Ten Benchmark Database Queries for Location-based Services

Yannis Theodoridis [(*)]

Department of Informatics
University of Piraeus
GR-18534 Piraeus, Hellas
URL: http://thalis.cs.unipi.gr/~ytheod
E-mail: ytheod@unipi.gr

Abstract

Location-based services (*l*-services for short) compose an emerging application involving spatio-temporal databases. In this paper, we discuss this type of application, in terms of database requirements, and provide a set of ten benchmark database queries (plus two operations for loading and updating data). The list includes selection queries on stationary and moving reference objects, join queries and unary operations on trajectories of moving objects. We also survey recent work in query processing for those query types, with emphasis on indexing of moving objects, and suggest candidates for efficiently supporting databases for *l*-services.

Keywords: location-based services, moving objects, spatio-temporal databases

---

[(*)] Also with the Data and Knowledge Engineering Group, Computer Technology Institute [http://dke.cti.gr].

1. INTRODUCTION

In recent years, we have been witnessing the explosion of emerging non-traditional database applications, such as location-based services, WWW repositories, etc. Focusing on the former, the main components of the underlying database includes spatial objects, and more particularly, *stationary* and *moving* (spatial) objects. It is not an exaggeration to say that the so-called Moving Objects Databases (MODs) are (or soon will be) ubiquitous.

Figure 1 illustrates such an example, where people (shoppers) are moving around and looking for shops offering goods that match their interests. Apart from shops, other buildings are also located around (a church, etc.) and transport means (e.g. trams) running scheduled routes support shoppers' motion. Shoppers could be assisted by palmtop devices, through wireless communication channels to find their way to buildings of interest. The (hypothetical) application illustrated in Figure 1 could be such a location-based service (hereafter, called *l-service*).

Although basic research in spatiotemporal databases has already a history of more than 10 years, current releases of commercial DBMSs (Oracle 9i, IBM DB2 etc.) do not efficiently support these types of applications. The reason could either be the lack of a commonly accepted data model (see the discussion in Section 2) or the lack of a 'killer' application, as several researchers claim (Seeger et al.,

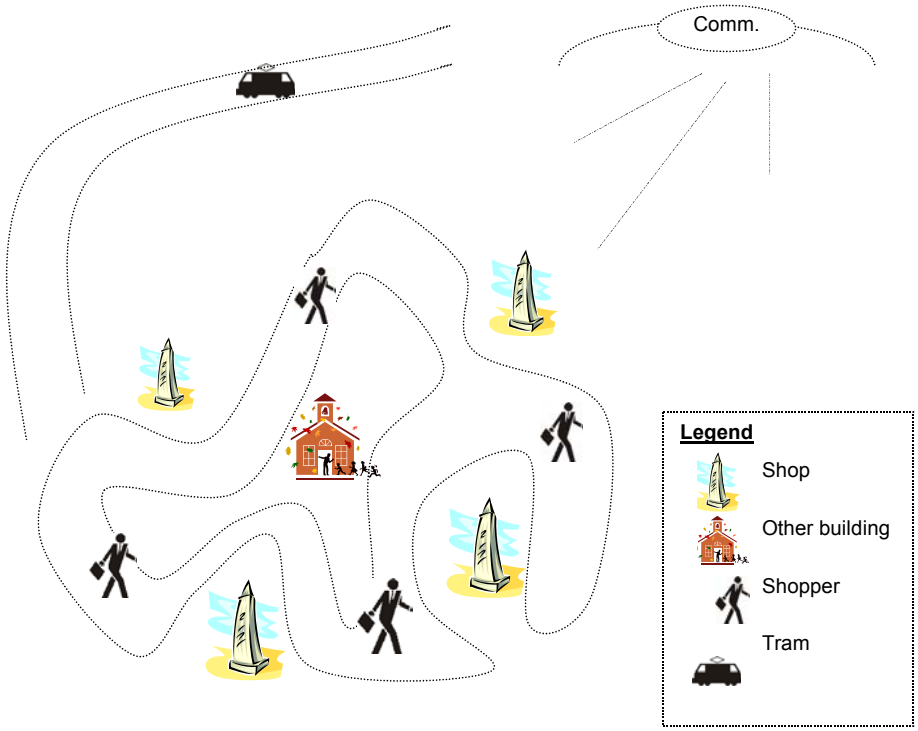


Figure 1: An *l*-service application

2001). To our opinion, it is partially both; moreover, it is due to the lack of a clear understanding of what kind of management should a MOD offer, what is the expected functionality, in terms of user interfaces, the SQL or SQL-like query language provided, query processing and optimization issues.

Recent basic research (in particular, in the European 'Chorochronos' research network (Sellis et al., 2002)) covered a wide spectrum of applications involving space and time, from continuously to discretely (i.e., elevation) moving objects or regions, to stationary objects changing other (non-spatial) information over time. In a previous paper (Theodoridis et al., 1998), we classified spatiotemporal applications as follows:

- With respect to the dataset mobility: *growing*, *evolving* or *fully-dynamic*; for those involving stationary objects of varying cardinality, moving objects of fixed cardinality, or moving objects of varying cardinality, respectively.
- With respect to the permitted time-stamp updates: *static*, *chronological* or *dynamic*; for those permitting append-only operations in the database in a batch mode (thus not affecting the past), dynamic insertions or updates of current status (however, not affecting information on past timestamps), or dynamic insertions of or updates on timestamps without time limitations (thus permitting even updates referring to the past), respectively.

However, such an umbrella covers very different applications, from urban planning to fleet management. In this paper, we claim that a distinction between applications for discretely changing, on the one hand, and (continuously or arbitrary) moving objects, on the other hand, is necessary. We also claim that, due to the lack of this distinction, the so-called spatiotemporal databases are so wide-ranging that their support by commercial DBMS becomes impractical. Instead, we should focus on specific applications with similar modeling and functionality and in this work we focus on MODs.

In the rest of the paper, we first discuss modeling issues (Section 2) and, then, present a list of benchmark queries involving moving objects (Section 3). Then, we provide guidelines for the efficient support of those queries, in terms of indexing and query processing (Section 4). In Section 5, we discuss related work and, finally, we conclude in Section 6.

2. MODELING AND QUERYING MOVING OBJECTS: A SURVEY

Due to the significant research work in spatial and temporal data modeling, early efforts in developing spatiotemporal data models were based either in the former or in the latter work. For example, Worboys (1994) generalized earlier work on spatial data and provided the first unified framework for spatiotemporal data. Cheng and Gadia (1994) presented an example of the other approach, i.e., from temporal to spatiotemporal data models. Unfortunately, both supported *discrete* changes of the locations of spatial objects, not *continuous* ones. Although discrete changes appear in some applications (e.g.

cadastral), in the vast majority of spatiotemporal applications the change of objects' locations is of the *continuous* type (location-based services, fleet management, monitoring of natural phenomena, etc.).

Under the methodology of constraint databases, a formal spatiotemporal model for discrete changes was proposed in (Grumbach et al., 1998) while Chomicki and Revesz (1999) provided a theoretical framework for continuously changing objects, with discrete change defined as a special case.

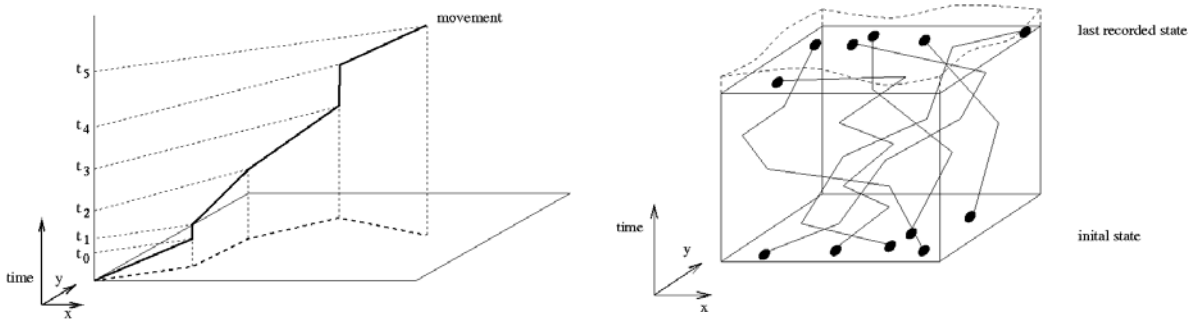
Towards a more integrated view of space and time, Güting et al. (2000) introduced the concept of *spatiotemporal data types* using the notion of Abstract Data Types (ADTs). In this work, a moving object (point, line, region) is defined as the temporal version of (static) spatial object of type α and is given by a function from *time* to α.

In the field of requirements analysis and conceptual modeling, Pfoser and Tryfona (1998) presented a set of modeling requirements motivated by cadastral and network utility management applications. Regarding spatiotemporal data, the concepts of *snapshot*, *changes*, and *versions* of objects and maps, *motion* and *phenomena*, were discussed. In (Parent et al., 1999), a spatiotemporal model, called MADS (Modeling of Application Data with Spatio-temporal features), was proposed. MADS supports both discrete and continuous changes and, moreover, allows the specification of relationships that have space- and/or time-related semantics, based on the so-called 9-intersection model (Egenhofer and Franzosa, 1991) for topological operators and Allen's temporal operators (Allen, 1983), respectively. Tryfona and Jensen (1999) extended the ER model including spatial and temporal aspects as well as their combinations (thus, the proposed model was called STER, for Spatio-Temporal ER).

SQL extensions to support spatiotemporal data have been also proposed in the past. Böhlen et al. (1998) proposed STSQL, a multi-dimensional extension to SQL-92. Actually, STSQL is not adequate to represent queries on moving objects, since it is an enhanced SQL-92 with valid and transaction time support. Recently, Erwig and Schneider (1999) proposed STQL (for Spatio-Temporal Query Language), an extended SQL exploiting the concept of ADTs.

As discussed in (Pfoser et al., 2000), the data obtained from moving point objects is similar to a "string", arbitrary oriented in 3D space, where two dimensions correspond to 2D (x-, y-) plane and one dimension corresponds to time[1]. Instead of a "string", in a MOD we store and manipulate a 3D polyline, representing the *trajectory* of the object (Figure 2).

[1] This framework can be easily extended to 4D space (3D original space + time) for applications involving objects moving above the ground (planes, birds, satellites, etc.).



(a) (b)

Figure 2: Moving objects. (a) a trajectory and (b) a collection of trajectories

Considering Figure 2, one can easily argue that handling continuously changing locations of moving objects in current DBMS technology is subject to two contradicting issues: on the one hand, current systems are not able to store and manipulate infinite sets, thus a lack of information is introduced, by default; on the other hand, monitoring systems (GPS and communication technologies) are inherently discrete, thus not able to continuously capture the location of an object. As a result of both, to obtain the entire movement, we have to *interpolate*, either using linear interpolation, which is the simplest method, or by using more complex approaches, such as polynomial splines (Bartels et al., 1987). Obviously this lack of complete information about the locations of objects introduces uncertainty, which needs to be captured in order to avoid false or missing results. For example, Pfoser and Jensen (1999) discuss the *lens* area that appears between two consecutively sampled locations at t_1 and t_2, and this area represents the "probable" locations of this object at an intermediate timestamp t ($t_1 < t < t_2$).

In the rest of the paper, we will use the *l*-service application illustrated in Figure 1, involving people looking around using their palmtops and shops offering goods (with offers being valid in specific time intervals)[2], while other buildings with no temporal characteristics would be close (restaurants, gas stations, etc.).

3. BENCHMARK DATABASE AND QUERIES

An example database for such an application consists of the following entities (and relationships among them): **Humans** include people looking around, their interests and requests on products (shoes, sportswear, etc.) as well as the routes they follow; **Buildings**, consisting of **Shops** and **Other**, store their locations (polygons) and, particularly for Shops, their time-dependent offers, i.e., **offer** is a multi-valued composite attribute consisting of traditional (e.g. product name) and temporal information (e.g.

[2] We could think of several similar applications. In the case, for example, of the Athens 2004 Olympic Games, it could be (moving) visitors trying to find (stationary) stadiums with interesting events taking place inside.

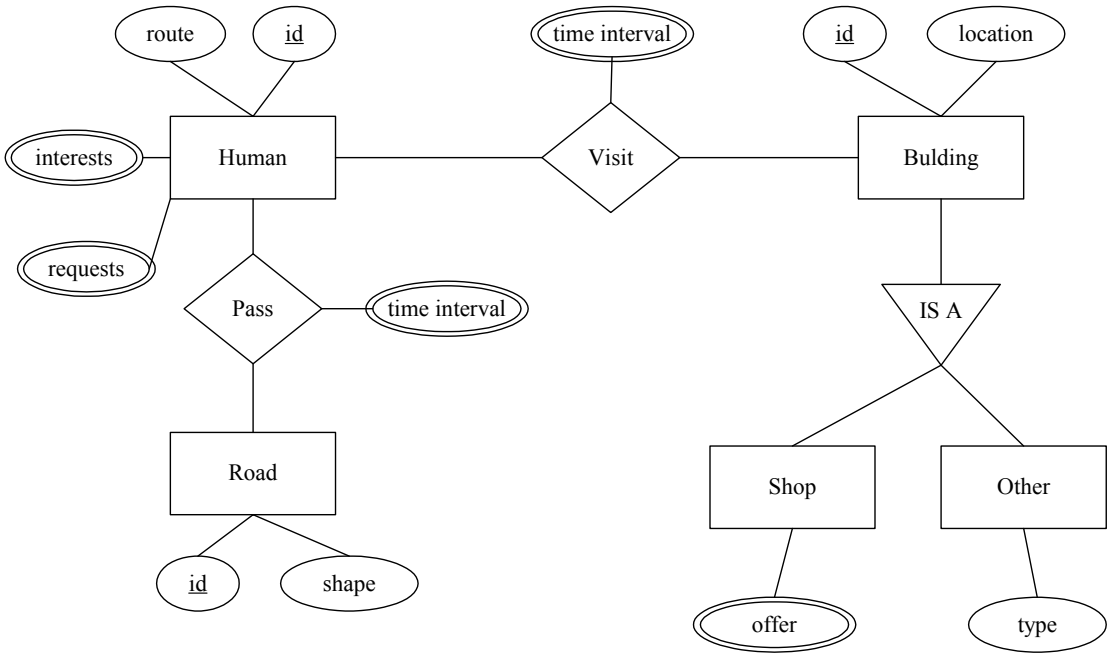


Figure 3: The ER-diagram of the benchmark database

time(s) the offer is valid); **Roads** store their shapes (polylines) and the time(s) when humans **Pass** roads or **Visit** buildings are recorded. Figure 3 illustrates the ER diagram of this database. Among the attributes that appear in this ER, the **route** of a human is of type *mpoint* (i.e. moving point) while the **location** of a building and the **shape** of a road are of (pure spatial) type *polygon* and *polyline*, respectively. In the Appendix, we provide a definition of the same database schema in ODL.

Before we start listing the queries of our benchmark, we state the functionality expected from the *mpoint* data type[3]. An illustration of those operators appears in Figure 4.

- `trajectory` (`mpoint reference`) is an operator that computes the spatial projection of *mpoint* onto the Euclidean plane; in other words, it returns a set of connected-or-not 2D line segments.
- Analogously, `life` (`mpoint reference`) is an operator that computes the temporal projection of *mpoint* onto the time axis, returning a set of connected-or-not 2D time intervals.
- `curr_space` (`mpoint reference`) is an operator that returns the current, i.e., the last recorded, information about the spatial location of *mpoint*.
- Analogously, `curr_time` (`mpoint reference`) is an operator that returns the current information about the time instance of *mpoint*.

[3] Analogous operators could be defined for *mline* or *mregion* data types, if required. However, we do not focus on those data types since we do not find them as useful as *mpoints* are in *l*-service applications. Also, we do not define similar operators, e.g. inside, on (static) spatial data types, such as point, polyline and polygon, to be used hereafter, since they are already supported by current releases of most well-known commercial DBMSs.

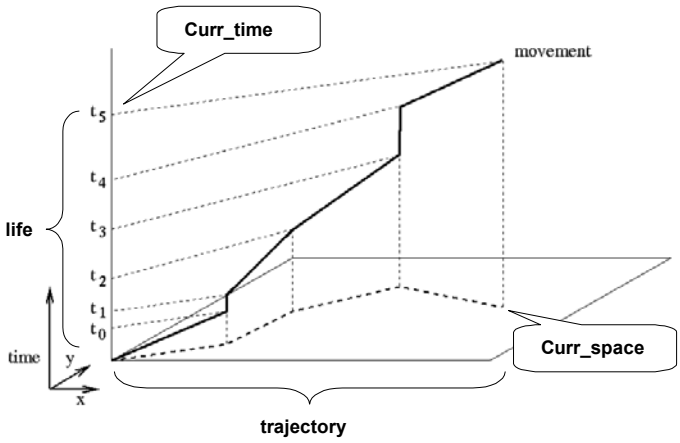


Figure 4: Operators on *mpoint* data type

In the sequel, we provide benchmark queries, both in natural language and in an 'imaginary' SQL-like language, supporting ADTs for *l*-service applications[4].

Following the examples of (Stonebraker et al., 1993; Patel et al., 1997), the first operation to be supported is the loading of the benchmark database and the building of appropriate indices (the indices required are discussed later).

Operation 0a: Initialization

"Load the benchmark database and build indices"

Moreover, an update operation should be present. The location of involved moving points should be updated either periodically or in ad hoc timestamps.

Operation 0b: Updates

"Generate a set of timestamps t_0, t_1, …, t_n; at each t_i compute updated locations of involved objects; update indices"

3.1. Queries on stationary reference objects

At least the typical point, range, distance-based and nearest-neighbor queries between moving (humans) and stationary objects (such as shops) should be present:

Query 1: point query – stationary reference object

"Are there any offers in the shop where I (i.e., id=20) am visiting now?"

[4] The SQL-like language presented here is not the focus of the paper. It is just a tool to provide a formal declaration of the proposed queries.

```
SELECT Offer.id, Offer.info
FROM Human, Shop, Offer
WHERE Offer.shop_id = Shop.id and Human.id = 20
AND curr_space(Human.route) INSIDE Shop.location;
```

Query 2: range query – stationary reference object

"Find humans located in a specific rectangular area, e.g. [0.23, 0.34, 0.85, 0.40], during 8am-2pm, Sept. 6, 2001"

```
SELECT Human.id
FROM Human
WHERE trajectory(Human.route) OVERLAP Rectangle((0.23,0.34),(0.85,0.40))
AND life(Human.route) RESTRICTED_BY Interval(2001/09/06:08.00, 2001/09/06:13.59)
TOGETHER;
```

In the above query, the role of the new operator `RESTRICTED_BY` is two-fold: first, it checks whether `life(reference)` overlaps a specific time interval `[t1,t2]`; if yes, (a) it creates a view `reference'` of `reference` such that: `life(reference')` ← `life(reference)` ∩ `[t1, t2]` and (b) returns TRUE; otherwise, it returns FALSE.

As a second modification, the `WHERE` clause has been extended by adding the predicate `TOGETHER` in order to enforce finding moving objects (`Human.route`, in our example) that simultaneously fulfill both conditions, in space and in time. In Section 4, we will discuss this peculiarity, in terms of efficient query processing (assuming a unified 3D coordinate system, for 2D space and 1D time).

Query 3: distance-based query – stationary reference object

"Find shops close (e.g. less than 500 m) to me (i.e., id=20), offering sportswear"

```
SELECT Shop.id, Offer.id, Offer.info
FROM Human, Shop, Offer
WHERE Offer.info = "sportswear"
AND Offer.shop_id = Shop.id AND Human.id = 20
AND Shop.location INSIDE Circle(curr_space(Human.route), 500);
```

Nearest-neighbor queries between trajectory data and stationary objects are also useful:

Query 4: nearest-neighbor query – stationary reference object

"Find the two nearest restaurants (i.e., type=1) to my (i.e., id=20) current location"

```
SELECT Other.id
FROM Human, Other
WHERE Other.type = 1 AND Human.id = 20
ORDER BY Spatial_Neighbor(Other.location, curr_space(Human.route)) ASC 2;
```

The SQL-like notation:

```
'ORDER BY Spatial_Neighbor(attribute, reference) ASC k'
```

adopted above is used to express k-NN queries between spatial attributes; in this example, k = 2. By definition, the results are sorted according to a norm (here, the restaurant - human distance), hence, they could be presented either by ascending (ASC) or by descending (DESC) order. A similar notation will be later used to express closest pair queries (Corral et al., 2000) and most similar trajectories queries (Kollios et al., 2002).

According to (Pfoser et al., 2000), the above queries are classified as *coordinate-based*. *Topological queries*, such as enter, cross, leave, bypass, are also useful:

Query 5: topological query – stationary reference object

"Find humans crossed Oxford St. in the morning of Sept. 6, 2001"

```
SELECT Human.id
FROM Human, Road
WHERE Road.id = "Oxford St."
AND trajectory(Human.route) CROSS Road.shape
AND life(Human.route) RESTRICTED_BY Interval(2001/09/06:09.00, 2001/09/06:11.59)
TOGETHER;
```

Again, the predicate `TOGETHER` enforces the simultaneous processing of the spatial and the temporal condition onto `Human.route`.

3.2. Queries on moving reference objects

Query 1 till Query 5 involve stationary reference objects. Variations involving moving reference objects (e.g. other trajectories) should be supported as well. In the following, we provide examples of distance-based and nearest-neighbor queries between moving objects, the so-called distance-join (Hjaltason and Samet, 1998) or closest-pair queries (Corral et al., 2000).

Query 6: distance-based query – moving reference object

"Find humans who've passed close (e.g., less than 100 m) to me (i.e., id=20) and have already requested sportswear"

```
SELECT Y.id
FROM Human X, Human Y
WHERE X.id = 20 AND "sportswear" IN Y.requests
AND Y.route INSIDE Strip(trajectory(X.route), 100);
```

`Strip(reference, k)` is the shape defined by a polyline `reference` extended by k units of measure at each side of. In other words, it is an irregular zone of width 2k with the polyline `reference` defining its centre. More about processing this query type will be discussed later, in Section 4.

Another query type we find useful is the so-called *most similar trajectories* (MST). Among a set of trajectories we ask to find those who are most similar to a reference trajectory, according to a norm (average Euclidean distance, area between, etc.). In line with k-NN queries, the SQL-like notation:

`'ORDER BY Similarity(attribute, reference, function) DESC k'`

is proposed to be used for expressing k-MST queries, where `function` is the norm to be applied.

Query 7: similarity-based query

"Find the 3 most similar trajectories to the one followed by me (i.e., id=20) on Sept. 6, 2001"

```
SELECT Y.id, Y.route
FROM Human X, Human Y
WHERE X.id = 20
AND life(X.route) RESTRICTED_BY Interval(2001/09/06:00.00, 2001/09/06:23.59)
AND life(Y.route) RESTRICTED_BY Interval(2001/09/06:00.00, 2001/09/06:23.59)
ORDER BY Similarity(trajectory(Y.route), trajectory(X.route), Distance()) DESC 3;
```

3.3. Join queries

Previous queries involved a single reference object (point, area, trajectory, etc.). Join queries are also typical in databases and join two different tables (or a table with itself; the so-called self-join). Next, we present two join queries, one based on distance and one based on similarity among trajectories.

Query 8: distance-join or closest-pair query

"Find the 5 closest shops offering shoes to humans asking for shoes as well"

```
SELECT Human.id, Shop.id, Offer.id
FROM Human, Shop, Offer
WHERE "shoes" IN Human.interests AND Offer.info = "shoes"
AND Offer.shop_id = Shop.id
ORDER BY All_Spatial_Neighbor_Pairs(Shop.location, curr_space(Human.route)) ASC 5;
```

Similar to `Spatial_Neighbor(attribute, reference)`, used to express k-NN queries in Query 4, the notation:

`'ORDER BY All_Spatial_Neighbor_Pairs(attribute, attribute) ASC k'`

is used to express k-CP queries.

Query 9: similarity-join query

"Find the 2 most similar pairs of trajectories in the entire database"

```
SELECT X.id, Y.id
FROM Human X, Human Y
WHERE X.id ≠ Y.id
ORDER BY Similarity(trajectory(X.route), trajectory(Y.route), Distance()) DESC 1;
```

Similar to `'Similarity(attribute, reference)'`, used to express k-MST queries in Query 7, the notation:

`'ORDER BY All_Similarity_Pairs(attribute, attribute) DESC k'`

expresses its extension, which can be called *most-similar-pairs-of-trajectories* (k-MSPT) and which, to the best of our knowledge, has not yet been studied in the spatiotemporal database literature (see relevant discussion in subsection 4.2). In the above query, k=1 since we are asking for a single pair (`X`,`Y`) of objects.

3.4. Unary operators on spatial and spatiotemporal data types

Supporting unary operators on spatial and spatiotemporal data types (e.g. the length or the duration of a trajectory, the area covered or the average speed achieved) is also a requirement for *l*-service applications:

Query 10: unary operators on spatial and spatiotemporal data types

"Find the traveled distance of myself (i.e., id=20) and my average speed in a specific time interval, from 11am to 2pm, Sept. 6, 2001"

```
SELECT id, Length(trajectory(route)), Speed(route)
FROM Human
WHERE id = 20
AND life(route) RESTRICTED_BY Interval(2001/09/06:11.00, 2001/09/06:13.59);
```

3.5. Summary

We have provided a list of ten queries to compose a benchmark for MODs supporting *l*-service applications. Of course, more queries could be added and the necessity of such an extension is a task for future work. However, based on related research work (Sellis et al., 2002) as well as DBMS vendors' white papers and data sheets, e.g. (Oracle, 2002), we consider that the above list constitutes the minimum functionality a MOD system should provide and we expect that soon coming releases of commercial DBMSs will partially cover it.

In this list we have **not** included *future queries* (i.e., queries involving anticipated future locations of moving objects). This is because the driving application of *l*-service that motivated this work is not appropriate for future queries, at least at the degree of appropriateness vehicle monitoring applications are, since the speed and the frequency of stops of humans-shoppers are usually unpredicted; a man or woman looking around, entering and leaving shops according to instantaneous attractions (e.g. contents of shop fronts) does not easily fit in the logic of "average speed", "expected direction", etc. Nevertheless, if one would require the inclusion of future queries, those dealing with present timestamps (e.g. Query 3 and Query 4) could be slightly modified to deal with future timestamps.

	Operation	Definition
Spatial Operations	**Rectangle** (point lower-left, point upper-right)	the rectangle defined by its `lower-left` and `upper-right` points
	Circle (point reference, number k)	the circle defined by centre `reference` and radius `k`
	Strip (polyline reference, number k)	the strip defined by a polyline `reference` extended by `k` units of measure at each side of the polyline
	Neighbor (geometry attribute, geometry reference)	returns all- nearest neighbors to `reference`, with respect to entries in `attribute`
	All_Neighbor_Pairs (geometry attribute1, geometry attribute2)	all- closest pairs between entries in `attribute1`, on the one hand, and in `attribute2`, on the other hand
Temporal Operation	**Interval** (date left, date right)	the interval defined by its `left` and `right` end points
Trajectory-based Operations	**Length** (mpoint reference)	the length (in distance measure) of the trajectory `reference`
	Duration (mpoint reference)	the duration (in time measure) of the trajectory `reference`
	Speed (mpoint reference)	the speed (in distance/time measure, i.e., length divided by duration) of the trajectory `reference`
	Similarity (mpoint attribute, mpoint reference, norm function)	all entries in `attribute`, ranked according to norm `function` with respect to `reference`

Table 1: List of useful operations

In order to support the proposed benchmark queries, we have utilized some operations. In Table 1, we list the most important operations used in this section (`geometry` could be any spatial data type; point, polyline, polygon, etc.).

4. PROCESSING BENCHMARK QUERIES

SQL-based query processing consists of four main steps: (a) translating an SQL query to its equivalent query execution plan (QEP); (b) generating several QEPs, which are equivalent to the original; (c)

evaluating the different QEPs, with respect to query optimization tools (also, taking into consideration the existing indices and join techniques); and (d) executing the 'optimal' QEP.

Obviously, query processing for MODs should follow the same methodology. In this section, we will discuss issues concerning step (c), and, in particular, the appropriate indices and specific query processing issues for the benchmark queries presented in Section 3.

4.1. Indices for moving objects

In the literature, work on indexing of moving objects is classified as follows: (i) indexing current locations of objects and asking current or future queries or (ii) indexing the past (and, sometimes, current) locations of objects and asking past or current queries. In the application of interest, *l*-service, we are interested in both (i) and (ii). A second classification has to do with the type of change supported, (a) discrete or (b) continuous. We consider (b) as a more interesting case than (a) as we already discussed in Section 1. In the sequel, we provide the requirements that an index should address and then try to choose from the existing proposals.

4.1.1. Indexing Requirements

Considering the benchmark queries discussed in Section 3, one by one, the following requirements arise:

- Initialization phase (numbered Operation #0a, in section 3) assumes an index that supports *batch loading* of data.
- Processing of Query 1 and Query 2 requires an index that efficiently supports *coordinate-based* queries (point and range, respectively). In our framework, point queries retrieve routes falling on a given point (x, y, t) in the 3D parameter space. Respectively, range queries retrieve routes intersecting a given 3D area (x_1, y_1, t_1, x_2, y_2, t_2).
- On the other hand, Query 3 is an example of a *distance-based* query. Those types of queries cannot be defined in a single 3D parameter space since the norm of distance in the (x-, y-) plane can by no means be identical to that in the t- axis[5]. Thus, two separate distance-based operations need to be supported; one for space and one for time. The former (`Circle`) appears in Query 3 while the latter could be of the type `Interval(t-δ,t-δ)`. In terms of indexing requirements, the above indicate that an index with a balanced efficiency for both space- and time- specific filtering is necessary.
- Processing *nearest-neighbor* queries, such as Query 4, is a well-studied subject in spatial databases. Due to the same reasons with distance-based queries discussed earlier, we distinguish between

spatial- and temporal- NN queries and avoid mixing them in a single spatiotemporal variation. Thus, indices appropriate for both types of NN queries are required[6].

- *Topological* queries, involving semantics such as enter, leave, cross, etc., are not plain coordinate-based, since in order to answer them we need to have knowledge about the 'evolution' of the trajectory; whether it 'started' inside or outside a given area, and so on. Query 5 is such an example. The support of this query type by spatiotemporal indices is not straightforward since they have to maintain the notion of the 'trajectory' as a single entity (not just as a set of line segments). The same requirement comes up after Query 10; efficient processing of unary operators, such as length, speed, area covered, etc. also assumes the special treatment of the 'trajectory'.
- Query 6 is a variation of distance-based Query 3. Here, the reference object is not a point (in space or time) but a trajectory itself. It is relevant to the so-called *strip* or *buffer* query in spatial database literature (Chan, 2001), which can be treated by spatiotemporal indices in at least two different ways[7].
- A similar requirement (and treatment) exists for the MST query type, expressed in Query 7; this query will be discussed in detail later, in subsection 4.2, together with its extension, Query 9.
- Query 8 is a typical join and obviously constitutes one of the most expensive operations in the list of the proposed benchmark queries. Regarding indexing requirements, what has been already discussed (coordinate-based vs. topological queries) also applies on joins.

4.1.2. Choosing from the existing *menu* of indices

A straightforward solution to provide index support to *mpoints* is to decompose them into pure spatial (sets of points or line segments) and temporal properties (sets of time instances or intervals) and build the corresponding indices, e.g. a classic R-tree (Guttman, 1984; Beckmann et al., 1990) and a RI-tree (Kriegel et al., 2000), respectively.

---

[5] It cannot be claimed, for example, that a spatial distance of e.g. 10 meters is equivalent to a temporal distance of e.g. 10 seconds (or minutes, etc.).

[6] An example *l*-service, based on NN queries, is the so-called Nearest Available Parking lot Application (NAPA), presented in (Chon et al., 2002). A variation of NN queries, the so-called *reverse nearest-neighbor query* (RNN) is also useful in such applications. In a RNN, data objects that have a given query object as their nearest neighbor have to be found (Stanoi et al., 2000).

[7] One solution is to consider strip query as a special case of range query, where the reference is an irregular zone instead of a rectangle; of course, a zone could be approximated by its Minimum Bounding Rectangle (MBR) for the purposes of the filter step. An alternative solution is its decomposition in a set of distance-based (sub-) queries, where a number of representative points is extracted by the trajectory and each of these points corresponds to the centre of a circle such as the total of the circles will approximate (by completely covering) the strip; in that case, the more agile is the reference trajectory the larger will be the number of the approximating circles, hence, the number of corresponding distance-based queries.

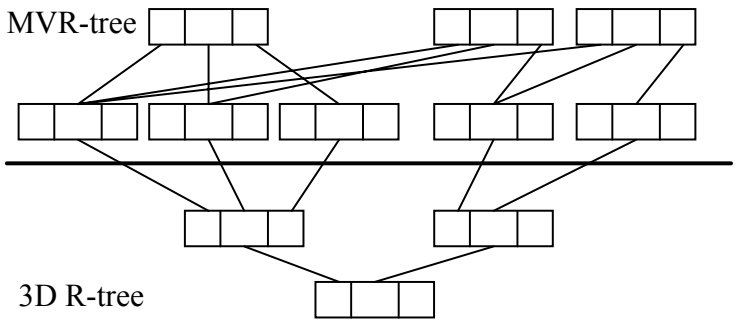


Figure 5: The MV3R-tree structure (Tao and Papadias, 2001)

A second, also straightforward, solution for the indexing of spatiotemporal data is the consideration of time as just an extra dimension and the representation of 2D moving points or regions as 3D polylines or polyhedra, respectively (cf. Figure 2). The 3D R-tree (Theodoridis et al., 1996) was exactly an index of 3D polylines and it was one of the early attempts in the field. It could support either discrete or continuous changes. The 3D R-tree could index past locations only. Therefore, a hybrid structure consisting of a 3D R-tree for past locations and a (pure spatial) 2D R-tree for current locations, so-called the 2+3 R-tree, was proposed in (Nascimento et al., 1999). However, the 2+3 R-tree supported discrete changes only.

Based on the observation that a spatiotemporal index preserving history could logically be represented by a 'forest' of spatial indices (as many as the number of different snapshots) that physically share common nodes, the HR-tree was proposed in (Nascimento and Silva, 1998). In a similar way, Kollios et al. (2001) recently proposed the partially-persistent R-tree (PPR-tree), actually a directed acyclic graph of nodes with a number of root nodes, where each root is responsible for recording a subsequent part of the ephemeral R-tree evolution. The disadvantage of both indexing techniques is that space requirements become prohibitive for agile datasets.

To overcome the shortcomings of the 3D R-tree and the HR-tree, Tao and Papadias (2001) proposed the MV3R-tree, consisting of a multi-version R-tree and small auxiliary 3D R-tree built on the leaves of the former (as illustrated in Figure 5). Through extensive experimentation, the MV3R-tree turned out to be efficient in both timestamp and interval queries with relatively small space requirements.

In a totally different approach, Pfoser et al. (2000) proposed the TB-tree (the trajectory-bundle tree). The TB-tree relaxes a fundamental R-tree property, i.e., keeping neighboring entries together in a node, and strictly preserves trajectories such that a leaf node only contains segments belonging to the same trajectory, as illustrated in Figure 6 (this is achieved by giving up on space discrimination). The TB-tree indexes past locations of objects and supports continuous changes.

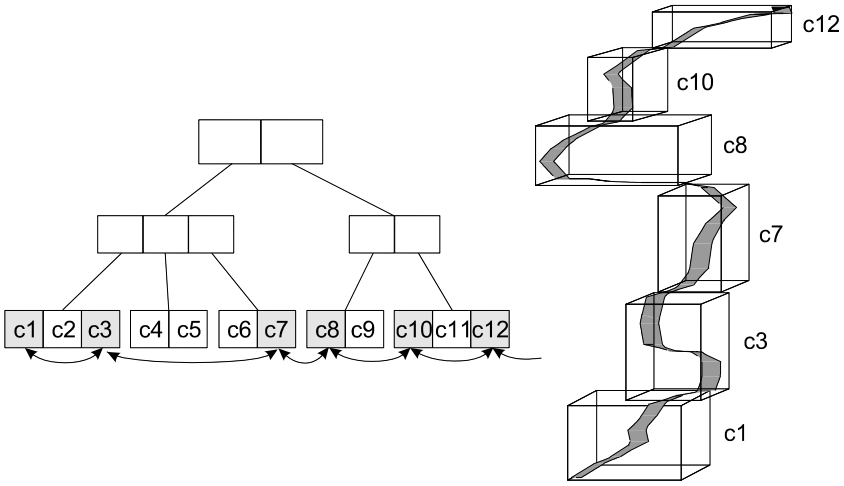


Figure 6: The TB-tree structure (Pfoser et al., 2000)

Moving to the field of indices for current locations (and future queries), Šaltenis et al. (2000) proposed the TPR-tree (for time-parameterized R-tree), which extends the R-tree to efficiently support current and anticipated future locations of moving points. The novelty of the TPR-tree is that bounding rectangles in the tree structure are functions of time, instead of fixed spatial objects. Recently, Šaltenis and Jensen (2002) extended previous work to support *expiring* information, i.e., data that is not valid after an expiration time passes, thus proposing the R^{EXP}-tree.

Also in the field of indexing current locations, Porkaew et al. (2001) provide algorithms for range and nearest-neighbor queries on both spatial and temporal dimensions. In particular, two alternative approaches are presented, one using Native Space Indexing (NSI) in which indexing is performed in the original space where motion occurs, and the other using Parameter Space Indexing (PSI) where a space defined by motion parameters (location, velocity, time) is used. Experimental results indicated that NSI outperforms PSI, especially because of the loss of locality associated with PSI.

Addressing the requirements presented in subsection 4.1.1, we suggest that an efficient index should equally support coordinate-based queries as well as queries based on the semantics of trajectories (such as topological queries and those involving unary operators on trajectory characteristics). Assuming that we are interested in:

i) *asking past or current queries* (see Subsection 3.5 for the reasons why we excluded future queries) and

ii) *supporting continuous changes* (discrete change is not the case in MODs we discuss here, see discussion in Section 1),

and considering current state-of-the-art as classified in Table 2, the list of candidates for MODs focusing on *l*-services would include at least the MV3R-tree and the TB-tree from the first group (indices supporting past or current queries) plus the TPR-tree and the NSI from the second group

	Indexing past and current locations (and asking past or current queries)	**Indexing current locations (and asking current or future queries)**
Supporting discrete changes	2+3 R-tree, HR-tree, PPR-tree, MV3R-tree	---
Supporting continuous changes	TB-tree, MV3R-tree	TPR-tree, R^{EXP}-tree, NSI, PSI

Table 2: A taxonomy of (some of) spatiotemporal indices

(indices supporting current or future queries)[8]. As a future work, a thorough experimentation among those techniques, based on the benchmark queries we propose in this paper, would be very interesting and could perhaps give better hints for an overall winner.

Efficient join algorithms should accompany the proposed indices. Regarding the 3D R-tree, the literature is extensive since it is actually an R-tree. In the processing of a join query A⋈B between two spatial datasets A and B, researchers usually distinguish among three different cases: (i) both sets, A and B, are supported by spatial indices, such as R-trees; (ii) only one set, either A or B, is supported by a spatial index; (iii) neither A nor B is supported by a spatial index. All these cases have been efficiently handled in the literature, see e.g. the works in (Brinkhoff et al., 1993; Koudas and Sevcik, 1997; Mamoulis and Papadias, 1999)[9]. On the other hand, work on join processing techniques exploiting pure spatiotemporal indices, such as those listed in Table 2, is very limited and should be further extended.

4.2. Specific query processing issues

Almost all benchmark queries listed above are typical selections and joins based on the spatiotemporal properties of data. Assisted by special purpose indices, such as the ones mentioned in the previous section, they can be efficiently processed in a state-of-the-art object-relational database system with appropriate extensions (e.g. indices). In the sequel, we focus on two benchmark queries that require rather complex handling, namely the queries on *most similar trajectories* (MST), Query 7 and Query 9.

[8] We excluded R^{EXP}-tree because expiring objects are not considered in our application, and PSI because of its inferiority against NSI, according to (Porkaew et al., 2001).

[9] In this paper, we will only consider the case of two-way joins. Processing multi-way joins is beyond the scope of the paper but the interested reader can find hints in (Papadias et al., 1999; Zhu et al., 2001).

The problem of finding the MSTs with respect to a given trajectory (Query 7) is relevant to finding similar time-series, e.g. the work in (Faloutsos et al., 1994). In both cases, data (either a trajectory or a time-series) is mapped to a vector in *n*-dimensional space and then a *p*-norm distance function is used to define the similarity measure[10]. However, trajectories appear to have some peculiarities that may require different approaches (Sclaroff et al., 2001). For example, two humans moving in a similar fashion bur with slightly different speeds cannot be detected as similar using Euclidean distance. A better approach is to use the Longest Common SubSequence (LCSS) model, a variation of the Time Warping model (Berndt and Clifford, 1994), which allows shifting in time by its definition. In (Kollios et al., 2002), efficient approximation algorithms and techniques to compute the similarity between trajectories, based on the LCCS model, are proposed. On the other hand, self-joining trajectory data with respect to their similarity (Query 9) has not been studied earlier, to the best of our knowledge, and constitutes an exciting topic of future research.

5. DISCUSSION AND RELATED WORK

Several benchmark databases (and queries) have appeared in the literature of non-traditional database applications. However, none of those benchmarks has addressed human motion for *l*-service applications as we do in this paper.

The 'A La Carte' benchmark (Günther et al., 1998) is a WWW-based tool consisting of a rectangle generator that builds datasets based on user defined parameters (cardinality, coverage, coordinates' distributions) and an experimentation module that runs experiments on either user built or stored sample datasets, including parts of the SEQUOIA 2000 storage benchmark (Stonebraker et al., 1993). The module is actually a spatial join performance evaluator that supports several spatial join strategies.

Most related to our work, the DOMINO prototype (Sistla et al., 1997; Wolfson et al., 1998) includes a model and a query language, called MOST and FTL, respectively, for moving objects, motivated by the application of vehicle management for the purpose of digital battlefield. MOST (for Moving Objects Spatio-Temporal) models present and future locations and, instead of consecutive locations that would require frequent updates, represents motion vectors, consisting of the direction and speed of an object. Future queries are supported in FTL language (for Future Temporal Logic) and two kinds of semantics, namely *may* and *must*, are incorporated in order to handle uncertainty in objects' locations.

Recently, Moreira et al. (2000) provided a set of 8 queries using a system for monitoring and control of fishing activities as a case study. Moving (vessels) and static objects (forbidden areas and harbors)

---

[10] The *p*-norm distance between two *n*-dimensional vectors $\overline{x}$ and $\overline{y}$ is defined as

are involved. Temporal, spatial and numeric (e.g. speed) projections of trajectories (*movements*, in that paper) are defined and the set of supported operations includes topological (in, touch, disjoint), direction (north, south, east, west, and their conjunctions) and distance relationships. Also in this work, semantics to handle uncertainty are incorporated (namely *surely*, *possibly*, *probably*).

Another complementary piece of research work includes the development of algorithms for generating large datasets of moving objects. Due to the lack of real data, such synthetic datasets provide the necessary input to research on query processing and indexing of MODs. So far, several generators have appeared, supporting unrestricted or restricted motion of points, rectangles, or regions, adding or not semantics of specific applications, etc.

- The GSTD algorithm ("Generating Spatio-Temporal Datasets") was proposed in (Theodoridis et al., 1999). A web interface for enabling users to generate and visualize their own datasets is described in (Theodoridis and Nascimento, 2000). The generator supports point or rectangular objects and starts by distributing object centers in the workspace according to certain distributions. After this initialization phase, the movement of objects is controlled by three key parameters: (a) the duration of object instances; (b) the shift of objects; and (c) the resizing of objects (only applicable to rectangular objects). The original version assumed unrestricted motion on the workspace while (Pfoser and Theodoridis, 2000) introduced restrictions as an infrastructure of stationary objects. The generator is available at: `http://www.cti.gr/RD3/GSTD`.
- The generator proposed in (Brinkhoff, 2000; Brinkhoff, 2002) focuses on network-based moving objects. The driving application is the field of traffic telematics. Important concepts of the generator are the maximum speed and the maximum edge capacity, the maximum speed of the object classes, the interaction between objects, etc. The generator is available at: `http://www.fh-oldenburg.de/iapg/personnen/brinkhof/generator.html`.
- The OPORTO generator (Saglio and Moreira, 2001) supports point and region moving objects and is motivated by the idea of fishing ships. Ships are attracted by shoals of fish, while at the same time they are repulsed by storm areas. Fishes themselves are attracted by plankton areas. Ships are moving points, whereas shoals, plankton and storm areas are moving regions. The generator is available at: `http://www-inf.enst.fr/~saglio/etudes/oporto/`.
- A Generator for Time-Evolving Regional Data (so-called, G-TERD) was proposed in (Tzouramanis et al., 2002). The basic concepts that determine the function of G-TERD are the structure of complex regional objects, their color, maximum speed, zoom and rotation-angle per time slot, the influence of other moving or static objects on the speed and on the moving direction of an object,

---

$$L_p(\bar{x},\bar{y}) = \left(\sum_{i=1}^{n} (x_i - y_i)^p \right)^{1/p}.$$

the position and movement of scene-observer, the statistical distribution of each changing factor and finally, time. The generator is available at: `http://delab.csd.auth.gr/stdbs/g-terd.html`.

Usually, the above research efforts are parts of a wider research framework for benchmarking environments for spatiotemporal and moving objects databases, including query languages, physical representations, access methods, collections of real and synthetic datasets and experimentation test beds.

6. CONCLUSIONS

In this paper, we discussed the requirements for databases supporting location-based service (thus, *l*-service) applications, a special case of spatiotemporal databases. In particular, we provided a list of ten representative queries, including selection queries on stationary and moving reference objects, join queries and unary operations on trajectories of moving objects. This list of queries could constitute a benchmark for an *l*-service database and implementations in current state-of-the-art database systems could be evaluated with respect to that, in terms of efficiency.

Next, we surveyed recent work in query processing for those query types, with emphasis on indexing of moving objects. Since we are interested in *asking past or current queries* and *supporting continuous changes* we suggested that a list of good candidates for indexing *l*-service databases would include at least the MV3R-tree (Tao and Papadias, 2001), the TB-tree (Pfoser et al., 2000), the TPR-tree (Šaltenis et al., 2000) and the NSI (Porkaew et al., 2001), and a comparative study based on their performance with respect to the proposed set of benchmark queries would be very interesting as a task for future work. Of course, we admitted that more research is needed in topics, such as the spatiotemporal join processing and the processing of the so-called most similar trajectories (MST) and most similar pairs of trajectories (MSPT) queries.

Open fields for future research also include:

- *Visualizing* benchmark queries and results: Visual representations of spatiotemporal SQL-like queries are useful for understanding and further exploiting their results. Although several visual query languages for (static) spatial data do exist in the literature, e.g. PQBE (for Pictorial Query-by-Example) (Papadias and Sellis, 1995) and Spatial-Query-by-Sketch (Egenhofer, 1996), we are not aware of their extensions to support motion. A single exception is the work in (Bonhomme et al., 1999), where a visual language for spatial data, called Lvis, was enhanced to support spatiotemporal queries. Those queries, however, were just combinations of typical spatial and temporal ones, without any notion of movement.

- Introducing *fuzziness* and *uncertainty*: Several natural spatiotemporal phenomena involve fuzziness, concerning "blurry" situations, and uncertainty, expressing the "not-exactly-known" reality (Pfoser and Tryfona, 2001). As already discussed, current technology allows us to sample an object's movement, thus introducing uncertainty of its location on non-sampled timestamps. Several queries of our benchmark (e.g. Query 5) are affected by this "problematic" issue and need special handling, e.g. by adding the *lens area* notion (Pfoser and Jensen, 1999).
- Enhancing benchmark query set with *motion mining*: Considering that future query languages will be SQL-like languages enhanced with data mining facilities, we claim that mining motion patterns is an emerging operation. A framework towards this direction is proposed in (Sclaroff et al., 2001).
- Extending the proposed framework to a *decentralized environment*: Assuming that a database for *l*-service could be distributed among partially overlapping cells (following the mobile telephony paradigm) or even among the moving objects themselves, indexing and query processing methodologies discussed in Section 4 need to be revisited.

REFERENCES

Allen, J. (1983). Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832-843, November 1983.

Bartels, R., J. Beatty, and B. Barsky (1987). *An Introduction to Splines for Use in Computer Graphics & Geometric Modeling*. Morgan Kaufmann Publishers, Inc., 1987.

Beckmann, N., H.-P. Kriegel, R. Schneider, and B. Seeger (1990). The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of ACM SIGMOD Int'l Conference on Management of Data*, SIGMOD'90, Atlantic City - NJ, USA, May 1990.

Berndt, D. and J. Clifford (1994). Using Dynamic Time Warping to Find Patterns in Time Series. In *Proceedings of AAAI Workshop on Knowledge Discovery in Databases*, KDD'94, Seattle – WA, USA, July 1994.

Böhlen, M., C.S. Jensen, and B. Skjellaug (1998). Spatio-Temporal Database Support for Legacy Applications. In *Proceedings of ACM Symposium on Applied Computing*, ACM-SAC'98, Atlanta – GA, USA, February/March 1998.

Bonhomme, C., C. Trépied, M.-A. Aufaure, and R. Laurini (1999). A Visual Language for Querying Spatio-Temporal Databases. In *Proceedings of ACM Symposium on Geographic Information Systems*, ACM-GIS'99, Kansas City - MO, USA, November 1999.

Brinkhoff, T., (2000). Generating Network-Based Moving Objects. In *Proceedings of the 12th Int'l Conference on Scientific and Statistical Database Management*, SSDBM'00, Berlin, Germany, July 2000.

Brinkhoff, T., (2002). A Framework for Generating Network-Based Moving Objects. *Geoinformatica*, 6(2):153-180, June 2002.

Brinkhoff, T., H.-P. Kriegel, and B. Seeger (1993). Efficient Processing of Spatial Joins Using R-trees. In *Proceedings of ACM SIGMOD Int'l Conference on Management of Data*, SIGMOD'93, Washington, D.C., USA, May 1993.

Chan, E.P.F. (2001). Evaluation of Buffer Queries in Spatial Databases. In *Proceedings of the 7th Int'l Symposium on Spatial and Temporal Databases*, SSTD'01, Los Angeles – CA, USA, July 2001.

Cheng, T.S. and S.K. Gadia (1994). A Pattern Matching Language for Spatio-Temporal Databases. In *Proceedings of the 3rd Int'l Conference on Information Knowledge and Management*, CIKM'94, Gaithersburg – MD, USA, November/December 1994.

Chomicki, J. and P.Z. Revesz (1999). A Geometric Framework for Specifying Spatiotemporal Objects. In *Proceedings of the 6th Int'l Workshop on Temporal Representation and Reasoning*, TIME'99, Orlando, Florida, USA, May 1999.

Chon, H.D., D. Agrawal, A. El Abbadi (2002). NAPA: Nearest Available Parking lot Application. In *Proceedings of the 18th IEEE Conference on Data Engineering*, ICDE'02, San Jose – CA, USA, February/March 2002.

Corral, A., Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos (2000). Closest Pair Queries in Spatial Databases. In *Proceedings of ACM SIGMOD Int'l Conference on Management of Data*, SIGMOD'00, Dallas - TX, USA, May 2000.

Egenhofer, M.J. (1996). Spatial-Query-by-Sketch. In *Proceedings of IEEE Symposium on Visual Languages*, VL'96, Boulder - CO, USA, 1996.

Egenhofer, M.J. and R.D. Franzosa (1991). Point-Set Topological Spatial Relations. *Int'l Journal of Geographical Information Systems*, 5(2):161-174, June 1991.

Erwig, M. and M. Schneider (1999). Developments in Spatio-Temporal Query Languages. In *Proceedings of the 10th Int'l Workshop on Database & Expert Systems Applications,* DEXA'99, Florence, Italy, September 1999.

Faloutsos, C., M. Ranganathan, and Y. Manolopoulos (1994). Fast Subsequence Matching in Time Series Databases. In *Proceedings of ACM SIGMOD Int'l Conference on Management of Data*, SIGMOD'94, Minneapolis, Minnesota, USA, May 1994.

Grumbach, S., P. Rigaux, and L. Segoufin (1998). Spatio-Temporal Data Handling with Constraints. In *Proceedings of ACM Symposium on Geographic Information Systems*, ACM-GIS'98, Washington, D.C., USA, November 1998.

Günther, O., V. Oria, P. Picouet, J.-M. Saglio, and M. Scholl (1998). Benchmarking Spatial Joins *à la Carte*. In *Proceedings of the 10th Int'l Conference on Scientific and Statistical Database Management*, SSDBM'98, Capri, Italy, July 1998.

Güting, R.H., M. Böhlen, M. Erwig, C.S. Jensen, N. Lorentzos, M. Schneider, and M. Vazirgiannis (2000). A Foundation for Representing and Querying Moving Objects. *ACM Transactions on Database Systems*, 25(1):1-42, March 2000.

Guttman, A. (1984). R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD Int'l Conference on Management of Data*, SIGMOD'84, Boston - MA, USA, June 1984.

Hjaltason, G.R. and H. Samet (1998). Incremental Distance Join Algorithms for Spatial Databases. In *Proceedings of ACM SIGMOD Int'l Conference on Management of Data*, SIGMOD'98, Seattle - WA, USA, June 1998.

Kollios, G., V.J. Tsotras, D. Gunopulos, A. Delis, and M. Hadjieleftheriou (2001). Indexing Animated Objects Using Spatiotemporal Access Methods. *IEEE Transactions on Knowledge and Data Engineering*, 13(5):758-777. September/October 2001.

Kollios, G., M. Vlachos, and D. Gunopulos (2002). Discovering Similar Multidimensional Trajectories. In *Proceedings of the 18th IEEE Conference on Data Engineering*, ICDE'02, San Jose – CA, USA, February/March 2002.

Koudas, N. and K.C. Sevcik (1997). Size Separation Spatial Join. In *Proceedings of ACM SIGMOD Int'l Conference on Management of Data*, SIGMOD'97, Tucson - AZ, USA, June1997.

Kriegel, H.-P., M. Pötke, and T. Seidl (2000). Managing Intervals Efficiently in Object-Relational Databases. In *Proceedings of the 26th Int'l Conference on Very Large Data Bases*, VLDB'00, Cairo, Egypt, September 2000.

Mamoulis, N. and D. Papadias (1999). Integration of Spatial Join Algorithms for Processing Multiple Inputs. In *Proceedings of ACM SIGMOD Int'l Conference on Management of Data*, SIGMOD'99, Philadelphia - PA, USA, June1999.

Moreira, J., C. Ribeiro, and T. Abdessalem (2000). Query Operations for Moving Objects Database Systems. In *Proceedings of ACM Symposium on Geographic Information Systems*, ACM-GIS'00, Kansas City, USA, November 2000.

Nascimento, M.A. and J.R.O. Silva (1998). Towards Historical R-trees. In *Proceedings of ACM Symposium on Applied Computing*, ACM-SAC'98, Atlanta – GA, USA, February/March 1998.

Nascimento, M.A., J.R.O. Silva, and Y. Theodoridis (1999). Evaluation of Access Structures for Discretely Moving Points. In *Proceedings of Int'l Workshop on Spatio-Temporal Database Management*, STDBM'99, Edinburgh, Scotland, UK, September 1999

Oracle (2002). Oracle ® Locator – Location-based Services for Oracle 9i. Oracle Technology Network data sheet. Available at: http://otn.oracle.com/products/oracle9i/datasheets/spatial/9iR2_locator_ds.html.

Papadias, D. and T. Sellis (1995). A Pictorial Query by Example Language. *Journal of Visual Languages and Computing*, 6(1):53-72, March 1995.

Papadias, D., N. Mamoulis, and Y. Theodoridis (1999). Processing and Optimization of Multi-way Spatial Joins Using R-trees. In *Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS'99, Philadelphia - PA, USA, May/June 1999.

Parent, C., S. Spaccapietra, and E. Zimányi (1999). Spatio-Temporal Conceptual Models: Data Structures + Space + Time. In *Proceedings of ACM Symposium on Geographic Information Systems*, ACM-GIS'99, Kansas City - MO, USA, November 1999.

Patel, J. et al. (1997). Building a Scalable Geo-Spatial DBMS: Technology, Implementation, and Evaluation. In *Proceedings of ACM SIGMOD Int'l Conference on Management of Data*, SIGMOD'97, Tucson - AZ, USA, June 1997.

Pfoser, D. and C.S. Jensen (1999). Capturing the Uncertainty of Moving-Object Representations. In *Proceedings of the 6th Int'l Symposium on Spatial Databases*, SSD'99, Hong Kong, China, July 1999.

Pfoser, D. and Y. Theodoridis (2000). Generating Semantics-Based Trajectories of Moving Objects. In *Proceedings of Int'l Workshop on Emerging Technologies for Geo-Based Applications*, Ascona, Switzerland, May 2000.

Pfoser, D. and N. Tryfona (1998). Requirements, Definitions and Notations for Spatiotemporal Application Environments. In *Proceedings of ACM Symposium on Geographic Information Systems*, ACM-GIS'98, Washington, D.C., USA, November 1998.

Pfoser, D. and N. Tryfona (2001). Capturing Fuzziness and Uncertainty in Spatiotemporal Applications. In *Proceedings of the 5th East-European Conference on Advances in Databases and Information Systems*, ADBIS'01, Vilnius, Lithuania, September 2001.

Pfoser, D., C.S. Jensen, and Y. Theodoridis (2000). Novel Approaches to the Indexing of Moving Object Trajectories. In *Proceedings of the 26th Int'l Conference on Very Large Data Bases*, VLDB'00, Cairo, Egypt, September 2000.

Porkaew, K., I. Lazaridis, and S. Mehrotra: Querying Mobile Objects in Spatio-Temporal Databases. In *Proceedings of the 7th Int'l Symposium on Spatial and Temporal Databases*, SSTD'01, Los Angeles – CA, USA, July 2001.

Saglio, J.-M. and J. Moreira (2001). Oporto: a Realistic Scenario Generator for Moving Objects. *Geoinformatica*, 5(1):71-93, March 2001.

Šaltenis, S. and C.S. Jensen (2002). Indexing of Moving Objects for Location-based Services. In *Proceedings of the 18th IEEE Conference on Data Engineering*, ICDE'02, San Jose – CA, USA, February/March 2002.

Šaltenis, S., C.S. Jensen, S.T. Leutenegger, and M.A. Lopez (2000). Indexing the Positions of Continuously Moving Objects. In *Proceedings of ACM SIGMOD Int'l Conference on Management of Data*, SIGMOD'00, Dallas - TX, USA, May 2000.

Sclaroff, S., G. Kollios, M. Betke, and R. Rosales (2001). Motion Mining. In *Proceedings of the 2nd Int'l Workshop on Multimedia Databases and Image Communication*, MDIC'01, Amalfi, Italy, September 2001.

Seeger, B. et al. (2001). Seeking the Truth – Curses and Blessings of Experiments. Panel discussion in the 7th *Int'l Symposium on Spatial and Temporal Databases*, SSTD'01, Los Angeles - CA, USA, July 2001.

Sellis, T., M. Koubarakis, et al. (2002). *Spatiotemporal Databases – the Chorochronos Project*. Springer-Verlag (LNCS Series), in press.

Sistla, P., O. Wolfson, S. Chamberlain, and S. Dao (1997). Modeling and Querying Moving Objects. In *Proceedings of the 13th IEEE Conference on Data Engineering*, ICDE'97, Birmingham, UK, April 1997.

Stanoi, I., D. Agrawal, and A. El Abbadi (2000). Reverse Nearest Neighbor Queries for Dynamic Databases. In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, Dallas - TX, USA, May 2000.

Stonebraker, M., J. Frew, K. Gardels, and J. Meredith (1993). The SEQUOIA 2000 Storage Benchmark. In *Proceedings of ACM SIGMOD Int'l Conference on Management of Data*, SIGMOD'93, Washington, DC, USA, May 1993.

Tao, Y. and D. Papadias (2001). MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In *Proceedings of the 27th Int'l Conference on Very Large Data Bases*, VLDB'01, Roma, Italy, September 2001

Theodoridis, Y. and M.A. Nascimento (2000). Generating Spatiotemporal Datasets on the WWW. *SIGMOD Record*, 29(3):39-43, September 2000.

Theodoridis, Y., M. Vazirgiannis, and T. Sellis (1996). Spatio-Temporal Indexing for Large Multimedia Applications. In *Proceedings of the 3rd IEEE Conference on Multimedia Computing and Systems*, ICMCS'96, Hiroshima, Japan, June 1996.

Theodoridis, Y., T. Sellis, A. Papadopoulos, and Y. Manolopoulos (1998). Specifications for Efficient Indexing in Spatiotemporal Databases. In *Proceedings of the 10th Int'l Conference on Scientific and Statistical Database Management*, SSDBM'98, Capri, Italy, July 1998.

Theodoridis, Y., J.R.O. Silva, and M.A. Nascimento (1999). On the Generation of Spatiotemporal Datasets. In *Proceedings of the 6th Int'l Symposium on Spatial Databases*, SSD'99, Hong Kong, China, July 1999.

Tryfona, N. and C.S. Jensen (1999). Conceptual Data Modeling for Spatiotemporal Applications. *Geoinformatica*, 3(3):245-268, September 1999.

Tzouramanis, T., M. Vassilakopoulos, and Y. Manolopoulos (2002). On the Generation of Time-Evolving Regional Data. *Geoinformatica*, 6(3):207-231, September 2002.

Wolfson, O., B. Xu, S. Chamberlain, and L. Jiang (1998). Moving Objects Databases: Issues and Solutions. In *Proceedings of the 10th Int'l Conference on Scientific and Statistical Database Management*, SSDBM'98, Capri, Italy, July 1998.

Worboys, M.F. (1994). A Unified Model for Spatial and Temporal Information. *The Computer Journal*, 37(1):26-34, 1994.

Zhu, H., J. Su, and O.H. Ibarra (2001). On Multi-Way Spatial Joins with Direction Predicates. In *Proceedings of the 7th Int'l Symposium on Spatial and Temporal Databases*, SSTD'01, Los Angeles – CA, USA, July 2001.

APPENDIX: THE BENCHMARK DATABASE IN ODL

```
interface Human (key id) {
     attribute string id;                        // unique feature identifier
     attribute mpoint route;                     // route followed by him/her
     attribute set<varchar> interests;           // list of products he/she is interested in
     attribute set<varchar> requests;            // list of products he/she has requested
     relationship set<Building> visit            // building(s) he/she has visited
          inverse Building::visitedBy;
     relationship set<Road> pass                 // road(s) he/she has passed
          inverse Road::passedBy;
};

interface Building (key id) {
     attribute string id;                        // unique feature identifier
     attribute polygon location;                 // location of the building
     relationship set<Human> visitedBy           // human(s) visited it
          inverse Human::visit;
};

interface Shop: Building {
     attribute struct ShopOffer
          {string info,                          // information about the offer
               set<time_interval> validtime}     // time(s) the offer is valid
          offer;
};

interface Other: Building {
     attribute integer type;                     // 1=restaurant, 2=gas-station, and so on
};

Interface Road (key id) {
     attribute string id;                        // unique feature identifier
     attribute polyline shape;                   // shape of the road
     relationship set<Human> passedBy            // human(s) passed it
          inverse Human::pass;
};
```