A State-of-the-Art in Spatio-Temporal Data Warehousing, OLAP and Mining

Leticia Gómez
Instituto Tecnológico de Buenos Aires, Argentina

Bart Kuijpers, Bart Moelans,
University of Hasselt and
Transnational University of Limburg, Belgium

Alejandro Vaisman
Universidad de la República, Uruguay

Abstract

Geographic Information Systems (GIS) have been extensively used in various application domains, ranging from economical, ecological and demographic analysis, to city and route planning. Nowadays, organizations need sophisticated GIS-based Decision Support System (DSS) to analyze their data with respect to geographic information, represented not only as attribute data, but also in maps. Thus, vendors are increasingly integrating their products, leading to the concept of SOLAP (Spatial OLAP). Also, in the last years, and motivated by the explosive growth in the use of PDA devices, the field of moving object data has been receiving attention from the GIS community, although not much work has been done to provide moving object databases with OLAP capabilities. In the first part of this paper we survey the SOLAP literature. We then address the problem of trajectory analysis, and review recent efforts regarding trajectory data warehousing and mining. We also provide an in-depth comparative study between two proposals: the GeoPKDD project (that makes use of the Hermes system), and Piet, a proposal for SOLAP and moving objects, developed at the University of Buenos Aires, Argentina. Finally, we discuss future directions in the field, including SOLAP analysis over raster data.

Keywords: GIS, OLAP, spatial OLAP, SOLAP, Spatio-temporal OLAP, Trajectory Data Warehousing, Trajectory mining.

INTRODUCTION

Geographic Information Systems (GIS) have been extensively used in various application domains, ranging from economical, ecological and demographic analysis, to city and route planning (Rigaux, Scholl, &
Voisard, 2001; Worboys, 1995). Spatial information in a GIS is typically stored in different so-called thematic layers (also called themes). Information in themes can be stored in data structures according to different data models, the most usual ones being the raster model and the vector model. In a thematic layer, spatial data is annotated with classical relational attribute information, of (in general) numeric or string type. While spatial data is stored in data structures suitable for these kinds of data, associated attributes are usually stored in conventional relational databases. Spatial data in the different thematic layers of a GIS system can be mapped univocally to each other using a common frame of reference, like a coordinate system. These layers can be overlapped or overlayed to obtain an integrated spatial view.

On the other hand, OLAP (On Line Analytical Processing) (Kimball, 1996; Kimball & Ross, 2002) comprises a set of tools and algorithms that allow efficiently querying multidimensional databases containing large amounts of data, usually called data warehouses. In OLAP, data is organized as a set of dimensions and fact tables. In the multidimensional model, data can be perceived as a data cube, where each cell contains a measure or set of (probably aggregated) measures of interest. As we discuss later, OLAP dimensions are further organized in hierarchies that favor data aggregation (Cabibbo & Torlone, 1997). Several techniques and algorithms have been developed for multidimensional query processing, most of them involving some kind of aggregate precomputation (Harinarayan, Rajaraman, & Ullman, 1996).

The need for OLAP in GIS

Different data models have been proposed for representing objects in a GIS. ESRI (http://www.esri.com) first introduced the Coverage data model to bind geometric objects to non-spatial attributes that describe them. Later, they extended this model with object-oriented support, in a way that behavior can be defined for geographic features (Zeiler, 1999). The idea of the Coverage data model is also supported by the Reference Model proposed by the Open Geospatial Consortium (http://www.opengeospatial.org). Thus, in spite of the model of choice, there is always the underlying idea of binding geometric objects to objects or attributes stored in (mostly) object-relational databases (Stonebraker & Moore, 1996). In addition, query tools in commercial GIS allow users to overlap several thematic layers in order to locate objects of interest within an area, like schools or fire stations. For this, they use indexing structures based on R-trees (Gutman, 1984). GIS query support sometimes includes aggregation of geographic measures, for example, distances or areas (e.g., representing different geological zones). However, these aggregations are not the only ones that are required, as we discuss below.
Nowadays, organizations need sophisticated GIS-based Decision Support System (DSS) to analyze their data with respect to geographic information, represented not only as attribute data, but also in maps, probably in different thematic layers. In this sense, OLAP and GIS vendors are increasingly integrating their products (see, for instance, Microstrategy and MapInfo integration in http://www.microstrategy.com/, and http://www.mapinfo.com/). Thus, aggregate queries are central to DSSs. Classical aggregate OLAP queries (like “Total sales of cars in California”), and aggregation combined with complex queries involving geometric components (“Total sales in all villages crossed by the Mississippi river and within a radius of 100 km around New Orleans”) must be efficiently supported. Moreover, navigation of the results using typical OLAP operations like roll-up or drill-down is also required. These operations are not supported by commercial GIS in a straightforward way. One of the reasons is that the GIS data models discussed above were developed with transactional queries in mind. Thus, databases storing non-spatial attributes or objects are designed to support those (non-aggregate) kinds of queries. Decision support systems need a different data model, where non-spatial data, probably consolidated from different sectors in an organization, are stored in a warehouse. Here, numerical data are stored in fact tables built along several dimensions. For instance, if we are interested in the sales of certain products in stores in a given region, we may consider that sales amounts are stored in a fact table with three dimensions Store, Time and Product. In order to guarantee summarizability (Lenz & Shoshani, 1997), dimensions are organized into aggregation hierarchies. For example, stores can aggregate over cities which in turn can aggregate into regions and countries. Each of these aggregation levels can also hold descriptive attributes like city population, the area of a region, etc. GIS-DSS integration requires warehouse data to be linked to geographic data. For instance, a polygon representing a region must be associated to its corresponding region identifier in the warehouse. In current commercial applications, the GIS and OLAP worlds are integrated in an ad-hoc fashion, probably in a different way (and using different data models) each time an implementation is required, even when a data warehouse is available for non-spatial data.

An Introductory Example. We present now a real-world example to introduce the problem. We selected four layers with geographic and geological features obtained from the National Atlas Website (http://www.nationalatlas.gov). These layers contain the following information: states, cities, and rivers in North America, and volcanoes in the northern hemisphere (published by the Global Volcanism Program - GVP). Figure 1 shows a detail of the layers containing cities and rivers in North America. Note the density of the points representing cities (particularly in the eastern region). Rivers are represented as polylines. Figure 2 shows a portion of two overlayed
layers containing states (represented as polygons) and volcanoes (points) in the northern hemisphere. There is also non-spatial information stored in a conventional data warehouse. In this data warehouse, dimension tables contain customer, stores and product information, and a fact table contains store sales across time. Also, numerical and textual information on the geographic components exist (e.g., population, area), stored as usual as attributes of the GIS layers. In the scenario above, conventional GIS and organizational data can be integrated for decision support analysis. Sales information could be analyzed with respect to geographical features conveniently *displayed in maps*. This analysis could benefit from the integration of both worlds in a single framework. Even though this integration could be possible with existing technologies, ad-hoc solutions are expensive because, besides requiring lots of complex coding, they are hardly portable. To make things more difficult, ad-hoc solutions require data exchange between GIS and OLAP applications to be performed. This implies that the output of a GIS query must be probably exported to become members of dimensions in a data cube, and merged for further analysis. For example, suppose that a business analyst is interested in studying the sales of nautical goods in stores located in cities crossed by rivers. She would first query the GIS, to obtain the cities of interest. She probably has stored sales in a data cube containing a dimension *Store*, with *city* as a dimension level. She would need to “manually” select the cities of interest (i.e., the ones returned by the GIS query) in the cube, to be able to go on with the analysis (in the best case, an ad-hoc customized middleware could help her). Of course, she must repeat this for each query involving a (geographic) dimension in the data cube.
GIS/Data warehousing integration can provide a more natural and general solution. This survey discusses some efforts in integrating spatial and multidimensional data.

**Spatio-temporal Data Warehousing and Mining**

The second part of this survey is devoted to spatio-temporal data warehousing, OLAP, and mining, in particular, addressing moving object databases (MOD). Moving object databases have been receiving increasing attention from the database community in recent years, mainly due to the wide variety of applications that technology allows nowadays. Trajectories of moving objects like cars or pedestrians, can be reconstructed from samples describing the locations of these objects at certain points in time. Although there exist many proposals...
for modeling and querying moving objects, only a small part of these proposals address the problem of aggregation of moving objects data in a GIS scenario. Many interesting applications arise, involving moving objects aggregation, mainly regarding traffic analysis, truck fleet behavior analysis, commuter traffic in a city, passenger traffic in an airport, or shopping behavior in a mall. Trajectory data warehouses have been proposed to help in the analysis of these scenarios. In this survey we review the relevant works and issues in the trajectory data warehousing problem.

Data Mining techniques are aimed at discovering hidden, non-trivial information and patterns in large databases. MOD are particularly suited for applying these techniques. For example, interesting patterns describing the movement followed by cars or pedestrians in a city can be discovered. These patterns can be used, for instance, to define the most appropriate spots to place street advertisement, or to make the public transport system more efficient. We also survey the latest findings in this area.

**Figure 2.** Two overlayed layers containing states in North America and volcanoes in the northern hemisphere.

**Novel Topics in Spatio-Temporal OLAP**
In the field of GIS and OLAP integration, most efforts address discrete spatial data. However, in the last few years, GIS practitioners (in particular in the environmental domain), envisioned the possibility of performing complex analysis tasks over continuous fields (i.e., raster data), which describe the distribution of physical phenomena that change continuously in time and/or space, like temperature, pressure, or land elevation. Besides physical geography, continuous fields like land use and population density, are used in human geography as an aid in spatial decision-making process. Although some work has been done to support querying fields in GIS, the area of spatial multidimensional analysis of continuous data is still almost in its infancy. Here we comment on current and future steps in this topic.

The remainder of this chapter is organized as follows. We first provide a brief background on GIS, data warehousing and OLAP. Then, we review the state-of-the-art in Spatial Data Warehousing and SOLAP. After that, we discuss a recently introduced taxonomy for spatio-temporal data warehousing. We move on to the study of spatio-temporal data warehousing, OLAP and mining, including an in-depth analysis of the GeoPKDD proposal for trajectory data warehouses (TDW). Following this analysis, we provide a detailed analysis of the Piet framework, aimed at integrating GIS, OLAP and moving objects data. We then present recent proposals on trajectory mining. After comparing TDW and Piet, we conclude the chapter with a description of research directions in the field of spatio-temporal OLAP.

A SHORT BACKGROUND IN GIS AND OLAP

GIS
In general, information in a GIS application is stored over several thematic layers. Information in each layer consists of purely spatial data on the one hand, that is combined with classical alpha-numeric attribute data on the other hand (usually stored in a relational database). Two main data models are used for the representation of the spatial part of the information within one layer, the vector model and the raster model. The choice of model typically depends on the data source from which the information is imported into the GIS.

The Vector Model. The vector model is used the most in current GIS (Kuper & Scholl, 2000). In the vector model, infinite sets of points in space are represented as finite geometric structures, or geometries, like, for example, points, polylines and polygons. More concretely, vector data within a layer consists in a finite number of tuples of the form (geometry, attributes) where a geometry can be a point, a polyline or a polygon. There are several possible data structures to actually store these geometries (Worboys, 1995).
The Raster Model. In the raster model, the space is sampled into pixels or cells, each one having an associated attribute or set of attributes. Usually, these cells form a uniform grid in the plane. For each cell or pixel, the sample value of some function is computed and associated to the cell as an attribute value, e.g., a numeric value or a color. In general, information represented in the raster model is organized into zones, where the cells of a zone have the same value for some attribute(s). The raster model has very efficient indexing structures and it is very well-suited to model continuous change but its disadvantages include its size and the cost of computing the zones. Spatial information in the different thematic layers in a GIS is often joined or overlayed. Queries requiring map overlay are more difficult to compute in the vector model than in the raster model. On the other hand, the vector model offers a concise representation of the data, independent on the resolution. Except when we address continuous fields, we will refer to the vector model in this survey, although, indeed, conceptually, each cell is, and each pixel can be regarded as a small polygon, allowing a uniform treatment; moreover, the attribute value associated to the cell or pixel can be regarded as an attribute in the vector model.

Data Warehousing and OLAP
The importance of data analysis using OLAP tools has increased significantly in recent years as organizations in all sectors are required to improve their decision-making processes in order to maintain their competitive advantage. OLAP systems are based on a multidimensional model, which allows a better understanding of data for analysis purposes and provides better performance for complex analytical queries. The multidimensional model allows viewing data in an n-dimensional space, usually called a data cube. In this cube, each cell contains a measure or set of (probably aggregated) measures of interest. This factual data can be analyzed along dimensions of interest, usually organized in hierarchies (Cabibbo & Torlone, 1997). Three typical implementations of OLAP tools exist: MOLAP (standing for multidimensional OLAP), where data is stored in proprietary multidimensional structures, ROLAP (relational OLAP), where data is stored in (object) relational databases, and HOLAP (standing for hybrid OLAP, which provides both solutions. In a ROLAP environment, data is organized as a set of dimension tables and fact tables, and we assume this organization in the remainder of the paper. There are a number of OLAP operations that allow exploiting the dimensions and their hierarchies, thus providing an interactive data analysis environment. Data warehouses are optimized for OLAP operations which, typically, imply data aggregation or de-aggregation along a dimension, called roll-up and drill-down, respectively. Other operations involve selecting parts of a cube (slice and dice) and re-orienting the multidimensional view of data (pivoting). In addition to the basic operations described above, OLAP tools provide a great
variety of mathematical, statistical, and financial operators for computing ratios, variances, ranks, etc.

### Temporal Data Warehouses

The relational data model as proposed by Codd (1970), is not well-suited for handling spatial and/or temporal data. Data evolution over time must be treated in this model, in the same way as ordinary data. This is not appropriate for applications that require past, present, and/or future data values to be dealt with. In real life such applications abound. Therefore, in the last decades, much research has been done in the field of temporal databases. Snodgrass (1995) describes the design of the TSQL2 Temporal Query Language, an upward compatible extension of SQL-92. The book, written as a result of a Dagstuhl seminar organized in June 1997 by Etzion, Jajodia, and Srivada (1998), contains comprehensive bibliography, glossaries for both temporal database and time granularity concepts, and summaries of work around 1998. The same author (Snodgrass, 1999), in other work, discusses practical research issues on temporal database design and implementation.

In the temporal data warehousing and OLAP field, Mendelzon and Vaisman (2000, 2003) introduce the TOLAP model, and developed a prototype and a datalog-like query language, based on a (temporal) star schema. Vaisman, Izquierdo, and Ktenas (2006) also present a Web-based implementation of this model, along with a query language, called TOLAP-QL. Along similar lines, Eder, Koncilia, and Morzy (2002) propose a data model for temporal OLAP supporting structural changes. Although these efforts, little attention has been devoted to the problem of conceptual and logical modeling for temporal data warehouses.

### SPATIAL DATA WAREHOUSING AND OLAP

*Spatial database systems* have been studied for a long time (Buchmann, Günther, Smith, & Wang, 1990; Paredaens, Van Den Bussche, & Gucht, 1994). Rigaux et al. (2001) survey various techniques, such as spatial data models, algorithms, and indexing methods, developed to address specific features of spatial data that are not adequately handled by mainstream DBMS technology. Although some authors have pointed out the benefits of combining GIS and OLAP, not much work has been done in this field. Vega López, Snodgrass, and Moon (2005) present a comprehensive survey on spatiotemporal aggregation that includes a section on spatial aggregation. Also, Bédard, Rivest, and Proulx (2007) present a review of the efforts for integrating OLAP and GIS.

**Conceptual Modeling and SOLAP**

Rivest, Bédard, and Marchand (2001) introduce the concept of SOLAP (standing for Spatial OLAP), a paradigm aimed at being able to explore...
spatial data by drilling on maps in an OLAP fashion. They describe the desirable features and operators a SOLAP system should have. Although they do not present a formal model for this, SOLAP concepts and operators have been implemented in a commercial tool called JMAP, developed by the Centre for Research in Geomatics and KHEOPS, see http://www.kheops-tech.com/en/jmap/solap.jsp. Stefanovic, Han, and Koperski (2000) and Bédard, Merret, and Han (2001), classify spatial dimension hierarchies according to their spatial references in: (a) non-geometric; (b) geometric to non-geometric; and (c) fully geometric. Dimensions of type (a) can be treated as any descriptive dimension (Rivest et al., 2001). In dimensions of types (b) and (c), a geometry is associated to members of the hierarchies. Malinowski and Zimányi (2004) extend this classification to consider that even in the absence of several related spatial levels, a dimension can be considered spatial: a dimension level is spatial if it is represented as a spatial data type (e.g., point, region), allowing them to link spatial levels through topological relationships (e.g., contains, overlaps). Thus, a spatial dimension is a dimension that contains at least one spatial hierarchy. A critical point in spatial dimension modeling is the problem of multiple-dependencies, meaning that an element in one level can be related to more than one element in a level above it in the hierarchy. Jensen, Kligys, Pedersen, and Timko (2004) address this issue, and propose a multidimensional data model for mobile services, i.e., services that deliver content to users, depending on their location. This model supports different kinds of dimension hierarchies, most remarkably multiple hierarchies in the same dimension, i.e., multiple aggregation paths. Full and partial containment hierarchies are also supported. However, the model does not consider the geometry, limiting the set of queries that can be addressed. That means, spatial dimensions are standard dimensions referring to some geographical element (like cities or roads). Malinowski and Zimányi (2006) also propose a model supporting multiple aggregation paths. Pourabbas (2003) introduces a conceptual model that uses binding attributes to bridge the gap between spatial databases and a data cube. The approach relies on the assumption that all the cells in the cube contain a value, which is not the usual case in practice, as the author expresses. Also, the approach requires modifying the structure of the spatial data to support the model. Shekhar, Lu, Tan, Chawla, & Vatsavai (2001) introduce MapCube, a visualization tool for spatial data cubes. MapCube is an operator that, given a so-called base map, cartographic preferences and an aggregation hierarchy, produces an album of maps that can be navigated via roll-up and drill-down operations.

Spatial Measures. Measures are characterized in two ways in the literature, namely: (a) measures representing a geometry, which can be aggregated along the dimensions; (b) a numerical value, using a
topological or metric operator. Most proposals support option (a), either as a set of coordinates (Bédard et al., 2001; Rivest et al., 2001; Malinowski & Zimányi, 2004; Bimonte, Tchounikine, & Miquel, 2005), or a set of pointers to geometric objects (Stefanovic et al., 2000). Bimonte et al. (Bimonte et al., 2005) define measures as complex objects (a measure is thus an object containing several attributes). Malinowski and Zimányi (2004) follow a similar approach, but defining measures as attributes of an n-ary fact relationship between dimensions. Damiani and Spaccapietra (2006) propose MuSD, a model allowing defining spatial measures at different granularities. Here, a spatial measure can represent the location of a fact at multiple levels of (spatial) granularity.

**Spatial Aggregation**

In light of the discussion above, it should be clear that aggregation is a crucial issue in spatial OLAP. Moreover, there is not yet a consensus about a complete set of aggregate operators for spatial OLAP. We now discuss the classic approaches to spatial aggregation.

Han et al. (1998) use OLAP techniques for materializing selected spatial objects, and proposed a so-called *Spatial Data Cube*, and the set of operations that can be performed on this data cube. The model only supports aggregation of spatial objects.

Pedersen and Tryfona (2001) propose the pre-aggregation of spatial facts. First, they pre-process these facts, computing their disjoint parts in order to be able to aggregate them later. This pre-aggregation works if the spatial properties of the objects are *distributive* over some aggregate function. The paper does not address forms other than polygons, although the authors claim that other more complex forms are supported by the method, and the authors do not report experimental results.

With a different approach, Rao, Zhang, Yu, Li, and Chen (2003), and Zhang, Li, Rao, Yu, Chen, and Liu (2003) combine OLAP and GIS for querying spatial data warehouses, using R-trees for accessing data in fact tables. The data warehouse is then exploited in the usual OLAP way. Thus, they take advantage of OLAP hierarchies for locating information in the R-tree which indexes the fact table. The work assumes that some fact table containing the identifiers of spatial objects exists. These objects happen to be points, which is quite unrealistic in a GIS environment, where different types of objects appear in the different layers.

Some interesting techniques have been recently introduced to address the data aggregation problem. These techniques are based on the combined use of (R-tree-based) indexes, materialization (or pre-aggregation) of aggregate measures, and computational geometry algorithms.
Papadias, Tao, Kalnis, and Zhang (2002) introduce the *Aggregation R-tree* (aR-tree), combining indexing with pre-aggregation. The aR-tree is an R-tree that annotates each MBR (Minimal Bounding Rectangle) with the value of the aggregate function for all the objects that are enclosed by it. They extend this proposal in order to handle historic information, denoting this extension aRB-tree (Papadias, Tao, Zhang, Mamoulis, Shen, and Sun, 2002). The approach basically consists in two kinds of indexes: a host index, which is an R-tree with the summarized information, and a B-tree containing time-varying aggregate data. In the most general case, each region has a B-tree associated, with the historical information of the measures of interest in the region. This is a very efficient solution for some kinds of queries, for example, window aggregate queries (i.e., for the computation of the aggregate measure of the regions which intersect a spatio-temporal window). In addition, the method is very effective when a query is posed over a query region whose intersection with the objects in a map must be computed on-the-fly, and these objects are totally enclosed in the query region. However, problems may appear when leaf entries partially overlap the query window. In this case, the result must be estimated, or the actual results computed using the base tables. In fact, Tao, Kollios, Considine, Li, and Papadias (2004) show that the aRB-tree can suffer from the distinct counting problem, if the object remains in the same region for several timestamps.

**MOVING OBJECTS AND TRAJECTORY DATA**

The field of moving object databases has been extensively studied in the last ten years, mainly regarding data modeling an indexing. Güting and Schneider (2005) provide a good reference to this large corpus of work. Moving objects, carrying location-aware devices, produce *trajectory* data in the form of a sample of \((O_{id},x,y,t)\)-tuples, that contain object identifier and time-space information. In this survey we will focus on the problem of building trajectory data warehouses and exploiting them through OLAP and data mining techniques. We first need to introduce some concepts about moving object data.

Wolfson, Sistla, Xu, and Chamberlain (1999) define a set of capabilities that a moving object database must have, and introduce the DOMINO system, that develops those features on top of existing database management systems (DBMS). Hornsby and Egenhofer (2002) introduce a framework for modeling moving objects, which supports viewing objects at different granularities, depending on the sampling time interval. The basic modeling element they consider is a *geospatial lifeline*, which is composed of triples of the form \(<Id,location,\text{time}>\), where \(Id\) is the identifier of the object, \(location\) is given by x-y coordinates, and \(\text{time}\) is the timestamp of the observation. The possible positions of an object between two observations is estimated to be
within two inverted half-cones that conform a lifeline bead, whose projection over the x-y plane is an ellipse.

Particular interest has received the topic of moving objects on road networks. Van de Weghe et al. propose a qualitative trajectory calculus for objects in a GIS (Weghe, Cohn, Tré, & Maeyer, 2005), based on the assumption that in a GIS scenario, qualitative information is necessary. Kuijpers, Moelans, and Van de Weghe (2006) show by means of experiments, the practical use of this calculus. For mining trajectories in road networks, Brakatsoulas, Pfoser, and Tryfona (2004) propose to enrich trajectories of moving objects with information about the relationships between trajectories (e.g., intersect, meets), and between a trajectory and the GIS environment (stay within, bypass, leave). They also proposed a data mining language denoted SML (for Spatial Mining Language). This language is oriented to traffic networks, and it is not clear how it could be extended to other scenarios.

Also in the framework of road traffic mining, Gonzalez, Han, Li, Myslinska, and Sondag (2007) use a partitioning approach for obtaining interesting driving and speed patterns from large sets of traffic data. They compute frequent path-segments at the area level with a support relative to the traffic in the area (i.e., a kind of adaptive support), and propose an algorithm to automatically partition a road network and build a hierarchy of areas.

The work of Lee, Han, and Whang (2007) is aimed at discovering common sub-trajectories using a partitioning strategy which divides a trajectory into a set of line segments, and then groups similar line segments together into a cluster.

Like in the case of spatial OLAP (and multidimensional databases, in general), from the conceptual modeling point of view, there has not been much interest from the database community. Malinowski and Zimányi (2006) propose a model to provide a graphical representation, based on the Entity/Relationship model, and on UML.

The problem of trajectory similarity and aggregation is a new topic in the spatio-temporal database literature. Meratnia and de By (2002) study trajectory aggregation by identifying similar trajectories, merging them in a single one, and dividing the area under study into homogeneous spatial units. Pelekis, Theodoridis, Vosinakis, and Panayiotopoulos (2006), and Pelekis and Theodoridis (2006) introduce a framework consisting of a set of distance operators based on parameters of trajectories like speed and direction, to determine if two trajectories can be considered similar. Frentzos, Gratsias, and Theodoridis (2007) propose an approximation method for supporting the k-most-similar-trajectory search using R-tree structures.

Finally, Kuijpers and Vaisman (2007) present a taxonomy of aggregate queries on moving object data.

**Adding Semantic Information to Trajectory Data**
Techniques that add semantic information to trajectory data have been recently proposed. Mouza and Rigaux (2005) present a model where trajectories are represented by a sequence of moves. They propose a query language based on regular expressions, aimed at obtaining so-called mobility patterns. Note that this language, as well as the proposals commented above, does not relate trajectories with the geographic environment where they occur, which limits the types of queries that can be addressed. Along the same lines, Damiani, Macedo, Parent, Porto, and Spaccapietra (2007) introduced the concept of stops and moves, in order to enrich trajectories with semantically annotated data (later in this survey we give more details about the stops and moves paradigm).

Giannotti, Nanni, Pinelli, and Pedreschi (2007) studied trajectory pattern mining, based on so-called Temporally Annotated Sequences (TAS), an extension of sequential patterns, where a temporal annotation between two nodes is defined. In this way, the sequence <1,2,2> defines a pattern that starts at position 1 and after 2 seconds arrives at position 2. In other words, a trajectory pattern is a set of trajectories that visit the same sequence of places with similar travel times between each of them. They also introduce the concept of Region of Interest (RoI), and focus on computing the RoIs dynamically from the trajectories. Similarly, Gómez, Kuijpers, and Vaisman (2008a, 2008b) propose to replace a trajectory by a sequence of its stops and moves, following the ideas of Alvares et al. (2007). This work differs from the one of Giannotti et al. (2007) in several ways. First, the authors work with stops and moves instead of pre-defined regions of interest. This allows identifying which of the RoIs are really relevant to a trajectory. Second, the stops and moves are used to “encode” or compress a trajectory, which, in many practical situations turns out to be enough to identify interesting sequences very efficiently. A third difference is that in this proposal, the user defines the places of interest of an application in advance, and then they compute the stops and moves to perform trajectory mining. Finally, the approach of Gómez et al. (discussed later in detail) is aimed at integrating trajectories and geographic data, an issue mentioned albeit not addressed in (Giannotti et al., 2007).

A TAXONOMY FOR SPATIO-TEMPORAL DATA WAREHOUSING AND OLAP

From the previous sections, the need of a formal definition of the meaning of a “SOLAP query” or a “Spatio-temporal OLAP query” becomes clear. Comparing proposals or assessing the capabilities of different approaches becomes difficult without a precise definition of
Vaisman and Zimányi (2009) tackle this problem in the following way: (a) first, they define a taxonomy of classes that integrate OLAP, spatial data, and moving data types; (b) for each class in this taxonomy, they define the queries that they support. For this, they use the classic tuple relational calculus extended with aggregate functions (Klug, 1982), and incrementally extend this calculus with spatial and moving data types, showing that each extension defines the kinds of queries in each class of the taxonomy. To address spatio-temporal scenarios they use the data types defined by Guting et al. (2000). The taxonomy, depicted in Figure 3 is defined as follows. There exist four basic classes: Temporal dimensions, OLAP, GIS, and moving data types. As a derived class, the addition of moving data types to GIS produces Spatio-Temporal data, typically allowing trajectory analysis in a geographic environment. Providing OLAP with the ability of handling temporal dimensions produces the concept of Temporal OLAP (TOLAP). The interaction of OLAP and GIS is denoted Spatial OLAP (SOLAP). The interaction between GIS and TOLAP is called Spatial TOLAP (S-TOLAP). Adding OLAP capabilities to spatio-temporal data results in Spatio-Temporal OLAP (ST-OLAP). Finally, if the latter supports temporal dimensions, we are in the Spatio-Temporal TOLAP class (ST-TOLAP).

Figure 3 The taxonomy for spatio-temporal OLAP, from Vaisman & Zimányi (2009).

The authors start from the class of OLAP queries, which includes all queries expressible in Klug's relational algebra with aggregates (Klug, 1982). Extending this algebra with spatial data types leads to the class of SOLAP queries. For example, “Total population in the districts
within 3KM from the district of Antwerp” is a SOLAP one, since it requires spatial types to solve the “distance” part of the query. Queries in the TOLAP class support evolution of the dimension instances in the data warehouse, a problem also referred to as slowly-changing dimensions (Kimball, 1996). Queries in the spatio-temporal OLAP (ST-OLAP) class account for the case when the spatial objects evolve over time, therefore, moving types must be included in the query. For example, the query “For each district and polluting cloud, give the duration of the passing of the cloud over the district.” Finally, Spatial TOLAP (S-TOLAP) covers the case when in addition to having spatial objects and attributes in the data warehouse, the dimensions are also temporal.

THE GEOPKDD APPROACH FOR TRAJECTORY DATA WAREHOUSING

Orlando, Orsini, Raffaetá, Roncato, and Silvestri (2007) were the first ones to propose a trajectory data warehouse (TDW) to analyze moving object data. This TDW is aimed at providing the infrastructure needed to deliver advanced reporting capabilities and facilitating the application of mining algorithms on aggregate data. It was devised for the GeoPKDD project (see http://www.geopkdd.eu). Since this project is based on the Hermes architecture for moving object data, we first give a brief overview of the Hermes system (Pelekis, N., Theodoridis, Y., Vosinakis, S., and Panayiotopoulos, T., 2006; Pelekis & Theodoridis, 2006).

The Hermes System for Location-Based Services. Hermes provides the functionality needed for handling two-dimensional objects that change location, shape and size, through four kinds of data types: (a) static base data types (b) static temporal data types; (c) static spatial types; (d) moving data types. Data of type (a) are the standard DBMS data types (integer, real, etc.). Data of type (b) are based on the so-called TAU temporal object model (Kakoudakis, 1996), and provides Hermes with temporal object-relational capabilities, through a library denoted TAU-TLL (Pelekis, 2002). The new temporal data types supported (extending the ODMG data model) are Timepoint, Period, and Temporal Element. The spatial data types (c) are provided by the Oracle Spatial library. The object type defined in Oracle, and used by Hermes, is called Sdo_Geometry. The Moving data type (d) encapsulates semantics and functionality of different data types: moving point, linestring, circle, rectangle, polygon, and moving collection. Below these types, a class hierarchy is defined. The basic type is moving point, defined as a sequence of different types of simple functions. It is based on the sliced representation proposed by Güting,
Böhlen, Jensen, Lorentzos, Schneider and Vazirgiannis (2000). Here, a temporal development of a moving object is decomposed in slices such that, between each slice, a simple function is defined. The idea is to decompose the definition of each moving type into several definitions, one for each function. The composition of these sub-definitions defines a moving type. This way, a unit_function models the case where a user is located at a point \((x_i, y_i)\) and moves, with initial velocity \(v\) and acceleration \(a\) or a linear or circular arc route. A flag indicates the type of movement. The point \((x_e, y_e)\) is the end point of the movement. The unit function along with the period object type, conforms the moving point data type, which is the basis for the other types. For instance, the type moving circle is formed by the function unit_moving_circle plus the period data type. In turn, the former is composed of three unit_moving_point objects. Details on these data types can be found in (Pelekos & Theodoridis, 2006). The objects belonging to the moving type are provided with a set of operations: (a) topological and distance predicates, like within_distance; (b) temporal functions, like add_unit (adds a new unit of movement), and at_instant (returns the union of the projection of a moving object at a time instant); (c) distance and direction operators (for instance, the distance between two moving objects); (d) set relationships (like intersection). Also, numeric operations on objects are supported, like area or length. In consequence, it would be easy to compute, for instance, the area of an object at a given time instant.

The Hermes architecture can be described as follows: the basic components are the TAU-TLL library, the Oracle spatial cartridge and the Hermes-MDC (Moving Data Cartridge), which includes the moving data types. PL-SQL statements, which are compiled and stored in binary form, use those cartridges and data types. Thus, the PL-SQL statements are available for interacting with Oracle 10g data structures. Applications written, for instance, in Java, can consume these data. The types of queries supported by Hermes are: (a) queries on stationary objects, like: point, range, distance-based, topological, and nearest-neighbor queries; (b) queries on moving reference objects (distance-based and similarity queries); (c) join queries; (d) queries involving unary operators (traveled distance, speed).

Actually, given that Hermes consists, essentially, in a set of data types, an application designer can define a database schema that uses these data types, and take advantage of their functionality. For example, to describe the movement of a toxic cloud, one could define a relation:

\[
\text{Cloud (id:integer, name: varchar, shape: moving_polygon)}
\]

Then, an application programmer could write code that uses these data structures, to find, for instance, when did the cloud arrived to California. Formally, the expressive power is provided by the data types, because there is no language associated to Hermes. Instead, a host language (Java, PL-SQL, both) is used.
Having introduced Hermes, we can continue with the discussion on trajectory data warehousing. In short, a TDW allows analyzing measures of interest like the number of moving objects in different urban areas, average speed, or speed change. Over the TDW, data mining techniques can also be used to discover traffic-related patterns, as we discuss later. An ETL (Extract-Transform-Load) procedure feeds a TDW with aggregate trajectory data, obtained from raw data consisting in the spatio-temporal positions of moving objects. A data cube is built from the TDW, aggregating measures for OLAP purposes. The trajectories to be analyzed present characteristics of different kinds: numeric (such as the average speed, direction, duration); spatial (geometric shape of the trajectory), and temporal (timing of the movement).

In order to support trajectory data, a spatio-temporal data cube should allow analysis along (a) temporal dimensions; (b) spatial dimensions at different levels of granularity (point, cell, road); (c) thematic dimensions, containing, for instance, demographic data. In this sense, hierarchies must take into account the fact that an element may rollup to more than one in an upper level. For instance, a road can probably cross more that one cell, yielding a relation instead of a function between a level cell to a level road. It is worth noticing that some proposals deal with this problem defining complex relationships (e.g., containment) in the dimension hierarchies (Jensen et al., 2004), which in general, lead to approximations. The Piet framework, discussed below, defines different GIS dimensions for different kinds of geometries, and the query language takes care of the problem of finding out the cells that intersect the road in the example above.

Typically, since the space is usually divided into cells, measures of interest are, among other ones: (a) The number of trajectories found in the cell (or started/ended their path in the cell, or crossed/entered/left the cell); (b) The average (or minimum or maximum) distance covered by trajectories in a cell; (c) The average (or minimum or maximum) time required to cover the distance in (b); The speed and change of speed (acceleration), direction and change of direction (turn). Finally, a TDW algebra should support typical OLAP operators like roll-up, drill-down, and slice and dice.

Figure 4, taken from Damiani, Vangenot, Frentzos, Marketos, Theodoridis, Veryklos, and Raffaeta (2007), depicts the trajectory warehouse architecture proposed in the GeoPKDD project. Initially, location data are captured, and handled by a so-called trajectory stream manager, which builds trajectories from these data (e.g., splitting the raw data according to some criteria), providing a trajectory identifier. This process is called trajectory reconstruction. Trajectories are stored in a relational table, denoted RelTrajectories, and then loaded into a moving object database (MOD), which is in turn
managed by the Hermes system introduced above. Basically, the MOD includes a relation MODTrajectories with schema (O_id, trajectory_id, trajectory), where trajectory is of type Moving Point. Actually, this

results in redundancy, since trajectories are stored twice. This redundancy is a consequence of the dependency of Hermes upon Oracle (MODTrajectories provides access to the moving point data type). Finally, an ETL (Extraction, Transformation, and Loading) process feeds the TDW. Queries to this warehouse can include geographical data.

From a modeling point of view, a TDW is based on the classic star schema. It contains a standard temporal dimension, and two spatial dimensions. The former ranges over equally sized time intervals, which are aggregated according to larger intervals as we move up in the dimension hierarchy (e.g., the interval [60,120] aggregates over the interval [0,120]). The spatial dimensions, denoted DimX and DimY, range over equally sized spatial intervals (x,y, respectively), defining the cells where measures are recorded. There is a fact table containing references to the dimensions and measures of the kinds commented above. Roll-up and drill-down are performed aggregating measures over the cells at different granularities (for instance, combining two or more cells). The key of this fact table is composed of the keys of the dimensions, namely dimX_id, dimY_id, dimT_id. We remark that the actual implementation has a slightly different form than this model, although for presentation clarity, in what follows we base ourselves on this structure. Details of the actual implementation are given in (Marketos, Frentzos, Ntousi, Pelekis, Raffaeta, & Theodoridis, 2008). It is important to note that no trajectory information is recorded in the TDW whatsoever. This information lies only in the MOD, and can be used for querying, along with the information in the TDW, in order to obtain higher level information. The implementation of the TDW

---

**Figure 4.** The trajectory warehouse architecture.
makes use of the Oracle 10g data warehousing tools. In addition, modeling and storing trajectories is performed with Hermes.

The Double Counting Problem and the ETL Process. During roll-up, and due to the characteristics of the TDW, double counting may introduce errors in these operations. Figure 5 gives an example of this problem: there is a square divided in regions R1 through R6. If we perform a roll-up to aggregate the number of trajectories in regions R4, R5, and R6, we would obtain a total of six trajectories (resulting from adding three trajectories in R4, two in R5, and one in R6), while the correct number to obtain would have been three trajectories. Braz et al. (2007) address this problem.

Figure 5. The double counting problem.

A relevant feature of the TDW proposal is the treatment given to the ETL process, that transforms the raw location data and loads it to the trajectory data warehouse. The design of this process is aimed at minimizing the amount of memory needed to load and transform raw data into trajectory data, and also to address the double counting problem described above. Orlando et al. (2007) define a model supporting two equivalent forms of trajectory representation: (a) the standard \((O_{id},x,y,t)\); (b) an alternative representation where coordinates in the trajectory database are replaced by cell identifiers that cover the \((x,y)\) points. In this case, the tuples in the trajectory database are of the form \((O_{id},Cell_{id},t)\). In addition, other information of interest could be recorded, like, for instance, signal strength.

Initially, raw location data (usually arriving as a continuous data stream) is transformed into trajectory data. In other words, this step is aimed at determining the starting and ending points of a trajectory. The solution consists in splitting the bulk data according to certain assumptions. For example: (a) Temporal gap (maximum time gap between two points in the same trajectory); (b) Spatial gap (maximum...
spatial distance between two points); (c) Maximum speed (used to detect noise); (d) Maximum noise duration (if there is a long sequence of noisy observations, a new trajectory is generated); (e) Tolerance distance D (if two observations are closer than a certain distance D, the latest one is considered redundant).

OLAP operations require aggregation of TDW measures over the set of cells. In the sequel we consider the following measures: (a) Number of trajectories in a cell, denoted presence \( C_{x,y}.\text{presence} \); (b) number of objects in a cell in a certain interval; (c) crossX, crossY, where crossX(Y) is the number of distinct trajectories crossing the spatial border between two cells along the horizontal (vertical) axis. The problem of double counting arises for some of these measures, like (a) above, not only during aggregation of the base data during a roll-up operation, but also in the loading phase. For example, suppose we have three consecutive observations 1, 2, and 3; further, 1 and 3 fall in the same cell, but 2 does not. When 3 arrives, the system stores a duplicate for \( C_{x,y}.\text{presence} \) (recall data come in a continuous input stream). The presence measure deserved an in-depth treatment in (Orlando et al., 2007), where the problem of multiple counting is addressed, and some strategies for approximating the results of computing the pre-aggregated facts were proposed. For instance, linear interpolation is used to prevent omitting in the result the cells crossed by a trajectory but such that no sampling occurred within them. Finally, two alternative functions for computing the aggregate presence are defined and compared against each other: one algebraic, and one distributive. The authors borrow from statistical methods. For example, knowing the values of presence for two cells, \( C_{x,y} \) and \( C_{x+1,y} \), and defining a new cell, \( C_{x',y} = C_{x,y} \cup C_{x+1,y} \), the aggregate presence over the new cell, will be:

\[
C_{x',y}.\text{presence} = C_{x,y}.\text{presence} + C_{x+1,y}.\text{presence} - C_{x,y}.\text{crossX}
\]

where \( C_{x,y}.\text{crossX} \) is the number of distinct trajectories crossing the spatial border between \( C_{x,y} \) and \( C_{x+1,y} \).

Some example queries are provided in Orlando et al. (2007), and the two presence functions implemented (i.e., distributive and algebraic). It is reported that algebraic presence is more difficult to implement because it requires the combination of several aggregate functions and using non/standard SQL operations. The experiments reported showed that the distributive function (sum) quickly reaches large errors when the roll-up granularity increases. The algebraic method resulted to be more accurate.

**Querying the TDW.** The TDW can be exploited using OLAP techniques. The aggregated measures allow us to obtain, for example, the variable number of moving objects in different urban areas. From the point of view of the expressive power of the TDW proposal, considerations here are similar to the ones made when discussing
**Hermes.** The data types provide the functionality and clients can consume them. Of course, this allows any external data to participate in any query. However, again, the formal model is embedded in the data types, and the TDW appears as an application, such that queries are built on top of the former. This is reflected in the fact that the warehouse contains only aggregated information, and the MOD contains the moving point type. The following is an example of a query over the MOD, showing a temporal intersection, taken from the TDW demo website.

```sql
SELECT m.trajectory.at_period(tau_tll.d_period_sec(tau_tll.D_Timepoint_Sec(2006,11,24,7,45,0), tau_tll.D_Timepoint_Sec(2006,11,24,7,52,0))).to_string() as trajectory
FROM modtrajectories m where m.obj_id=1 and m.traj_id=87
```

Here we can see that the table in the FROM clause is MODTrajectories, which includes the moving point data type. These kinds of queries could also use the fact tables that contain aggregate data. Dimensions and fact tables could also be analyzed using any OLAP viewer.

**Exploiting the TDW.** Leonardi et al. (2010) presented T-Warehouse, an implementation supporting the concepts explained above. T-Warehouse allows analyzing trajectory data at different levels of aggregation. Figure 6 shows an example analyzing velocity and presence for traffic in the city of Milano. A triangle's base represents the measure presence, while a triangle's height represents velocity, showing the correlation of these two measures: when the presence is high, the speed is low (a large base and a small height).
Figure 6. A T-Warehouse screen showing presence and speed.

Enhancing the TDW to Support ad-hoc OLAP

In Figure 4 we show that the moving object database is obtained after a reconstruction from trajectories that are given in a streamlined fashion. This reconstruction task may vary according with the trajectory analysis requirements, which can be different for different applications. For instance, there may be a considerable difference on the semantic definition of a trajectory given by a traffic analyst and a logistics manager. This way, when analyzing a fleet of trucks moving in a city and delivering goods to various locations, a logistic manager may be interested in all different trajectories between the different delivery points, while the traffic analyst may only be interested in a single trajectory for the whole day. In the first case, the reconstruction process would result in a larger number of smaller trajectories than in the second case. In later steps of the TDW process, two different trajectory data cubes are built in order to allow users to apply OLAP techniques oriented to their purposes. Marketos & Theodoridis (2010) present an extension of the OLAP data model for TDW with the following features: (a) A flexible fact table able to answer queries considering different semantic definitions of trajectories; (b) A parameter that supports the choice of semantics for aggregation queries over trajectory data; (c) An ETL method loading raw location data in the flexible data cube; (d) OLAP techniques to support the different visions explained above. The proposal is denoted ad-hoc OLAP.

Figure 7, taken from (Marketos & Theodoridis, 2010) illustrates different possible scenarios. Figure 7a shows a raw dataset of timestamped locations. Different analytical needs may result to different set of reconstructed trajectories. Figures 7b and 7c illustrate the reconstructed trajectories for the logistic manager and for the traffic manager, respectively, while Figure 7d considers a compressed trajectory of the movement. The approach of building a TDW for each set of reconstructed trajectories, explained in the previous section, requires to repeatedly execute an ETL process to build different trajectory data cubes. The proposal of Marketos & Theodoridis is aimed at avoiding this.
THE PIET FRAMEWORK

The Piet data model (http://piet.exp.dc.uba.ar/piet) was introduced in Escribano, Gomez, Kuijpers, and Vaisman (2007) and Gómez, Haesevoets, Kuijpers, and Vaisman (2009). The core idea is the integration of spatial, spatio-temporal, and non-spatial data in a single framework, oriented to solve many of the problems discussed in Section “Data Warehousing and OLAP”.

The model defines a GIS dimension as composed of a set of graphs, each one describing a set of geometries in a thematic layer. A GIS dimension is, as usual in databases, composed of a schema and instances. Figure 8 shows the schema of a GIS dimension: the bottom level of each hierarchy, denoted the Algebraic part, contains the infinite points in a layer, and could be described by means of linear algebraic equalities and inequalities (Paredaens, Kuper, & Libkin, 2000). Above this part there is the Geometric part, which stores the identifiers of the geometric elements of the GIS, and is used to solve the geometric part of a query. Each point in the Algebraic part may correspond to one or more elements in the Geometric part (e.g., if more than one polylines intersect with each other). Thus, at the GIS dimension instance level we will have rollup relations (denoted $r_{L}^{\text{geom1} \rightarrow \text{geom2}}$). For instance, $r_{\text{Point} \rightarrow \text{Poly}}^{\text{city}}(x, y, pg_1)$ says that, in a layer $L_{\text{city}}$, a point $(x,y)$ corresponds to a polygon identified by $pg_1$ in the Geometric part. The authors propose a mechanism to precompute the overlayed layers in the map, that turns these relations back into rollup functions, i.e., where a point $(x,y)$ corresponds to exactly one geometry identifier. Finally, there is the OLAP part for storing non-spatial data. This part contains the conventional OLAP structures, as defined in (Hurtado, Mendelzon, &
Vaisman, 1999). The levels in the geometric part are associated to the OLAP part via a function, denoted $\alpha_{\text{dimLevel}\rightarrow \text{geom}}$. For instance, $\alpha_{\text{riverId}\rightarrow g_r}$ associates information about a river in the OLAP part (riverId) in a dimension $\text{Rivers}$, to the identifier of a polyline $(g_r)$ in a layer denoted $L_r$, which represents rivers in the Geometric part.

**Example 1.** Figure 8 shows a GIS dimension schema with three layers, for rivers, volcanoes, and states, respectively. The schema is composed of three graphs. For example, the graph for rivers, contains edges saying that a point $(x,y)$ in the algebraic part relates to line identifiers in the geometric part, and that, in the same portion of the dimension, lines relate to polyline identifiers. In the OLAP part there are two dimensions, representing states and rivers, associated to the corresponding graphs, as the figure shows. This way, a river identifier at the bottom layer of the dimension representing rivers in the OLAP part, is mapped to the polyline level in the geometric part in the graph representing the structure of the rivers layer.

Figure 9 shows a portion of a GIS dimension instance for the rivers layer in the dimension schema of Figure 8. We can see that an instance of a GIS dimension in the OLAP part is associated (via the $\alpha$ function) to the polyline $p_l$, which corresponds to the Colorado river. For clarity, only four different points are shown, at the point level $(x_1,y_1) \ldots (x_4,y_4)$. There is a relation $r_{\text{point},\text{line}}^{L_r}$ containing the association of points to the lines in the line level. Analogously, there is also a relation $r_{\text{line},\text{polyline}}^{L_r}$ between the line and polyline levels, in the same layer.

Time in the OLAP part is represented by a $\text{Time}$ dimension (there could be more than one Time dimension, supporting different notions...
of time). As it is well-known in OLAP, this dimension may have different configurations that depend on the application at hand.

![Figure 9. A GIS dimension instance for Figure 4.](image)

**Measures and Facts.** A key point in the Piet model is the way it accounts for measures and fact tables. Most of the proposals discussed in previous sections consider spatial measures, and apply OLAP operators over them. Piet is capable of working in this way, operating over the GIS dimensions (the authors define the concept of spatial aggregation for this), but also to use facts defined in the OLAP part, to support spatial DSS queries. Thus, elements in the geometric part are associated with facts, each fact being quantified by one or more measures, not necessarily a numeric value. The following example gives the intuition of a so-called GIS fact table. For details, we refer the reader to (Gómez et al., 2009).

**Example 2.** Consider a fact table containing state populations. Also assume that this information is stored at the polygon level. In this case, the fact table schema would be \((\text{polyId}, L, \text{population})\) where polyId is the polygon identifier, represents the states layer, and population is the measure. If information about, for example, temperature data, is stored at the point level, we would have a base fact table with schema \((\text{point}, L, \text{temperature})\), with instances of the form \((x_1, y_1, L, 25)\). Note that temporal information could be also stored in these fact tables, by simply adding the time dimension to the fact table. This would allow storing temperature information across time.

Example 2 shows that a GIS fact table is basically a standard OLAP fact table where one of the dimensions is composed of geometric objects in a layer. Classical fact tables in the OLAP part, defined in terms of the OLAP dimension schemas can also exist. For instance,
instead of storing the population associated to a polygon identifier, as in Example 2, this information may reside in a data warehouse, with schema (state, population).

**Geometric Aggregation**

Based on the data model described above, the notion of geometric aggregation is defined in Piet. In general, geometric aggregation queries are hard to evaluate because they require the computation of a double integral representing the area where some condition is satisfied. Thus, Piet addresses a class of queries denoted summable, of the form: \( \sum_{g \in S} h(g) \), where \( h \) is a function (represented, for instance, by a fact table), and the sum is performed over all the identifiers of the objects that satisfy a condition. For example, the query “total population of the cities crossed by the Colorado River” would read (here we assume there is a layer for cities):

\[
Q = \sum_{g_{id} \in C} ft_{pop}(g_{id}, L_c).
\]

\[
C = \{ g_{id} \mid (\exists x)(\exists y)(\exists pl_1)(\exists c \in dom(Ci)) (\alpha_{L_r,Rivers}^{ri \rightarrow pi}(\text{‘Colorado’}) = (pl_1) \land r_{L_r}^{pi \rightarrow pi}(x, y, pl_1) \land \alpha_{L_c,Districts}^{c}(c) = g_{id} \land r_{L_c}^{pi \rightarrow pi}(x, y, g_{id})) \}.
\]

The meaning of the query is: \( \alpha_{L_r,Rivers}^{ri \rightarrow pi}(\text{‘Colorado’}) \) maps the identifier of the Colorado river to a polyline in layer \( L_r \) (representing rivers). The relation \( r_{L_r}^{pi \rightarrow pi}(x, y, pl_1) \) contains the mapping between the points and the polylines representing the rivers that satisfy the condition. The other functions are analogous. Thus, the identifiers of the geometric elements that satisfy both conditions can be retrieved, and the sum of \( ft_{pop} \) (which represents the population associated to a polygon) over these objects can be performed.

**Piet-QL**

Piet comes equipped with a query language, Piet-QL (Gómez, Vaisman, & Zich, 2008), supporting the following kinds of queries: (a) pure GIS queries; (b) pure OLAP queries; (c) GIS queries filtered with aggregation (i.e., filtered using a data cube); (d) OLAP queries filtered using a geometric or geographic condition. Piet-QL also allows to place constraints over a data cube, including pre-aggregated facts into the WHERE clause. A typical example of a Piet-QL query (of the class C above) is: “Names of cities with sales, in provinces crossed by Dyle river, such that the cities had sales greater than 5000 units.”

```
SELECT GIS lcl.name
```
FROM bel_city lc1, bel_prov lp2, bel_river lr2
WHERE contains(lp2, lc1) AND
intersects(lp2, lr2) AND lr2.name="Dijle"
AND lc1 IN(
    SELECT CUBE
        filter([Store].[Store City].Members,
            [Measures].[Unit Sales]>5000)
    FROM [Sales])
    AND lp2 IN(
        SELECT CUBE
            filter([Store].[Store City].Members,
                [Measures].[Unit Sales]>0)
        FROM [Sales])

If we consider the classification proposed by Peleikis et.al. (2004), attribute, point, range, distance-based, nearest neighbor and topological queries are supported by Piet-QL. Note that these queries could be used to build the other ones, that include aggregation and OLAP capabilities. According with the classification of Vaisman and Zimányi (2009), Piet-QL falls in the SOLAP class.

Figure 10 shows a screen of the Piet implementation. We can see a Piet-QL query of type (c), and the result represented as a map layer, with the districts satisfying the query represented in green.
Figure 10. The carrier sets of a point, a polyline and a polygon are the dotted lines.

Overlay Pre-computation in Piet. Many interesting queries in GIS require computing intersections, unions, etc., of objects that are in different layers. Hereto, their overlay has to be computed. For the summable queries defined above, on-the-fly computation of the sets “C” would be costly, mainly because most of the time we will need to go down to the Algebraic part of the system to compute the intersection between the geometries. In addition to the typical R-tree-based techniques commented in previous sections, Piet implements a different strategy for materialization, consisting in three steps: (a) partitioning each layer in sub-geometries according to the carrier lines defined by the geometries in each layer (see below); this allows detecting which geographic regions are common to the layers involved; (b) pre-computing the overlay operation; (c) evaluating the queries using the layer containing all the pre-computed sub-geometries.

Figure 11. The carrier sets of a point, a polyline and a polygon are the dotted lines.

The carrier set of a layer (denoted $C_L$) induces a partition of the plane into open convex polygons, open line segments and points. Thus, the rollup relations $r$ will turn into functions (given that no two points can map to the same open convex polygon). Given $C_L$ and a bounding box, we denote the convex polygonization of $L$, the set of open convex polygons, open line segments, and points induced by $C_L$ that are strictly inside the bounding box. Given two layers $L_1$ and $L_2$, and their carrier sets $C_{L1}$ and $C_{L2}$, the common sub-polygonization of $L_1$ according to $L_2$, denoted $CSP(L_1, L_2)$ is a refinement of the convex polygonization of $L_1$, computed by partitioning each open convex polygon and each open line segment in it along the carriers of $C_{L2}$. This can be generalized for more than two layers. Figure 11 illustrates the carrier sets of a point, a polyline and a polygon.

Experimental evaluation showed that overlay pre-computation (i.e., pre-computing the common sub-polygonization) in general can perform better than R-trees, and also be competitive with aR-trees, except when
the query region must be computed in running time, because computing the intersection between the query region and the common sub-polygonization, turns out to be expensive in some situations (Escribano et al., 2007).

## QUERYING AND MINING TRAJECTORY DATA

Moving objects can be integrated in the Piet framework, by means of a distinguished fact table that we denote *Moving Object Fact Table* (MOFT).

Figure 12 (left) shows a (very) simplified map of Paris, containing two hotels, denoted Hotel 1 and Hotel 2 (H1 and H2 from here on), the Louvre and the Eiffel tower. We consider three moving objects, O1, O2 and O3. Object O1 goes from H1 to the Louvre, the Eiffel tower, spends just a few minutes there, and returns to the hotel. Object O2 goes from H2 to the Louvre, the Eiffel tower, (spending a couple of hours visiting each place), and returns to the hotel. Object O3 leaves H2 to the Eiffel tower, visits the place, and returns to H2. Figure 12 (center) shows part of these trajectory samples. All points of the same trajectory are temporally ordered and stored together (i.e., the raw trajectories table is sorted by \( O_{id} \) and \( t \)). In what follows, we will use the object identifier as the trajectory identifier, unless specified, although it is usual to generate a trajectory identifier in a pre-processing step, as explained in the TDW section.

In this scenario, a GIS user may be interested in queries like “number of persons going from H1 to the Louvre and then to the Eiffel tower (stopping to visit both places) in the same day”. Also, a data mining analyst may want to identify interesting patterns in the trajectory data using association rule mining or sequential patterns algorithms, like “people do not visit two museums in the same day”. Complex queries that aggregate non-spatial information, and also involve GIS and moving object data, must also be addressed. For instance, “total sales in museum shops, for museums located on the left bank of the Seine, such that people visit them before going to the Eiffel Tower in the same day”.

A moving object fact table (MOFT for short, see the table in the center of Figure 12), contains a finite number of identified trajectories. Definition 1 formalizes this.

**Definition 1 (Moving Object Fact Table)** Given a finite set \( \mathcal{T} \) of trajectories, a *Moving Object Fact Table* (MOFT) for \( \mathcal{T} \) is a relation
with schema $<O_{id}, T, X,Y>$, where $O_{id}$ is the identifier of the moving object, $T$ represents time instants, and $X$ and $Y$ represent the spatial coordinates of the objects. An instance $\mathcal{M}$ of the above schema contains a finite number of tuples of the form $<O_{id}, t, x,y>$ that represent the position $(x,y)$ of the object $O_{id}$ at instant $t$, for the trajectories in $\mathcal{T}$.

In practice, the MOFTs can contain huge amounts of data. For instance, suppose a GPS takes observations of daily movements of one thousand people, every ten seconds, during one month. This gives a MOFT of $1000 \times 360 \times 24 \times 30 = 259,200,000$ records. In this scenario, querying trajectory data may become extremely expensive. Note that a MOFT only provides the position of objects at a given instant. Sometimes we are not interested in such level of detail, but we look for more aggregated information instead. For example, we may want to know how many people go from a hotel to a museum on weekdays. Or, we can even want to perform data mining tasks like inferring trajectory patterns that are hidden in the MOFT. These tasks require semantic information, not present in the MOFT. In the best case, obtaining this information from that table will be expensive, because it would imply a join between this table and the spatial data.

Gómez, Kuijpers, & Vaisman (2008b) present an in-depth study on how moving object data analysis can benefit from replacing raw trajectory data by a sequence of stops and moves. The authors propose to use the notion of stops and moves in order to obtain a concise MOFT, that can represent the trajectory in terms of places of interest, characterized as stops. This table cannot replace the whole information provided by the MOFT, but allows to quickly obtain information of interest without accessing the complete data set. In this sense, this concise MOFT, which we will denote SM-MOFT, behaves like a summarized materialized view of the MOFT. The SM-MOFT will contain the object identifier, the identifier of the geometries representing the Stops, and the interval $[t_s,t_f]$ of the stop duration. Obviously, we do not need to store the information about the moves, which remains implicit, because we know that between two stops there could only be a move. Definition 2 formalizes the above.

**Definition 2 (SM-MOFT)** Let the set $\mathcal{P}_A = \{C_1 = (R_{C_1}, \Delta_{C_1}), \ldots, C_N = (R_{C_N}, \Delta_{C_N})\}$ be the PoIs of an application, and let $\mathcal{M}$ be a MOFT. The **SM-MOFT** $\mathcal{M}^{sm}$ of $\mathcal{M}$ with respect to $\mathcal{P}_A$ consists of the tuples $(O_{id}, g_{id}, t_s, t_f)$ such that (a) $O_{id}$ is the identifier of a trajectory in $\mathcal{M}$; (b) $g_{id}$ is the identifier of the geometry of a PoI $C_k = (R_{C_k}, \Delta_{C_k})$ in $\mathcal{P}_A$, such that the trajectory with identifier $O_{id}$ in $\mathcal{M}$ has a stop in this PoI during the time interval $[t_s,t_f]$. This interval is called the *stop interval* of this stop.
The table in Figure 12 (right) shows the SM-MOFT for our example of the beginning of this section. Note that we only need to represent stops, since between two stops, a move is implicitly represented. For example, object O1 is at H1 between instants 1 and 10, and at L between 20 and 30. That means, in the interval [11,19] a move is assumed. Also note that since O1 just spent some minutes at E, the Eiffel tower is not a stop for O1.

The SM-MOFT is a way of implementing the notion of semantic similarity of trajectories, which can be used to discover trajectory patterns in an efficient way, as we show below. Figure 13 shows three trajectories that, although different if we consider the positions of the three objects, from the point of view of an application could be considered similar, since the three objects traverse the same kinds of places (identified by coloured circles).

**Figure 12.** Three trajectories (left), the MOFT (center), and the SM-MOFT (right)
Figure 13. Three semantically similar trajectories.

Trajectory Mining with RE-SPaM

Gómez and Vaisman (2009) introduce a pattern language, denoted RE-SPaM, based on regular expressions, and aimed at obtaining sequential patterns in trajectory databases using the notion of semantic trajectory. Four key features characterize RE-SPaM:

1. The items to be mined are not only composed of identifiers of stops, but are also complex objects, composed of attributes that can be organized in hierarchies. This allows adding OLAP capabilities to the language in a very natural way, and is a clear difference of classic sequential pattern mining algorithms, like SPIRIT, introduced by Garofalakis et al. (1999).

2. The support of rollup functions allows performing mining at different levels of aggregation. Thus, complex sequential patterns can be found, at different granularity levels.

3. It can be proved that RE-SPaM is actually a subset of the first-order language introduced in Section “Geometric Aggregation” extended to support moving objects.

4. As a consequence of the above, not only semantic trajectories are supported, but also, if necessary, one can go back to the base data, in order to support any kind of queries, for instance, most of the ten queries in the benchmark proposed by Theodoridis (2003) (in fact, aggregation is not considered in such benchmark).
RE-SPaM could be used as a query language over a trajectory database, or to prune the patterns obtained during the mining process. Used in the latter way, a RE-SPaM expression is used during the typical sequential pattern discovery technique to indicate which are the patterns that the user wants to obtain. Therefore, during mining, all candidates that do not verify the pattern are pruned early in the process.

The RE-SPaM data model is basically composed of category schemas, category occurrences, category instances, and the table of items (ToI). For example, our tourist application above can include four category schemas, namely hotels, restaurants, airports and tourist attractions. Each category schema is composed of a set of attributes that describe it. An element in a category is denoted a category occurrence, and the set of all occurrences in all categories in an application is denoted a category instance. A set of category instances for our running example is shown in Figure 14, where, for example, the category hotels has two occurrences. A value of the attribute geom represents the geometric extension of the corresponding category occurrence (e.g., in the first tuple, pol1 can be Point(10 20)). Adding a time interval to a category occurrence, produces an item. The time interval of an item is described by its initial and final instants, and denoted [ts, tf]. A pair (Oid, item) is a tuple in the ToI. For the same Oid the time-ordered sequence of items represent the semantic trajectory of the object. Figure 15 shows a normalized instance of the ToI corresponding to the category instances of Figure 14. Normalization arises from the fact that the table contains only the Oid of the objects, plus their the category occurrence identifier, and the temporal attributes. All other attributes are stored elsewhere. The figure shows two moving objects, $O_1$ and $O_2$; Over this model, a pattern language based on regular expressions is built. The atoms in RE-SPaM are constraints expressed as formulas over attributes of the complex items defined above. Constraints consist in conjunctions of expressions, enclosed in

<table>
<thead>
<tr>
<th>Category</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>hotels</td>
<td>{(ID, H1), (categoryName, hotel), (geom, pol1), (star, 9)}</td>
</tr>
<tr>
<td></td>
<td>{(ID, H2), (categoryName, hotel), (geom, pol2), (star, 6)}</td>
</tr>
<tr>
<td>restaurants</td>
<td>{(ID, R1), (categoryName, restaurant), (geom, pol3), (typeOfFood, French), (price, cheap)}</td>
</tr>
<tr>
<td></td>
<td>{(ID, R2), (categoryName, restaurant), (geom, pol4), (typeOfFood, French), (price, expensive)}</td>
</tr>
<tr>
<td></td>
<td>{(ID, R3), (categoryName, restaurant), (geom, pol5), (typeOfFood, Italian), (price, cheap)}</td>
</tr>
<tr>
<td>airports</td>
<td>{(ID, A1), (categoryName, airport), (geom, pol6), (type, International)}</td>
</tr>
<tr>
<td></td>
<td>{(ID, A2), (categoryName, airport), (geom, pol7), (type, Local)}</td>
</tr>
<tr>
<td></td>
<td>{(ID, A3), (categoryName, airport), (geom, pol8), (type, International)}</td>
</tr>
<tr>
<td>attractions</td>
<td>{(ID, C1), (categoryName, touristattraction), (geom, pol9), (name, Cathedral of O.L.), (price, free)}</td>
</tr>
<tr>
<td></td>
<td>{(ID, C2), (categoryName, touristattraction), (geom, pol10), (name, Castle of the D.), (price, free)}</td>
</tr>
</tbody>
</table>
Figure 14. Category instances.

<table>
<thead>
<tr>
<th>OID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>{[ts,04/08/2008 14:05], (tf, 04/08/2008 14:33), [ID,R2])}</td>
</tr>
<tr>
<td></td>
<td>{[ts,04/08/2008 17:30], (tf,04/08/2008 18:48), [ID,R3])}</td>
</tr>
<tr>
<td></td>
<td>{[ts,08/08/2008 06:22], (tf,08/08/2008 07:05), [ID,R1])}</td>
</tr>
<tr>
<td>O2</td>
<td>{[ts,19/08/2008 09:00], (tf,19/08/2008 10:20), [ID,R1])}</td>
</tr>
<tr>
<td></td>
<td>{[ts,19/08/2008 17:00], (tf,19/08/2008 18:12), [ID,R2])}</td>
</tr>
</tbody>
</table>

Figure 15. A normalized Tol

 squared brackets. The regular expression language is built in the usual way, supporting the standard operators ('()','*','+','?','.','|'). The language also supports variables (strings preceded by '@'). A pattern expressing trajectories of tourists who visit hotel H1 and then a place characterized as 'cheap' or that serves French food, reads:

[ID="H1"].([price="heap"]|typeOfFood="French")

Note that the second constraint does not mention any ID, only categorical attributes. The disjunction is evaluated as follows: 'cheap' places are restaurants R1 and R3. Places that serve French food are R1 and R2. During the mining process, the items which satisfy these conditions are computed, without the need of explicit enumeration of all the possibilities, allowing writing concise expressions.

Functions in RE-SPaM can be defined ad-hoc, and are of the forms 

\[
\text{functionName}(\text{attr}, \ldots) = \text{constant}, \text{ and functionName}(\text{attr}, \ldots) = \text{variable}.
\]

Syntactically, the first parameter can be an attribute of a category occurrence (for example, typeOfFood in the tourist application example), or a temporal attribute. All other parameters must be literals, and the function also returns a literal. For example, a function compares (\text{price}, c), compares the attribute price with a literal, and returns 'equal', 'less', or 'greater than'; the first parameter is an attribute of the category occurrences of restaurants and tourist attractions, and the second one is a constant. The function can be invoked as compares (\text{price}, “100”). Also rollup functions à la OLAP can be defined to return ranges of time for a temporal attribute of an item (e.g., “Early Morning”, “Morning”,..). Below we show other examples of RE-SPaM queries.

35
Q1. “Trajectories of tourists who visit hotel H1, then optionally stop at restaurant R3 and the Zoo, and either end at H1 or visiting the Eiffel Tower”

[ID="H1"][ID="R3"][ID="Z"][ID="E"] | ID="H1"]

Q2. “Trajectories that visit hotel H1, then, optionally visit different places, and finish at the Eiffel Tower or going back to H1.”

[ID="H1"][ID="E"] | ID="H1"]

Here, the empty condition allows avoiding enumeration of the items. The next queries show the use of variables.

Q3. “Trajectories that start at a place characterized by price, then stop either at the zoo or the Eiffel Tower, and end up going to a place that serves French food, and has the same price range as the initial stop.”

[price=@x].( [ID="Z"] | [ID="E"] ).[typeOfFood="French" and price=@x]

Finally, Q4 below shows the use of a rollup function in a constraint.

Q4. “Trajectories that stopped at two places (the second one having cheap prices), at the same part of the day (e.g., both of them during the morning), on 10/10/2008”

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

RE-SPAM++: Extending RE-SPaM to Support Geographic Information

Since objects move in a geographic environment, it is important to allow geometric conditions to be included in the patterns. Therefore, RE-SPAM was extended with this capability by Gardella et al. (Gardella, Gómez & Vaisman, 2010). This extension is denoted RE-SPaM++. The language includes SOLAP conditions in RE-SPaM constraints using Piet-QL, the query language supporting Piet, accomplishing the goal of integrating SOLAP and moving object data in a single framework. Syntactically, the extension is very simple: a
WITH statement is added to the Piet-QL SELECT clause. This statement generates a sort of materialized view that is used in a RE-SPaM expression. Thus, the language allows not only single statements but also programs comprising sequences of Piet-QL and RE-SPaM statements. Next, we show an example of a RE-SPaM++ query.

Q5. “Trajectories that stop at a place which belongs to a region that contains a river, and whose next stop is an airport or a tourist attraction.” The query reads in RE-SPaM++:

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

[containedBy(geom,regRiver.the_geom)='true'].
([categoryName='Airport']|[categoryName='Tourist Attraction'])
```

The Piet-QL part of the query returns a set of geometric objects (polygons) representing regions containing rivers, in the cursor regRiver(the_geom). In the RE-SPaM part of the query, the first constraint checks if the PoI is contained in one of the regions in the set. In other words, when an item in the Table of Items is being evaluated (e.g., during the mining process or just using RE-SPaM++ as a query language), the corresponding PoI geometry (represented by the attribute geom) is compared against each one of the geometric elements in the cursor.

Figure 16 shows a screenshot of the implementation of RE-SPaM++, and Figure 17 depicts the result, where we can see trajectories of lengths 1 to 4 satisfying the pattern. Figure 18 shows in more detail a trajectory of length two over real trajectories verifying a pattern.
Figure 16. The RE-SPaM query interface
HERMES AND PIET: SIMILARITIES AND DIFFERENCES

We now compare the two proposals that, in our opinion, more comprehensively address the issue of spatio-temporal OLAP: Hermes-GeoPKDD, and Piet. The other proposals discussed in sections “Spatial Data Warehousing and OLAP” and “Spatio-Temporal Data Warehousing, OLAP and Mining” address different parts of the problem, but no spatio-temporal OLAP as a whole. However, and even though there exists some degree of overlapping, both approaches tackle different parts of the SOLAP problem. The analysis is performed in terms of the capabilities to fulfill SOLAP requirements.

Hermes does not specifically address SOLAP support. It is left open (although not explicitly stated) as a possible application of the general framework, but no formal model supports spatial data aggregation, probably because Hermes has been designed as an architecture to support spatio-temporal data, not as a model for spatio-temporal decision support. On the other hand, Piet focuses on GIS-OLAP integration, and is oriented specifically toward aggregate queries and spatial decision-support, although, as showed, standard spatial queries are also supported.

Integrating geographic data and warehouse data is not built into the Hermes model, while Piet handles this integration through the “α” function. A tool to semi-automatically match geometric elements in the
GIS layers to non-spatial objects in the warehouse was implemented for Piet. On the Hermes side, integration of an external warehouse would require defining, in an ad-hoc fashion, how geographic objects will be mapped to warehouse objects.

Being conceived as a SOLAP system, the Piet formal data model also integrates naturally into the SOLAP framework the problem of modeling and analyzing trajectory data, either using the whole trajectory data (i.e., the MOFT), or the semantic trajectory represented via the SM-MOFT.

Probably the strongest point of the Hermes proposal is the analysis and implementations of the ETL process for trajectory data analysis. On the other hand, Piet does not have similar automatic data loading machinery, and assumes that data has already been loaded into a (continuous) trajectory file.

The approaches of Hermes and Piet for building a trajectory data warehouse (TDW) are quite different. Hermes loads only aggregate measures into a fact table, and dimensions conform cells in a three-dimensional space \((x, y, t)\). The main achievement, in this sense, is the treatment of double counting for some of the measures. Trajectory data in the GeoPKDD proposal is stored in the moving object database (MOD), and it is used to extract higher level knowledge that may also be used to feed the TDW. In this sense, the TDW could be considered an application, developed over the underlying architecture that uses the traditional star schema. Instead, in the Piet/RE-SpaM (Piet’s regular query language for trajectories) approach, the MOFT and SM-MOFT do not store aggregated measures (actually, the “facts” here are represented by the existence of the trajectory in the database), but just the base trajectories (or the “semantic” trajectories, in the SM-MOFT). Actually, the MOFT is, basically, the RELTrajectories table in the TDW approach. Aggregation over the ‘cells’ hierarchy could be supported by RE-SPaM, although the language is mainly oriented to trajectory pattern mining. Further, aggregation is performed over trajectories that satisfy a certain pattern – see the work by Gómez, Kuijpers, & Vaisman (2008a) for details on the different aggregate operators and their arguments. In summary, implementing aggregation over cells in Piet in the way proposed in GeoPKDD would not be trivial.

Hermes-MDC is based on moving objects that can change shape or position over time, while Piet assumes that the regions and geometric objects, in general, are static, and that traceable objects (e.g., representing pedestrians, buses, cars) move through the geographic space. In other words, Piet does not provide temporal support for the GIS part of the model, only for the moving objects whose trajectories are being analyzed. However, a temporal extension for Piet has been proposed (Gómez, Kuijpers & Vaisman, 2010), and it is being implemented at the time this survey is written (see below).

In summary, using the taxonomy proposed by Vaisman and Zimányi, Hermes fall in the Spatio-Temporal class, while Piet falls in the
SOLAP class. Piet and RE-SPaM together would fall in the Spatio-Temporal OLAP class, and the TDW proposal also belongs to this class.

Finally, while all Piet software components are open source: the database, postgres, and its GIS extension postGIS (http://postgis.refractions.net), the Mondrian OLAP server (http://www.mondrian.sourceforge.net), and Java. On the other hand, Hermes is built as an extension of Oracle 10g. Table 1 summarizes the comparison between Hermes, the TDW, and Piet/RE-SPaM.

<table>
<thead>
<tr>
<th></th>
<th>Hermes</th>
<th>Trajectory DW</th>
<th>Piet</th>
<th>RE-SPaM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIS-OLAP integration</td>
<td>No</td>
<td>Through Hermes-MDC</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SOLAP Formal model</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>Fact table</td>
<td>N/A (can be defined ad-hoc)</td>
<td>Pre-aggregated measures</td>
<td>External, defined in the OLAP part</td>
<td>MOFT, SM MOFT (no pre-aggregation)</td>
</tr>
<tr>
<td>Dimensions</td>
<td>N/A (can be defined ad-hoc)</td>
<td>Spatial dimensions, Time dimension</td>
<td>GIS dimension, regular OLAP dimensions</td>
<td>OLAP hierarchies integrated in the query language</td>
</tr>
<tr>
<td>Support for spatial aggregation</td>
<td>Ad-hoc</td>
<td>Yes</td>
<td>Yes</td>
<td>Partially</td>
</tr>
<tr>
<td>Support of querying an external DW</td>
<td>Ad-hoc</td>
<td>Ad-hoc</td>
<td>Provided by model &amp; query language</td>
<td>Provided by model &amp; query language</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------</td>
<td>--------</td>
<td>-----------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Spatial Queries</td>
<td>Point, Range, distance, nearest-neighbor</td>
<td>Point, Range, distance, nearest-neighbor</td>
<td>Point, Range, distance, nearest-neighbor</td>
<td>Point, Range, distance, nearest-neighbor</td>
</tr>
<tr>
<td>Mining capabilities</td>
<td>No</td>
<td>Through external functionality</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Support of changing geometric objects</td>
<td>Yes</td>
<td>Through Hermes-MDC</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Temporal support</td>
<td>Yes</td>
<td>Through Hermes-MDC</td>
<td>No</td>
<td>Only for moving points</td>
</tr>
<tr>
<td>ETL Support and tools</td>
<td>N/A</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Semantic trajectory support</td>
<td>N/A</td>
<td>Through external functionality</td>
<td>N/A</td>
<td>Built-in</td>
</tr>
<tr>
<td>Open Source Architecture</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Needs non-standard data libraries for querying?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 1. Comparing Hermes, TDW, Piet and RE-SpaM.

**FUTURE DIRECTIONS IN SPATIO-TEMPORAL OLAP**

An Emerging Topic: SOLAP for Continuous Fields

Continuous fields describe the distribution of physical phenomena that change continuously in time and/or space. Examples of such phenomena are temperature, pressure, and land elevation. Besides physical geography, continuous fields (from now on, fields), like land use and population density, are used in human geography as an aid in spatial decision making process. Some work has been done to support querying fields in GIS, although the area of spatial multidimensional analysis of continuous data is still almost unexplored. Integrating spatiotemporal continuity within multidimensional structures poses numerous challenges. Further, existing multidimensional structures and models dealing with discrete data, are not adequate for the analysis of continuous phenomena. Multidimensional models and associated query
languages are thus needed, to support continuous data.

In a seminal work on the topic, Tomlin (1990) proposes an algebra for fields, denoted map algebra, based on the notion that a map is used to represent a continuous variable (e.g., temperature). There are three types of functions in Map algebra: local, focal, and zonal. Local functions compute a value at a certain location as a function of the value(s) at this location in other map layer(s), allowing queries like “Total desert land in a country, where a region is classified as desert if the annual rain is less than 500 mm per year”. Focal functions compute each location's value as a function of existing values in the neighboring locations of existing layers, allowing aggregate queries like “Local altitude in clay soil regions, in a map containing soils distribution in some portion of land”. Zonal functions compute a location's new value from one layer (containing the values for a variable), associated to the zone (in another map) containing the location, supporting queries like “Total area in a province, with elevation greater than 1000 m above sea level”. Câmara et al (2005) and Cordeiro et. al. (2005) formalize and extend these functions, supporting more topological predicates. Mennis et al. (2005) extend map algebra operators to support time-varying fields.

In spite of the above, regarding fields and multidimensional models, the joint contribution of the GIS and OLAP has been limited. Shanmugasundaram et. al (1999) propose a data cube representation that deals with continuous dimensions not needing a predefined discrete hierarchy. They focus on using the known data density to calculate aggregate queries without accessing the data. The representation reduces the storage requirements, but continuity is addressed in a limited way. Ahmed and Miquel. (2005) use interpolation methods to estimate (continuous) values for dimension levels and measures, based on existing sample data values. Continuous cube cells are computed on-the-fly, producing a continuous representation of the discrete cube. A sequel of this proposal introduces SOLAP concepts, and a SOLAP application supporting some form of continuous data (Ahmed, 2008). These works are based on a data model devised for OLAP, not for spatial OLAP, leading to a representation of spatial dimensions which is not the best one. The approach of Vaisman and Zimányi (2009) is based on the Multidim conceptual model discussed above (Malinowski & Zimányi, 2008), which is extended in a natural way to support this new data type. The authors characterize multidimensional queries over
fields using the taxonomy described above in this survey, denoting this class of queries SOLAP-CF (standing for SOLAP with Continuous Fields). Along the lines of their previous work, they make use of the relational calculus supporting aggregate functions, extending it with a field data type. They also analyze the operators that this data type must include to support and extend map algebra. They also discuss different implementation choices for the abstract model, using regular gridded digital elevation models (DEM), and triangulated irregular networks (TIN) as data structures. Finally, recently Gómez et al. (2010) presented an implementation of this model over a gridded structure.

Extending Piet with Temporal Capabilities

We commented above that Piet does not support for querying the history of spatial objects, that is geometric objects are assumed to be static. This is a limitation in real-world applications, which are temporal in nature. For example, in a cadastral system, parcels can change shape, and be merged or split. That is, not only alphanumeric attributes can change, but also geometric ones. Therefore, queries like “Total production by year per square mile for each parcel of land, for the parcels in Antwerp” cannot be answered in Piet. In another scenario, a pollution stain can move forward or backward, grow and/or shrink (or even disappear), split into two or more, or be merged with other ones. Environmental control agencies monitor these changes in order to keep the situation within certain limits. Interesting information can be obtained in this scenario, as long as we have a data model and query language allowing to represent a dynamic setting. For instance, the evolution of the stain across time, how far from a school it had passed, or how many people were exposed to its effect. Addressing situations like the ones above requires extending non-temporal SOLAP data models and query languages with temporal capabilities. Gómez et al. (2010) extended the Piet data model to support spatio-temporal OLAP. The work presents a formal data model, based on the timestamp temporal model, and proposes a query language, extending Piet-QL. This model would fall into the Spatio-Temporal OLAP class in the Vaisman-Zimányi taxonomy.

Acknowledgements. This research has been partially funded by the European Union under the FP6-IST-FET program, Project n. FP6-14915, “GeoPKDD: Geographic Privacy-Aware Knowledge Discovery and Delivery”, (www.geopkdd.eu) and by the Research Foundation Flanders (FWO-Vlaanderen), Research Project G.0344.05, and the Argentinian National Scientific Agency, project PICT 2004 11-21.350.
REFERENCES


Frentzos, E., Gratsias, K., & Theodoridis, Y. (2007). Index-based most simila


