

# Indexing Temporal Data with Virtual Structure

Bela Stantic Justin Terry Rodney Topor Abdul Sattar

Institute for Integrated and Intelligent Systems  
Griffith University, Brisbane, Australia  
{B.Stantic, J.Terry, R.Topor, A.Sattar}@griffith.edu.au

**Abstract.** Temporal and spatio-temporal data are present in many modern application systems, including monitoring moving objects. Such systems produce enormous volume of data, and therefore efficient indexing method is crucial. In this paper, we investigate and present a new concept based on virtual index structure, which can efficiently query such data. Concept is based on spatial representation of interval data and a recursive triangular decomposition of that space. The empirical performance of presented concept is demonstrated to be superior to its best known competitors.

## 1 Introduction

Modern database system, such as used for tracking moving objects, contain a significant amount of time dependent data [6]. Over the last two decades due to increased number of spatio-temporal applications interest in the field of temporal databases has increased significantly with contributions from many researchers [1], [8]. Because temporal databases are in general append only, they are usually very large in size, thus efficient access method is even more important in temporal databases than in conventional databases [2]. A number of access methods for temporal data that utilise the relational database systems built-in functionalities have been proposed, however, they have either space complexity problem or are generally tailored to be efficient only for specific query types. In [5] it has been shown that the RI-tree is superior to main competitors, the Window-List, Oracle Tile Index (T-Index) and IST-technique. However, the RI-tree need to tailor query transformation to the specific query types. It is our intention to propose an efficient access method for temporal data with logarithmic access time and guaranteed minimum space complexity that can answer a wide range of query types with the same query algorithm.

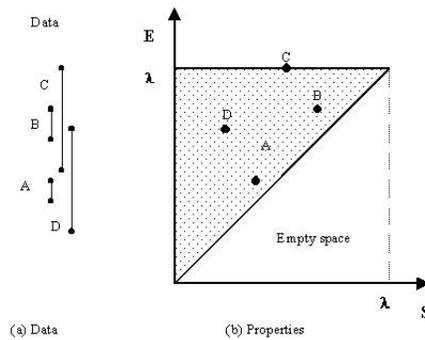
In this paper we present and investigate the Triangular Decomposition Tree (TD-tree) access method to index and query temporal data. In contrast to previously proposed access methods for temporal data this method can efficiently answer a wide range of query types, including point queries, intersection queries, and all nontrivial interval relationships queries, using a single algorithm without dedicated query transformations. It also can be built within commercial relational database system as it uses only built-in functionalities within the SQL:1999 standard and therefore no modification to the database kernel is required.

The TD-tree is a space partitioning access method. The basic idea is to manage the temporal intervals by a virtual index structure that relies on a two-dimensional representation of intervals and a triangular decomposition method. The resulting virtual binary

tree stores a bounded number of intervals at each leaf and hence may be unbalanced. As data is only stored in leaves, traversing the tree avoids disk accesses and tree depth therefore does not affect performance. Using the interval representation, any query type can be reduced to a spatial problem of finding those (triangular) leaves that intersect with the spatial query region. The efficiency of the TD-tree is due to the virtual internal structure so there is no need for physical disk I/O's, query algorithm that ensures pruning, and efficient clustering of interval data. On top of the advantages related to the usage of a single query algorithm for different query types and better space complexity the empirical performance of the TD-tree is demonstrated to be superior to its best known competitors.

## 2 The triangular decomposition tree (TD-tree)

The structure of our indexing method is based on the observation, that all data and query intervals of interest represented in two dimensional space lie in the isosceles, right-angle triangle with vertices at  $(0,0)$ ,  $(0,\lambda)$  and  $(\lambda,\lambda)$ , which lies above the line  $E = S$ . We call this triangle the *basic triangle* Figure 1. This is due to nature of interval space transformation and fact that  $i_s < i_e$ .



**Fig. 1.** Interval representation in two-dimensional space

Given that our region of interest is a triangle, our proposal is to recursively decompose the basic triangle into two smaller triangles, whenever triangle covers more than the chosen number of interval objects defined by blocking factor  $b$ . In such a triangular decomposition, each triangle is uniquely identified by its *apex* position  $(s, e)$ , and its *direction*  $d$ , the direction of the arrow from the midpoint of the triangle's hypotenuse to the apex.

Given a triangle in this decomposition, its apex and direction uniquely determine the apex and direction of each of its two subtriangles. The position  $(s, e)$  of the apex of each subtriangle  $C$  of a parent triangle  $P$  at any level  $l$  is possible to find out by knowing only the position of the parent apex and its level. Because we can identify the apex and

direction of every node of a TD-tree, starting from basic triangle *we do not need to store the internal tree nodes*. Thus, a TD-tree is a virtual tree. If a node has identifier  $\phi$ , the lower and upper children of the node have identifiers  $\phi 0$  and  $\phi 1$  respectively. The length of the identifier is thus one greater than the depth of the node. Information about leaf nodes themselves are stored in a separate directory, containing an identifier and number of records per leaf. The root node stores the blocking factor  $b$ ,  $\lambda$ , and current maximum depth of the tree.

It can be shown that the every query corresponds to a rectangular region of the two-dimensional interval space, defined by the top-left and bottom-right corners of this region. The task of query evaluation is to find all data intervals that occur within this query region. The particular region chosen depends on whether we are performing an intersection query, an overlaps query, a contains query, and so on, but in each case the query evaluation algorithm is identical, this is an important property of our approach.

### 3 Experimental evaluation

To show the practical relevance of our approach, we performed an extensive experimental evaluation of the TD-tree and compared it to the RI-tree [5]. The RI-tree was chosen, since it provides the same practically important properties as our approach. It is easy to implement and integrate, it uses standard RDBMS methods which provides scalability, update-ability, concurrency control and space efficiency. Furthermore it has been shown [5] that the RI-tree is superior to main competitors so the performance results of the TD-tree can be transferred to these indexing techniques. In our experiment we tested performance on intersection queries. However, because of the nature of our query algorithm, which compares the data region with the rectangular query region, results for intersection query applies to the other query types.

The Theory of Indexability [3] identifies I/O complexity cost, measured by the *number of disk accesses*, as one of the most important factors for measuring query performance, which in conjunction with *CPU usage* is used to assess the performance of the query process.

When making performance measurements of index structures it is important to consider parameters such as space requirements, physical disk I/O, CPU usage, clustering, updates, and locking. In our analysis we have concentrated on space requirements, physical disk reads, CPU usage and clustering of data. Because both the RI-tree and TD-tree rely on the relational paradigm, updates and locking are handled well by the RDBMS itself.

The TD-tree requires only one virtual index structure, which means only leaf nodes have to be stored. The list of leaf nodes are stored in the directory table and its size is very small comparing to the table itself. In our experiment the TD-tree directory for one million interval objects required only 26 data blocks. While The RI-tree requires two composite index structures *lowerIndex* (on *node* and *Start* - start of the interval) and the *upperIndex* (on *node* and *End* - end of the interval). In our experiment the RI-tree composite indexes for one million interval objects required 6708 data blocks (3354 each index).

The TD-tree enables efficient usage of clustering of the data by one dimension, i.e. region, as every region associate with block size. Clustering data improves the query performance and reduces the number of physical I/O, clustering ensures higher number of answers per physical disk I/O. In contrast, the RI-tree can not efficiently use clustering of data as it has to decide which dimension to use start or end. If it is clustered by *node* it will not result in similar improvements, as in RI-tree *node* are fixed size and are too large to provide effective clustering.

## 4 Conclusions

We described a new approach to efficiently manage vast volume of temporal and spatio-temporal data within the commercial RDBMS. More specifically, in this paper we:

- Presented the triangular decomposition tree (TD-tree) concept that can efficiently answer a wide range of query types with single algorithm;
- Experimentally evaluated the TD-tree by comparing its performance with the RI-tree, and demonstrated its overall superior performance.

The TD-tree is a unique access method as it uses tree structure and at the same time has some characteristics of hashing because it only stores data in leaf nodes. As a wide range of query types of interest can be reduced to rectangular region, it is possible to answer such queries using a single algorithm without requiring any query transformation. This itself, and the fact that the TD-tree can be incorporated within commercial RDBMS and utilised in a lot of spatio-temporal applications that manage vast volume of data, makes the TD-tree superior to other methods currently used or proposed in literature. Additionally, presented concept of regular decomposition and virtual internal structure can be extended and applied to more dimensions to efficiently manage high dimensional data.

## References

1. C. Date, H. Darwen, and N. Lorentzos. *Temporal Data and the Relational Model*. Morgan Kaufmann, 2002.
2. C. E. Dyreson, R. T. Snodgrass, and M. Freiman. Efficiently Supporting Temporal Granularities in a DBMS. Technical Report TR 95/07, 1995.
3. J. Hellerstein, E. Koutsupias, and C. Papadimitriou. On the Analysis of Indexing Schemes. *16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1997.
4. H.-P. Kriegel, M. Potke, and T. Seidl. Object-relational indexing for general interval relationships. In *Proc. 7th Intl Symposium on Spatial and Temporal Databases (SSTD01)*, 2001.
5. H.-P. Kriegel, M. Potke, and T. Seidl. Managing intervals efficiently in object-relational databases. *Proceedings of the 26th International Conference on Very Large Databases*, pages 407–418, 2000.
6. H. Marios, G. Kollios, V. J. Tsotras, and D. Gunopulos. Indexing Spatialtemporal Archives. *The VLDB Journal*, 15(2), 2006.
7. S. Ramaswamy. Efficient Indexing for Constraint and Temporal Databases. In *Proceedings of the 6th International Conference on Database Theory*, pages 419–431, 1997.
8. R. T. Snodgrass. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, 2000.