent with these schemas. It treats vector geometry and topology up to three dimensions. It defines standard spatial operations for use in access, query, management, processing, and data exchange of geographic information for spatial (geometric and topological) objects of up to three topological dimensions embedded in coordinate spaces of up to three axes.

While the collection of operation can vary from one model to another, there are some fundamental classes of spatial operations that are found in most spatial models or existing spatial DBMS extensions. We enumerate here some of these classes. The first example class contains predicates such as: **intersects**, **inside**, **touches**, **overlaps**, **disjoint**, **covers**, etc. These operations verify a specific relationship between two geometries. For example **intersects** tests whether two geometries intersect or not, **inside** verifies if the first geometry is contained by the second one, and so forth.

Another class of operations contains geometry combination functions also called set operations. In mathematics, two set of items, A and B, can be combined using different set-theory operations such as A **minus** B, A **union** B, and A **intersection** B. Similar functions are available in spatial databases that act on a pair of geometries instead of a pair of sets. Other examples in this class are functions like **common_border** (e.g., find the common border line of two adjacent region) or **touch_points** (e.g., find the isolated points of the intersection between a region and a line).

The third class that we give as example comprises geometric analysis functions also called numeric operations. These operations compute a numeric property of an individual geometry such as the **perimeter**, **length**, **area**, or **centroid**.

Some of these operations, e.g., **union** or **centroid**, are also available as aggregate functions in DBMSs that have spatial extensions. Unlike the listed spatial functions that operate on a single geometric object or on a pair of geometric objects, these *spatial aggregate* functions operate on a set of geometric objects. Like other aggregate functions in the well-known database relational model, these spatial aggregates are specified in the SELECT list of an SQL statement or in a predicate of a HAVING clause.

Besides these fundamental classes of operations, there are many other interesting and powerful operations that have been studied by the spatial database research community. We mention the operations for proximity analysis such as **within_distance** (e.g., find all data within a specified distance from a query location) and **nearest_neighbors** (e.g., find the nearest neighbors to a query location).

II.2.1.3 Integrating the Model into the Relational DBMS

The spatial types and operations are implemented into the DBMS and become available in the Data Manipulation Language (DML), e.g., SQL-like language, of the DBMS along with the existing type and operations. Let us consider the next relational schema:

```
cities (name: string, location: point)
rivers (name: string, route: line)
lakes (name: string, area: region)
forests (name: string, area: region)
```

one can formulate queries by using spatial data types operations within predicates or within the SELECT clause (in an SQL-like language).

Query 1: *"How many cities lay on the Danube river?"*

```
SELECT COUNT(*)
FROM cities AS c, rivers AS r
WHERE intersect(c.location, r.route) and r.name = 'Danube'
```

Here the predicate **intersect** verifies if the attribute location, which is a *point* type, intersects the attribute route, which is a *line* type.

Query 2: *"Are there any lakes on the border of the Rambouillet forest? If any, what is their common border with the forest?"*

```
SELECT l.name, common_border(l.area, f.area)
FROM lakes AS l, forests AS f
WHERE touches(l.area, f.area) AND f.name = 'Rambouillet'
```

The operation **common_border** computes the common intersection *line* between two adjacent regions. The adjacency between two regions is checked by the predicate **touches**.

Query 3: *"What are the cities that are completely surrounded by a forest?"*

```
SELECT c.name
FROM cities AS c, forests AS f
WHERE inside(c.location, f.area)
```

The predicate **inside** tests whether the *point* location of city is inside the *region* covered by a forest or not.

For the integration of the spatial data model to be complete, the optimization aspect must also be considered. Multidimensional access methods are required to accelerate the response time of predicates such as **intersects** or **inside**. A preferred candidate among the existing spatial access methods is the R-tree (Figure II.7). Its structure is a hierarchical organization of the Minimum Bounding Rectangles (MBRs) of the geometries that are indexed (see Figure II.6). Such access methods can also be used by other spatial operations like spatial joins. We will discuss more on spatial indexes in Chapter IV (Section IV.2). Moreover, one must provide cost functions for all methods in order to be used by the query optimizer. The cost functions are commonly based on statistics about the distribution of objects in space (needed for selectivity estimation).

Although many DBMSs today already offer spatial extensions (see the next section), this schema for DBMS extension (i.e., data types, language, and optimization) remains valid for other kinds of extensions, e.g., for spatio-temporal models, which represents one central topic of interest of this work (see Section II.4).

II.2.1.4 Oracle Spatial

Spatial databases are important for many applications as we discussed in the previous section. The demand for such systems continuously increased in the last years. This is mainly due to the to the advances in *geocoding* (mapping textual addresses to geographical locations) and proliferation of GPS devices, wireless computing, and mobile phone devices that are capable of accurately reporting their position. Applications that can take advantage of this information, e.g., traffic control, data mining, fleet monitoring, and location-aware service, are in high demand.

Most of these applications belong to the spatio-temporal application domain. While the spatio-temporal databases are today mainly a research subject (we discuss in more detail on this matter in Section II.3), the spatial database technology has matured. So now all the major

DBMS vendors (e.g., Oracle, IBM DB2, SQL Server, Informix) and even open source databases (e.g., PostgreSQL, MySQL) offer spatial extensions. This is a very important aspect especially for spatio-temporal databases, because the purely spatial part and the temporal part constitute fundamental bricks in the construction of the spatio-temporal model. Moreover, most applications require a mix of pure spatial (e.g. road network) and spatio-temporal (e.g. vehicles) features.

At the implementation level of our work, we aimed at developing a spatio-temporal database extension for moving objects with sensors. Therefore, when choosing the database environment for this extension, the two main capabilities that we search for in a DBMS were *an extensible DBMS architecture* and *developed existing spatial capabilities*. We decided to use the Oracle DBMS, which fulfills both demands. Oracle Spatial, which is the spatial extension (so called *cartridge*) of the Oracle Server, is considered the most advanced spatial database available today. In our implementation and experiments we frequently used Oracle Spatial. We will shortly present the main spatial capabilities of Oracle Spatial in the rest of this section, as we think that this will ease the reading of the second part of this document.

Oracle Spatial [10] provides a support for *spatial features* in an Oracle database. A spatial feature is characterized by a *geometric representation* that describes its shape in some coordinate space. Oracle Spatial permits the storage, update and query of such spatial features.

Oracle Spatial includes several components among which we list the following:

- A spatial schema which is added to the standard metadata. It stores additional information related to *geometric data types* to establish their storage, syntax and semantics.

- *Operators, functions, and procedures* for performing spatial queries and spatial analysis operations.

- A *spatial indexing mechanism*.

- Functions and procedures for administrative task (e.g., tuning operations).

- Other interesting components are: (i) a *network data model* for representing networks as a set of nodes and a set of links; (ii) a *topology data model* to capture the concepts of nodes, edges and faces in a topology; (iii) *GeoRaster* for querying and indexing raster image and gridded data.

Oracle Spatial supports several primitive types, and geometries composed of collections of these types (Figure II.4). Self-crossing polygons are not allowed, although self-crossing line strings are supported. A self-crossing line string does not have any implied area.

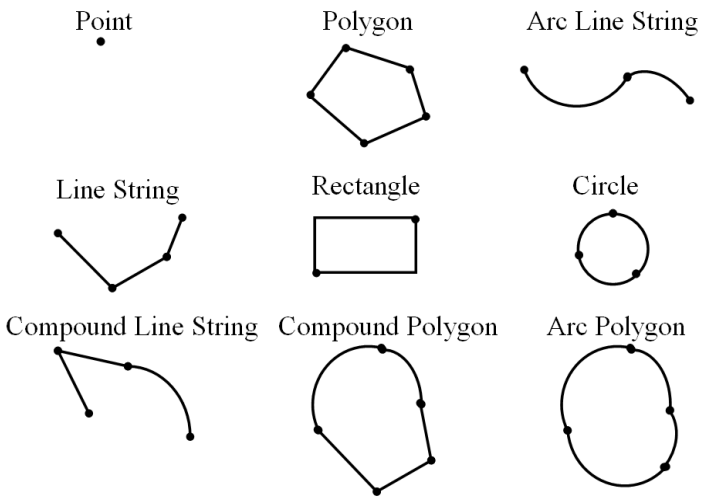


Figure II.4: Geometric types in Oracle Spatial

Spatial also supports the storage and indexing of three-dimensional geometric types. A supplementary dimension can be used as a *measure value along linear features*. This is a very useful capability of Oracle Spatial. For example, some objects (e.g., hotels, gas stations, etc.) are better identified by their position along a road instead of the latitude/longitude values. This kind of location referencing using a measure value (Figure II.5) is called *Linear Referencing System (LRS)*. In Figure II.5 each point on the polyline has three dimensions. The first two dimensions correspond to *x*, *y* coordinates, whereas the third dimension represents the relative distance measured from the start point of the geometry.

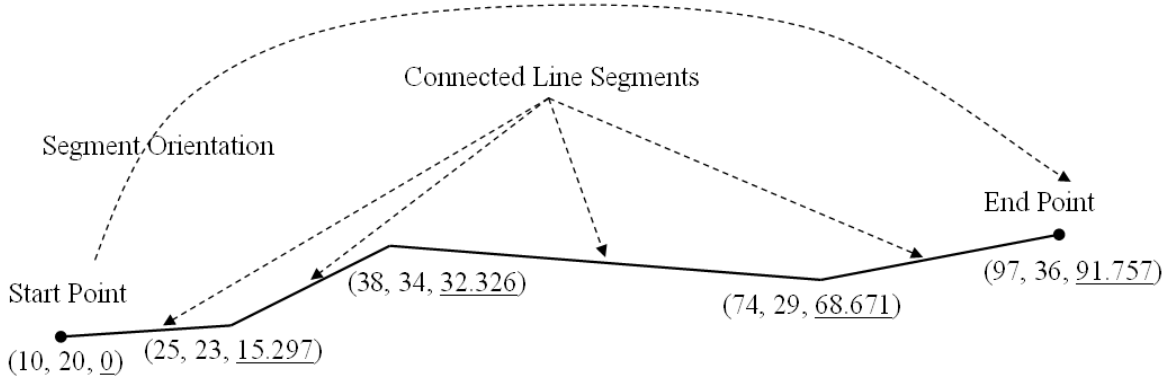


Figure II.5: Example of geometry with LRS measure

All the classes of operations that we enumerate in Section II.2.1.2 are covered by the collection of function in Oracle Spatial, including spatial join, distance, within distance, and nearest neighbors operations. Most of these spatial operations are defined as *operators* in the DBMS. The operators are a subset of the existing operations, mostly predicates, that benefit of an index based evaluation in addition to the basic function implementation. One example is the operator SDO_RELATE that evaluates topological criteria between two geometries based on the nine-intersection model proposed by Egenhofer et al. [8].

The operators use a spatial R-tree [60] index (a Quadtree can be alternatively used, but this is recommended in very few cases) that can index spatial data of up to four dimensions. An R-tree index approximates each geometry by a single rectangle that minimally encloses the geometry (called the minimum bounding rectangle, or MBR) (Figure II.6). For a layer of

geometries, an R-tree index consists of a hierarchical index on the MBRs of the geometries in the layer (Figure II.7). We present in more detail the R-tree and its variants in Chapter IV of the thesis.

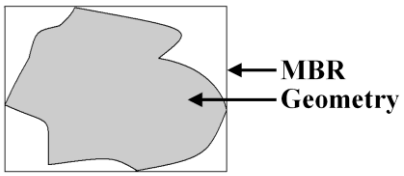


Figure II.6: Example of a MBR enclosing a geometry

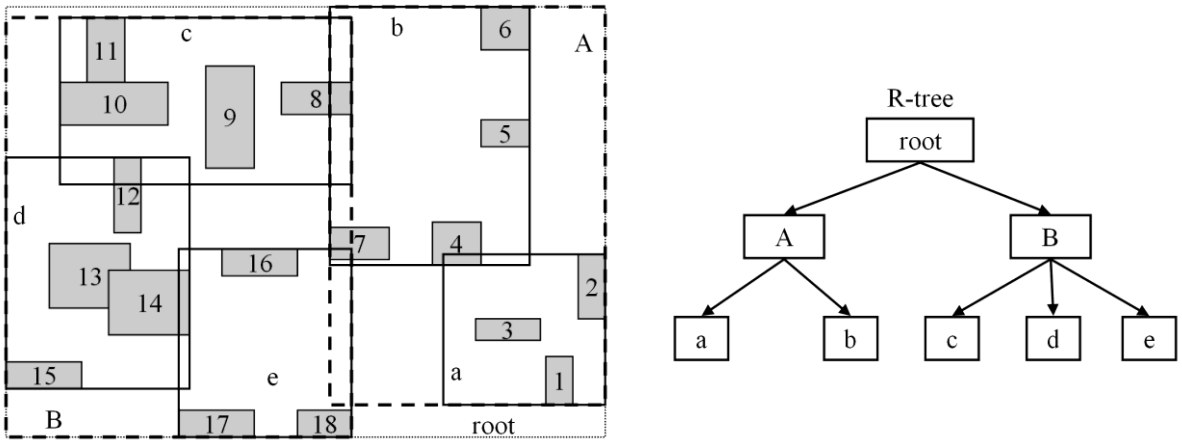


Figure II.7: Example of an R-tree hierarchical index on MBRs

There are many other features available in Oracle Spatial, which we will not discuss here. We refer to the reference book by Oracle for more details [11]. However, we can not conclude this section without evoking one important aspect: visualization of spatial objects using maps. After all, a map is certainly worth a thousand words. Visualization is complementary to spatial database extensions and is performed by tools that are working on top of the spatial DBMS. For example, Oracle includes MapViewer with his application server, which is a pure Java server-side tool that can be used to visualize maps. Other applications for visualizing spatial data exist such as ArcGIS, GeoServer, MapInfo, GeoMedia, MapServer. In our experiments we used MapViewer and GeoServer (see Chapter VI).

II.2.2 Temporal Databases

The entities that are modeled and stored in a database can change. This variation takes place in time. For many applications it is very important to track the temporal evolution of the data. This applies to banking, GIS, medical records, multimedia, and most certainly to spatio-temporal applications (e.g., transport, meteorology). A database that is capable of storing time-evolving data is called a temporal database.

However, conventional DBMSs consider that the data can only have a single state. When a modification is made in the database (e.g., through update, insert or delete operations), the data passes to a new state, while the old one is normally lost. For the standard DBMSs, a simple way to include the time dimension is by using date attributes. Nevertheless, this approach has many disadvantages e.g., it adds complexity to querying, and thus it is inefficient and error-prone.

Similar to spatial databases, an appropriate extension is needed to integrate the temporal aspects in the DBMS data model, query language and optimizer. Many temporal data models and query languages have been proposed. Their presentation falls out of the scope of this work. A good survey article can be found in [12]. Instead, we describe the basic concepts of the time domain and some of the used temporal data types. A few access methods for time intervals are presented in Chapter IV (Section IV.3).

II.2.2.1 The Time Domain

A) Structure

The work on temporal logic considers two structural models of time. That is, the time can either be *linear* or *branching*. In the linear model, time advances from past to future on a single time-line. In the branching model, time advances as in the linear model from past to "now", where it can split in several time-lines. Each resulting time-line represents a possible future and can split in turn further on.

In the temporal logic, time is generally regarded as a partially ordered set. However, several axioms can be furthermore considered, which reveal different characteristics of the time structure. For instance, the axiom that considers time as a totally order set implies that the time is linear.

Other characteristics of time are density, boundedness and relativity [12]. For linear time, there are two models of *density*: time can be *discrete* or *dense*. In the discrete model, time is isomorphic to the natural numbers. The atomic (non-decomposable) unit of time in this model is called a *chronon*. Hence, a time unit is represent as a line segment on the time line in the discrete model. The dense model considers that another time point can be found in-between any two moments of time. This model has at its turn two variants. In the first one, the time is isomorphic to the rationals, whereas in the second one it is isomorphic to the reals. This second dense model is *continuous* and has no gaps.

Another property of time is *boundedness*. Time can be bounded in the past and/or in the future. It also can be *absolute* or *relative*. For example, "10 AM, August 15, 2009" is an absolute time, whereas "15 minutes" is a relative time. The proper terms for absolute and relative time are *anchored* and *unanchored* respectively. Notice that anchored time is absolute only with respect to another time (i.e., the A.D. dating system).

B) Dimensionality

From the database perspective, there are two dimensions of time: *valid time* and *transaction time*. These dimensions are orthogonal, i.e., they are not homogeneous and have different semantics. Valid time corresponds to the time of a real world event. It is the time when a fact became true in the real world. Moreover, valid time can correspond to a future time, if we know that an event will occur in the future. Transaction time is the time interval during which a fact exists in the database. The time interval is bounded by the insertion transaction time and the deletion transaction time. Transaction time is always bounded (i.e., it is limited in the past by the database creation time and in the "future" by the present time), whereas valid time can be bounded or not.

Transaction time is used to stamp changes in the database. Temporal databases use both linear and branching transaction time. Valid time is independent of transaction time. While transaction time monotonically continues to grow, valid time can vary indeterminably.

Another kind of time encountered in the temporal databases is the *user-defined time*. For this kind of time, it is considered that only the user knows the semantics of the values and therefore, it is not interpreted by the DBMS as for the valid and transaction times.

The temporal data models can support zero or several time dimensions. A data model that does not consider any time dimensions is called a *snapshot*. A data model supporting only valid time is called the valid-time model, whereas the model that considers only the transaction time is named the transaction-time model. Finally, a data model that supports both valid and transaction times is a *bitemporal* model.

In our work, we consider time to be linear, dense and continuous, i.e., isomorphic to the real numbers. We take into account both anchored and unanchored times and include in the data model only the valid time.

II.2.2.2 Temporal Data Types

These concepts of time had been captured in a number of data types. The ODMG Standard [6] defines the following temporal data types:

- *Date*: instances of Date represent a particular day of a year. It supports the fields YEAR, MONTH and DAY.

- *Time*: instances of Time represent a particular second within the 24 hours of a day. The *Time* data type supports the fields HOUR, MINUTE and SECOND. It is possible to specify a precision, i.e., the number of decimal places of accuracy to which the SECOND field will be kept.

- *Timestamp*: instances of Timestamp represent a particular fraction of a second (usually a microsecond) of a particular day. The Timestamp data type supports the fields YEAR, MONTH, DAY, HOUR, MINUTE and SECOND. It represents unique points in time.

- *Interval*: instances of Interval represent, in a temporal context, an unanchored time interval, i.e., the interval bounds are not specified (e.g., “2 hours”).

The first three data types are also present in the SQL-92 standard. Other data types that are present in several temporal and spatio-temporal models (i.e., [4], [5], [13]) are:

- *Instant*: similar to Timestamp, the instances of Instant represent a certain point in time in the continuous model or a certain chronon in the discrete model.

- *Period*: instances of Period represent an anchored time interval, i.e., the interval limits are specified.

- *Periods*: a set of disjoint Period elements on the timeline.

II.3 Modeling Moving Objects

The research in the spatio-temporal database area started more than a decade ago, when the necessity to represent and query in a database data that are both spatial and temporal related became evident. However, managing spatio-temporal data was not a straightforward

task. As we have seen in the previous section, the work in the spatio-temporal domain was preceded by another decade of research in both temporal [1] and spatial [2] databases. This effort constitutes the basis for spatio-temporal database models.

Spatio-temporal data modeling deals with the representation in a database of any kind of moving entity. Moreover, the users should be allowed to formulate any kind of questions about such movements. However, the spatio-temporal application domain is quite vast as the type of entities that move (e.g., people, animals, vehicles, trains, forests, lakes, countries, storms, flood areas, etc.) as well as the type of changes for these entities (e.g., buses moving in a city; a forest fire or a flood area; the area closed for a certain time after a traffic accident; glaciers formed at some time, still existing or melted; etc.), are very divers.

Throughout the relatively young history of research on spatio-temporal modeling, a substantial number of models have been presented (e.g., the snapshot model, the history graph model, the spatio-temporal entity-relationship model, spatio-temporal object-oriented data models, etc.). A review of the spatio-temporal database models can be found in [14]. However, among the many existing models, there is an emergent one, i.e., the *moving object* data model [4], which gave rise to the field of moving object databases (MOD). Its main advantage is the ability to model *continuous* movement and/or changes [15]. Clearly, when we try an integration of space and time, we are dealing with geometries changing over time. In general, geometries cannot only change in discrete steps, but continuously, and then we are talking about *moving objects*. If only the position in space of an object is relevant, then *moving point* is a basic abstraction. If also the extent is of interest, then the *moving region* abstraction captures moving as well growing or shrinking regions. In this document, we only reference the moving object data model in the more general field of spatio-temporal modeling.

We concisely review in this section, the main working directions in the field of moving objects databases.

II.3.1 Current and Past Moving Objects

We distinguish two complementary approaches for modeling the applications dealing with moving objects. One approach is concerned with keeping track of a group of objects moving *right now*. The *location based* applications are the beneficiary of this type of approach. In this case, the applications require the support of *continuous or predictive* queries, which are based on the current positions of moving objects. The second approach tries to manage in a database spatio-temporal data. Here, the main concern is to integrate and query in a database the *past* states of *time-dependent geometries*. These two complementary axes of research, which have been developing in parallel for more than a decade, constitute the main elements of the moving objects database domain.

II.3.1.1 Current Moving Objects

The location based approach was introduced by Wolfson et al. in [16]. It is concerned with the current and expected near future movement of moving point objects. They propose a model named MOST (Moving Objects Spatio-Temporal) which describes the database by dynamic attributes. Opposite to traditional static attributes, the dynamic attributes vary continuously over time according to some given function even if they are not explicitly updated. The dynamic attributes are used to describe the object's *x*, *y* coordinates in the Euclidian space. They are represented by three sub-attributes: *value*, *updatetime* and *function*. The sub-attribute *value* gives the value of the dynamic attribute at the time *updatetime*. In [16]

they consider that the valid time and the transaction time are the same in the database. In short, the sub-attribute *value* corresponds to the coordinate on an axis recorded at the *updatetime*, whereas the sub-attribute *function* corresponds to the component of the velocity vector on that axis. Hence, the value of a dynamic attribute at a time *t* between two updates, is computed as *value* + *function*(*t*). Every time a query ask for a dynamic attribute, the DBMS returns the value of the attribute computed at the time the query arrived.

In this context, three types of queries are discussed, i.e., *instantaneous* query, *continuous* query and *persistent* query. An instantaneous query arrived at time t_q is "evaluated on the infinite future history beginning at t_q" [16], i.e., the query may refer to the future. However, the results are returned once, immediately after the query evaluation. An example of such queries is "find the gas stations located within 3 km of my position". A continuous query at time t_q is defined "as a sequence of instantaneous queries at each time $t \geq t_q$". Assuming that the previous query does not return any results when evaluated at time t_q, we can continue to reevaluate the query at each $t \geq t_q$, until a satisfactory answer is obtained. Finally, a persistent query at time t_q is "a sequence of instantaneous queries, all at time t_q", which are evaluated at each time $t \geq t_q$ the database is updated. The example query given in this case is "retrieve the objects whose speed on the *x*-axis doubles within 10 minutes" [16]. The particularity of this type of queries is that it requires saving information about past states of the database.

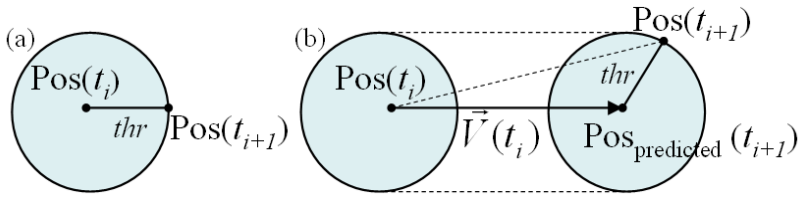


Figure II.8: Point-based (a) vs. vector-based (b) update policies with accuracy threshold *thr*

They also propose a language for the above types of queries which is called FTL (Future Temporal Logic) and which is based on temporal logic to allow specifying temporal relationships between objects. Another interesting problem that needs to be considered in the context of current moving objects is the update management. Clearly, a trade-off must be accepted between accuracy and update cost. A higher accuracy in locating the moving objects requires more frequent updating. However, frequently updating is not feasible when tracking a high number of objects. This aspect is discussed in [17]. The idea is to send with an update not only the current location of the moving object, but also the current velocity vector (see Figure II.8). This approach permits having the same location error for less frequently updates.

II.3.1.2 Past Moving Objects

The second axis of research in the field of MOD is oriented towards handling historical spatio-temporal data in a database. In this case, of main concern is the modeling and the querying of time-dependent geometries. There are two approaches that try to deal with this problem. One of the approaches is based on abstract data types, whereas the other one is based on constraints. Both approaches have been considered in the European project CHOROCHRONOS [18], which included among the many objectives the design and implementation of a spatio-temporal database system.

Gütings et al. proposed a data model for moving objects. The data model consists in a set of abstract data types (ADTs). A query language comprising a collection of operations is also defined. The main advantage of this approach is that it can be directly implemented as a database extension in virtually any DBMS. Since our proposed model for moving objects with sensors relies on these proposals [4], [5], [19], we give a thorough description of this framework in the following section, i.e. Section II.4.

The second paradigm for modeling past spatio-temporal data is based on constraints. A constraint data model represents spatio-temporal objects as infinite collections of points satisfying first-order formulae. For instance, a (time) interval can be defined as a conjunction of two order constraints that indicate the upper and lower limits. A convex polygon can be defined as the intersection of a set of half-planes. Therefore, it is represented in the constraint data model by the conjunction of the inequalities defining each half-plane.

Compared to the abstract data type approach, the constraint model has the advantage of *uniformity* in the representation of data. That is, all spatio-temporal objects benefit from the same representation (e.g., linear constraints in the first-order form), whereas a data type is required for each kind of moving object (e.g., moving real, moving point, moving region) in the abstract data type model. The model also allows a straightforward modeling of indefinite (i.e., imprecise) information. Other advantages of this approach is that it imposes no limitation in the dimensionality of the represented data and also, that the model can be integrated in standard languages of relational calculus and algebra such as SQL. However, for this integration, constraint databases require an extended relational model [20].

The infinite set of points that represent a spatio-temporal object can be described and manipulated through a *finite representation*, which is called *symbolic representation*. This is the equivalent of the *discrete model* in the abstract data type paradigm (see Section II.4.2). Hence, a spatio-temporal object is defined a finite set of linear (or polynomial) constraints. For example, the trajectory of a tourist may be represented by the constraints $(13 \le t < 14 \wedge x = 12 \wedge y = 6)$ and $(14 \le t < 18 \wedge x - y \ge 3 \wedge y \ge 5 \wedge y \le 7 \wedge x \le 13)$. The first constraint may indicate that the person is having lunch between 13 and 14 hours at a location with the coordinates (12, 6), and the second constraint may represent the visited area in the time interval 14 to 18 hours.

There are two main shortcomings of the constraint data model. A first problem is that often, the constraint representation of a spatio-temporal object has several components (as in the above example). Therefore, several tuples are required to be stored in the database for the same object. This makes querying more complicated and less natural. A second shortcoming is that constraint language is not sufficiently expressive. For example, important operations such as distance can not be expressed in the basic constraint language. These two shortcomings are addressed in [21] by Grumbach et al., which propose the DEDALE approach. Their constraint data model uses *nested attributes* for spatio-temporal data to overcome the first enunciated shortcoming. Moreover, they propose a query language having an SQL-like syntax, which hides the underlying complexity of the data model. A prototype implementation called DEDALE was also presented in [18] and [21].

Although the constraint data model appears to be very interesting, there are some important shortcomings in comparison to the abstract data type model. The constraint model does not seem to have a straightforward implementation as a database extension, since it demands modifications at the core of the relational algebra. Modern DBMSs allow users to create database extensions, but these are limited (at the modeling level) to new data types and

new operations. This makes the abstract data type paradigm a better candidate for such extensions. Moreover, the query language of the ADT model (see the following section) is, in our view, more expressive and intuitive. Finally, the approach based on ADTs offers a data model and a query language for objects moving in networks, which is of main importance to our work. Therefore, we choose to build our data model on the ADT approach, which we summarize next.

II.4 Modeling Moving Objects Trajectories

The work that we present in this document is concerned by historical data of moving objects with sensors that are constrained by the network. The most comprehensive proposal to model the historical MO is, in our view, the framework of Güting et al. [4]. Indeed, this proposal covers the abstract modeling, language and implementation [19], [22]. Moreover, it explicitly models the constrained moving objects and the relative position on the network [5]. As discussed in Chapter III, our proposal is based on this model and extends it with specific data for mobile sensors. Therefore, we summarize this model and list the used notations in the rest of this section. We begin with the abstract model [4] that introduces new data types and classes of operations for moving objects in the two-dimensional space. We continue with the network model defined in [5] and used for modeling the constrained movement of objects. We terminate this section with the discrete model [19], which is required for the implementation phase, and give a few examples of existing prototypes of moving objects databases.

II.4.1 Abstract Model

An abstract model consists in a set of *abstract data types* (ADTs) and a collection of *operations*. The data types are used to describe the entities of the modeled world, whereas the operations serve to operate over the data types. The ADTs and the operations form a so called *algebra* [4]. At the abstract level, the modeling is only concerned by the richness and the correctness of the data model. An abstract data model should be simple, but in the same time expressive. Closure is another important property that needs to be verified by the data model. The aspects regarding the implementation are not considered in the abstract model.

Güting et al. proposed in [4] an abstract model for representing and querying moving objects that move freely in the 2D space. The set of data types contains base, spatial, temporal and spatio-temporal types. The collection of operations comprises several classes of operations such as predicates, aggregations, or numerical operations. For each data type its definition domain or *carrier set* needs to be defined. For each operation, its signature, i.e., the number and possible types of the input arguments and the result types, is given. We summarize this abstract data model in the next section.

II.4.1.1 Moving Objects in 2D Space

A) Data Types.

The type system defined in [4], [5] is presented in Table II.1. Notice that we also included the *GRAPH* types (which were defined later in [5] when constrained movement was considered) for a complete view of the type system, although we properly discuss this aspect in Section II.4.1.2. The left column contains the *kinds* that describe certain subsets of types, while the right column contains the *type constructors*. Among the many types of the system, the central one are considered to be *moving(point)* and *moving(region)*. As their name indicates, these types are intended to model moving entities without spatial extension and with (possibly) variable spatial extension respectively. Nevertheless, all the listed types are

necessary in order to obtain a closed system. For instance, if we consider a *moving(point)* object, two operations that realize projections into the ranges and into the domain will return a spatial object (e.g., a *line*) or a set of time intervals respectively.

$\rightarrow BASE$	*int*, *real*, *string*, *bool*
$\rightarrow SPATIAL$	*point*, *points*, *line*, *region*
$\rightarrow GRAPH$	*gpoint*, *gline*
$\rightarrow TIME$	*instant*
$BASE \cup SPATIAL \cup GRAPH \rightarrow TEMPORAL$	*moving*, *intime*
$BASE \cup TIME \rightarrow RANGE$	*range*

Table II.1: The type system defined in [4], [5]

The types are: scalar types (*BASE*), 2D space types (*SPATIAL*), network space related types (*GRAPH*), scalar or spatial types varying in time (*TEMPORAL*), set of disjoint intervals (*RANGE*). Examples of types are: *real*, *point* (2D position), *gpoint* (position on the network), *gline* (line on the network), *moving(point)* (2D position varying in time) and *moving(gpoint)* (network position varying in time). All base types have the usual interpretation. For example, if we note with A_α the carrier set (definition domain) for the type α, then for the *real* type the carrier set is: $A_{real} = R \cup \{\perp\}$, where $\{\perp\}$ is null (or undefined). The time is isomorphic to the real numbers. The *range* data types are disjoint intervals and are used to make projections or selections on the *moving* types. Spatial types describe entities in the Euclidean space, while for the *GRAPH* types the space is represented by a network space (see Section II.4.1.2). 2D types mainly correspond to standard definitions [7] (see Figure II.1).

The fundamental types of this algebra are the temporal types. These are derived from the base types, spatial types and graph types by using the type constructor *moving*. For a given type α the moving constructor does a mapping from time to α. Figure II.9 depicts some examples of moving types. The precise definitions as presented in [4] are given next:

Definition II.1: Let α be a data type to which the *moving* type constructor is applicable, with carrier set A_α. Then the carrier set for $moving(\alpha)$, is defined as follows:

$A_{moving(\alpha)} = \{ f \mid f : \overline{A}_{instant} \rightarrow \overline{A}_\alpha$ is a partial function $\wedge\ \Gamma(f)$ is finite $\}$ [4].

Definition II.2: Let α be a data type to which the *intime* type constructor is applicable, with carrier set A_α. Then the carrier set for $intime(\alpha)$, is defined as follows:

$A_{intime(\alpha)} = A_{instant} \times A_\alpha$ [4].

The Definition II.1 states that a $moving(\alpha)$ type object is a partial function from time to α. This means that a $moving(\alpha)$ type object is an infinite set of pairs (instant, α). $\Gamma(f)$ represents the number of continuous components of f. The condition imposed to $\Gamma(f)$ to be finite is needed to ensure that the projections over f will yield a finite number of components. Moreover, this condition is required in order to obtain a data model that is implementable.

Several operations, e.g., time-slice operations, over a *moving*(α) can return a single pair (instant, α). For practical reasons, the authors include in the type system a dedicated type for such cases. This type is called *intime* (cf. Definition II.2).

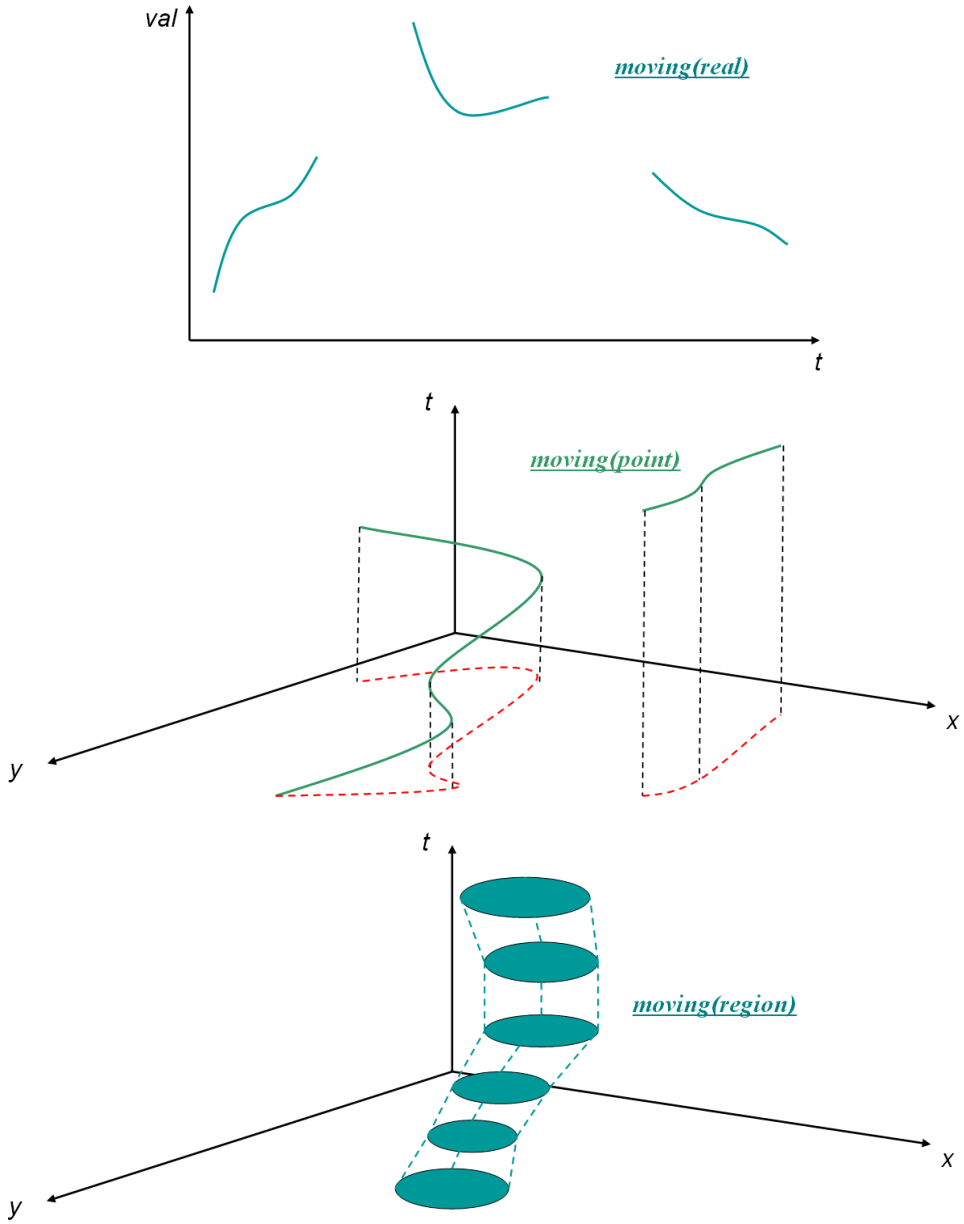


Figure II.9: Examples of moving types

B) Operations

The second step in defining an algebra is to design a collection of operations over the types of the type system. The design of the operations adheres to three principles [4]: "(i) Design operations as generic as possible. (ii) Achieve consistency between operations on non-temporal and temporal types. (iii) Capture the interesting phenomena".

Designing operations for a framework that is intended to be representative for spatio-temporal databases is difficult, mainly due to its generality. The purpose of the three principles is to guide this process. The first principle suggests that operations should not be inspired of one application domain or another. This could lead to a proliferation of operations.

This can also be avoided by using the same operation on non-temporal and temporal types (cf. to the second principle). This is done by a process called *lifting*, where the same operation over non-temporal types is extended to a signature that includes temporal types (see the next paragraph). Besides, specific temporal operations need to be included in the system. Nevertheless, the collection of operations is not determined as opposed to the type system, since there is no clear way of how to achieve closure in this case. One could include new operations if necessary (e.g., implied by certain spatio-temporal applications).

Predicates	**isempty** $=, \neq$**, intersects, inside** $<, \leq, >, \geq$**, before** **touches, attached, overlaps, on_border, in_interior**
Set Operations	**intersection, union, minus** **crossings, touch_points, common_border**
Aggregation	**min, max, avg, center, single**
Numeric	**no_components, size, perimeter, duration, length, area**

Table II.2: Example of classes of operations on non-temporal types [4]

Projection to Domain/Range	**deftime, rangevalues, locations, trajectory** **routes, traversed, inst, val**
Interaction with Domain/Range	**atinstant, atperiods, initial, final, present** **at, atmin, atmax, passes**
Lifting	(all new operations inferred)

Table II.3: Example of classes of operations on temporal types [4]

The first classes of considered operations are the operations over the non-temporal types. They are listed in Table II.2. We already covered the semantics and the importance of most of these operations in Section II.2.1.2. Therefore, we will not insist on this aspect here. We find more interesting the process, i.e., the so called (temporal) *lifting*, by which these operations are extended to temporal types. The temporal lifting permits generating from a non-temporal operation with the signature $\alpha_1 \times \alpha_2 \times ... \times \alpha_n \rightarrow \beta$, the temporal equivalent one having the signature $\alpha_1' \times \alpha_2' \times ... \times \alpha_n' \rightarrow moving(\beta)$, where $\alpha_i' \in \{\alpha_i, moving(\alpha_i)\}$. Each of the arguments can become temporal, which makes the result temporal as well. The operations that result from lifting are given the same name as the operation they originate from. For example, the **inside** predicate with signature

$$point \times region \rightarrow bool$$

is lifted to the signatures

$$moving(point) \times region \rightarrow moving(bool)$$
$$point \times moving(region) \rightarrow moving(bool)$$
$$moving(point) \times moving(region) \rightarrow moving(bool).$$

Finally, specific classes of operations are added to deal with temporal types (Table II.3). Notice that they include the lifted non-temporal operations. The temporal operations operate

on *moving* types, which are in fact partial functions. Therefore, the basic classes of temporal operations are represented by projections into domain and range, and by interactions with values from domain and range. We present some of these operations in the remainder of this section.

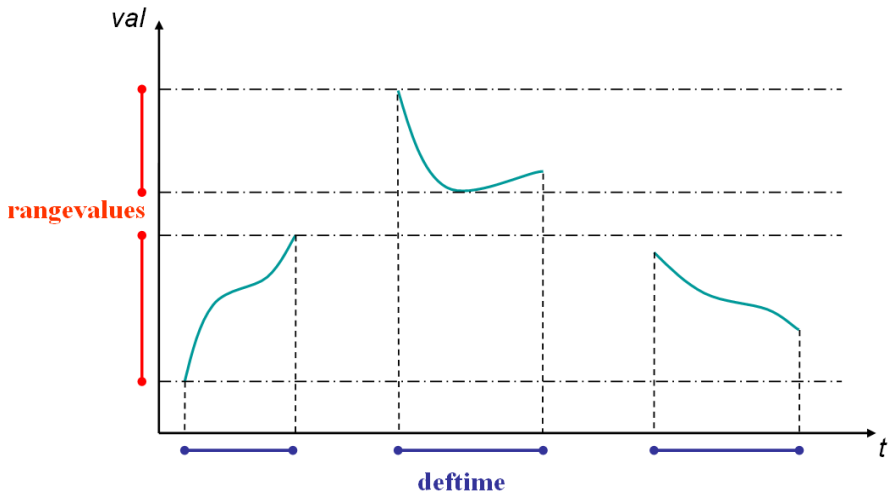


Figure II.10: Example of deftime and rangevalues projections over a *moving(real)*

Let us consider first the operations that yield the domain and range of the *moving* types. The domain function **deftime** returns the times for which a function is defined (Figure II.10). The operation **rangevalues** returns values assumed over time as a set of intervals (Figure II.10). The projection in space (i.e., the *line*) of a continuously moving object is obtained by the operation **trajectory**. For values of *intime* types, the two trivial projection operations **inst** and **val** are offered, yielding the two components.

Consider the below example relations for illustration of operations on temporal types:

```
trips(id: int, route: moving(point))
departments(name: string, area: region)
polluted_zones(name: string, area: moving(region))
```

Some simple query examples that use these operations and their formulation in an SQL-like language are given next:

Query 1: *"What are the departure and arrival times of the trip having the id 5?"*

```
SELECT min(deftime(route)), max(deftime(route))
FROM trips
WHERE id = 5
```

Query 2: *"What is the length of the route part that lies within the Yvelines department for a given trip?"*

```
SELECT length(intersection(trajectory(route), area))
FROM trips, departments
WHERE id = 5 AND name = 'Yvelines'
```

One observation regarding the projection operations is that they eliminate one of the two "dimensions" of the spatio-temporal objects (i.e., spatial or base value + time). Thus, the results are either (temporal) intervals or geometries on which we can further operate by using non-temporal operations (like in the Query 1 example). We can clearly see now the importance of the purely spatial and temporal parts in the foundation of a spatio-temporal model.

The second class comprises the operations that relate the functional values of moving types with values either in their (time) domain or their range. These operations are useful to make selections or clippings on the moving entities in accordance with certain criteria. For instance, we would like to clip a moving real to the times its value was 3 or it was between 5 and 8. Or, we would like to select the part of a moving point when its position was inside a given area. Or, one might be interested to determine the value of the moving real within a time interval $[t_1;t_2]$.

Figure II.11 gives examples of interactions with time domain values, while Figure II.12 presents examples of interactions with range values. Examples of operations that realize restrictions in the time domain are **atinstant**, **initial**, **final** and **atperiods**. The function **atinstant** is a time-slice operation, i.e., it restricts a *moving*(α) to a given time instant, which returns an *intime* object, i.e., a pair (instant, α). Similarly, **initial** and **final** are time-slice operations, which return the first *intime* and last *intime* of a temporal object. Operation **atperiods** restricts a *moving*(α) at a set of disjoint periods of time. The predicate present is also proposed to verify if a *moving*(α) is defined at a certain time instant or set of time periods.

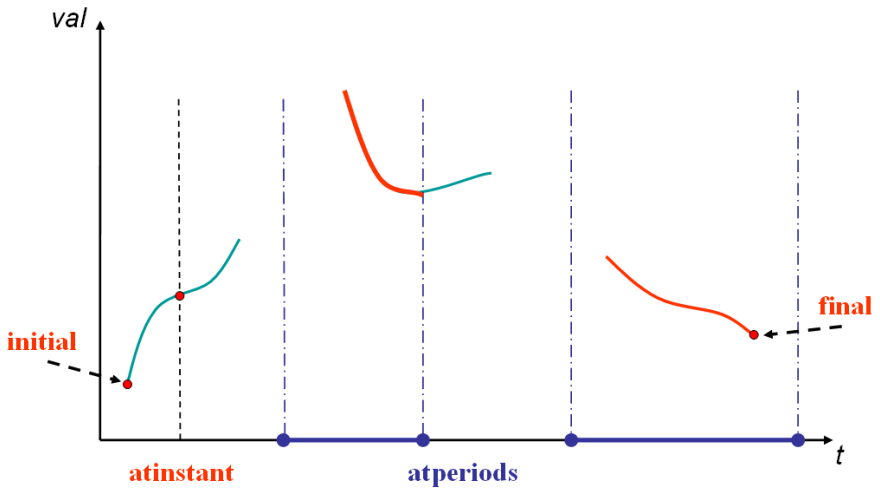


Figure II.11: Examples of interactions with time domain values

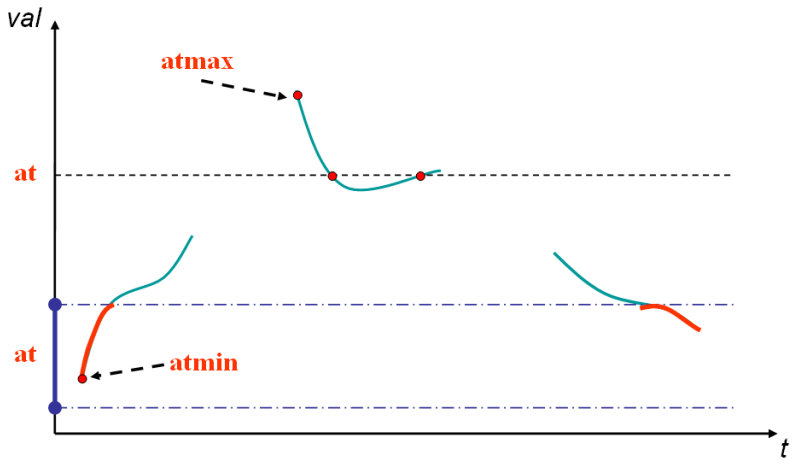


Figure II.12: Examples of interactions with range values

Orthogonally, different operations that realize restrictions to values in the range are proposed, such as **at**, **atmin** and **atmax**. For instance, one can clip a moving real to the times when its value was between 5 and 9 by using the operation **at**. The operations **atmin** and **atmax** reduce a moving value to the times when it was minimal or maximal respectively. Finally, to verify if a moving value ever assumes certain indicated values, the predicate **passes** should be used.

Some simple query examples that use these operations are:

Query 3: *"When and where did the trip with the id 5 leave the Yvelines department?"*

```
SELECT inst(final(at(route, area))),
       val(final(at(route, area)))
FROM trips, departments
WHERE id = 5 AND name = 'Yvelines'
```

Query 4: *"For which periods of time was the trip with the id 5 within the pollution area of 'Lille'?"*

```
SELECT deftime(at(route, area))
FROM trips, polluted_zones
WHERE id = 5 AND name = 'Lille'
```

II.4.1.2 Moving Objects in Networks

In many cases, objects of the real world do not move freely in the two-dimensional space, but rather within spatially embedded networks. For example, vehicles travel on highways or on roads networks, trains and subways move on railways networks, and even airlines have determined routes. It would be beneficial to consider the network space instead of the 2D space in the case of constrained movement for several reasons. The first is that the 2D model does not capture the relationship between the trajectory and the network space, while this information is essential for analysis. Querying relatively to a network space is more suited

(e.g., what are the gas stations on the A6 highway) for constrained moving objects. The second is that it limits the representation of the trajectory, estimated by linear interpolation between the reported positions, while the MO follows in fact the geometry of the network. In addition, the constrained model allows for dimensionality reduction by transforming the network in a 1D space by juxtaposing of all line segments [101] (see Chapter IV). This leads to better storage and query performance than with the free trajectory model.

The *GRAPH* types (Table II.1) in the above type system were added in [5], when the initial framework was extended for moving objects in networks. It includes two network space related types, i.e., *gpoint* (position in the network space) and *gline* (line in the network space). Of course, these types can become temporal by applying the *moving* constructor. We then obtain *moving(gpoint)* (network position varying in time) and *moving(gline)* (line in the network varying in time). The main novelty here [5] is the proposed network model and its integration in the comprehensive framework for modeling and querying moving objects [4]. We present this network model in the rest of the section.

The real transportation networks are complex. Thus, it is difficult to have a universal network model that is completely independent of an application domain. The proposed network models in the literature, that we are aware of, take into account the targeted application domain that will make use of the model (e.g., moving objects databases, multimodal transportation systems).

The network model proposed in [5] considers the case of constrained moving objects in road networks. In short, a road network is composed by a number of routes and the junctions between them. To localize an object in the network, its relative position on a certain route is given. This is similar to the concept of linear referencing (see Figure II.5) widely use in GIS. There are several reasons to model the road network in term of routes, i.e., paths in a graph instead of graph edges. The roads in a real network extend over the junctions. Named streets do not begin nor end between two consecutive road junctions. Moreover, addresses are given relatively to the streets or by indicating the distance (e.g., in kilometers or miles) from the starting point of a route. Also, using longer routes has a positive impact on the representation of a moving object, which may become smaller, since it is not necessary to change the description every time the object passes a junction.

Another aspect included in this network model is that road traffic can be bidirectional. Therefore, for certain routes we can distinguish between *up* and *down* travel directions. These routes are named *dual* routes. Those that have a single possible travel direction are called *simple* routes. Moreover, for a given junction and its the routes, all the allowed traversals of the junction are modeled as a *connectivity code*.

Formally, "a *network* is defined as a pair $N = (R, J)$ where R is a finite set of distinct routes and J is a finite set of junctions in R" [5].

The definition domain of routes is:

$$\begin{aligned} Route = \{(id, l, c, kind, start) \mid\ & id \in \underline{int},\ l \in \underline{real},\ c \in \underline{line}, \\ & kind \in \{simple, dual\}, \\ & start \in \{smaller, larger\}\} \text{ [5].} \end{aligned}$$

A route has several attributes: the *id* is a route identifier, *l* gives the length of the route, *c* represents its geometry, *kind* indicates if bidirectional movements are allowed (i.e., the route

is *dual*) or not (i.e., the route is *simple*), *start* indicates how the route distances are measured (i.e., relatively to the start point or the end point of the route).

A location in the network space is defined as a *route location*:

$$RLoc(R) = \{(rid, d, side) \mid (rid, d) \in RMeas(R), side \in \{up, down, none\},$$
$$\text{for } (rid, l, c, kind, start) \in R : kind = simple \Leftrightarrow side = none\} \text{ [5].}$$

In order to locate an entity in the network, one should specify the route identifier *rid*, the relative distance *d* measured from one of the two ends of the route, and the *side* of the route. Notice that the side attribute is of importance only for dual routes. For simple routes the side has always the value *none*.

Figure II.13 gives a simple network example. The small rounds represent route locations, whereas the squares depict junctions. Notice that dual routes are represented by double polylines. The red portion indicates a network region, which is composed by a set of route intervals.

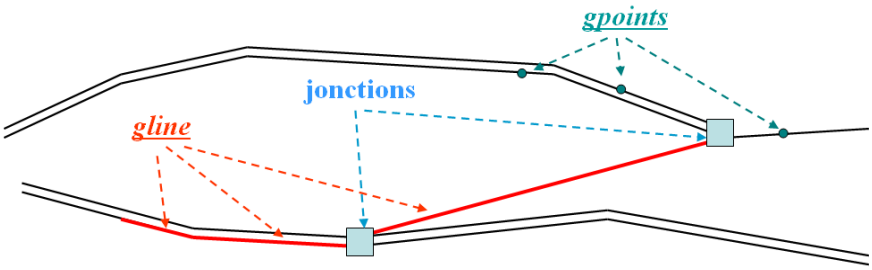


Figure II.13: A simple network example

Given the defined network model, the initially 2D framework is readily extended to constrained movement. First, new network related types are defined. A *gpoint* is defined as *route location* and a *gline* as set of disjoint *route intervals* (a route interval is a route section between two route locations). Then, the collection of operations described in the previous section, is extended to the network types. For example, the operation **trajectory** over a *moving(gpoint)* will return a *gline*, which is the in-network projection of a constrained moving object trajectory. The semantic of some operations, such as **distance**, needs to be adapted for network types (i.e., distance by route). Finally, new classes of operations are added to analyze the interaction between the network and the 2D space, as well as specific operations such as computing shortest path in the network. One can refer to [4] and [5] for more details. Recently, other more complex network models have been proposed. These models deal with different problems in transportation networks such as multimodal transportation systems [23] or location-based services [24].

II.4.2 Discrete Model

We presented in the previous section a comprehensive framework for moving objects that move freely in the two-dimensional space or that are constrained by a network. The framework consists of data types and operations that encapsulate the data types, together forming an algebra. The main question to be considered when designing such algebras is "exactly which types and operations should be offered?" This aspect was dealt with in the above section. However, there is another important question that arises and need to be

answered in order to have a complete design. This question is “at what level of abstraction should these types and operations be described?” [19]

The different entities that we encountered in the moving objects database area such as a line, a region, a moving real or a moving point can be represented at two level of abstraction. For instance, a line can be viewed as a set of curves in the plane or as a set of polylines (see Figure II.14). A moving point in the 2D space can be defined as an infinite set of pairs (instant, x-coordinate, y-coordinate) or as a polyline of line segments in a three-dimensional space (2D + time) (see Figure II.15). The first type of representation is given in terms of *infinite sets*, whereas the second one is a *finite* one. The introduced terms for these two levels of representation are *abstract model* [4] and *discrete model* [19] respectively. Thus, the discrete representation is an approximation of the abstract representation.

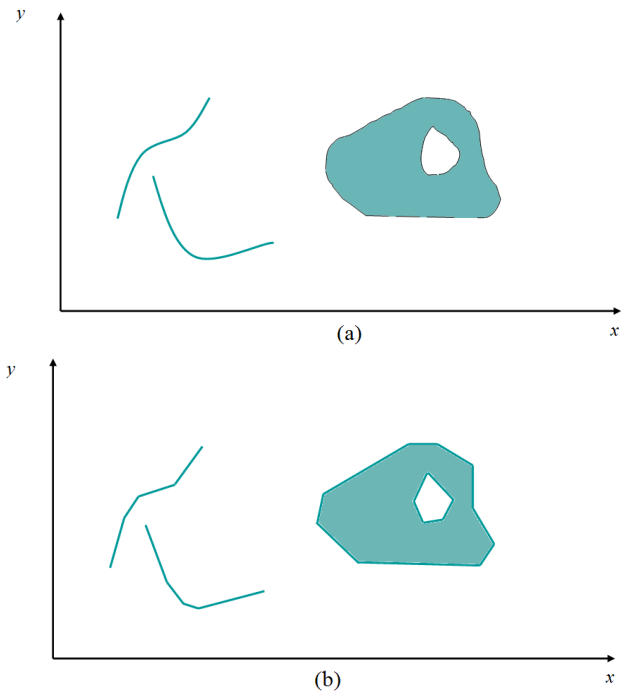


Figure II.14: (a) *line* and *region* values of the abstract model (b) *line* and *region* values of the discrete model

The authors in [19] argue that both the abstract and the discrete modeling are needed when designing a novel data model. In a first step, one should conceive an abstract model, as this one is not concerned with any details regarding the implementation. However, an abstract model does not have a straightforward implementation. Hence, the second step will be to choose a discrete representation for the abstract model. Moreover, there are numerous possible discrete representations that can be employed when passing from the abstract model to the discrete model. For example, moving real values can be modeled either by linear functions or by Chebyshev polynomials. All these representations are included in the abstract model, but only one can be chosen for a discrete model.

In this section we present the discrete data model implementing the presented abstract data model [4], [5]. Forlizzi et al. introduced in [19] corresponding “discrete” types (i.e., their domains are defined in terms of finite representation) for all the abstract data types.

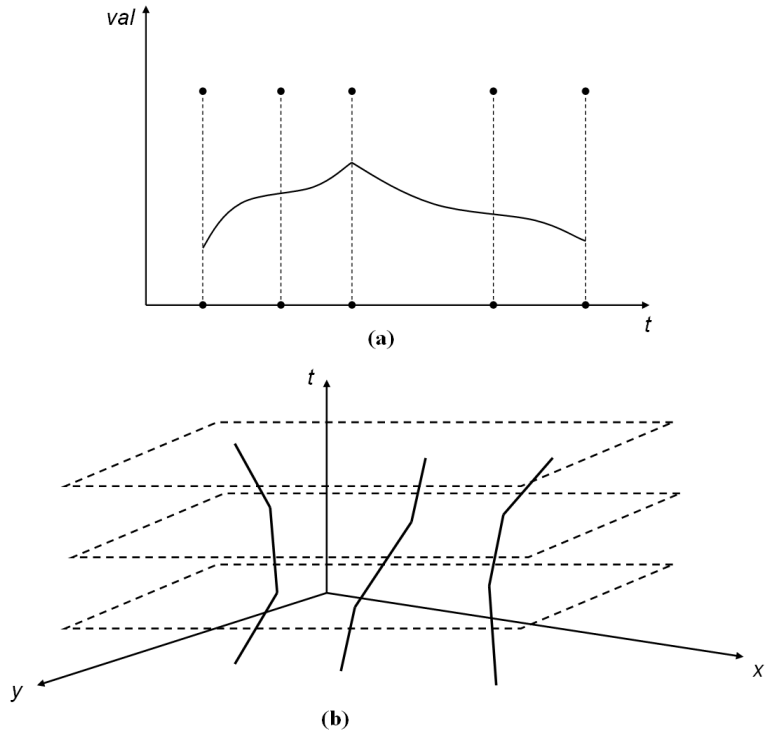


Figure II.15: Sliced representation of *moving(real)* and *moving(points)* value

The discrete model is based on the so called *sliced representation*. The idea is to decompose each modeled entity into a finite number of parts and then describe each part by means of a "simple function". Figure II.15 presents the sliced representation for a moving real and a moving point. The temporal development of these entities is decomposed in several disjoint time intervals. Within each interval the evolution of the moving real or of the moving point is given by a function. For instance, second degree polynomials can be used to model moving real values and linear functions for moving point. Choosing between such functions is a trade-off between richness of representation and computational cost of the operations that encapsulate the data types. That is, complex functions can better represent the temporal variation of the data, but increase the complexity of the operations.

The slices that compose an object type are called *units*. Unit types are defined in [19] for all moving types. Here we give the definitions of the unit types for *moving(real)* and *moving(point)*, i.e., *ureal* and *upoint*, which are the most related to our proposed model (see Chapter III). We also include the definition of an interval, since it is used in the definitions of the other unite types. One can refer to [19] for the unit definitions for *moving(line)* and *moving(region)*.

"Let (S, <) be a set with a total order. The representation of an interval over S is given by the following definition." [19]

$$Interval(S) = \{(s, e, lc, rc) \mid s, e \in S, lc, rc \in bool, s \leq e, (s = e) \Rightarrow (lc = rc = true)\}$$

[19].

An interval is defined as a tuple of four values. The first two, i.e., s and e, represent the lower (start) and the upper (end) limits of the interval. The last two are booleans that indicate if the interval is left-closed (i.e., lc) and/or right-closed (i.e., rc).

The carrier set for the type *ureal* (i.e., the unit of a *moving(real)*) is

$$D_{ureal} = Interval(Instant) \times \{(a, b, c, r) \mid a, b, c \in real, r \in bool\}$$ [19]

and the evaluation at time instant *t* is defined by:

$$v((a,b,c,r),t) = \begin{cases} at^2 + bt + c & if\ \neg r \\ \sqrt{at^2 + bt + c} & if\ r \end{cases}$$ [19].

The units that compose a *moving(real)* are defined by a time interval and function within this interval. In [8] they choose second degree polynomials and square root of the second degree polynomials as functions. Second degree polynomials offer a good trade-off between the richness of representation and the computational cost. The square root of such polynomials can be used to model the temporal variation of the distance between two moving objects in the Euclidian space.

The structure of units for *moving(point)* types is similar, i.e., a temporal interval and a simple function within the interval. In this case, they choose linear function to represent the movement of a moving point. Operations over this type of objects can be quite complex. Hence, linear functions seem to offer an interesting trade-off for this type of objects. Nevertheless, more complex functions like polynomials of a degree higher than one are conceivable. Formally, the carrier set of *upoint* (i.e., the unit of a *moving(point)*) is defined as:

$$D_{upoint} = Interval(Instant) \times \{(x_0, x_1, y_0, y_1) \mid x_0, x_1, y_0, y_1 \in real\}$$ [19]

and the evaluation at time instant *t* is defined by:

$$v((x_0, x_1, y_0, y_1), t) = (x_0 + x_1 \cdot t, y_0 + y_1 \cdot t) \quad \forall t \in Instant$$ [19].

This describes the temporal evolution of 2D points. There is no formal definition in [5] for the carrier set of *ugpoint* (i.e., the unit of a *moving(gpoint)*). In [5] they informally define the discrete data types as data records. However, using the above notations, we can define the the carrier set of *ugpoint* as:

$$D_{ugpoint} = Interval(Instant) \times \{(nid, rid, a, b, side) \mid nid, rid \in int, a, b \in real, side \in \{up, down, none\}\}$$

where *nid* is a network identifier, *rid* is a route identifier, *side* gives the side on the road, and *a* and *b* are the coefficients of a linear function that evaluates the relative position on the route at time instant *t*, i.e., the relative position *pos* is computed as *at + b*.

These discrete representations offer a precise basis for the implementation of a spatio-temporal database extension that can be added to a suitable extensible architecture (e.g., as a cartridge in Oracle Server or as a data blade in Informix Universal Server). Algorithms for a large subset of the operations have been presented in [22].

II.4.2.1 Existing MOD Systems

The research area of moving objects databases (MOD) has flourished in the last decade. There has already been a lot of research in the last years ranging from data models and query

languages to implementation aspects such as efficient index structures. However, there are not many active prototype MOD systems that we are aware of.

The first system that implemented a moving objects data model and query language completely integrated into a DBMS environment was SECONDO [28]. SECONDO is a prototyping environment that was designed for database extensions. The extensibility mechanism is based on algebras, which can be implemented as new modules in the system serving for non-standard applications. The system's extensibility covers all aspects that are required by a novel application such as optimizations of the query processing or extensions of the user interface. The complete SECONDO system, including the moving objects algebra, is freely available for download [31].

An algebra corresponds to a data model, i.e., a set of data types encapsulated by a collection of operations. Several algebras are already implemented in SECONDO. For example, there is an algebra with relations and tuples as data types and operations like projection or hashjoin. Index structures such as B-trees and R-trees are also offered as algebras. The *Spatial* algebra implements the types *point*, *points*, *line*, and *region* following the implementation of the ROSE Algebra [25]. The *Temporal* algebra mainly implements the temporal data types and operations, e.g., *moving(point)* and *moving(region)*, introduced in [4] and [5]. Recently, demonstrations of spatiotemporal pattern queries in SECONDO [29] and nearest neighbor search on moving object trajectories in SECONDO [30] have been presented.

Another example of MOD system is HERMES Moving Data Cartridge (HERMES-MDC) [13], [26], [27]. HERMES is implemented as a data cartridge in the Oracle object-relational DBMS. The main goal of HERMES is to support modeling and querying of continuously moving objects. Taking advantage of the extensibility interfaces provided by modern ORDBMS, HERMES DB engine is developed as a system extension that provides trajectory functionality to Oracle ORDBMS. More specifically, HERMES defines a trajectory data type and a collection of operations (e.g., nearest neighbor searching, similar trajectory searching) as an Oracle data cartridge, which is further enhanced by a special trajectory preserving access method, namely the TB-tree (see Chapter IV Section IV.4.2).

HERMES provides support for Location Based Services (LBS) applications. A user can pose several types of requests in form of spatio-temporal range queries and time-slice queries, or trajectory similarity queries. Moreover, the users can build services on top of HERMES to serve the specific needs of their spatio-temporal applications. To this end, HERMES proposes a three-tier architectural framework. The first tier is represented by the ORDBMS extended with a data cartridge for trajectory data storage and spatio-temporal query capabilities, as described above. The second tier is the application tier that acts as a middleware. New functionalities can be implemented here by using JSP/Java code and published as JSP pages, wsdl documents, jars, etc. For example, Oracle MapViewer is used at this level to enable data visualization. The final tier is represented by the client application. Such applications enable the client to send http request and receive corresponding responses from the server.

We also mention a few examples of prototype implementations of spatio-temporal database system that were presented in the European project CHOROCHRONOS [18] such as Concert, Dedale, Tiger and GeoToolKit.

II.5 Conclusions

This chapter presents the related work in the general field of moving objects databases (MOD), with an emphasis on modeling and querying the (historic) trajectories of moving objects. We showed that spatio-temporal databases find their origins in spatial and temporal databases. Moreover, a spatio-temporal system includes a spatial component and a temporal component as foundational bricks. Therefore, we began this section by presenting the fundamental concepts in spatial database systems and temporal database systems. Nowadays, these two fields are mature, one strong evidence being the fact that many commercial or open source database systems include spatial and/or temporal models integrated in their type system and query language.

In the second part of this chapter, we discussed about the spatio-temporal modeling. The management of MOD has received particular attention in the recent years because of the advances and the omnipresence of mobile and geo-location technologies, such as cellular phones or GPS. Güting's book is a summary of progresses in this area [3]. There are two complementary perspectives of this field. One deals with the problem of managing real-time positions of a collection of moving objects. The other one tries to manage the complete trajectories of moving objects. Our work falls in the second perspective. Therefore, we detailed the related work for modeling moving object trajectories in the second part of this chapter. The most comprehensive proposal to model the historical MO is, in our view, the framework of Güting et al. [4]. Indeed, this proposal covers the abstract modeling, language and implementation [19]. Moreover, it explicitly models the constrained moving objects and the relative position on the network [5]. As discussed in the next chapter, our proposal is based on this model and extends it with specific data for mobile sensors. Therefore, we summarized first this model and listed the used notations.

Chapter III: Modeling and Querying Mobile Location Sensor Data

Moving objects databases are an important research topic in recent years. As shown in the previous chapter, a lot of work dealt with modeling, querying and indexing objects that move freely or in networks. However, a moving object – such as a vehicle - could report some measures related to its state or to its environment, which are sensed throughout his movement. Managing such data is of major interest for some applications such as analyzing driving behavior or reconstructing the circumstances of an accident in road safety, or identifying, by means of a vibration sensor, the defects along a railway in maintenance. However, this management is not covered by the existing approaches. In this chapter, we propose a new data model and a language that handle mobile location sensor data. At this end, we introduce the concept of measure profile to capture the measure variability in space, along with specific operations that permit to analyze the data. We also describe their implementation using the object-relational paradigm.

III.1 Introduction

Integrating mobile technology and positioning devices (GPS, cell phone) has led to produce a large amount of moving object data every day. A wide range of applications like public transportation, traffic management, location-based services, relies on these data. Besides, a moving object can easily be equipped with sensor devices that report on its state or on its environment. The generated mobile location sensor data are meaningful for many applications such as reconstructing the circumstances of an accident in road safety, identifying defects from vibration sensors along a railway in maintenance, or analyzing the exposure to hazard (e.g. pollutant) along a trip. As an example, in the field of road safety, the observation of natural driving behavior (on normal route for usual journeys) tends to replace the tests on simulators or those limited to dedicated circuits. Known as "naturalistic driving", these studies are based on data collected on a large scale and over a significant period of time [34].

However, studies reported in the literature have limited the volume of data and the possibilities of their exploitation. Thus, in a naturalistic driving campaign carried out on 100 cars, undertaken by the administration of U.S. Highway Safety [32], the storage is limited to situations of near-accident. As emphasized in their report [32], a large-scale database would be very useful to researchers and engineers to study the driving behavior and contribute to improving the vehicle equipment and road planning.

However, the challenge of a large-scale study is the management and analysis of a large mass of spatio-temporal data. A database system that supports this type of data and efficient querying is needed. We aim to study and develop such a database management system. This subject closely joins the field of databases for moving objects. However, the moving object - here the vehicle - is associated with additional data describing the state of the object or measures (speed, acceleration, steering wheel angle, the action on the breaking or gas pedals, etc.) recorded throughout his trip. These measures are variable in space and time.

For the type of applications we address, the measures are more important than the mere spatio-temporal location. However, most work on mobile object databases consider only the

location of the moving object and can not be generalized to measures ranging along a spatio-temporal trajectory. Moreover, although these values initially correspond to a temporal data stream, their variation is more dependent on their location on the network than time: for example, the variation of speed is usually constrained by the geometry of the road and the speed limit. Also, the temporal analysis of data of different trajectories is irrelevant because on the one hand they are asynchronous and on the other hand, this comparison makes sense only if these paths overlap in space. Therefore, we must capture the spatial variability of these measures and allow its manipulation through the data model and query language.

To our knowledge there is no such proposal in the state of the art. Nevertheless, among the works on moving objects databases (MOD), the one proposed in [4], [5] (see Section II.4) provides a solid basis for modeling and querying moving objects. The idea of representing in a continuous way the temporal variation of the location or scalar values permits a good abstraction of moving objects. We extend this approach to capture the continuous spatial variation of scalars. The extension of the existing algebra consists in new types and several classes of operations. The types capture the variation in space of any measure, which includes data from mobile sensors as a particular case. New operations are needed to operate on the measures.

This chapter provides the following contributions. First, we present a new concept of "space variant measure" and show its usefulness in the context of a naturalistic driving study. Second, we create a data model as an extension of the one proposed by Güting et al. [4], [5]. Finally, we extend the existing query language with new classes of operations that are necessary in this novel application context. Besides the aforementioned application, the proposed model is interesting for other applications that generate and/or operate geo-localized data streams. This is the case of rail (the TGV in France has many sensors), air or sea transportation. The measures can be observed or calculated and may be related to a trajectory of an object or a location. Thus, we can reason about the speed of a moving object, on the legal speed or inclination of a road or on the adherence at each location of the road depending on the weather. We can model the fine data on the mobility (where, when and at what speed) of vehicles (equipped with GPS), freight (labeled by RFID), or persons (GSM / GPRS) and meet the needs of management applications for fleets or road traffic, logistics and design of mobile networks.

The rest of this chapter is organized as follows: we motivate our approach by presenting a real application and a query scenario in Section III.2. We summarize the related work in Section III.3. Section III.4 describes the proposed model. It presents the new type and the corresponding operations, and demonstrates its use by expressing some query examples. Section III.5 discusses several aspects of the implementation. Finally, Section III.6 concludes and offers directions for future work.

III.2 Motivation and Examples

In this section, we present a real application that has motivated our work. A query scenario is used as an example throughout this chapter.

As indicated in the introduction, naturalistic driving studies have become more popular within the last years. These studies are essentially based on data gathered in normal (natural) driving conditions. They become economically possible thanks to the existing equipment in modern vehicles. Indeed, the large number of in-vehicle sensors is accessible via an interface

(CAN bus) on which it is possible to connect an in-vehicle data logger. The CAN bus provides access to several measures including speed, acceleration, steering wheel angle, the action on the breaking or gas pedals, etc. The recording device can also receive data streams from other sources, such as a GPS sensor or radar (giving the distance to adjacent vehicles). This provides a comprehensive database on natural driving on the road. This database can provide valuable information on the use and utility of driver assistance systems (ABS, ESP, etc.) and can highlight near-accident (near-crash) situations. Moreover, according to the principle of black boxes on airplanes, it will provide information prior to an accident.

Figure III.1: DIRCO data logger

INRETS (French acronym for "National Institute for Research on Transport and Safety") has developed a data logger (called DIRCO) (Figure III.1) for naturalistic driving campaigns [35]. This is an on-board recording device connected to the vehicle's CAN (Controller Area Network) bus. It records measures such as: vehicle speed, speed of each wheel, longitudinal acceleration, odometer, steering wheel angle, brake pedal (0/1), ABS (0/1), etc. DIRCO offers the possibility of connecting other data sources as well, such as a GPS, an inertial station measuring the 3D acceleration and angle of the vehicle, or cameras[1]. DIRCO acquires each data stream as a time sequence. The data from a source are stored in a specific file and each record is a tuple: $(t_i, \alpha_i^1, \alpha_i^2, ..\alpha_i^n)$, where t_i is the i^{th} time instant and α_i^k is the i^{th} value provided by the k^{th} sensor. As a detail, DIRCO allows an acquisition at very high frequency of up to 10ms cycles. The data flows from different sources are asynchronous.

While it may function as a black box for vehicles in order to reconstruct the circumstances of an accident, the DIRCO is primarily a research tool that can help to analyze the driving behavior, vehicle safety and diagnose problems related to road infrastructure. Its 16GB of flash memory allows data acquisition, camera off, for several months. A simple

[1] The video stream is stored separately and its management is not in the scope of this work

scenario is to equip several vehicles such as buses or cars with DIRCO, retrieve and centralize these data and then analyze them in order to identify behavioral patterns of driving.

This type of approach is also appropriate to the evaluation of recently emerged ADAS (Advanced Driver Assistance Systems). Whether the system is already well known as a GPS or speed control device, or it is an experimental system such as obstacle detection, all require an accurate and extensive assessment of their impact on driving. The European Commission is funding since 2008 large-scale projects to evaluate mature technologies in the category of "intelligent transportation" systems. A particular aspect of these projects is to recourse to systematic collection of driving data with devices similar to DIRCO. Among these projects there is LAVIA (see Chapter VI), or euroFOT (http://www.eurofot-ip.eu/) where hundreds of topics will be studied for at least a year. The data will be analyzed to assess the studied systems and draw conclusions regarding their generalization to future generations of vehicles.

Given this kind of application, one can easily understand the importance of developing a database adapted to the characteristics of these data, whether in relation to the storage volume, or in relation to geo-localized and temporal features. The different types of studied systems induce a large variability in the methods of analysis and often involve a high level of required detail (e.g., situations of near-accident). Some indicators can be calculated by using common database management systems, but sometimes at the cost of heavy programming and prohibitive computational time. In addition, no system seems at present able to manage speed profiles[2] (or any other information) measured at different times and positions but on the same road. However, a large number of queries in this context need this kind of approach.

To illustrate the contribution of our model in this context, we refer throughout the article to the following typical queries. These queries consider a database that stores past trajectory data collected from a large number of vehicles sharing a given road network:

Q1. What is the acceleration profile along a given route segment for a given trip?

Q2. What is the difference between the vehicle's speed profile and the speed limit along a road segment?

Q3. How many times was the ABS enabled for a given trip?

Q4. What are the trips where the practiced speed exceeds a specified speed profile (e.g., the speed limit) by a certain value and what is the difference?

Q5. What is the ratio between speed and engine RPM for a given trip?

Q6. What is the average profile of acceleration for all vehicles passing through a certain road section (e.g., curve)?

Q7. Calculate the maximal speed profile of all vehicles passing through the indicated road section.

Q8. Find the average speed profile actually practiced (85th percentile of the passing vehicles) on a road before and after the installation of a speed camera.

[2] The concept of profile is introduced in Section III.3

Q9. What is the average profile of the fuel consumption on a road before and after the installation of a traffic calming device (e.g., a speed cushion)?

Q10. What is the minimum and maximum profile of fuel consumption on a road, and what is its difference with the profile of the studied driver?

Modeling temporal sequences is feasible by using functions over time [4], but it is not useful for the above type of analysis. Indeed, the measures from the trips are collected at different times and comparing these profiles makes sense only if they were measured in the same place. What matters is not the time at which the measure was recorded, but rather where it took place on the road. The concept of spatial profile of a measure (speed, acceleration) reflects the relationship between the measure and space. However, this notion of profile could not be defined in the model Güting or any other model. It is therefore necessary to extend the existing model with new data types. Moreover, the above queries demand specific operations on the measure profiles. This kind of operations that was not necessary in the context of analyzing only the MO trajectories, are of major importance in this context.

III.3 Related Work

The management of MOD has received particular attention in the recent years because of the advances and the omnipresence of mobile and geo-location technologies, such as cellular phones or GPS. Many works focus on modeling and language. We concisely presented some of the related work in this area in Section II.3. Then, we detailed in Section II.4 the state-of-the-art framework for modeling and querying moving objects. Our proposal is based on this model and extends it with specific data for mobile sensors.

However, to our knowledge, there are very few works that address the general problem of managing mobile objects equipped with sensors, such as the considered type of application. Most of the works on sensor data only consider the temporal aspect, i.e., they model the data as time series. But as we indicated earlier, the main characteristic of the sensors in the considered application domain is their *continuous* mobility in space and time. Moreover, the values issued by this type of sensors are more dependent on location than time.

The sole work (i.e., in the database research area of moving objects and sensor data) that we are aware of that considers the sensor mobility (in the sense that we described above) is the one in [33]. This work deals with data management in the context of sensor networks. Several applications are targeted such as monitoring of environmental or urban phenomena, management of fleets of vehicles, or traffic monitoring. The generic UML model is given in Figure III.2. A sensor network is composed by sensors. Each sensor can issue different types of measures. The model distinguishes between three types of sensors: fixed, agile and mobile. An agile sensor is a mobile sensor that has limited mobility, i.e., their mobility rate can not exceed a certain threshold value. A data record coming from a mobile sensor has the next attributes: *idR*, *idS*, $\{S_i, T_i, \{A_i^j\}\}$, where *idR* is the data record identifier, *idS* is the sensor identifier, S_i is the spatial location at time instant *i*, T_i is the time instant, and $\{A_i^j\}$ is the set of reported measures at instant *i*. This is similar with the raw data recorded by DIRCO (Section III.2).

We consider the problem of modeling mobile sensors data to be a very important aspect of moving objects databases. Simple models like the one in [33] are very limited in terms of modeling and querying these kinds of data. Moreover, as mobile sensors represent a special

case of moving objects, a unified model that includes both the spatio-temporal trajectory and the sensor data of a moving object is the right answer for modeling and querying moving objects equipped with sensors. Therefore, we extend in the next section the framework of Güting et al. with new data types and new operations.

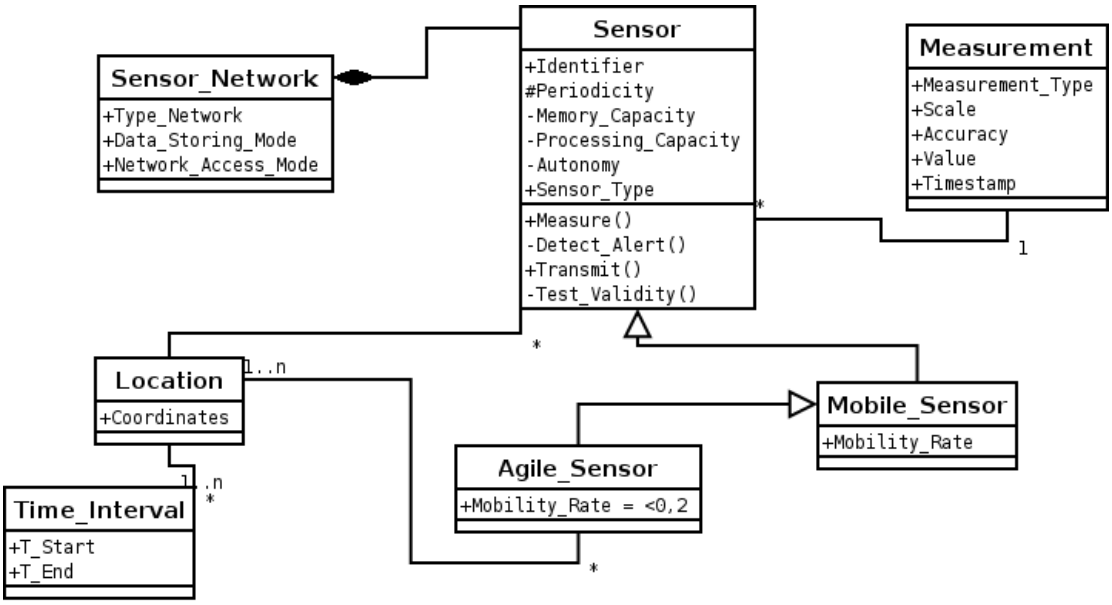


Figure III.2: Data model for measures and sensors [33]

III.4 Proposed Abstract Model

III.4.1 Introduction of New Data Types

Like the algebraic model in [5] described in Section II.4, our model includes a spatio-temporal type to model the trajectory of the moving object, and a temporal type to model the data generated by sensors. The temporal type is a function of time to base types (e.g., *real*, *int*). It expresses the variability of sensor measures from the temporal point of view.

However, the temporal view is not sufficient to model the *data from mobile sensors*, because the measures are often closely related to space. For completeness, the model should describe beside the evolution over time, the spatial evolution of measures.

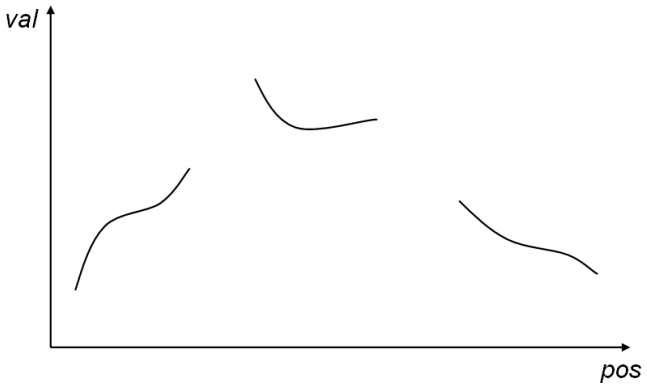


Figure III.3. Example of spatial profile of a real value

To this end, we extend the model of [5]. We introduce a new concept describing the spatial profile of measures. The idea is to describe the evolution of a measure in space. This concept is divided into two categories: *SVARIANT* to describe the profile in a two-dimensional space, and *GVARIANT* for the profile along the network.

We propose two new type constructors called *smoving* and *gmoving* (see Table III.1):

$BASE \rightarrow SVARIANT$	*smoving*, *inpoint*
$BASE \rightarrow GVARIANT$	*gmoving*, *ingpoint*

Table III.1. New data types

The definitions of these type constructors are given below using the notation in [4]:

Definition III.1: Given α a *BASE* type having the carrier set A_α, then the domain of definition for $smoving(\alpha)$ is defined as follows: $A_{smoving(\alpha)} = \{ f \mid f : \overline{A}_{point} \rightarrow \overline{A}_\alpha$ is a partial function and $\wedge\Gamma(f)$ is finite $\}$, where $\overline{A}_\beta = A_\beta \setminus \{\perp\}$ and $\Gamma(f)$ denotes the set of maximal continuous components of the function f.

Definition III.2: Given α a *BASE* type having the carrier set A_α, then the domain of definition for $gmoving(\alpha)$ is defined as follows: $A_{gmoving(\alpha)} = \{ f \mid f : \overline{A}_{gpoint} \rightarrow \overline{A}_\alpha$ is a partial function and $\wedge\Gamma(f)$ is finite $\}$, where $\overline{A}_\beta = A_\beta \setminus \{\perp\}$ and $\Gamma(f)$ denotes the set of maximal continuous components of the function f.

The spatial varying types obtained through the *smoving* or *gmoving* type constructors are functions, or infinite sets of pairs (position, value). It is practical to have a type for representing any single element of such a function, i.e., a single (position, value)-pair, for example, to represent the result of a slice operation at a given point in space. The *inpoint* and *ingpoint* type constructors converts a given base type α into a type that associates positions in (2D or network) space with values of α.

Definition III.3: Let α be a *BASE* data type with carrier set A_α. Then the carrier set for $inpoint(\alpha)$, is defined as follows:

$$A_{inpoint(\alpha)} = A_{point} \times A_\alpha .$$

Definition III.4: Let α be a *BASE* data type with carrier set A_α. Then the carrier set for $ingpoint(\alpha)$, is defined as follows:

$$A_{ingpoint(\alpha)} = A_{gpoint} \times A_\alpha .$$

Since our work focuses on constrained movement, we only detail the second category of types in the sequel, i.e., *gmoving* and *ingpoint*. These definitions state that a spatial profile of a measure is a partial function. Each value f in the domain of $gmoving(\alpha)$ is a function describing the evolution in the network (graph) space of a *BASE* value. The *gmoving* type constructor describes an infinite set of pairs (*position, value*), where the position is a *gpoint*.

Figure III.3 presents a spatial profile of a real measure on a given road. The x-axis represents the relative position on the road that can vary between 0 and 1.

The condition "$\Gamma(f)$ is finite" means that f consists of only a finite number of continuous components. For example, the profile in Figure III.3 has 3 continuous components. This condition is needed as a precondition to make the design implementable. It also ensures that projections of ***gmoving*** objects (e.g. on the spatial axis) have only a finite number of components.

The presented model is an *abstract model*, which means that in general the domains or carrier sets of its data types are infinite sets. To be able to implement an abstract model, one must provide a corresponding *discrete model*, i.e., define finite representation for all the data types of the abstract model. This is done by the *sliced representation* introduced in [19]. Thus, a time dependent or spatial dependent value is represented as a sequence of slices such that within each slice the evolution of the value can be described by some "simple" function (e.g., linear functions or quadratic polynomials). More details on the sliced representation are given in Section II.4.2 of the thesis. We discuss the discrete representation for the new types in Section III.5.2.

III.4.2 Introduction of New Operations

As for the type system definition, we use the operations in the algebra of Güting et al. [4], [5] as a starting point. By introducing new types, we have to (i) extend the existing operations (there where the extension makes sense) and (ii) add new specific operations for the target application type.

There are two ways for extending the existing operations. The first is to overload the operations with new signatures for the new type. We apply this procedure to operations that perform projections and interactions to/with the domain and range, as the *gmoving* type is a partial function similar to *moving* types.

The second way of extending the existing operations is to use a similar process with the *temporal lifting*, described in Section II.4.1.1. The temporal lifting permits generating from a non-temporal operation with the signature $\alpha_1 \times \alpha_2 \times ... \times \alpha_n \to \beta$, the temporal equivalent one having the signature $\alpha_1' \times \alpha_2' \times ... \times \alpha_n' \to moving(\beta)$, where $\alpha_i' \in \{\alpha_i, moving(\alpha_i)\}$. Each of the arguments can become temporal, which makes the result temporal as well. We adopt this principle to generate the equivalent space variant operations. We propose a *spatial lifting* for the non-gvariant non-temporal operations. The operation induced by the spatial lifting is available for a signature $\alpha_1' \times \alpha_2' \times ... \times \alpha_n' \to gmoving(\beta)$, where $\alpha_i' \in \{\alpha_i, gmoving(\alpha_i)\}$.

We have also defined new operations that apply to *GVARIANT* and *TEMPORAL* types, which are necessary in this context. Table[3] III.2 presents the list of the new operations, i.e., extended from the existing ones or newly introduced. We describe in this section some operations. Other operations are explained with the example queries in the next section. There are five classes of operations. The first two correspond to the extension of existing operations, while the last three are new types of operations. Moreover, the first four groups represent conventional operations, i.e., those which take as input one or more objects (values) in accordance with their signature and return an object (a value). The last class includes

[3] In Table III.2 α is always a *BASE* type (e.g., real, int, bool)

aggregate operations, i.e., that return a single result based on a group of objects (similar to aggregates in the relational model).

The first class of operations comprises the projection in the network or value (range) domains. Thus, **trajectory** returns to the network path of a trip. The operation **rangevalues** performs the projection in the range and returns one or several intervals of base values. Operations **val** and **pos** return respectively the value or the network position for an *ingpoint* type, which is defined as a pair (*gpoint*, *value*). The second class of operations concerns the interaction with the domain (network space) and range (values). They make selections or clippings according to criteria on one of the axes of variation (network space or values). Thus, **present** is a predicate that checks if the input object is defined at a given position in the network. Finally, the predicate **passes** allows one to check whether the moving value ever assumed one of the values given as a second argument.

Class	*Operation*	*Signature*
Projection to Domain/Range	**trajectory**	$gmoving(\alpha) \rightarrow gline$
	rangevalues	$gmoving(\alpha) \rightarrow range(\alpha)$
	pos	$ingpoint \rightarrow gpoint$
	val	$ingpoint \rightarrow \alpha$
Interaction with Domain/Range	**atpos**	$gmoving(\alpha) \times gpoint \rightarrow ingpoint$
	atgline	$gmoving(\alpha) \times gline \rightarrow gmoving(\alpha)$
	present	$gmoving(\alpha) \times gpoint \rightarrow bool$ $gmoving(\alpha) \times gline \rightarrow bool$
	at	$gmoving(\alpha) \times \alpha \rightarrow gmoving(\alpha)$ $gmoving(\alpha) \times range(\alpha) \rightarrow gmoving(\alpha)$
	atmin	$gmoving(\alpha) \rightarrow gmoving(\alpha)$
	atmax	$gmoving(\alpha) \rightarrow gmoving(\alpha)$
	passes	$gmoving(\alpha) \times \beta \rightarrow bool$
Basic Algebraic Operations	**sum, sub, mul, div**	$moving(\alpha) \times moving(\alpha) \rightarrow moving(\alpha)$ $gmoving(\alpha) \times gmoving(\alpha) \rightarrow gmoving(\alpha)$
Calculations	**mean[avg], min, max**	$moving(\alpha) \rightarrow real$ $gmoving(\alpha) \rightarrow real$
	no_transitions	$moving(int) \rightarrow int$ $gmoving(int) \rightarrow int$
Aggregates	**min_agg, max_agg, sum_agg, avg_agg**	$\{moving(\alpha)\} \rightarrow moving(\alpha)$ $\{gmoving(\alpha)\} \rightarrow gmoving(\alpha)$
	percentile	$\{moving(\alpha)\} \times real \rightarrow moving(\alpha)$ $\{gmoving(\alpha)\} \times real \rightarrow gmoving(\alpha)$
	count_agg	$\{moving(\alpha)\} \rightarrow moving(int)$ $\{gmoving(\alpha)\} \rightarrow gmoving(int)$

Table III.2. Examples of operations for new types

The third class of operations considers the basic algebraic operations ('+', '-', '.' and '/'), which we include in non-gvariant non-temporal collection of operations. Therefore, they

become subject to temporal and spatial lifting. We use named functions, i.e., **sum**, **sub**, **mul** and **div**, as for all defined operations. These operations are useful for the analysis of sensor measures. For example, they can calculate the difference between the speed profiles of two moving objects on the common part of their trajectories, or return the difference between the practiced speed and the speed limit on a route. These operations take as input two functions of the same type (*GVARIANT* or *TEMPORAL*) and calculate a result function of which the definition domain is the intersection of the input objects' domains. For the division, the parts where the operation is not defined, are also eliminated from the domain of the result function.

The fourth class of operation addresses the same categories of types, i.e., *GVARIANT* or *TEMPORAL*. The specified functions are: **mean**, **min**, **max** and **no_transitions**. Each of these operations takes as input a function of time or space and returns a value representing the aggregate of the input function. Their interest is to calculate an average or an extreme value for any measure, given a temporal or spatial interval.

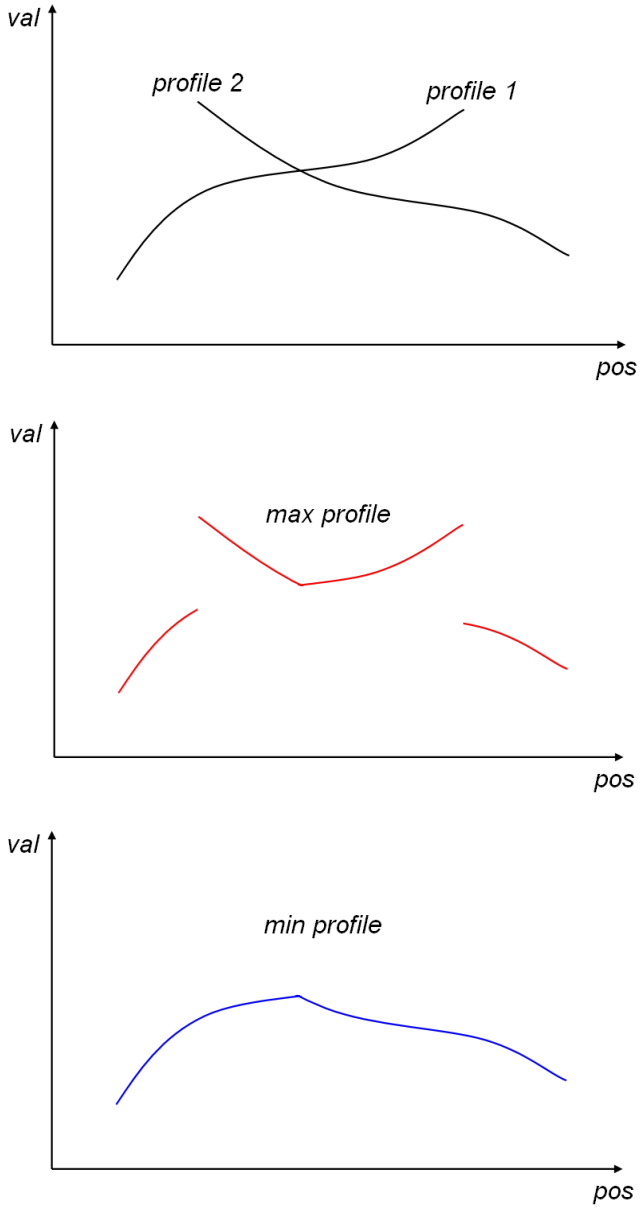


Figure III.4. Example of using max_agg (second graph) and min_agg (third graph) on two profiles (first graph)

The last class of operations concerns the aggregates. Aggregate operations return a single object result given a set of objects of the same type (see Figure III.4). Unlike the previous class, these operations define aggregations of groups of objects. Some of these aggregates return an object of the same type as the input type, e.g., the average (**avg_agg**), the minimum (**min_agg**) and maximum profile (**max_agg**). The aggregate **count_agg** returns the number of profiles in the definition domain in the form of a *moving(int)* or *gmoving(int)* object. Finally, the function **percentile** computes the profile below which is found a certain percentage of profiles in the input set. The definition domain of the result function for an aggregate operation is the union of domains of aggregated functions. The usefulness of aggregate operations is shown by the queries Q6 to Q10.

The proposed collection of operations is only a basis, however rich, of functionality. Other operations may be added to meet specific needs of some applications. Thanks to advances in the extensibility of the existing DBMS, these types and operations can be easily integrated into the DBMS. Thus, it becomes possible to use them through the standard SQL language. Then, the problem of optimizing queries must be addressed. This is exactly the plan that we followed to implement our data server for moving objects with sensors.

III.4.3 Query Examples

The great interest of using the extension capabilities of a DBMS is to easily integrate new types and operations in the SQL standard interface. The query examples in this section are based on a relational schema with one table that contains information on vehicle trips as follows:

```
vehicle_trip(mo_id: int, trip: moving(gpoint),
g_speed: gmoving(real), t_speed: moving(real),
g_acceleration: gmoving(real), t_acceleration: moving(real),
g_RPM: gmoving(real), t_RPM: moving(real),
g_odometer: gmoving(real), t_odometer: moving(real),
g_ABS: gmoving(bool), t_ABS: moving(bool),
g_breakSwitch: gmoving(real), t_breakSwitch: moving(real))
```

In addition to the spatio-temporal trajectory, i.e., the "trip", the table contains sensor data reporting the speed, acceleration, RPM, odometer, ABS and brake pedal state. These data are modeled by functions of space (prefixed with g_) and of time (prefixed with t_). The parameters are prefixed with the symbol "&" and could be either given by the user at runtime, or existing from previous calculations.

Q1. *What is the acceleration profile along a given route segment for a given trip?*

```
SELECT atgline(g_acceleration, &aGline)
FROM vehicle_trip
WHERE mo_id = &anID
```

The operation **atgline** returns the longitudinal acceleration profile restricted to the sub-space specified by the geometry aGline given as parameter.

Q2. *What is the difference between the vehicle's speed profile and the speed limit along a road segment?*

```
SELECT sub(g_speed, &legalSpeed)
FROM vehicle_trip
WHERE inside(trajectory(&legalSpeed), trajectory(g_speed)) = 1
```

The difference between two functions describing measure profiles is calculated using the operation **sub**. An indexed predicate like **inside** could accelerate the response time of the query. This operation has two *gline* parameters and checks whether the first is included in the second. To obtain the projection in space of a measure profile, we use the operation **trajectory**.

Q3. *How many times was the ABS enabled for a given trip?*

```
SELECT no_transitions(g_ABS)/2
FROM vehicle_trip
WHERE mo_id = &anID
```

This query simply illustrates the use of **no_transitions,** which is applicable to discrete *BASE* types (e.g., *bool*, *int*) and returns the number of transitions for a given discrete function.

Q4. *What are the trips where the practiced speed exceeds a specified speed profile (e.g., the speed limit) by a certain value and what is the difference?*

```
SELECT mo_id, sub(g_speed,&legalSpeed)
FROM vehicle_trip
WHERE intersects(trajectory(&legalSpeed),
          trajectory(g_speed)) = 1
     AND max(sub(g_speed,&legalSpeed)) > &threshold
```

There are two new operations in this query. First, the predicate **intersects** is similar to **inside**, the only difference being that it only searches for an intersection between the two *gline* parameters and not for inclusion. Second, the operation **max** is an aggregate of a function. We use it to verify if the maximal value assumed by the function given as parameter is above a certain threshold value. As in the previous query, the parameter for **max** is represented by the difference between the practiced and legal speed profiles.

Q5. *What is the ratio between speed and engine RPM for a given trip?*

```
SELECT div(t_speed, t_RPM)
```

```
FROM vehicle_trip
WHERE mo_id = &anID
```

This query shows the usefulness of basic algebraic operations for comparing temporal profiles of the same moving object. The profile obtained by dividing the vehicle speed to the engine RPM can be used to detect the behavior regarding the gear shifting of a driver.

Q6. *What is the average profile of acceleration for all vehicles passing through a certain road section (e.g., curve)?*

```
SELECT avg_agg(g_acceleration)
FROM vehicle_trip
WHERE inside(trajectory(&aCurve), trajectory(trip))=1
```

We determine with this query the average acceleration profile of all vehicles passing through the indicated route section. The function **avg_agg** generates a new *gmoving(real)* object from the set of objects of the same type, passed as a parameter, i.e., all tuples of the table that match the predicate in the WHERE clause.

Q7. *Calculate the maximal speed profile of all vehicles passing through the indicated road section.*

```
SELECT max_agg(atgline(g_speed, &aRoad))
FROM vehicle_trip
WHERE intersects(trajectory(g_speed), &aRoad) = 1
```

First we find all the trips that intersect the given road. For these trips, we select by the function **atgline** the speed profile that corresponds to the road. Then we aggregate the resulted profiles in order to obtain the maximal profile, by using the aggregate **max_agg**.

Q8. *Find the average speed profile actually practiced (85th percentile of the passing vehicles) on a road before and after the installation of a speed camera.*

```
SELECT percentile(atgline(g_speed,&aRoad), 85)
FROM vehicle_trip
WHERE intersects(trajectory(g_speed),&aRoad)) = 1
     AND inst(initial(trip)) < &instalationDate
```

The query finds the average speed profile (85^{th} percentile) before installing a speed camera. A similar query should be posed to find the same profile after the installation of the camera. As for the query Q7, we filter the trips by retaining only those that intersect the given road, and that begin before the installation date of the camera. To do this we use the combination of functions **inst** and **initial** that return the start date of a trip. Finally, we apply the **percentile** function on all selected profiles. The second parameter of this function is the n^{th} percentile.

III.5 DBMS Extension and Implementation Issues

The main benefit of such a model is that it can be implemented as a database extension. Hence, the proposed types and operations become available in the SQL-like language of the DBMS like in the above examples. As we explained in Section II.4.2.1, the current support for spatio-temporal data in the existing DBMSs is limited. There are only a few prototypes that implement spatio-temporal DBMS extensions, e.g., SECONDO [28] and HERMES [26]. Nevertheless, the database system remains the main actor in the data chain processing.

Figure III.5 depicts the whole data-chain processing for the application that motivated this work (see Section III.2). The raw data coming from the natively embedded sensors in the vehicle and from other sensors (GPS, camera, etc.) are recorded by the data logger. These data have a simple format. For each trip, a new file is created. The file contains a temporal sequence for each sensor, i.e., the reported value at a given time. After a period of time, the recorded data files are gathered in a central database. The data is firstly map-matched by using a road network database. Then, it is transformed accordingly to the spatio-temporal data model implemented in the DBMS. Finally, it is integrated in the spatio-temporal server (STDBMS) and becomes available for querying in an extended SQL-like language. Several services, e.g. visualization, analyzing tools, etc., can be based on the STDBMS, which may in turn be employed by the end-user applications.

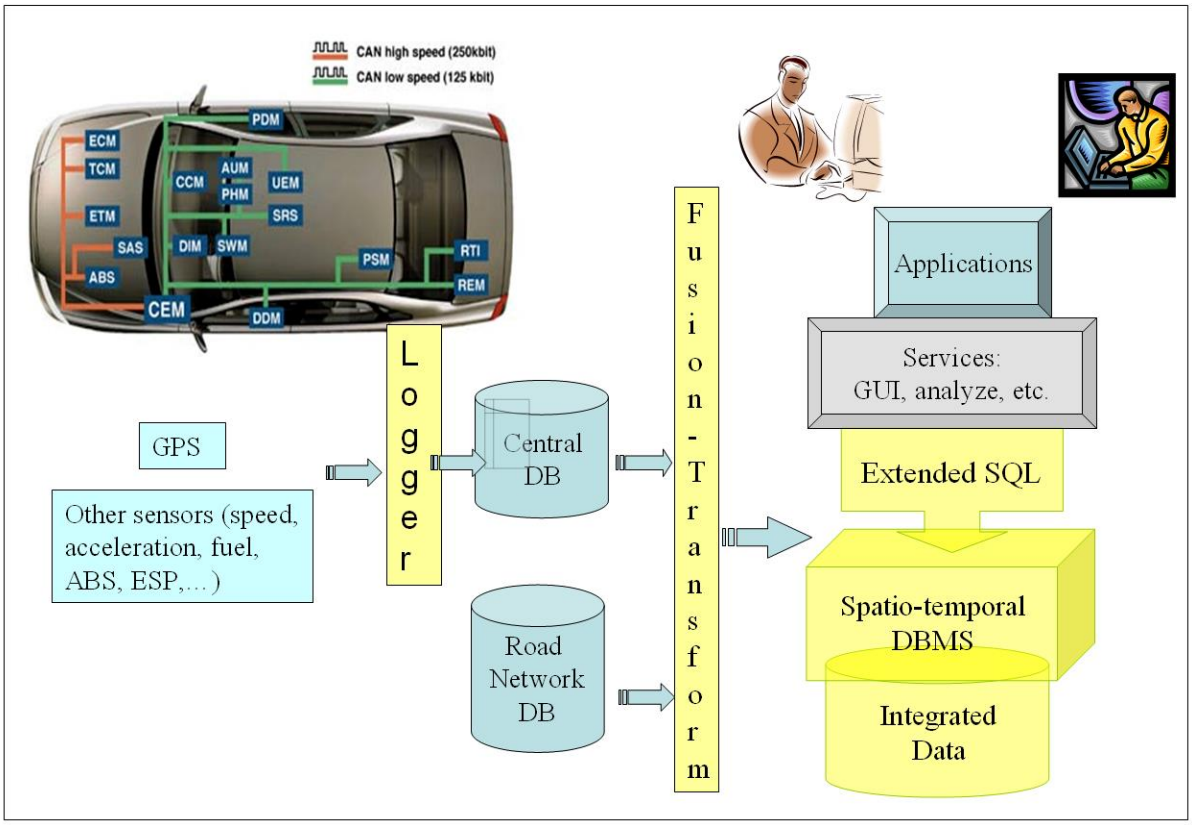


Figure III.5. Example of data chain processing

There are many advantages in using a specialized STDBMS over a data processing based on the files that contain the raw data. Here are some examples:

- Passing from “flat files” to an object oriented model, which offers an abstract, “simpler” object view over the data.

- Passing from ad hoc implementations (programs) that demand an important development effort, to a library of functions available in SQL.

- Analyzing and querying the data in all its dimensions (spatial, temporal and sensor values). Also combining all the dimensions due to SQL expressivity.

- Capturing into the model the continuous variability of the data in all dimensions. This can better estimate the results.

- Managing asynchronous temporal data with different temporal scales because of the continuous view of data.

- Managing important volumes of data efficiently, due to internal optimizations of the DBMS.

- Using advanced available technologies of off-the-shelf DBMSs [9].

III.5.1 Database System Architecture

To integrate a new model into a DBMS, an extensible DBMS architecture is needed. The DBMS should offer interfaces to register components such as the following [3]:

- Data structures for the data types

- Functions corresponding to operations over the data types

- Index structures for the data types

- Computational cost and selectivity estimation for the operations

- Extension of the query optimizer to include the cost and the selectivity of the operations

- Extensible user interface to include the presentation of the data types

Most commercial or open source databases today offer extension possibilities. New algebra modules can be developed by users to respond to the demands of non-standard applications. For example, such modules can be added as a *data cartridge* in Oracle Server, as a *data blade* in Informix Universal Server or as an *extender* in IBM DB2. Creating new data types and operations to encapsulate them, is feasible in most database. The creation of new index structures and the extension of the query optimizer are also available, but can be limited to a certain extent.

Rather than developing a prototype from scratch, we chose to implement the proposed model under such an existing system, i.e., the Oracle DBMS. Thus, all types are implemented as new object types in Oracle 11g. The operations are implemented in Java (Oracle DBMS integrates a Java Virtual Machine) and stored as a package in the database. These operations can be used in SQL queries along with the existing operations in the DBMS.

Finally, some filtering operations, i.e., operations used to identify the MOs that verify a certain spatial, temporal or on-value predicate, are indexed in order to accelerate the query response time and to provide a system scalable with the dataset size (see Section III.5.4). We integrated the indexes by using the data cartridges in Oracle. The general architecture of the system is given in Figure III.6. Notice that other DBMS systems that allow DBMS extensions can be used. For example, we currently test the system portability to SECONDO. We discuss in more detail the technical aspects of the implementation and test several query scenarios in Chapter VI.

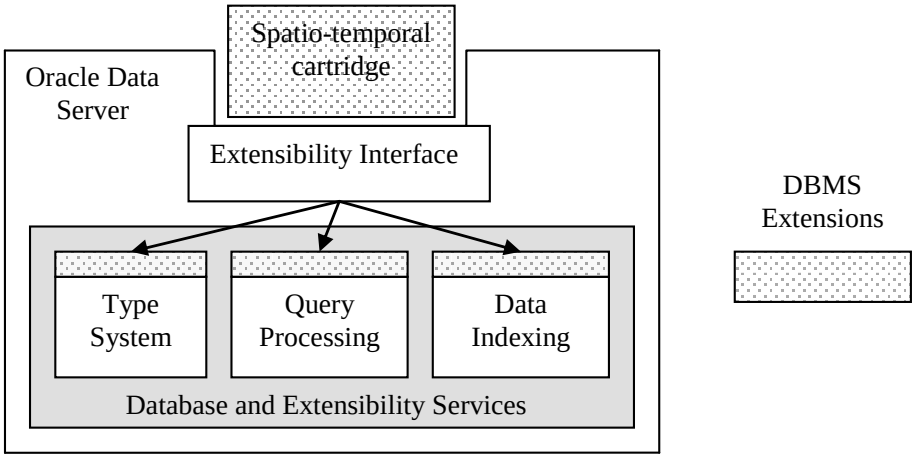


Figure III.6. Database System Architecture

III.5.2 Data Type Representation

The model in [4], [5] that we extended in Section III.4 is an abstract model. A finite representation of this abstract model is needed in order to be able to implement it. For all *moving* types, the so-called *sliced representation* has been proposed in [19]. We presented this discrete representation in Section II.4.2. In short, a *moving* object in the abstract model is a temporal partial function. The sliced representation represents the moving object as a set of so-called *temporal units* or *slices*. A temporal unit for a moving data type α is a time interval where values taken by an instance of α can be described by a "simple" function. The "simple" functions used for the representations are the linear function or quadratic polynomials. The motivation for this choice is a trade-off between the richness of the representation and the simplicity of the representation of the discrete type and of its operations. For example, a unit for a *moving(real)* object is represented as a tuple (a, b, c, t_1, t_2), where a, b, c are the coefficients of a quadratic polynomial and t_1, t_2 is the unit time interval. The moving value at a time instant t inside the unit time interval is computed as $a \times t^2 + b \times t + c$. As indicated in Chapter II (Section II.4.2), more complex functions for unit representation can be imagined but are not considered in this document. The current version of our implemented prototype uses only linear functions for the unit representation.

For all *gmoving* types that we introduced in Section III.4.1, we adopt the sliced representation as proposed in [19]. This is straightforward as the sole difference is to replace the unit time interval, which is the support for temporal profiles, with a unit spatial interval, which is the support for spatial profiles. Figure III.7 depicts a simple example of a spatial profile on a given road. The carrier set for the type *ugreal* (i.e., the unit of a *gmoving(real)*) is

$$D_{ugreal} = \{(rid, side, pos_1, pos_2, a, b, c) \mid rid \in int, side \in \{up, down, none\}, a, b, c, pos_1, pos_2 \in real\}$$

and the evaluation at position *gpoint* = (*rid*, *side*, *pos*) is defined by:

$$v((a,b,c),gpoint) = a \cdot pos^2 + b \cdot pos + c$$

A spatial interval (or a route interval) given a network space has the following elements: (*rid*, *side*, pos_1, pos_2), where *rid* is a road identifier, *side* gives the route side, and pos_1, pos_2 are relative positions on the road. Hence, a *gmoving(real)* object will contain a set of units with the following attributes: *(a, b, c, rid,* pos_1, pos_2*)*. To calculate a value for a given position, we first locate the corresponding unit, i.e., where the spatial interval includes the position, then calculate the value as $a \times pos^2 + b \times pos + c$.

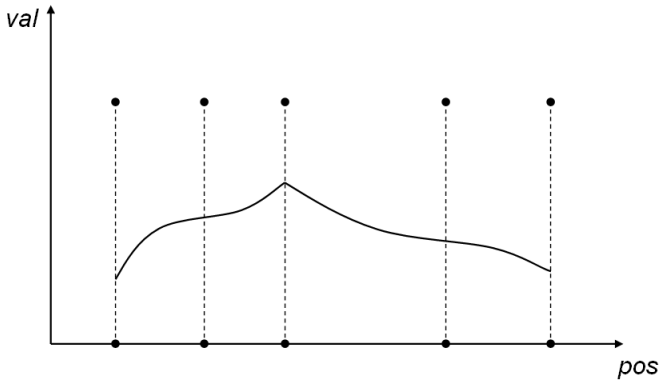


Figure III.7. Example of sliced representation of a spatial profile

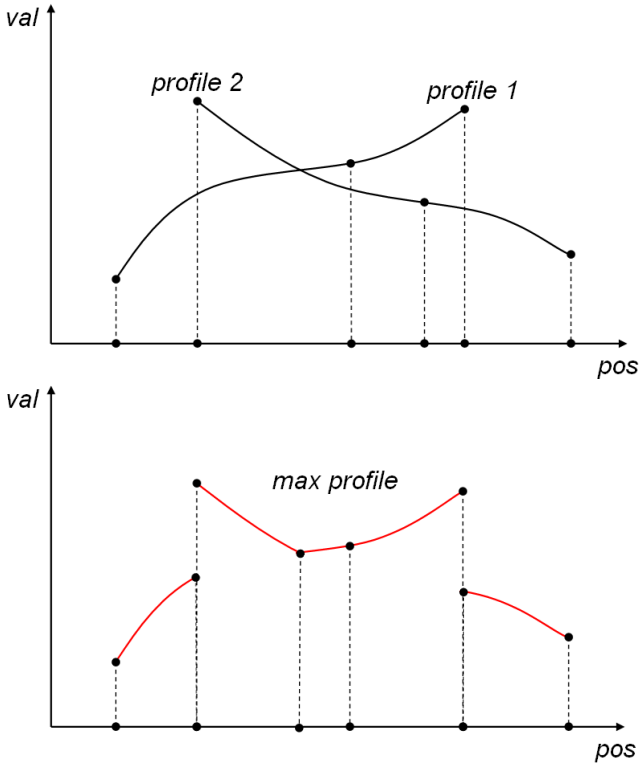


Figure III.8. Example of fragmentation after using the max_agg (second graph) on two profiles (first graph)

III.5.3 Handling Aggregation Functions

Unlike the functions on one or two profiles, aggregate functions operate on a set containing a (possibly) large number of profiles. This can lead to a fragmentation of the result profile in a large number of small units and a degradation of the performance. Consider the example in Figure III.8. The first graph shows two profiles with their decomposition into units and the second one represents the maximum aggregate of these profiles. Notice that the units of the two profiles do not have the same spatial distribution. The space intervals of the units of the two profiles overlap partially. This is common because the unit slicing is unique to each profile and depends on the variability of the observed measure at the observation time. Therefore, the result of an operation on a set of profiles is another profile that contains more units that the initial profiles. This process of fragmentation of the result is not disturbing when the calculation is done only on two profiles. However, in the case of the aggregation it can significantly slow down the computation time.

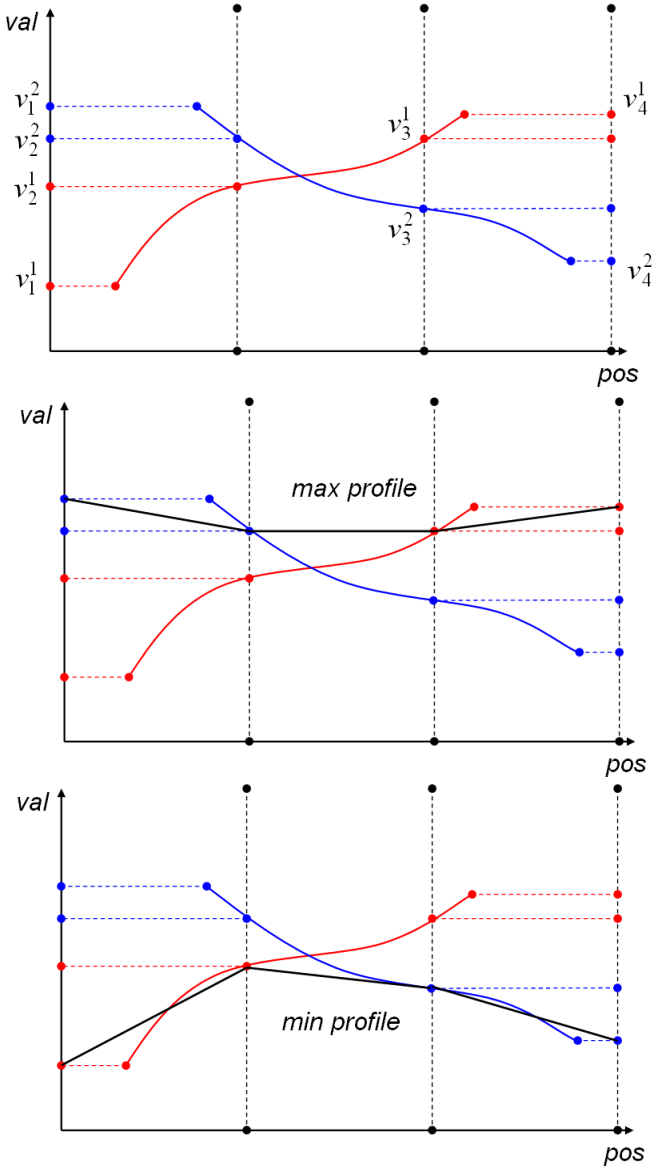


Figure III.9. Example of calculating the max_agg (second graph) and min_agg (third graph) on two profiles (first graph) using a regular slicing

To accelerate the aggregate operations we propose a regular temporal or spatial slicing of profiles, independent of the initial slicing. This method offers a compromise between efficiency and quality of results. Thus, for aggregates on *gmoving* types for example, we uniformly divide the space, beginning with the start point of each road in intervals of a given length, e.g., 10 meters. Smaller intervals will produce higher quality results but at a cost of a slower performance, and vice versa.

Figure III.9 presents an example of using uniform slicing for computing aggregates on two spatial profiles. The first graph shows the profile decomposition by regular intervals (represented by the vertical dotted lines). For each interval, we compute or extrapolate first the values on the end limits of the interval. Thus, for the first profile (in red) we find the values v_1^1 and v_2^1 the first interval, v_2^1 and v_3^1 for the second interval, and v_3^1 and v_4^1 for the third interval. From these values computed for all profiles, we apply the corresponding scalar aggregate function (e.g. the aggregate max for **max_agg**) in order to generate the values of the resulting profile on the same limits of the intervals.

Overall, using this approach to implement the aggregate functions produces approximate results, but it offers in return a good optimization of this costly type of operation. The analysis of the result quality depending on the granularity of slicing is left for future work.

III.5.4 Operators

In the field of spatio-temporal databases, the indexing techniques that permit processing efficiently the spatial, temporal and on-value queries are complementary to modeling the moving objects. We do not detail here the access methods in the spatio-temporal context as this is discussed in the second part of this document (i.e., Chapter IV and Chapter V). Instead, we present in this section the mechanism by which the indexes are linked to the implemented algebra, i.e., object types and operations. This mechanism is based on *operators*.

Operators are a subset of the algebra operations, mostly predicates such as **present** or **passes**, that benefit of an index based evaluation in addition to the basic function implementation (see Chapter IV Section IV.6). The functional implementation is used when the operator is invoked in the select list of a SELECT command or in the ORDER BY and GROUP BY clauses. However, when the operator appears in the condition of a WHERE clause, the DBMS optimizer chooses between the index implementation and the functional implementation, taking into account the selectivity and cost while generating the query execution plan.

The operators that we implement are *spatial*, *temporal* and *on-value* predicates, or predicates that combine two of the three possible dimensions (i.e., *spatio-temporal*, *on-value spatial* and *on-value temporal*). For example, to select only the profiles that spatially intersect a given network region, one will use the **present** operator.

```
SELECT mo_id
FROM vehicle_trip
WHERE present(g_speed, &aGLine) = 1
```

In the same way, the **passes** operator is used to select only the objects for which a certain measure assumes a given value (e.g., acceleration is above $10 m/s^2$).

```
SELECT mo_id
FROM vehicle_trip
WHERE passes(t_acceleration, 10) = 1
```

Finally, the two-dimensional predicates verify that the conditions in each dimension are simultaneous verified. For example, the spatio-temporal operator **intersectsAtPeriods** ensures that the trajectory of a MO intersects (or is included) a (in) spatial network region at a given time interval. Similar reasoning can be applied for the rest of the operators. The complete list of proposed operators is given in Table[4] III.3.

```
SELECT mo_id
FROM vehicle_trip
WHERE intersectsAtPeriods(trip, &aGLine, &aTimeInterval) = 1
```

Type of Operator	*Operator*	*Signature*
Temporal	**present**	$moving(gpoint) \times periods \rightarrow bool$ $moving(\alpha) \times periods \rightarrow bool$
Spatial	**intersects,** **inside**	$moving(gpoint) \times gline \rightarrow bool$ $gmoving(\alpha) \times gline \rightarrow bool$
On-value	**passes**	$moving(\alpha) \times range \rightarrow bool$ $gmoving(\alpha) \times range \rightarrow bool$
Spatio-temporal	**intersectsAtPeriods,** **insideAtPeriods**	$moving(gpoint) \times gline \times periods \rightarrow bool$
On-value Spatial	**passesAtGLine**	$gmoving(\alpha) \times range \times gline \rightarrow bool$
On-value Temporal	**passesAtPeriods**	$gmoving(\alpha) \times range \times periods \rightarrow bool$

Table III.3: Operators

III.6 Conclusions and Future Work

The use of sensors embedded in vehicles leads to new applications, which give rise to new research problems. In this chapter, we addressed the problem of modeling and querying. We stressed the limitations of existing work on moving objects. The new application type that we introduced in Section III.2 concerns moving objects equipped with sensors that produce additional data streams, which are strongly related to the trajectory of the object. A DBMS capable of managing in a unified manner the data concerning the moving object and the corresponding data flows is very useful for this kind of applications.

Our contribution is to propose a model for such a DBMS by extending an existing framework for moving objects. We first analyzed the limitations of modeling sensor data. Indeed, existing models can represent the data flows from a temporal point of view. We have

[4] In Table III.3 α is always a *BASE* type (e.g., real, int, bool)

shown that these measures are equally dependent of object's position and a representation relative to space is needed. Therefore, we have extended the existing type system with functions that describe the evolution of measures in space. We have also proposed a collection of operations in view of the enhanced system. We introduced the concept of *spatial lifting* inspired by the idea of the existing *temporal lifting*. We have redefined all the temporal operations and changed the semantics of some of them for the new types. Finally, we proposed new operations, and aggregates on the new types. An illustration of use of the DBMS is given by query examples involving the new defined types and operations. The current prototype includes a partial implementation of the algebra as a data cartridge in Oracle DBMS.

We presented in [37] a demonstration for a real application related to naturalistic driving in conjunction with intelligent transportation systems and the results are promising for a scenario on a larger scale. We discuss more on this matter in Chapter VI.

The presented application type usually generates very large datasets. As future work, we intend to study proper indexing techniques for the new types. Although this is a similar to the query optimization problem in moving object databases (see Chapter IV), sensor data types and distribution may lead to specific optimizations in our system. We also investigate the problem of mining such databases [36].

Chapter IV: Spatio-Temporal Indexing

Extending a database system to accommodate novel kinds of applications requires new specific models and optimizations that take into account the specificity of the data. We covered in the previous two chapters some aspects regarding the modeling of spatio-temporal data. In the next two chapters we discuss about optimization aspects, which are complementary to modeling. As indicated in Section III.5.1, there are many possible directions for optimizing a database system such as efficient algorithms for the operations over the data types, appropriate access methods, query plan evaluation and optimization, and so on. Among all these possible directions, we will focus on the indexing aspect in the next two chapters of this work.

IV.1 Introduction

Indexing has always been and continues to be a central research topic in the database world. Its main task is to find solutions to access a (set of) record(s) in a database as fast as possible, based on some search criteria. Thus, the access methods support the base search operations such as *exact search queries* or *range queries*, without being limited to these types of operations. Many access methods for the standard data types (e.g., numerical values, strings of characters) have been proposed, such as tree-based index structures (e.g. B-tree, B^+-tree) or hash-based index structures (e.g., extensible hashing, linear hashing).

However, more and more applications in domains such as multimedia, transport, biology, meteorology, require database support. These application need to store and query more and more complex data. Moreover, the volume of produced data appears to increase with its complexity. Hence, it is of major importance to provide appropriate access methods for such applications. In this context, the spatial, temporal and spatio-temporal applications make no exception. An impressive number of multidimensional spatial access methods (SAM) has already been proposed for spatial data. Similarly, a number of temporal access methods to index valid and/or transaction times has been devised. These two types of indexes constitute the basis of spatio-temporal access methods (STAM), which aim mainly at indexing the continuous movement of spatial objects.

In this chapter we present some of the related work on spatial, temporal and spatio-temporal indexing. The extent of this area is huge and a thorough presentation falls out of the scope of this work. Instead we focus on the aspects that are more related to the contributions of the thesis. Thus, is Section IV.2 we give an overview of a few spatial access methods. In Section IV.3 we focus on the interval indexing techniques. Spatial and interval indexing are very important in a spatio-temporal DBMS for efficient processing of spatial and temporal operators (see Chapter III Section III.5.4). Section IV.4 describes a few proposals for indexing past and current and near-future movement. We present in detail the work concerning the indexing of the past trajectories of moving objects in networks in Section IV.5, since this is directly related with our proposed method in Chapter V. In Section IV.6 we revisit the spatio-temporal operators introduced in Chapter III, and discuss their indexing by the access methods previously presented in this chapter. Finally, we conclude the chapter in Section IV.7.

IV.2 Spatial Access Methods

Similar to any kind of data, spatial data requires indexing methods to optimize the access of the specific spatial operations such as *range queries* (i.e., find all objects that overlap a given search window) and the *nearest neighbor queries* (i.e., find all objects with a minimum distance from a given object). An important number of multidimensional spatial access methods having various structures has been proposed in more than a decade of research in the area of spatial databases. Probably the most famous one is the R-tree (with his related structures), which made his way into many commercial database systems. In this section we present a few methods for spatial indexing. We focus on the widely known spatial access methods such as the R-tree family and on the methods that are reused in spatio-temporal indexing such as such as space-filling curves. A very good survey on multidimensional access methods can be found in [38]. This section is based on this survey.

IV.2.1 Data and Query Specificity

Any access method needs to consider the specificity of the data to be indexed as well as the specificity of the operations that it supports. Therefore, we proceed by presenting the data specificity and the query specificity for spatial data [38]. A first characteristic of spatial data is its *complexity*. Spatial data can range from a simple point value to complex geometries that may be composed by several hundreds polygons or more. This variety in spatial data is not a characteristic of conventional data, which has well defined limits. Another property of spatial data is that it tends to be *large*. Maps for instance, often occupy hundreds of megabytes or even several gigabytes of memory. It can then be easily understood that the support of access methods for secondary storage is indispensable. Moreover, the queries over spatial data may involve *costly* geometrical computations. Such operations are usually more expensive than standard relational operations. Finally, similar to any type of data, the spatial data can by *dynamic*. Hence, spatial indexes need to consider the possible modifications in the data (i.e., insertions, deletions, updates) and avoid an important degradation of the index performance.

As an access method is designed to support some kind of search within the data, it is equally important to discuss the types of spatial search operations. Searches in a spatial database can not only be based on non-spatial attributes (e.g., numerical or alphanumerical values), but also on spatial attributes. Such queries require fast execution of geometrical search operations. The spatial operations that benefit from an indexed implementation are called *spatial operators*. Such examples are the point query or the range query. Figure IV.1 depicts several examples of spatial queries. We can add to these types of queries two other spatial operators, i.e., the nearest neighbor operator and the spatial join operator.

Among these operations, the *region query* is probably the most employed. Given a subspace *S* of a d-dimensional Euclidian space E^d, a region query returns all the geometries in the database that overlap *S*. If all the faces of *S* are iso-oriented, i.e., they are parallel with the coordinate axis of the Euclidian space, then the query is called a *range query* or a *window query*. Moreover, given a point $p \in E^d$, a *point query* in *p* can be seen as a degenerate region query that returns all the spatial objects containing *p*. All these types of queries require appropriate multidimensional access methods to rapidly reach to the objects of a database that are situated at a certain location in space.

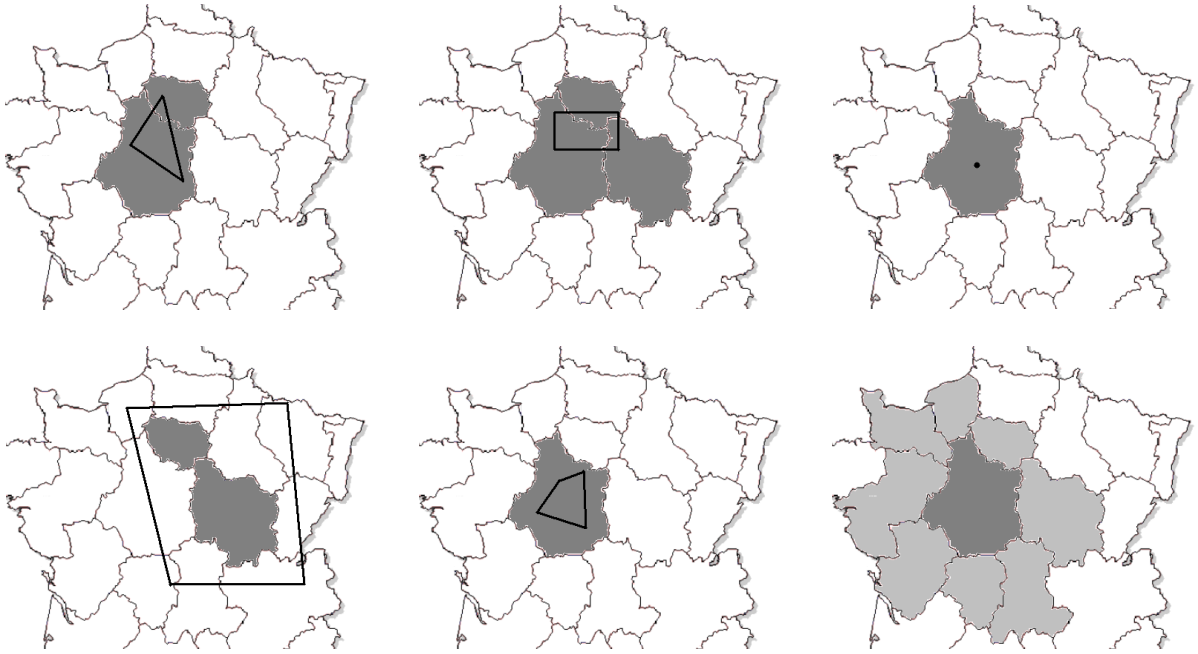


Figure IV.1: Examples of (from left to right first row) region query, range query and point query, and (from left to right second row) of containment query, enclosure query and adjacency query

The main difficulty to index spatial data is that, in a multidimensional space (i.e., two dimensions or more), there is *no total order* between the spatial objects which preserves spatial proximity. This means that there exists no mapping from a multidimensional space to a one-dimensional space such that any two objects that are close in the source space will be closed in the target space. Therefore, multidimensional data can not fully benefit from the reliable one-dimensional access methods such as the B-tree [39] or extendible hashing [40] (see also Section IV.2.2.1). A straightforward idea would be to use a one-dimensional index for each dimension of the indexed space. However, this has been shown to be very inefficient [41], as this type of approach can not fully benefit from a possible high pruning in one dimension since the dimensions are considered separately.

Another issue with spatial data is that the objects can have extremely *diverse and complex shapes*. Since it is more efficient for an index structures to handle similar data objects, a good idea would be to approximate the spatial objects with simpler shapes and then index the resulting objects (see Figure IV.2). Multidimensional access methods use the *minimum bounding box* (*MBB*) (also called the *minimum bounding rectangle* – MBR – for two-dimensional data) to approximate and index a given geometry.

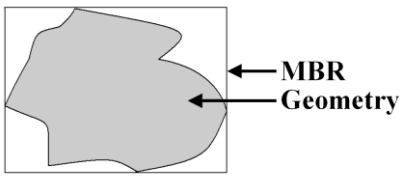


Figure IV.2: Example of a MBR enclosing a geometry

Given the fact that a spatial access method will only consider the MBBs of the indexed objects, the query processing can not be based on the spatial index alone and has to proceed in two steps (see Figure IV.3). In a first phase, a spatial query uses the spatial index to look up

candidate data. This is the *filter step*. Part of the candidate data objects may be directly added to the query result set (e.g., the objects whose MBBs are included in the search space of a range query). For the remaining objects in the candidate set an exact test between the objects' geometries and the query search space needs to be performed. This is the *refinement step*, which requires loading the exact object geometries from the disk. The candidates that verify the test are also included in the final query results.

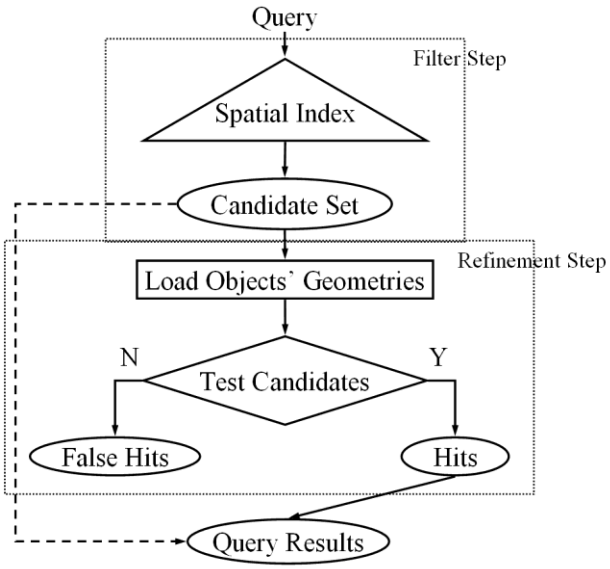


Figure IV.3: Two-step spatial query processing

As for any type of index structures, a main concern of spatial indexing is efficiency. The efficiency of an index structure can be characterized by two aspects [38], i.e., *memory space efficiency* and *time efficiency*. In the case of memory space efficiency, the interest is to reduce as much as possible the index size. This is important for storage saving, but also because a smaller index can be better fitted in the main memory or cache memory, thus improving the overall index performance. In the case of time efficiency, the concern is to reduce as much as possible the query response time. However, the response time for a query depends on various factors such as implementation or hardware configuration. Therefore, a more objective performance measure is needed to evaluate an index structure. The number of disk accesses is a very popular performance measure in the database world. This measure is based on the assumption that the performance of an access method is bounded by the number of I/O rather than the CPU cost (since an I/O operation is at least five orders of magnitude more expensive than a CPU operation). However, in the case of spatial data this assumption is not always true [43], due to important geometric computations that are required by spatial queries. Hence, for spatial index structures both the number of disk accesses and the execution time should be considered for performance evaluations.

IV.2.2 Basic Data Structures

IV.2.2.1 One-Dimensional Access Methods

One-dimensional access methods constitute a basis for multidimensional access methods. There are mainly two types of one-dimensional access methods, i.e., hashing techniques such as extendible hashing [40] or linear hashing [44], [45], and hierarchical access methods such as B-tree [39], [46].

The *B-tree* and its variants hierarchically organize a set of keys. A B-tree is a balanced tree (i.e., all leaf nodes are situated at the same tree level), in which each node is stored as a separate disk page and corresponds to a certain interval of key values. The immediate descendents of an interior node v separate its interval $I(v)$ into mutually disjoint subintervals. The leaf nodes contain pointers to the data objects. Such pointers can also exist in the intermediary nodes depending on the variant of the B-tree. Each B-tree has a global lower and upper bound defined for its nodes that indicate the minimum and maximum number of items in a node. The lower bound does not apply to the root node. The lower bound is intended to increase the storage efficiency of the index structure. If the number of descendants of a node drops below this lower limit, then the node is deleted and its content is distributed among the neighbor nodes. The upper limit is physically determined by the size of a disk page. If the number of values in a node exceeds the upper limit due to an insertion, then the node is split into two nodes and the split is propagated up the tree.

The B-trees are implemented virtually in all modern DBMSs, mostly as B^+-trees (Figure IV.4). The differences from a classical B-tree are the fact that all records are stored in the leaf nodes and that the leaf nodes are linked, which optimizes the searches for range queries. Hierarchical structures are better adapted for highly skewed data and for range queries than the hashing techniques. The performance of hashing methods is conditioned by the distribution of the data and the employed hash function. However, hash indexes outperform the hierarchical counterparts for exact match queries and update operations, if the data has uniform distribution.

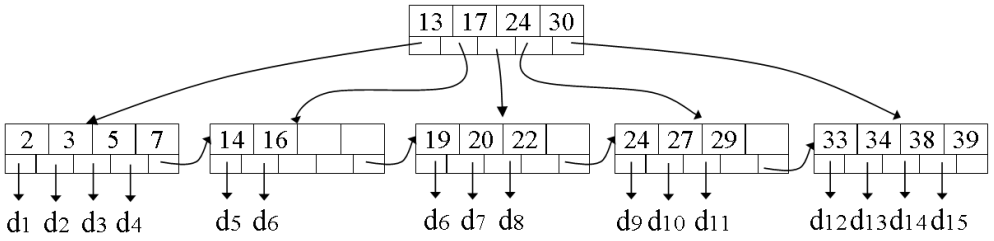


Figure IV.4: A simple B^+-tree example

IV.2.2.2 Main Memory Multidimensional Access Methods

The first multidimensional access methods have been conceived as main memory structures. However, they were adapted later to paged secondary memory in order to be usable with large spatial data sets. Examples of such access methods are the K-D-tree [47], [48], the BSP-tree [49], [50] and the quadtree [51]. Among the three indexes, we concisely present next the quadtree.

The *quadtree* is an access method for *d*-dimensional data, having a structure very similar to the K-D-tree. Since the structure handles only point data, for spatial objects with extent the quadtree indexes their centroids. The main idea is to recursively divide the indexed space by (*d*-1)-dimensional hyperplanes that are iso-oriented. The main difference from the K-D-tree is that instead of using a single hyperplane, the quadtree uses 2^d such hyperplanes. Therefore, the quadtree is not a binary tree as it is the case for the K-D-tree, but each intermediary node has 2^d descendents. The hyperplanes usually separate the search space in equal sized partitions, but this is not obligatory. The decomposition of the search space is recursively continued until the number of objects in each partition drops below a threshold value. Hence, the quadtree is not necessary a balanced tree. Higher density zones correspond to deeper path searches in the tree structure. In the case of two-dimensional data, the space is divided into

four partitions at each level. These partitions are usually named NE, SE, SW and NW (see Figure IV.5).

Given a search region, all partitions that intersect it are recursively visited until the leaf nodes are reached. An example of quadtree having an occupancy of one object per partition is given in Figure IV.5.

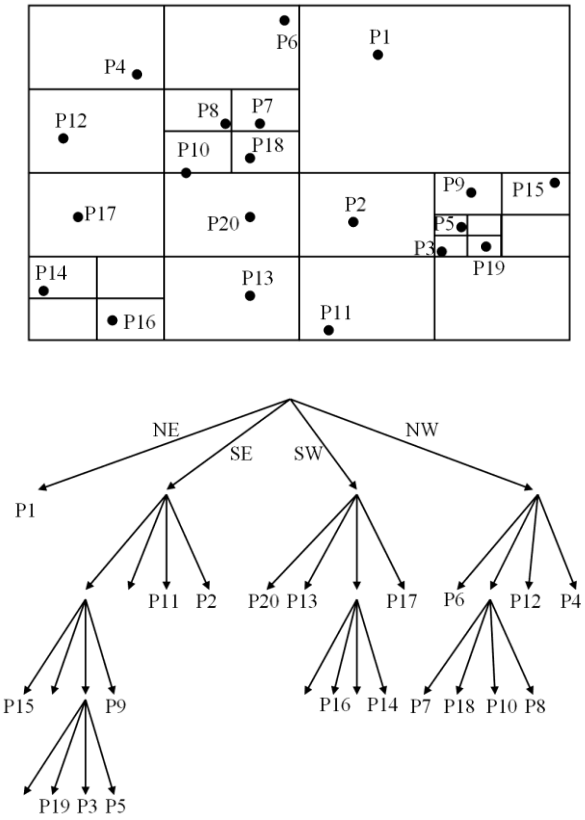


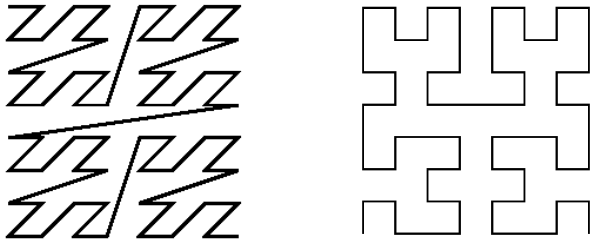
Figure IV.5: Example of a quadtree

IV.2.3 Space-Filling Curves

As indicated in Section IV.2.1, the main difficulty to index spatial data is that there is no total order between the spatial objects that preserves spatial proximity in a multidimensional space. Hence, there is no mapping from a multidimensional space to a one-dimensional space such that any two objects that are close in the source space will be also closed in the target space. Nevertheless, heuristic solutions may be found such that, for two objects that are close in the multidimensional space, there is a high probability that they will be close after the mapping into the one-dimensional space. Then, any one-dimensional access method (e.g. a B^+-tree) could be used to handle the mapped data.

All the proposed approaches start by dividing the indexed space by using a grid. Then, each grid cell is labeled with a unique number representing the position in the total ordered space. A heuristic used for labeling is called a *space-filling curve* and is independent of the existing data items. Finally, the data points are sorted and indexed according to the label of the grid cell that contains them. The main advantage of the space-filling curves is that they are practically insensitive to the number of dimensions of the data space as long as the resulting one-dimensional key can be arbitrarily large.

Several space-filling curves had been proposed, among which we list the row-wise enumeration of the cells [52], the Peano curve [53], also known as the z-ordering [54], the Hilbert curve [55] and the Gray ordering [56]. The comparative experimental tests in [57] conclude that the Hilbert curve and z-ordering are the best choices for spatial data, while the authors in [58] and [59] prefer the Hilbert curve amongst those two (see Figure IV.6).



(a) (b)

Figure IV.6: Two space-filling curves: (a) z-ordering and (b) Hilbert curve

IV.2.4 The R-Tree Family

The R-tree and its variants are probably the most well known spatial access methods today. To some extent, they represent for spatial data what B-trees represent for numeric data. Most commercial or open source DBMSs that offer support for spatial data (e.g., Oracle, IBM DB2, Informix, PostgreSQL) implement an R-tree index.

The structure and the properties of an *R-tree* [60] are similar to those of a B-tree. Given a set of *d*-dimensional items, an R-tree hierarchically organises their minimum bounding boxes (MBBs). Each node ν has associated a *d*-dimensional interval $I(\nu)$ including all the MBBs of its entries. If ν is an interior node then $I(\nu)$ is the MBB containing the MBBs of its descendants. MBBs of nodes situated on the same tree level may overlap. If ν is a leaf node then $I(\nu)$ represents the MBBs of the objects contained by ν. Leaf nodes also include references to the exact spatial description of the indexed objects.

Similar to the B-tree, an R-tree is a balanced tree. Each node corresponds to a disk page. The minimum and maximum occupancy values are also defined for an R-tree. The lower bound prevents the degeneration of the tree and provides efficient storage. Notice that the root node can exceptionally have a lower bound of two items. The upper bound is determined by the size of a disk page. For a lower bound of *m* and *N* index records the maximum tree height is $\log_m(N)-1$.

Given a query having a search interval $I(q)$, all the nodes ν in the tree for which $I(q)\cap I(\nu)\neq\varnothing$ have to be visited recursively down to the leaf nodes. Several paths in the index tree may be required to be traversed to answer a query. This is also true for point queries, since the intervals of the interior nodes situated at the same tree level may overlap. For instance, considering the query *Q* and the index structure presented in Figure IV.7, there two paths in the tree that need to be visited to answer to *Q*, i.e., $A\rightarrow b\rightarrow 7$ and $B\rightarrow e\rightarrow 16$. At the leaf nodes, the exact geometries of the objects whose MBBs intersect $I(q)$ may need to be retrieved to filter out the false hits (see Figure IV.3). In our example, considering that *Q* is a range query, then only the geometry of object 16 needs to be retrieved and tested for intersection with $I(q)$. This is not required for object 7, which can be directly added to the result set, since its MBB is covered by $I(q)$.

To insert a new object *o* in the index, a search operation that considers the MBB of the item (i.e., $I(o)$) is needed first. However, only one path from root to a leaf node is covered by this search. When several intersections between $I(o)$ and the entries of an interior node occur, the search algorithm chooses the descendant that needs the minimum enlargement of its bounding interval to accommodate the object *o*. If several such descendants are found, then one should choose the descendant with the smallest minimum bounding interval. Finally, $I(o)$ is inserted in the leaf node returned by the search. Subsequently, the interval of the leaf node may need to be enlarged and the enlargement has to be propagated upwards. Moreover, if the number of entries exceeds the upper bound, then the leaf node has to be split into two nodes and its entries have to be distributed among those. Since new intervals are created for the nodes, these changes also need to be propagated up the tree.

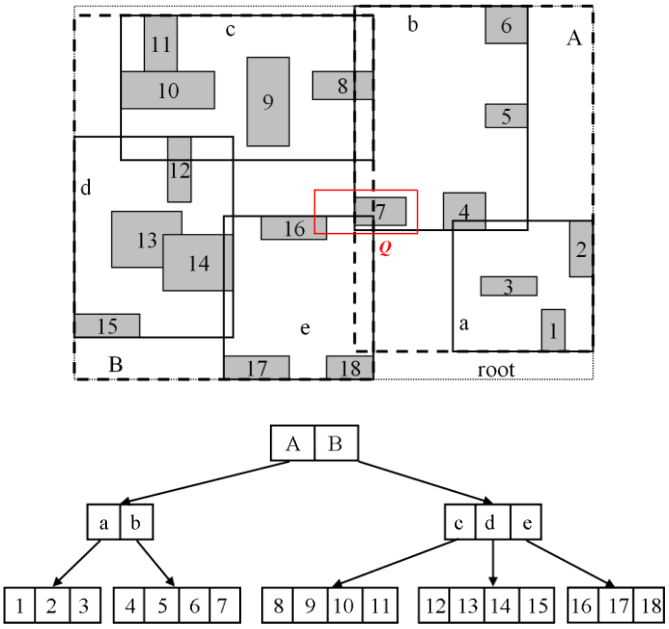


Figure IV.7: Example of an R-tree hierarchical index on MBBs

To delete an object *o*, an exact match query is processed first to determine if the object exists in the data set and, if this is the case, to retrieve the leaf node containing the object's MBB. Then, the object's interval is removed from the leaf node. If the number of the remaining entries is above the index lower bound, then we only need to re-compute the new bounding interval of the leaf node and propagate the (possible) changes up the tree. However, if the deletion causes a node underflow, then several operations are required to be executed. First, the leaf node is copied to a temporary node and then, is completely removed from the index. The changes in the bounding intervals caused by this removal are propagated upwards. Second, all orphaned entries of the temporary node need to be reinserted into the index.

IV.2.4.1 The R$^+$-Tree

An important problem of the R-tree index structure is the *overlapping* between the bounding intervals of the interior nodes situated at the same tree level, which makes that multiple paths need to be traversed to answer a query. Sellis et al. propose in [61] the R$^+$-tree index to deal with this problem. The bounding intervals on the same tree level do not overlap in an R$^+$-tree. If an object's MBB intersects more than one index interval, then the object has to be stored on several disk pages.

In an R^+-tree index, only one path needs to be traversed to answer a point query. Moreover, range queries also tend to be faster since a smaller number of paths in average needs to be traversed to answer a query in an R^+-tree than in the classic R-tree. However, insertion and deletion operations are more costly in an R^+-tree.

IV.2.4.2 The R^*-Tree

Beckmann et al. proposed in [62] another optimized version of the R-tree, which they call R^*-tree. One important observation that they make in [63] is that insert operations in an R-tree can strongly affect the performance of search operations. Thus, they improve the original insert algorithm proposed by Guttman in [60]. A first modification concerns the splitting of a node due to an overflow. Instead of immediately dividing the node, they first reinsert in the index *p* entries contained in the overflowed leaf node. This is called *force reinsert*. The value of *p* may vary, although the suggested value in [62] is 30% of the maximum number of entries in a disk page.

Another modification concerns the criteria used by the insertion algorithm to uniquely identify the leaf node that will accommodate a new object. In the original algorithm, the only objective, when deciding which node to choose among the possible candidates, is to minimize the enlargement of the area of a candidate node. Several objectives are added in [63] for an insert operation. The increase of the overlapping area between bounding intervals of the nodes at the same level should also be minimized. This will decrease the probability of having multiple search paths for a query. Moreover, the perimeter of the bounding interval of a node should be minimized. This means that rectangles having square shapes are preferred since they have the most compact representation. Finally, storage utilization should also be maximized.

The experimental results presented in [63] show that R^*-tree significantly outperforms the R-tree for different data distributions and types of queries. They measured the number of disk accesses for point, intersection and enclosure queries with various sizes. They also tested the performance of spatial joins and insertion operations, and compared the storage utilization for the two index structures. All the tests indicate the R^*-tree as a clear winner.

IV.3 Access Methods on Intervals

Managing intervals is required by many applications in very different domains such as temporal databases (e.g., for transaction time and valid time ranges), spatial applications (e.g., for line segments of a space-filling curve) or engineering databases (e.g., for inaccurate measures with tolerances). Moreover, intervals are also present as *range* (*period*) types (see Chapter II Section II.4) in spatio-temporal databases. Therefore managing intervals efficiently concerns this research area as well.

As for the spatial operators, the access methods that handle intervals need to support several types of range-based queries. Given a reference (time) range, these queries return all the indexed ranges that overlap the given reference. Several particular types of queries are possible for a range query, such as the intersection query, the inclusion query and the containment query. They are illustrated in Figure IV.8.

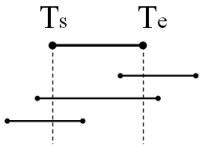
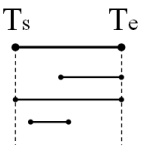
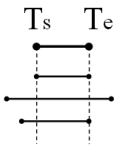
Query type	Intersection	Inclusion	Containment
Examples of input query ranges and possible outputs			

Figure IV.8: Particular types of queries over intervals

A variety of methods has been published concerning the management of intervals in a database, most of them addressing temporal applications. Such examples are the Time Index [64], Time Index$^+$ [65], TP-index [66], Interval B-tree [67], Interval B$^+$-tree [68], MAP21 approach [69] and RI-tree [70]. Most of these methods are based on the B-tree or the B$^+$-tree indexes. Moreover, since they are typically based either on the augmentation of existing indexes (i.e., B-trees) or on the definition of new structures, most of them share the limited support for an integration into existing systems. A few of them (e.g., MAP21 approach and RI-tree) use standard B$^+$-trees, which permits a fast and proper integration with existing RDBMS. In this section we concisely present these two methods.

The *MAP21 approach* [69] is based on the idea of mapping one-dimensional intervals to single one-dimensional values. Then, the mapped values are indexed with a B$^+$-tree. Given a range $V^k = [V_s^k, V_e^k]$, the indexing value is computed by a function Φ as $\Phi(V^k) = V_s^k \cdot 10^\alpha + V_e^k$. α is the maximum number of digits needed to represent any time value in the database. This function maps distinct ranges into distinct points. Moreover, the ordered points in the resulting index represent a lexicographical order of the ranges. The right open-ended ranges are indexed under another tree, which they refer to as an Open-Ended Tree (OET). The OET is a standard B$^+$-tree where the indexed points are the valid start time of the open-ended ranges. Figure IV.9 shows a simple example of a MAP21 tree and the OET indexing a set of time intervals. One important advantage of this approach is that it can be easily integrated in any DBMS which provides B$^+$-trees as indexing structure.

Data Set		
ID	**T_s**	**T_e**
I1	0	3
I2	0	5
I3	0	7
I4	3	8
I5	6	7
I6	4	now
I7	7	now

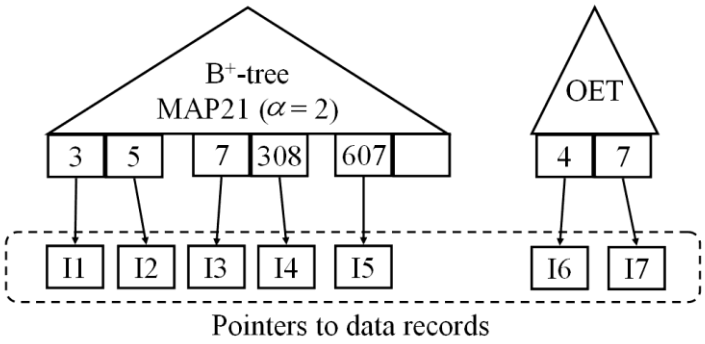


Figure IV.9: Example of the MAP21 indexing approach

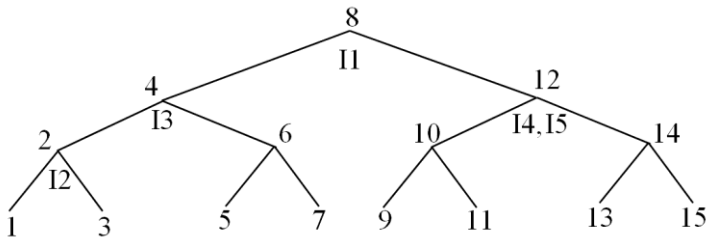
Given a query interval $Q = [T_s, T_e]$, to find all ranges that intersect with *Q*, one needs only to scan the ranges between $[T_s - \Delta, T_s]$ and $[T_e, T_e + \Delta]$, where Δ is the maximum length of the indexed time ranges. Hence, one needs to retrieve all the pointers to data records that correspond to the indexed values situated in the interval $[(T_s - \Delta)10^\alpha + T_s, T_e 10^\alpha + (T_e + \Delta)]$. Similar reasoning can be applied to answer to inclusion or containment type of queries.

Kriegel et al. proposed in [70] the *Relational Interval tree* (RI-tree). The index structure of the RI-tree follows the concept of Edelsbrunner's main memory interval tree [71]. As with the MAP21 approach, the RI-tree can be built in top of the SQL layer of any RDBMS. Moreover, its structure offers optimal complexity for index storage and index operations in terms of disk accesses. The RI-tree requires $O(n/b)$ disk pages of size b for indexing n intervals. Insert or update operations in an RI-tree need $O(\log_b n)$ disk accesses, whereas intersection queries demand $O(h \cdot \log_b n + \frac{r}{b})$ I/O operations for a query that returns r results.

An RI-tree consists of a binary tree and two relational indexes. The height h of the binary tree depends on the maximum bound of the indexed intervals. Thus, a binary tree of height h can index intervals which have the interval bounds covered by the range $[1,2^h-1]$. Notice that the binary tree is never materialized. Only the root value, i.e., 2^{h-1}, is stored in the index metadata. The binary tree is called the virtual backbone of the RI-tree. Each interval of a data set is labeled before insertion into the index with a node value from the binary tree. The node corresponds to the virtual location of the interval in the binary tree. The location assigned to the interval is the first node that is reached when descending the tree from the root, which has a value in the interval. Figure IV.10 (b) shows the virtual backbone and the locations of the intervals in the data set presented in Figure IV.10 (a). Since the maximal interval bound is 15, the binary tree has four levels.

Data Set		
ID	**T_s**	**T_e**
I1	3	9
I2	1	3
I3	4	6
I4	12	15
I5	10	13

(a)



(b)

***lowerIndex* (*node, lower, id*)**

(2, 1, I2)
(4, 4, I3)
(8, 3, I1)
(12, 10, I5)
(12, 12, I4)

***upperIndex* (*node, upper, id*)**

(2, 3, I2)
(4, 6, I3)
(8, 9, I1)
(12, 13, I5)
(12, 15, I4)

(c)

Figure IV.10: Example for an RI-tree. a)five intervals. b)virtual backbone and registration positions of the intervals. c)resulting relational indexes *lowerIndex* and *upperIndex*

The RI-tree materializes two relational indexes indicated as the *lowerIndex* and the *upperIndex*. The lowerIndex contains for each indexed interval its node label in the virtual backbone (i.e., *node*), its lower bound (i.e., *lower*) and its identifier (i.e., *id*). Similarly, the upperIndex consists of the node label, the upper bound and the *id*. Figure IV.10 (c) presents the *lowerIndex* and the *upperIndex* for the example data set. For instance, the interval identified by I3 is registered at the node 4. Therefore, two entries are added for this interval to

the *lowerIndex* and the *upperIndex*, i.e., (4, 4, I3) and (4, 6, I3) respectively. The reader may refer to [70] for the aspects regarding the index operations or other details related to the RI-tree index.

Beside the above listed methods for one-dimensional intervals, even the multidimensional access methods such as the R-tree or its variants (i.e., R^+-tree and R^*-tree) (see Section IV.2.4) may be employed to index time intervals. However, this is not recommended since the long time intervals and the significant degree of overlapping between time intervals, which usually occur in temporal data sets, can induce severe performance problems [72].

IV.4 Spatio-temporal Access Methods (STAM)

IV.4.1 Specifics of Spatio-Temporal Data, Querying and Indexing

We reviewed in the previous two sections some of the proposed access methods for spatial data and temporal (range) data. The multitude of proposed indexes testifies that the support for fast search at the physical level is critical to maintain a scalable database system. Of course, this observation remains true also for spatio-temporal data. Hence, robust indexing techniques are needed in a spatio-temporal DBMS for fast retrieval of spatio-temporal objects. Various spatio-temporal queries require the support of efficient access methods. A straightforward approach would be to consider the temporal dimension of the spatio-temporal data as a supplementary spatial dimension. One could then use a multidimensional access method such as the quadtree or the R-tree (see Section IV.2). However, considering spatio-temporal data as 3D (2D + time) spatial data and then using spatial access methods to index it, has several shortcomings. A spatio-temporal access method should take into account the specificity of spatio-temporal data and querying, as will be discussed next. The spatio-temporal data has several particularities [73]:

i) Supported data sets: Spatio-temporal data include all kind of moving entities as well as the changes for these entities. Two base abstractions for spatio-temporal data are *moving region* and *moving point*. Applications that handle *moving region* data can also manage *moving point* data since the points can be treated as degenerated regions. However, optimized solutions may be found for *moving point* data since their representation is simpler.

The data model has also an important impact on the indexing method. The trajectory of a moving object is obtained by periodic sampling. Some works model the trajectory as a sequence of points, i.e., each database record has the format (x, y, t_s, t_e). The MO is considered static within each time interval (t_s, t_e). Other works use (linear) interpolation between positions reported by two consecutive updates, in order to bring closer the model to the real object movement.

ii) Approximations of moving objects: In spatial databases, the R-tree and its variants index the minimum bounding boxes (MBBs) of a set of spatial objects (see Section IV.2.4). As indicated in the introduction of this section, a direct approach to index spatio-temporal data would be to employ a three-dimensional R-tree. However, the nature of the spatio-temporal data makes that the MBB of a spatio-temporal entity contains a vast amount of dead space. For instance, Figure IV.11 presents the trajectory of a moving point and its corresponding MBB. The MBBs representing spatio-temporal objects can be very large. It will be unwise then to index the MBBs of spatio-temporal data, since this may lead to high overlap and dramatic degradation of the index performance. Therefore, alternative representations such as trajectory decomposition schemes should be investigated.

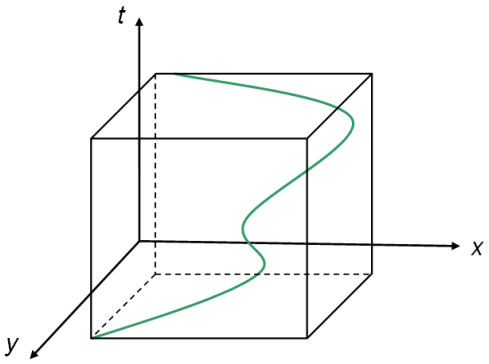


Figure IV.11: The MBB of a moving object occupies a large portion of the data space

iii) Time dimension peculiarity: Since time is usually considered continuous, i.e., isomorphic to reals, there is a certain similarity between the time dimension and the dimensions of the space. However, for any observed spatio-temporal entity, time is the only attribute taking monotonically increasing values. This peculiarity of the time dimension suggests that the time dimension cannot be considered as just another spatial dimension. This is another reason not to use multidimensional access methods to index spatio-temporal data. Instead, appropriate access methods that take into account the specifics of the temporal and spatial dimensions should be devised.

iv) Support of specific spatio-temporal queries: Any access method is designed to support a certain type of operations. Spatio-temporal databases inherit the types of operations addressed in spatial databases. There are three main spatio-temporal operations: selection, join, and nearest neighbor queries. Besides, typical queries that are supported by a STDBMS include time slice queries, e.g., *"Find all objects that cross a certain area at time t"* and range (or window) queries *"Find all objects that cross a certain area in the time interval* $[t_1, t_2]$*"*.

v) Querying about the past, the present and the future: Time slice queries and window queries may ask about the past, current, or future times. Some queries are concerned only with the past, e.g., trajectory queries *"What was the location of a certain moving object an hour ago?"* Other queries are concerned only with the future, e.g., *"Find the objects that will enter the city center in a certain time interval"*. When querying about the past the entire dataset is known in advance, i.e., we deal with past trajectory data. In this case reducing the retrieval costs in large data sets is important. When querying about present and (near) future movements of MOs, the objective is minimizing update and retrieval costs. Most spatio-temporal access methods either index the past (i.e., index historical spatio-temporal data) or keep track of the current status of spatio-temporal data.

vi) Free vs. constrained movement: The type of movement is also an important factor when indexing spatio-temporal data. Most moving objects indexes consider that the objects are moving freely in the space. However, in several real-life applications, the object movement is constrained. This leads to a specific model of data representation and specific querying (see Section II.4.1.2). This aspect must also be considered by the indexing techniques.

vii) Indexing by data partitioning vs. indexing by space partitioning: There are two types of approaches in the existing access methods for spatio-temporal data. The first type is based on data partitioning. Such approaches use an R-tree-like structure to organize the objects. The index is a hierarchy of object MBBs. At the leaf level, a MBB of a node groups together a

number of objects that are spatially close. If too many objects are contained in a leaf node, i.e., the node can no longer be held by a disk page, a split occurs. Also, two neighboring nodes can be merged if their objects can be stored on a single disk page. Such an approach partitions the data, since there is an upper bound for the number of items in a leaf node. Dense regions will have many small MBBs, whereas sparse regions will be covered by a few large MBBs.

The approaches in the second category are based on space partitioning. First, the space is partitioned using a grid. Then, the objects are indexed by using their grid cell identifier with a B^+-tree. This type of approach has two important advantages over the first type. Such index structures can be integrated easily into any DBMS, since they depend only on the B^+-tree and the grid index. These are well established structures in virtually any DBMS. Moreover, the index operations in a B^+-tree (e.g., search, insertion, and deletion) can be performed more efficiently than in an R-tree. The difference in performance between the two structures is even more significant in a concurrent environment. This represents an essential aspect for real-time applications, where frequent queries and updates arrive simultaneously.

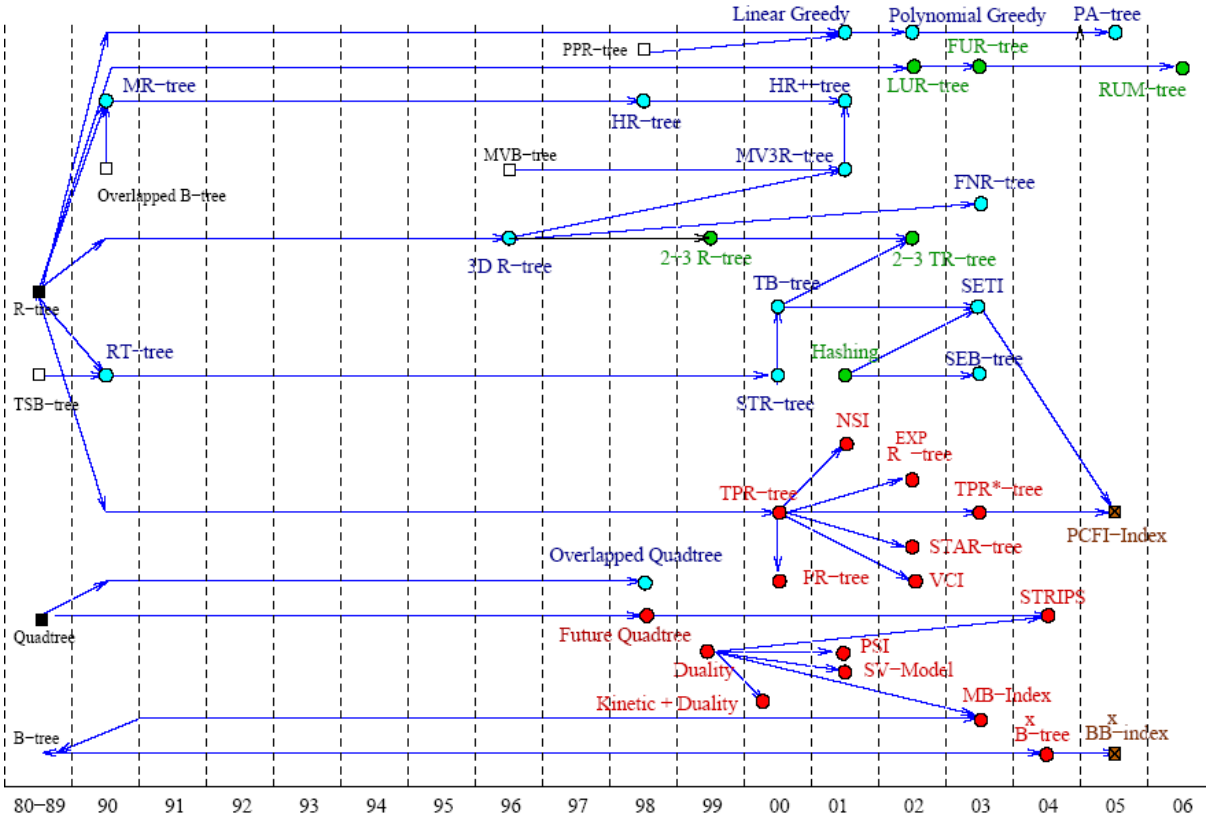


Figure IV.12: Spatio-temporal access methods (blue → past; green → present; red → future; brown → all) [74]

An impressive number of spatio-temporal access methods have been proposed in the last decade. Figure IV.12 [74] gives the evolution of spatio-temporal access methods (up to 2006) with the underlying spatial (black square) and temporal structures (white square). The lines in the figure indicate the relation between a new proposed spatio-temporal index structure and the original structure that is based upon [74]. The blue dots represent the spatio-temporal indexes for the past, the green stands for indexing current positions, the red indicate indexes for current and (near) future time, whereas the brown gives the access methods that cover all

moments in time. Most of these methods focus on range queries and on indexing only moving point[5] objects. We concisely present a few methods in the next sections.

IV.4.2 Indexing the Past

There are mainly three categories of spatio-temporal indexing schemes for historical data. The first category augments the temporal aspect into already existing spatial access methods. Examples of indexes in this category are the RT-tree [75], 3D R-tree [76] and STR-tree [77]. The second category manages both the spatial and temporal aspects into one structure. In this category we find the HR-tree [78], HR+-tree [79], MV3R-tree [80] or the PPR-tree [81]. The approaches in the third category index mainly the temporal dimension, while treating the spatial dimension as second priority. The index methods that we list in this category are the TB-tree [82], SETI [83], SEB-tree [84] and PIST [85]. All these methods consider that the moving objects move freely in the two-dimensional space. From this multitude of methods we shortly present next SETI [83] and PIST [85] since these access methods are related to some extent to our index proposal in Chapter V.

SETI, a *Scalable and Efficient Trajectory Index* [83], is an access method that has a two-level structure. Each level deals separately with the spatial and with the temporal dimensions of spatio-temporal data. This makes search and insert operations very efficient. SETI can handle historical *trajectories* of MO. A trajectory is composed of a sequence of *segments*. Each segment is computed as the line connecting two consecutive updates, i.e., a three-tuple $u_i(x_i, y_i, t_i)$ where x_i and y_i represent the coordinates in the Euclidian space, at time t_i. SETI focuses on range queries, i.e., select all objects within a given area and a given time period, and time-slice queries, i.e., select all objects within a given area at a given time instant.

At the first level, the space is partitioned by a number of non-overlapping cells (see Figure IV.13). A trajectory segment can only be located in one cell. Therefore, if a segment in the original data set extends over several cells, it is split at the boundary of each cell. This, however, leads to an increase in the data set size.

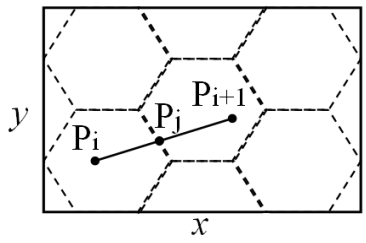


Figure IV.13: Spatial partitioning and segment splitting in SETI

At the second level, the segments contained by each cell are stored together in one or several data pages. A data page can only contain data segments from a single cell. For each data page its *lifetime* is then computed. The lifetime represents the minimum time interval that completely covers the time-spans of all the segments in a data page. Finally, one 1D R-tree is built on the lifetime values of all the data pages which are logically associated to a spatial cell. These indexes are sparse since there is only one entry for a data page, instead of one entry for

[5] In the rest of the chapter we use the terms moving objects and moving point objects interchangeably

each indexed data segment. This decreases the index overhead and improves insertions. Moreover, SETI can be easily implemented in all the DBMSs that provide an R-tree index.

The search algorithm for a range query executes in four steps. *i) Spatial Filtering*: The list of the cells that intersect with the spatial region of the query is generated in this phase. *ii) Temporal Filtering*: For each cell in the candidate list, the corresponding index is search to find the data pages whose lifetimes overlap the temporal predicate of the query. A list of candidate data pages is produced at the end of this phase. *iii) Refinement Step*: The segments in all candidate data paged have to be verified for exact intersection with the query window. However, not all the data pages have to be (completely) processed in this phase. If a data page corresponds to a cell that is covered entirely by the spatial query predicate, then only the temporal intersection between the segments and the query needs to be checked. Moreover, if the lifetime of such a data page is included in the time interval of the query, then all the segments in the data page qualify for the result and no additional processing is needed. *iv) Duplicate Elimination*: This step is necessary for queries that require the unique trajectory identifiers and not the trajectory segments intersecting the query window. Since multiple segments in the result can belong to the same trajectory, the duplicates must be eliminated. The reported experimental results in [83] show that SETI outperforms the TB-tree [82] and the 3D R-tree [76] in both querying and data insertion.

The general framework of the *practical index for spatio-temporal* (*PIST*) data [85] uses a technique similar to SETI. There are, however, many differences between PIST and SETI both in the data model and index structure. The data model employed by PIST can be viewed as a step-wise interpolation instead of a linear interpolation between two consecutive observations of a moving object. Thus, an object is considered to be stationary as long as its position is not updated in the database. Therefore, the indexed data records are 2D positions associated with a time interval, i.e., (oid, x, y, t_s, t_e), where *oid* identifies an object, (*x*, *y*) are spatial coordinates, and $[t_s, t_e)$ indicate the open-ended non-null interval during which an object remained at position (*x*, *y*).

Similar to SETI, PIST separates the indexing of the spatial and temporal dimensions. At the spatial level, it partitions the data space using a grid. Then, it creates a B^+-tree for each grid cell to index the time intervals of all the data records situated inside that cell (cf. Figure IV.14). Since PIST handles historical data, the space partitioning is static and it is computed before the index creation. An important feature introduced by PIST, is that the number of grid cells is determined by a cost model depending on the data set size and an expected query size. Moreover, the grid granularity can vary in different zones of the data space conditioned by the distribution of the data. PIST also considers spatio-temporal range queries, the query processing being similar to the search algorithm of SETI.

Although quite similar to the SETI approach, PIST introduces several new and important features. PIST proposes a cost model to determine the number of grid cells that minimizes the number of disk accesses of a query, assuming a uniform distribution of the data and an average query size. Based on this cost model, Botea et al. propose a heuristic to adapt the spatial partitioning to the data distribution. They also propose an optimal splitting of the long data records to improve the processing time of the temporal predicate of a query. Finally, PIST relies on the efficient and ubiquitous B^+-tree, which makes it easy to integrate in virtually any RDBMS.

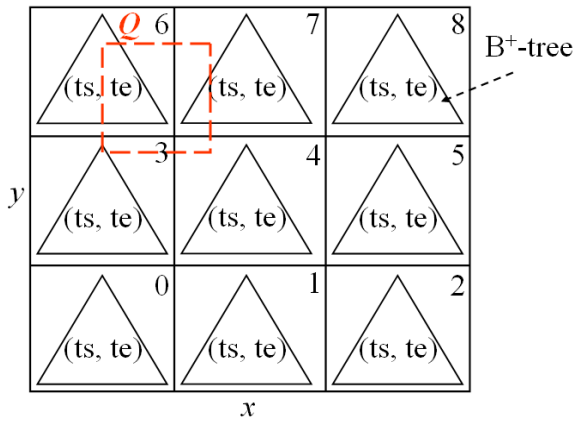


Figure IV.14: PIST's approach for a 3 × 3 regular grid

IV.4.3 Indexing Current and Future Positions

Complementary to the above listed methods, there are many works that deal with indexing the current and anticipated future positions of moving objects. In this context, besides optimizing the retrieval cost for a query, minimizing the update cost is a major concern. Frequent updating is not feasible for a large set of objects. Therefore, a trade-off between communication cost and location error has to be accepted (see Section II.3.1). Moreover, to predict the future positions of moving objects and to minimize the number of updates, one needs to store extra information (e.g., the velocity). A moving object in the *d*-dimensional space is characterized by its location $\vec{x}_{ref} = (x_1, x_2, ..., x_d)$ at the reference time t_{ref} and by a velocity vector $\vec{v} = (v_1, v_2, ..., v_d)$. A future location $\vec{x}_t$ of the moving object at any instance time $t > t_{ref}$ can be computed by $\vec{x}_t = \vec{x}_{ref} + \vec{v}(t - t_{ref})$.

A number of spatio-temporal indexes have been proposed, such as R-tree-based indexes, e.g., the TPR-tree [86], the TPR*-tree [87], the STAR-tree [88], and the R^{EXP}-tree [89]; the quadtree-based index STRIPES [90], and the B^+-tree-based B^x-tree [91] and ST^2B-tree [92], to name but a few. In the rest of this section we choose to summarize the structure of the ST^2B-tree index.

The *ST^2B-tree* [92] is a self-tunable spatio-temporal B^+-tree index for current and anticipated positions of moving objects. The index structure is practically a B^x-tree [92] enriched with a self-tuning mechanism. An important problem that needs to be considered in a database that keeps track of the current positions of moving objects is the high dynamicity of the data set. Since the observed objects move continuously, there can be significant changes in the data distribution over time. Thus, the result of a same query concerning one region in the data space can vary greatly over time. Therefore, traditional static indexes may not perform well in the long run. Index structures with self-tuning mechanism capable of keeping up with the changes in data distribution and query load are suitable in this case. One of the main contributions of the ST^2B-tree index is that it re-examines the self-tuning mechanism (in database systems) in the context of dynamic (spatio-temporal) databases.

The ST^2B-tree is built on the B^+-tree without requiring any modifications for the search or update operations of the underlying structure. For each spatio-temporal object in the data set, a one-dimensional key is computed. The keys are then indexed by a B^+-tree. A key consists of two components, i.e., a temporal part KEY_{time} and a spatial part KEY_{space}. The underlying B^+-

tree of the ST^{2}B-tree is logically divided in two subtrees, BT_0 and BT_1, from a temporal point of view (Figure IV.15). Each subtree has associated a time interval for a time period *T*. Thus, the time ranges covered by BT_0 are [2*iT*, (2*i*+1)*T*), whereas those covered by BT_1 are [(2*i*+1)*T*, (2*i*+2)*T*) respectively, where *i* = 0, 1, 2 . . .As time passes the two subtrees roll over alternatively and the value of *i* increases. The index uses the roll over to self-adjust with time. Given an update coming from an object at time t_{up}, the object will be inserted in the subtree covering t_{up}. Hence, the temporal component $KEY_{time} \in \{0,1\}$ identifies the subtree that will index the moving object.

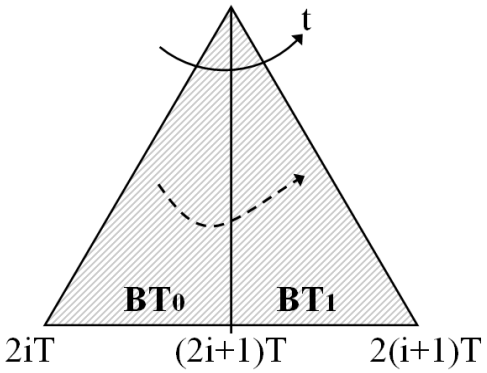
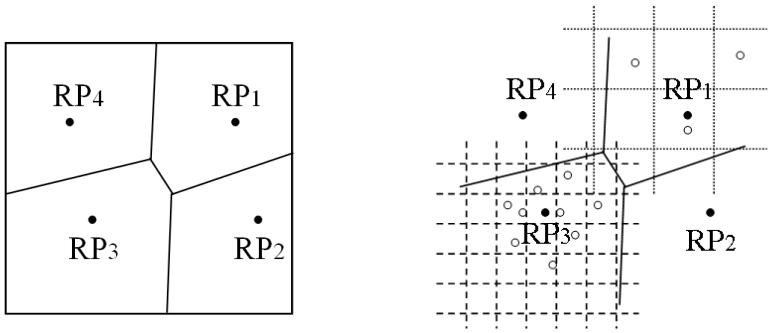


Figure IV.15: The ST^{2}B-tree rotates with time

The spatial component KEY_{space} is computed based on a two-level partitioning of the space (Figure IV.16). At the top level, the 2D space is partitioned into *n* disjoint regions that form a Voronoi Diagram (Figure IV.16a). Hence, a set of *n* reference points $\{RP_0, RP_1, ..., RP_{n-1}\}$ is used to partition the data space into n disjoint regions $\{VC_0, VC_1, ..., VC_{n-1}\}$ in terms of the distance to the reference points. The reference points are computed based on the spatial distribution of the moving objects at a certain time instant.



(a) First-level space partitioning (b) Second-level space partitioning

Figure IV.16: ST^{2}B-tree: Space partitioning

At the second level, each region is partitioned with a uniform grid. The grid granularity is specific to each region (Figure IV.16b) depending on the object density of the region. Thus, more sparse regions will have a coarser grid than the denser regions. A cost model permits to automatically compute the grid granularity given the density of the data in a region such that the number of disk accesses for a search query is minimized.

After each *T* time when a new (logical) rotation of the B$^+$-tree occurs, the spatial partitioning is recomputed based on the current spatial distribution of the tracked MOs. In this

way, the index configuration keeps up with the modifications in data distribution and density. This process can significantly improve the index query performance and especially update performance in the long run. Querying and updating in ST^2B-tree is similar to the other related access methods. The reader can refer to [92] for additional details.

IV.4.4 Generating Test Data for Benchmarking

In this chapter we reviewed some of the existing work on spatio-temporal indexing. Virtually any proposed access method is evaluated and often compared with other (similar) existing approaches. The experimental evaluation is necessary to expose the index behavior under certain test conditions and to compare its performance with some other techniques. However, real data sets are not frequently available and are difficult to obtain. Hence, a good idea would be to design generators for test data and use them to create synthetic data sets whenever needed.

Such data generators have been developed in the research area of moving objects databases. Examples of generators for objects moving freely in the Euclidian space are GSTD [93] and G-TERD [94]. GSTD is designed to generate *moving point* data and *rectangles* with various distributions according to the users' requirements. G-TERD is a generator for *moving region* data. The time-evolving regions are modeled as a set of overlapping rectangles. Several parameters can be tuned by users, such as data space length on each axis, time length, minimum and maximum speed of an object, and so on.

Brinkhoff developed a generator for objects moving in networks [95]. The generator considers several aspects to simulate realistic car traffic. For instance, an object moves on the shortest path between its start point and its destination. Several classes of moving objects can be generated, each class corresponding to some moving speed value. The speed is automatically decreased when the number of objects situated on the same network edge exceeds a certain threshold value. Also, weather conditions can be simulated by creating random moving regions that impact the speed of the moving objects that intersect these regions.

The above mentioned generators are available for download on the web (*GSTD* [96], *G-TERD* [97], Brinkhoff [98]). In the last years, more and more real spatio-temporal data are becoming available for free use. These data are collected during different research experiments such as animal or vehicle tracking. Pfoser published several real data sets on his website [99].

IV.5 Indexing the Past Trajectories of Moving Objects in Networks

In the previous section we reviewed some of the indexing approaches for moving objects, in this vast domain of the spatio-temporal access methods. Most of these approaches, which include all the above listed methods, assume free movement of the objects in the two-dimensional space. In many cases, however, objects move in spatially embedded networks. Then, the movement is constrained to a network space instead of the Euclidian space. A network space is considered to be 1.5 dimensional in the literature. This is because in a network space, there is a discrete dimension, e.g., a certain road, and a continuous dimension, e.g., the relative position along a given road (see Section II.4.1.2 in Chapter II).

The constrained movement requires a specific data representation but also specific queries. For example, the queries' criteria may refer to the network instead of an arbitrary region in space. As we emphasized in Chapter III, in this work we are focusing on constrained movement. Hence, we present in this section the existing methods for indexing the (past) trajectories of moving objects in networks. To the best of our knowledge, there are only three access methods that consider constrained movement of moving objects and that are optimizing range queries. They represent trajectories with reference to a network, i.e., indicate the relative position of the moving objects on network edges. The main idea in those approaches is to decompose a three-dimensional problem into two sub-problems with lower dimensions and then use a combination of two levels of R-trees to index the trajectories (see Figure IV.17). We present these methods in this section.

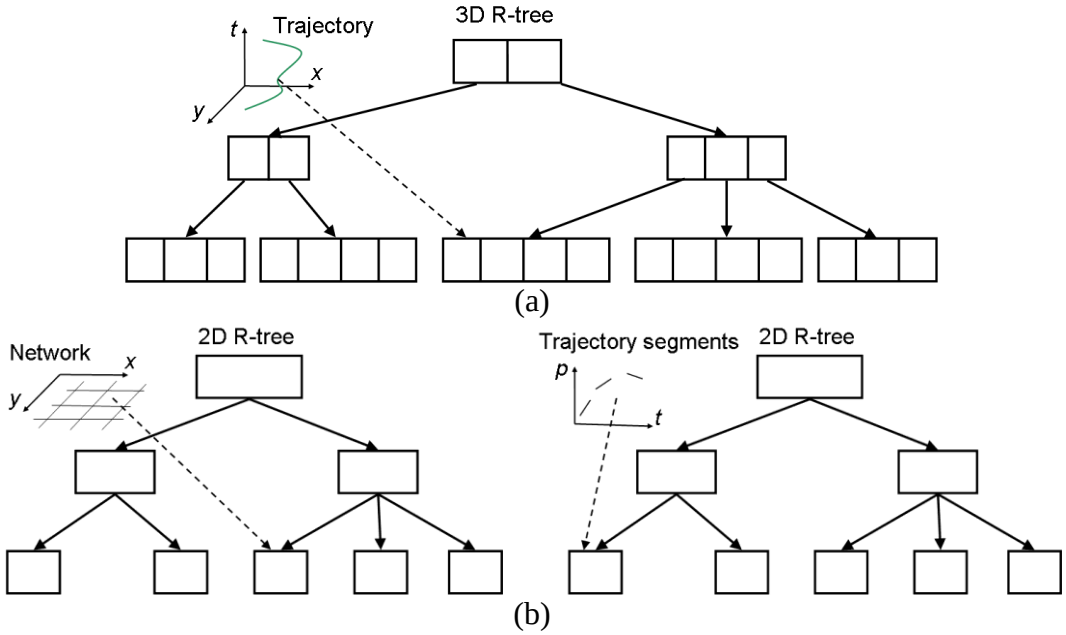


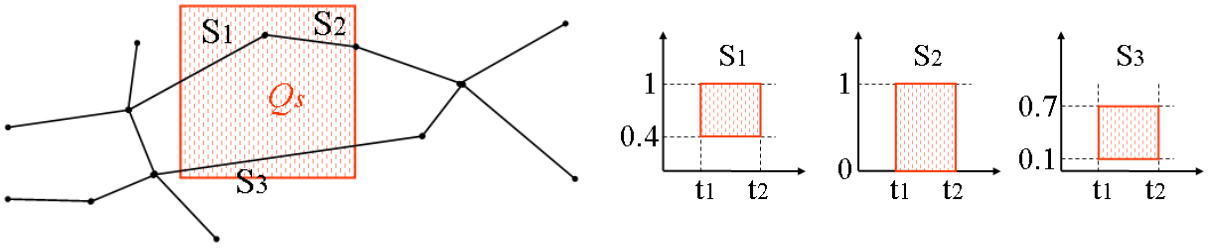
Figure IV.17: Replacing one 3D index (a) with two 2D indexes (b)

The *Fixed Network R-Tree* (*FNR-tree*) was proposed in [100] to solve the problem of indexing objects moving in fixed networks. The FNR-tree has a two level structure. On the first level, there is a 2D R-tree that indexes the edges of the road network. The network model considers a separate edge for each line segment in the geometrical representation of the network (see also Section V.2.1 in Chapter V). A leaf node in the 2D R-tree will contain several entries of the form (*EdgeId*, *MBB*, *orientation*) representing the edge's identifier, minimum bounding box and orientation. Since each edge corresponds to a line segment, the *orientation* is a boolean value for the two possible diagonals of a rectangle (i.e., the MBB). At the second level, a 1D R-tree is created for each leaf of the 2D R-tree. The 1D R-tree indexes the time intervals corresponding to objects moving on the line segments situated in the leaf of the 2D R-tree. A leaf entry in a 1D R-tree has the form (*MOId*, *EdgeId*, $T_{entrance}$, T_{exit}, *Direction*). $T_{entrance}$ and T_{exit} represent the times a moving object entered and left a certain edge. The last attribute indicates in which direction the edge was traversed.

A new entry is inserted into the index each time a moving object traverses completely an edge. The first-level tree is used to find the traversed edge. Then, a time interval, i.e., the edge traversal time, is inserted in the corresponding 1D R-tree. Since the updates arrive in chronological order, the insertions in the second-level indexes are always located in the right-most leafs of the trees.

Given a spatio-temporal query window $(x_1, x_2, y_1, y_2, t_1, t_2)$, the search algorithm proceeds in three steps. In the first step, the top 2D R-tree is used to identify the list of network edges that overlap the rectangle (x_1, x_2, y_1, y_2) (Figure IV.18(a)). The result is temporary stored in main memory. In the second step, the bottom 1D R-tree corresponding to leaf nodes that contain the intersected edges, are searched for intersection with the interval (t_1, t_2). A list of candidate trajectory segments is generated in this phase. Finally, the third step consists in refining the candidate list by eliminating the trajectory segments that are fully outside the query window. To achieve this, the objects' three-dimensional movements are reconstituted by using the coordinates of the edges previously loaded in main memory.

A major disadvantage of the FNR-tree is its limitation in trajectory modeling. Since only the time intervals are stored in the bottom 1D R-trees, it is assumed that the objects cannot stop, change speed or direction in the middle of an edge. Moreover, the network model creates an edge for each existing line segment. This may lead to a high number of entries in the index structure, since a new update is issued each time an object traverses an edge.



(a) (b)

Figure IV.18: Example of query mapping from the 2D space (a) to the network space (b)

The second index structure was proposed in [101] by Pfoser and Jensen. For simplicity we will refer to this approach as *PJ-tree* in the rest of the thesis. This approach uses two 2D R-trees: one for indexing the road edges and the other for accessing the 2D transformed trajectory segments (cf. Figure IV.17). First, the network edges are sorted and numbered by using a Hilbert curve. Then, the same mapping is applied to trajectories. The 3D coordinates (*x*, *y*, *t*) of a trajectory are mapped into a 2D coordinate space (*p*, *t*). The position *p* is computed as the sum between the edge identifier (obtained after the network mapping) and the relative position on the edge. Similar to all approaches, a linear interpolation is employed between two consecutive trajectory points. Finally, the mapped trajectory segments are indexed by a single 2D R-tree.

The search algorithm requires two steps to process a spatio-temporal range query $(x_1, x_2, y_1, y_2, t_1, t_2)$. This is similar to a search in the FNR-tree. The spatial part $Q_s = (x_1, x_2, y_1, y_2)$ of the query is mapped to a set of edge intervals, in the first step. The edge intervals correspond to the intersection between Q_s and the network edges (Figure IV.18(a)). This is accomplished by applying Q_s to the index that indexes the network edges. Additionally, the obtained edge intervals are "lifted" into the time dimension with the time interval (t_1, t_2) of the query (Figure IV.18(b)). Therefore, the set of edge intervals is transformed into a set of rectangles representing the final mapped query (see Figure IV.18). In the second step, the 2D R-tree indexing the mapped trajectory segments is employed to find the segments that intersect with each rectangle in the mapped query. In [101], it is shown that this method can outperform the direct approach that uses a 3D R-tree to index the trajectories.

Compared to the FNR-tree, the PJ-tree eliminates the problem of modeling the trajectories, i.e., a moving object can stop and can change speed or direction in the middle of an edge. However, since it makes the same choice of associating an edge to each line segment, it shares the same disadvantage with the FNR-tree, i.e., a high number of entries in the index structure. Another shortcoming of the PJ-tree is coming from the fact that is uses a single 2D R-tree to index the trajectory segments, whereas the FNR-tree uses a 1D R-tree for each node of the top 2D R-tree that indexes the network edges. This can make the second step of the query evaluation more expensive for the PJ-tree.

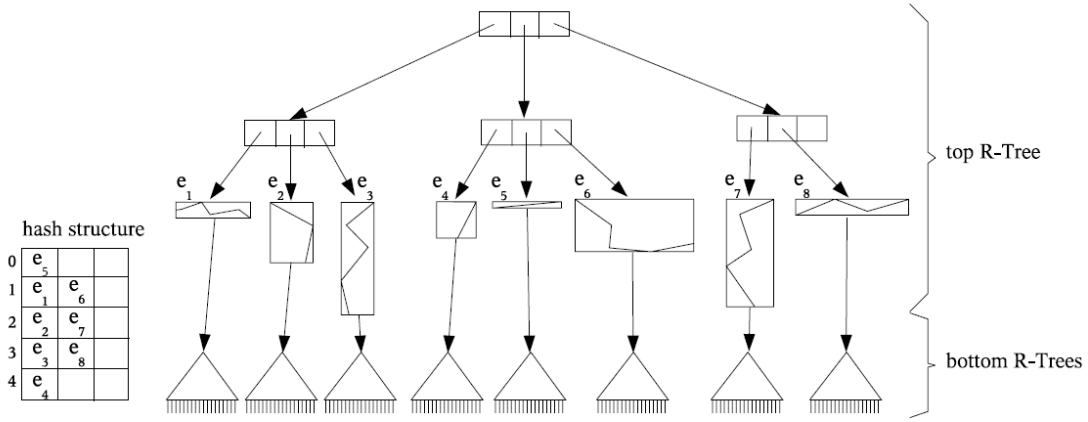


Figure IV.19: Example of the MON-tree index structure [102]

These limitation are addressed in [102] by the *MON-tree*. The MON-tree (Figure IV.19) is composed of a 2D R-tree (the top R-tree) that indexes the network edges and a set of 2D R-trees (the bottom R-trees) that index the object movements along the edges. The number of the bottom R-trees is equal to the number of network edges, i.e., there is one 2D R-tree for each edge in the network. An additional hash structure used to map each edge to its corresponding tree helps speed up insertions. As for the before mentioned methods, given a 3D spatio-temporal query, the top R-tree is used to find the precise intersection between the spatial part of the query and the network (Figure IV.18). Based on this intersection, a set of sub-queries is generated for each intersected part of each involved edge. Then, the corresponding bottom R-trees are accessed in order to respond to the sub-queries.

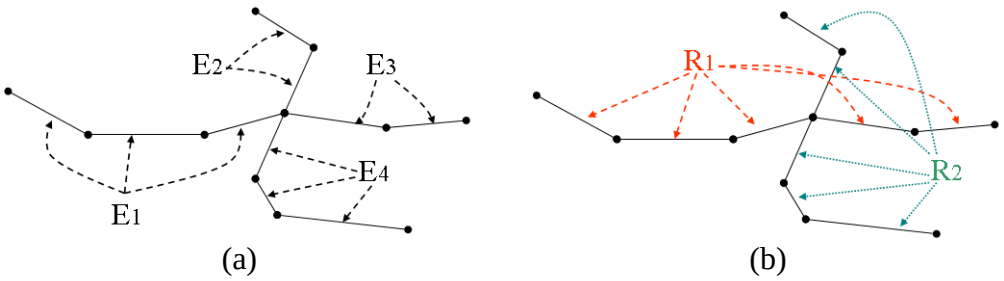


Figure IV.20: Example of a network (a) in the edge oriented model and (b) in the route oriented model

MON-tree can handle two network models: an edge oriented model and a route oriented model (Figure IV.20). In the edge oriented network model, each road corresponds to the polyline between two road junctions, whereas in the route oriented network model the roads can extend over the road junctions. We formally define the network models in the next chapter. The experimental evaluation in [102] of the MON-tree against the FNR-tree shows

that the first method always outperforms the second. Moreover, the MON-tree on a route oriented network model shows better results.

IV.6 Indexing Spatio-Temporal Operators

We introduced in Section III.5.4 the main operators that need indexing in a spatio-temporal database. These operators are *spatial*, *temporal* and *on-value* predicates, or predicates that combine two of the three possible dimensions (i.e., *spatio-temporal*, *on-value spatial* and *on-value temporal*). The access methods that we covered in this chapter can be used to index all the listed operators.

Hence, for *spatial* indexing, the R-tree index family (Section IV.2.4) appears to be the most appropriate (and most adopted) solution. Also, another important argument in his favor is that the R-tree index has made his way into most commercial DBMSs that have spatial support.

The *temporal* and *on-value* operators require interval indexing. The intervals correspond to time periods or range values. The B^+-tree is a straightforward and efficient solution, being available in virtually any DBMS. However, more refined solution can be employed such as the B^+-tree-based approaches, e.g., MAP21 and RI-tree (Section IV.3).

To index *spatio-temporal* data, we need first to position the targeted application type from a temporal perspective and from a spatial perspective. From the temporal perspective, the application can deal either with historical (past) data or with real-time data. From the spatial perspective, the movement can be modeled as free or constrained. In our work, we consider historical data issued by objects that travel in networks. Therefore, the methods that can be used in this case are the ones presented in Section IV.5.

Given that we consider time to be linear and continuous, i.e., isomorphic to the real numbers, there is a certain similarity between the temporal dimension and the ranges over the base types (e.g., real numbers). Hence, the methods used to index the spatio-temporal operators can also be employed for the *on-value spatial* operator.

Finally, the last type of operator is the *on-value temporal* predicate. The data needed to be indexed in this case, can be seen as two-dimensional data (where one dimension is the time and the other one is represented by ranges over base types). Therefore, a straightforward solution would be to use a multidimensional access method such as a 2D R-tree.

The research in the area of spatio-temporal indexing was and continues to be prolific. For the type of application that we study (i.e., managing mobile sensor data), there are many techniques that cover well the demand of indexing such data in all its dimensions. However, whereas some problems have been intensively and extensively studied, there are some aspects that, in our view, still need to be studied. An example is the spatio-temporal indexing for constrained movement. Moreover, moving objects equipped with sensors produce additional data streams that are strongly related to trajectories. Such data streams can be indexed by the existing techniques. Nevertheless, more appropriate methods, which take into account the specificity of these data flows, should be studied.

IV.7 Conclusions

Building scalable systems for more and more complex data has always been a major concern in the world of databases. The moving objects database area makes no exception. The applications related to moving objects generate huge amounts of data, for those that are concerned with the past, or demand specific efficient handling, for those concerning the present and near-future.

In this chapter we presented the related work on indexing spatio-temporal data. We began with the spatial and temporal access methods, since the spatial and temporal support are required in a spatio-temporal database (see Section II.3). Moreover, as the spatio-temporal databases find their origins in the spatial and temporal databases, it is much easier to understand the spatio-temporal indexing by reviewing first the spatial and temporal access methods. Since the domain is really vast, we only focused on a few methods that we consider emblematic in the context of our work.

Regarding the works that cover the spatio-temporal access methods (STAMs), we came upon the same two complementary perspectives that we discussed in the context of spatio-temporal modeling (i.e., managing real-time positions of a collection of moving objects vs. managing the complete trajectories of moving objects). For both perspectives we presented a couple of approaches among the nearly two dozen listed.

In our work, we focus on moving objects in networks. Therefore, we concluded the presentation of STAMs with the related work in this area. Opposite to free movement STAMs, there are not as many indexes for constrained movement. We think that important contributions can still be brought out in this area. Thus, in the next chapter we present our proposed access method for in-networks trajectories.

Chapter V: PARINET: A Tunable Access Method for in-Network Trajectories

In this chapter we present PARINET, a new access method to efficiently retrieve the trajectories of objects moving in networks. The structure of PARINET is based on a combination of graph partitioning and a set of composite B^+-tree local indexes. PARINET is designed for historical data and relies on the distribution of the data over the network as for historical data, the data distribution is known in advance. Because the network can be modeled using graphs, the partitioning of the trajectory data is based on graph partitioning theory and can be tuned for a given query load. The data in each partition is indexed on the time component using B^+-trees. We study different types of queries, and provide an optimal configuration for several scenarios. PARINET can easily be integrated in any RDBMS, which is an essential asset particularly for industrial or commercial applications. The experimental evaluation under an off-the-shelf DBMS shows that PARINET is robust. It also significantly outperforms both MON-tree and another R-tree based access methods which are the reference indexing techniques for in-network trajectory databases.

V.1 Introduction

Current indexing techniques for objects moving in networks decompose the network into roads, and then index the spatio-temporal location of the MOs on each road with a specific index, e.g., a 2D R-tree. One of this approaches' shortcomings is the way the space is decomposed: it is solely determined by the road network, and takes neither into account the distribution of trajectory data, nor the queries on the data. Moreover, indexing both the spatial and temporal dimensions for a given road is not always useful, since most often the spatial dimension (i.e., relative positions) tends to be less selective than the temporal one. In addition, more recent access methods for non-constrained MOs, e.g., [85], [92], have proposed to partition the 2D space according to the data distribution. The index structure proposed in this chapter is also based on data partitioning, and takes into account the distribution of the data over the network. Instead of using a 2D grid as in the previous methods, the partitioning of the data is based on graph partitioning theory in order to integrate the network topology. In addition, we propose a cost model that allows tuning correctly the index structure for a given query load.

In summary, we propose a new access method called PARINET (PARtitionned Index for in-NEtwork Trajectories) to efficiently retrieve the (past) trajectories of objects moving in networks. It proceeds by partitioning the data and indexing the partitions with composite B^+-trees. This allows exploiting the built-in B^+-tree, a robust and efficient index structure that exists in every database system. Similarly to the MON-tree [102], PARINET is capable of answering two kinds of queries on historical constrained trajectories, namely range and window queries (i.e., queries that return all objects whose movements overlap a query window or pieces of the trajectories that overlap a query window respectively).

The contributions presented in this chapter are the following:

- We propose a novel access method called PARINET, based on graph partitioning and time interval indexing.

- We present a cost model that combines the statistics on the data and the query workload to estimate the number of disk accesses for a given index configuration.

- We show how our access method can automatically choose a good index configuration, based on the provided cost model, the data distribution and the query workload using well-known graph partitioning algorithms.

- We characterize the query types in the network constraint MO context and provide different test scenarios.

- We have implemented PARINET using an off-the-shelf DBMS and validated our approach using an extensive experimentation that shows its efficiency and its scalability properties.

The rest of this chapter is organized as follows: Section V.2 introduces the context for PARINET by defining the network model, the data model and the query types. Section V.3 contains the description of the proposed access method along with the cost model and the tuning process. The experimental results are given in Section V.4. Finally, we conclude and discuss the directions for future work in Section V.5.

V.2 PARINET Context

The constrained movement requires a specific data representation but also specific query models (see Chapter II Section II.4.1.2). It is important that the data representation be related to the network space. The queries criteria refer to the network instead of an arbitrary region in space. This section presents the context for PARINET. We first introduce the network model, the data model and the query model. Then, we conclude by giving the hypotheses that guided this proposal.

V.2.1 Network Model

The network model defined for PARINET is similar to those in [101] and [102]. We use two representations for the road network: a geometric view and a topologic view. The geometric view (or 2D view) captures the approximate geographic locations of the road network components. This is the base view of the road network. The topologic view uses a graph in order to represent the road sections and the intersections. It is useful in the partitioning of the network. The notations we use are similar to the ones in [24].

The geometric representation of a *road network* is given by a tuple $RN^{2D} = (S,C)$, where S is a set of segments and C is a set of connections. A *road segment* $s \in S$ is a 2D line segment defined by (p_s, p_e), where $p_s = (x_s, y_s)$, $p_e = (x_e, y_e)$ and $p_s \neq p_e$; p_s and p_e are respectively the start and end points of the segment. A connection $c \in C$ is a tuple (p, S^c) where p is a geographical point that represents the location in the 2D space of the connection and S^C is a set of segments that meet at the connection. The list of segments in S^C should have p as one of their end points. Figure V.1 gives a simple example of a geometric representation of a road network.

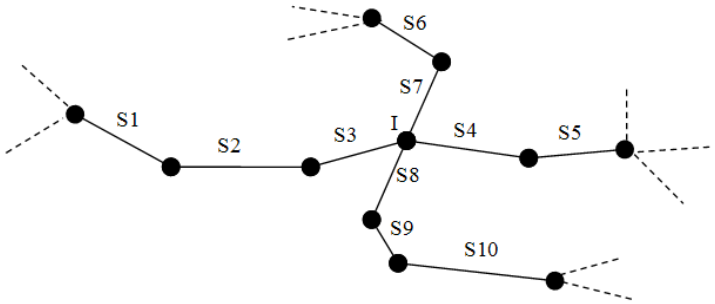


Figure V.1: Example of a geometric representation of a network

Definition V.1: Given a road network RN^{2D} as described above, we define a *road* in RN^{2D} as $Road = (rid, S^c, start)$, where rid is a unique identifier, S^c is a set of connected segments that form a non self-intersecting polyline in RN^{2D} (which may be open or closed (a cycle)) and $start$ is one of the two endpoints of the polyline. Each segment belongs to one road only.

Definition V.2: Given a road network RN^{2D} as described above, we define the set of junctions in RN^{2D} as $Junctions = \{ j \mid j \in C \wedge card(c(S^c)) \geq 3 \}$.

Different granularities can be superimposed to a road resulting in different network models.

Definition V.3: For a given road network RN^{2D}, we define three possible network models:

- *Segment oriented network model*: each segment corresponds to a road.
- *Edge oriented network model*: each road is defined as the polyline between two *junctions*.
- *Route oriented network model*: the complete roads are considered without split. They can extend over the junctions. Notice that several configurations are possible for the route model on the same road network.

In the example in Figure V.1, we have:

- 10 roads (S1,…,S10) in the segment oriented model;
- 4 roads: (S1, S2, S3), (S4, S5), (S6, S7) and (S8, S9, S10) using the edge oriented model;
- 2 roads: (S1, S2, S3, S4, S5) and (S6, S7, S8, S9, S10) in the route oriented model.

In the sequel, we will employ the general term *road network* to denote a road network modeled as one of the three possible network models. When necessary, we will indicate the specific network model for the given network.

Definition V.4: Given a road network RN^{2D}, we define a position in the network space as a pair (rid, pos), where rid is a road identifier and $pos \in [0,1]$ is the relative position on the road measured from the $start$ end of the road.

This is closely related to the concept of linear referencing widely used in GIS for transportation and available in DBMSs as Oracle Spatial or GIS tools as ArcGIS.

Definition V.5: Given a road network RN^{2D}, we define a *road connection* rc as a tuple (p,R^c) where p is the geographical point location in 2D space of the road connection and R^c is the set of roads that meet at the connection. p has the same coordinates as one of the two end points of each road in the given connection.

Based on the 2D representation of a road network RN^{2D}, we construct the topologic representation of the network. In this representation, a network is defined as an undirected weighted graph $G=(V,E)$ with V a set of vertices and $E \subseteq V \times V \times \mathrm{N}$ a set of edges. Each $v \in V$ corresponds to one road connection in RN^{2D}. Given $v_1, v_2 \in V$, there is an edge $e=(v_1, v_2, w)$ in G iff there is a road in RN^{2D} between the corresponding road connections. The weight w is given by the function W, which depends on the data distribution and is defined in Section V.3.3.3. Notice that in our network model, the roads are non-oriented. But taking into account the traffic orientation is a straightforward extension that can be achieved by splitting the two-way roads into two edges.

V.2.2 Data Model

As mentioned earlier, we intend to index the complete trajectories of the MOs in a network. An object moving on a road network reports its position at different moments in time. We assume that such an update is issued each time the MO changes its speed or passes on a different road in the network. An update contains the identifier of the MO, the network position (as given in Definition V.4) and the associated time instant: $(moid, rid, pos, t)$. We define the *trajectory* of a moving object as a non-regulated sequence of *units* (i.e., the time intervals are not of equal size). Each unit is a tuple defined by two consecutive updates: $(moid, rid, [pos_1, pos_2], [t_1, t_2])$; t indicates the time interval, while pos gives the relative position on the road at the beginning and the end of the time interval [102]. For each unit, it is assumed that the MO moves at constant speed, i.e., a linear interpolation is considered over each interval. Given a road, the relative position on the road and the time can be viewed as the two orthogonal axes of a 2D space. In this space, we denote by *unit segment* the 2D line segment bounded by the points (pos_1, t_1) and (pos_2, t_2). Also, a *unit minimum bounding rectangle* (unit MBR) is the rectangle that contains the unit segment (Figure V.2).

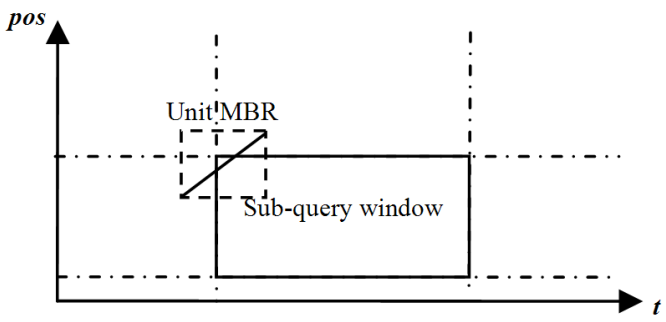


Figure V.2: Example of unit and sub-query intersection

V.2.3 Query Types

In this paper, we consider two types of queries: *2D queries* and *path queries*. Both query types are composed of a spatial part and a temporal interval: $Q = (Q_s, Q_t)$. The queries return either all the MO that have lied within the area of Q_s, at a certain time interval Q_t, or only the pieces of the trajectories that overlap the query. The difference between the two types of queries lies in the spatial part Q_s.

The spatial component of the first type of queries is a 2D region. Hence, the first type of queries represents "standard" spatio-temporal queries [100], [101] and [102]. Thus, Q_s is a 2D region (usually a rectangle). In the rest of the paper, we will refer to this type of queries as *2D queries*. To support 2D queries, a transformation of Q_s is performed first. The exact intersection between the 2D region and the network is computed. Then the initial region in Q_s is replaced with the intersected network region. Formally, the new Q_s is a set of road sections: $Q_s = \{rs_1, rs_2, \ldots rs_n\}$ where $rs_i = (rid_i, [pos^i_{11}, pos^i_{12}], [pos^i_{21}, pos^i_{22}], \ldots [pos^i_{k1}, pos^i_{k2}])$ and $\{rs_i \neq rs_j\}$ and $pos^i_{m1} \leq pos^i_{m2} \wedge pos^i_{m2} < pos^i_{(m+1)1}$. Each rs_i represents a set of disjoint and ordered intervals on one road [102]. Multiple intersection intervals with the query region are possible when the road is a polyline, which is the case for an edge or route network model. Usually, one can use a 2D R-tree over the network to speed up the computation of the mapping between a 2D region and a network region (see the top 2D R-tree in Figure IV.23 in Chapter IV).

The constrained movement suggests another type of useful query. For example, "find in a database all the MOs whose trajectories intersect a given MO trajectory", or "find the number of MOs that traverse a given road section at a certain time (interval)" are *path queries* that need to refer to the network. Path queries represent a new type of spatio-temporal queries that we introduce. In a path query, the spatial part, Q_s, represents a path in the network, i.e., a sequence of connected road sections. For this type of queries, no mapping is needed from the 2D space to the network space and Q_s has the same formalization as above.

V.2.4 Observations

In this subsection, we give a short informal intuition of the PARINET index structure. We use a filtering and refinement approach. The main idea of our proposal is that an approximate index search could deliver very good performance in terms of computation time, while offering at the same time good results in terms of physical accesses. The overall performance of such an access method can surpass the "exact" index search used in the existing methods.

Actually, in a network space, the spatial dimension is composed of a discrete component (the road identifier) and a continuous component (the relative position on the road). PARINET is based on the three following observations:

Observation 1: The relative position dimension is usually less selective than the temporal dimension. Using an index on time for filtering candidates followed by a refinement step should be more efficient than using an R-tree on the two dimensions.

The MON-tree [102] and the PJ-tree [101] fully index the bi-dimensional space (relative positions and time) with a 2D R-tree. Nevertheless, it is expected to have an important amount

of overlapping of the indexed units in the spatial dimension, because in general, trajectories traverse entirely the road segments in their path. Moreover, except for queries on very small regions, the usual queries cover many road segments. Therefore, indexing only the temporal dimension might be more efficient, since time is more selective in this case. For this reason, we use a B^+-tree combined with sorted data on time. This offers an efficient sequential range scan of the tuples that intersect the temporal query interval Q_t.

Observation 2: The partitioning of the network space should not be made only on a road identifier basis, as it is the case for the existing methods. It should be based on the data distribution and the network topology.

Indeed, while the alternative of one index by road offers the advantage of an exact filtering on one component of the spatial dimension, it nevertheless has a few shortcomings. The partitioning is strictly related to the static road view of the network and does not consider the data statistics (distribution of MO over the network). This is an important aspect and is even more relevant in a historical context. Moreover, the performance of the existing methods, e.g., MON-tree depends on the granularity of the employed network (section, edge, or route based model). Another argument is that a network can contain several thousand roads and having a separated index for each one could degrade the system performance even for small data sets.

Instead, we propose an index structure that takes into account the data distribution over the network and the network topology. The network will be partitioned in network regions that will be balanced with respect to the amount of data in each region. Therefore, the parts of the network with less traffic (e.g., the peripheral ones) will have larger extents than the busy zones (e.g., the central ones). Queries are most of the time defined on regions where road segments are close or connected. A general rule is to group together the objects that are close, which will help return more results in a few page accesses (for instance, R-trees are based on this rule). Because we are dealing with a network, the grouping should take into account the connectivity of road segments, i.e., the network topology, in addition to the data distribution.

Observation 3: The access method should be supplemented with a good quality cost model that will allow tuning the structure for better performances.

Most of the existing methods offer an empirical evaluation of the performance. Some of the recent works, e.g., [85], [92], propose analytical cost models, which are useful for the index (self)-tuning. Our objective is to provide an administration tool for tuning the index.

V.3 PARINET Index

In this section we present the index structure and its operations. Section V.3.3 proposes a cost model based on query and data sizes, and formalizes PARINET tuning in terms of a graph partitioning problem. Finally, we show how one can automatically tune PARINET for a better performance, given a road network, the distribution of the data to be indexed, and an expected query workload.

V.3.1 Index Structure

Based on the above-mentioned observations, the intuition of our approach is to create a B^+-tree index on time intervals for the set of roads in each partition (retuned by the partitioning phase) rather than creating an index for each isolated road. The partitioning is

based on both the data distribution and the network topology (cf. Observation 2), i.e., the partitions are balanced in terms of the amount of data and the partitions separate the network into regions (i.e., connected or close roads are grouped in the same partition) (see Figure V.5 in Section V.3.3.3).

The discussion on how one can choose a good number of partitions and how the partitioning is obtained is presented in Section V.3.2. For now, we assume that this aspect is solved and a good partitioning can be obtained for a given network and a given data distribution. As a result of this operation, each road will be assigned to a certain partition (cluster).

Given a data set D containing trajectories of MOs in a network as a set of trajectory units: $D = \{(moid, rid, [pos_1, pos_2], [t_1, t_2])\}$, the index is built in three steps: partitioning the trajectories' units based on their road identifier, sorting the partitions on the time intervals and indexing each partition using a composite B$^+$-tree on (t_1, t_2) interval. Note that an interval-based B-tree such as the RI-tree [70] could be used for indexing time intervals, but we chose a simple B$^+$-tree to allow an easier implementation. The index structure is quite simple. An example is given in Figure V.3. A table *RP* (Road Partitioning) that contains one entry for each cluster keeps some basic information on the partitioning: the list of road identifiers for a cluster and a pointer to the B$^+$-tree index over the unit segments in the cluster. As we partition the data according to the spatial dimension, the time (t_{min}, t_{max}) represents the entire spanning time of the indexed trajectories. Therefore, only one *RP* table is necessary to report the relationship between the partition attribute (i.e., the *rid*) and the partition index. In the sequel, we present the search and insertion operations for PARINET.

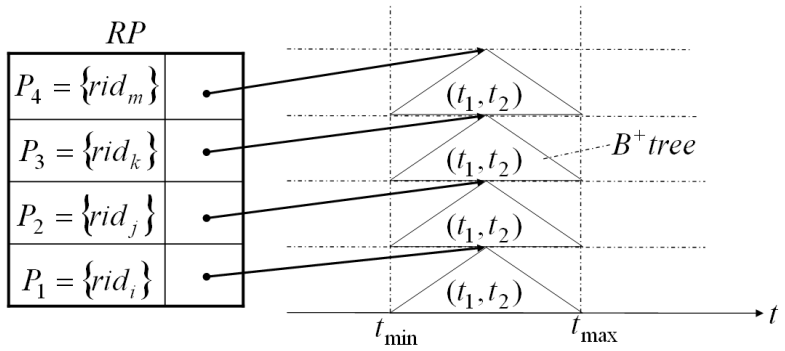


Figure V.3: Example of PARINET index structure

V.3.2 Index Operations

V.3.2.1 Search Algorithm

Given a (2D or path) spatio-temporal query $Q = (Q_s, Q_t)$ where $Q_s = \{rs_1, rs_2, \ldots rs_n\}$ and $Q_t = [t_s, t_e]$ (see Section V.2.3), PARINET can find all the objects that have traversed the road sections in Q_s during the time interval of Q_t (range query), or simply return the trajectory units that intersect Q (window query). Data retrieval is performed in three steps. First, we identify the partitions that contain the road identifiers in the query. Then, we use the B$^+$-tree indexes of the selected partitions and look up candidate data. Finally, we perform an exact match search among the candidates.

Based on the set of road identifiers $\{rid_{q1}, rid_{q2}, \ldots rid_{qn}\}$ in Q_s and on the distribution table RP, we determine the set of partitions $\{P_{p1}, P_{p2}, \ldots P_{pm}\}$ that include all the roads in the query. Note that $m \leq n$, but in general $m < n$ and we might also have $m << n$ depending on partition and query sizes. This means that the total number of searched partitions is smaller than the total number of accessed roads in general, as it is the case in a road oriented partitioning.

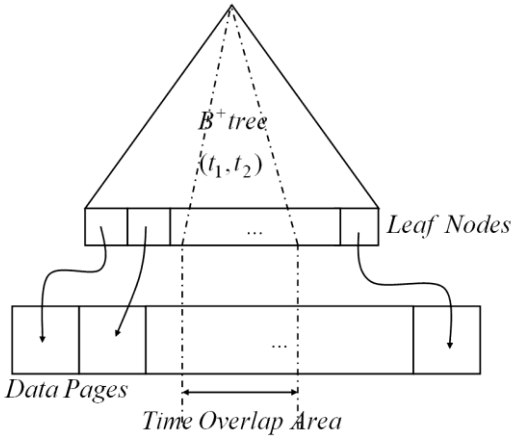


Figure V.4: Example of index range scan

Then, for each accessed partition we perform a range scan by using the B$^+$-tree index in order to find the data pages that temporally overlap Q_t (see Figure V.4). Note that this may lead to false positives, because the filtering is based only on time and does not consider the road identifiers or the relative positions on the road. However, the capability of accessing groups of roads that are likely to appear together in a query will lower the number of false positives.

Finally, at the refinement step, for each candidate data we determine if it truly intersects Q, i.e., the unit segment intersects one of the sub-query windows (as depicted in Figure V.2). The actual intersection between the unit segment and the sub-query window is computed only if the unit MBR is not completely covered by the window.

V.3.2.2 Index Maintenance

Historical trajectories are mainly static data. Therefore, the index construction can be done once and periodically updated. Practically, the data needs to be organized in partitions and the insertion of each unit in a bucket must be made with respect to (t_1, t_2) values. In a RDBMS for example, this can be done using a temporary view or table. Afterwards, a B$^+$-tree index is built for each bucket on (t_1, t_2). If additional data must be added later, this can be directly appended to the existing partitions with the update of the corresponding index for each inserted unit. Usually, newly added trajectories are more recent than the existing ones, which have less impact on the B$^+$-tree. In case the added trajectory units follow the same distribution as the initial ones, the overall performance remains the same. Otherwise, a complete reorganization of the index structure is needed in order to maintain the best performance. Note that the reorganization is not frequent and the index can be considered stable.

V.3.3 Data Partitioning

V.3.3.1 Problem Statement

PARINET is based on the partitioning of the road network. Moreover, the partitioning must take into account the data distribution over the network (e.g., total number of unit segments for each road) and the network topology. It is clear that, for a given query load, different partitioning of the same data will lead to different performances. Our goal is to automatically find the best partitioning scenario for a given query load. This is possible as the network and the data to be indexed are known in advance.

In this section, we will present a cost model that estimates the number of disk accesses that are necessary in order to answer a query load, given a certain configuration of our index. Then, using the cost model, we will rewrite the partitioning problem as an optimization problem and use a graph partitioning algorithm to resolve it. We assume that the overall performance (mainly the response time) of the index is directly related to the number of disk accesses.

V.3.3.2 Cost Model

In this section we present a cost model that estimates the number of physical disk accesses for a given query and index configuration. The notations used in this section are explained in Table V.1.

The total number of disk accesses for a query is the sum of the physical accesses in each accessed partition. We consider that the table RP, which gives the distribution of road identifiers in the partitions, is sufficiently small to fit in main memory. For each accessed partition, we have disk accesses for the range scan in that partition. A range scan comprises the index search and the data page scan (see Figure V.4). We obtain the next formula for the total number of disk accesses: $DA_Q = \sum_{p \cap Q_S} (IA_p + PA_p)$ (1).

DA_Q	Total number of disk accesses for a query
IA_p	Number of index accesses in a partition
IA_p^f	Number of fixed index accesses in a partition
IA_p^v	Number of variable index accesses in a partition
PA_p	Number of page accesses in a partition
N_p	Number of units (tuples) in partition p
$Pages_p$	Number of data pages in a partition
ρ_p^t	Temporal data distribution in a partition (percentage of $Pages_p$ per time unit)
T_{max}	Maximum unit time interval in the data set
BS_i	Index block size (number of entries) per index page
BS_d	Data block size (number of entries) per data page

Table V.1: Notations

The data access cost is the number of pages containing the data that overlap with Q_t. Given the distribution of the data in time ρ_p^t, the number of pages read is:

$$PA_p = Pages_p \times \int_{Q_t+T_{max}} \rho_p^t \cdot dt = \frac{N_p}{BS_d} \times \int_{Q_t+T_{max}} \rho_p^t \cdot dt \quad (2).$$

For simplicity, we consider a uniform temporal distribution such as $\rho_p^t = \rho_p = ct$. In this case (2) becomes:

$$PA_p = \frac{N_p}{BS_d} \times (Q_t + T_{max}) \times \rho_p \quad (3)$$

The number of index accesses is composed of a fixed cost and a variable cost. The fix cost comprises the accesses done to reach the leaf nodes from the index tree root, which is equal to the tree height. This can be computed based on the number of index entries and the tree fanout: $IA_p^f = \log_{fan} N_p$ (4). A typical value for IA_p^f is 3 when $fan \approx 100$ and the number of index entries is in the millions of tuples. The variable index cost reflects the number of pages with leaf nodes that overlap with Q_t. Similar to PA_p, we obtain:

$$IA_p^v = \frac{N_p}{BS_i} \times (Q_t + T_{max}) \times \rho_p \quad (5).$$

From (1), (3), (4) and (5) we obtain:

$$DA_Q = \sum_{p \cap Q_s} \left[\log N_p + N_p (Q_t + T_{max}) \rho_p \left(\frac{1}{BS_i} + \frac{1}{BS_d} \right) \right] \quad (6).$$

One advantage of our method is that it permits a simple estimation of the disk accesses for given query load, based on some statistics on the indexed data. This estimation can be used to automatically tune the index for better performance. In short, we can modify the average area of network partitions by changing the total number of partitions n. Intuitively, given a query of a certain size, the number of disk accesses needed to answer the query will decrease with the partition size, because less false positives will be examined. However, increasing the number of partitions after a certain point will result in a performance loss. This is due to the fact that more partitions need to be observed, which increases the fixed index physical accesses and query overhead. The cost model is verified in the experimental evaluation presented in Section V.4.

V.3.3.3 Using Graph Partitioning

Assuming that the above cost model is accurate, we can estimate the performance of our access method for a given configuration, without effectively constructing the index. Therefore, we can search among some of the possible configurations and materialize the best one with respect to the cost model. A possible index configuration corresponds to a network partitioning into a given number of parts that respects some given constraints (cf. Observation 2).

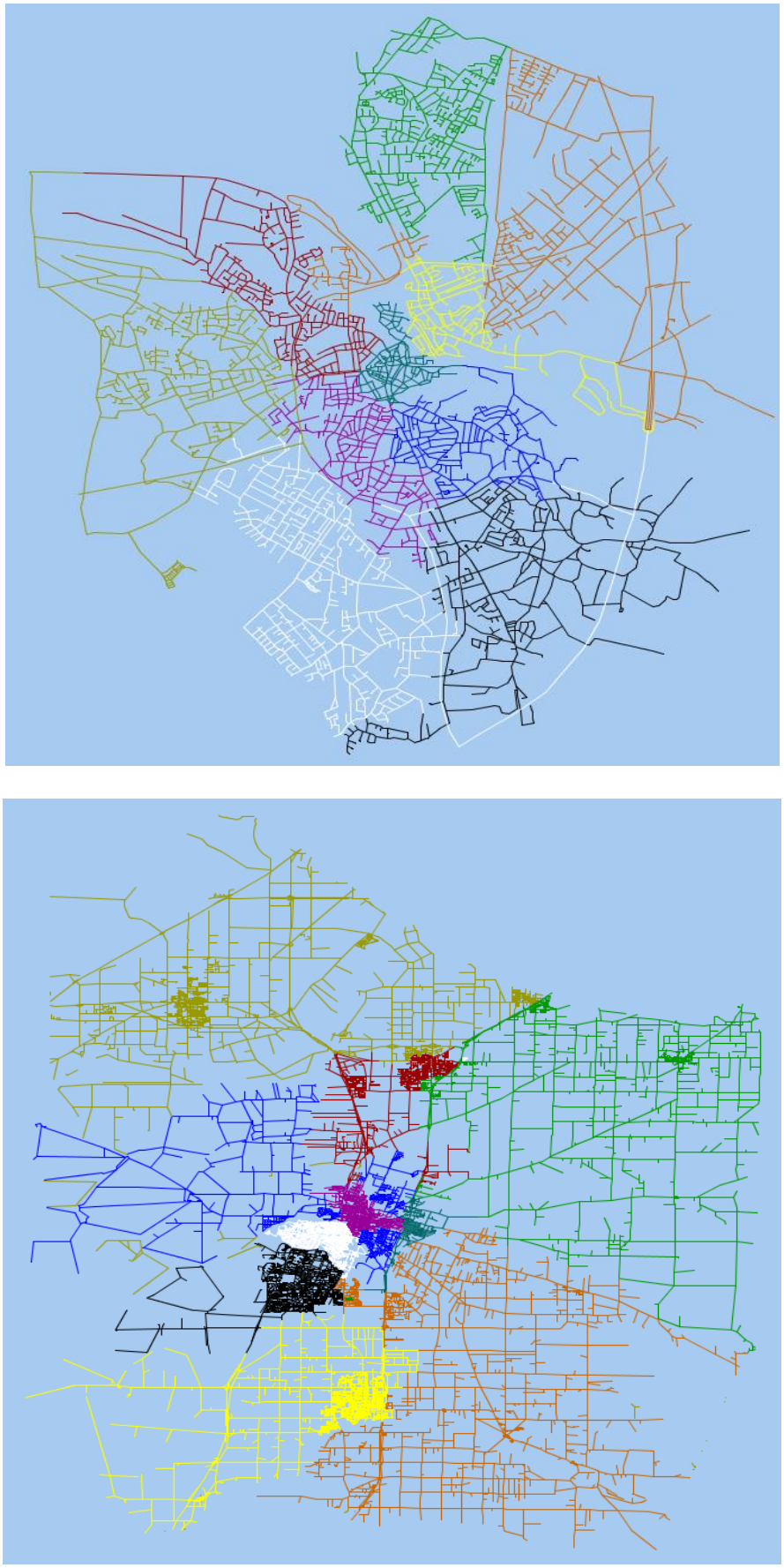


Figure V.5: Example of weighted network partitioning performed by METIS on our test road networks (Oldenburg up and Stockton down) (see Section V.4)

Graph partitioning is an important problem that has been extensively studied in the last decades. The problem is to partition the vertices of a graph in n roughly equal parts, such that the number of edges connecting vertices in different parts is minimized [106]. The problem was extended to graphs where each node and each edge can have weights. Therefore, the resulting partitions can be balanced in term of node weights instead of number of nodes, for example (see Figure V.5). The graph partitioning problem is NP-complete [106]. However, many algorithms have been developed to find high quality partitions extremely fast based on specific heuristics [109]. Public implementations are also available, e.g., METIS [107].

As formulated in Section V.3.1, the constraints imposed by PARINET on the network partitioning, can be entirely satisfied by the graph partitioning algorithms. The formalization of our approach is the following: given an undirected network graph $G=(V,E)$ and a data set D (as described in Section V.2.2), we compute the weight function of the graph roads $W : E \rightarrow \mathrm{N}$. W associates for each road in G the number of units from D on that road. Let $L(G)$ be the line graph of G. W is a node weight function of $L(G)$. Let $P=\{P_1,P_2,\ldots P_n\}$ be the partitioning of $L(G)$ in n parts, such that the partitions are *contiguous* and *balanced* in terms of total weight. Let $Q_L=\{Q_1,Q_2,\ldots Q_k\}$ be a query load. We define the quality indicator of P over $L(G)$ as: $QI^n_{Q_L}=\sum_{i=1}^{k} DA_{Q_i}$, where DA_{Q_i} is computed by (6).

The goal is to find the partitioning such that QI_{Q_L} is minimal. Our approach was to implement a program that is based on METIS [107] and that returns the partitions with the best QI_{Q_L} by iterating through the possible index configurations. METIS takes as input a weighted node graph and a number n of parts. It partitions the input graph in n parts such as the partitions are fairly balanced and contiguous (although this is not guaranteed, non-contiguous portions are exceptions), which is conform to our demands for the partitioning of the road network (cf. Observation 2). By iterating n from 1 to $card(E)$, we choose the partitioning with the best QI for the materialization of the index structure. Notice that our experimental results showed that a step of 100 for n in the iteration is sufficient because usually $QI^n_{Q_L}$ has small variations with n. Thus, the computation time for the optimal partitioning takes about one minute on our testing machine, which is negligible compared to the time necessary for testing several index configurations.

V.4 Experimental Evaluation

In this section, we experimentally compare our method against the PJ-tree and the MON-tree under an off-the-shelf DBMS implementation. We also test the performance of our method with different map, data and query sizes. Then, we evaluate the proposed cost model. Finally, we show that PARINET is robust with the variation of the query size when tuned correctly. All the experiments were conducted under Oracle 11g Enterprise Edition installed on a Pentium 4 3.2 GHz machine having 2.5 GB of RAM running Windows XP. Implementing PARINET under Oracle is straightforward using the available table partitioning mechanism. A given data set is stored in a relational table where each tuple represents a unit having the following attributes: $(moid,rid,pos_1,pos_2,t_1,t_2)$. The table is partitioned based on the rid value. Each partition contains a list of rid values. Oracle allows creating an index for each partition. First, we insert the units in the partitioned table ordered by (t_1,t_2). Then, we

build a local index for each partition. The table T that keeps the mapping between the road identifiers and the partitions is implicit because the DBMS internally manages the partitions based on the table metadata.

V.4.1 Data Sets and Queries

Available real trajectories data, such as INFATI [105], are not representative enough in terms of trajectory variety and data size. Moreover, the underlying road network required by the experimentation is rarely available free of charge. Therefore, we used the generator for moving objects in networks proposed in [95] to create realistic data sets. The generator is available [98] with several network examples and we used two of them: the road networks for the city of Oldenburg (Germany) and the city of Stockton (San Joaquin County, CA). The networks are represented in a segment oriented model, i.e., each line segment represents a road and has a unique identifier. This is the smallest granularity for a network representation. We used the networks directly in this format for PARINET tests and transformed Oldenburg for the comparison between PARINET and the R-tree based access methods. Oldenburg has 7035 segments and 6105 nodes, while Stockton has 24123 segments and 18496 nodes. Stockton contains more than three times the number of roads of Oldenburg. With regard to the distribution of the generated MOs, we set the generator for a region-based distribution. In this approach, the network regions with higher node density have a higher probability of containing a starting point for a MO. The position of each MO is reported each time it passes a node. We generated 10 classes of MOs, each class corresponding to a maximum speed. The generator also simulates weather conditions or similar events that impact the motion and speed of the MOs.

We have two collections of data sets. One collection is composed of small data sets that we use to compare PARINET against the R-tree based indexes. The other one consists of larger data sets and is used for a deeper analysis of PARINET. The reason for having two different data sets is that the R-tree based indexes do not scale well for large data sets. Table V.2 presents the statistics for the first data set collection. They are all based on a transformed map of Oldenburg (see Section V.4.2). Table V.3 gives the statistics for the second data set collection based on the original Oldenburg and Stockton maps.

The data sets have different number of units, trajectories, trajectory length or time span, and map size. In average, a MO trajectory in Stockton has twice the number of units of a trajectory in Oldenburg, because of the network size. Hence, for the same number of units, we have twice as many MOs in Oldenburg than in Stockton. This will allow testing the index behavior according to the map and the data set sizes in terms of total number of units, of MO and of trajectory length.

We tested the two types of queries: 2D and path. For each type of query and for each map, we generated three scripts, each script containing queries of fixed size. The statistics for the generated query sets are given in Table V.4 and Table V.5. For the 2D queries, we first randomly generated a 2D square window and a time interval. The intervals have the same relative size in all the dimensions. Then, we transformed the query as presented in Section V.2.3 and generated the final script.

For the path queries, we randomly selected some trajectories from the data set and used them to generate the spatial interval of the queries. We generated queries where the size of the spatial interval is 0.25%, 0.5% and 1% respectively of the total number of roads in the network. Each road section in a query contains the entire road segment. For each of the three

spatial windows, the time interval was fixed to 2.5%, 5% and 10% of the total time of the data set. We chose a smaller spatial window due to the large number of roads in a network. The temporal interval is randomly chosen within the temporal interval of the data set.

For a given query set, data set and index configuration, we measured the average time per query and the average number of disk accesses per query. We emptied the cache between each query run to avoid any influence of the cache on query processing evaluations.

Data set name	# of units	# of MO	# of time units	# of MO per time unit
Old 1	124079	3929	400	10
Old 2	273543	8890	600	15
Old 3	510761	15823	800	20

Table V.2: Tested data sets statistics for PJ-tree, MON-tree and PARINET

Data set name	# of units	# of MO	# of time units	# of MO per time unit
Oldenburg 1	685515	15964	800	20
Oldenburg 2	3489751	79785	800	100
Stockton 1	690890	8285	830	10
Stockton 2	3448008	41475	830	50

Table V.3: Tested data sets statistics for PARINET

Query set name	Spatial interval in each dimension	Avg. # of intersected roads by the queries	Temporal interval	# of queries in the set
Old 2D Q1	2.5%	9.5	2.5%	105
Old 2D Q2	5%	34	5%	64
Old 2D Q3	10%	60.2	10%	49
Sto 2D Q1	2.5%	23.4	2.5%	108
Sto 2D Q2	5%	96.3	5%	100
Sto 2D Q3	10%	156	10%	98

Table V.4: Tested 2D query statistics

Query set name	Spatial interval	# of roads in the query	Temporal interval	# of queries in the set
Old P Q1	0.25%	17	2.5%	84
Old P Q2	0.5%	35	5%	75
Old P Q3	1%	70	10%	63
Sto P Q1	0.25%	60	2.5%	71
Sto P Q2	0.5%	120	5%	66
Sto P Q3	1%	241	10%	34

Table V.5: Tested path query statistics

V.4.2 PARINET vs. PJ-tree vs. MON-tree

We compared PARINET with PJ-tree and MON-tree. A comparison between MON-tree and FNR-tree [100] is already provided in [102] as discussed in Section IV.5 (Chapter IV).

Implementing MON-tree under Oracle is also straightforward and follows the same technique as for PARINET. However, there are some differences from the method presented in [102]. Firstly, the MON-tree in [102] uses a modified version of an R-tree that is capable of handling multiple query windows in one index scan. Oracle implements an R-tree index, but, to the best of our knowledge, does not allow changes to the built-in indexes. Secondly, for network queries (cf. Section V.2.3), the top R-tree on the network is no longer needed. The mapping of the 2D queries is performed apart and not considered in the index performances.

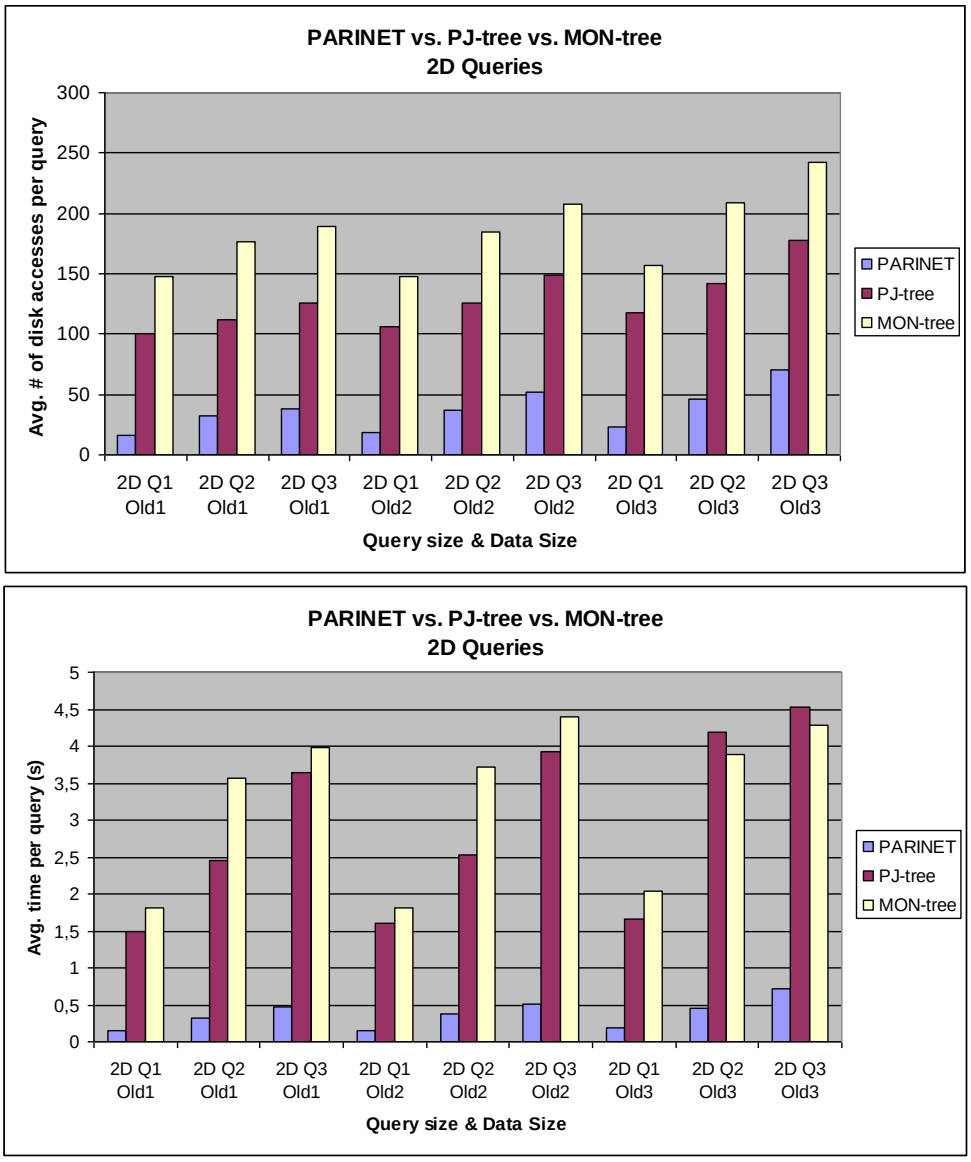


Figure V.6: Performance comparison between PARINET, PJ-tree and MON-tree for 2D queries

Moreover, as reported in [102], the MON-tree performs better on route-oriented network models. We concatenated the segments of the Oldenburg map and generated longer routes. From the 3328 segments that constitute the core of the Oldenburg network, we generated 186 routes, i.e., an average of 17.892 segments per route. Then, three data sets (see Table V.2) were generated on this transformed map. Note, that the data generation was done as previously explained, i.e., a MO reports its position each time it traverses a segment node. This means that several units are generated for a MO that traverses a route. This aspect is important for the MON-tree and the PJ-tree, because the selectivity on the relative positions

on a route is significant for this network model. PJ-tree uses a single 2D R-tree to index the data and is also easy to implement under an off-the-shelf DBMS.

As the longer routes can be seen as segment clusters, we consider them as separate partitions for PARINET. This kind of partitioning also helps validate our first observation (see Section V.2.4III-D). A proper partitioning based on the software METIS is used in the next section.

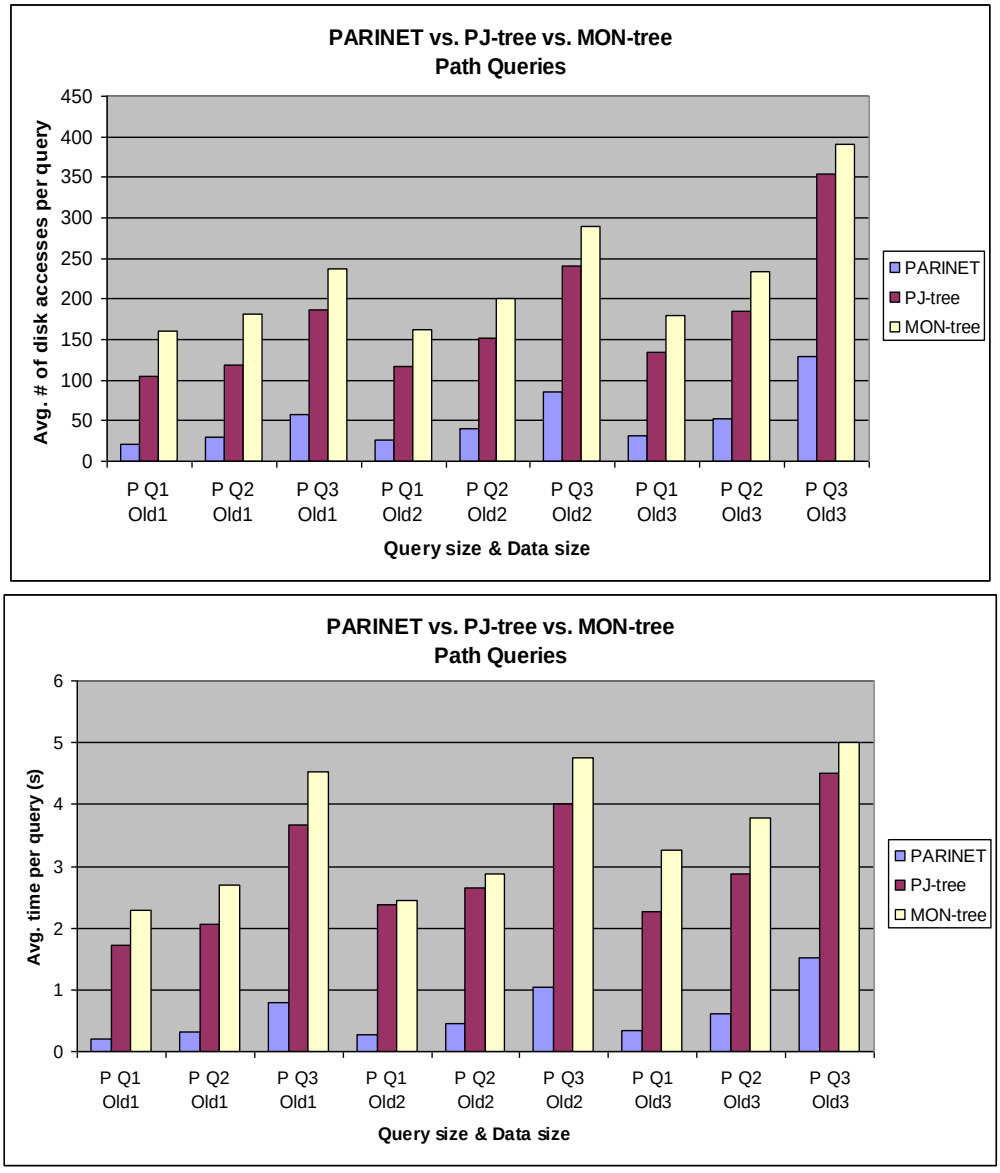


Figure V.7: Performance comparison between PARINET, PJ-tree and MON-tree for path queries

Figure V.6 and Figure V.7 present the comparative results for the three methods for Old 1, 2 and 3 data sets (see Table V.2) and the two types of queries. PJ-tree outperforms MON-tree for most of the tests, except for the execution time of the largest 2D queries and the largest data set. Using several indexes as in MON-tree instead of one as in PJ-tree leads to an increased overhead which can be significant for small data sets. We observe that MON-tree shows better performance than PJ-tree for 2D queries on large data sets. However, the number

of disk accesses and the execution time were already high for these two methods for the tested data sets.

The performance of PARINET is always better than the R-tree based indexes from both execution time and disk access point of views. The difference increases with the query size. The average time per query, which is sometimes one order of magnitude smaller for PARINET came as a surprise. This is certainly due to the sequential type of disk accesses in PARINET as the data is clustered and sorted.

V.4.3 PARINET in Depth

In this section we report the results of a more elaborated set of experiments on PARINET. We used the largest data sets (see Table V.3) for the tests. We use METIS [107] for computing the partitions and generate the indexes for 100, 200, 500, 1000, 1500 and 3000 partitions. We analyze the index performance with respect to the map, the data set and the query sizes. Then, we evaluate the proposed cost model. Finally, we discuss the index robustness with the query size variations.

V.4.3.1 PARINET Query Performance

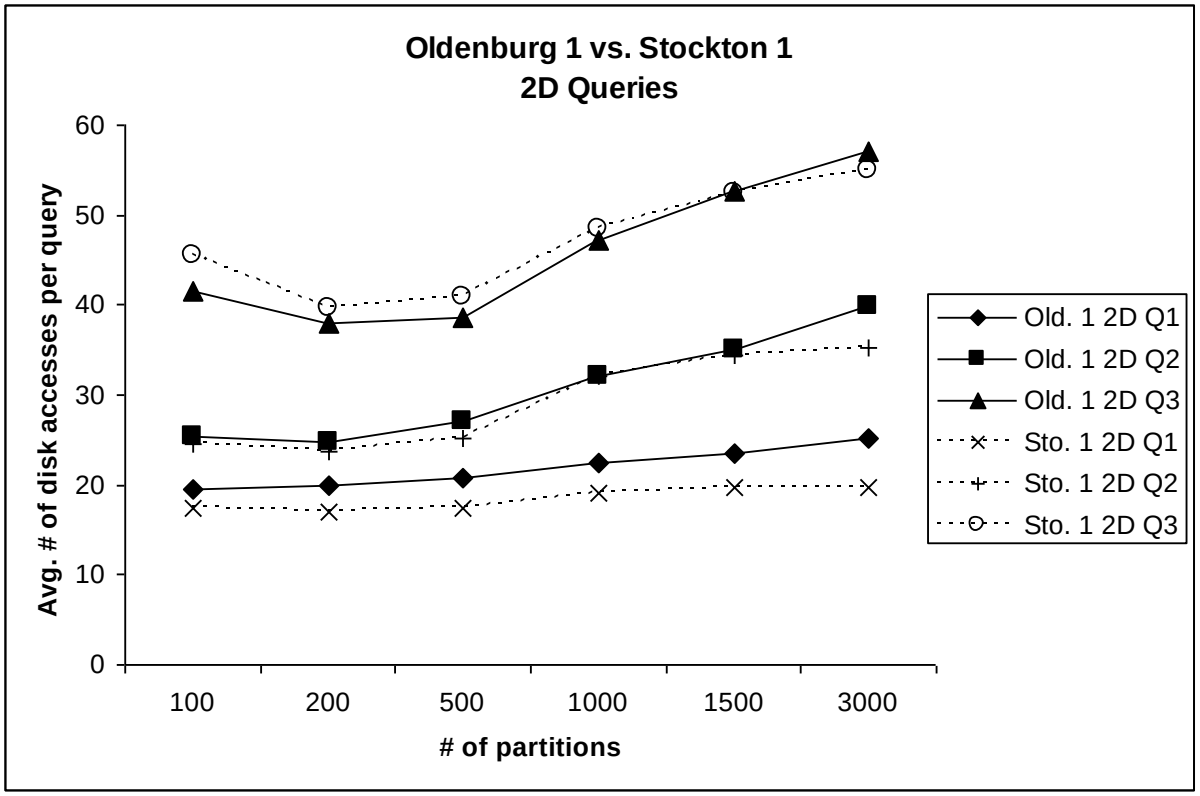
Figure V.9 to Figure V.11 present the results obtained for 2D and path queries over Oldenburg 1, Oldenburg 2, Stockton 1 and Stockton 2 data sets (see Table V.3). The tests confirm our second observation, i.e., the index performance depends on the data set size, the query size and the query type. A smaller number of partitions is necessary for an optimal performance when dealing with smaller data sets. For our data sets, 100-200 partitions offer the best performances for the smaller data sets, and 500-1000 partitions are needed for the largest.

The index performance also depends on the query size and the query type. For 2D queries, when the query size is large, we need to reduce the number of partitions in order to maintain an optimal index performance. For example, the optimal performance is obtained for 1000 partitions for the small 2D queries over the Oldenburg 2 and Stockton 2 data sets and for 500 partitions for the largest query sets over the same data sets. For the path queries the opposite holds true.

The query type influences the index performance. Path queries are more demanding that 2D queries, which is foreseeable considering the fact that they extend over more network regions for a given partitioning scheme and a spatial query size. The performance for the 2D queries depends on the degree of data density over the network. We can see that for the same amount of data (e.g., Oldenburg 2 and Stockton 2) and similar queries, the indexes yields similar results.

As usual, the execution time depends on disk accesses. However, the execution time is more sensitive to the number of partitions. This is more noticeable for the largest data sets and a high number of partitions (i.e., 3000). For higher partition numbers, the variations for the disk accesses are minor, but they can be significant for the execution time. For instance, in Figure V.10, we observe small variations in the disk access values for 500 partitions or more (upper graphic) whereas the execution time becomes worse (lower graphic). This is expected as indicated in Section V.3.3.2. Therefore, a good technique will be to choose a break point where increasing the number of partitions becomes useless in term of disk access.

Overall, we find that PARINET has a solid performance record when correctly tuned and scales much better with the data and query sizes than the R-tree based methods. The query performance remains good both for disk accesses and execution times, for queries containing from 17 up to 150 road identifiers. Also, we did not observe any influence of the number of MOs or the trajectory lengths on the query performance. Only the query size and the number of units in the data set affect the performances.



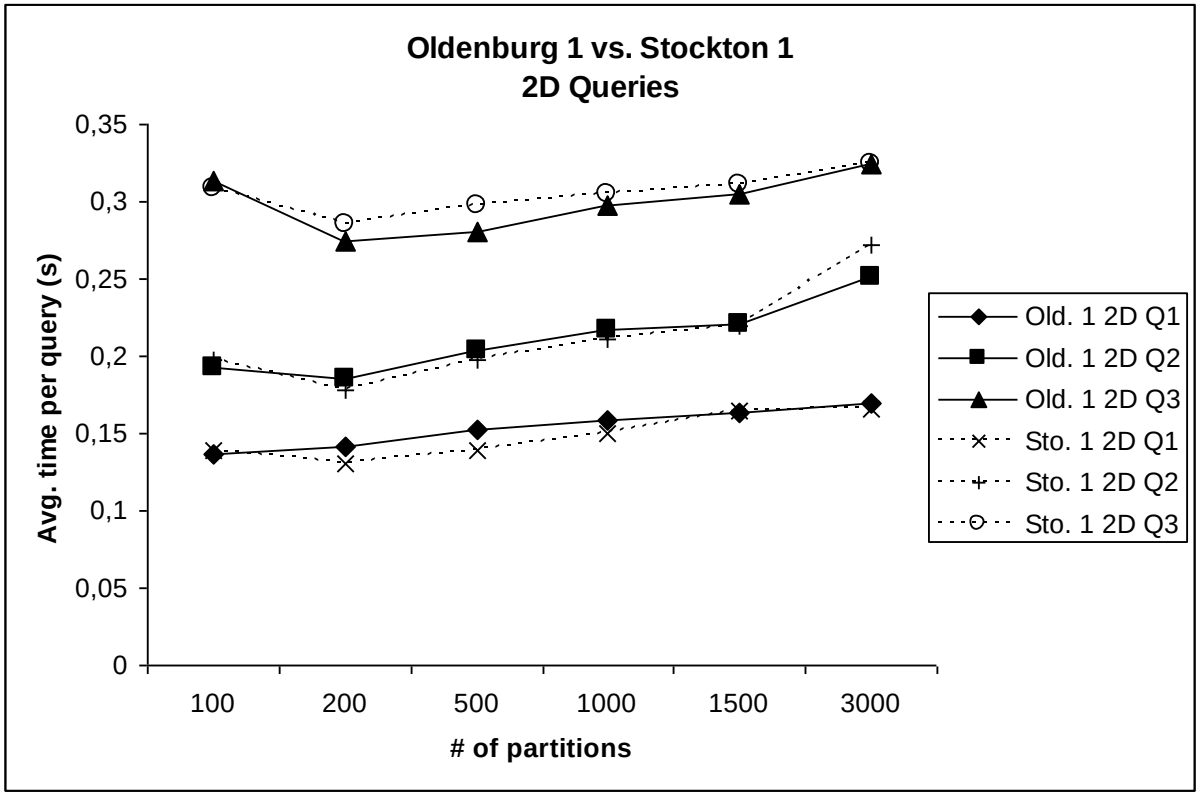
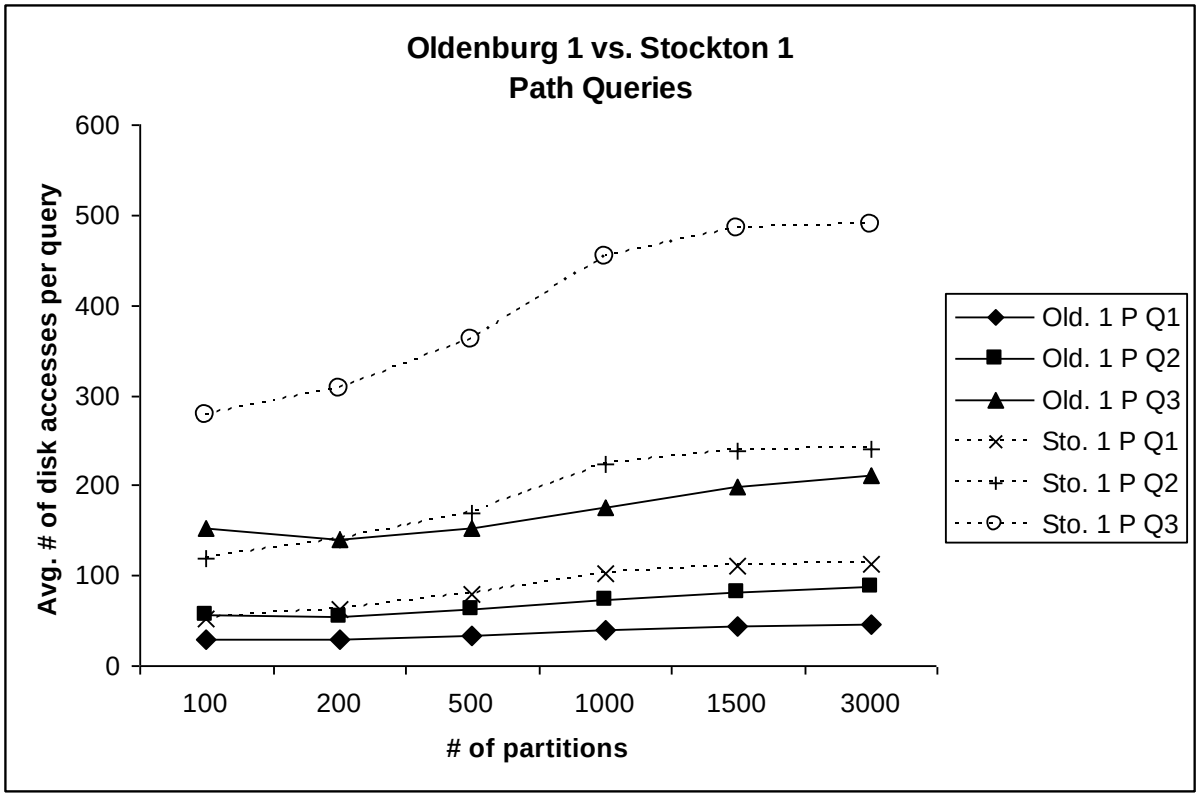


Figure V.8: PARINET query performance - 2D queries on small data sets



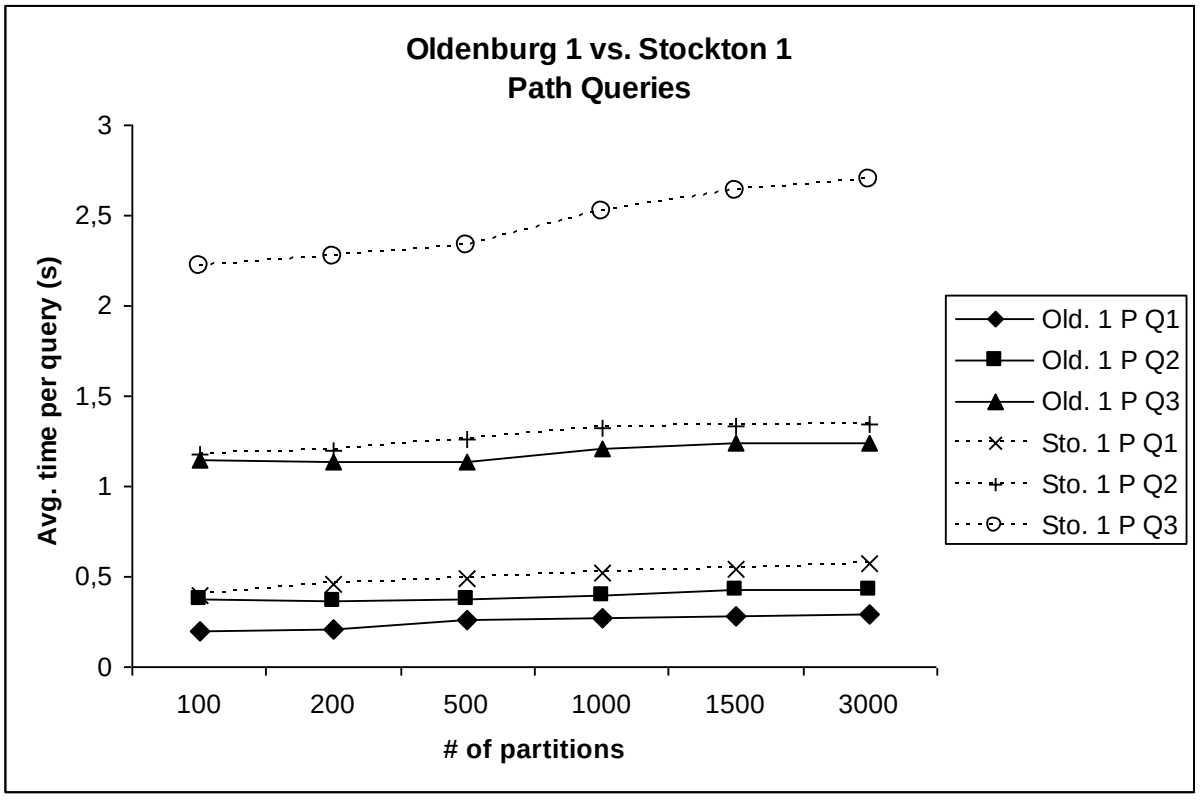
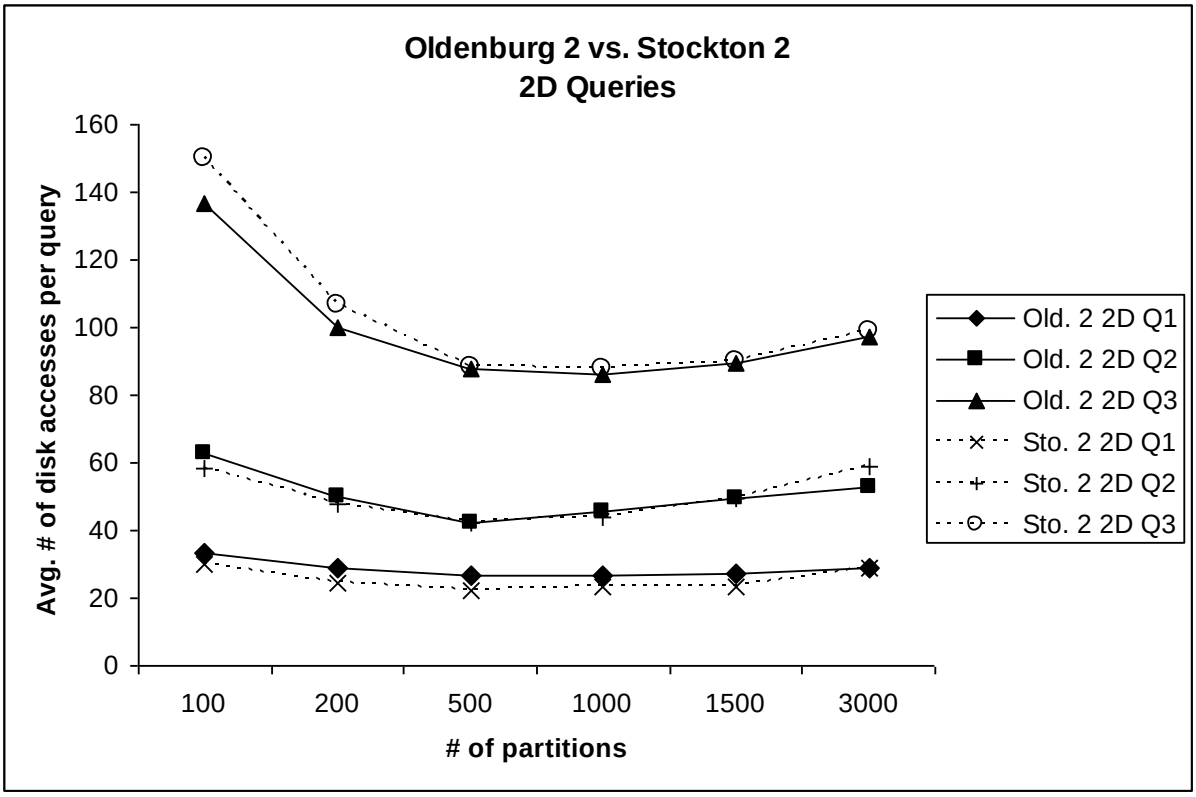


Figure V.9: PARINET query performance - path queries on small data sets



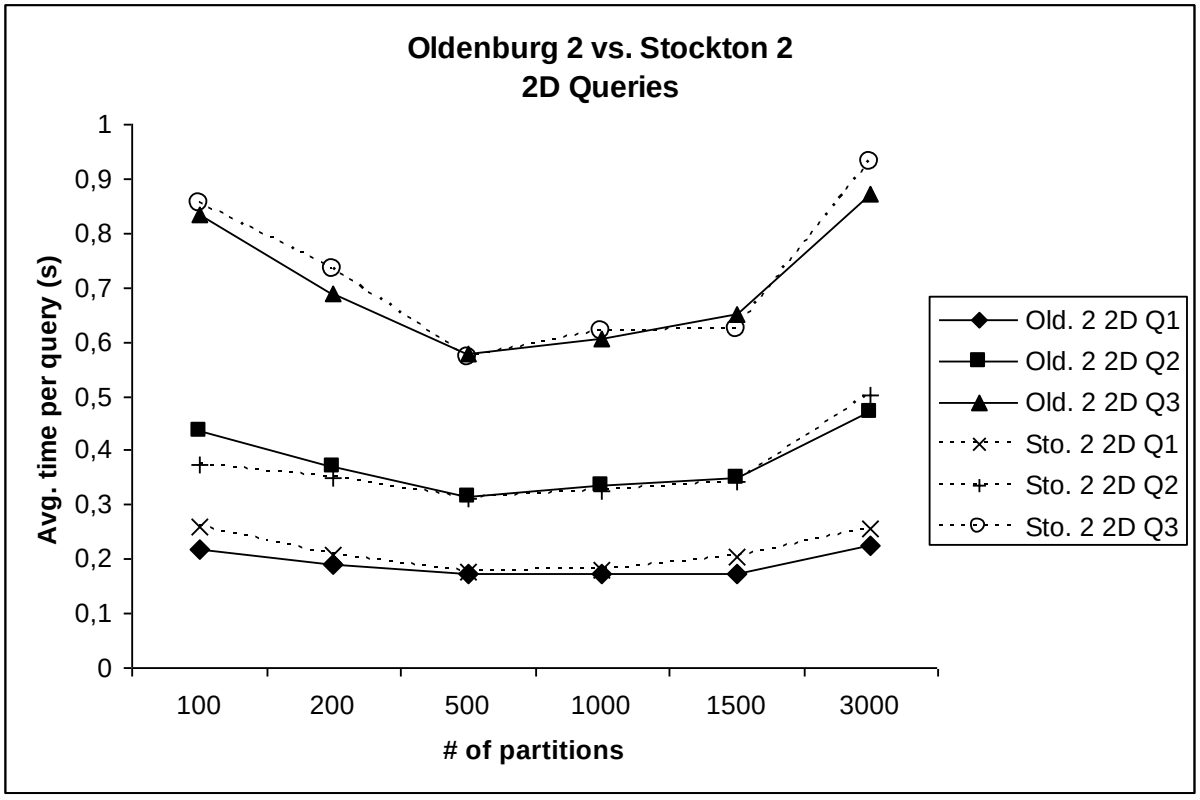
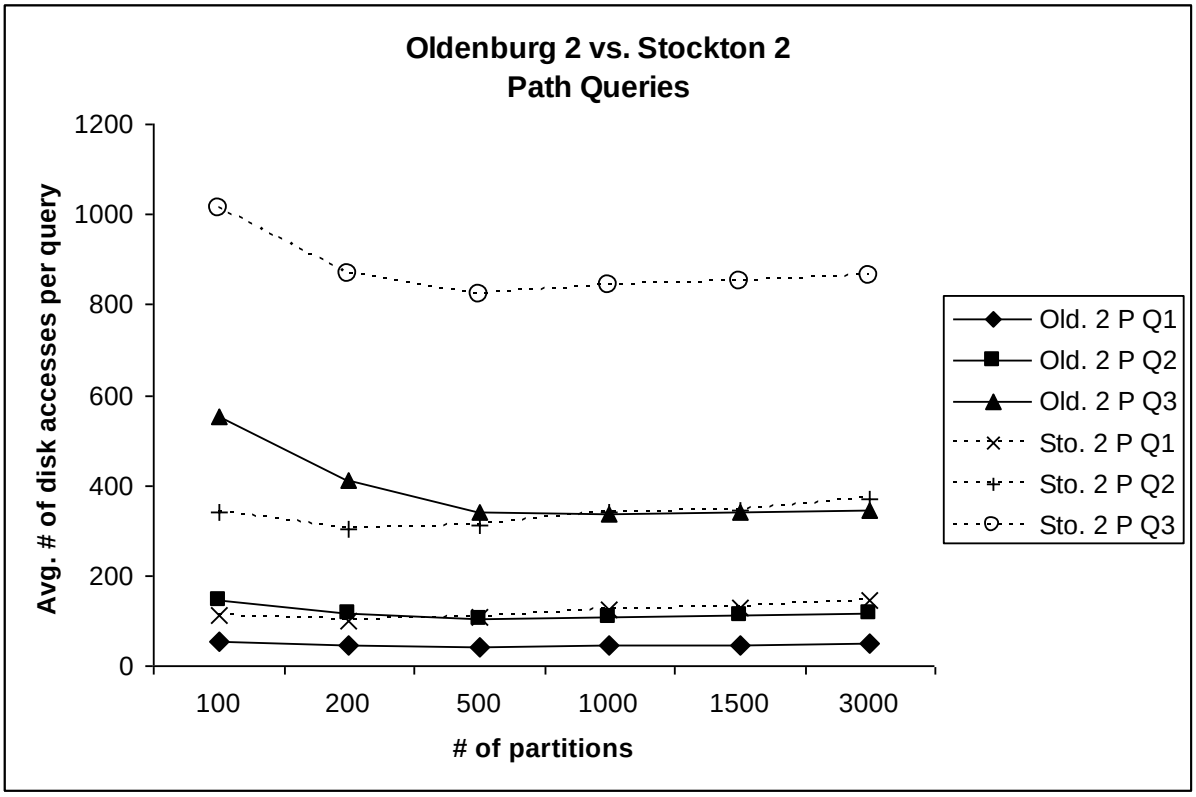


Figure V.10: PARINET query performance - 2D queries on large data sets



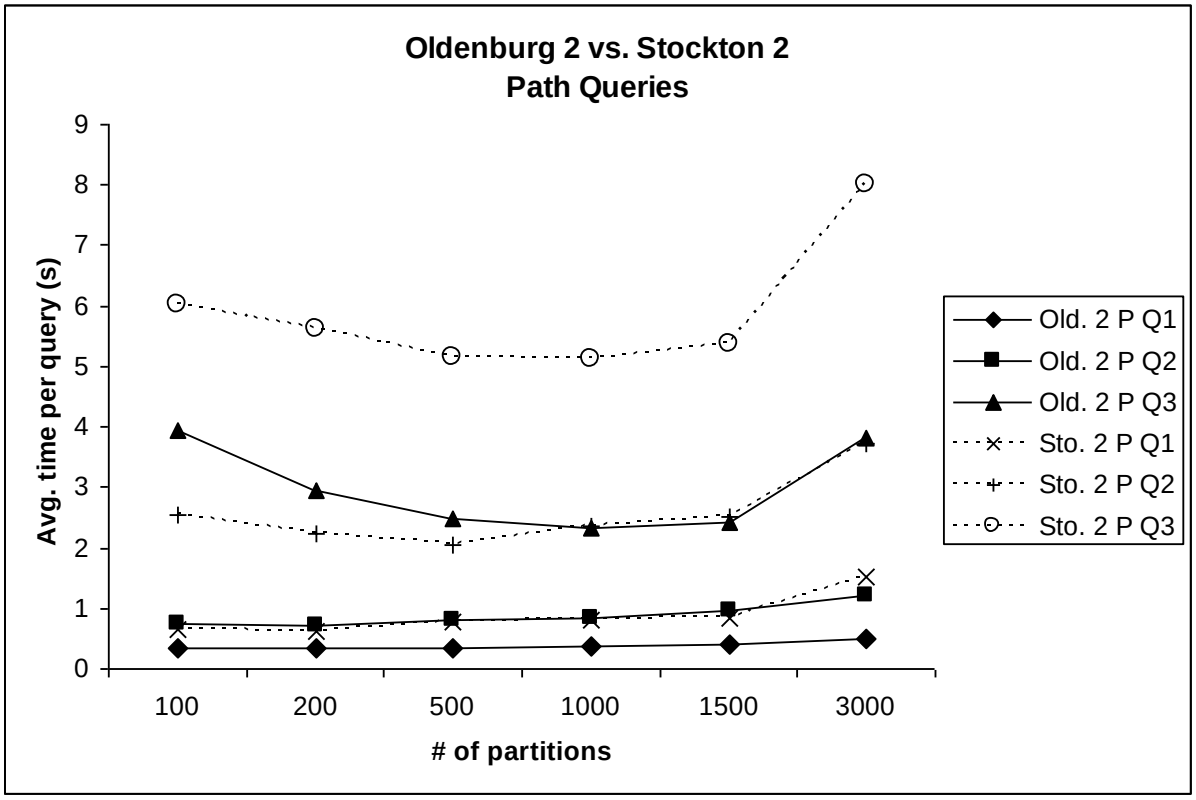


Figure V.11: PARINET query performance - path queries on large data sets

V.4.3.2 PARINET Cost Model Evaluation

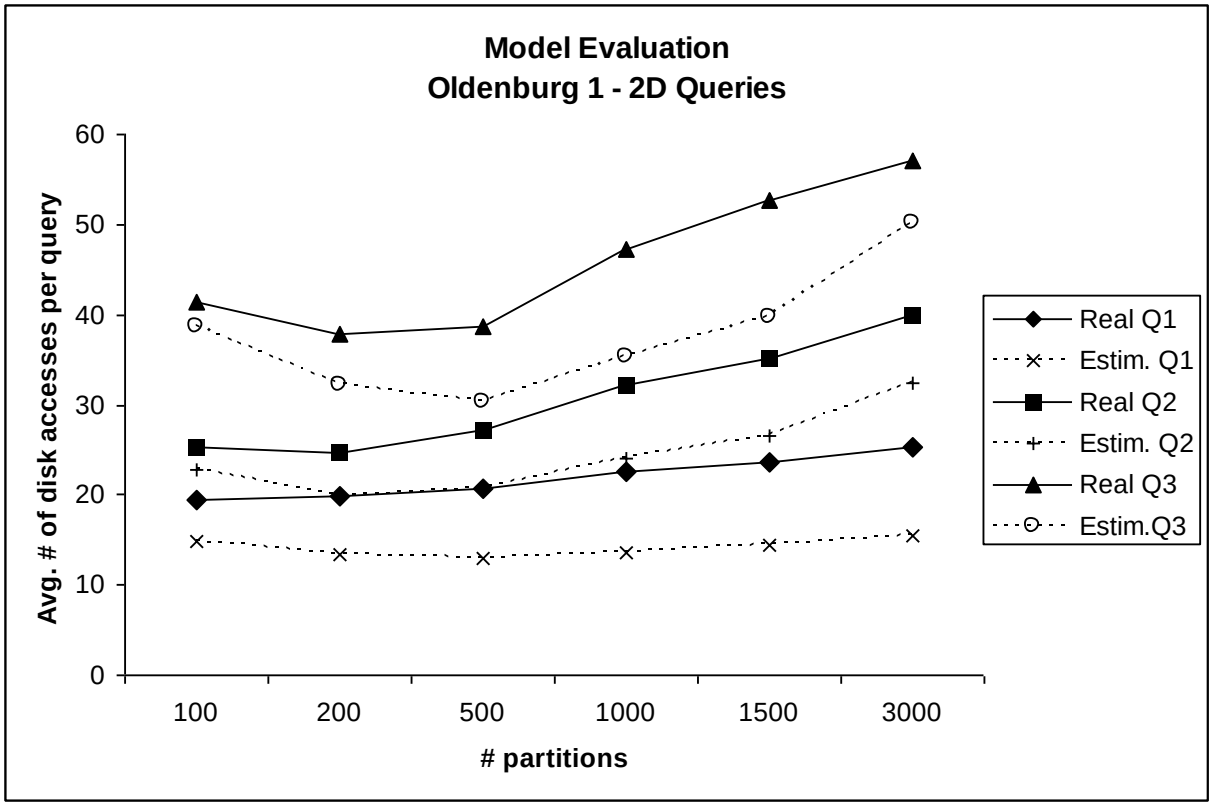
The above tests on the PARINET performances confirm our observations, i.e., given a data set, only some configurations of the index offer near optimal performance for a query load. The tuning of the number of partitions has a relatively small impact on disk accesses needed to process the query, for relatively small data sets and small query sizes. This aspect was observed for Oldenburg 1 and Stockton 1 data sets. Nevertheless, choosing a near optimal number of partitions can significantly improve the query execution time for large queries, even for small data sets. On the other hand, the number of partitions is important for the query performance for large data sets.

In Section V.3.3.2, we presented a cost model that estimates the number of disk accesses for a query load and a given index configuration. Our tests showed that the execution time of a query usually depends on the number of disk accesses. Therefore, the cost model can estimate the performance of our access method for a given configuration, without effectively constructing the index. This is very important considering that the index creation is costly. If the cost model is accurate, we can automatically find and materialize a good configuration among the possible ones.

In this section, we experimentally evaluate the proposed cost model. For all the tests presented in Section V.4.3.1, we also calculated the number of disk accesses by using Equation (6) defined in Section V.3.3.2. Practically, we implemented a program that takes as input the network graph with the data distribution for each road, a query load and a number of partitions, and outputs an estimated number of disk accesses. We consider a uniform temporal distribution of the data, which is a good approximation for the generated data sets. For non uniform data distribution, the real temporal distribution must be used, as in Equation (2), in order to obtain a good estimation of disk accesses.

The results are presented in Figure V.12 to Figure V.15. As we can see from these figures, the cost model offers a good estimation for the number of disk accesses. The cost model is much more accurate for large data sets. We think that this is due to fact that the DBMS generates additional disk accesses when managing partitioned tables. Such disk accesses are not considered in our cost model and they are more noticeable for small data sets when the number of disk accesses for a query is low.

For large data sets, the cost model is a bit optimistic for a high number of partitions and pessimistic when the number of partitions is small. Also, the model is works better for queries that require a large number of disk accesses, such as the path queries. The estimated near optimal number of partitions doesn’t always coincide with the experimental optimal, but is close enough. In conclusion, we think that the cost model is good enough to be used for tuning the index.



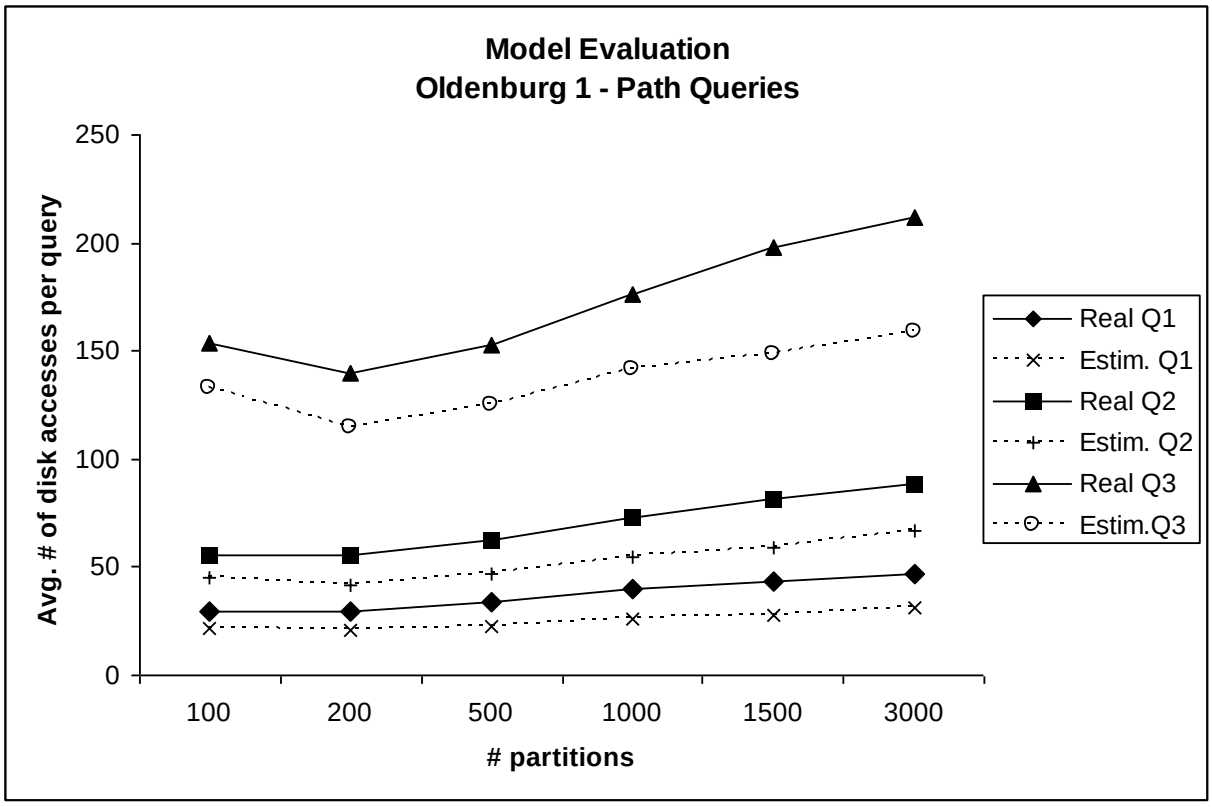
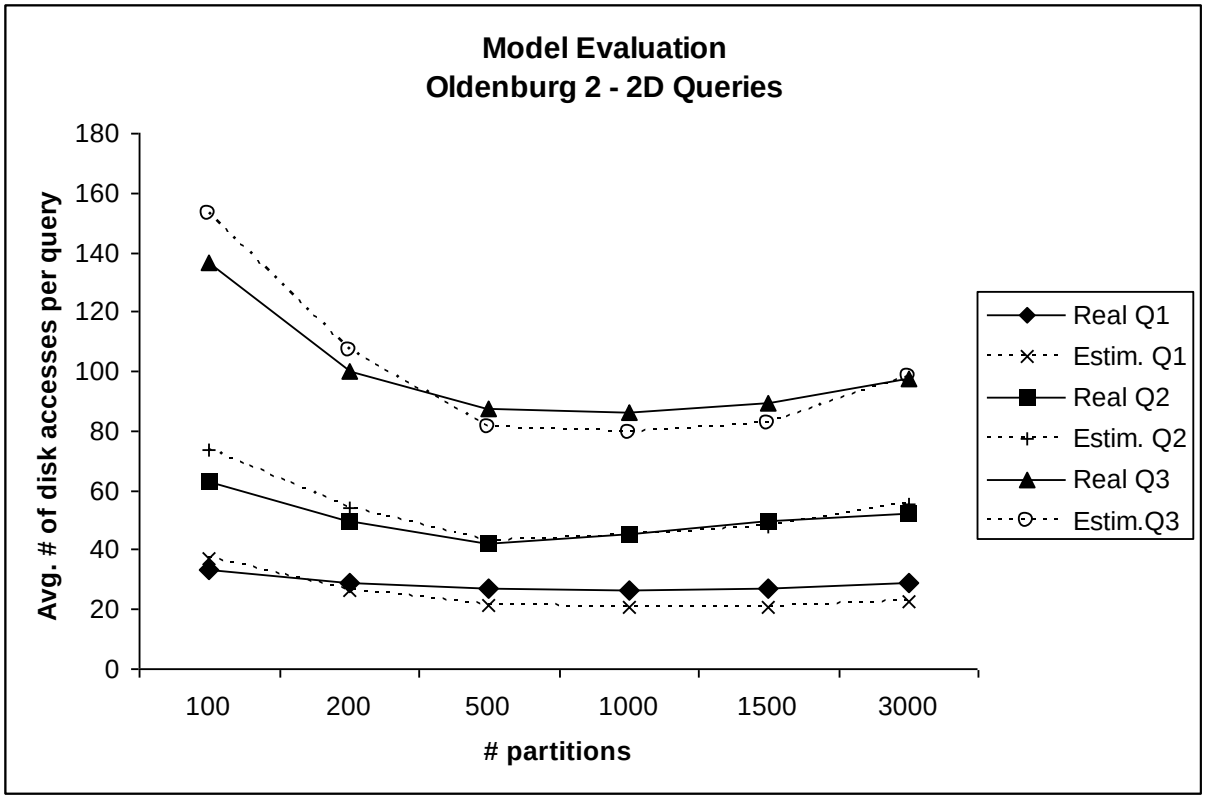


Figure V.12: PARINET cost model evaluation for small data set



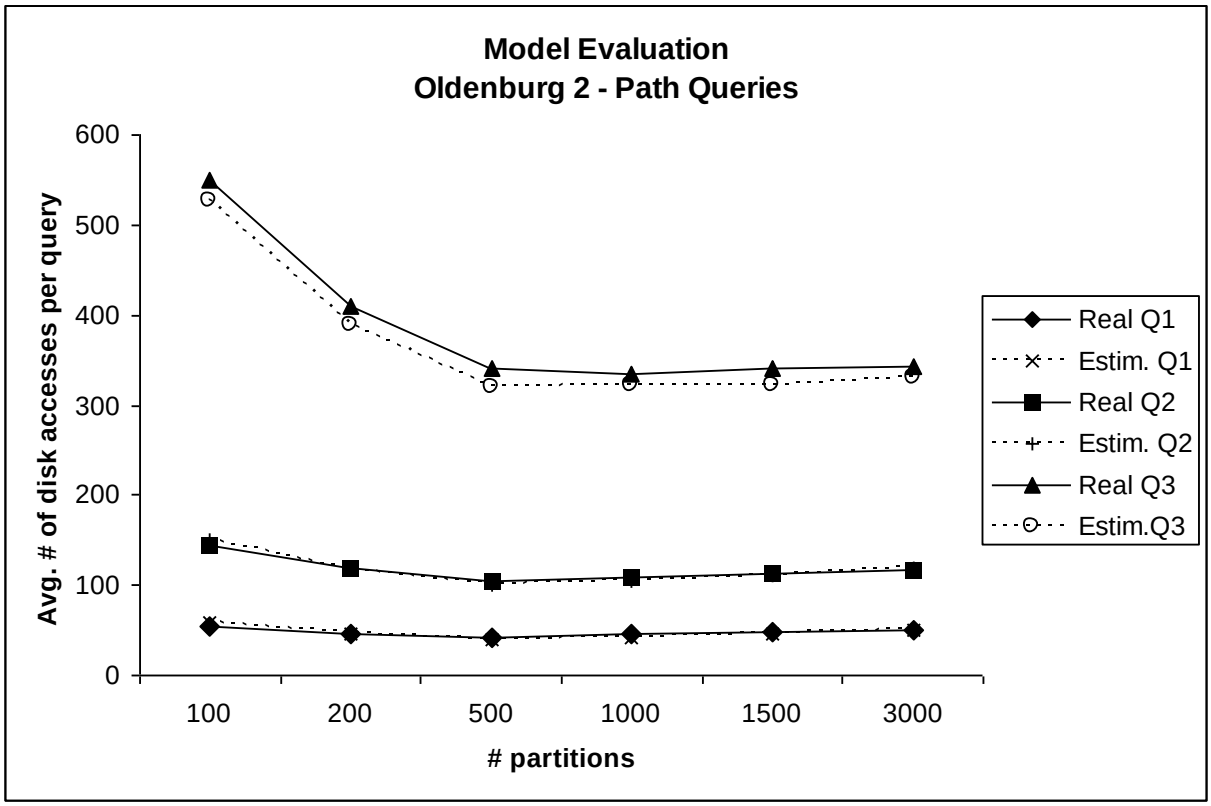
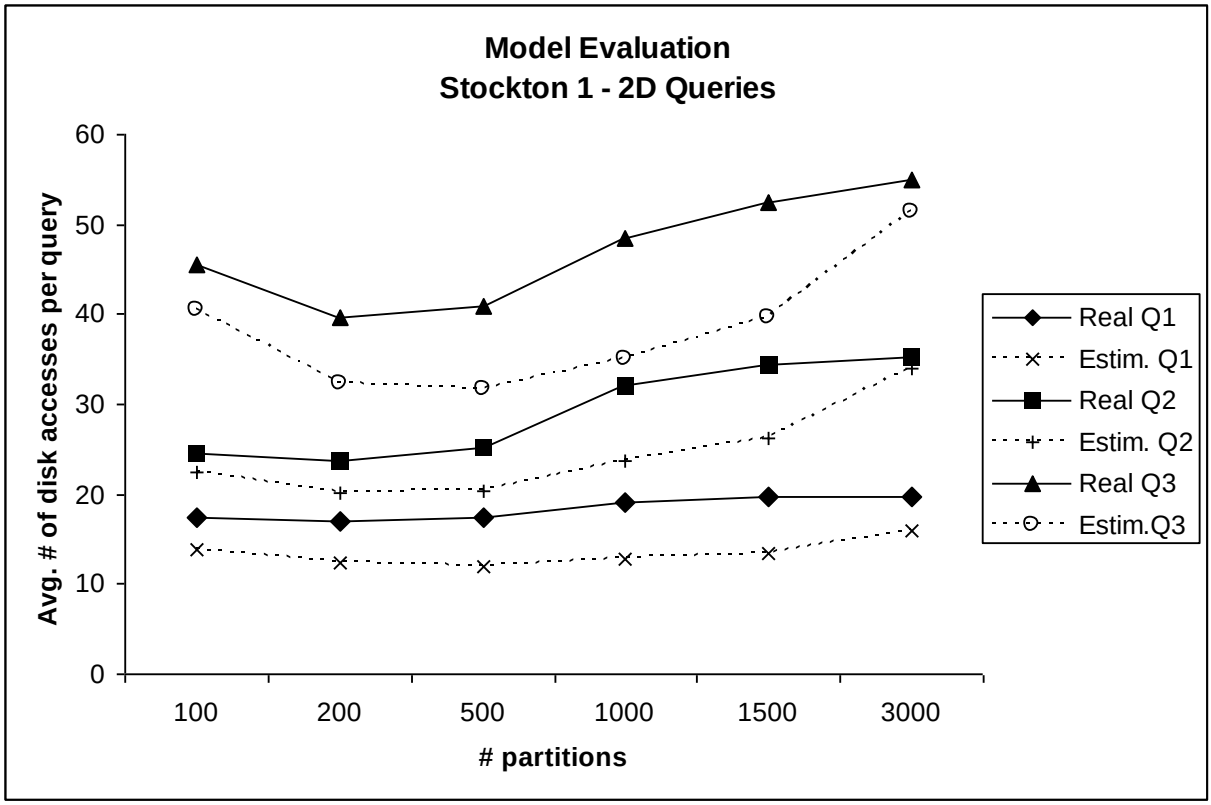


Figure V.13: PARINET cost model evaluation for large data set



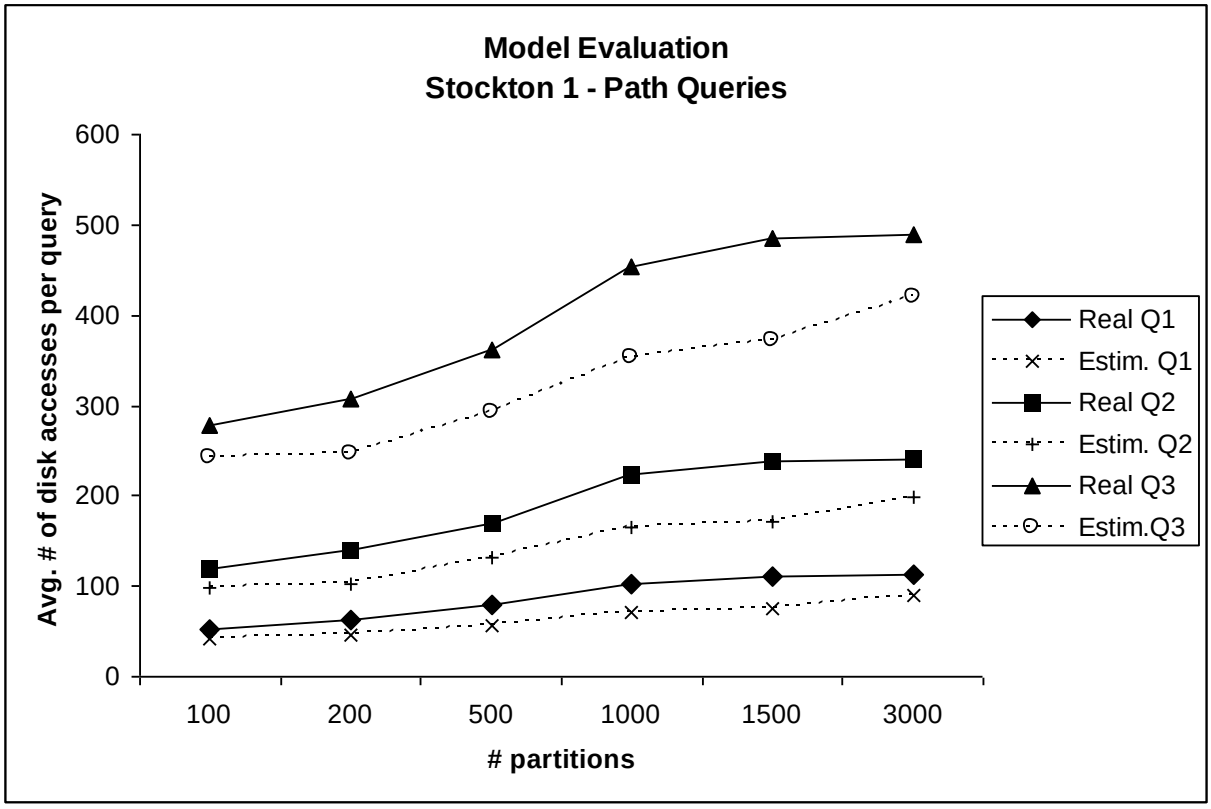
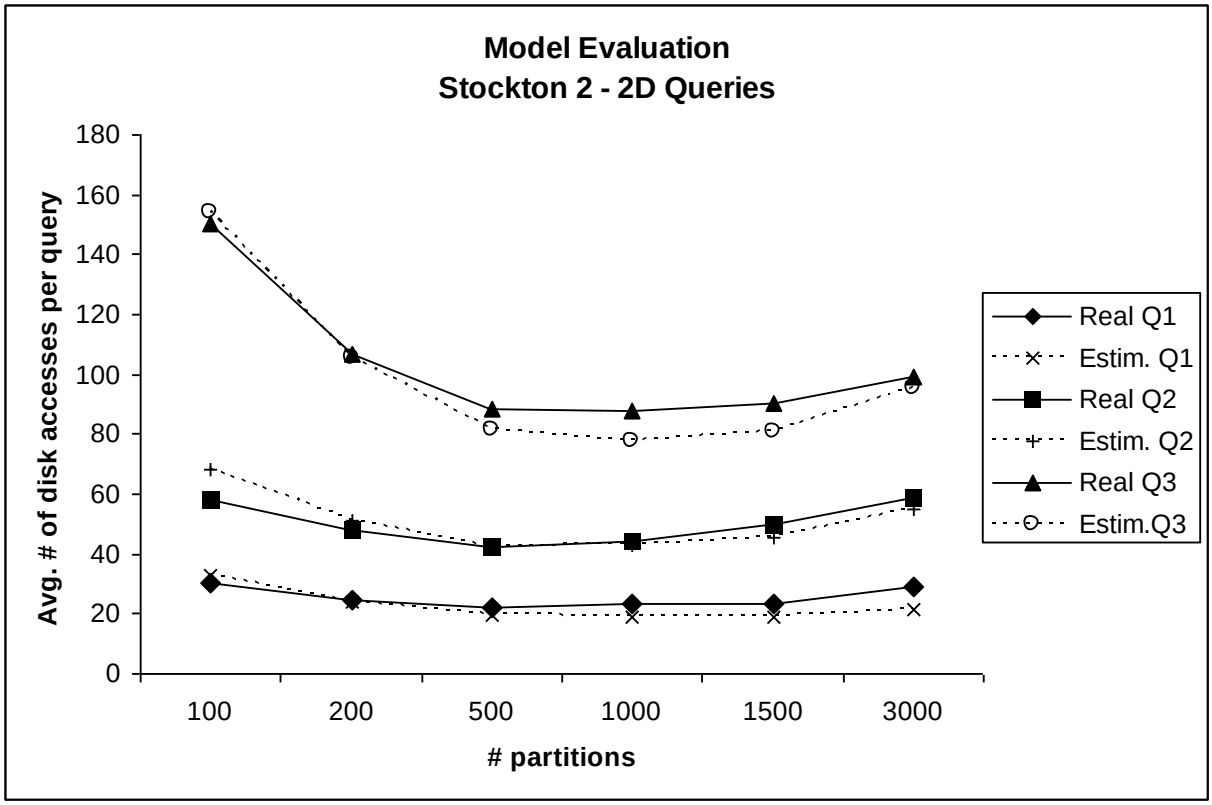


Figure V.14: PARINET cost model evaluation for small data set



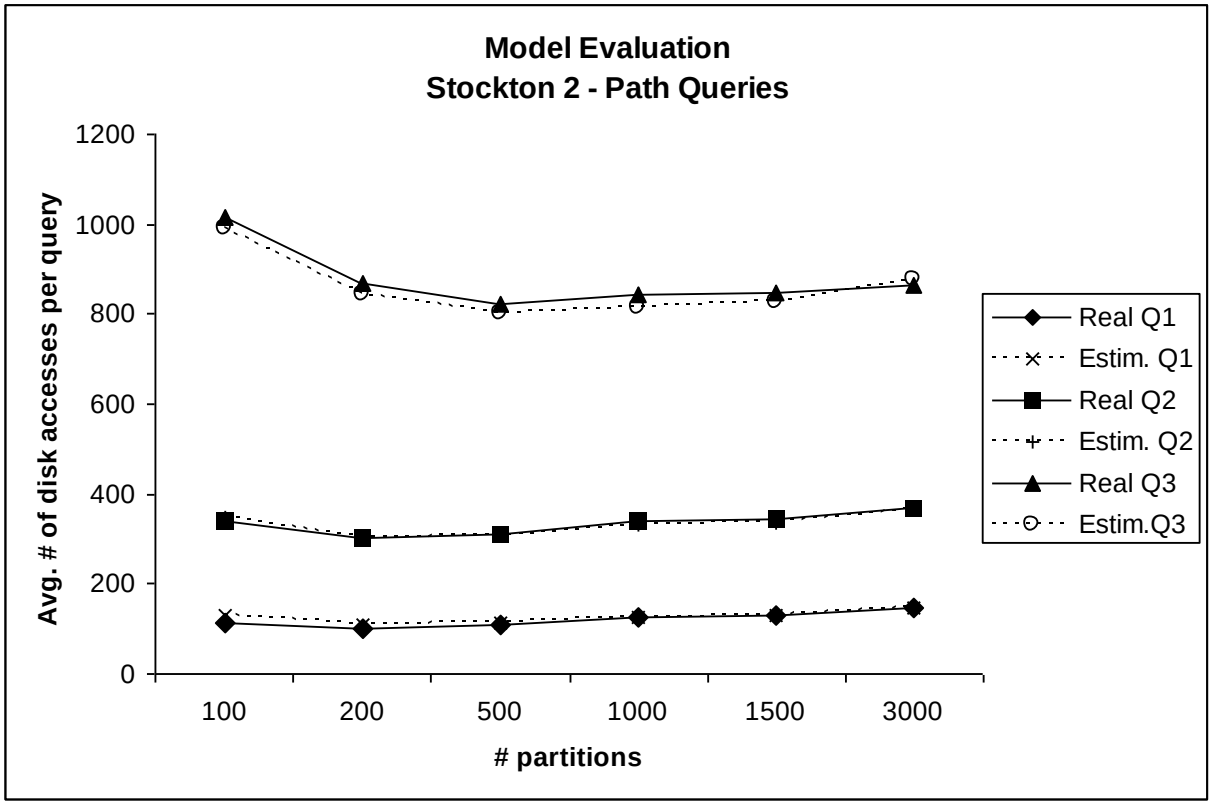


Figure V.15: PARINET cost model evaluation for large data set

V.4.3.3 PARINET Query Robustness

The proposed access method deals with historical data. It can be automatically tuned for optimal performance given a data set and a query load. However, even if the data is historical, this does not mean that the queries are known in advance. Moreover, the queries at a given time may differ a lot (different users may pose totally different queries), and the queries may change across time (during the day or week or month). In such cases, the index itself should be able to handle very different queries at the same time and should be able to adapt to changes over time. We are interested in a robust index structures whose performance that does not degrade much with reasonable variations between the expected and actual query sizes.

Regarding the index evolution over time, we propose the following model. A log (sample) of the past queries is maintained and periodically (e.g., each week) the cost program computes the relative distance between the present and the optimal cost, using the cost model. The index will be rebuilt if the distance exceeds a certain threshold value (e.g., 20%).

Nevertheless, the queries may vary a lot during a given time interval. In the remainder of this section, we analyze the index robustness with regard to the query size. For that, we use the tests in Section V.4.3.1 on the largest data sets (i.e., Oldenburg 2 and Stockton 2).

We consider that the index is tuned for the Q1 query load. We observe that the performance degrades when the query load changes to Q2 and Q3. For example, for a given data set 1500 partitions offer a near-optimal performance for Q1, while 1000 for Q2 and 500 for Q3 are near-optimal. We measure the performance degradation as: $D = \frac{P_Q^{local} - P_Q^{optimal}}{P_Q^{optimal}} \times 100$. The results for Oldenburg 2 and Stockton 2 (largest data sets) are given in Table V.6. In most cases the performance degradation is more than acceptable, for a variation between the expected and real query size that is important (up to 6 times more roads in the spatial part of the query and 4 times larger the extent of the time interval). The results indicate good robustness of PARINET to query variations.

Query set name	Avg. # of intersected roads by the queries	Temporal interval	Degrad. (disk accesses)	Degrad. (exec. time)
Old 2D Q1	9.5	2.5%	0%	0%
Old 2D Q2	34	5%	5.52%	6.25%
Old 2D Q3	60.2	10%	1.87%	4.85%
Sto 2D Q1	23.4	2.5%	0%	0%
Sto 2D Q2	96.3	5%	3.88%	4.16%
Sto 2D Q3	156	10%	0.32%	8.75%
Old P Q1	17	2.5%	0%	0%
Old P Q2	35	5%	4.52%	11.18%
Old P Q3	70	10%	1.69%	2.73%
Sto P Q1	60	2.5%	0%	0%
Sto P Q2	120	5%	2.64%	8.48%
Sto P Q3	241	10%	5.19%	9.32%

Table V.6: PARINET query robustness

V.5 Conclusions and Future Work

In this chapter we introduced a new access method called PARINET (PARtitionned Index for in-NEtwork Trajectories), for efficient retrieval of the trajectories of objects moving in networks. Its structure is based on graph partitioning and on indexing the partitions with composite B^+-trees, which are ubiquitous in the database world. PARINET can be easily integrated into any DBMS, which is an essential feature particularly for industrial or commercial applications. Unlike the existing approaches, PARINET relies on the expected query size and on the distribution of the data to be indexed, which can be known in advance given the fact that we deal with historical data. Thus, it is easily tunable and adaptable to any spatial distribution of moving object trajectories. The experimental evaluation under an off-the-shelf DBMS shows that our approach significantly outperforms the reference R-tree based indexes. Also, PARINET showed robust performance with the query size and scaled well with the data size, when tuned correctly. PARINET uses a cost model in order to adapt to a data distribution and a query size. The experimental results show that the cost model offers good results, suggesting near-optimal index configuration.

As future work, we intend to improve the method for an optimal handling of the index evolution in time. In addition, we plan to use a PARINET base access method for indexing several data types related to in-network MO trajectories. As indicated in Chapter IV (Section IV.6), *on-value spatial* operators can be indexed by PARINET like methods. However, the specificity of these data types may require more appropriate access methods.

Chapter VI: CALM Prototype

VI.1 Introduction

In the previous chapters we introduced a data model and a query language for mobile location sensor data. Moreover, some optimization aspects including indexing the trajectories of constrained moving objects were presented. Based on those proposals, we are currently developing a database extension that can manage this kind of spatio-temporal data. We name this prototype CALM (from the French acronym "mobile location sensors", i.e., "CApteurs à Localisation Mobile"). Although it is far from being complete, the current development state permits testing CALM with some "simple" queries. We partially implemented the system's core functionalities, i.e., the data types, most of the operations and some indexes. Orthogonally, we also implemented a graphical interface to visualize the results of some queries, e.g., to visualize (parts of) trajectories on a road map, and worked on the preprocessing of the raw data before integrating it in the DBMS.

Such a spatio-temporal DBMS (STDBMS) can be useful to a wide range of applications like transportation, traffic management, and environmental studies. As an example, many applications in the intelligent transportation system (ITS) research area require complex analysis of data collected in naturalistic driving conditions. A lot of data coming from ITS experiments is already available and a lot more is expected to be produced in the near future. Since we had access to this kind of data, most of our experiments were oriented towards the usage of the in-vehicle sensor data collected in naturalistic driving conditions.

In this chapter we present some practical aspects of the prototype CALM. In Section VI.2 we review the whole data-chain processing. This time we detail the upstream data processing (i.e., before integrating the data into the STDBMS) and the downstream data processing (i.e., post-processing of the results obtained in the STDBMS) from the STDBMS point of view. In the upstream, an important work is required before integrating the data in the STDBMS. In the downstream, tools that can help analyzing the data are needed, such as a graphical interface or a data mining tool. In Section VI.3 we evaluate our prototype by using a naturalistic driving data set and several query scenarios. We give some application perspectives and conclude this chapter in Section VI.4.

VI.2 Data-Chain Processing

As emphasized in the introduction, the database is an important component of the whole data-chain processing because it will contain a huge amount of data that will be quite unworkable if badly designed. Nevertheless, besides a proper spatio-temporal DBMS there are other aspects that need to be dealt with in the upstream and the downstream of the STDBMS. Figure VI.1 depicts the whole data-chain processing for a naturalistic driving application type (see Section III.2). The raw data coming from the natively embedded sensors in the vehicle and from other sensors (GPS, camera, etc.) are recorded by a data logger. These data have a simple format. For each trip, a new file is created. The file contains a temporal sequence for each sensor, i.e., the reported value at a given time. After a period of time, the recorded data files are gathered in a central database. The data are firstly map-matched by using a road network database. Then, it is transformed accordingly to the spatio-temporal data

model implemented in the DBMS. Finally, the data are integrated in the spatio-temporal server and become available for querying in an extended SQL-like language. Several services, e.g. visualization, analyzing tools, etc., can be based on the STDBMS, which may in turn be employed by the end-user applications.

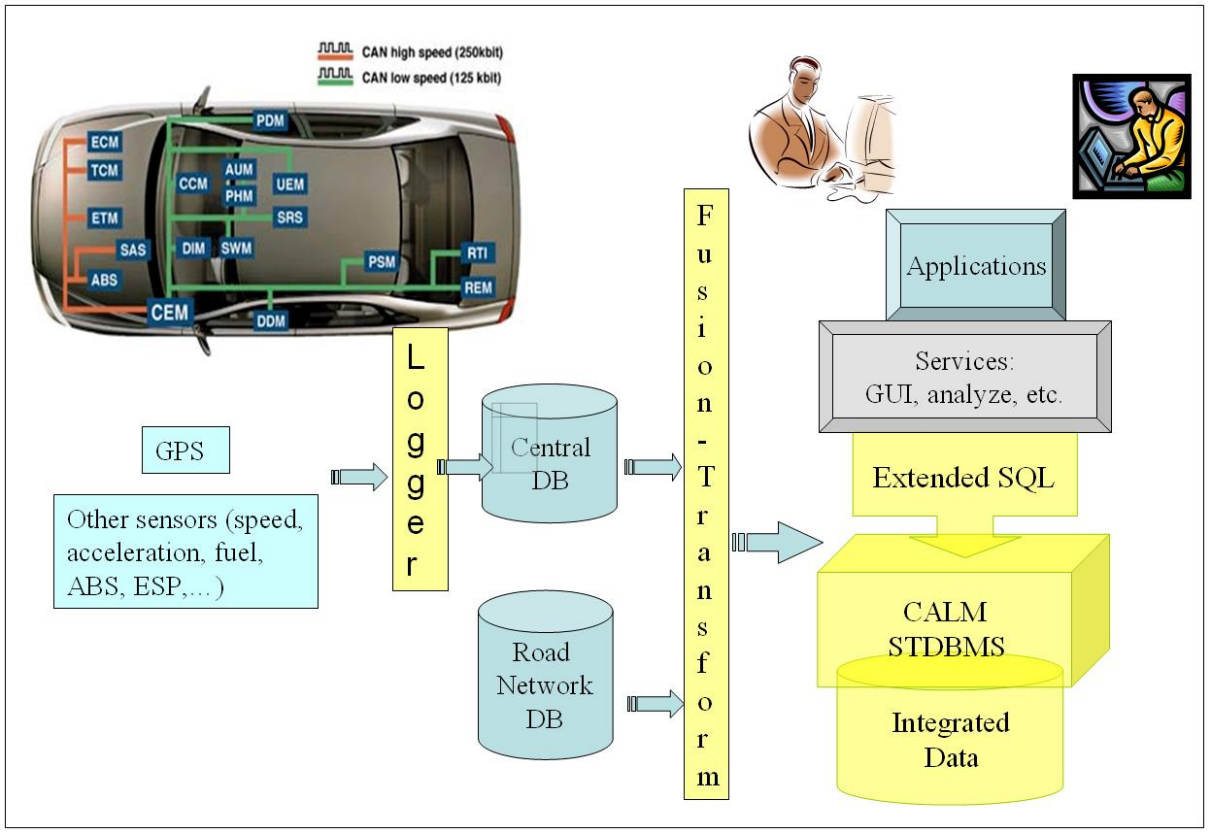


Figure VI.1: Data-chain processing in naturalistic driving applications

VI.2.1 Upstream Data Processing

The positions issued by the GPS sensor are two-dimensional coordinates in the latitude longitude reference system (Figure VI.2). Unfortunately, these data are not precise due to the *measurement error* caused by the limited GPS accuracy, and the *sampling error* caused by the sampling rate [109]. For example, it is not uncommon to have an error up to ten meters for standard commercial GPSs (Figure VI.3). A pre-processing step that matches the trajectories to the road network is needed. This technique is commonly referred to as *map matching* [113]. The network itself can contain errors. However, it is expected that these errors be less significant and less frequent that the GPS errors. Therefore, a usual approach is to trust the map and to match the GPS points to the road network.

After the GPS positions are matched to network positions (i.e., *gpoint* cf. Chapter II), we have to transform the time sequences into data objects that can be loaded in the STDBMS. For example, the sequence of GPS positions recorded for a given trip is transformed into a *moving(gpoint)* object (Figure VI.4). This is done by using linear interpolation between each two consecutive data records. The same process is applied to sensor data sequences. Therefore, spatial and temporal profiles (e.g., *gmoving(real)* and *moving(real)* object types) of sensor measures are associated to each trajectory. Once the data is integrated into the STDBMS, it can be queried in a SQL-like language (Figure VI.5).

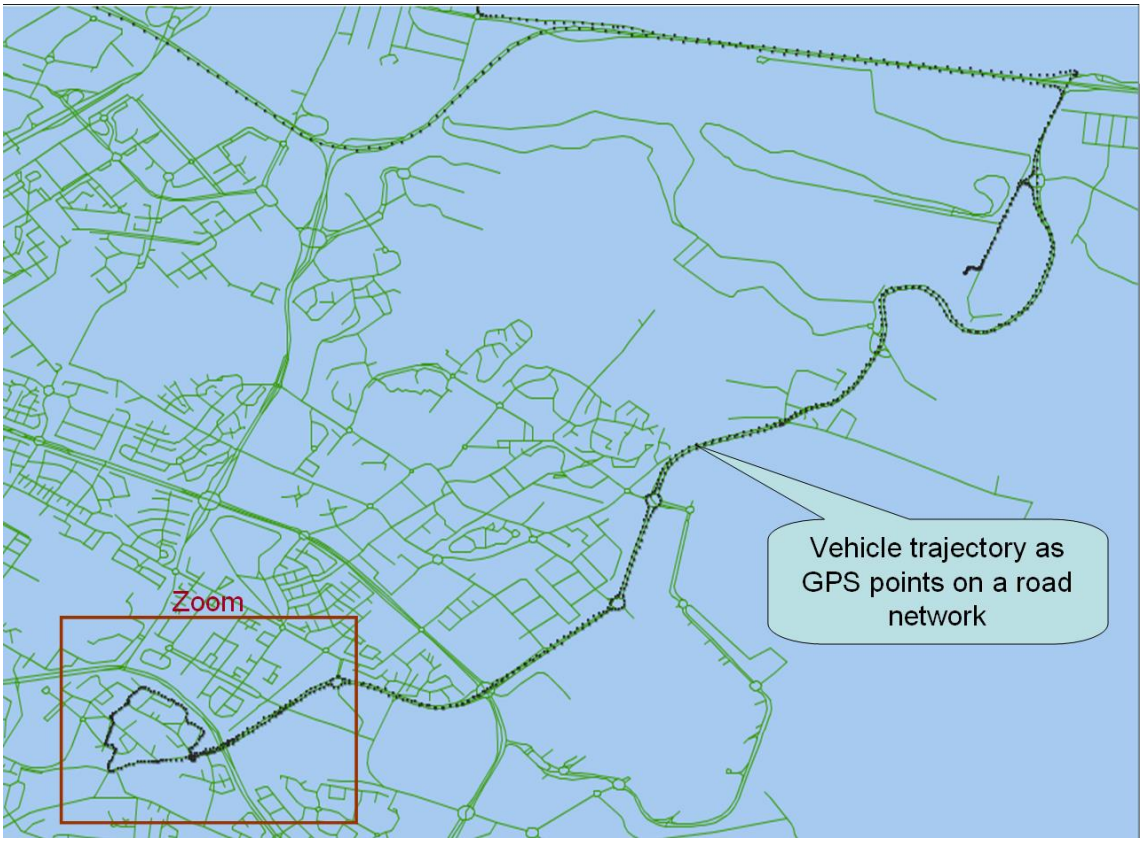


Figure VI.2: Trajectory GPS points and a road network

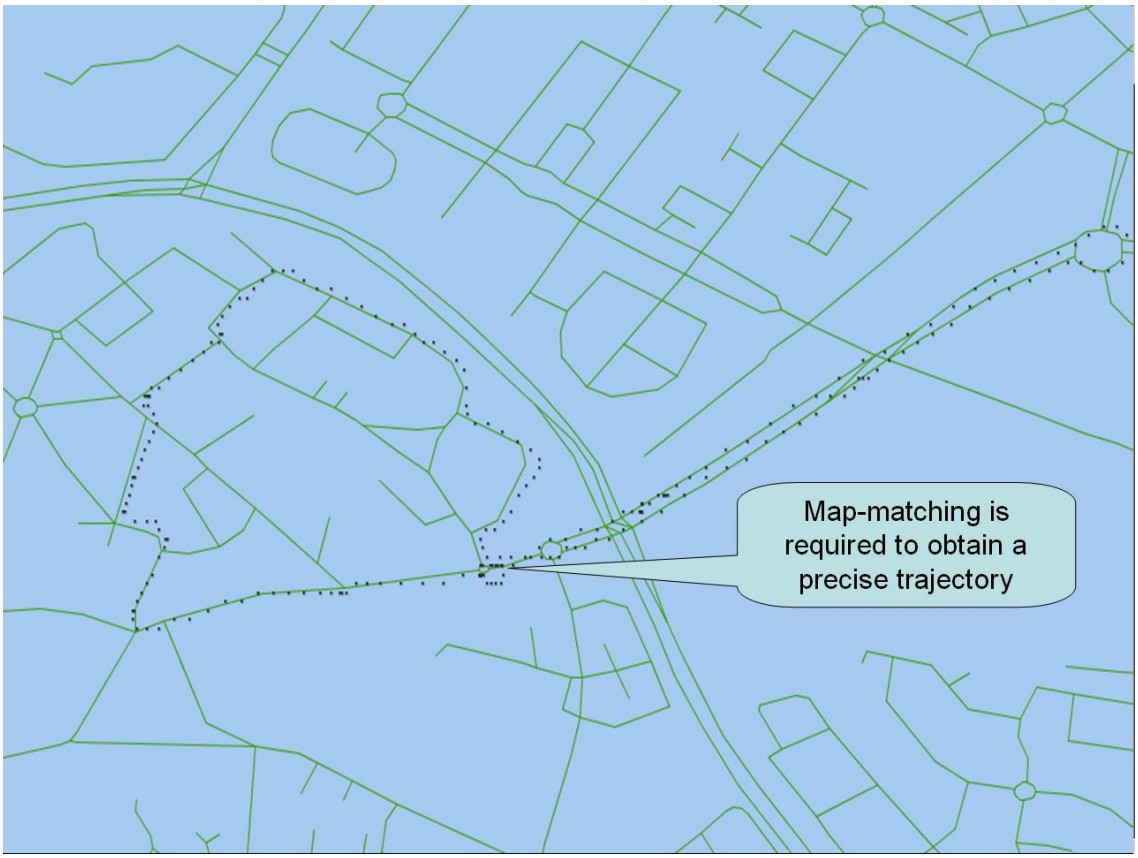


Figure VI.3: The GPS recordings and the maps are error prone

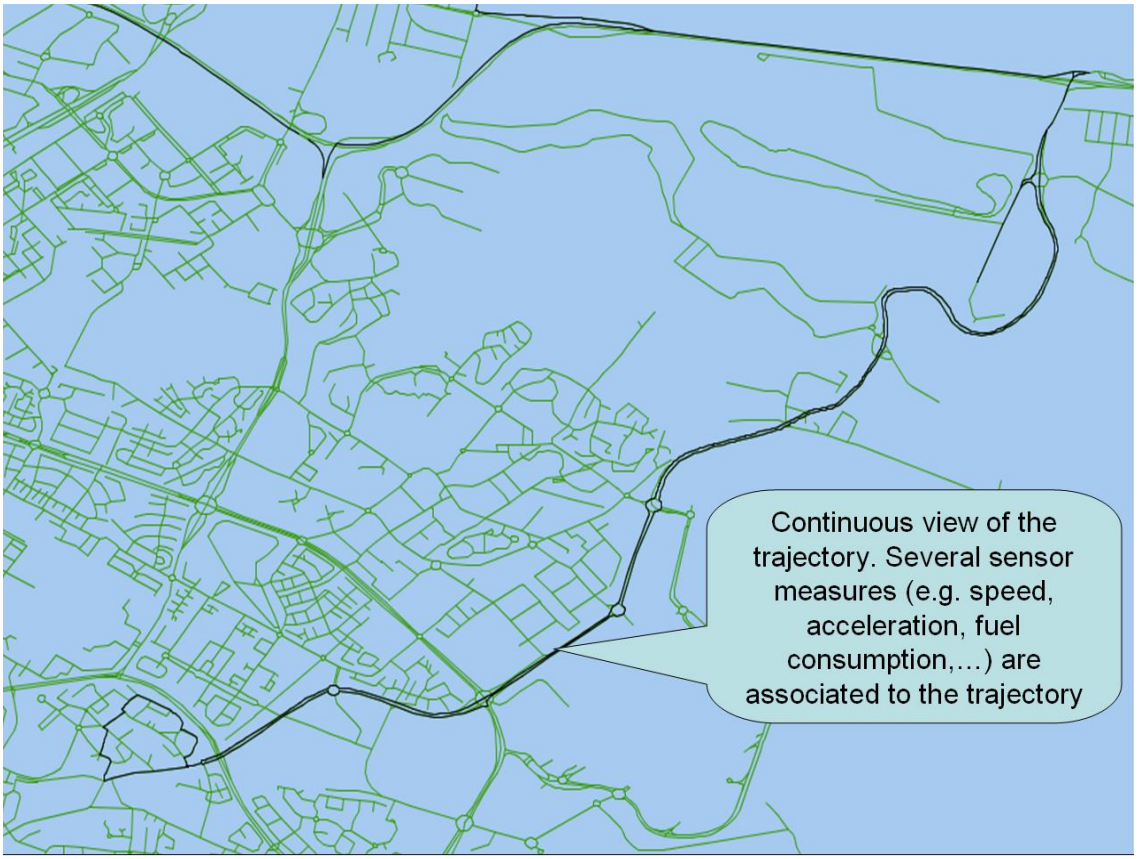


Figure VI.4: Continuous view of a trajectory in the network

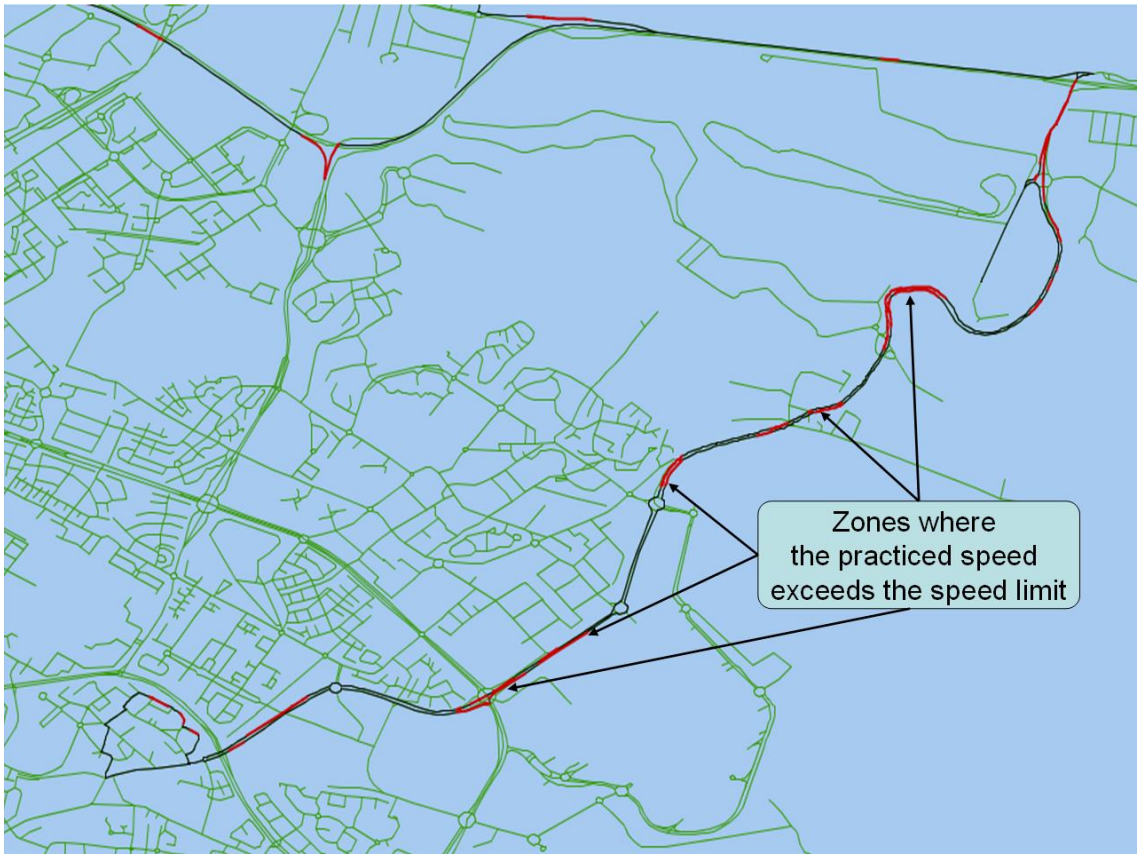


Figure VI.5: Visual analysis of sensor measures

VI.2.1.1 A Basic Map-Matching Algorithm

To do the map-matching we use a basic algorithm proposed in [110]. This approach takes advantage of the fact that the entire trajectory is known in advance. This corresponds to our case, where the trajectory is a sequence of historic position samples that need to be mapped. We present next this map-matching algorithm.

The map-matching algorithm uses a position-by-position and edge-by-edge approach. Given a sequence of two-dimensional points, the algorithm considers and matches each position in the sample. For each point p_i in the sequence, the list of candidate edges c_j is determined first. This list contains the edge matched to the previous position p_{i-1} and all its exiting edges. For example, in Figure VI.6 the candidate edges for p_i are c_1, c_2 and c_3, where c_1 is the edge matched to p_{i-1} and c_2 and c_3 are incident edges.

Then, for each candidate edge a matching score is computed as the sum of two *similarity measures* s_d and s_α. s_d indicates a measure of similarity based on the weighted distance between the position sample and a candidate edge, i.e., $s_d(p_i, c_j) = \mu_d - a \cdot d(p_i, c_j)^{n_d}$ [110], where μ_d, a and n_d are scaling factors. The distance between a point and an edge is either the perpendicular line distance if the projection of the point falls inside the edge, or, otherwise, the distance between the point and the closest endpoint of the edge. A smaller distance corresponds to a higher probability for matching the position to that edge.

The similarity in *orientation* s_α depends on the angle measured between the line segment $l_i = p_{i-1}, p_i$ and a candidate edge, and is computed by $s_\alpha(p_i, c_j) = \mu_\alpha \cdot \cos(\alpha_{i,j})^{n_\alpha}$ [110], where μ_α and n_α are scaling factors.

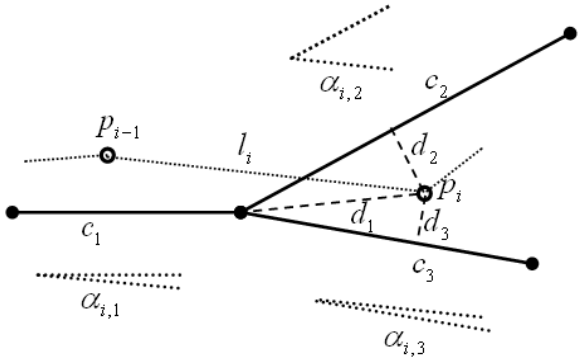


Figure VI.6: Incremental map-matching example

The scaling factors μ_d, n_d, a, μ_α, and n_α are used to weight between distance and orientation. They are empirically established. The final similarity score $s = s_d + s_\alpha$ is computed for each candidate edge and the matching is done to the edge having the highest score. The algorithm advances to the next position sample only if the projection of the current point on the matched edge falls between its end points. Otherwise, if the matched edged for p_i is the same as for p_{i-1}, then it will be excluded from the candidate list of edges in the next iteration.

This basic algorithm is improved in [110] with a recursive look-ahead strategy. The idea is to use a few more future position samples in order to have a more global view when choosing among candidate edges. This is possible since all the position samples are known in advance. Thus, for each candidate edge c_j, the best candidate among its exiting edges $c_{j,k}$ is recursively computed. The best matching edge in the original candidate set will be the one with the best subpath score. The number of edges in the look-ahead is fixed. In [110] they propose a look-ahead of four edges (the candidate edge plus three more edges), as a good trade-off between map-matching quality and computational cost.

An example of map-matching with a look-ahead of two edges is given in Figure VI.7. For the position sample p_i the score for each candidate edge c_1, c_2 and c_3 is computed first. In addition, one determines that the best candidate edge for p_{i+1} is $c_{2,1}$ if p_i is matched to c_2, or $c_{3,1}$ if p_i is matched to c_3. The final score for p_i is then computed as the sum of the scores of the best subpath of each candidate edge, i.e., $s(p_i, c_j) = \sum_{k,l=0}^{depth} s(p_{i+l}, c_{j,k})$ [110]. For the point p_i and the candidate edges c_2, c_3 the scores are: $s(p_i, c_2) = s(p_i, c_2) + s(p_{i+1}, c_{2,1})$ and $s(p_i, c_3) = s(p_i, c_3) + s(p_{i+1}, c_{3,1})$. The basic algorithm finds c_3 as the matching edge for p_i, whereas p_i will be matched to c_2 by the algorithm that uses the look-ahead strategy.

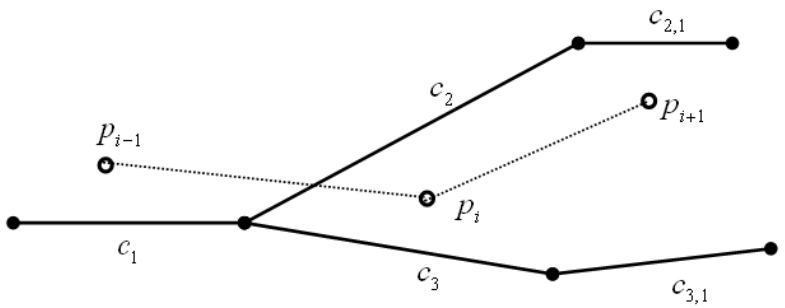


Figure VI.7: Incremental map-matching example with look-ahead

Our experimental evaluation showed that in practice, this basic map-matching algorithm is relatively fast and offers good results if the position sampling rate is sufficiently high (e.g., each two seconds for a vehicle moving in a road network).

IV.2.2 Downstream Data Processing

The STDBMS permits analyzing the sensor data by means of queries. These are very useful for performing filtering and calculations guided by the user. However, this type of analysis remains limited. Data mining tools for example can offer a global analysis of mobility traces. Such approaches can be built entirely on the STDBMS. The data mining algorithms typically use aggregate data and are costly. A direct analysis of trajectories of moving objects, which are characterized by their large volumes of data, is not always feasible. Therefore, a higher level representation of the data (i.e., a symbolic representation) is necessary for global analyses. Hence, the same data can have several representations in the STDBMS corresponding to different types of analysis. We had already started to work in this direction (i.e., in-network trajectory clustering [36]). However, we do not detail these aspects here.

Moreover, even for simple queries, the DBMS can not be very useful to the end user unless the result of the query is numerical. As indicated in Section III.5.1, database extensions should include user interface extensions to handle the presentation of data types. In this section, we present the graphical user interface (GUI) that we developed to display the spatio-temporal data types in our system.

We use Oracle's MapViewer and *GeoServer* [111] to handle the presentation of data types in CALM. However, most of the GUI features are available with GeoServer. GeoServer is an open source software server written in Java that allows users to share and edit geospatial data. Designed for interoperability, it publishes data from any major spatial data source using open standards. Being a community-driven project, GeoServer is developed, tested, and supported by a diverse group of individuals and organizations from around the world.

GeoServer permits to display some of the satellite or road maps available on the web, such as GoogleMaps [112]. Google offers this service free of charge, without viewing limitations. Hence, we can use GoogleMaps as a base map, which offers a better readability of test areas. However, the use of online maps implies the need of an internet connection.

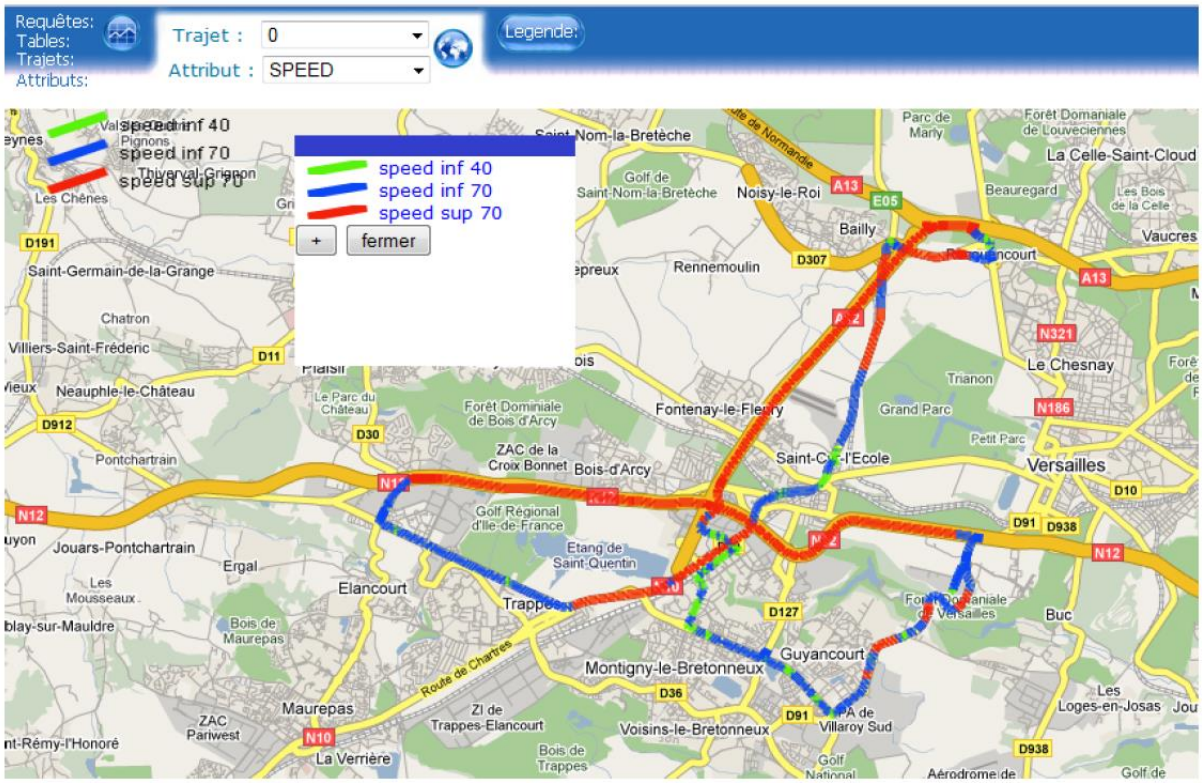


Figure VI.8: Variation of a measure along the vehicle's trajectory

CALM has three important data types that need to be visualize. The spatio-temporal trajectory of a moving object is contained in a *moving(gpoint)* data object. For this data type an animation simulates the movement of the object along the road network. Another possibility is to visualize the entire trajectory at once, i.e., the projection in space of *moving(gpoint)* which is a *gline*, as depicted in Figure VI.8.

The other two important types in the system are the spatial and the temporal profiles of measures, i.e., *gmoving(real)* and *moving(real)*. To handle the presentation of spatial profiles of measures, we proceed by associating colors to interval values. Figure VI.8 for example,

gives the variation of the speed along the trajectory. The zones that are colored green correspond to a speed inferior to 40 km/h, the blue ones represent values in the interval 40 km/h to 70 km/h, and the red ones indicate the places where the vehicle's speed is above 70 km/h. Such rules can be defined by users. A number of rules can be grouped to form a style, which in turn can be applied when visualizing the data.

The presentation of temporal profiles of measures is much simple. We use classic graphics to represent such profiles (Figure VI.9). Also, one can choose between the temporal variation of a measure and the variation of a measure with the traveled distance (Figure VI.10). Zooming on these graphics is another feature that is supported by the current GUI prototype (Figure VI.9).

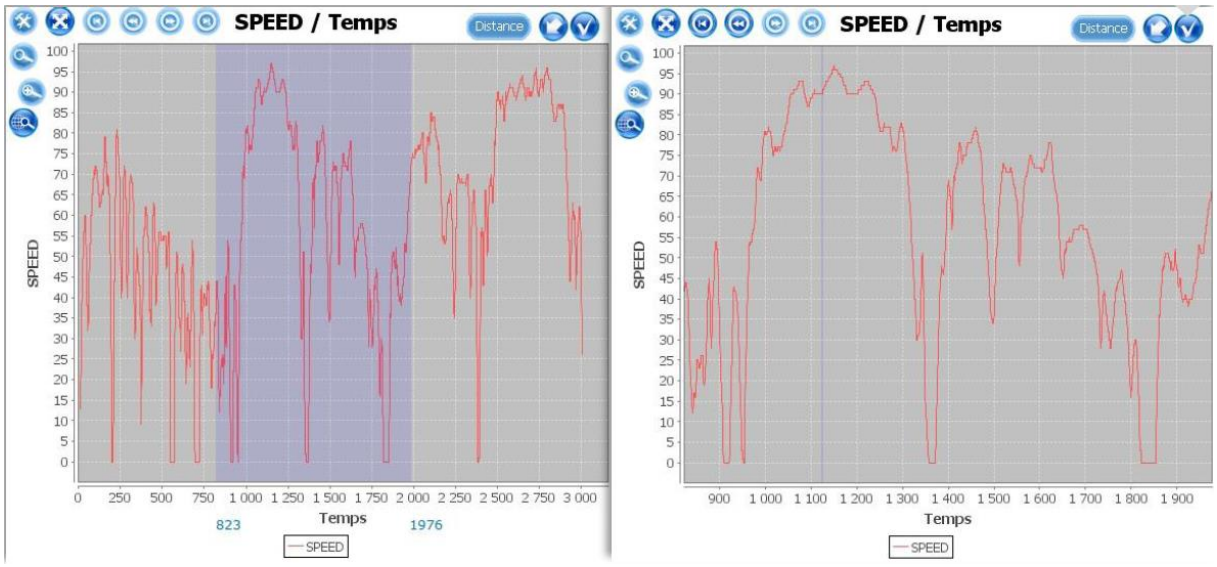


Figure VI.9: Temporal variation of a measure

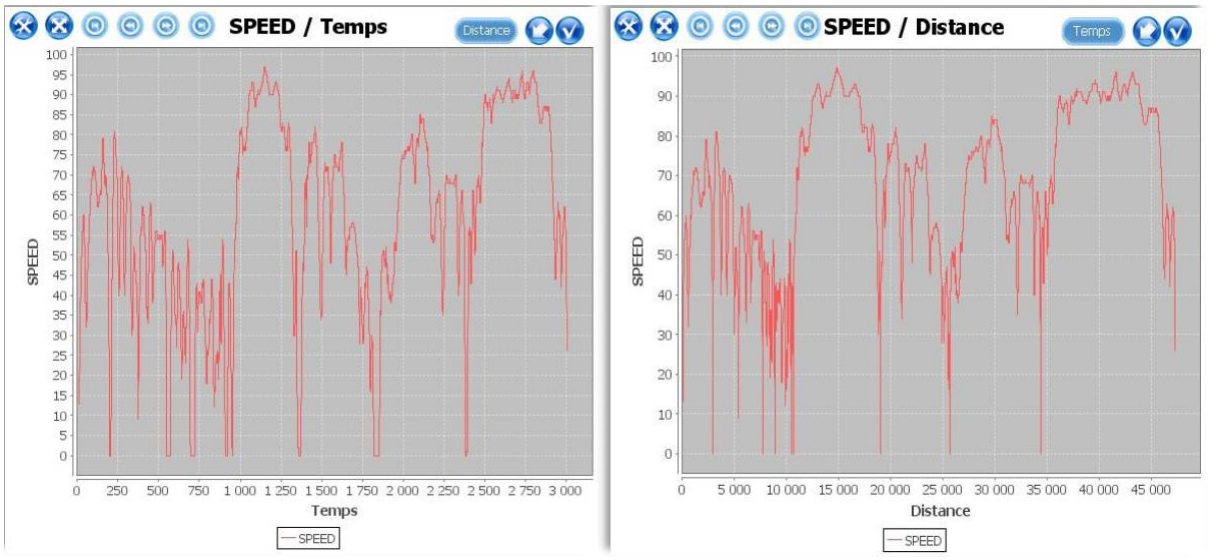


Figure VI.10: Variation of a measure in time or by distance

The spatial and temporal profiles of a measure are two complementary views that give the evolution of sensor values along the spatio-temporal trajectory of the moving object to which

the sensors are attached. A practical feature in our graphical interface is the dynamic linking between the spatial and the temporal profile of a measure (Figure VI.11). Thus, the user can select a measure at a certain time instant and automatically obtain the network location corresponding to this time instant, and vice-versa.

Figure VI.11: Dynamic linking between the spatial and the temporal profile of a measure

VI.3 Experimental Evaluation

As indicated in the first part of this chapter, CALM is only a prototype. There are several features that still need to be implemented or optimized in the system. In particular, we are still working on the implementation of the aggregate functions and on the GUI. At this point, a full experimental evaluation of the database is not possible. Moreover, a good benchmark can not be performed without realistic data sets. Synthetic data sets can be used, especially for performance evaluation, but they can not completely replace real data sets. However, at this time it is not easy to obtain this type of spatio-temporal data.

In this section we present a simple evaluation of our prototype, which is based on two query scenarios and on a (small) real data set coming from a complementary study in the LAVIA project [35]. First, we introduce the LAVIA system and the available data set and then we query it and present the results.

VI.3.1 The LAVIA System

The French Ministry of Transport launched in early 2000 an important program of experimentation and evaluation in order to better appreciate the effects of intelligent speed adaptation system in terms of drivers' acceptance and effects on their driving behaviors. In this context, the LAVIA device (from the French acronym "Speed limiter that adapts to the speed limiter", which is equivalent of foreign ISA - Intelligent Speed Adaptation), was also evaluated.

The LAVIA system is based on the manual speed limiting devices appeared as an accessory on the French cars in the early year 2000. This system ensures that the driver will

not exceed a pre-selected speed. Above this speed the accelerator is deactivated. The system can nevertheless be switched off in an emergency by pressing hard once on the accelerator. It is a short step to move from this to a system that automatically sets the vehicle's highest speed at the speed limit (legal speed) for the current location. For this, LAVIA uses a GPS and a number of in-vehicle sensors (e.g. the odometer and the gyrometer) (cf. Figure VI.12). By using appropriate map-matching techniques, the vehicle's position on the road network is computed. Finally, once the road has been identified, the on-board computer can find the speed limit on the road by consulting an on-board speed database with the speed limits of all the roads or streets in the region. In the sequel, we will show how our extended DBMS can be used to assess the impact on the driving behavior of such a device.

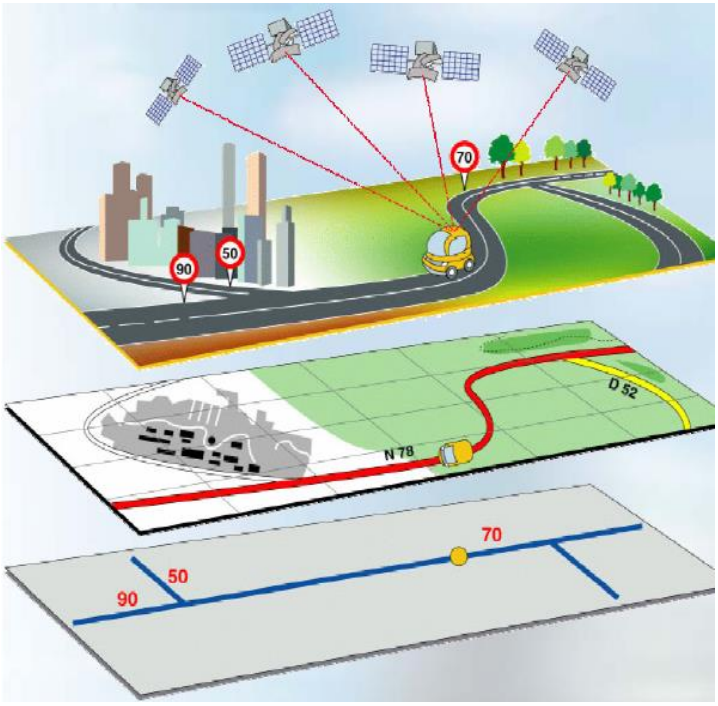


Figure VI.12: LAVIA functioning

VI.3.2 Tested Data Set

Our tested data set consists of eight trips of two drivers. Each person did the same trip four times. The duration of a trip is approximately 45 minutes and its length is approximately 47 km. We mention that the exterior conditions (trip time, congestions, weather, vehicle weight, engine temperature, etc.) were similar for each trip. For each driver, the four trips correspond to four driving styles: normal, nervous, economical and LAVIA.

The driver is instructed before each trip. For a normal driving style, there are no specific recommendations. This style corresponds to a week-end leisure driving, when the driver is not rushed. For a nervous style, the driver should drive like when he is hurried and consider the time factor as a priority, of course without taking any risks. The economical style intends to minimize the fuel consumption by pursuing the following recommendations: change to a superior gear as soon as possible (e.g. from 2500 rpm for a petrol engine); maintain a constant speed at the highest possible gear; anticipate the decelerations in order to avoid strong breaking; decelerate softly by releasing the gas pedal; cut the engine for all stops longer than one minute. Finally, the LAVIA driving corresponds to a normal style with the LAVIA system active.

The described data set is loaded into the database in form of a relational table containing the introduced object types. Each row in the table corresponds to a trip and contains the spatio-temporal evolution of the vehicle (i.e., the trip), and the spatial and temporal evolutions of each sensor embedded in the vehicle. The relational table has the following structure:

```
vehicle_trip(mo_id: int, trip: moving(gpoint),
g_speed: gmoving(real), t_speed: moving(real),
g_acceleration: gmoving(real), t_acceleration: moving(real),
g_gasPedal: gmoving(real), t_gasPedal: moving(real),
g_RPM: gmoving(real), t_RPM: moving(real),
g_gear: gmoving(int), t_gear: moving(int),
g_odometer: gmoving(real), t_odometer: moving(real),
g_ABS: gmoving(bool), t_ABS: moving(bool),
g_breakSwitch: gmoving(real), t_breakSwitch: moving(real)
g_fuel: gmoving(real), t_fuel: moving(real))
```

The underlined words represent data types in the system. For each sensor, we have two views of the registered measure: a spatial view (prefixed with g_) and a temporal view (prefixed with t_). The spatial view offers the evolution of the measure as a function of the vehicle's position on the road network (graph). The temporal view is a function of time.

VI.3.3 Query Examples

VI.3.3.1 Generic Queries

Given the above data set, some possible queries are:

Q1: *What are the practice speed and the legal speed for a given trip (e.g. economical drive) and a given route?*

Q2: *Compare the practice speed for a LAVIA drive and a normal drive with the speed limit for a given route.*

Q3: *How many times did the driver brake for a given trip?*

Q4: *What is the average fuel consumption for a given trip?*

Q5: *Where does the practice speed exceed the speed limit for a given LAVIA trip?*

Q6: *Where does the practice speed exceed with five percent the legal speed for a given LAVIA trip?*

Q7: *What is the percentage of the time passed above the speed limit for a given trip?*

Q8: *What is the percentage of the trip length passed above the speed limit for a LAVIA trip?*

In the sequel, we rewrite the above queries in the extended SQL language (cf. Chapter II and III) and present the obtained results for the tested data set. The proposed operations are represented with bold characters.

```
Q1: SELECT at(g_speed, trajectory(&legalSpeed)), &legalSpeed
FROM vehicle_trips
WHERE mo_id = &anEconomicalTrip;
```

Here, the *&legalSpeed* is a *greal* type variable that gives the profile of the speed limit along a certain network path. We first obtain the network path by using the **trajectory** function, which is a projection into the space. Then, we select the practice speed **at** this network path for an economical trip for example. In order to have a visual result, which is more readable, we trace the curves of the two selected profiles in Figure VI.13.

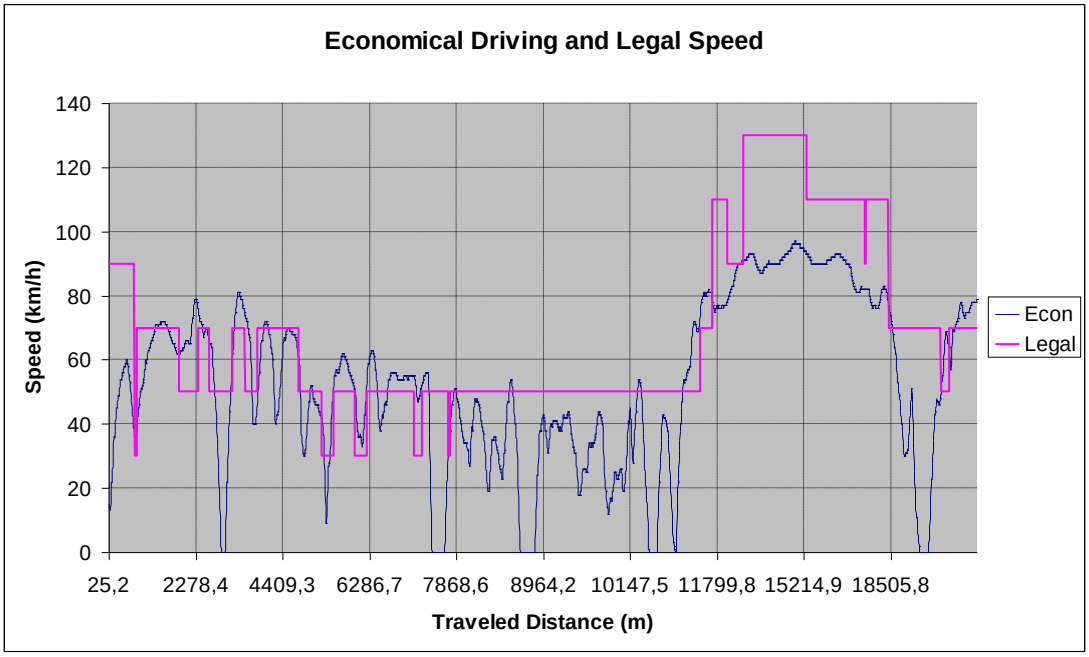


Figure VI.13: Comparison between the practiced speed and the legal speed for an economical trip

```
Q2: SELECT at(g_speed, trajectory(&legalSpeed)), &legalSpeed
FROM vehicle_trips
WHERE mo_id in (&aLAVIATrip, &aNormalTrip);
```

This query is similar to Q1. It selects two speed profiles corresponding to a normal and a LAVIA driving styles for a given route. The graphic in Figure VI.14 compares the selected speed profiles with the speed limit profile. The LAVIA speed profile is very close to the speed limit profile with small peaks around the speed limit due to the vehicle inertia. For the normal speed profile, there are significant exceeding areas for the 50km/h and 70km/h speed limits. These usually correspond to built-up areas. On the other hand, we observe that for speed limits of 110km/h and 130km/h, which correspond to highways, the normal speed profile is below the LAVIA profile. We do not hazard into any conclusions, since this is out of the

scope of this experimentation. These examples only try to show the usefulness of the proposed database system as a tool in analyzing this kind of data.

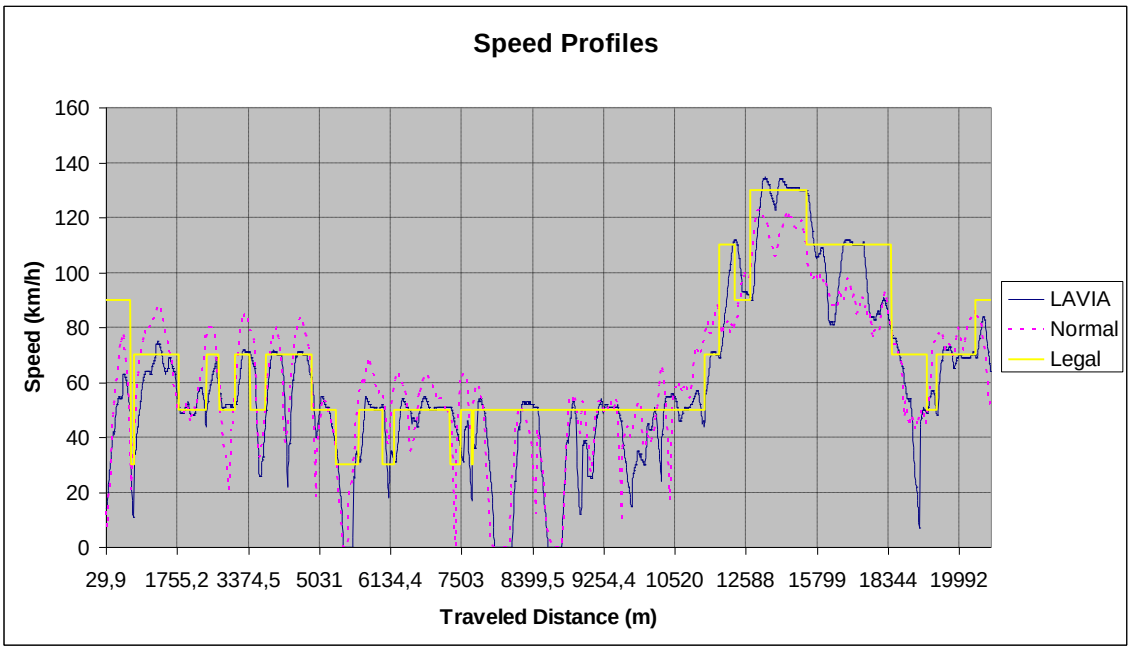


Figure VI.14: Comparison between the practiced speed and the legal speed for a LAVIA and a normal trip

```
Q3: SELECT no_transitions(t_brakeSwitch)/2
FROM vehicle_trips
WHERE mo_id = &aTrip;
```

Driver	Driving style			
	Economical	Normal	LAVIA	Nervous
A	32	55	44	73
B	40	43	33	84

Table VI.1: Number of brakes for a trip

```
Q4: SELECT avg(g_fuel)
FROM vehicle_trips
WHERE mo_id = &aTrip;
```

The queries Q3 and Q4 are examples of using aggregate functions on the measure profiles. The results for these queries are displayed in Table VI.1 and Table VI.2 respectively.

Driver	Driving style			
	Economical	Normal	LAVIA	Nervous
A	7.27	7.62	8.27	9.29
B	7.2	7.7	8.1	9.1

Table VI.2: Average fuel consumption for a trip (l/100km)

```
Q5: SELECT trajectory(g_speed),
                trajectory(greaterThan(g_speed, &legalSpeed)
FROM vehicle_trips
WHERE id = &aLAVIATrip;

Q6: SELECT trajectory(g_speed),
           trajectory(greaterThan(g_speed, &legalSpeed*1.05)
FROM vehicle_trips
WHERE id = &aLAVIATrip;
```

Queries Q5 and Q6 select the trajectory of a given LAVIA trip together with the regions of the trajectory where the practiced speed exceeds the local speed limit (or the speed limit augmented with 5% for Q6). The parts where the practiced speed exceeds the speed limit are obtained by using the restriction function **greaterThan**. The graphical results for the two queries are given in Figure VI.15 and Figure VI.16. The green color indicates the road network; the black color draws the selected vehicle trajectory, while the red one is showing the parts where the practiced speed is above the legal speed. By comparing the two pictures, we see that for a LAVIA trip, the vehicle speed is mostly close to the speed limit, exceeding it with a very small margin.

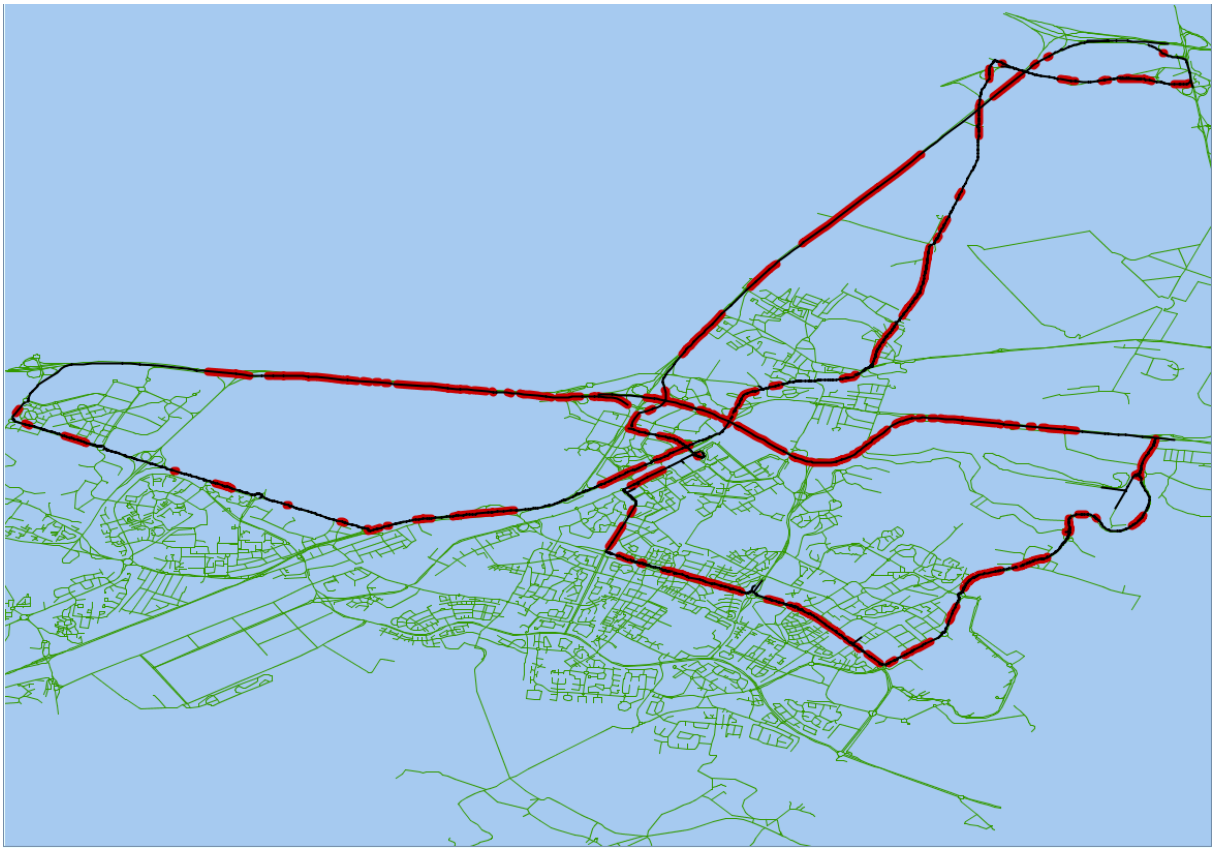


Figure VI.15: Comparison between the practiced speed and the legal speed for a LAVIA trip

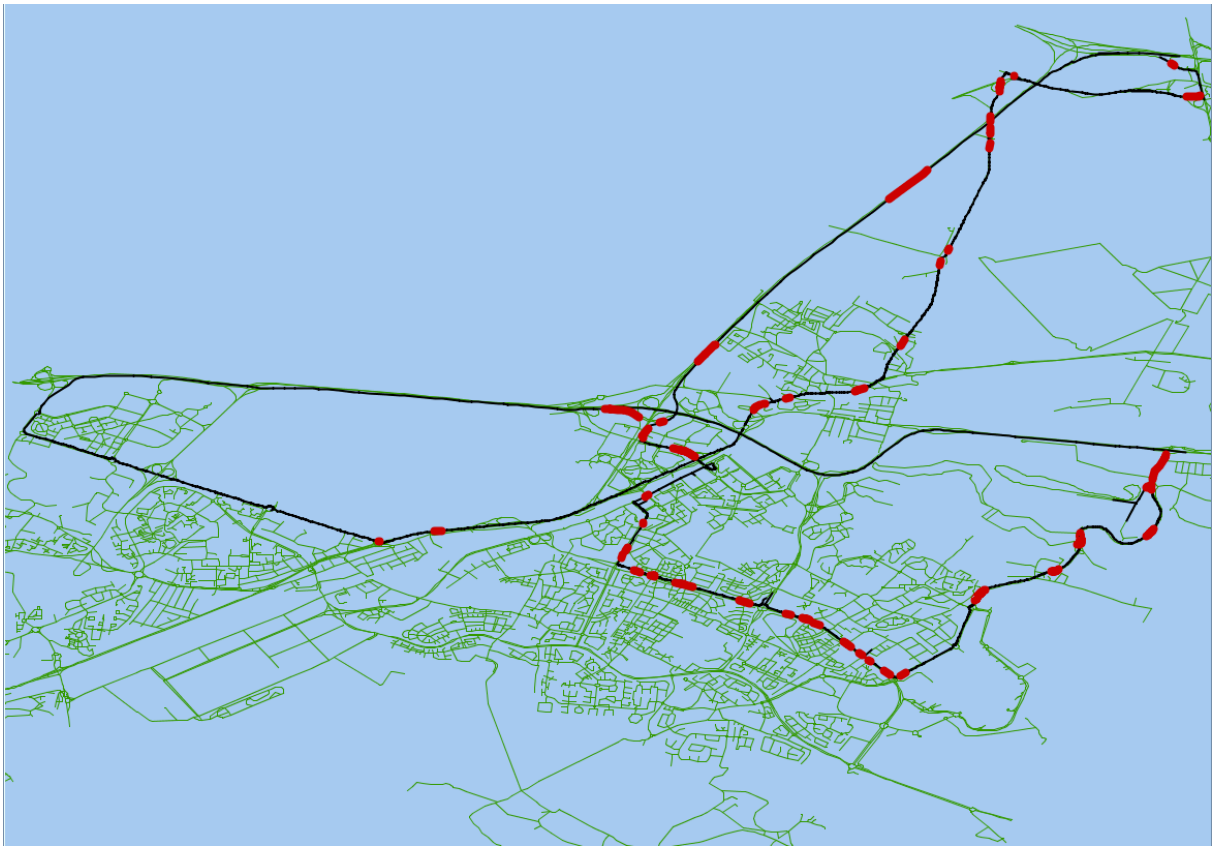


Figure VI.16: Comparison between the practiced speed and the legal speed for a LAVIA trip

```
Q7: SELECT duration(deftime(at(trip,
     trajectory(greaterThan(g_speed,&legalSpeed)))) /
          duration(deftime(trip))
FROM vehicle_trips
WHERE mo_id = &aTrip;


Q8: SELECT length(trajectory(greaterThan(g_speed,
&legalSpeed))) /
          length(trajectory(g_speed))
FROM vehicle_trips
WHERE id = &aLAVIATrip;
```

Query Q7 computes the percentage of time passed above the speed limit for a given trip. The function **deftime** is a projection that obtains the definition time interval(s) for the trip. Then, **duration** computes the length of the temporal interval(s). Query Q8 is similar to query Q7. The difference is that it computes the percentage of the trip length passed above the legal speed. The results are given in Table VI.3 and Table VI.4 respectively.

Driver	Driving style			
	Economical	Normal	LAVIA	Nervous
A	19.3	33.7	29.4	52.1
B	24.4	25.8	21.5	46.7

Table VI.3: Percentage (%) of trip time passed over the legal speed

Driver	Driving style LAVIA		
	Legal speed	1.05*Legal speed	1.1*Legal speed
A	40.91	8.97	5.63
B	23.01	9.67	6.56

Table VI.4: Percentage (%) of trip length passed over the legal speed

VI.3.3.2 Computing Indicators Linked to Fuel Consumption

Beside the above query examples, there are many studies in the naturalistic driving area that are based on the analysis of this kind of data. For example, assessing the impact of driving behavior on fuel consumption is an important topic today. Some classic indicators such as the average and the variance of speed, acceleration, deceleration, or engine RMP, and certainly the average fuel consumption, represent main statistical values for this kind of study. Other not so obvious indicators can be devised in this context. Here are some examples formulated as queries and the corresponding results for the test data set:

Q1: *At what average RPM value the gear shifting is done for a given trip?*

```
SELECT mo_id, avg(val(atpos(v.g_RPM,
                            GPOINT(0, t.RID, t.POS1,
t.SIDE))))
FROM LAVIA_ITS_ADT v,
     THE (SELECT cast(
          locations(atTransitions(g_gear)).rints as
SECTIONS_V)
     FROM LAVIA_ITS_ADT WHERE mo_id = v.mo_id) t
GROUP BY mo_id
ORDER BY mo_id;
```

Driver	Driving style			
	Economical	Normal	LAVIA	Nervous
A	1532.45	1641.68	1981.84	2698.55
B	1767.37	1780.25	1853.94	2691.49

Table VI.5: Average RPM gear change

Q2: *What percentage of the trip duration is spent in a non-optimal gear (i.e., RPM is above 3500)?*

```
SELECT mo_id, duration(deftime(greaterThan(t_RPM, 3500)))
                                        / duration(deftime(t_RPM))
FROM LAVIA_ITS_ADT;
```

Driver	Driving style

	Economical	Normal	LAVIA	Nervous
A	0	0.023	0.006	0.298
B	0	0	0	0.242

Table VI.6: % of time in non-optimal gear

Q3: *What percentage of the trip length (or of the trip duration) is spent in engine brake (i.e., gear engaged and gas pedal not engaged)?*

```
SELECT mo_id, length(intersection(
                trajectory(greaterThan(toGReal(g_gear), 0))),
                trajectory(equalTo(g_gasPedal, 0)) ) )
        / length(trajectory(g_gasPedal))
FROM LAVIA_ITS_ADT;


SELECT mo_id, duration(intersection(
                    deftime(greaterThan(toMReal(t_gear), 0)),
                    deftime(equalTo(t_gasPedal, 0)) ) )
        / duration(deftime(t_gasPedal))
FROM LAVIA_ITS_ADT;
```

Driver	Driving style			
	Economical	Normal	LAVIA	Nervous
A	0.139	0.148	0.155	0.217
B	0.163	0.113	0.146	0.207

Table VI.7: % of space in engine brake

Driver	Driving style			
	Economical	Normal	LAVIA	Nervous
A	0.222	0.243	0.251	0.313
B	0.276	0.230	0.251	0.328

Table VI.8: % of time in engine brake

Q4: *What is the total number of stops and their average duration for a given trip?*

```
SELECT mo_id,
            no_components(deftime(equalTo(t_speed, 0))),
                        mean(deftime(equalTo(t_speed, 0)))
FROM LAVIA_ITS_ADT;
```

Driver	Driving style			
	Economical	Normal	LAVIA	Nervous
A	8	11	12	15
B	9	12	11	13

Table VI.9: Number of stops for a trip

Driver	Driving style			
	Economical	Normal	LAVIA	Nervous
A	3.416	4.012	4.211	3.435
B	6.431	5.724	3.995	6.282

Table VI.10: Average stop time (seconds)

Q5: *Compute the Relative Positive Acceleration (RPA), i.e.,* $RPA = \frac{1}{T}\int va^{+}dt$ *, for a given trip; where T is the trip duration, v is the speed, and a+ is the positive acceleration.*

```
SELECT mo_id, mean(mul(t_speed,
greaterThan(t_acceleration,0)))
FROM LAVIA_ITS_ADT;
```

Driver	Driving style			
	Economical	Normal	LAVIA	Nervous
A	4.160	6.702	7.171	13.01
B	4.248	4.620	5.263	12.23

Table VI.11: RPA for a trip (meter^2/second^3)

Q6: *Compute the Positive Kinetic Energy (PKE), i.e.,* $PKE = \frac{\sum(v_f^2 - v_s^2)}{L}$ *when* $a > 0$*, for a given trip; where L is the trip length,* v_f *and* v_f *are the final speed and start speed in the intervals where the acceleration a is positive.*

```
DECLARE
sumV NUMBER;
dist NUMBER;
cnumber NUMBER;
CURSOR crs IS
    SELECT DISTINCT mo_id FROM LAVIA_ITS_ADT ORDER BY mo_id;
BEGIN
  OPEN crs;
  LOOP FETCH crs INTO cnumber;
    EXIT  WHEN crs % NOTFOUND;
    dbms_output.put_line(cnumber);
    sumV := 0;
    FOR r_c IN (SELECT val(atInstant(t_speed, t.E)) AS Vf,
                              val(atInstant(t_speed, t.S)) AS
Vs
                FROM LAVIA_ITS_ADT v,
                  THE (SELECT cast(deftime(greaterThan
                     (t_acceleration,0)).intvs as INTERVALS_V)
                  FROM LAVIA_ITS_ADT WHERE mo_id = v.mo_id) t
                  WHERE mo_id = cnumber)
      LOOP
```

```
        sumV := sumV + r_c.Vf*r_c.Vf - r_c.Vs*r_c.Vs;
      END LOOP;
    SELECT length(trajectory(trip))
    INTO dist
    FROM LAVIA_ITS_ADT WHERE mo_id = cnumber;
    dbms_output.put_line('PKE = ' || ROUND(sumV/dist, 3));
  END LOOP;
  CLOSE crs;
END;
```

Some complex queries such as Q6 may require the use of PL/SQL. However, this is not expected to be a frequent case.

Driver	Driving style			
	Economical	Normal	LAVIA	Nervous
A	0.202	0.294	0.343	0.595
B	0.223	0.224	0.275	0.557

Table VI.12: PKE for a trip (meter/second^2)

VI.4 Application Perspectives

Embedding sensors in moving object has opened the way for many new applications in domains such as transportation, traffic management, and environmental studies. Most of these applications need to analyze enormous amounts of spatio-temporal data. Naturalistic driving experiments are emblematic for this application domain. Recent projects such as LAVIA pointed out the necessity of a database system that is capable of managing such data. CALM prototype is the materialization of our attempt to respond to this demand. In this chapter we presented the current development state of the prototype by using a real data set and several query scenarios. A lot of work is still needed in order to have a fully functional system. However, the results obtained so far confirm the validity of our approach and are promising for a scenario on a larger scale.

For application perspectives we return once more to the naturalistic driving domain. Among the main topics that can be investigated in the naturalistic driving approach, it should be mentioned: behavior pattern classification (e.g. quiet vs. aggressive, quiet vs. sporty driving), behavior stability over the time, relationship between driver's behavior and infrastructure characteristics, pre-crash study and driver countermeasures for crash avoidance, impact of ADAS on safety in terms of reduction of fatalities or serious accident, impact of driving behavior on fuel consumption, etc. More surprisingly is the fact that naturalistic driving opens the way towards a dual analysis, based on detection of widespread abnormal driving behavior allowing then, infrastructure diagnosis and black spot detection. A few applications motivating the future development of the proposed STDBMS system are presented next.

i) Legal speed analysis: Most of the transportation studies are involving speed profiles. Usually, these profiles are considered as time series without any spatial considerations. It is interesting for the user to deal with speed profiles as functions of both continuous time and space. It would be of great interest to obtain all the speed profiles for the trips passing through

two given locations. Moreover, assuming available a speed limit database, one may want to compare those legal speeds to the speed profiles of all the drivers taking this route by using the following query:

"Given the speed limit database on a specific area, retrieve all the places where instantaneous speed is 30km/h above the speed limit for a given percentage of the passing vehicles."

ii) Fixed speed enforcement camera impact assessment: During the past few years, the French road network has been covered with safety cameras. Impact of such measures on risk reduction is great, according to the observed drastic reduction of death statistics (number per year reduced by at least 4000 in five years), but the potential impact on the driver's behavior is unfamiliar. It is of great interest to study the route choice made by the drivers before/after the installation of a new speed control system. Here is an example of an interesting query:

"Given an origin/destination, retrieve all the route choice for all the trips passing through those two locations, before/after the camera installation."

iii) Relation between infrastructure and speed: The main problem in order to evaluate the impact of an infrastructure on the speed is to find enough valid cases on a given road section. However, it is hard to achieve a good representation of the population in order to do robust statistics on the topic. Another solution is to find similar infrastructures on different sites and to gather speed data on every site. Moreover, we can even find small variation between sites, which permits to evaluate the impact of the variation of one parameter. The curve, for example, is a well studied road object. We can collect speed data on equivalent curves (for instance with a curvature between 450 and 500 meters, on a rural road). Then, the data can be filtered, on a route basis, in order to eliminate constrained vehicles. Finally, by using the resulted population of trips, we can evaluate the impact of variation of parameters between sites. One example of query could be:

"Find all speed profiles of non constrained vehicle crossing a curve having a radius between 450 and 500 meters."

iv) Driver behavior analysis: Many transportation research projects focus on understanding precisely the driver behavior to assess the impact of advanced driving assistance systems (ADAS). Such an objective can be reached by studying various statistical indicators (mean number of gear-shift per km, percentage of time over the speed limit, average speed or speed profile, average consumption, etc.) for all the trips of studied drivers. This is made easily tractable using the proposed DBMS.

For example, if a specific road needs to be studied in terms of risk, it is relevant to use the 85^{th} percentile of the speed. The 85^{th} percentile is the speed at or below which 85% of all vehicles are observed to travel past a nominated point. This is the speed that "reasonable" people tend to adopt according to the road environment. The 85^{th} percentile is called the "operating speed" and is not related to the roads posted limit (all road encourage us to travel at a certain speed, and every road is different).

"Given a route, retrieve the "operating speed" profile, after/before the installation of a safety camera."

VII. Conclusion

VII.1 Summary of the Contributions

New technologies such as GPS, sensors and ubiquitous computing are pervading our society. The movement of people and vehicles may be sensed and recorded, thus producing large volumes of mobility data. Environmental monitoring, transportation and distribution, emergency services, and telecommunications all have challenging problems in representing and querying databases describing moving objects.

This thesis addresses the problem of managing mobile location sensor data. A moving object – such as a vehicle – may report some measures related to its state or to its environment, which are sensed throughout his movement. Managing such data is of major interest for some application such as analyzing the driving behavior or reconstructing the circumstances of an accident in road safety, or identifying, by means of a vibration sensor, the defects along a railway in maintenance. Moreover, since objects, in many cases, do not move freely in the two-dimensional space but rather within spatially embedded networks (roads, highways, railroads, even airlines have fixed routes), one should include spatial networks into the data model.

The state-of-the-art database management systems fail to handle such complex data and their processing. However, more than a decade of research in the area of spatio-temporal databases has produced numerous solutions for modeling and indexing data related to moving objects. In this thesis we analyzed the limitations of existing work in modeling, querying and indexing moving objects with sensors on road networks. Then, we proposed new solutions to deal with these limitations. The main contributions of the thesis are a data model and a query language for moving sensor data, and an access method for in-network trajectories. We also implemented these proposals as a spatio-temporal DBMS extension and evaluated them. These contributions are summarized next:

- *Design of a data model and a query language for moving sensor data*. A first contribution of the thesis is a new data model and a language that handle mobile location sensor data. This model is devised as an extension of the state-of-the-art moving object data model proposed by Güting et al.. To this end, we introduced the concept of measure profile to capture the measure variability in space, along with specific operations that permit to analyze the data. We also described their implementation using the object-relational paradigm.

- *Design of an access method for in-network trajectories*. A second contribution is represented by a new access method (i.e., PARINET) to efficiently retrieve the trajectories of objects moving in networks. PARINET can easily be integrated in any RDBMS, which is an essential asset particularly for industrial or commercial applications. Unlike the existing approaches, PARINET relies on the expected query size and on the distribution of the data to be indexed, which can be known in advance given the fact that we deal with historical data. Thus, it is easily tunable and adaptable to any spatial distribution of moving object trajectories. The experimental evaluation under an off-the-shelf DBMS shows that our approach significantly outperforms the reference R-tree based indexes. Also, PARINET showed robust performance with the

query size and data size, when tuned correctly. PARINET uses a cost model in order to adapt to a data distribution and a query size. The experimental results show that the cost model offers good results, suggesting near-optimal index configuration.

- *Development of a spatio-temporal DBMS extension*. Based on the above proposals, we developed a spatio-temporal DBMS prototype called CALM. CALM is implemented as an Oracle DBMS extension. We created new data types, new operations and new index types by using the extensibility capabilities of Oracle 11g data server. Complementary to the spatio-temporal DBMS extension, we implemented a graphical user interface to handle the presentation of data types in CALM. To this end, we used GeoServer coupled with GoogleMaps.

VII.2 Outlooks

When we started working at this thesis, our principal (challenging) objective was to design and to develop a spatio-temporal database system that is capable of managing mobile location sensor data. We summarized in the previous section the contributions that we have managed to accomplish. Clearly, a lot of work is still needed to achieve this ambitious goal. Moreover, as indicated in Chapter VI, the data-chain processing in the case of mobile sensor data does not begin nor end with the spatio-temporal DBMS. Significant data processing is needed before (i.e., in the upstream) integrating the raw collected data into the STDBMS, and after (i.e., in the downstream) querying the integrated data. The upstream and downstream data processing also raise interesting challenges that may need to be studied. In the rest of this section, we indicate some perspective work for the STDBMS prototype, and for the upstream and downstream data processing.

VII.2.1 Outlooks for CALM

In Chapter VI, we presented an evaluation of the prototype CALM. The experimentation was based on a real data set. A significant shortcoming for the evaluation was the fact that we only had access to a small data set. However, we are certain that we will be able to re-conduct a similar evaluation on a much larger data set (at least a few thousand trips) in the near future. This is important in order to show the system's scalability with the data set.

In this context, we need to implement index types for all the spatial, temporal and spatio-temporal operators listed in Chapter III (see Table III.3). We will also test PARINET as an access method for *on-value spatial* operators. Most likely, we will need to adapt and refine PARINET to take into account the specificity of the data types. Since time is isomorphic to the real numbers, there is a certain similarity between the temporal dimension and the "on-value" dimension. However, the similarity between time and measures is very limited. For instance, the time monotonically grows along a certain trip from the start point to the end point, whereas the vehicle's speed oscillates along the trip being influenced by many factors such as the speed limit and the traffic congestion.

The current PARINET index maintenance as proposed in Chapter V is a straightforward approach. As future work, we intend to improve the method for an optimal handling of the index evolution in time. Moreover, we believe that PARINET can be extended for indexing the current and near-future movements of in-network moving objects. This would provide an access method for capturing the in-network positions of moving objects at all points in time (i.e., past, present and anticipated future).

In Chapter III (Section III.5.3), we proposed a method for optimizing the aggregate functions. A large data set will permit evaluating the proposed approach. Thus, we could experimentally estimate the benefits of such an approach in terms of efficiency, as well as the margins of error introduced by the approximations.

Our proposed data model includes a collection of operations to handle the mobile sensor data (see Table III.2). As indicated in Chapter III, this collection of operations forms a basis of functionality. Other operations may be added to meet the needs of some applications. For example, the analytical functions, such as **percentile**, are useful for downstream data processing (e.g., data mining algorithms). Optimizing such kind of operations may represent a challenging task.

CALM is implemented as a DBMS extension of Oracle 11g data server. We are currently testing the system's portability to SECONDO. Beside being an open source DBMS, SECONDO is an environment particularly geared for extension by algebra modules for non-standard applications. Compared to commercial DBMSs, SECONDO offers the advantage of being complete, i.e., all aspects needed by such DBMS extensions are addressed. For instance, the extension of the optimizer is limited to operators and their indexes in Oracle, whereas there are no such limitations in SECONDO. Another advantage is given by the fact that several algebras for moving objects are already available in SECONDO. Therefore, we think that it will be interesting to compare two such implementations in terms of performance and extendibility capabilities. Moreover, this public platform will allow more users to work with CALM and contribute to its development.

VII.2.2 Outlooks for Upstream Data Processing

Data acquisition in naturalistic driving like applications is done at high frequencies. For example, for the data set that we used to evaluate CALM, the sensor data has been recorded at a frequency of 250 milliseconds (i.e., four recordings each second). However, DIRCO is capable of an acquisition rate of up to 10 ms (i.e., 100 recordings each second). Given that nearly two dozens sensors are simultaneously monitored in each vehicle, we can easily understand that this type of application will generate enormous volumes of data. This leads to storage, transmission and computation challenges.

In the experimental evaluation presented in Chapter VI, we used a linear interpolation between two consecutive records in order to transform the raw data to the corresponding data types in the STDBMS. This simple approach can not be used for large data sets. Hence, the need for *compression* techniques arises. There are multiple objectives for the data compression [113]:

- Obtain a lasting reduction in data size.
- Use interpolation functions that allow various computations at acceptable complexity (e.g., constant, linear, second degree polynomials).
- Obtain compressed data with known, small margins of error, which are preferably parametrically adjustable.

Therefore, our interest is with lossy compression techniques, which eliminate some redundant or unnecessary information under well-defined error bounds. The algorithms evaluated in [113] appear to be a good starting point for the work in this direction.

VII.2.3 Outlooks for Downstream Data Processing

Analyzing mobile sensor data by means of queries (e.g., which perform filtering and calculations guided by the user) is feasible within the STDBMS. Nevertheless, an exploratory or predictive global analysis of mobility traces remains limited within the STDBMS. Examples of queries in this case are: How to compare the adaptation of road infrastructure to its current use? How to organize car-pooling knowing a set of commuting? How to predict the traffic to improve navigation and trip planning?

This type of analysis requires appropriate data mining tools that can be built entirely on top of the STDBMS. The data mining algorithms typically use aggregate data and are costly. A direct analysis of trajectories of moving objects, which are characterized by their large volumes of data, is not always feasible. Therefore, a higher level representation of the data (e.g., a symbolic representation) is necessary for global analyses. Hence, the same data can have several representations in the STDBMS corresponding to different types of analysis. We had already started to work in this direction (i.e., in-network trajectory clustering [36]) and we plan to continue in the future.

References

1. Tansel, A.U., Clifford, J., Gadia, S., Jajodia, S., Segev, A., and Snodgrass, R.: Temporal Databases: Theory, Design and Implementation, Benjamin/Cummings Publishing Company, (1993)

2. Guting, R.H.: An Introduction to Spatial Database Systems, VLDB Journal 4 (1994), 357-399.

3. Güting R.H., and Schneider, M.: Moving Objects Databases, Morgan Kaufmann, (2005)

4. Güting, R.H., Bohlen, M.H., Erwig, M., Jensen, C.S., Lorentzos, N.A., Schneider,M., Vazirgiannis, M.: A foundation for representing and querying moving objects. ACM Transactions on Database Systems 25(1), 1–42 (2000)

5. Güting R.H., Almeida, V.T., and Ding Z.: Modeling and Querying Moving Objects in Networks. VLDB Journal 15(2): 165-190, (2006)

6. Cattel, R.G.G., and Barry D.K.: (eds.), *"The Object Database Standard: ODMG 2.0"*, Morgan Kaufman Publishers, May 1997.

7. ISO 19107:2003, Geographic Information – Spatial Schema, WG 2

8. Egenhofer, M.J. and Herring, J.R: Categorizing topological spatial relations between point, line, and area objects, in Egenhofer, M.J., Mark, D.M. and Herring, J.R. (Eds.): The 9-Intersection: Formalism and its Use For Natural-Language Spatial Predicates, National Center for Geographic Information and Analysis, Report 94-1, Santa Barbara, CA, (1994)

9. ORACLE Database Data Cartridge Developer's Guide, 10g Release 2 (10.2), (2007)

10. ORACLE Database – Oracle Spatial Developer's Guide, 11g Release 1 (11.1), (2008)

11. Kothuri, R., Godfrind, A., and Beinat, E.: Pro Oracle Spatial, Apress, (2004)

12. Ozsoyoglu, G., and Snodgrass, R. T.: Temporal and Real-Time Databases: A survey. (1995)

13. Pelekis, N.: STAU: A Spatio-Temporal Extension to ORACLE DBMS, PhD Thesis UMIST, Department of Computation, (2002)

14. Pelekis, N., Theodoulidis, B., Kopanakis, I., Theodoridis, Y.: Literature Review of Spatio-Temporal Database Models, The Knowledge Engineering Review Journal, 19(3), 235-274, June 2005

15. Yeh, T.S., and B. de Cambray: Modeling Highly Variable Spatio-Temporal Data. 6th Australasian Database Conference (ADC'1995), 221-230, (1995)

16. Sistla, A.P., Wolfson, O., Chamberlain, S., Dao, S.: Modeling and querying moving objects. In: Proceedings of the 13th International Conference on Data Engineering (ICDE), pp. 422–432 (1997)

17. Wolfson, O., Chamberlain, S., Dao, S., Jiang, L.,Mendez, G.: Cost and imprecision in modeling the position of moving objects. In: Proceedings of the 14th International Conference on Data Engineering (ICDE), pp. 588–596 (1998)

18. Koubarakis, M., Pernici, B., Schek, H.J., Scholl, M., Theodoulidis, B., Tryfona, N., Sellis, T., Frank, A.U., Grumbach, S., Güting, R.H., Jensen, C.S., Lorentzos, N., Manolopoulos, Y., Nardelli, E. (eds.): Spatio-temporal databases: The CHOROCHRONOS approach. Springer-Verlag, Lecture Notes in Computer Science 2520 (2003)

19. Forlizzi, L., Güting, R.H., Nardelli, E., Schneider, M.: A data model and data structures for moving objects databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 319–330 (2000)

20. Rigaux, P., Scholl, M., Segoufin, L., Grumbach, S.: Building a constraint-based spatial database system: Model, languages, and implementation. Information Systems 28(6), 563–595 (2003)

21. Grumbach, S., Rigaux, P., Segoufin, L.: The DEDALE system for complex spatial queries. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 213–224 (1998)

22. Cotelo Lema, J.A., Forlizzi, L., Güting, R.H., Nardelli, E., Schneider, M.: Algorithms for moving objects databases. The Computer Journal 46(6), 680–712 (2003)

23. Booth, J., Sistla, A.P., Wolfson, O., Cruz, I.F.: A data model for trip planning in multimodal transportation systems. EDBT 2009: 994-1005

24. Speicys, L., and Jensen, C.S.: Enabling Location-based Services - Multi-Graph Representation of Transportation Networks. GeoInformatica 12(2): 219-253 (2008)

25. Güting, R.H., Schneider, M.: Realm-Based Spatial Data Types: The ROSE Algebra VLDB J. 4(2): 243-286 (1995)

26. Pelekis, N., Frentzos, E., Giatrakos, N., Theodoridis, Y.: HERMES: Aggregative LBS via a Trajectory DB Engine, SIGMOD 2008: 1255-1258

27. Pelekis, N., Theodoridis, Y.: Boosting Location-Based Services with a Moving Object Database Engine, Proceedings of MobiDE, (2006)

28. de Almeida, V.T., Güting, R.H., Behr, T.: Querying Moving Objects in SECONDO. MDM 2006: 47

29. Sakr, M.A., Güting, R.H.: Spatiotemporal Pattern Queries in SECONDO. SSTD 2009: 422-426

30. Güting, R.H., Braese, A., Behr, T., Xu, J: Nearest Neighbor Search on Moving Object Trajectories in SECONDO. SSTD 2009: 427-431

31. SECONDO - An Extensible Database System. Available at: http://dna.fernuni-hagen.de/Secondo.html/

32. NHTSA (2006). The 100-Car Naturalistic Driving Study, Phase II – Results of the 100-Car Field Experiment. Report No. DOT HS 810 593

33. Servigne, S., Devogele, T., Bouju, A., Bertrand, F., Gutierrez, C., Laucius, S., Noel, G., Ray, C.: Gestion de masses de données temps réel au sein de bases de données capteurs, Conférence Spatial Analysis and GEOmatics, SAGEO 2007, Clermont-Ferrand, France, Juin 2007

34. Burns, P.C.: International Harmonized Research Activities – Intelligent Transport Systems (IHRA-ITS) working group report. 19th International Technical Conference on the Enhanced Safety of Vehicle (ESV), 2005

35. Ehrlich, J., Marchi, M., Jarri, P, Salesse, L., Guichon, D., Dominois, D, Leverger, C.: LAVIA, the French ISA project: Main issues and first results on technical tests, 10th World Congress & Exhibition on ITS, November 2003

36. Kharrat, A., Sandu Popa, I., Zeitouni, K., Faiz, S.: Clustering algorithm for network constraint trajectories. The 13th International Symposium on Spatial Data Handling, SDH 2008

37. Sandu Popa, I., Zeitouni, K., Dupin, F., Saint-Pierre, G., Glaser, S.: Using in-Vehicle Sensor Data for Naturalistic Driving Analysis, 15th World Congress on Intelligent Transportation Systems (ITS'2008), 2008

38. Gaede, V. and Günther, O.: Multidimensional Access Methods. ACM Comput. Surv. 30(2): 170-231 (1998)

39. Bayer, R. and McCreight, E. M.: Organization and maintenance of large ordered indices. Acta Informatica 1(3), 173-189, (1972)

40. Fagin, R., Nievergelt, J., Pippenger, N. and Strong, R.: Extendible hashing: A fast access method for dynamic Files. ACM Trans. Database Systems 4 (3),315-344, (1979)

41. Kriegel, H.-P.: Performance comparison of index structures for multikey retrieval. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 186-196, (1984)

42. Brinkhoff, T., Kriegel, H.-P., Schneider, R., and Seeger, B.: Multi-step processing of spatial joins. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 197-208, (1994)

43. Hoel, E. G. and Samet, H.: Benchmarking spatial join operations with spatial output. In Proc. 21st Int. Conf. on Very Large Data Bases, pp. 606-618, (1995)

44. Litwin, W.: Linear hashing: A new tool for file and table addressing. In Proc. 6th Int. Conf. on Very Large Data Bases, pp. 212-223, (1980)

45. Larson, P. A.: Linear hashing with partial expansions. In Proc. 6th Int. Conf. on Very Large Data Bases, pp. 224-232, (1980)

46. Comer, D.: The ubiquitous B-tree. ACM Computing Surveys 11 (2), 121-138, (1979)

47. Bentley, J. L.: Multidimensional binary search trees used for associative searching. Communications of the ACM 18 (9), 509-517, (1975)

48. Bentley, J. L.: Multidimensional binary search in database applications. IEEE Trans. Software Eng. 4 (5), 333-340, (1979)

49. Fuchs, H., Kedem, Z., and Naylor, B.: On visible surface generation by a priori tree structures. Computer Graphics 14 (3), (1980)

50. Fuchs, H., Abram, G. D., and Grant, E. D.: Near real{time shaded display of rigid objects. Computer Graphics 17 (3), 65-72, (1983)

51. Samet, H.: The quadtree and related hierarchical data structure. ACM Computing Surveys 16 (2), 187-260, (1984)

52. Samet, H.: The design and analysis of spatial data structures. Reading, MA:Addison-Wesley, (1989)

53. Morton, G.: A computer oriented geodetic data base and a new technique in file sequencing. IBM Ltd., (1966)

54. Orenstein, J. and Merrett, T. H.: A class of data structures for associative searching. In Proc. 3rd ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, pp. 181-190, (1984)

55. Faloutsos, C. and Roseman, S.: Fractals for secondary key retrieval. In Proc. 8th ACM SIGACT-SIGMOD-SIGARTSymp. on Principles of Database Systems, pp. 247-252, (1989)

56. Faloutsos, C.: Multiattribute hashing using Gray-codes. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 227-238, (1986)

57. Abel, D. J. and Mark, D. M.: A comparative analysis of some two-dimensional orderings. Int. J. Geographical Information Systems 4 (1), 21-31, (1990)

58. Jagadish, H. V.: Linear clustering of objects with multiple attributes. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 332-342, (1990)

59. Faloutsos, C. and Rong, Y.: DOT: A spatial access method using fractals. In Proc. 7th IEEE Int. Conf. on Data Eng., pp. 152-159, (1991)

60. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 47-54, (1984)

61. Sellis, T., Roussopoulos, N., and Faloutsos, C.: The R+-tree: A dynamic index for multi-dimensional objects. In Proc. 13th Int. Conf. on Very Large Data Bases, pp. 507-518, (1987)

62. Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B.: The R^*-tree: An efficient and robust access method for points and rectangles. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 322-331, (1990)

63. Nascimento, M. A., and Dunham, M. H.: Indexing Valid Time Databases via B^+-Trees. IEEE Trans. Knowl. Data Eng. 11(6): 929-947, (1999)

64. Elmasri, R., Wuu, G.T.J., and Kouramajian, V.: The Time Index and the Monotonic B^+-tree. In A. Tansel et al., editors, Temporal Databases: Theory, Design and Implementation, chapter 18, pages 433–456. Benjamin/Cummings, Redwood City, CA, 1993.

65. Kouramajian V. et al.: The Time $Index^+$: An incremental access structure for temporal databases. In Proceedings of Third International Conference on Knowledge and Management (CIKM'94), pages 296–303, Gaithersburg, MD, November 1994

66. Shen, H., Ooi, B.C., and Lu, H.: The TP-Index: A dynamic and efficient indexing mechanism for temporal databases. In Proceedings of the 10th IEEE International Conference on Data Engineering, pages 274–281, Houston, TX, February 1994

67. Ang, C-H., and Tan, K-P.: The interval B-tree. Information Processing Letters, 53(2):85–89, January 1995

68. Bozkaya T., and Özsoyoglu Z. M.: Indexing Valid Time Intervals. Proc. 9th Int. Conf. on Database and Expert Systems Applications, LNCS 1460, 541-550, 1998

69. Nascimento, M. A., and Dunham, M. H.: Indexing Valid Time Databases via B^+-Trees. IEEE Trans. on Knowledge and Data Engineering 11(6): 929-947, 1999

70. Kriegel, H-P., Pötke, M., Seidl, T.: Managing Intervals Efficiently in Object-Relational Databases. VLDB 2000: 407-418

71. Edelsbrunner H.: Dynamic Rectangle Intersection Searching. Institute for Information Processing Report 47, Technical University of Graz, Austria, 1980

72. Goh C. H., Lu H., Ooi B. C., Tan K.-L.: Indexing Temporal Data Using Existing B^+-Trees. Data & Knowledge Engineering, Elsevier, 18(2): 147-165, 1996

73. Theodoridis, Y., Sellis, T. K., Papadopoulos, A., Manolopoulos, Y.: Specifications for Efficient Indexing in Spatiotemporal Databases. SSDBM 1998: 123-132

74. Mokbel, M. F., Aref, W. G.: Location-Aware Query Processing and Optimization. MDM 2007: 229

75. Xu, X., Han, J., and Lu, W.: RT-Tree: An Improved R-Tree Indexing Structure for Temporal Spatial Databases. In Proc. of the Intl. Symp. on Spatial Data Handling, SDH, pages 1040–1049, July 1990

76. Theodoridis, Y., Vazirgiannis, M., and Sellis, T.: Spatio-Temporal Indexing for Large Multimedia Applications. In Proc. of the IEEE Conference on Multimedia Computing and Systems, ICMCS, June 1996

77. Porkaew, K., Lazaridis, I., and Mehrotra, S.: Querying Mobile Objects in Spatio-Temporal Databases. In Proc. of the Intl. Symp. on Advances in Spatial and Temporal Databases, SSTD, pages 59–78, Redondo Beach, CA, July 2001

78. Nascimento, M. A., and Silva, J. R. O.: Towards historical R-trees. In Proc. of the ACM Symp. on Applied Computing, SAC, pages 235–240, Feb. 1998

79. Tao, Y. and Papadias, D.: Efficient Historical R-trees. In Proc. of the Intl. Conf. on Scientific and Statistical Database Management, SSDBM, pages 223–232, July 2001

80. Tao, Y. and Papadias, D.: MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In Proc. of the Intl. Conf. on Very Large Data Bases, VLDB, pages 431–440, Sept. 2001

81. Kollios, G., Tsotras, V. J., Gunopulos, D., Delis, A., and Hadjieleftheriou, M.: Indexing Animated Objects Using Spatiotemporal Access Methods. IEEE Trans. on Knowledge and Data Engineering, TKDE, 13(5):758–777, 2001

82. Pfoser, D., Jensen, C. S., and Theodoridis, Y.: Novel Approaches in Query Processing for Moving Object Trajectories. In Proc. of the Intl. Conf. on Very Large Data Bases, VLDB, pages 395–406, Sept. 2000

83. Chakka, V. P., Everspaugh, A., and Patel, J. M.: Indexing Large Trajectory Data Sets with SETI. In Proc. of the Conf. on Innovative Data Systems Research, CIDR, Asilomar, CA, Jan. 2003

84. Song, Z. and Roussopoulos, N.: SEB-tree: An Approach to Index Continuously Moving Objects. In Mobile Data Management, MDM, pages 340–344, Jan. 2003

85. Botea, V., Mallett, D., Nascimento, M. A., Sander, J.: PIST: An Efficient and Practical Indexing Technique for Historical Spatio-Temporal Point Data. GeoInformatica 12(2): 143-168 (2008)

86. Saltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A.: Indexing the positions of continuously moving objects. In: Proceedings of the 21st ACM SIGMOD International Conference on Management of Data. (2000) 331–342

87. Tao, Y., Papadias, D., Sun, J.: The TPR*-tree: an optimized spatio-temporal access method for predictive queries. In: Proceedings of the 30th International Conference on Very Large Data Bases. (2003) 790–801

88. Procopiuc, C.M., Agarwal, P.K., Har-Peled, S.: STAR-tree: an efficient self-adjusting index for moving objects. In: Revised Papers from the 4th InternationalWorkshop on Algorithm Engineering and Experiments. (2002) 178–193

89. Saltenis, S., Jensen, C.S.: Indexing of Moving Objects for Location-Based Services. In: Proceedings of the 18th International Conference on Data Engineering. (2002) 463–472

90. Patel, J.M., Arbor, A., Chen, Y., Chakka, V.P.: STRIPES: an efficient index for predicted trajectories. In: Proceedings of the 23rd ACM SIGMOD International Conference on Management of Data. (2004) 635–646

91. Jensen, C.S., Lin, D., Ooi, B.C.: Query and update efficient B^+-tree based indexing of moving objects. In: Proceedings of the 30th International Conference on Very Large Data Bases. (2004) 768–779

92. Chen, S., Ooi, B. C., Tan, K.-L., Nascimento, M. A.: ST^2B-tree: a self-tunable spatio-temporal B^+-tree index for moving objects. SIGMOD Conference 2008: 29-42

93. Theodoridis, Y., Silva, J. R. O., and Nascimento, M. A.: On the generation of spatiotemporal datasets. In Proceedings of the Sixth International Symposium on Spatial Databases (SSD). (1999)

94. Tzouramanis, T., Vassilakopoulos, M., and Manolopoulos, Y.: On the generation of time-evolving regional data. GeoInformatica 6:207-231. (2002)

95. Brinkhoff, T.: A framework for generating network-based moving objects. GeoInformatica 6(2): 153-180. (2002)

96. GSTD: Benchmarking spatio-temporal databases: the GSTD tool. Available at: http://db.cs.ualberta.ca:8080/gstd/index.html

97. G-TERD: Generator for time-evolving regional data. Available at: http://delab.csd.auth.gr/stdbs/g-terd.html

98. Brinkhoff, T.: Network-based generator for moving objects. Available at: http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/

99. Pfoser, D.: Where can I get spatio-temporal data? Available at: http://dke.cti.gr/people/pfoser/data.html

100. Frentzos, E.: Indexing objects moving on fixed networks. In Proc. of the 8th Intl. Symp. on Spatial and Temporal Databases (SSTD), pp. 289-305, (2003)

101. Pfoser, D., and Jensen, C.S.: Indexing of network constrained moving objects. In Proc. of the 11th Intl. Symp. on Advances in Geographic Information Systems (ACM-GIS), (2003)

102. de Almeida, V. T., Güting, R. H.: Indexing the Trajectories of Moving Objects in Networks. GeoInformatica 9(1): 33-60 (2005)

103. Garey, M. R., and Johnson, D. S.: Computers and Intractability; A Guide to the Theory of NP-Completeness, (1979)

104. Hadjieleftheriou, M., Kollios, G., Tsotras, J., and Gunopulos, D.: Indexing spatiotemporal archives, VLDB Journal 15(2): 143-164 (2006)

105. Jensen, C. S., Lahrmann, H., Pakalnis, S., and Runge, J.: The INFATI Data, Aalborg University, TimeCenter TR-79. Available at: http://www.cs.aau.dk/TimeCenter (2008)

106. Karypis, G., and Kumar, V.: A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs, SIAM Journal on Scientific Computing, Vol. 20, No. 1, pp. 359-392, (1999)

107. METIS - Family of Multilevel Partitioning Algorithms. Available at: http://glaros.dtc.umn.edu/gkhome/views/metis

108. Pelanis, M., Saltenis, S., and Jensen, C. S.: Indexing the past, present, and anticipated future positions of moving objects, ACM Trans. Database Syst. 31(1): 255-298 (2006)

109. Pfoser, D., and Jensen, C. S.: Capturing the uncertainty of moving-object representations. In Proc. 6th SSD conf., pages 111–132, (1999)

110. Brakatsoulas, S., Pfoser, D., Salas, R., Wenk, C.: On Map-Matching Vehicle Tracking Data. VLDB 2005: 853-864

111. GeoServer. Available at: http://geoserver.org/display/GEOS/Welcome

112. GoogleMaps. Available at: http://maps.google.com/

113. Meratnia, N., and de By, R. A.: Spatiotemporal Compression Techniques for Moving Point Objects. EDBT 2004: 765-782