Efficient Index Structures for Spatio-Temporal Objects

Carsten Kleiner, Udo W. Lipeck
Universität Hannover, Institut für Informatik, Lange Laube 22, 30159 Hannover, Germany
E-mail: ckul@informatik.uni-hannover.de

Abstract

In this article we present a family of four tree-based access structures for indexing spatio-temporal objects. Our indexing methods support spatio-temporal, as well as purely spatial and purely temporal queries. In order to handle sets of extended spatio-temporal objects we propose to specialize generalized search trees by combining the advantages of the well-known spatial structures R*-tree ([1]) and SS-tree ([18]). We consider size-based (R*-tree like) and distance-based (SS-tree like) penalty metrics for insertions, and we view the temporal dimension either as a regular third or as a special dimension. We evaluate the four access methods on different real-life datasets and identify one of them to be the most efficient access structure for the case of general spatio-temporal data with known extents in every dimension. This method combines the R*-tree split policy with penalty metric and insertion policy from the SS-tree and treats the temporal dimension as a special dimension.

1. Introduction

A spatio-temporal database system tries to solve the task of modeling and storing real-world data in order to support solving real-world problems. Among the topics in need for support by a spatio-temporal database system are multimedia, geographic and cartographic applications. Most of these applications require to store geometric objects with temporally varying geometric location. These objects can have between 0 and 3 dimensions. Moreover for the temporal dimension one may want to store transaction time or valid time ([8], [13]) information for each object and its position in space. Since databases storing these kind of data can become extremely large even for rather simple applications there is a need to store and query those data efficiently. Our emphasis is on querying spatial as well as temporal properties.

Among the multiple topics of research for spatio-temporal data, e.g. query languages or modeling of spatio-temporal objects ([2]), this article focuses on efficient storage and access structures. Much work has been done in developing efficient storage and access structures for real-world objects. But most papers focus either on the spatial attributes (see [3] for a survey) or on the temporal ones (see [12] for issues in that context). There are only very few papers dealing with spatio-temporal access methods (e.g. [10], [11], [16]). Most of these papers focus on storing and querying moving points. In contrast, in this article we want to investigate methods for 2D-regions that change their location and size over time. Unlike [2] or [9] we focus on discretely moving objects; that means objects change their shape, size or location only at distinct points in time. To the best of our knowledge no performance results on access structures for this class of applications have yet been published.

This restriction is commonly used, see [11], and it fits our application needs very well, since data can usually only be recorded, and thus changes detected, at certain points in time. Our choice of focus was motivated by applications from cartography and physical geography which are described in the full article. Thus we confined ourselves to study storage and access structures for two-dimensional polygons whose geometry is valid over a certain timespan.

The remainder of the paper is organised as follows: in section 2 we detail and formally define the supported predicates and introduce the developed access structures in detail. Easy comparison is provided since all methods are based on the generalized search tree (GiST) approach ([6]). In Section 3 we describe the data used for the tests and the test environment. In Section 4 we present the results for the performance comparisons and discuss conclusions to be drawn. Finally in section 5 we give a summary and hints for further improvements and research in this area.

2. Access Structures and Embedding into GiST

The goal of this paper is to develop an efficient storage and access structure for two-dimensional extended objects that change their geometry over time. For the application at hand there was no need to include a transaction time dimension as is commonly proposed in the temporal database
literature (e. g. [7], [8], [12] and [13]). Thus we only have to model one temporal dimension (valid-time). Moreover objects are assumed to change their geometry at discrete points in time; the method does not support continuously changing geometries.

We specialize the GiST approach ([6]) of generalized search trees for our purposes. Our basic data organization is R-tree ([5]) like as in the original GiST paper since we deal with sets of extended objects. But we use two insertion policies (one as in the $R^*$-tree and one as in the SS-tree) which are induced by different penalty metrics. As earlier research in spatial databases has shown ([3],[17]) the original GiST should be extended to support forced re-inserts as documented by the often observed superiority of the $R^*$-tree over the regular R-tree for pure spatial access methods.

The details of forced reinsert can be found in [1] and are assumed here.

2.1. Supported Query Predicates

The objects to be stored in the $STT$ are represented by their spatial bounding box (as commonly used for spatial databases) and their valid time interval. Formally we will be storing 6-tupels in the tree of the form $STO = (x_{ll}, y_{ll}, x_{ur}, y_{ur}, t_{start}, t_{end})$ representing a spatio-temporal object with $(x_{ll}, y_{ll})$ as lower left corner of the (spatial) bounding box, $(x_{ur}, y_{ur})$ as upper right corner of the (spatial) bounding box and valid time from $t_{start}$ up to $t_{end}$.

Now we have to declare what predicates will be supported by the $STT$. The goal should be to efficiently support both real spatio-temporal predicates as well as pure spatial and pure temporal predicates. One can imagine many important predicates in each of these categories. In this article we restrict ourselves to the most important predicates required by our applications; these are overlap$(STO_1, STO_2)$, contains$(STO_1, STO_2)$, contained$(STO_1, STO_2)$ and equal$(STO_1, STO_2)$. The pure spatial versions (name preceded by $sp$) and pure temporal versions (name preceded by $t$) are also supported. The semantics of the predicates should be obvious from their names, contains$(STO_1, STO_2)$ means $STO_1$ contains $STO_2$ and contained$(STO_1, STO_2)$ similarly $STO_1$ is contained in $STO_2$. A couple of other predicates can be derived from these by the user, others could be easily incorporated by analogy. But representative predicates for each of the temporal as well as spatial and spatio-temporal groups identified by [12] are included, so that validity of the results should be sufficiently general.

A search key in the tree indexing the spatio-temporal objects corresponds to the predicate contains() as described in the original GiST-article for spatial objects. Here the key domain consists of the two spatial dimensions and one temporal dimension. In section 2.3 we will develop different access structures for interpreting these dimensions as standard three-dimensional space or as two-plus-one-dimensional space. The GiST approach enables us to reuse all methods for both approaches except for the part defining the penalty metric for the insertion policy.

2.2. Definition of the required GiST-Methods

In order to implement a generalized search tree for a concrete specialization one needs to define the following methods: consistent$(E,q)$, union$(P)$, pickSplit$(P)$, penalty$(E_1, E_2)$\footnote{compress$(E)$ and decompress$(E)$ are identities here}. The terminology for the method arguments follows the terminology in the original article ([6]): $P$ represents a set of node entries, $E$ represents one single node entry and $q$ stands for an arbitrary supported query predicate. Three of those methods are equal for all of our access structures and are described briefly in the sequel, while the penalty$(E_1, E_2)$-method differs and is described at the end of this section.

The method consistent$(E, q)$, which guides search through the tree to answer a query $q$ (which can be any of the 12 predicates discussed in the previous section) for a spatio-temporal object $STO_q$, returns a boolean value which tells, if there might be answers to $q$ below the current entry $E$. For interior, non-leaf nodes we use overlap for a contained- or overlap-query and contains for a contains- or equal-query. The correctness of this definition is obvious; the same principle was used for the R-tree extension of GiST in a spatial domain ([6]). For pure spatial and pure temporal queries we use the spatial and temporal versions of the predicates, respectively. If a node $E$ is on the leaf level we always apply the search predicate $q$ directly.

The method union$(P)$ can be easily defined to return a spatio-temporal object consisting of the spatial bounding rectangle and the temporal bounding interval for all entries in $P$. The computation can be done by a simple min and max scan through all entries $E_i$ of $P$ for all dimensions.

Finally for the pickSplit$(P)$-method we proceed as follows: along each of the possible split axes ($j \in x, y, t$) (where $x$ and $y$ denote the two spatial and $t$ the temporal dimension) we sort the entries of $P$ by lower value (ties by upper value) along that axis; then for every possible partition $\{E_1, \ldots, E_i\}, \{E_{i+1}, \ldots, E_{M+1}\}$ of the $M+1$ entries ($M$ is the maximum, $m$ the minimum number of entries per node) of $P$ we compute

$$mMeasure(P,j) = \sum_{i=m}^{M+1-m} \text{margin}(E_1 \cup \ldots \cup E_i) + \text{margin}(E_{i+1} \cup \ldots \cup E_{M+1})$$
We use the axis with minimal value for mMeasure as split axis. Along that split axis we compute the overlap (oMeasure) for every partition and choose the partition with minimal oMeasure. In case of ties we use minimal size (sMeasure) to break the tie. These measures are defined (for $i = m, \ldots, M + 1 - m$) by:

$$oMeasure(P(i)) = \text{size}(STO(E_1 \cup \ldots \cup E_i) \cap STO(E_{i+1} \cup \ldots \cup E_{M+1}))$$

$$sMeasure(P(i)) = \text{size}(STO(E_1 \cup \ldots \cup E_i)) + \text{size}(STO(E_{i+1} \cup \ldots \cup E_{M+1}))$$

For the definition of size see section 2.3. The distribution with minimal measure as described above induces the split size of the entries which means with minimal measure as described above induces the split size which is closest (in the Euclidean distance sense) to the midpoint of the object to be inserted. This penalty is applied on all levels of the tree.

2.3. Generalization of Area, Margin, Overlap and Distance for Spatio-Temporal Objects

The four different access methods presented depend on the spatial terms area, margin, overlap and distance which need to be appropriately generalized to spatio-temporal space. We suggest to consider a three-dimensional and a two-plus-one-dimensional generalization; we denote the former by index 3d and the latter by index 2+1d. We usually speak of size rather than area in the following to indicate the change from spatial to spatio-temporal space.

Three-Dimensional Generalization

For the three-dimensional case we simply extend the common measures from two to three dimensions. We obtain:

$$size_{3d}(STO) = (x_{ur} - x_{ll}) \cdot (y_{ur} - y_{ll}) \cdot (t_{end} - t_{start})$$

$$overlap_{3d}(STO_1, STO_2) = size_{3d}(STO_1 \cap STO_2)$$

$$margin_{3d}(STO) = 4 \cdot (x_{ur} - x_{ll} + y_{ur} - y_{ll} + t_{end} - t_{start})$$

$$dist_{3d}(STO_1, STO_2) = \sqrt{\sum_{j \in \{x,y,t\}} dDist(STO_1, STO_2, j)^2}$$

where dDist(STO_1, STO_2, j) denotes the distance in dimension j of the midpoints of STO_1 and STO_2. Using these definitions together with the methods described in the previous section we obtain an access method called a spatio-temporal tree, either together with the size-based penalty metric (called STT^{size}_3) or with the distance-based penalty metric (called STT^{dist}_3). The STT^{size}_3-tree can be viewed as regular R*-tree in three dimensions.

Two-Plus-One-Dimensional Generalization

In other papers on spatio-temporal data (e.g. [14], [15]) authors have stated that a representation of time as just another dimension (like a third spatial dimension) is not appropriate. One of the reasons for this is the fact that spatial proximity introduced by using the time as in the previous subsection does not reflect the orthogonal nature of this dimension properly. There are more sophisticated reasons (e.g. inefficient use of space for moving points) for treating the time completely separately. For that reason we generalized the measures in a two-plus-one-dimensional way also. For this we obtain the following definitions:

$$size_{2+1d}(STO) = (x_{ur} - x_{ll}) \cdot (y_{ur} - y_{ll}) + (t_{end} - t_{start})^2$$

$$overlap_{2+1d}(STO_1, STO_2) = \frac{area(spatial(STO_1) \cap spatial(STO_2))}{area(spatial(STO_1) \cap spatial(STO_2))} + length(temporal(STO_1) \cap temporal(STO_2))^2$$
margin \(2+1d\) (STO) = border(spatial(STO)) + 2 \cdot length(temporal(STO))

dist \(2+1d\) (STO, STO\(_2\)) = ddist(STO, STO\(_2\), t) + \sqrt{ddist(STO, STO, x)^2 + ddist(STO, STO, y)^2}

In these definitions spatial(STO) denotes the spatial and temporal(STO) the temporal part of a spatio-temporal object. In order to combine the 2D-spatial with 1D-temporal measures we used simple addition of these measures while observing dimensionality, e.g. in computation of overlap \(2+1d\) we need to add the square of the temporal length to the spatial area. We assume that the measures in pure-spatial and pure-temporal dimensions have to be appropriately scaled before being combined with one another. One could e.g. scale all values from the domain to a fixed range by a simple multiplication. The fact if the extent in every dimension is known exactly in advance has an influence on the performance of the queries as we will see in section 4.4.

Similar to the previous section we obtain the STT\(_{size}\) \(2+1d\)-tree by using these generalizations together with the size penalty metric and finally the STT\(_{dist}\) \(2+1d\)-tree using them together with the distance penalty metric.

3. Test Environment

We have implemented the tree structures for access support described in the previous section in Java (JDK 1.2). The implementation was based on a GiST implementation by the University of Cape Town which includes forced reinsert.

Hardware and Technical Environment

We assumed the page sizes to be 1024 Bytes (resulting in \(M = 20\) maximum entries per node in our implementation). There was no caching used in order to provide for a fair comparison of the four tree structures.

We ran overlap-queries of different selectivity on our test data with queries taken from spatio-temporal overlap, pure spatial and pure temporal overlap. We counted the number of node accesses necessary to answer the query and also the number of entries that needed inspection. The former should be a measure proportional to disk I/O-time since a new page needs to be loaded for every node. The latter should measure CPU time as all entries are in main memory after a node is loaded. We also monitored total elapsed time. Our experiments showed that computation time is negligible compared to I/O time as most articles on this subject assume. Total time was proportional to the number of nodes and can therefore also be omitted.

Data used for Generating Test Results

For our tests we used real polygon data from the German topographic-cartographic information system (ATKIS) of parts of the Hannover region\(^3\) as well as polygons from the biotope-database in [4]. These polygons were transformed to spatio-temporal data by an algorithm derived from [14]. Each polygon changed its shape at discrete points in time which were drawn from a uniform distribution over the whole time domain.

For the ATKIS data objects were relatively large polygons and covered the spatial domain almost completely. They had few spatial overlap, but after the generation of moving regions they had substantial overlap in spatio-temporal space. The biotope data set differs structurally from the first in containing relatively small rectangles covering only a small part of space and having virtually no overlap. Total sizes of the datasets were 121676 spatio-temporal objects for the Hannover dataset and 127420 spatio-temporal objects for the biotopes dataset.

Each spatial object was assumed to exist over the whole time interval with changing geometry to generate the spatio-temporal objects. We also experimented with fewer coverage of the temporal dimension by each spatial object. The results there were the same as reported in section 4 but less significant. Thus to stress our findings we used complete coverage of the time interval used. From an application point of view this is reasonable since areas in the real world usually do exist for a long time but with changing geometry.

4. Test Results and Conclusions

In this section we discuss test results for the different types of queries and conclude with some general remarks applying to all of the query types.

4.1. Spatio-Temporal Query

In the first group of experiments we used actual spatio-temporal overlap queries with selectivities between 0.001% and 1%. These values were chosen small compared to traditional spatial access methods, but that seems reasonable for the application since one spatial object appears with different geometries at different times and might therefore not be queried as often as in pure spatial databases. In the subsequent sections on pure spatial and pure temporal queries we increased the selectivities to values used in previously published research. In figure 2 we show the number of nodes accessed to answer the queries.

The access trees based on the distance penalty metric clearly outperform those based on size in all selectivities. In addition to that the setup time for those trees is about one third to one fourth of that of the size-based trees since

\(^3\)Source: ATKIS®-DLM25-Daten der LGN - Landesvermessung + Geobasisinformation Niedersachsen
the penalties are far easier to compute. The size-based tree with the 2+1d metric is particularly bad for the biotope data. This is due to the different coverages of the dimensions in that data set: whereas the spatial objects are relatively sparse the temporal dimension is very densely populated.

Between the two distance-based access methods there is no significant difference so that both could be chosen for an application. The superiority of these methods to the size-based methods seems to be due to a better subdivision of space by these methods. As we will see in section 4.4 the size-based methods favor index entries with small extent in two dimensions and large extent in the third (stick-shaped objects) by the penalty definition. This results in a bad division of space since the axis having the largest extent is mostly chosen as split axis. The distance-based method favors clustering of the objects resulting in cube-shaped index entries. During the split method this causes every dimension to be chosen as split axis with comparable probability.

The difference between the methods gets less significant with increasing selectivity as the difference in nodes is almost constant with increasing absolute numbers. Overall the number of nodes accessed is very small and thus the indexing method is very efficient (e.g. the STT_{size 3d}-tree inspects only 222 entries out of 121676 to answer a query with selectivity 0.01% resulting in 13 result objects).

4.2. Pure Spatial Query

In the second group of experiments we tried to evaluate the ability to adapt to pure spatial queries as in spatial database systems. This is also an important measure as an access method should be as generally applicable as possible and moreover even spatio-temporal databases might start off their lives with pure spatial data which are gaining temporal diversity over time. Thus the performance of the access structures for spatial queries is also very important.

As in the previous subsection the distance-based methods outperform the size-based methods. But the difference is not so significant, especially for the dataset with relatively large coverage and overlap (Hannover Dataset) there is no big difference anymore. Between the two distance-based methods there is a slight edge towards the 2+1d-tree whereas in the size-based the 3d-tree is much more efficient than its 2+1d counterpart. The smaller difference between the methods may be due to the size-based-tree using x- or y-axis as split axis more often than the time axis and thus it gets a better division of space for this category than in the previous section. Overall the number of nodes accessed is slightly larger than in the previous section for comparable selectivities (which could be expected as the temporal subdivisions do not give any help for answering pure spatial queries). The only exception is the STT_{size 3d}; this can be attributed to the observed favouring of choosing x- or y-axis as split axis. In conclusion one should use the STT_{dist 2+1d} as indexing method for a spatio-temporal application with a large amount of pure spatial queries.

4.3. Pure Temporal Query

Finally we also evaluated the performance of the different tree structures on pure temporal queries. This measure is important because spatio-temporal data may be queried to obtain the databases state at a certain timestamp or in a certain interval. This is needed for valid-pure-timeslice-queries (Show all geometries in the area in December 1956) or for valid-range-queries (Display all streets that existed from 1970 to 1979). Therefore a spatio-temporal access structure should also support pure temporal queries.

The distance-based access methods again clearly outperform the size-based ones. Within the distance-based meth-
ods there is no big difference with a small edge towards the 2+1d-tree. Within the size-based methods this time also the 2+1d-tree performs better; this seems to be due to the different definition of the size measures leading to a more favourable situation for the time axis being chosen as split axis, which in turn leads to a better division of the temporal dimension. Therefore if using the size-based trees one should consider the 3d variant, if more pure spatial queries are asked and the 2+1d variant, if more temporal queries need to be supported.

The number of nodes accessed was largest in this category for comparable selectivities. This can be explained with the fact that only one of three dimensions is queried for and thus the penalties and subdivisions used for the other two (spatial) dimensions do not help in answering queries. Still the methods are very efficient since even for 18% selectivity we need only inspect about 30% of the entries using a STT\textsuperscript{dist}\_2\_1\_d or STT\textsuperscript{dist}\_3\_d-tree. Overall one should use the STT\textsuperscript{dist}\_2\_1\_d-tree as access method for applications requiring many pure temporal queries.

4.4. General Results

In general the STT\textsuperscript{dist}\_2+1d proved to be the most efficient access method for regions with changing geometry over time in spatio-temporal applications. The experiments also showed that all the access structures were very efficient methods as the number of objects to be checked for answering queries was very small compared to the size of the dataset. The only method with serious performance problems in some cases (but still far more efficient than naive searching) was the STT\textsuperscript{size}\_2+1d-tree which should thus be used with care.

The superiority of the distance-based penalty metric over the size-based penalty metric is due to its better subdivision of the whole domain. Since all access methods use the same split method one may think at first glance that they should equally subdivide space. But by going into deeper details one sees that the different penalty metrics cause different distributions of the spatio-temporal objects over the tree. This results in nodes having different shapes before being split. Thus the same split method produces different divisions of space in the different methods. In particular the size-based penalty metric by using straight multiplication of size enlargements in each of the three dimensions favors nodes to have a stick-like shape: small extents in two and large extent in the third dimension. This has a stronger impact for three-dimensional spatio-temporal objects than it had in pure spatial access methods because of using multiplication in the penalty metric definition. This metric was optimal in the original R*-tree, but for the spatio-temporal domain this leads to choosing mainly one axis as split axis (the one which has largest extent in the original node) resulting in a poor subdivision of space for the other dimensions. The distance-based penalty on the other hand treats all three dimensions equally by clustering the objects in the nodes based on minimizing their distance to the nodes centroid. This leads to cube-like shapes in the nodes which in turn causes the split method to split along each of the three axis with almost equal probability. This claim is further enforced by the fact that the difference between the two penalty metrics is larger for the Biotope database (see figures 2 and 4): since objects are smaller and have small overlap the clustering works even better compared to the large objects in the Hannover dataset. The size-based method is more likely to produce stick-like index entries on the small objects since they only have small extent whereas large objects help to avoid stick-like entries already due to their shape. Thus the distance-based index trees finally show a very good division.
of space leading to better query performance.

In the tests whose results were shown in the previous subsections we had the advantage of knowing the extent of the data space in each dimension exactly. In real applications, especially if using very dynamic data structures, one would not know the total extent of data to be stored. This could pose a problem since scaling is needed in order to be able to combine temporal and spatial measures. If the extent is known only approximately, scaling can also only be done approximately.

In other experiments we examined the behaviour of the different structures under varying scaling factors. We discovered that the denser a dimension is populated the more weight should be given to it (which can be done by adjusting the scaling factor) in the distance-based trees. In the data used in this chapter the temporal dimension is more densely populated than the spatial dimensions (this is reasonable for our applications since objects usually change their extents and do not cover the whole geographic space, but exist during the whole time period). We were even able to improve the efficiency of the access methods by overscaling the temporal dimension. That means we gave it a heavier weight than given by the absolute coordinate values. Thus in more static environments one should consider using such a technique to obtain even more efficient structures. On the other hand in extremely dynamic environments, maybe with virtually no bounds on the spatio-temporal values, one should consider using a size-based tree instead of a distance-based tree. In this area more research is necessary to obtain further insight and also to formally prove these observations.

Finally we also used some of the other query predicates. The results were similar to the reported results in all categories. Therefore we omit these results here.

5. Summary and Future Research

5.1. Summary

In this article we developed four different variants of spatio-temporal trees as access structures to efficiently support querying regions with changing geometries over time. We formally defined the query predicates supported by the access structures which were based on the known R*-tree and SS-tree access methods for spatial data. We also put the known methods into a broader context by implementing them as special instantiations of the generalized search tree index structure. This has the advantage of easy extendibility and also facilitates comparisons very well. After formally defining the metrics used in our spatio-temporal access methods we presented the results of detailed tests on different datasets. We compared the different methods and stated that for most applications the STT_{dist}^{\text{2+1d}}-tree will be the most suitable access structure. We also named situations in which one can expect other methods to work better.

5.2. Future Research

A couple of open problems in this context remain to be solved. Firstly our as well as experiments by other researchers (e. g. [17]) showed the importance of the penalty metric on the performance of queries with spatial and as in our case with spatio-temporal index structures. In the spatio-temporal context we also observed a great importance of scaling values on the performance of access methods; since we need to compute a single penalty value to compare different bounding spheres we have to combine a spatial measure with a temporal measure. This can be done by assigning certain weights or by optimizing the combina-
tion of spatial and temporal measures using different definitions (e.g. norms) in our two-plus-one-dimensional generalization. As a first step our approach led to pretty good results but still leaves room for further investigation and improvement.

Another interesting aspect that definitely needs more attention in the future is the embedding of our access method into a commercial database management system. The tests were run on a simulation of those access structures in main memory (including swap space). The real strength of the methods is their good support for queries on externally stored data. Thus results of query performance in a database system or geographic information system would be very interesting. Whereas database systems, in particular object-relational systems, nowadays have become more easily extensible (we currently experiment with the extensible indexing interface of Oracle 8i), GIS have to undergo a substantial development before being comparably easy to extend. A current trend in GIS seems to go into the direction of using commercial DBMS as datastore within GIS which in turn would make results of extending indexing in databases very important. We are currently working on incorporating the proposed access methods into Oracle and are looking forward to evaluating their performance.

References


