Specifications for Efficient Indexing in Spatiotemporal Databases

Yannis Theodoridis Timos Sellis Apostolos N. Papadopoulos Yannis Manolopoulos

Computer Science Division
Dept. of Electrical and Computer Engineering
National Technical University of Athens
Zographou 15773, Athens, HELLAS (GREECE)
{theodor, timos}@cs.ntua.gr

Department of Informatics
Aristotle University of Thessaloniki
Thessaloniki 54006, HELLAS (GREECE)
{apapadop, manolopo}@athena.auth.gr

Abstract: *A new issue that arises in modern applications involves the efficient manipulation of (static or moving) spatial objects, and the relationships among them. As a result, modern database systems should be able to efficiently support that type of data. Towards this goal, appropriate extensions of multidimensional access methods can be exploited in order to index and retrieve spatiotemporal objects, satisfying users' demands. This paper introduces the basic specifications such a spatiotemporal index structure should follow, evaluates existing proposals with respect to the above specifications, and illustrates issues of interest involving object representation, query processing, and index maintenance.*

1. Introduction

Efficient storage and retrieval techniques for non-traditional data, such as geometric objects in multidimensional space, are necessary in modern database systems. *Spatiotemporal Database Management Systems* (STDBMS), in particular, should (i) offer appropriate data types and query language to support (static or moving) spatial data, (ii) provide efficient indexing and retrieval methods, and (iii) exploit cost models for specialized spatiotemporal operations for query processing and optimization purposes. Relevant applications include Geographic Information Systems (GIS), Transaction / Valid Time Databases, Multimedia Systems, as well as Scientific and Statistical Databases, such as databases for medical, weather, environmental, astrophysics etc. applications.

The primary goal of an STDBMS is the accurate modeling of the real world; that is a dynamic world, which involves objects whose position, shape and size change over time. Real world examples include storage and manipulation of trajectories, fire or hurricane front monitor, simulators (e.g. flight simulators), weather forecast etc. For example, consider the query: "*find the past days in which the solar magnetic wind showed patterns similar to today's patterns over the region of Mediterranean sea*". Worboys [28] also provides a survey of, mainly GIS oriented, spatiotemporal applications on administrative areas, road networks, and land ownership. On such data sets, *selection*, *join* or *nearest-neighbor* queries need to be supported.

Figure 1 illustrates two real world example applications that include spatiotemporal data [22]. The first one is taken from the maritime world, where there is a need for the representation and reasoning on moving point objects; the second one comes from the environmental applications, and specifically from forest fire management, where there is a need to represent and analyze regions that change their location over time.

Figure 1a illustrates a simplified example of two vessel trajectories (A, B) in the international waters (IW) between two countries. Assuming a navigation on a plane surface, each trajectory can be represented with a line object in three-dimensional space defined by the Cartesian system (x, y, t). Two typical queries regarding this situation, which are handled as *spatiotemporal join* operations, are: "*Did any vessel cross any national boundary (NB_i) on the 3rd of July*?" or "*Is there any chance for a vessel collision to occur*?". As a second example, Figure 1b illustrates three snapshots (F_{i1}, F_{i2}, F_{i3}) of a burning area (F_i) in a forest. Assuming a fire on a plane surface (a projection of the earth surface), the burning area consists of a set of regions, possibly with holes (e.g. lakes). Each fire can be represented as a complex solid in three-dimensional space defined by the Cartesian system (x, y, t). Some typical queries regarding this situation, which are handled as *spatiotemporal selection* operations, are: "*Did city C_1 ever burn?*" or "*Did fire-front of F_i ever meet river Rv? If yes, when did that happen?*".

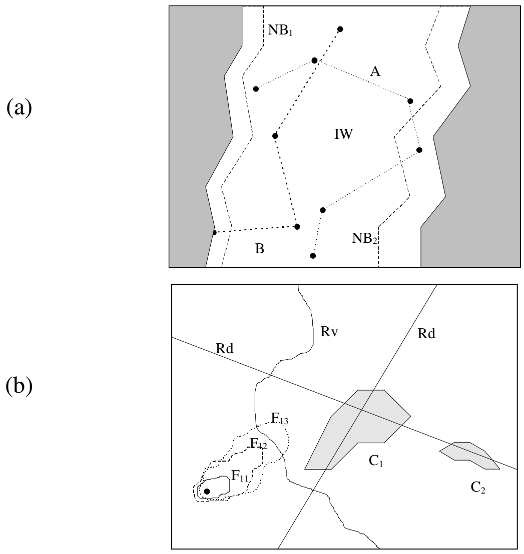

Figure 1: Two real world examples with spatiotemporal objects

In the literature, several spatial access methods have been proposed for multidimensional space without, however, taking the time aspect into consideration. These methods are capable of manipulating points, arbitrary shaped objects (e.g. polygons), or raster data. A recent survey can be found in [8]. On the other hand, temporal access methods have been proposed to index valid and/or transaction time, where space is not considered at all. A large family of access methods has been proposed to support multiversion / temporal data [2, 11, 12, 13], by keeping track of data evolution over time (e.g. assume a database consisting of medical records, or employees' information, or bank transactions, etc.). For a survey on temporal access methods see [23].

To the best of our knowledge, we are aware of only four indexing methods that consider both spatial and temporal attributes of objects, namely *MR-trees* and *RT-trees*, proposed by Xu et al. in [29], *3D R-trees* by Theodoridis et al. in [27], and *HR-trees*, proposed by Nascimento and Silva in [15]. These approaches have the following characteristics:

- 3D R-trees treat time as another dimension using a 'state-of-the-art' spatial indexing method, namely the R-tree [5, 9],
- MR-trees and HR-trees embed the concept of overlapping into R-trees in order to represent successive states of the database [14], and
- RT-trees couple time intervals with spatial ranges in each node of the tree structure by adopting ideas from R-trees and TSBT trees [13].

In this paper, we study the specific issues that arise when handling spatiotemporal (i.e., time-evolving spatial) objects, and differentiate the possible solutions on efficient indexing and retrieval from those involving static spatial objects. Our study constitutes the first attempt towards a *specification and classification scheme* for spatiotemporal access methods (STAMs). The rest of the paper is organized as follows: In Section 2 we discuss the motivation for this study. Section 3 describes a specification and classification scheme for efficient indexing and query processing in spatiotemporal databases. Section 4 surveys and classifies relevant index structures proposed in the past. We conclude in Section 5, giving also directions for future work.

2. Motivation

In the literature, there has been a limited work on formalization and modeling of spatiotemporal databases. Moreover, different background of researchers (temporal versus spatial databases) has usually led to different (and possibly incompatible) definitions. A primitive classification, for example, distinguishes between *discrete* and *continuous* modeling. In the rest of the paper, we adopt the following (discrete) definition for *spatiotemporal objects*.

> Definition: A spatiotemporal object *o* (declared by its identification number *o_id*) is a time-evolving spatial object, i.e., its evolution (or history) is represented by a set of triplets (*o_id*, s_i, t_i), where s_i is the location of object *o_id* at instant t_i (s_i and t_i are called *space-stamp* and *time-stamp*, respectively).

According to the above definition, a two-dimensional point- (non-point) time-evolving object is represented by a line (solid) in three-dimensional space. Figure 2 illustrates two examples: (a) a moving point and (b) a moving region (*mpoint* and *mregion*, respectively, according to the terminology proposed in [7]). In this paper we study space of dimensionality d = 2. However, 3- or, in general, d-dimensional spatial objects could be supported in a similar way.

One of the tasks to be supported by an STDBMS is the efficient indexing and retrieval of spatiotemporal objects. This task demands robust indexing techniques and fast access methods for a wide set of possible queries on spatiotemporal data. As an obvious solution to the problem, one could use an off-the-shelf access method for spatial data, such as the R-tree [5, 9] or the Quadtree [20]. However, several issues arise when attempting to consider spatiotemporal objects as three-dimensional spatial objects and manipulate them by using conventional methods from the spatial database literature, as will be discussed later.

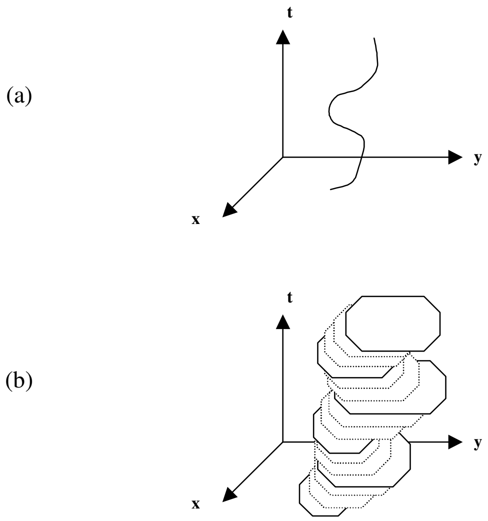

Figure 2: 2-dimensional time-evolving spatial objects

(i) data sets supported

For all applications it is important to know beforehand the nature of the data. As a first classification, a spatiotemporal database consists of time-evolving point or region objects. A *region* data type implementation could support both point- and non-point objects, since points could be treated as zero-sized regions.

(ii) valid vs. transaction time

Two separate time dimensions for database facts have been addressed in temporal database literature [10]: *transaction time* (i.e., time when the fact is current in the database and may be retrieved) and *valid time* (i.e., time when the fact is true in the modeled reality). An STDBMS would support one or even both time dimensions. Thus we distinguish among three cases: *valid-time* (also called *historical*), *transaction-time* (also called *rollback*), and *bitemporal* databases, and respective indexing methods are also classified according to the above taxonomy.

Apart from the above taxonomy, spatiotemporal databases can be characterized with respect to (a) the mobility of the database and (b) the batch or dynamic loading of data.

(iii) database mobility

There are cases where the cardinality of the database is static over time, but database objects may change their location. An example includes a military application where the positions of certain ships have to be monitored over time. The reverse one (i.e., the cardinality changes over time but the locations of all spatial objects stored in the database remain static) is also a case of interest. For instance, consider a sequence of maps using points to show the seismic activity of a region. In a third case, both the database cardinality as well as the objects' location may vary with time. All these cases are of interest for the spatiotemporal databases (obviously, the remaining one, where a database consists of a fixed number of non time-evolving objects, can be manipulated with conventional database approaches).

(iv) loading of data

Orthogonal to the above classification is the issue of bulk- or dynamically- loaded databases. In other words, we distinguish between applications with current time-stamps t_i bulk-loaded in the database (and no update in past instants be allowed) and applications with dynamic insertions / updates of objects' time-stamps allowed (not applicable, however, to transaction-time databases). The design of an efficient STAM also depends on the above classification for the database of interest.

It is well known that there exist two great families of spatial indexing methods, namely R-trees [5, 9] and Quadtrees [20]. Both structures have been incorporated into several prototype or commercial DBMS (e.g. Oracle8^{TM} Spatial Data Cartridge, Illustra^{TM} Spatial Datablade Modules) and GIS software. We argue that (a) specific representations of spatial and temporal attributes of time-evolving spatial data, and (b) specific access on data to support queries of interest are necessary extensions to those methods for STDBMS purposes:

(v) approximations of moving objects

R-trees and variants approximate spatial objects by their *Minimum Bounding Rectangles* (MBRs) in order to construct the spatial index. Although the representation of spatiotemporal objects by their MBRs has been already adopted to represent multimedia objects for authoring purposes [27], it is an inefficient solution due to the nature of moving objects. In other words, since objects are moving around in the work space, their three-dimensional MBRs usually include a vast amount of dead space, a fact that undoubtedly leads to extremely large and overlapping data rectangles, which in turn leads to inefficient indexing.

Figure 3 illustrates this case, where a point object moves from point *A* to point *B*. It is evident that the corresponding MBR covers a large portion of the data space, thus leading to high overlap and therefore to small discrimination capability of the index structure.

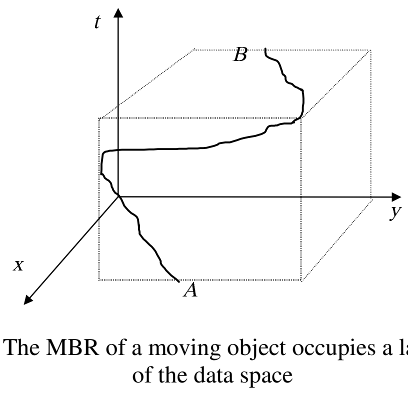

Figure 3: The MBR of a moving object occupies a large portion of the data space

Due to that side effect, alternative representations (as an example, Orenstein's decomposition schemes [16]) should be investigated. On the other hand, in order to store an object in a Linear Quadtree, we have to decompose it in a number of maximal blocks. It is anticipated that complex objects, which evolve in a dynamic environment, will consist of a great number of maximal blocks, and therefore an excess number of quadcodes will have to be inserted in the structure.

(vi) time (t-) dimension peculiarity

In the case of transaction-time databases there is a *t*-dimension peculiarity, i.e., time can be considered as an attribute taking monotonically increasing values. In such a case, the time dimension is not just another dimension, but appropriate actions should be taken in order to take advantage of this special characteristic. This property is interpreted into the following constraint: for each two consecutive triplets (o_id, s_i, t_i) and (o_id, s_{i+1}, t_{i+1}) of an object o_id, it is always $t_{i+1} > t_i$. Apart from making *t*-dimension a special case, the above property also causes a side effect: all objects' instances with t_i lower than a threshold could be considered as '*obsolete*' ones. In other words, a (practically large) portion of a STAM could be *packed* in order to reduce index space, since no updates are permitted for the corresponding data. Alternatively to packing, a *purging* process (also presented in [2] to save space in the multiversion B-tree) could be adopted in order to move the specific nodes to tertiary storage, e.g. optical disks, or even delete them.

(vii) support of specific spatiotemporal queries

The target of indexing methods is the efficient retrieval of data that satisfy the constraints of a user defined query. In spatial database literature, three main operations have been addressed: *spatial selection*, *spatial join*, and *nearest-neighbor* queries. Apart from the above common purpose queries, STDBMS users could be also interested in other, spatiotemporal application oriented ones. As an example, *timeslice queries* [10], where a timeslice could be an instant or an interval, are important for practical applications since by a series of them a visualization of the spatiotemporal database within a time interval could be succeeded. Another example includes the *history queries*, which retrieve, for example, all instances of a specified object identified by *o_id*, and involve visualization issues. The definition of a widely accepted set of spatiotemporal queries is obviously necessary, not only for the purposes of an appropriate index design but also for the design of spatiotemporal data models, query languages, benchmarks, etc.

The above issues, and possibly several others not presented in this listing, constitute the design of a STAM to be a subject different from that of a pure spatial index (i.e., by considering that *t*- direction is an extra dimension in multidimensional space). In the next section, we elaborate on those issues in order to present a specification list that should be supported by an efficient indexing method for spatiotemporal databases.

3. A List of Specifications

According to the discussion of Section 2, we classify three types of specification issues to be considered:

- issues on *data types and data sets supported*,
- issues on *index construction*, and
- issues on *query processing*.

3.1. Data Types and Data Sets Supported

As illustrated in Figure 2, both point and non-point spatial objects need to be efficiently supported by appropriate STAMs. In accordance to spatial indexing area, those indexes could be classified in PAMs (point access methods) and SAMs (spatial access methods) respectively, with respect to the data types supported. Obviously, SAMs also support point objects since points are zero-sized regions.

A second classification concerns the time dimension(s) supported. The respective classification discussed in Section 2 is illustrated in Table 1. Recall that, since at least one time dimension support is a fundamental property of an STDBMS, the fourth case of *snapshot* databases (i.e., no consideration of valid or transaction time) is not of interest in our discussion.

	Transaction time: NO	**Transaction time: YES**
Valid time: NO		*transaction-time*
Valid time: YES	*valid-time*	*bitemporal*

Table 1: Classification of spatiotemporal databases (with respect to the time dimensions supported)

In the area of spatial databases, a score of indexing methods has been proposed for special cases where data are static or dynamic. In the context of spatiotemporal databases, the temporal dimension invalidates all previous approaches with respect to the nature of spatial data (i.e. whether they are static or dynamic). However, as explained in the previous section, in the latter context we may visualize three cases of interest with respect to the motion of objects and the cardinality of the database through time. The above classification (with respect to the mobility of the data set) is illustrated in Table 2.

	Cardinality: static in time	**Cardinality: dynamic in time**
Still objects		*growing*
Moving objects	*evolving*	*full-dynamic*

Table 2: Classification of spatiotemporal databases (with respect to the mobility of the data set)

In addition, by assuming that all objects' instances are known in advance and no insertion / update is allowed (i.e., entries are *bulk-loaded* or *batch inserted* in the database) is different from the case that only current instances are dynamically inserted / updated (in a so-called *chronological* or *append-only database*), or the case that insertions of / updates to any object instance referred to any time-stamp (not meaningful in transaction- time databases, see Table 1) are allowed. The above classification (with respect to the time-stamp update issue) is illustrated in Table 3.

	Batch-only insertions	**Dynamic insertions**
Past time-stamp insertion / update **not allowed**	*static*	*chronological*
Past time-stamp insertion / update **allowed**		*dynamic*

Table 3: Classification of spatiotemporal databases (with respect to the time-stamp update issue)

In the spatial database literature, only two out of three classes (i.e., the *static* and *dynamic* ones) are meaningful, with dozens of methods proposed for each one (an exhaustive survey is found in [8]), since chronological data is meaningless in pure spatial databases. On the other hand, a STAM would be able to support one, two, or all three of the above database classes.

3.2. Index Construction

Due to the aforementioned peculiarity of *t*-dimension and the nature of spatiotemporal data, we argue that an efficient STAM should take several issues into consideration during its construction, which extend relevant ideas that exist in pure spatial indexes. In the sequel, we discuss issues on efficient handling of new entries, 'obsolete' entries, and granularity change. Our discussion considers hierarchical tree structures since hashing methods do not seem to fit well to multidimensional indexing purposes.

- *Insert / Split operations:*

As soon as a new triplet (o-id, o_i, t_i) of an object o is inserted into the database, the root entries are checked in order to select the one that "fits" the new entry. At the next levels, the same procedure is repeated until the leaf level is reached. Then the entry is inserted into one of the leaf pages, with respect to a performance criterion. That should take spatial and temporal coordinates of the entry into account, however by keeping the *t*- axis peculiarity in mind. As usually, an insertion may cause a *page overflow*; in that case, appropriate handling of overflowing entries could result to either a split operation, or a forced reinsertion procedure, or even using overflow pages (e.g. *supernodes* in [3]). Total (e.g. similar to the R^+-tree split operation [21]) or partial disjointness (i.e., according to space or time coordinates) criteria would be considered to efficiently handle page overflow.

- *Pack / Purge operations to handle 'obsolete' entries:*

Index pages consisting of '*obsolete*' entries could be packed or purged in order to reduce disk space. Such reorganization techniques could be introduced in the design of a STAM due to the nature of the data involved. Under the assumption that the average page capacity of dynamic tree indexes is about 67%, the following '3-to-2' merge technique could be used: three consecutive 'obsolete' pages, each being around 67% full, would be merged into two full pages. In general, alternative 'n - to - n-1' merge techniques, with n being a tuned parameter, could be applied. Alternative to packing, by purging pages consisting of '*obsolete*' entries and removing them from the index organization a remarkable space saving could be achieved. The purge operation actually leads to unbalanced trees because several nodes are removed without affecting the rest of the index structure, and especially the pointers directing to those nodes. However, this situation is not harmful, as long as a search that encounter such a 'dangling' pointer will never follow it [2].

- *Change time-stamp granularity:*

Another case of reorganizing a STAM involves the *time-stamp granularity* (i.e., the unit of measure in time dimension). When the time-stamp granularity of a time dimension changes, e.g. from an hour to a day, then the underlying index should be reorganized in order to express objects' time-stamps according to the updated unit of measure.

3.3. Query Processing

The major objective a STAM is to efficiently handle query processing. The broader is the set of queries supported, the more applicable and useful the access method becomes. A set of fundamental query types as well as some specialized queries are discussed in the sequel.

- *selection queries:*

Queries of the form "*find all objects that have lied within a specific area (or at a specific point), during a specific time interval (or at a specific time instant)*" are expected to be the most common ones addressed by STDBMS users. Assuming a hierarchical tree structure, the retrieval procedure is straightforward: starting from the root node(s), a downwards traversal of the index is executed by applying the criterion of intersected intervals (for time) and ranges (for space) between the query window and each node approximation. It is important to point out that *pure temporal* or *pure spatial* selection queries need to be supported as well.

- *join queries:*

Queries of the form "*find all pairs of objects that have lied spatially close (i.e., within distance X), during a specific*

Specification		Explanation (domain)	Note
Spec1	*Data types supported*	`point` | `region`	(1)
Spec2	*Database classification I: with respect to time dimension(s) supported*	`valid-time` | `transaction-time` | `bitemporal`	
Spec3	*Database classification II: with respect to the data set mobility*	`growing` | `evolving` | `full-dynamic`	
Spec4	*Database classification III: with respect to time-stamp update*	`static` | `chronological` | `dynamic`	(2)
Spec5	*Specific object approximation*	`NO` | `YES`	
Spec6	*Handling 'obsolete' entries (e.g. support of bulk-loading / packing / purging operations)*	`NO` | `YES`	(3)
Spec7	*Specific (application-oriented) query processing operations*	`NO` | `YES`	
Notes: (1) `region` includes both point- and non-point objects (2) `dynamic` classification is meaningless in `transaction-time` databases (see Spec2) (3) meaningless in `dynamic` databases (see Spec4)			

Table 4: List of specifications for spatiotemporal indexing

time interval (or at a specific time instant)" are also crucial in spatiotemporal databases. An immediate application is accident detection by comparing vehicle trajectories. The retrieval procedure is also straightforward: starting from the two root nodes, a downwards traversal of the two indexes is performed in parallel, by comparing the entries of each visited node according to the *overlap* operator, such as the synchronized tree traversal proposed in [4] for R-tree structures.

- *nearest-neighbor queries:*

Given an object *X*, the nearest-neighbor query requests for the *k* closest objects with respect to *X*. For example the query: "*find the 5 closest ambulances with respect to the accident place*" is a nearest-neighbor query. Evidently, such a query can be supported by the algorithm proposed in [19]. However, consider the query: "*find the 5 closest ambulances with respect to the accident place in a time interval of 2 minutes before and after the accident, knowing the directions and velocities of ambulances and the street map*". Evidently, more sophisticated algorithms are required, towards spatiotemporal nearest-neighbor query processing.

According to the discussion in Section 2, other (application oriented) queries of interest include among others the *timeslice* (which can be transformed into 'partially-point' or range queries using appropriate query windows) and *history* (which could be supported by the maintenance of appropriate *to-next* pointers) operations. Although a formal set of spatiotemporal operators, in correspondence to the topological / directional models [6, 18] between spatial regions or Allen's relations [1] between temporal intervals, is lacking, proposals for spatiotemporal indexing should be able to support specific spatiotemporal operations, such as the ones described above or an extended list that appears in [7], together with the fundamental selection, join, and nearest-neighbor ones. In conclusion, Table 4 illustrates a classification scheme that applies for indexing methods on spatiotemporal data.

In the next section, methods proposed in the past to index spatiotemporal objects are presented and evaluated with respect to the list of Table 4.

4. A Survey of STAMs

Although spatiotemporal data handling is very important in modeling real world applications, research in spatiotemporal indexing is quite limited. The emphasis is given either in pure spatial indexing supporting multidimensional data or temporal indexing for conventional data types (e.g. numbers, strings). The efforts performed in the past towards spatiotemporal indexing can be classified in the following categories:

- *methods that treat time as another dimension* [27],
- *methods that incorporate the time information* into the nodes of the index structure, but without assuming another dimension [29], and
- *methods that use overlapping index structures* in order to represent the state of the database in different (valid or transaction) time instants [15, 29].

Let us trace through the advantages and disadvantages of the aforementioned approaches, and explain the reasons why they fail to solve the problem of spatiotemporal indexing to a sufficient degree.

Assuming that time is another dimension is a simple idea, since the tools to handle multidimensional data are already available (e.g. R-trees and variants [5, 9]). The 3D

R-tree implemented in [27] considers time as an extra dimension in the original two-dimensional space and transforms two-dimensional regions in three-dimensional boxes (MBRs). Figure 4, for example, illustrates (a) a set of spatial objects with a specific lifespan for each one and (b) the corresponding 3D R-tree. The retrieval of objects that fulfil a spatiotemporal range constraint ("*find all objects that overlap object D both in space and time*") is also illustrated and its implementation is a typical three-dimensional range query in the R-tree structure.

(a)

(b)

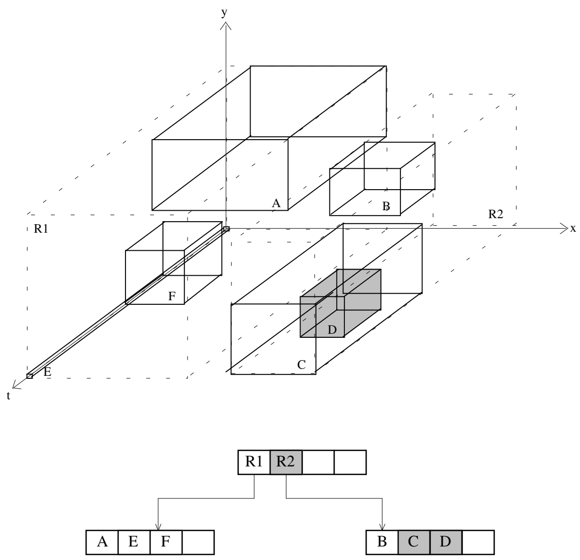

Figure 4: The three-dimensional MBRs stored in an R-tree structure.

Since the particular application considered in [27] (i.e., multimedia objects in an authoring environment) involves objects that do not change their location through time, no dead space is introduced by their three-dimensional representation. However, if the above approach were used for moving objects, a lot of empty space would be introduced, as illustrated in Figure 3.

The second approach, which has been followed in [29], is based on the incorporation of the time information, by means of time intervals, inside the structure. The effort leads to the RT-tree, which is a spatiotemporal version of an R-tree. Both the spatial and temporal information is maintained separately. Each entry, either in a leaf or a non-leaf node, contains entries of the form (S, T, P), where S is the spatial information (MBR), T is the temporal information (interval), and P is a pointer to a subtree or the detailed description of the object. Let $T = (t_i, t_j)$, $i \leq j$, t_j be the current time-stamp and t_{j+1} be the consecutive one. If an object does not change its spatial location from t_j to t_{j+1}, then its spatial information S remains the same, whereas the temporal information T is updated to T', by increasing the upper value of the interval, i.e., $T' = (t_i, t_{j+1})$. However, as soon as an object changes its spatial location, a new entry with temporal information $T = (t_{j+1}, t_{j+1})$ is created and inserted into the RT-tree. The most important limitations of this approach are listed below:

- Evidently, if we assume that the number of objects that change is large, then many entries are created and the RT-tree grows considerably. Therefore the structure is preferred for databases of low mobility.
- Since the instances of the objects that correspond to several time-stamps are maintained in a single tree and are not separated, queries that focus on a specific time-stamp face the overhead of the rest ones.
- When a node overflows, a decision must be taken in order to choose the characteristics where the split will be based. The spatial or temporal or both characteristics can be used, but no proposal has been presented on when to use each approach.

The research efforts of the third category include the MR-tree and HR-tree, which are influenced by the work on overlapping B-trees [14]. Both methods support the following approach: different index instances are created for different transaction time-stamps. However, in order to save disk space, common paths are maintained only once, since they are shared among the structures. The collection of structures can be viewed as an *acyclic graph*, rather than a collection of independent tree structures. The concept of overlapping tree structures is simple to understand and implement. Moreover, when the objects that have changed their location in space are relatively few, then this approach is very space efficient. However, if the number of moving objects from one time instant to another is large, this approach degenerates to independent tree structures, since no common paths are likely to be found. Figure 5 illustrates an example of overlapping trees for two different time instants t_0 and t_1. The dotted lines represent links to common paths / subpaths.

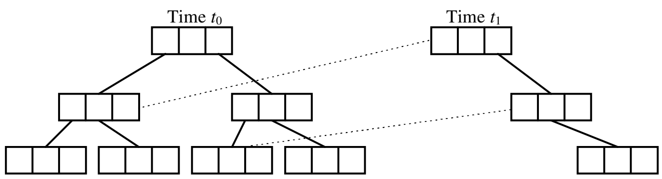

Figure 5: Overlapping trees for two different time instants t_0 and t_1.

Among the aforementioned proposals, it is only the 3D R-tree that has been implemented and experimentally tested [27]. The so-called *unified scheme* was evaluated in comparison with a simple scheme that separates temporal and spatial information of objects by maintaining two different indexes (e.g. one 1D or segment R-tree and one 2D R-tree, respectively). The retrieval cost for several pure temporal, pure spatial and spatiotemporal operators was measured and appropriate guidelines were extracted. An example of the latter case includes the '*layout*' query (a kind of timeslice operation), where all objects' instances at

STAM / Specification	MR-tree	RT-tree	3D R-tree	HR-tree
Spec1: *Data types supported*	region	region	region	region
Spec2: *Type of time supported*	transaction-time	transaction-time	valid-time	transaction-time
Spec3: *Data set mobility*	full-dynamic	full-dynamic	growing	full-dynamic
Spec4: *Time-stamp update*	chronological	chronological	static	chronological
Spec5: *Specific object approximation*	YES	YES	NO	YES
Spec6: *Handling 'obsolete' entries*	NO	NO	NO	NO
Spec7: *Specific query processing operations*	YES (timeslice)	NO	YES (timeslice)	YES (timeslice)

Table 5: Evaluation of existing STAMs

a specific instant t = T have to be retrieved. No experimentation of the other methods has been performed, to the best of the authors' knowledge; only simple analytical considerations of best and worst case for the storage space of MR-trees and RT-trees can be found in [29]. Therefore, there is a lack of performance comparison among the proposed approaches, with respect to the space occupied, the construction time, and the response time in order to answer a variety of spatiotemporal queries.

Table 5 classifies the methods that have been proposed for spatiotemporal indexing purposes. The classification is based on the specification list proposed in Section 3. Recall that it is not claimed to be a *complete set of specifications* in any sense but a (open to additions) list of criteria, which should be supported in order to meet the needs of efficient spatiotemporal query processing.

In particular, all methods index regions (including points as a special case) and three out of four methods support transaction-time (although it is not declared in [29]). Apart from that, the 3D R-tree implementation in [27] handles *growing* and *static* databases (since objects are accompanied by a valid-time lifespan without, however, changing their location in space and all objects' instances are known in advance) while the rest proposals are classified as *full-dynamic* (with respect to the data set mobility) and *chronological* (with respect to time-stamp updates). Concerning the rest specifications, the 3D R-tree takes no care of specific approximations, since each object is represented by its MBR, but implements a specific spatiotemporal operation (namely, the *layout* query, which is a kind of a timeslice operation). On the other hand, overlapping trees, such as the MR-tree and the HR-tree, maintain a set of MBRs per object, although with no links to each other, and efficiently handle timeslice operations due to their design specifications (although in [15, 29] the typical point and range queries are only discussed).

As extracted from Table 5, although all proposals aim at organizing time-evolving spatial objects in an efficient manner, a limited support of specific spatiotemporal operations (Spec7) or the lack of specific object approximations other than MBRs (Spec5) is the general rule. In addition to that, although all methods support static or chronological databases (Spec4), no special handling of 'obsolete' object instances (Spec6) is considered. Therefore, it is claimed that the existing methods should be extended and / or revised and, moreover, new proposals are needed.

5. Conclusion

Spatiotemporal data support is considered to be an important research direction, since many applications need to manipulate data that change over time. STDBMS, in particular, should (i) offer appropriate data types and query languages for time-evolving spatial objects, (ii) provide efficient indexing techniques and access methods for spatiotemporal query processing, and (iii) exploit cost models for query optimization purposes. Recent research on modeling and querying includes temporal elements attached to components of spatial objects [28], alternative continuous or discrete models of moving points or regions [7], or motion vectors that describe the current status of moving objects [24].

The specific needs of STDBMS users also require appropriate indexing techniques on spatiotemporal data. Although conceptually the problem seems to be easy to solve, several issues arise when one attempts to design an indexing method for a d-dimensional spatiotemporal database. Due to the peculiarities of the time dimension, the workspace of interest cannot be directly considered to be just a d+1 – dimensional one.

In this paper, we have studied several discussion points and we have presented a list of specifications that an efficient STAM should follow. Issues on (a) data types and data sets supported, (b) index construction, and (c) query processing operations are addressed and existing proposals [15, 27, 29] are evaluated according to the above issues. To our knowledge, it is the first attempt towards a *specification and classification scheme* for STAMs. The main conclusion is that the existing methods do not follow the full list of specifications proposed. Thus they should be extended and / or revised while new proposals should be investigated.

We are currently working on the design and implementation of spatiotemporal indexing schemes in order to evaluate them under real experimentation. More specifically, since up to now schemes based on the R-tree only have been proposed, we are working towards making Overlapping B+trees [14] and the multiversion B-tree (MVBT) structure [2] suitable for storing quad / octcodes and perform experiments for various queries. A second task of current work includes the design and implementation of a benchmarking environment in order to provide performance comparison of the existing STAMs. A variety of tests using synthetic and real spatiotemporal data sets are necessary in order to better understand the spatiotemporal indexing and retrieval issues. Also, similarity-based queries are of importance in spatiotemporal databases, and, therefore efficient tailor-made access methods for similarity retrieval have to be proposed. As a further task of research, the study of cost models for spatiotemporal operations, which can be based on analytical performance cost formulae for spatial query processing [17, 25, 26], is a step towards a complete set of appropriate STDBMS support tools.

Acknowledgements

Research partially supported by the European Commission funded TMR project "*CHOROCHRONOS: A Research Network for Spatiotemporal Database Systems*", contract number ERBFMRX-CT96-0056, URL address: http:// www.dbnet.ece.ntua.gr/~choros/. Also, partial support from national PENED and EPET projects. Many thanks to David Kerr, Mario Nascimento, and Michael Vazirgiannis for their comments on earlier drafts of the paper.

References

[1] J. F. Allen, "Maintaining Knowledge about Temporal Intervals", *Communications of ACM*, vol. 26(11), pp. 832-843, 1983.

[2] B. Becker, S. Gschwind, T. Ohler, B. Seeger, P. Widmayer, "An Asymptotically Optimal Multiversion B-tree", *The VLDB Journal*, vol.5(4), pp. 264-275, December 1996.

[3] S. Berchtold, D. A. Keim, H.-P. Kriegel, "The X-Tree: An Index Structure for High-Dimensional Data", *Proceedings of the 22nd Conference on Very Large Data Bases* (VLDB), 1996.

[4] T. Brinkhoff, H.-P. Kriegel, B. Seeger, "Efficient Processing of Spatial Joins Using R-trees", *Proceedings of ACM SIGMOD Conference*, 1993.

[5] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", *Proceedings of ACM SIGMOD Conference*, 1990.

[6] M. J. Egenhofer, R. Franzosa, "Point Set Topological Relations", *International Journal of Geographical Information Systems*, vol. 5, pp. 161-174, 1991.

[7] M. Erwig, R. H. Gutting, M. Schneider, M. Vazirgiannis, "Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases", Technical Report, CHOROCHRONOS-TR-97-08, September 1997.

[8] V. Gaede, O. Gunther, "Multidimensional Access Methods", Technical Report ISS-15, Humboldt-University of Berlin, Germany, August 1995. Address for downloading: `http://www.wiwi.hu-berlin.de/~gaede/survey.rev.ps.Z`.

[9] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching", *Proceedings of ACM SIGMOD Conference*, 1984.

[10] C. S. Jensen, J. Clifford, R. Elmasri, S. K. Gadia, P. Hayes, S. Jajodia (ed.), "A Consensus Glossary of Temporal Database Concepts", *ACM SIGMOD Record*, vol.23(1), pp. 52-64, 1994.

[11] C. Kolovson, M. Stonebraker, "Indexing Techniques for Historical Databases", *Proceedings of ACM SIGMOD Conference*, 1989.

[12] S. Lanka, E. Mays, "Fully Persistent B+-trees", *Proceedings of ACM SIGMOD Conference*, 1991.

[13] D. Lomet, B. Saltzberg, "Access Methods for Multiversion Data", *Proceedings of ACM SIGMOD Conference*, 1989.

[14] Y. Manolopoulos, G. Kapetanakis, "Overlapping B^+-trees for Temporal Data", *Proceedings of the 6^{th} JCIT Conference*, 1990.

[15] M. A. Nascimento, J. R. O. Silva, "Towards Historical R-trees", *Proceedings of ACM Symposium on Applied Computing* (ACM-SAC), 1998.

[16] J. Orenstein, "Redundancy in Spatial Databases", *Proceedings of ACM SIGMOD Conference*, 1989.

[17] A. N. Papadopoulos, Y. Manolopoulos, "Performance of Nearest Neighbor Queries in R-trees", *Proceedings of the 6th International Conference on Database Theory* (ICDT), 1997.

[18] D. Papadias, Y. Theodoridis, "Spatial Relations, Minimum Bounding Rectangles, and Spatial Data Structures", *International Journal of Geographic Information Science*, vol. 11(2), pp. 111-138, March 1997.

[19] N. Roussopoulos, S. Kelley, F. Vincent, "Nearest Neighbor Queries", *Proceedings of ACM SIGMOD Conference*, 1995.

[20] H. Samet, "The Quadtree and Related Hierarchical Data Structures", *ACM Computing Surveys*, vol. 16(2), pp. 187-260, 1984.

[21] T. Sellis, N. Roussopoulos, C. Faloutsos, "The R^+-tree: A Dynamic Index for Multi-Dimensional Objects", *Proceedings of the 13th International Conference on Very Large Data Bases* (VLDB), 1987.

[22] E. Stefanakis, T. Sellis, “Towards the Design of a DBMS Repository for Temporal GIS”, *ESF-GISDATA: Time and Motion of Socio-Economic Units*, Taylor & Francis, 1998.

[23] B. Salzberg, V. Tsotras: "A Comparison of Access Methods for Time Evolving Data", to appear in *ACM Computing Surveys*. Address for downloading: `ftp://ftp.ccs.neu.edu/pub/people/salzberg/tempsurvey.ps.gz`.

[24] A. P. Sistla, O. Wolfson, S. Chamberlain, S. Dao, "Modeling and Querying Moving Objects", *Proceedings of the 13th IEEE Conference on Data Engineering* (ICDE), 1997.

[25] Y. Theodoridis, T. Sellis, "A Model for the Prediction of R-tree Performance", *Proceedings of the 15th ACM Symposium on Principles of Database Systems* (PODS), 1996.

[26] Y. Theodoridis, E. Stefanakis, T. Sellis, "Cost Models for Join Queries in Spatial Databases", *Proceedings of the 14th IEEE Conference on Data Engineering* (ICDE), 1998.

[27] Y. Theodoridis, M. Vazirgiannis, T. Sellis, "Spatio-Temporal Indexing for Large Multimedia Applications" *Proceedings of the 3rd IEEE Conference on Multimedia Computing and Systems* (ICMCS), 1996.

[28] M. F. Worboys, "A Unified Model for Spatial and Temporal Information", *The Computer Journal*, vol. 37(1), pp. 26-34, 1994.

[29] X. Xu, J. Han, W. Lu, "RT-tree: An Improved R-tree Index Structure for Spatiotemporal Databases", *Proceedings of the 4th International Symposium on Spatial Data Handling* (SDH), 1990.