AN EFFICIENT INDEXING STRUCTURE FOR TRAJECTORIES IN GEOGRAPHICAL INFORMATION SYSTEMS

By

SOWJANYA SUNKARA

Bachelor of Science (Engineering) in Computer Science
University of Madras
Chennai, TamilNadu, India
2004

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2007

AN EFFICIENT INDEXING STRUCTURE FOR TRAJECTORIES IN GEOGRAPHICAL INFORMATION SYSTEMS

Thesis Approved:

Dr. Xiaolin (Andy) Li

Thesis Adviser

Dr. John Chandler

Dr. Hongbo Yu

Dr. A. Gordon Emslie

Dean of the Graduate College

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table Page

ABSTRACT

Technologies dealing with location such as GPS are producing more and more data of moving objects. Spatio-temporal databases store information about the positions of individual objects over time. Real-world applications of spatio-temporal data include vehicle navigation, migration of people, tracking and monitoring air-based, sea or land-based vehicles. Spatio-temporal database is needed to manage these data, so as to solve the problems in spatio-temporal applications. A spatio-temporal database adopts an exhaustive search strategy for querying the trajectories. This is very time-consuming when processing large datasets for the given spatio-temporal query conditions. As a result, efficient Spatio-Temporal indexing methods are highly demanded to improve the performance of the system in searching such large datasets.

Referring to Fixed Network R-Tree (FNR-Tree) and the trajectory oriented indexing method Trajectory Bundle-Tree (TB-Tree), in this thesis, an indexing structure is proposed for trajectories to efficiently support spatiotemporal queries called the FNR-TB Tree. This indexing structure will help in analyzing space-time relationships among space-time paths, an example of spatio-temporal data, in a large dataset. This technique discriminates the space dimension from the time dimension for indexing and thus is expected to give better performance than the existing methods. Furthermore, since this

method utilizes the advantages of the TB-Trees, it yields a better performance for spatiotemporal queries than other existing methods for retrieving trajectories.

CHAPTER 1

INTRODUCTION

Over the past few years, there has been a continuous progress in the wireless communications sector and the positioning technologies. This makes it increasingly plausible as well as necessary to track as well as record the changing positions of objects which are capable of continuous motion. Spatio-temporal data come from various sources, for example, the movements of people, animals, vehicles, airplanes, boats, extra-terrestrial objects, military objects, weather (areas of high/low pressure, and storms), the changing borders of countries or cities – the list is endless because anything that changes position and/or shape through time can be classified as spatio-temporal data.

Technological advances in the wireless sector (e.g. Global Positioning System (GPS) for Geographical Information Systems applications and other wireless technologies) as well as hardware advances have made it easier to obtain position information about spatio-temporal data. Applications include areas such as transportation (cars moving in the highway system), satellite and earth change data, planetary movements etc. Spatio-temporal databases store and manipulate these moving objects that may change their spatial locations and/or their shapes over time.

1.1 Motivation

A first approach to represent the movement of objects would be to simply store the position samples. These position samples at discrete instances of time are obtained by GPS and other communication technologies. These successive position samples of an object can be viewed as a sequence of line segments that collectively form the trajectory

for that object. Thus a trajectory is obtained by sampling the positions of a moving point across time and interpolating in between samples [4]. In order to obtain the entire movement of an object that forms the trajectory, linear interpolation is used. The sampled positions then become the endpoints of line segments of polylines, and the movement of an object is represented by an entire polyline in 3D space.

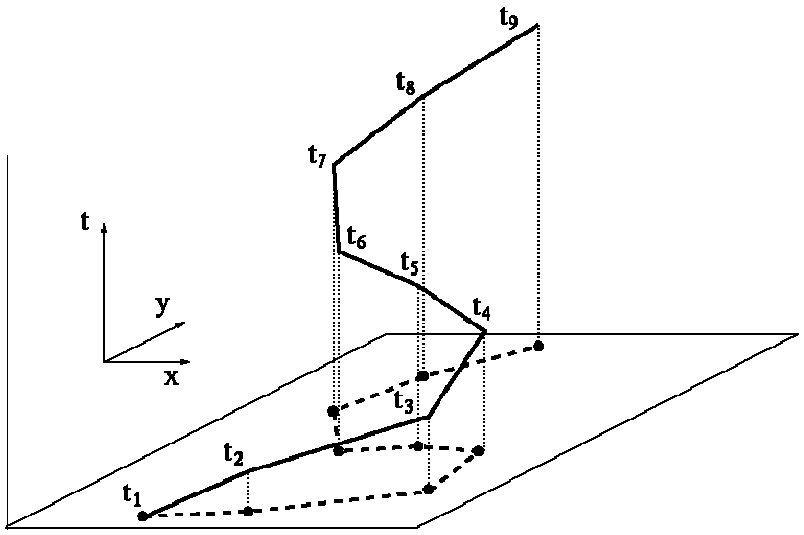

Figure 1.1: Trajectory of a moving object in 2+1 dimensional space [4]

The solid line in Figure 1.1 represents the movement of a point object. Space and time co-ordinates are combined to form a single coordinate system. The dashed line shows the projection of the movement in the 2D plane. Thus, a trajectory segment for an object moving in a k dimensional space is essentially a line in a k+1-dimensional space, with time as the additional information.

1.2 Overview

Trajectory data for moving objects is continuously changing between any two successive updates of the location of a moving object. This poses a problem in representing the location of the object at all times because most convenient models for

data representations are static in nature. A commonly used model for representing trajectory data approximates the motion of an object as a straight line segment between two consecutive updates [5]. In this thesis, this model is used for representing the trajectories.

A key problem in the moving objects scenario is the ability to index efficiently the movements of mobile objects. There are two different, although related indexing problems that must be solved in order to support applications involving continuous movement. One problem is indexing of current and anticipated future positions of moving objects. The other problem is indexing of histories, or trajectories, of the positions of moving objects.

Indexing techniques have been used very effectively for managing large data sets in other application domains, and hence it is expected that indexing techniques play a crucial role in managing trajectory data sets too. Many indexing techniques have been proposed recently for indexing and querying trajectories. Trajectories are three-dimensional entities consisting of space and time, and they can be indexed using spatial access methods. However there are problems. Trajectories are decomposed into their constituent line segments, which are then indexed.

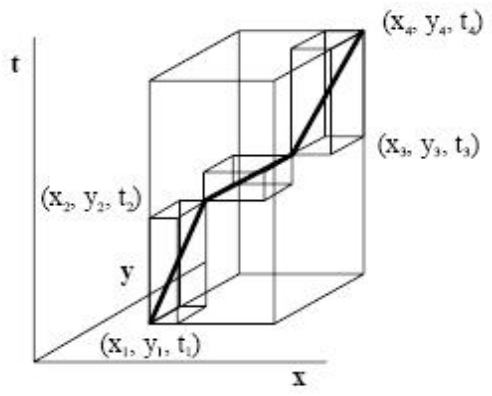

Figure 1.2 Approximating Trajectories using MBBs [4]

The use of R-Tree for indexing trajectories is shown in Figure 1.2. The R-Tree approximates the data objects by Minimum Bounding Objects (MBBs). But this technique is proved to be inefficient as it introduces large amounts of "dead space". It is evident from Figure 2 that the MBBs cover large parts of space, whereas the actual space occupied by the trajectory is small. This leads to high overlap and consequently to a small discrimination capability of the index structure. Furthermore, the storage requirement of an R-Tree based index is typically high enough to make it less practical to store historical databases, which tend to grow large rapidly. For example, the size of an R-Tree that stores trajectories of 100 moving objects was about 95MB [2]. It appears that the size of an R-Tree based index may grow too fast to be used for large-scale applications in practice.

In this thesis, the issue of indexing large trajectory data sets (for GIS applications) is addressed and a new indexing structure called the FNR-TB tree structure, an extension to the Fixed Network R-Tree (FNR) is proposed and presented. The general idea that describes the FNR-TB Tree is a forest of TB-Trees on top of a two dimensional (2D) R-

Tree. The 2D R-Tree is used to index the spatial data of the network while the TB trees are used to index the time interval of each objects' movement inside a given link of network.

1.3 Thesis Outline

The thesis work is organized as under: Chapter 2 explains the background of the problem. The spatio-temporal databases, data and spatio-temporal queries are explained. An extensive survey of the existing spatio-temporal indexing methods is also explained. An existing solution for indexing the trajectories namely the Fixed Network R-Tree is explained with the insertion and search algorithm in Chapter 3. Chapter 4 describes the proposed FNR-TB Tree approach for indexing trajectories. The new modified index structure which will give better query performance is explained with algorithms for insertion and searching the index structure. Experimental evaluation of the proposed technique is provided in Chapter 5 by comparing it with the original FNR- Tree. Finally, conclusions and any further considerations are made.

CHAPTER 2

BACKGROUND AND RELATED WORK

In the past, research in spatial and temporal database systems has mostly been done independently. Spatial database research has focused on supporting modeling and querying of the geometry associated with objects in a database. Several spatial access methods have been proposed in the literature for storing multi-dimensional object (e.g. points, line segments, areas, volumes, and hyper-volumes) without considering the notion of time. On the other side, temporal database have focused on extending the knowledge kept in a database about the current state of the real world to include the past, in the two senses of transaction time and valid time. This chapter explains what constitute the spatio- temporal database, the trajectory model used, the various queries are explained and finally the proposed methods for trajectory indexing are indexed.

2.1 Spatio Temporal Databases

Spatio-temporal databases have received considerable attention during the past few years due to the accumulation of large amounts of multi-dimensional data evolving in time, and the emergence of novel applications such as traffic supervision and mobile communication systems. Spatio-temporal databases deal with geometry changing over time. In general, geometry cannot only change in discrete steps, but continuously, for moving objects. If only the position in space of an object is relevant, then "moving point"

is a basic abstraction; if also the extent is of interest, then the "moving region" abstraction captures moving as well as growing or shrinking regions.

There are many important applications of the spatiotemporal database management systems (STDBMSs). Some of them can be seen in Geographic Information Systems (GIS), environmental information system and multimedia. There are also many applications that create the data that is managed by spatiotemporal database system and changes over time. For example global change (as in climate or land cover changes), transportation (traffic surveillance data, intelligent transportation systems), social (demographic, health, etc.), telecommunications (mobile phones), multimedia (animated movies) applications and so on. Put briefly, a spatiotemporal database is a database that embodies spatial, temporal, and spatiotemporal database concepts, and captures simultaneously spatial and temporal aspects of data. All the individual spatial and temporal concepts (e.g., rectangle or time interval) must be considered.

Indexing techniques have been used since the advent of relational database management systems with success. Indexing is even more important when the data is more complex and, for spatial database systems, due to high performance requirements, access methods should be used on every relation for supporting queries efficiently. In spatial applications, the assumption is always almost true that a spatial index exists on a spatial relation [9]. Following these ideas, for moving objects databases, which is a spatio-temporal application, and consequently more complex, the need for good indexing techniques is obvious.

2.2 Trajectory Model

There are two main types of spatiotemporal databases [23], those that manage historical information and those that manage current information for current/predictive query purposes. The focus of this thesis is on the first category, which deals with the complete history of moving objects represented through trajectories. In the trajectory model, a moving object's movement in the 2D plane over time can be treated as a trajectory in the 3-Dimension spatio-temporal space. A moving object's trajectory consists of many trajectory segments. An object is considered as moving at the same speed and in the same direction in a segment.

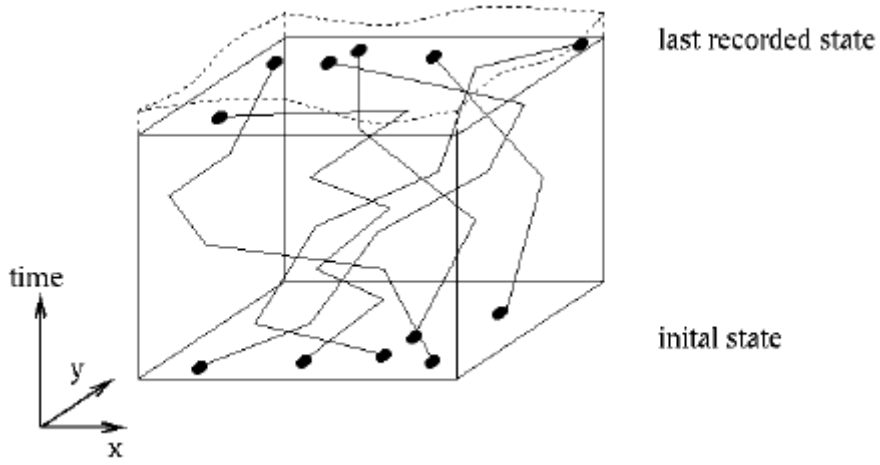

Figure 2.1: Trajectories of moving point objects in spatiotemporal workspace [2]

Figure 2.1 illustrates the spatio temporal workspace (the cube) and several trajectories. Time moves in the upward direction, and the top of the cube is the most recent position sample. The wavy dotted lies at the top symbolize the growth of the cube with time.

2.3 Query Types

A typical search on sets of objects' trajectories includes a selection with respect to a given range, a search inherited from spatial and temporal databases. Queries of the form "find all objects within a given area (or at a given point) some time during a given time interval (or at a given time instant)" or "find the k-closest objects with respect to a given point at a given time instant" [6] remain very important.

Queries on moving data can be broadly classified into two categories: queries that ask questions about the future positions of moving objects, and queries that ask questions about the historical positions of moving objects. The former class of queries can be answered by storing current position, speed and the direction of the moving objects [13]. For the second class of queries, Pfoser et al. [2] further classify historical queries into two different sub-classes: coordinate based queries and trajectory-based queries.

- Coordinate-based queries include (a) time-interval, which selects all objects within a given area and given time period, (b) time-slice queries, which select all the objects present in a given area at a time instant, as well as point, range, and nearest-neighbor queries in the resulting 3D space, and
- Trajectory-based queries can be further classified in "topological" queries, which involve the complete information of the movement of an object, and "navigational" queries, which involve derived information, such as speed and heading. Topological queries involve the whole or a part of the trajectory of an object. They are deemed very important, but also rather expensive.

2.4 Spatio-Temporal Data Generators

Wireless computing devices can generate spatio-temporal data. Depending on the density of the network, the accuracy of such an approach can be extremely high (within 2 meters) [17]. Typically, the required wireless network infrastructure means that data can only be generated within restricted areas such as an office building or a university campus. Not surprisingly, due to the private and proprietary nature of data generated using any of the above technologies; it can be difficult to obtain real datasets to spatio-temporal data for experimental purposes. Some animal tracking, hurricane, and public bus datasets are publicly available [18]; however, the number of records (in the hundreds) is insufficient for large scale database benchmarking.

Due to the lack of readily available real datasets and the need to generate data in a controlled fashion for experimental purposes, synthetic spatio-temporal data generators have been developed. The most notable synthetic generators include the Generate Spatio Temporal Data (GSTD) Tool [19], the Network-based Generator of Moving Objects [20], a generator for Time-Evolving Regional Data (G-TERD) [21], and the City Simulator by IBM.

The GSTD [19] generates data according to prescribed statistical distribution. Currently uniform, Gaussian, and Skewed distributions are supported. Both point and moving region (rectangle) spatio-temporal data can be generated. The cardinality of the dataset can be easily adjusted in order to perform large scale experiments. For point data, GSTD requires an initial, time and center distribution. The initial distribution defines where objects begin within the unit space that GSTD assumes. The time distribution controls the timestamps when object locations are updated. The center distribution

controls the movement of points in space. Advanced features supported by the GSTD include support for a framework, i.e., obstacles that objects cannot enter, support for multiple datasets with different properties, and an on-line dataset visualizer. Even with the use of a framework, the objects generated by the GSTD can move anywhere and along any path within the unit space where obstacles do not exist.

On the other hand, the Network-based Generator of Moving Objects [20] can generate a dataset where objects follow a given network, e.g., roads. In such a scenario the maximum allowable speed and capacity of a network is important, as well as the interaction of moving objects. Objects are assumed to have a starting location and a destination. A key advantage of the Network-based generator is that synthetic data that follows the real topology of a network, e.g., the roads of a city, can be generated. Objects can also belong to a class, e.g., cars vs pedestrians, with different properties such as maximum velocity.

2.5 Spatio Temporal Access Methods

Having described the types of data and queries, and the data model for moving point objects, this section defines the access methods that have been proposed for those types of data and queries. A detailed survey of the existing Spatio-temporal access methods for various types of queries is discussed in this section.

Most of the access methods are variations of hierarchical index structures such as R-Trees and Quad Trees. The R-Trees and the Quad trees cannot handle temporal data which are inherent in trajectories. In these methods, objects move freely in the Euclidean space. Because moving objects change their locations continuously on road networks, the

amount of trajectory information for a moving object is generally very large. Since all the approaches are based on R-tree, a short overview of the R-tree is given.

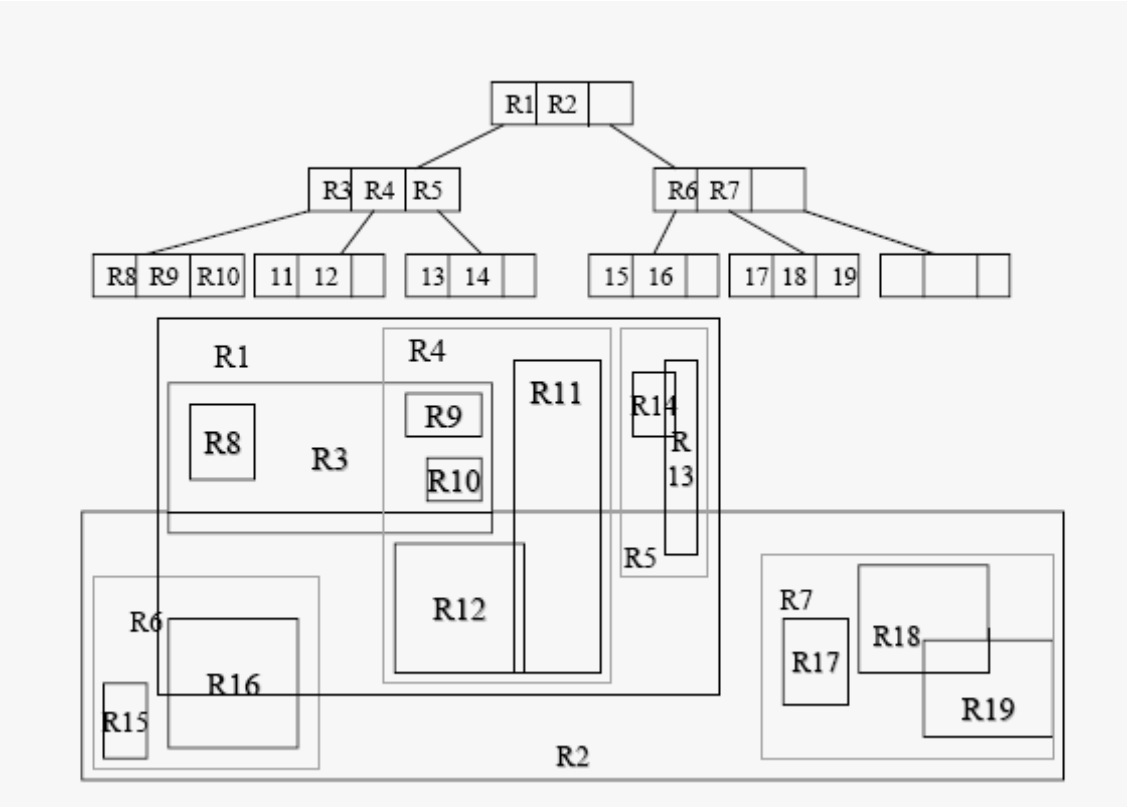

Figure 2.2 R-Tree structure for 2D rectangles [16]

The R-tree shown in Figure 2.2 is a height-balanced tree with the index records in its leaf nodes containing pointers to actual data objects. Leaf node entries are of the form (id, MBB), where id is an identifier that points to the actual object and MBB (Minimum Bounding Box) is an n-dimensional interval. Non-leaf node entries are of the form (ptr, MBB), where ptr is the pointer to a child node and MBB is the covering n-dimensional interval. A node in the tree corresponds to a disk page. Every node contains between m and M entries.

The insertion of a new entry into the R-tree is done by traversing a single path from the root to the leaf level. The path is chosen with respect to the least enlargement

criterion (ChooseLeaf algorithm by Guttman (1984)) and covering MBBs are adjusted accordingly. In case an insertion causes splitting of a node, its entries are reassigned to the old node and a newly created one (according to one of the three alternative algorithms, Exhaustive, QuadraticSplit or LinearSplit, proposed by Guttman (1984)). To delete an entry from the R-tree, a reverse insertion procedure applies, i.e., covering MBBs are adjusted accordingly. In case the deletion causes an underflow in a node, i.e., node occupancy falls below m, the node is deleted and its entries are re-inserted. When searching an R-tree, we check whether a given node entry overlaps the search window (assuming a range query). If so, we visit the child node and thus recursively traverse the tree. Since overlapping MBBs are permitted, at each level of the index there may be several entries that overlap the search window.

In the context of spatiotemporal data this technique proves to be inefficient. Figure 2 shows that in approximating the line segments with MBBs, we introduce large amounts of "dead space." It is evident that the corresponding MBB covers a large portion of the space, whereas the actual space occupied by the trajectory is small. This leads to high overlap and consequently to a small discrimination capability of the index structure. Another aspect not captured in R-trees is the knowledge about the specific trajectory a line segment belongs to. To smoothen these inefficiencies, Pfoser et. al. [19] modifies the R-tree as follows: A line segment can only be contained in four different ways in an MBB. This extra information is stored at the leaf level by simply modifying the entry format to (id, MBB, orientation), where the orientation's domain is {1,2,3,4}. Assuming we number the trajectories from 0 to n, a leaf node entry is then of the form (id, trajectory#, MBB, orientation).

The RT-Tree [1] couples time intervals with spatial ranges in each node of the tree structure. Each node contains both the spatial extent and the time interval during which the object exists. Searching in these tree nodes is guided only by the spatial information with only secondary importance to the temporal information. This feature makes the RT-Tree inefficient for answering queries based on the temporal dimension as this would require a complete scan of the whole index.

The 3D R-Tree [1], based on the R-Tree treats time as another dimension similar to space. This straightforward method of indexing spatio-temporal data is the most space-efficient since it stores only the different object versions without redundant copies. Since the temporal attributes and the spatial attributes are tightly integrated in 3D R-tree nodes, a time-interval query can be treated as an ordinary window query and thus can be processed very efficiently. However, the performance of time-slice queries can be poor, since all objects are indexed in a single tree and thus the query processing time will depend on the total number of entries in history.

The MR-tree, the HR-tree and the Overlapping Linear Quadtree are all based on the concept of overlapping trees. The basic idea is that, given two trees where the second one is an evolution of the first one, the second tree can be represented by registering only the modified branches of the first one. That is, only the modified branches are actually created and the branches that do not change are simply re-used, while each tree keeps a root of its own.

The HR-tree can handle a time-slice query very efficiently since time-slice queries are processed inside the corresponding R-tree only. For time-interval queries, however, it can be very slow, because time-interval queries require searching as many trees as the

number of distinct timestamps in the interval. Although the HR-tree adopts an overlapping technique, the benefit tends to be limited, because even a single object moving between two consecutive timestamps may cause the replication of multiple nodes. Thus, the size of the HR-tree can be still prohibitive for practical applications.

Recently, Tao and Papadias proposed the MV3R-tree [1], a structure that combines the concepts of multi version B-trees and 3D R-trees. The key idea is to utilize a multi version R-tree for time-slice queries and a 3D R-tree for time interval queries. Although the MV3R-tree can deal efficiently with both time-slice and time interval queries by combining the benefits of both structures, the MV3R-tree requires more space than the 3D R-tree and still needs high management cost due to the combination of the MVR-tree and the 3D R-tree.

Pfoser et al., proposed [2] spatio-temporal access methods that store trajectories as line segments aiming at processing spatial range queries and trajectory queries. They are the Spatio Temporal R-Tree (STR) and the Trajectory Bundle (TB) Tree. The STR-Tree is an extension of the R-Tree to support query processing for the trajectories of moving objects. The insertion procedure in the STR-tree considers spatial proximity and partial trajectory preservation. That is, the STR-tree tries to keep line segments belonging to the same trajectory together. Also, the authors proposed the TB-tree, which aims for an access method that strictly preserves trajectories such that a leaf node only contains line segments belonging to the same trajectory.

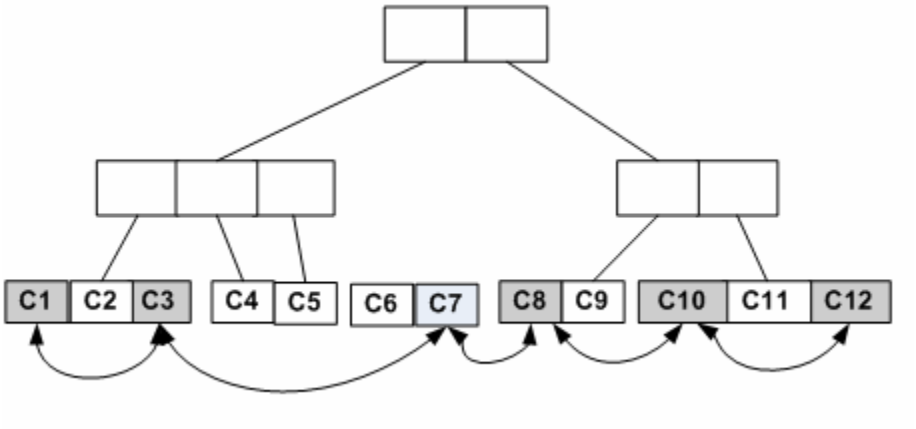

Fig 2.3 TB-Tree Structure

As such, the structure of the TB-Tree is actually a set of leaf nodes, each containing a partial trajectory, organized in a tree hierarchy. A doubly linked list connects the leaf nodes including parts of the same trajectory in a way that preserves trajectory evolution. Figure 3.1 shows a part of the TB-Tree structure and a trajectory illustrating the approach. The gray band is fragmented across six nodes c1, c3, etc. Although the TB-tree supports trajectory queries much more efficiently than the R-tree does, its performance on time-slice and time-interval query is worse than the R-tree for a large number of moving objects since the TB-tree does not consider spatial discrimination. As for the STR-tree, although designed to combine the benefits of the TB-tree and the R-tree, it usually performs worse than the TB-tree and is rather a weak compromise.

Later SETI and SEB trees are developed based on static partitioning of the 2-dimensional space. After TB-trees, these are the most recently proposed indexing structures for trajectory-oriented queries. The main observation in these techniques is that the change of the spatial dimension is limited while the temporal dimension is continuously evolving. So it is possible to partition the space statically. In the SETI

(Scalable and Efficient Trajectory Index) tree [5], the space is partitioned into static and non-overlapping zones. Within each partition, the trajectory segments are indexed using an R-Tree. The SEB (Start/End timestamp B-Tree) tree [3] which is similar in idea to that of SETI, partitions the space that can be overlapped. The key differences from SETI being that in SEB, trajectories are not considered only 2-dimensional points are indexed.

Although a vast majority of the above mentioned spatio-temporal access methods index objects moving in the whole of the 2D space, real world applications involve objects moving in a constrained space. According to [1], there are three scenarios about the movement of objects in a 2-dimensional space: the unconstrained movement (e.g., vessels at sea), the constrained movement (e.g., pedestrians on an urban environment), and the movement in transportation networks (e.g., trains, cars). Generalizing the second two categories, we can say that the way in which an urban network is handled depends on the required precision of the application. A network can be represented as having infrastructure (e.g., buildings) blocking the objects movement or it can be represented by a set of connected line segments.

For the constrained movement scenario, a two-step query processing is proposed in [1]. A pre-processing step is added where the infrastructure is queried and the query window is divided into a set of smaller query windows, from which the regions covered by the infrastructure have been excluded. This set of smaller query windows is then passed to a second step where a modified version of the R-Tree and the TB-Tree are queried using an approach proposed in [13].

For applications that do not require keeping track of the exact positions of the moving objects in terms of co-ordinates, recently, two index structures for indexing the

trajectories of moving objects in networks have been proposed. Both use the same idea of converting a 3-dimensional problem into two sub-problems with lower dimensions. The first of these, the Fixed network R-tree (FNR-Tree) [4] consists of a top-level 2D R-Tree, whose leaf nodes contain pointers to 1D R-Trees. The 2D R-Tree indexes the edges of the network, indexing the corresponding line segments. For every leaf node in the 2D R-Tree, there is a 1D R-Tree indexing the time interval of the objects traversing the line segments inside of its entries.

Another indexing technique called the MON-Tree (Moving Objects in Networks Tree) [9] is also proposed which indexes two different network models viz., edge oriented and route oriented models where the edges and routes are no longer single line segments but modeled as polylines. The indexing structure consists of a hash table and a 2D R-Tree at the top level indexing the polylines and a set of bottom R-Trees indexing the objects movements along the polylines.

The second index structure proposed by Pfoser et al., [6] uses a Hilbert Curve to linearize the network line segments i.e., the 2-dimensional movement space (without the temporal dimension) is mapped to a 1-dimensional space. The line segments on the Hilbert Curve are then indexed using a 2D R-Tree and the objects moving on the edges are indexed using a global 2D R-Tree unlike the FNR-Tree where there is a bottom R-Tree for each leaf node to index the objects movements on the edges.

According to the experiments performed [4], these indexing structures do not seem to perform well for time-slice/interval queries and for efficient retrieval of the trajectory information for trajectory-oriented queries. These indexing methods do not preserve moving objects' trajectories. For example in the FNR-Tree, to retrieve the object

trajectories with a time interval and without any space interval would mean an exhaustive search of all the 1D R-Trees.

CHAPTER 3

PROBLEM DEFINITION

A first motivation for this thesis is to improve the efficiency in query processing of the trajectory model presented in section 2.2. Trajectory data sets of line segments exhibit characteristics that are different from general three dimensional data sets. Rather than using an R-tree for storing 3-D line segments, these differences can be exploited to design a more efficient indexing structure. In a 3D R-tree, the temporal and the spatial dimensions are treated equally. However, for trajectory data sets, there are important differences in the characteristics of these dimensions. More specifically, the boundaries of the spatial dimensions remain constant or change very slowly over the lifetime of the trajectory data set growth, whereas the time dimension is continually increasing.

Since the extent of the spatial dimensions does not change, an indexing structure could partition the spatial dimensions statically. Within each spatial partition, the indexing structure only needs to index lines in a 1-D (time) dimension. Consequently, such an approach will not exhibit the rapid degradation in index performance that is generally observed for 3-D indexing techniques. The Fixed Network R-Tree (FNR) indexing mechanism capitalizes on this observation to achieve good spatial and temporal discrimination. The remaining of this chapter explains the insertion and search algorithms of the 2D R-Tree and the FNR-Tree upon which the proposed method is built. The shortcoming of the existing method is highlighted and the need for the proposed approach is explained.

3.1 2D R-Tree Algorithms for the top level FNR-Tree

A new entry is inserted into the R-Tree by traversing a path from the root to the leaf level. The path is chosen with respect to the least enlargement criterion and the covering MBB's are adjusted accordingly. If an insertion results in a node exceeding the maximum number of entries, the entries are reassigned into the old node and newly created one. If a new entry has to be inserted into a database, a new index record has to be added to the R-Tree. This is also the only chance for the R-Tree to grow in height, namely if there is a node overflow, the node has to be split. In the case that the split reaches the root, the height will grow.

R-Tree Insertion Algorithm

1. Let E be the new entry.
2. Invoke ***ChooseLeaf*** to select a leaf node in which to place the new entry.
   We choose the Node in which to place the newly inserted entry based on the least enlargement criterion
3. If the node selected by Step 1 has room for another entry, then insert it.
   Otherwise invoke ***SplitNode*** to obtain "Node" and a new node "NewNode".
   Both nodes contain the new entry and all the old entries of "Node"
4. Propagate Changes upwards to the root of the tree using ***AdjustTree*** algorithm, passing also "NewNode" because of the previous performed split.
5. If the node split propagation caused the root to split, create a new root whose children are the two resulting nodes.

Figure 3.1 R-Tree Insertion Algorithm

Algorithm 2: R-Tree Chooseleaf Algorithm

Select a suitable leaf node to place the new entry E in.

1.Start with the root node.

2.If the selected node is a leaf then return else if ***the node*** is not a leaf check all entries of the node and select the entry whose rectangle needs least enlargement in order to include the newly inserted interval (X1, Y1, X2, Y2)

Figure 3.2 R-Tree Chooseleaf Algorithm

Algorithm 3: R-Tree AdjustTree Algorithm

1. Ascends from the leaf node (N) adjusting upwards.
2. If N is the root of the tree then stop.
3. Let P be the parent node of N and let E_N be N's entry in P. Adjust E_N so that it tightly encloses all entry rectangles in N
4. If N has a partner NN resulting from an earlier split, create a new entry $E_{NN.}$ The pointer of E_{NN} points to the node NN (resulted from a split)
5. Adjust E_{NN}'s covering rectangle so as to cover the NN 's covering rectangle
6. Add E_{NN} to P if there is room. Otherwise invoke the ***SplitNode*** to produce P and PP containing E_{NN} and all P's old entries.

Figure 3.3 R-Tree AdjustTree Algorithm

Algorithm 4: R-Tree SplitNode Algorithm

1. Find pair of entries E1 and E2 that maximizes *area* (J) – *area* (E1) – *area* (E2) where J is covering rectangle.
2. Pick the pair with least affinity, i.e., the pair that wastes maximum space
3. Put E1 in one group, E2 in the other.
4. If one group has M-m+1 entries, put the remaining entries into the other group and stop. If all entries have been distributed then stop.
5. For each entry E, calculate d1 and d2 where di is the area increase in covering rectangle of group i when E is added.
6. Find E with maximum |d1 - d2| and add E to the group whose area will increase the least.
7. Two other methods are used for this algorithm:
8. ***PickSeeds***: Selects the two first elements of the groups.
9. ***PickNext***: Selects one of the remaining entries to put in a group.

Figure 3.4 R-Tree SplitNode Algorithm

Algorithm 4: R-Tree Pick Seeds Algorithm

Used to select the first two entries while splitting node.

1. For each pair of entries E1 and E2, compose a rectangle J comprising of the both.
2. Calculate d = area(J) – area(E1) – area(E2)
3. Choose the pair with the largest *d.*

   This algorithm aims at choosing pair of nodes which would have minimum overlap or are the farthest.

Figure 3.5 R-Tree PickSeeds Algorithm

Algorithm 5: R-Tree PickNext Algorithm

This algorithm picks the next node from the set Q

1. For each entry E in Q and not yet inserted, calculate

   d1 = The area increase required in the covering rectangle of the Group 1 to include E.

   d2 = the area increase required in the covering rectangle of the Group 2 to include E.
2. Choose an entry with the maximum difference between d1 and d2

   This algorithm aims to minimize the time required in finding the smallest area split at the cost of accuracy.

Figure 3.6 R-Tree PickNext Algorithm

To conduct a range search on an R-Tree, we start from the root and recursively visit all child nodes whose MBB's intersect with the search window.

Algorithm 6: R-Tree Search Algorithm

1. Let *T* be the root of an R-Tree. We search all index records whose rectangles overlap a specified search rectangle *S*.
2. If T is not a leaf, then apply ***Search*** on every child whose root is pointed by *child-pointer* and that overlaps with S.
3. If T is a leaf, return all entries which overlap with S as the result set.

Figure 3.7 R-Tree Search Algorithm

3.2 Fixed Network R-Tree (FNR-Tree)

The FNR-Tree structure consists of a top level 2D R-Tree whose leaf nodes contains pointers to 1D R-Trees. The 2D R-Tree is used to index the edges of the network, indexing their corresponding line segments. For every leaf node in the 2D R-Tree, there is a 1D R-Tree indexing the time interval of the objects traversing the line segments inside of its entries.

Since the spatial objects stored in the 2D R-Tree are line segments, the leaf node entries are of the form *{MBB, orientation}*, where *MBB* is the minimum bounding box (MBB) of the line segment and *orientation* is a flag that describes the two possible orientations that a line segment can assume inside a MBB.

Each leaf entry of the 1D R-Tree is of the form *{moid, edgeid,* $t_{entrance,}$ $t_{exit,,}$ *direction}*, where *moid* is the identification of the moving object, *edgeid* is the identification of the edge, $t_{entrance}$ and t_{exit} represent the time interval when the moving object traversed the edge, and *direction* is a flag that describes the direction of the moving object, more specifically, the direction 0 means that the movement began at the left-most node of the edge (bottom-most for vertical edge), and 1 otherwise.

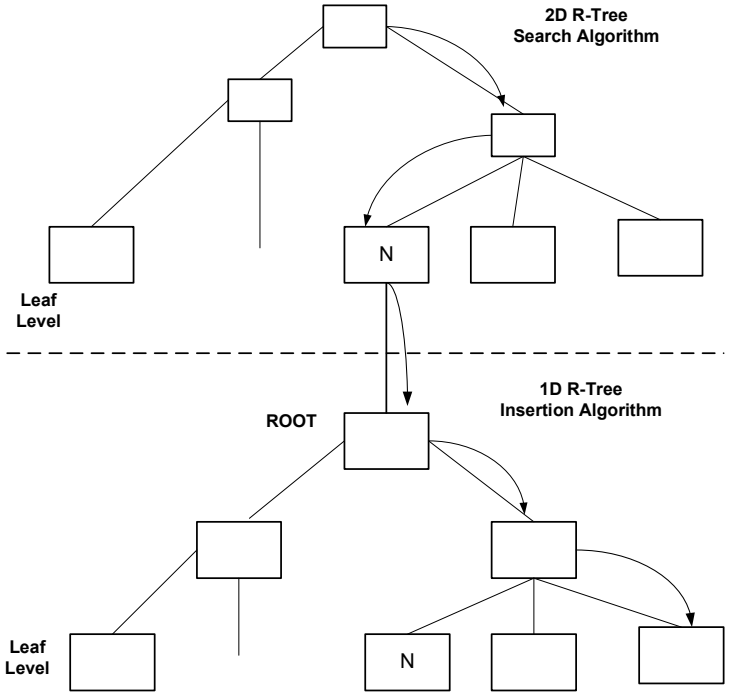

Figure 3.8 Insertion of a new entry in a FNR-Tree

As shown in Figure 3.8, the insertion algorithm is executed each time a moving object leaves a given line segment of the network. It takes the line segment on which the object was moving, its direction, and the time interval when the object traversed the line segment. The algorithm first searches in the 2D R-Tree for the line segment and then inserts the time interval (associated with the direction) of the moving object into the associated 1D R-Tree. Instead of using the insertion algorithms of the R-Tree, in the 1D

R-Tree, every new entry is simply inserted into the most recent (right-most) leaf of the 1D R-Tree. This is possible because time is monotonic and the insertions are done in time increasing order.

3.3 Need for the proposed approach

The FNR-Tree outperforms the R-Trees in most cases, but it has a critical drawback that it has to maintain a tremendously large number of R-Trees. This is because it constructs as large number of R-Trees as the total number of segments in the networks.

A very important characteristic of the space-time paths, a representation for trajectories, is the ability to be able to answer the trajectory-based queries efficiently. Queries about trajectories help in analyzing the properties of the trajectories and their relationships with other trajectories. This is useful for finding spatio-temporal clusters of the datasets. A common type of query is a trajectory retrieval, which aims at efficient retrieval of line segments representing trajectories of moving objects. Trajectory- based queries help in understanding the relationships between the various space-time paths and hence helps in their analysis. A number of access methods have been proposed based on the R-Tree method.

Among the spatio-temporal access methods discussed, Trajectory Bundle (TB)-Trees [2] are found to perform well for trajectory-based queries. As shown in Fig. 3.9, a TB-tree calls for a modified R-Tree structure wherein leaf nodes that contain information from the same trajectory are linked together. Also a doubly-linked list connecting the leaf nodes is implemented that helps in retrieving the entire trajectory from any arbitrary segment.

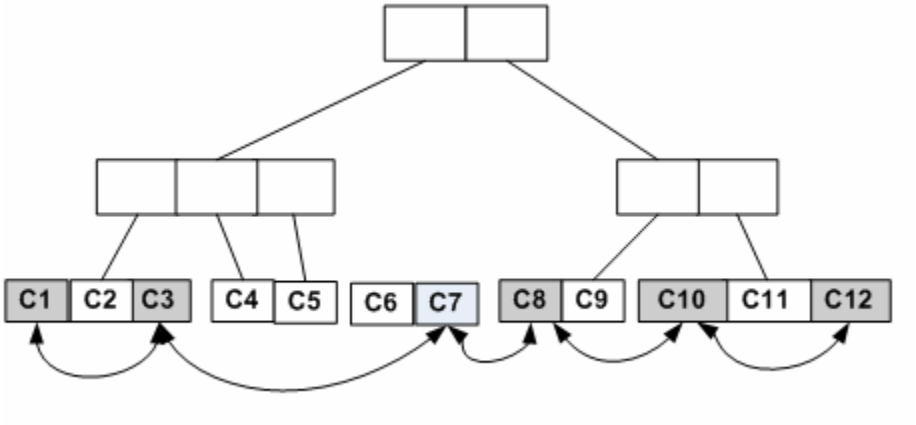

Figure 3.9 TB-Tree structure showing the linked list

This can lead to reduced query time when retrieving the complete trajectory of an object. Thus with the TB-Tree, an access method can strictly preserve trajectories such that a leaf node only contains segments belonging to the same trajectory. Pfoser [6] in his proposed approach involving the reduction of dimensionality has also suggested using TB-Trees in the place of R-Trees for indexing the trajectory data. In addition, in the spatio-temporal access method proposed in [4][9], using TB-Trees in the place of the 1D R-Tree (FNR-Tree) helps improve the performance of the queries. So using TB-Trees is a meaningful choice which is expected to improve the performance for spatio-temporal queries [2]. The insertion and search algorithms for the TB-Trees are explained as follows.

Algorithm Insert (N,E)

1. Invoke **FindNode (N,E)**
2. **IF** node 'N' is found,

    IF 'N' has space,

        Insert new segment.

    ELSE

        Create new leaf node for new segment

ELSE

    Create new leaf node for new segment

Figure 3.10 TB-Tree Insertion Algorithm

Figure 3.10 shows the insertion algorithm that aims to divide the trajectory of a moving object into pieces containing M line segments, M being the maximum number of entries in a leaf node. To insert a new entry, the most recent line segment in the index that has the same trajectory number must be found. This step is carried out by the FindNode procedure given in Figure 3.11. It traverses the tree from the root and steps into every child node that intersects the MBB of the new line segment. The leaf node that contains a segment connected to the new entry is chosen. If the leaf node is full a splitting strategy is needed. Splitting the leaf node violates the trajectory preservation property so a new leaf node is created instead.

Algorithm FindNode (N,E)

1. **IF** node 'N' is not a leaf,

   FOR EACH entry 'E' of N whose MBB intersects with the MBB of E,

   Invoke FindNode ($N^{'}$,E), where $N^{'}$ is the childnode of N pointed to by $E^{'}$

ELSE

IF N contains an entry that is connected to E,

RETURN N

Figure 3.11 TB-Tree FindNode Algorithm

Intuitively, the TB-tree grows from left to right, i.e., the left-most leaf node was the least recent (first) and the right-most node is the most recent (last), we inserted. As shown in figure 3.9, the leaf nodes are connected with a superimposed linked list. This makes it possible to retrieve the complete or partial trajectory corresponding to an arbitrary leaf node, at minimal cost.

CHAPTER 4

FNR-TB TREE FOR INDEXING TRAJECTORIES

Spatio-temporal index is very important for efficient spatio-temporal algorithms. Referring to the advantages of the trajectory-oriented indexing method TB Tree [2], in this chapter, a new indexing structure Fixed Network R-Tree and Trajectory Bundle (FNR-TB) Tree is proposed which is an improvement over the Fixed Network R-Tree (FNR) approach. The strength of the TB-trees is that it preserves trajectories as well as allows for R-Tree typical range search in Euclidean spaces. The TB-Tree offers fast accesses to the trajectory information of moving objects. So using this data structure can improve the performance of the queries about the Trajectories. The insertion and search algorithms of the proposed approach are also discussed.

4.1 Index Structure

The index structure proposed is a modified version to FNR-Tree indexing structure for moving objects in fixed networks. The structure consists of a top level 2D R-Tree (the top R-Tree) whose leaf nodes contain pointers to TB-Trees (the original FNR-Tree had 1D R-Trees).

The 2D R-Tree is used to index the edges of the network, indexing their corresponding line segments. For every leaf node in the 2D R-Tree, there is a TB-Tree indexing the time interval of the objects traversing the line segments inside of its entries. The leaf node entries of the top level 2D R-Tree has entries of the form *{MBB, orientation}*, where *MBB* is the minimum bounding box of the line segment and

orientation is a flag that describes the two possible orientations that a line segment can assume inside a MBB.

Each leaf entry of the TB-Tree will be of the form *{MO_id, edgeid, enter_time,exit_time, direction}* where *MO_id* is the identification of the moving object, *edgeid* is the identification of the network edge, *enter_time* and *exit_time* represent the time interval when the moving object traversed the edge and *direction* is the flag that describes the direction of the moving object.

4.2 Insertion Algorithm

Figure 4.1 shows the insertion algorithm that is executed each time a moving object leaves a given segment of the network. It takes the line segment on which the object was moving, its direction, and the time interval when the object traversed the line segment. The algorithm first searches in the 2D R-Tree for the line segment and then inserts the time interval of the moving object into the associated TB-Tree. To insert a new entry into the TB-Tree, we have to first find the leaf node that contains its predecessor in the trajectory. The search starts from the root and steps into every child node that overlaps with the interval $t = (t_1, t_2)$.

FNR-TB Tree Insertion Algorithm

1. Execute Guttman's search algorithm in the 2D R-Tree in order to find the line segment, which leaves the moving object. Locate the leaf node that contains the given line segment and the corresponding TB-Tree
2. Execute insertion algorithm in the bottom level TB-Tree.

Figure 4.1 FNR-TB Tree Insertion Algorithm

4.3 Search algorithm

The search algorithm, shown in figure 4.2 consists of three steps. It receives a spatio-temporal query window $w = (x_1, x_2, y_1, y_2, t_1, t_2)$ and, in a first step finds all edges whose line segments overlap the rectangle $r = (x_1, x_2, y_1, y_2)$. Then, in the second step, for all the TB-Trees pointed to by the edge found in the first step, it looks for the moving objects traversing it with the time interval overlapping $t = (t_1, t_2)$. Finally, in the third step, the corresponding edge among those of the first step is searched by the *edgeid* information of the 1D R-Tree leaf node. The object's 3-dimensional movement is then re-composed and those parts that are fully outside of the spatio-temporal query window *w* are rejected. This query is commonly called range query in the literature. The main functionality of the FNR-TB Tree index is to answer these kinds of query.

FNR-TB Tree Search Algorithm

1. Execute Guttman's search algorithm in the 2D R-Tree in order to find the line segments of the network contained in the spatial query window. Locate the corresponding 2D R-Tree leaves. Store these line segments in main memory.
2. Execute Pfoser's search algorithm in each one of the TB-Trees that corresponds to the leaf nodes of the 1st step.
3. For each entry in every leaf of the TB-Trees that were retrieved from the 2nd step, locate via the line segment id stored in main memory – the corresponding line segment among those of the 1st step. Reject those entries that correspond to a line segment that is fully outside the spatial window of the spatio-temporal query.

Figure 4.2 FNR-TB Tree Search Algorithm

4.4 FNR-TB Tree Creation

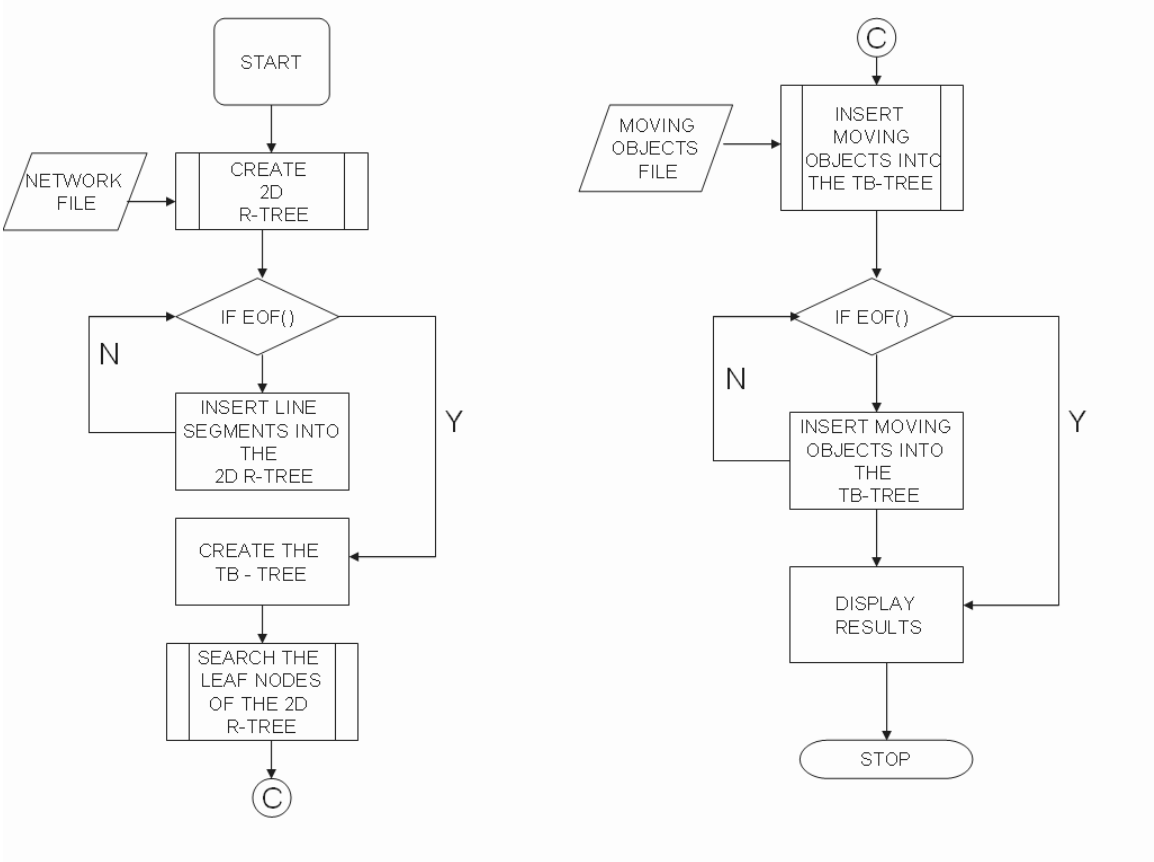

Figure 4.3 Creation of the proposed index structure

Figure 4.3 shows the various steps involved in the creation of the FNR-TB Tree and the subsequent range query processing. It starts with the creation of the top level 2D R-Tree whose input is the network file containing the trajectory segments. The next step is the creation of the bottom level TB-Tree whose input is the objects moving over the network. As a result of the creation of the index structure, parameters such as the number of node accesses, the total space utilization, the number of nodes of the FNR-TB Tree and the spatial and temporal extent are displayed.

The next step is the execution of range queries on the FNR-TB Tree. The X_{min}, X_{max}, Y_{min}, Y_{max}, T_{min} and T_{max} values are given as input. The number of node accesses to obtain the moving objects traveling over the network during the given interval is calculated.

CHAPTER 5

EXPERIMENTAL EVALUATION

The performance of the FNR-TB Tree is examined by experimental evaluation. For performance analysis, we compare the access scheme with the original FNR-Tree for moving object trajectories. The FNR-TB Tree and the original FNR-Tree are implemented using Microsoft Visual Basic 6.0. A set of experiments were conducted with the simulated FNR-TB Tree and the FNR-Tree.

5.1 Datasets

Unlike spatial data, where there exist several popular real datasets for experimentation purposes (e.g., the TIGER-Line files of geographic features, such as roads, rivers, lakes, boundaries covering the entire United States), well-known and widely accepted spatiotemporal datasets for experimental purposes are missing. The performance study consists of experiments on synthetic datasets. The GSTD generator of spatio temporal datasets is utilized to create trajectories of moving objects under various distributions. GSTD allows the user to generate a set of line segments stemming from a specified number of moving objects.

The performance study is based on synthetic datasets created using a network-based generator and the real world network of Oldenbourg [4]. While the output of the generator was of the form (id, t, x, y), the FNR-TB Tree implementation can utilize those data only if (x, y) are the co-ordinates of a node of the network. Therefore, the generator

was modified in order to produce a record of the form (id, t, x, y) each time a moving object was passing through each node of the network.

5.2 Experimental setup

A graphical user interface is used in the system to show the result of a spatiotemporal range query. The index structures viz, the existing FNR-Tree and the proposed FNR-TB Tree were implemented in Microsoft Visual Basic 6.0.

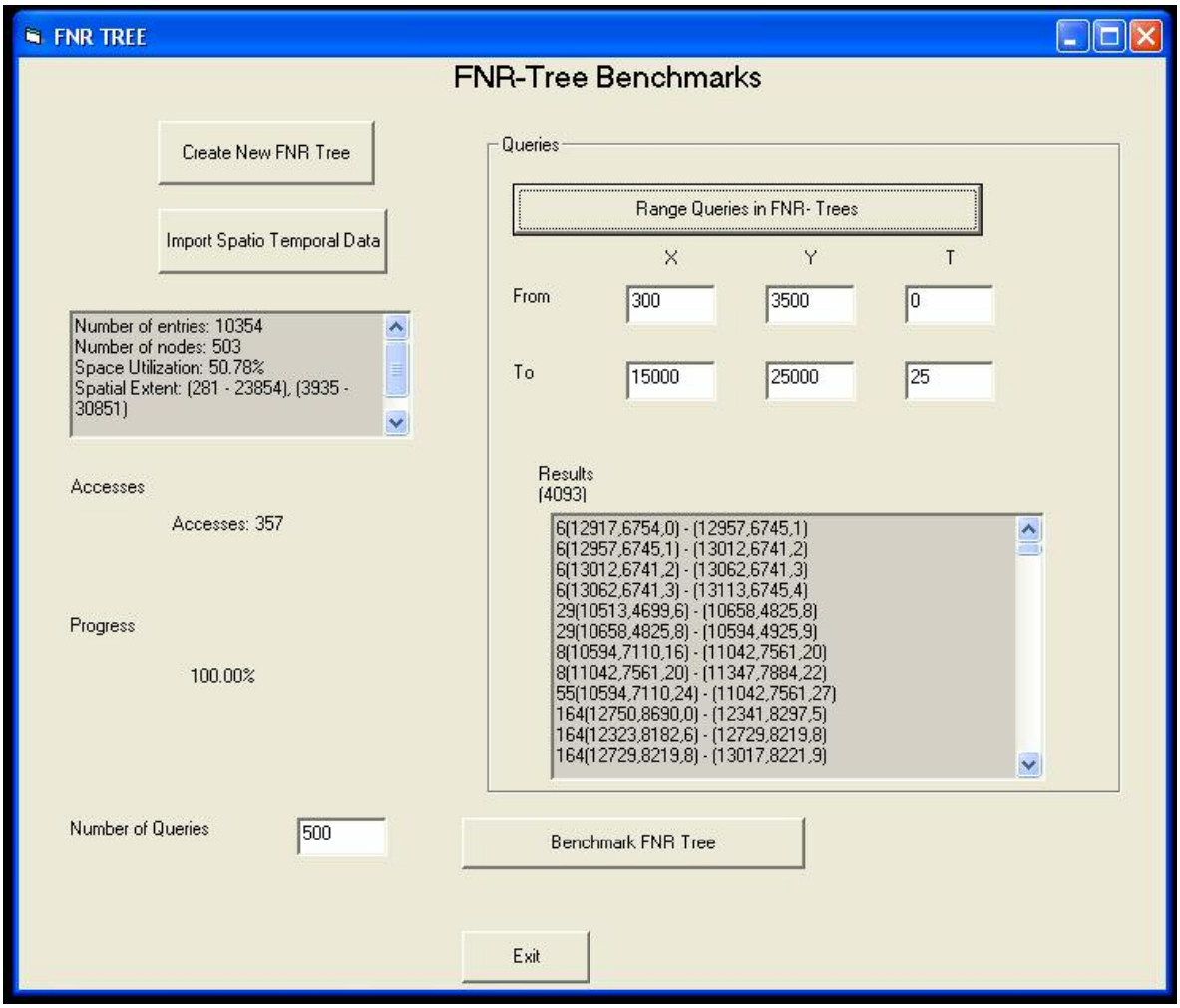

Figure 5.1: A screenshot of the FNR-Tree Prototype User Interface

Figure 5.1 and Figure 5.2 shows the Visual Basic interface of the FNR-Tree and FNR-TB Tree respectively executed with a range query. Range queries are important for both spatial and spatiotemporal data.

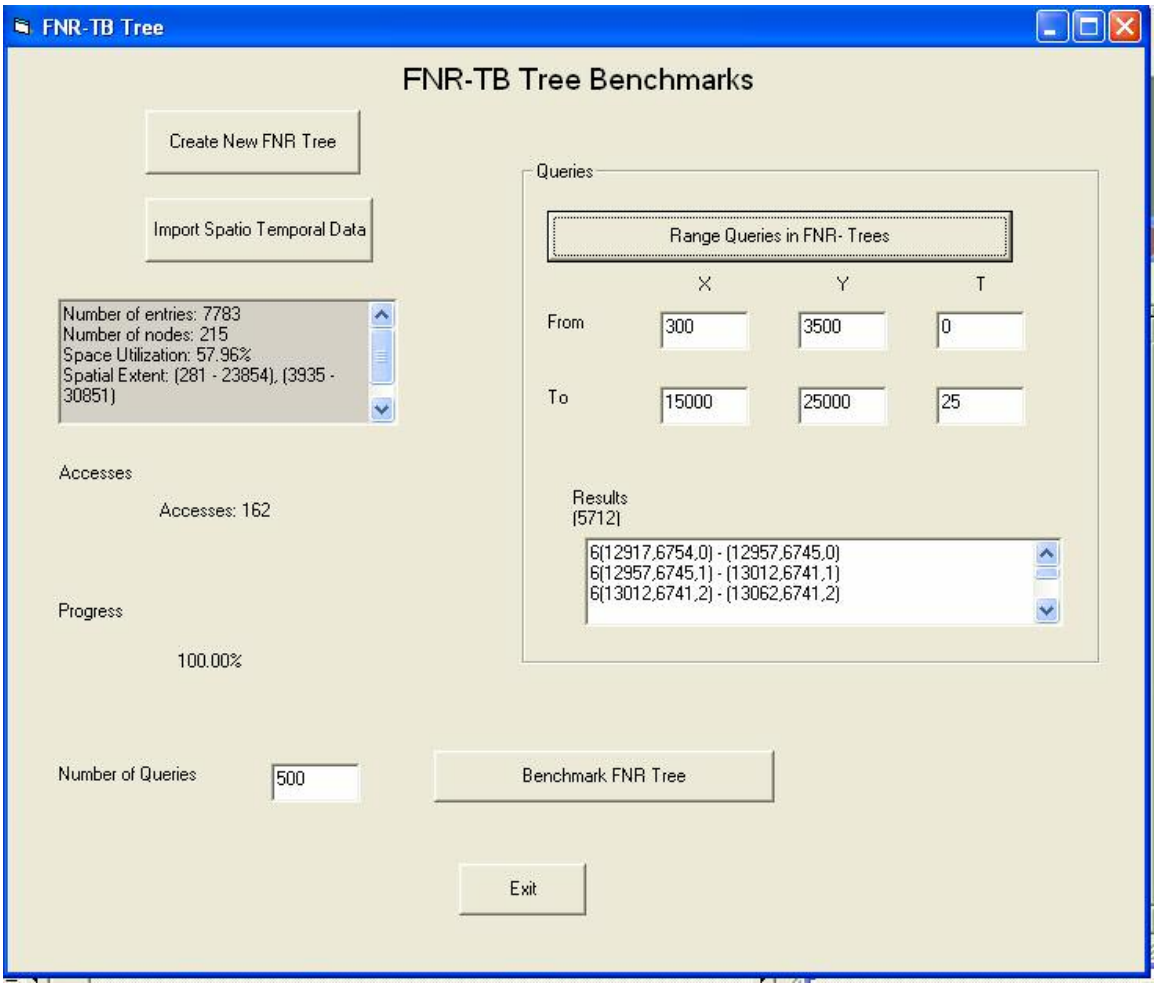

Figure 5.2: A screenshot of the FNR-TB Tree Prototype User Interface

5.3 Experimental Results

The results from the performance study are as follows. The page size for the leaf and non-leaf nodes is chosen to be 1024 bytes, which leads to a fanout of 50 for the leaf and non-leaf nodes of the 2D R-Tree. The fanout for the TB-Trees inside the FNR-TB Tree was 31 for the leaf nodes and 36 for the non-leaf nodes.

	FNR Tree	FNR-TB Tree
Index Size	~ 6 KB per object	~ 4 KB per object
Space Utilization	~ 96%	~ 98%

Table 5.1: Performance Comparison

The R-Tree in the FNR-Tree is roughly twice as big as the TB-Tree index in the FNR-TB Tree mainly because of the R-Tree's smaller space utilization. The average space utilization of R-Tree is between 55% and 60%, whereas it approaches 100% in case of the TB-Tree. Also the R-Tree does not know about the natural discrimination of data [4]. Its sole purpose is to group objects according to spatial characteristics i.e., spatial proximity. The TB-Tree puts connected segments in the same node and does not consider spatial discrimination. It thus exploits the temporal discrimination of the data.

The size of the trajectory index benefits the query performance. In this proposed method, the index is reduced to roughly a third since an index node contains four (in the top level 2D R-Tree) and two (in the bottom level TB-Tree), and not six coordinates per segment. Mapping trajectories from three to two dimensions reduce the dead space in the index.

Dead space is defined to be the extra portion of the embedding space covered by the approximating bounding box with respect to the space covered by the original object [14].

6.4 Range Query Performance

Query performance should benefit from the trajectory index. Range Queries are very important for spatiotemporal data. Range queries can be divided into two sub-categories: those having the same size in each dimension (spatial and temporal), and those with different size in the spatial and temporal dimensions.

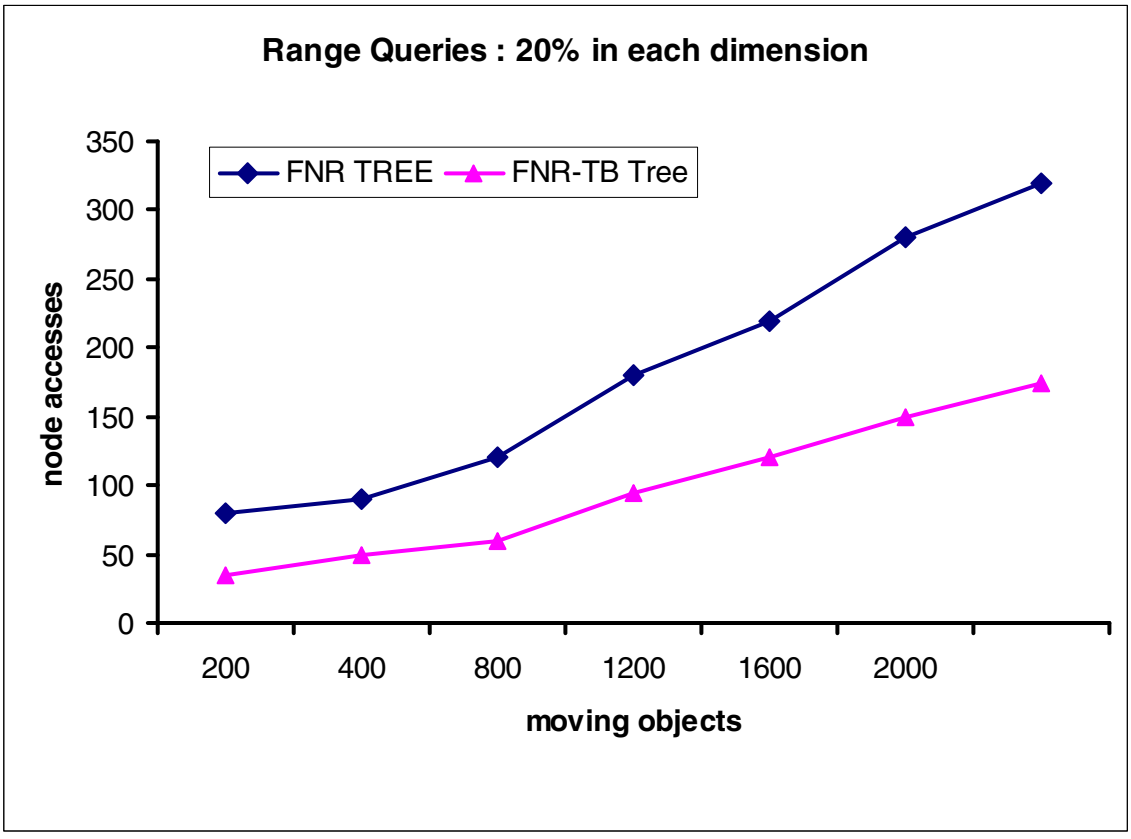

Figure 5.3: Range Query performance in FNR-TB and FNR- Tree

Figure 5.3 shows the range query processing of the trajectory data for the Oldenbourg network. Considering the insertion algorithm of the FNR-TB Tree, we can see that insertion requires a pre-searching of the top 2D R-Tree in order to find the Bottom TB-Tree in which to place the newly inserted line segment. Consequently, for each insertion there are a (stable) number of node accesses which correspond to the spatial search of the 2D R-Tree. So the number of node accesses increases with the size of the road network under consideration.

FNR-TB Tree shows better performance in the queries with larger temporal extent. This can be explained below: When a query is executed against the FNR-TB Tree, the search algorithm starts with the spatial search in the 2D R-Tree. Then, having located the corresponding TB-Trees inside the FNR-TB Tree, the performance of a searching among them, requires minimum node accesses, because the inserted data have already been sorted in ascending order and the formed TB-Trees have very low overlap and high space utilization and are also found to work well for trajectory based queries.

The experiments that were conducted in order to evaluate the performance of the FNR-TB Tree showed that it supports range queries much more efficiently than the original FNR-Tree. Also if the temporal extent is larger than the spatial extent, the performance of the FNR-TB Tree increases significantly. This helps in faster retrieval of trajectory segments and hence faster query processing. Hence the observation.

CHAPTER 6

CONCLUSION

6.1 Summary

A wide and increasing range of database applications has to deal with spatial objects whose position changes continuously over time, called moving objects. The main interest of these applications is to efficiently store and query the positions of these continuously moving objects. To achieve this goal, index structures are required. This thesis aims at developing a spatiotemporal access method suitable for objects moving on fixed networks and also at efficient retrieval of line segments representing trajectories of moving objects. Trajectory-based queries help in understanding the relationships between the various space-time paths and hence helps in their analysis. For this purpose, a number of access methods have been proposed based on the R-Tree method. Among these methods, TB-Tree (Trajectory Bundle Tree) is well suited for answering trajectory- based queries [2].

In this thesis, a new indexing technique called the FNR-TB Tree, a simple modification to the FNR-Tree is proposed for indexing the trajectories of moving point objects. One shortcoming of the FNR-Tree is the inability to answer topological (trajectory-based) queries. This kind of queries requires the preservation of each moving object's trajectory. This is taken care by the TB Trees in the FNR-TB Trees. The general idea that describes the FNR-TB Tree is a forest of several TB Trees on top of a single 2-dimeansional R-Tree. The 2D R-Tree indexes the spatial data of the network, while the TB-Trees are used

to index the time interval of each objects' movement inside the given link of the network. Unlike previously proposed trajectory indexing mechanisms, FNR-TB tree decouples the indexing of the spatial and the time dimensions which leads to greater search and update efficiencies.

In the conducted performance study, comparing this novel access method with the original Fixed Network R-Tree (FNR) under the available Oldenbourg dataset and Range queries, the proposed FNR-TB Tree approach was shown to outperform the FNR-Tree in most cases. The FNR-TB Tree has high space utilization and supports range queries much more efficiently.

6.2 Future Work

Work in the context of spatiotemporal query processing has dealt with the indexing of trajectories by proposing new access methods.

- The proposed FNR-TB approach focuses only on historical queries, so as a part of the future work this approach can be extended to answer queries on the future positions of moving objects.
- It would be interesting to use and study the performance of the TB-Tree with the MON-Tree, another index structure for moving objects on networks.
- Another challenging focus of future work is to apply this method in a practical setting e.g., using off-the-shelf database technology and applying the presented approach in an application context such as location-based services for mobile devices.

REFERENCES

1. Mohamed F. Mokbel, Thanaa M. Ghanem, and Walid G. Aref. Spatio-temporal Access Methods. IEEE Data Engineering Bulletin, 26(2):40-49, June 2003.

2. Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis. Novel Approaches in Query Processing for Moving Object Trajectories. In Proceeding of the International Conference on Very Large Data Bases, VLDB, pages 395-406, Cairo, Egypt, September 2000.

3. Zhexuan Song and Nick Roussopoulos. SEB-Tree: An Approach to Index Continuously Moving Objects. In Proceeding of the International Conference on Mobile Data Management, MDM, pages 340-344, Melbourne, Australia, January 2003.

4. Elias Frentzos. Indexing Objects Moving on Fixed Networks. In Proceeding of the International Symposium on Advances in Spatial and Temporal Databases, SSTD, pages 289-305, Santorini Island, Greece, July 2003.

5. V. Prasad Chakka, Adam Everspaugh, and Jignesh M. Patel. Indexing Large Trajectory Data Sets with SETI. In Proceeding of the International Conference on Innovative Data Systems Research, CIDR, Asilomar, CA, January 2003.

6. Dieter Pfoser and Christian S. Jensen. Trajectory Indexing Using Movement Constraints. GeoInformatica, 9(2):93-115, June 2005.

7. George Kollios, Dimitrios Gunopulos, and Vassilis J. Tsotras. On Indexing Mobile Objects. In Proceeding of the ACM Symposium on Principles of Database Systems, PODS, pages 261-272, Philadelphia. PA, May 1999.

8. Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. Indexing the Positions of Continuously Moving Objects. In Proceeding of the ACM International Conference on Management of Data, SIGMOD, pages 331-342, Dallas, TX, May 2000.

9. V. T. de Almeida and R. H. Guting. Indexing the trajectories of moving objects in networks. In Proc. of the 16th Intl. Conf. on Scientific and Statistical Database Management (SSDBM), 2004

10. Hongbo Yu and Shih-Lung Shaw. Representing and Visualizing Travel Diary Data: A Spatio-temporal GIS Approach. *Proceedings of the 2004 ESRI International User Conference*, available at http://www.esri.com/library/userconf/archive.html.

11. Hae Don Chon, Divyakant Agrawal, and Amr El Abbadi. Range and kNN Query Processing for Moving Objects in Grid Model. Mobile Networks and Applications, 8(4):401–412, 2003.

12. Eija Kaasinen. User Needs for Location-Aware Mobile Services. Personal Ubiquitous Computing, 7(1):70–79, 2003.

13. D. Papadopoulos, G. Kollios, D. Gunopulos, and V. J. Tsotras. Indexing mobile objects on the plane. In Proc. of the 13th Intl. Workshop on Database and Expert Systems Applications (DEXA), pages 693{697, 2002.

14. Y. Manolopoulos., Y. Theodoridis, and V. Tsotras.: Advanced Database Indexing. Kluwer Academic Publishers, Boston/Dordrecht/London, 2000.

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

16. A. Guttman.: R-Trees: a dynamic index structure for spatial searching, In Proceedings of the 13th Association for Computing Machinery SIGMOD Conference, 1984, 47-57.

17. Eija Kaasinen. User Needs for Location-Aware Mobile Services. Personal Ubiquitous Computing, 7(1):70–79, 2003.

18. Mario A. Nascimento, Dieter Profser, and Yannis Theodoridis. Synthetic and Real Spatiotemporal Datasets. IEEE TCDE Bulletin, 26(1):26–32, 2003.

19. Yannis Theodoridis, Jefferson R. O. Silva, and Mario A. Nascimento. On the Generation of Spatiotemporal Datasets. In Proc. of SSD, pages 147–164, 1999.

20. Thomas Brinkhoff. A Framework for Generating Network-Based Moving Objects. Geoinformatica, 6(2):153–180, 2002.

21. T. Tzouramanis, M. Vassilakopoulos, and Y. Manolopoulos. C.P. Kolovson. ``Indexing Techniques for Historical Databases,'' In Tansel et al., editors, Temporal Databases - Theory, Design, and Implementation, Benjamin/Cummings, 1993.

22. On the Generation of Time-Evolving Regional Data. Geoinformatica, 6(3):207–231, 2002.

23. M. Mokbel., T. Ghanem., W. Aref.: Spatio –Temporal Access Methods. IEEE TCDE Bulletin 26 (2003) 40-49

APPENDICIES

APPENDIX A

EXPLANATION OF THE INTERFACE

An implementation of the original FNR-Tree in Figure 5.1 and the proposed FNR-TB Tree in Figure 5.2 on Microsoft Visual Basic 6.0 Platform is shown. The working of the various parts of the implementation is as follows

1. First, the top level R- tree in the 2-dimensional space is created (2D R-Tree) by loading the data from the network file.
2. Next step is the creation of the bottom level Trees (1D R-Trees for FNR Tree and TB-Trees for FNR-TB Tree) in the time dimension by loading the spatio temporal moving objects details from another file.
3. The various results calculated and obtained after loading the data include
    - the number of entries which is the number of moving objects in the 3D space
    - the number of nodes (internal and child) containing the entries
    - the space utilization which shows the space saved as a result of the proposed tree structure for indexing spatio-temporal data
    - the spatial and temporal extent of the trees obtained from the road network file and
    - the total number of node accesses for the entire tree creation.

4. Range queries are executed on the FNR-Tree and the FNR-TB Tree by specifying the X, Y, T dimensions for the query window (Minimum Bounding Box)
5. The number of access for the nodes satisfying the given query conditions are displayed. The data (segments of the trajectory) satisfying the given spatio-temporal query conditions are calculated. The results are displayed in the text box and the number of spatio – temporal moving objects found within the query window are displayed in the Results label.
6. The FNR-Tree as well as the FNR-TB Tree is tested by generating random values for the X, Y and T Dimensions on over 500, 1000 and 2000 moving objects. The node accesses for the sample number of queries are obtained.

APPENDIX B

CODE LISTING

R-TREE INSERTION

```
Public Sub Insert(ByRef X1 As Long, ByRef Y1 As Long, ByRef X2 As Long,
ByRef Y2 As Long, ByRef EntryId As Long)

' The Insertion Method of the R-Tree.

' XMin, XMax, YMin, YMax are the extents of the Line Segment inserted
in the R-Tree
    Dim XMin As Long, XMax As Long, YMin As Long, YMax As Long
' calculate the min and max in each dimension
    XMin = Min(X1, X2)
    YMin = Min(Y1, Y2)
    XMax = Max(X1, X2)
    YMax = Max(Y1, Y2)

' EntryId is the Id of the Line Segment
  Dim Orientation As Boolean
' Calculate the value of "Orientation"
' If the line segment is in the form ("/","|","-") then, the _
  Orientation becomes True
    If (XMin = X1 And YMin = Y1) Or (XMin = X2 And YMin = Y2) Then
        Orientation = True
    Else
    ' If the line segment is in other form ("\") then, _
    the Orientation becomes False
        Orientation = False
    End If

'set the internal number of accesses to be 0
    MyAccesses = 0
' 1st Step of the algorithm INSERT described by Antonin Guttman _
  in the R-Tree paper.
' We choose the Node in which to place the newly _
  inserted entry based on the least enlargement criterion _
  (Algorithm ChooseLeaf by Antonin Guttman)
    Dim Node As R2TreeNode
    Set Node = ChooseLeaf(XMin, XMax, YMin, YMax)
' 2nd Step of the algorithm INSERT described by Antonin Guttman _
  in the R-Tree paper.

' Create a new entry (Ent)
    Dim Ent As R2TreeEntry
    Set Ent = New R2TreeEntry
' Asign the appropriate properties (The MBB of the entry defined by _
  XMin, XMax, YMin, YMax)
    Ent.ChildXMin = XMin
    Ent.ChildXMax = XMax
```

```
   Ent.ChildYMin = YMin
    Ent.ChildYMax = YMax
    Ent.EntryId = EntryId
    Ent.Orientation = Orientation

' If the Node selected by ChooseLeaf has room for another entry, _
  install the new entry in it
    If Node.Entries.Count < MaxEntriesLeaf Then
        Node.AddEntry Ent
        '3rd Step of the algorithm INSERT described by Antonin Guttman
_
        in the R-Tree paper.
        ' Propagate Changes upwards using AdjustTree algorithm
        AdjustTree Node
    Else
' If the Node has not room for another entry, invoke SplitNode
Algorithm _
  to obtain "Node" and a new node ("NNode"). Both nodes contain the _
  "Ent" and all the old entries of "Node"
        Dim NNode As R2TreeNode
        SplitNode Node, NNode, Ent

        '3rd Step of the algorithm INSERT described by Antonin Guttman
_
        in the R-Tree paper.
        ' Propagate Changes upwards using AdjustTree algorithm, passing
_
        also NNode because of the previous performed split.
        Set Node = AdjustTree(Node, NNode)

        ' 4th step of the Insert Algorithm (grow tree taller)
        'Create a new root whose children are the old root node _
        and the new one resulted from the split propagation

        ' AdjustTree algorithm returns a node only if the node _
        split propagation caused the Root to split
        If Not Node Is Nothing Then

            'Create a new node
            Dim NewRootNode As R2TreeNode
            Set NewRootNode = New R2TreeNode
            'increase the accesses by 1 because create a new node
            MyAccesses = MyAccesses + 1
            ' The new node belongs to the current tree
            Set NewRootNode.ParentR2Tree = Me
            ' Create two new entries (NewEntry1, NewEntry2) of the new
root
            Dim NewEntry1 As R2TreeEntry, NewEntry2 As R2TreeEntry
            Set NewEntry1 = New R2TreeEntry
            Set NewEntry2 = New R2TreeEntry
            ' Set the one new entry pointing to the old tree root
            Set NewEntry1.ChildNode = RootNode
            ' adjust the covering rectangle of this first new entry
            NewEntry1.ChildXMin = RootNode.XMin
            NewEntry1.ChildXMax = RootNode.XMax
```

```
            NewEntry1.ChildYMin = RootNode.YMin
            NewEntry1.ChildYMax = RootNode.YMax

            ' Set the second new entry pointing to the resulted node
            Set NewEntry2.ChildNode = Node
            ' adjust the covering rectangle of the second new entry
            NewEntry2.ChildXMin = Node.XMin
            NewEntry2.ChildXMax = Node.XMax
            NewEntry2.ChildYMin = Node.YMin
            NewEntry2.ChildYMax = Node.YMax

            ' Insert the two new entries in the new Root
            NewRootNode.AddEntry NewEntry1
            NewRootNode.AddEntry NewEntry2

            ' Set the Tree's RootNode to be the new Root
            Set RootNode = NewRootNode
            'increase the accesses by 1 because we modify the tree's
root
            MyAccesses = MyAccesses + 1

        End If

    End If

End Sub
```

TB-TREE INSERTION

```
Public Sub Insert (ByRef MovingObjectId As Long, ByRef LineSegmentId As
Byte, ByRef T1 As Long, ByRef T2 As Long, EntryPoint As Boolean)
' the insertion method of the TB Tree
' calculate the extent of the intervals( time) inserted in the TB-Tree
Dim TMin As Long
Dim TMax As Long
TMax = Max(T1, T2)
TMin = Min(T1, T2)

' accesses and entries of the TB-Tree
MyAccesses = 0
MyEntriesCount = MyEntriesCount + 1


 'declare a node of the TB-Tree(which can be a root or internal node)
Dim node As Variant


' declare a new entry
Dim NewEntry As TBTreeLeafEntry
Set NewEntry = New TBTreeLeafEntry

'assign the properties to the node
NewEntry.EntryPoint = EntryPoint
NewEntry.LineSegmentId = LineSegmentId
```

```
NewEntry.MovingObjectId = MovingObjectId

'set the MBR properties for time
NewEntry.MaxPoint = TMax
NewEntry.MinPoint = TMin

If NewEntry.MinPoint = T1 Then
        NewEntry.Orientation = 1
    ElseIf NewEntry.MinPoint = T1 And _
           NewEntry.MaxPoint = T1 Then
        NewEntry.Orientation = 2
    ElseIf NewEntry.MaxPoint = T1 And _
           NewEntry.MinPoint = T1 Then
        NewEntry.Orientation = 3
    ElseIf NewEntry.MaxPoint = T1 Then
        NewEntry.Orientation = 4
End If


' in the original TB-tree we choose the node in which to place _
the new enrty  only by finding the node containing an entry belonging _
to the same moving object and connected to the new entry
' find a leaf node to insert the new segment, that _
  contains its predecessor in the trajectory

Set node = ChooseLeaf(MovingObjectId, TMin, TMax)

' if ChooseLeaf returns a node then place the entry in the new node _
else create a new node for the moving object
' if this is the first entry of the node, then the node holds the
moving object id
If MyEntriesCount = 1 Then
   node.Id = MovingObjectId
End If

' if the ChooseLeaf returned an existing node i.e., entriescount > 0,
and _
the node has room for another entry, install the entry in it
If (node.EntriesCount > 0 And node.EntriesCount < Me.MaxLeafEntries) Or
_
    MyEntriesCount = 1 Then
    node.EntriesCount = node.EntriesCount + 1
    node.AddEntry NewEntry

    ' propagate changes upwards using AdjustTree Algorithm
       AdjustTree node
Else
```

```
' create a new node
' set the Nnode's moving object id to be the current MovingObjectId

    Dim Nnode As Variant
    Set Nnode = New TBTreeLeaf
   ' Set Nnode.ParentTBTree = Me
        Nnode.Id = MovingObjectId
        Nnode.IsLeaf = 1

    ' add the new entry in the NNode
    Nnode.EntriesCount = Nnode.EntriesCount + 1
    Nnode.AddEntry NewEntry



'propagate changes upwards using AdjustTree Algorithm, passing also _
NNode because of the previous performed split
'Set node = ChooseLastLeaf()
  'Set node = AdjustTree(node, Nnode)
  AdjustTree Nnode

  'update the pointers from and to the new node
    MyNextNode = MyNextNode + 1
    MyNodesCount = MyNodesCount + 1
    Nnode.ptrCurrentNode = MyNextNode
    Nnode.ptrPreviousNode = node.ptrCurrentNode
    node.ptrNextNode = Nnode.ptrCurrentNode

' adjust tree algorithm retuns a node only if the node _
split propagationcaused the root to split
If Not IsEmpty(node) Then
    'create a new node
    Dim NewRootNode As TBTreeNode
    Set NewRootNode = New TBTreeNode
    MyNextNode = MyNextNode + 1
    MyNodesCount = MyNodesCount + 1
    NewRootNode.ptrCurrentNode = MyNextNode

    ' create two new entries ( NewEntry1, newEntry2) of the new root
    Dim NewEntry1 As TBTreeNodeEntry
    Dim NewEntry2 As TBTreeNodeEntry
    Dim OldRootNode As TBTreeNode

    ' NewEntry1.ptrChildNode = RootNode
    ' set the one new entry pointing to the old tree root
   ' Set OldRootNode = New TBTreeNode
   ' Set OldRootNode = RootNode
   ' OldRootNode.ptrParentNode = MyNextNode

    MyAccesses = MyAccesses + 1
    Set NewEntry1 = New TBTreeNodeEntry
     Set NewEntry2 = New TBTreeNodeEntry
    'NewEntry1.MBR = CoveringMBR(RootNode)
   ' Set
'   NewEntry1.ptrChildNode = RootNode
    NewEntry1.MaxPoint = RootNode.TMax
```

```
    NewEntry1.MinPoint = RootNode.TMin

    'set the second entry pointing to the resulted node
'    NewEntry2.intNode = node.ptrCurrentNode
    'NewEntry2.MBR = CoveringMBR(node)
'    node.ptrParentNode = MyNextNode
    NewEntry2.MaxPoint = node.TMax
    NewEntry2.MinPoint = node.TMin

    NewRootNode.EntriesCount = 2
    NewRootNode.AddEntry NewEntry1
    NewRootNode.AddEntry NewEntry2

    'set the Tree's RootNode ot be the new root
  '  RootNode = NewRootNode.ptrCurrentNode
    NewRootNode.IsLeaf = 0
    'MyHeight = MyHeight + 1
    'MyAccesses = MyAccesses + 1
   End If
End If
```

VITA

Sowjanya Sunkara

Candidate for the Degree of

Master of Science

Thesis: AN EFFICIENT INDEXING STRUCTURE FOR TRAJECTORIES IN GEOGRAPHICAL INFORMATION SYSTEMS

Major Field: Computer Science

Biographical:

Personal Data: Born in Chennai, TamilNadu, INDIA on June 24, 1982

Education:

Received B.S (Engineering) degree in Computer Science from the University of Madras, Chennai, TamilNadu, India in 2004.
Completed the requirements for the degree of Master of Science with a major in Computer Science at Oklahoma State University, Stillwater, Oklahoma in May, 2007

Experience:

- Teaching / Graduate Assistant, Oklahoma State University, Stillwater, Oklahoma
- GIS Application Developer, PhotoScience, St.Petersburg, Florida

Name: Sowjanya Sunkara Date of Degree: May, 2007

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: AN EFFICIENT INDEXING STRUCTURE FOR TRAJECTORIES IN GEOGRAPHICAL INFORMATION SYSTEMS

Pages in Study: 56 Candidate for the Degree of Master of Science

Major Field: Computer Science

Technologies dealing with location such as GPS are producing more and more data of moving objects. Spatio-temporal databases store information about the positions of individual objects over time. Real-world applications of spatio-temporal data include vehicle navigation, migration of people, tracking and monitoring air-based, sea or land-based vehicles. Also the location technologies, such as GPS and telegraphy, are producing more and more data of moving objects. Spatio-temporal database is needed to manage these data, so as to solve the problems in spatio-temporal applications. A spatio-temporal database adopts an exhaustive search strategy for querying the trajectories. This is very time-consuming when processing large datasets for the given spatio-temporal query conditions. As a result, efficient Spatio-Temporal indexing methods are highly demanded to improve the performance of the system in searching such large datasets.

Referring to Fixed Network R-Tree (FNR-Tree) and the trajectory oriented indexing method Trajectory Bundle-Tree (TB-Tree), in this thesis, an indexing structure is proposed for trajectories to efficiently support spatiotemporal called the FNR-TB Tree. This indexing structure will help in analyzing space-time relationships among space-time paths, an example of spatio-temporal data, in a large dataset. This technique discriminates the space dimension from the time dimension for indexing and thus is expected to give better performance than the existing methods. Furthermore, since this method utilizes the advantages of the TB-Trees, it yields a better performance for spatiotemporal queries than other existing methods for retrieving trajectories.

ADVISER'S APPROVAL: Dr. Xiaolin (Andy) Li