Indexing Objects Moving on Fixed Networks

Elias Frentzos

Department of Rural and Surveying Engineering, National Technical University of Athens,
Zographou, GR-15773, Athens, Hellas
efrentzo@central.ntua.gr

Abstract. The development of a spatiotemporal access method suitable for objects moving on fixed networks is a very attractive challenge due to the great number of real-world spatiotemporal database applications and fleet management systems dealing with this type of objects. In this work, a new indexing technique, named Fixed Network R-Tree (FNR-Tree), is proposed for objects constrained to move on fixed networks in 2-dimensional space. The general idea that describes the FNR-Tree is a forest of 1-dimensional (1D) R-Trees on top of a 2-dimensional (2D) R-Tree. The 2D R-Tree is used to index the spatial data of the network (e.g. roads consisting of line segments), while the 1D R-Trees are used to index the time interval of each object's movement inside a given link of the network. The performance study, comparing this novel access method with the traditional R-Tree under various datasets and queries, shows that the FNR-Tree outperforms the R-Tree in most cases.

1 Introduction

Most recently developed spatiotemporal access methods (3D R-Tree [16], HR-Tree [6, 7], TB-Tree, STR-Tree [10], TPR-Tree [13], Mv3R-Tree [14], PPR-Tree [3]) are designed to index objects performing any kind of movement in a two-dimensional space. These methods are general and do not take into consideration the special requirements of the application in which they are used. However, in real-world applications several conditions having to do with the demanded level of generalization and the way of perception of data, can improve the performance of the spatiotemporal indexes. For example, in a research for the emigrational habits of some animal population (dolphins, whales e.t.c.), more likely is that the precise position of each separate animal is not of interest, but the region in which is contained the population. This can lead to dramatic reduction of the number of moving objects that should be indexed by an indexing technique, and to the consequently increase of its performance.

Another condition that can be used to improve the performance of spatiotemporal indexes is the existence of restrictions in the space in which moving objects realize their movement. Although the vast majority of spatiotemporal access methods index objects moving in the whole of the two dimensional space, in most real-world applications the objects are moving in a constrained space: planes fly in air-paths, cars and pedestrians move on road networks, while trains have fixed trajectories on railway networks. These kind of special conditions (moving restrictions) have recently been subject of the researching interest [5, 8, 9, 11].

As it is obvious, the development of a spatiotemporal access method suitable for objects moving on fixed networks is a very attractive challenge because a great number of real-world spatiotemporal database applications have to do mainly with objects (e.g. cars) moving on fixed networks and fleet management systems. In the next sections of this paper, we focus on the movement of such objects constrained to move on a fixed urban network and we study how we can use this restriction in order to develop alternative access methods to index those data.

Specifically, we develop and evaluate the performance of a new index, named Fixed Network R-Tree (FNR-Tree), proposed for objects constrained to move on fixed networks in a two dimensional space; an extension of the well-known R-Tree [2]. The general idea that describes the FNR-Tree is a forest of 1 dimensional (1D) R-Trees on top of a 2 dimensional (2D) R-Tree. The 2D R-Tree is used to index the spatial data of the network (e.g. roads consisting of line segments), while the 1D R-Trees are used to index the time interval of each object's movement inside a given link of the network.

The outline of this paper is as follows. Section 2 focuses on the motivation for the development of a new access method and summarizes the related work. Section 3 presents the structure and the algorithms of the FNR-Tree. Section 4 provides a performance study for the FNR-Tree and compares it with the conventional 3D R-Tree [16]. Finally, section 5 presents conclusions and directions for future work.

2 Motivation

Much work has been recently conducted in the domain of indexing spatiotemporal data and several spatiotemporal access methods have been proposed, which can be separated into two major categories: those indexing the past positions of moving objects (3D R-Tree [16], HR-Tree [6], TB-Tree, STR-Tree [10]), and those indexing the current and predicted movement of objects [12, 13]. Because our interest is in recording the past positions of moving objects in order to use them for post-processing purposes, we will only consider access methods of the first type.

According to [8], there are three scenarios about the movement of objects in a two 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 and, typically, cars). Generalizing the second two categories, we can say that the way in which an urban network is handled depends of the required precision of the application. For example, if the application requires the precise knowledge of the position of each moving object, then the network can be faced as in Figure 1(a), where the existing infrastructure (building squares in gray regions) restricts the movement of objects. On the contrary, if the application does not require the precise knowledge of the position of each moving object, the same area can be represented by a set of connected line segments forming a road network.

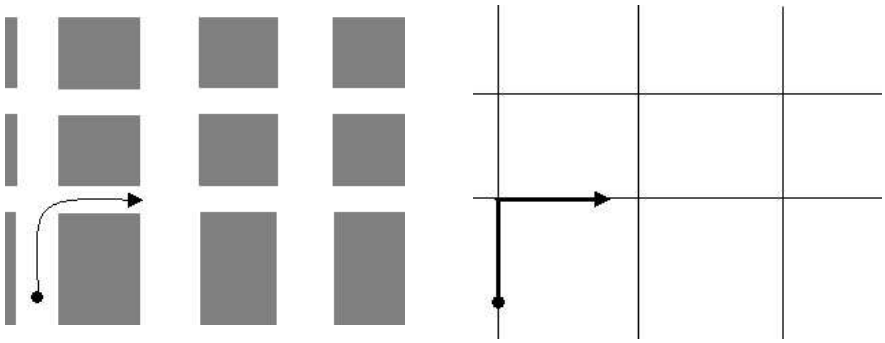


Fig. 1. (a) Movement of object restricted by the infrastructure in urban area, (b) Movement of object on fixed network representing the same urban area

Pfoser et al. [9] use the restrictions placed in the movement of objects by the existing infrastructure, in order to improve the performance of spatiotemporal queries executed against a spatiotemporal index. The strategy followed does not affect the structure of the index itself. Instead, they adopt an additional preprocessing step before the execution of each query. In particular, provided that the infrastructure is rarely updated, it can be indexed by a conventional spatial index such as the R-Tree. TB-Tree [10] or 3D R-Tree [16] can be used to index the pure spatiotemporal data. Then, a preprocessing step of the query, divides the initial query window in a number of smaller windows, from which the regions covered by the infrastructure have been excluded. Each one of the smaller queries is executed against the index returning a set of candidate objects, which are finally refined with respect to the initial query window.

In the same paper, an algorithm is provided for the implementation of the query-preprocessing step, based on the work presented in [4]. According to that, the number of node accesses required by an R-tree-based index to answer a window query, depends not only by the window area as well as from its dimensions. Consequently, what concerns is not only the minimization of the query area (which is achieved by removing the section containing the infrastructure from the initial window) but also the minimization of the perimeter of the query window. In the corresponding evaluation, the performance of two spatiotemporal indexes (TB and 3D R-tree) was compared, using or not the described query preprocessing step (i.e., dividing the initial window in smaller windows), and it was shown that the query performance was improved for both indexes.

As previously reported, there exist applications that do not require to keep track of each moving object's exact position, by terms of coordinates (Figure 1(b)); it is enough to project the object's position onto a fixed network constituted by a set of connected line segments. This type of application is very common, for example, management of vehicles moving in an urban road network (fleet-management applications) does not require each vehicle's precise position but its relative location inside the road network. Thus, the trajectory of a vehicle could be represented by a set $\{[route_1, t_1], [route_2, t_2], ...[route_i, t_i]\}$, where each pair denotes the time of entry of a given vehicle in network's line segment $route_i$.

According to Kollios et al. [5], the domain of the object's trajectories moving in a network is not the 2+1 dimensional space, but a space with 1.5 dimensions, as line segments consisting the network can be stored in a conventional index of spatial data (such as the R-tree). Then, indexing of objects moving in such a network is a problem with only one dimension. Though [5] consider the indexing of objects moving on fixed networks from the theoretical aspect, they do not propose any access method that could be used in real-world applications.

Recently, Papadias et al. [11] adopted the same assumption of movement on fixed networks, in order to create a structure that answers spatiotemporal aggregate queries of the form "*find the total number of objects in the regions intersecting some window* q_s *during a time interval* q_t". The Aggregate R-B-Tree (aRB-Tree) is a combination of R- and B-trees and is based on the following idea: the lines of the network are stored only one time and indexed by an R-tree. Then, in each internal and leaf node of the R-Tree, a pointer to a B-tree is placed, which stores historical aggregate data about the particular spatial object (e.g. the MBB of the node).

Pfoser in [8] proposes a method for simplifying the trajectory data obtained by objects moving on fixed networks. A two-dimensional network is transformed to a one-dimensional set of segments and the trajectory data is mapped accordingly. Consequently, a simple two-dimensional access method (such as R-Tree) can be used in order to index the positions of objects moving on a fixed network. Although this is an easy to implement approach, one may argue that it has disadvantages such as the modification of line segments distribution in space and the modification in general of the distance and the topological relations between spatial objects (i.e. connections between network's line segments).

On the contrary, the proposed FNR-Tree does not have these disadvantages because it is not based in any transformation technique. Moving objects' positions are recorded each time an object passes a node of the network. It assumes that moving parameters do not change between two network nodes, and therefore, linear interpolation can be used in order to calculate each object's position on a given time instance. In the following section, we will present the structure and the algorithms of the FNR-Tree and then we will show that it outperforms the 3D R-Tree for a wide palette of queries.

3 The Fixed Network R-Tree (FNR-Tree)

The FNR-Tree is based on the well-known R-Tree [2]. The general idea is that for a road network consisting of *n* links, the FNR-Tree can be considered as a forest of several 1D R-Trees on top of a single 2D R-Tree. The 2D R-Tree is used to index the spatial data of the network (e.g. roads consisting of line segments), while each one of the 1D R-Trees corresponds to a leaf node of the 2D R-Tree and is used to index the time intervals that any moving object was moving on a given link of the network. Therefore, the 2D R-Tree remains static during the lifetime of the FNR-Tree – as long as there are no changes in the network.

3.1 The FNR-Tree Structure

Before we describe in detail the structure of the proposed index, let us recall the structure of the original R-Tree. It is a height-balanced tree with the index records in its leaf nodes containing pointers to the actual data objects. Leaf node entries are in the form (id, MBB), where id is an identifier that points to the actual object and MBB (Minimum Bounding Box) is a n-dimensional interval. Non-leaf node entries are of the form (ptr, MBB), where ptr is a pointer to a child node, and MBB the bounding box that covers all child nodes. A node in the tree corresponds to a disk page and contains between m and M entries.

The 2D R-Tree inside the FNR-Tree, is slightly different: Since the only spatial objects that are inserted in the 2D R-Tree are line segments, leaf node entries can be modified in the form (*MBB, orientation*), where "*orientation*" is a flag that describes the exact geometry of the line segment inside the MBB and takes values from the set {0,1} (Figure 2(a)). A similar approach was followed in [10] to represent segments of trajectories in a 3D R-tree-based technique. The structure of each leaf node entry of the 2D R-Tree inside the FNR-Tree is of the form *(Line Id, MBB, Orientation)*, and the structure of each non-leaf (internal) node of the 2D R-Tree inside the FNR-Tree, is of the form *(Pointer to child Node, MBB)*. Each leaf node of the 2D R-Tree contains a pointer that points to the root of a 1D R-Tree; consequently, for each leaf node of the 2D R-Tree, there is a corresponding 1D R-Tree.

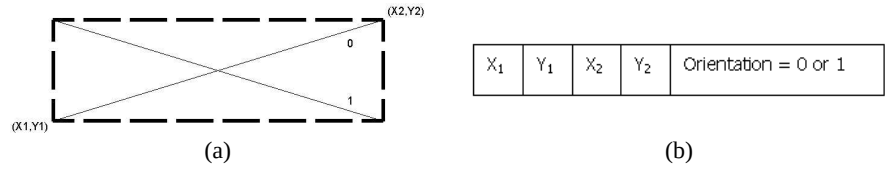


Fig. 2. (a) Ways that a line segment can be contained inside a MBB and the orientation flag, (b) The corresponding leaf node entry of the 2D R-Tree inside the FNR-Tree

Each leaf entry of an 1D R-Tree contains the time intervals that a moving object was moving on a line segment included in the MBB of the corresponding leaf of the 2D R-Tree, the *ID* of the moving object, the *ID* of the corresponding line segment and the *direction* of the moving object. The *direction* is another flag that describes the direction of the moving object and takes values from the set {0, 1} (Figure 3). Specifically, the *direction* becomes 0 when the moving object inserted in the line segment from the left-most node, and 1 when the moving object inserted from the right-most node. In the special case where the line segment is vertical, the *direction* becomes 0 for object inserted in the line segment from the bottom-most node and 1 for objects inserted from the top-most node. The structure of each leaf node entry of the 1D R-Trees inside the FNR-Tree is of the form *(Moving Object Id, Line Segment Id, $T_{entrance}$, T_{exit}, Direction)*. The structure of each internal (non-leaf) node entry of the 1D R-Trees inside the FNR-Tree is of the form *(Pointer to child Node, $T_{entrance}$, T_{exit})*.

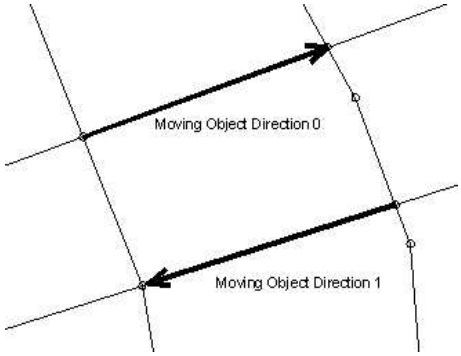


Fig. 3. The direction flag

3.2 The FNR-Tree Algorithms

The insertion algorithm of the FNR-Tree is executed each time a moving object leaves a given line segment of the network. It takes as arguments the 3-dimensional interval defined by the coordinates of the start and the end of the line segment (X_{Start}, X_{End}, Y_{Start}, Y_{End}), the direction of the moving object according to what was previously reported, as well as the time interval while the moving object was moving inside the given line segment ($T_{Entrance}$, T_{Exit}). The insertion algorithm is illustrated in figure 4.

FNR-Tree Insertion Algorithm

1. Execute Guttmann'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 1D R-Tree.
2. Execute insertion algorithm in the 1D R-Tree.

Fig. 4. FNR-Tree Insertion Algorithm

For the insertion algorithm of the 1D R-Tree, a choice could be Guttman's insertion algorithm, which is executed in three steps: ChooseLeaf algorithm selects a leaf node in which to place a new index entry. In case an insertion causes a node split, one of three alternatives split algorithms (Exhaustive, QuadraticSplit, LinearSplit) is executed. Finally, changes in the leaf MBB are propagated upwards by the AdjustTree algorithm. ChooseLeaf and Split are based upon the least enlargement criterion and are designed to serve spatial data that are inserted in the R-Tree in random order. Trees formed using these algorithms usually have small overlap between nodes as well as small space utilization.

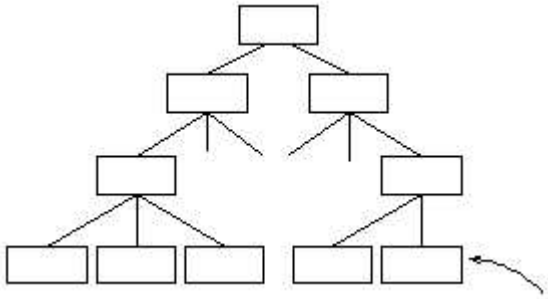


Fig. 5. New entries are always inserted in the right-most node of each 1D R-Tree

However, considering the 1D R-Trees of the FNR-Tree, the 1-dimensional time intervals are inserted in the tree in increasing order for the reason that time is monotonic. This fact leads us to the following modification of the insert algorithm of the 1D R-Tree as shown in Figure 5. Every new entry is simply inserted in the most recent (right-most) leaf of the 1D R-Tree. In case this leaf node is full, a new node is created and the entry is inserted in it. The new leaf node is inserted in the father node of the last leaf. Then, changes are propagated upwards using the conventional AdjustTree algorithm of the R-Tree. In general, this is the same technique that was used by Pfoser et al. [10] in the insertion algorithm of the TB-Tree.

This insertion technique results to 1D R-Trees with full leaves and very low overlapping. The space utilization of the FNR-Tree using the conventional insertion technique in the 1D R-Trees was 65% while, after the illustrated modification, it reached 96%.

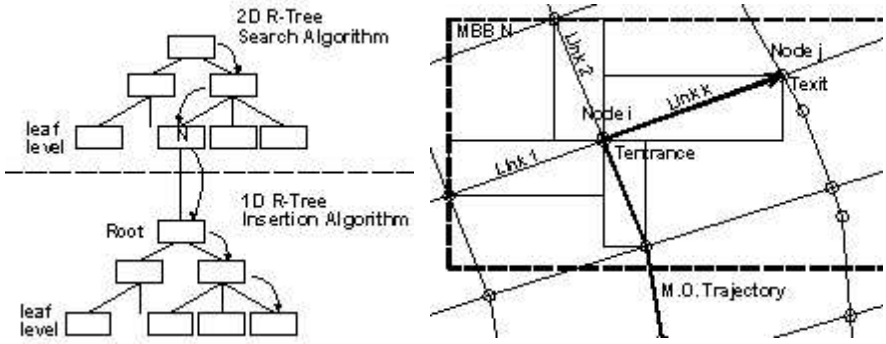


Fig. 6. Insertion of a new entry in the FNR-Tree

As shown in figure 6, the insertion algorithm is executed when the moving object reaches a node (*Node j*) of the network. The first step requires a spatial search among the 2D R-tree (with arguments, the coordinates of *Node i* and *Node j*) in order to find the line segment *Link k* that is contained in the *MBB N* of the 2D R-tree *leaf node N*. Next, we follow the pointer to the 1D R-tree, in which is inserted the *one dimensional interval* (T_{entry}, T_{exit}), together with the *Moving Object Id*, the *Link Id* and the *Direction*

flag. The newly inserted entry is always placed in the right-most 1D R-tree leaf node. If it is required, changes in the MBB of the leaf node (or a newly created leaf node), are propagated upwards until the 1D R-tree *Root*.

The FNR-Tree search algorithm with a spatiotemporal window (3D interval) as an argument is illustrated in figure 7.

FNR-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 Guttman's search algorithm in each one of the 1D R-Trees that correspond to the leaf nodes of the 1st step.
3. For each entry in every leaf of the 1D R-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 spatiotemporal query.

Fig. 7. FNR-Tree Search Algorithm

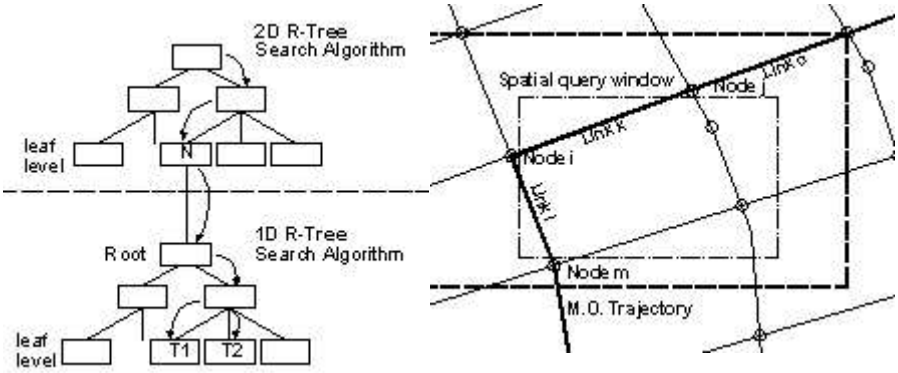


Fig. 8. Searching in the FNR-Tree

Suppose we would search the FNR-Tree with a spatiotemporal query window (X_1, Y_1, X_2, Y_2, T_1, T_2) (Figure 8). The first step requires a spatial search (with arguments the 2D interval (X_1, Y_1, X_2, Y_2)) among the 2D R-Tree in order to locate the line segments and the corresponding 2D R-Tree leaves (in particular, *leaf node N* with *MBB N*) which cross the spatial query window. Next, a search (with arguments the 1D interval (T_1, T_2)) is executed in each one of the 1D R-Trees that correspond to the leaf nodes of the 1st step. In our example, the search led us to the 1D R-Tree leaf nodes *T1* and *T2* that (among others) contain *link k*, *link l* and *link o*. In the final step we retrieve from the main memory the coordinates of each link selected in the 2nd step and we reject those, which are fully outside the spatial query window (in particular, *link o*).

4 Performance Study

In order to determine in which conditions, the FNR-Tree is efficient, we conducted a set of experiments using a Visual Basic implementation that simulates the behavior of the FNR-Tree. We have chosen the page size for the leaf and non-leaf nodes to be 1024 bytes, which lead to a fanout of 50 for the leaf and non-leaf nodes of the 2D R-Tree. The fanout for the 1D R-Trees inside the FNR-Tree was 73 for the leaf, and 85 for the non-leaf nodes.

So as to have a direct comparison with another spatiotemporal access method, we have chosen the conventional 3D R-Tree [16]. It was also implemented in Visual Basic and (in order to provide an as fair as possible performance comparison) was modified as described in the [10]: a MBB can contain a given 3d line segment in 4 different ways only (figure 9). Taking, therefore, into consideration that the 3D R-Tree can contain only simple line segments, we can replace the pointer to the actual object with a flag that takes values from the domain {1,2,3,4}. Consequently, the entry format of the leaf nodes of the 3D R-Tree is of the form *(Id, MBB, Orientation)*. The fanout for the 3D R-Tree was 36 for both leaf and non-leaf nodes.

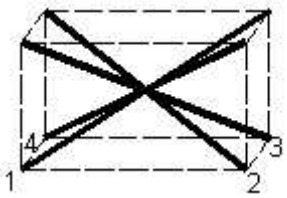


Fig. 9. Alternative ways that a 3D line segment can be contained inside a MBB

The reason that the 3D R-Tree was chosen instead of another spatiotemporal access method (e.g. TB-Tree) is explained below: The FNR-Tree is designed to efficiently answer spatiotemporal queries of the form "*find all objects within a given area during a given time interval*" [15] (also called spatiotemporal range queries), or the form "*determine the positions of moving objects at a given point of time in the past*" [16] (also called timeslice queries). According to the performance study for the TB-tree and the STR-Tree [10], the R-Tree with the modifications illustrated in the previous paragraph, has better performance for a number of moving objects and over. This number of moving objects (break-even point) depends on the size of the window query: for a 10% range query it is 200 moving objects, whereas for a 20% range query, it is over 1000 moving objects [10]. Therefore, in order to rate the performance of the FNR-Tree with a large set of moving objects, we had to compare it with the R-Tree.

4.1 Datasets

While there are several real spatial datasets for the conduction of experiments, real moving-object datasets are not widespread. Owing to this lack of real spatiotemporal data, our performance study was based upon synthetic datasets created using a network-based data generator [1] and the real-world road networks of Oldenbourg and

San Jose. For each of the two networks we produced trajectory datasets of 200, 400, 800, 1200, 1600 and 2000 moving objects, where each object's position was sampled 400 times. While the output of the generator was of the form {id, t, x, y}, the FNR-Tree can utilize those data only if (x, y) are the coordinates 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. The maximum volume of line segments inserted in the FNR and the 3D R-Tree was approximately 1000K and came up for the 2000 objects of San Jose.

4.2 Results on Tree Size and Insertion Cost

The size of the created index structures is shown in the Table 1. As shown, the size of the FNR-Tree is in any case much smaller than the 3D R-Tree. The ratio between the index size of FNR and 3D R-Tree varies between 0.30 and 0.45 for large number of moving objects. For example, the size of the FNR-Tree for 2.000 moving objects in the network of Oldenbourg is about 11 MB, while the size of the respective 3D R-Tree is 38 MB.

Table 1. Index Size and Space Utilization

	Oldenburg		San Jose	
	FNR-Tree	3D R-Tree	FNR-Tree	3D R-Tree
Index Size	~6 KB per object	~19 KB per object	~9 KB per object	~23 KB per object
Space Utilization	96%	65%	92%	65%

The results for the space utilization are similar. As shown in table 1, the space utilization of the 3D R-Tree is about 65% and remains steady regardless of the volume of moving objects. On the contrary, the space utilization of the FNR-Tree grows, as the number of moving objects is growing. Thus, the space utilization of the FNR-Tree with 200 moving objects on the network of Oldenburg is 80%, while, for 1200 objects and over, and for the same network, reaches the 96%. Similar – but not identical - are the results for the network of San Jose: the space utilization for a small number of moving objects (200) is 66%, while for 1200 and over it reaches 92%.

The results on nodes accesses per insertion are illustrated in Table 2. Each insertion of a 3D line segment in the FNR-Tree of Oldenbourg and San Jose requires 8 and 12 node accesses respectively, while an insertion in the 3D R-Tree requires an average of 4.5 node accesses for both networks. This demerit of FNR-Tree can be explained if we have in mind its insertion algorithm: The algorithm requires a pre-searching of the top 2D R-Tree in order to find the 1D R-Tree in which to place the newly inserted line segment. Consequently, for each insertion there is a (stable) number of node accesses which correspond to the spatial search of the 2D R-Tree. This also explains the significant difference between Oldenbourg's and San Jose's node accesses per insertion: San Jose's road network is much larger than Oldenbourg's (~24000 vs. ~7000 line segments), consequently, the spatial search in the corresponding 2D R-Tree requires more node accesses.

Table 2. Node accesses per insertion

	Oldenburg		San Jose	
	FNR-Tree	3D R-Tree	FNR-Tree	3D R-Tree
Node Accesses per insertion	~8	~4.5	~12	~4.5

4.3 Results on Search Cost

Range queries are very important for spatiotemporal data. They are usually of the form "*find all objects within a given area during a given time interval*" and can be expressed by a 3D query window. Range queries can be divided in two subcategories: those having the same size in each dimension (spatial and temporal), and those with different size in the temporal and the spatial dimensions. Of special interest are queries belonging to the second subcategory having the temporal dimension set to 100% as such a query can answer questions of the form "*find all moving objects having ever passed through given area*", or, "*When the moving object x passed through a given area*". Timeslice queries are of the form "*Determine the positions of moving objects at a given point of time in the past*" and they are also subclass of the range queries having the temporal dimension of the 3D window equal to 0.

Range and timeslice queries were used in order to evaluate the performance of the FNR-Tree. Both were executed against the FNR-Tree and the 3D R-Tree having the same data, at the same time, so as to direct compare the performance of those access methods. In particular, we used sets of 500 queries with the following query windows:

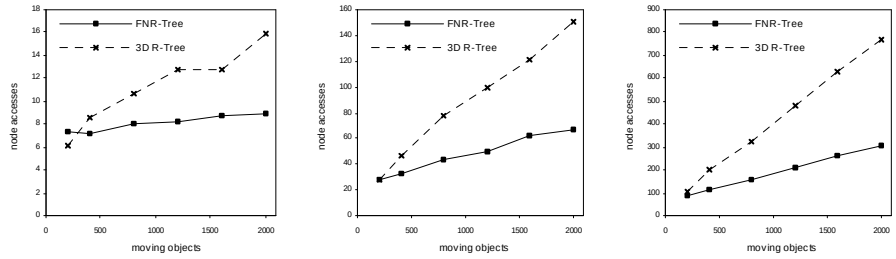
- 3 query windows with equal size in all three dimensions and a range of 1%, 10% and 20% in each dimension, resulting on 0.0001%, 0.1% and 0.8% of the total space respectively.
- 2 query windows with a range of 1% in each spatial dimension and 10% and 100% in the temporal dimension and, 1 with a range of 10% in each spatial dimension and 100% in the temporal dimension, resulting on 0.001%, 0.01% and 1% of the total space respectively.

We also used sets of 500 timeslice queries, using spatial window with range 1%, 10%, and 100% in each dimension.

Range Queries with Equal Spatial and Temporal Extent. Figures 10 and 11 show the average number of node accesses per query for various ranges and datasets. In particular, figure 10 shows the average number of node accesses for range queries with a window of 1%, 10% and 20% in each dimension for objects moving on the network of Oldenbourg, while figure 11 shows the same for objects moving on the network of San Jose. As it is clearly illustrated, the FNR-Tree has superior range query performance over the 3D R-Tree, for a number of moving objects and above, in all query sizes, on both networks.

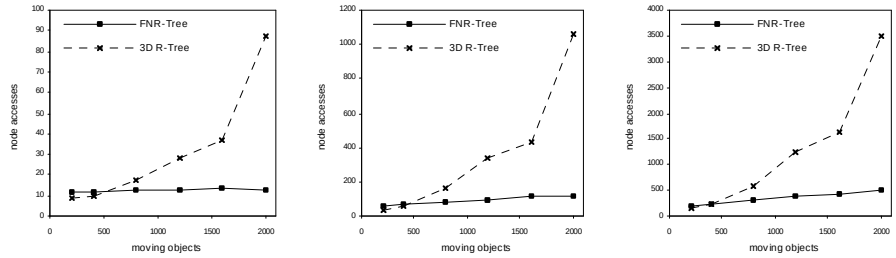
The break-even point at which the FNR-Tree has better range query performance depends on the network and the query size. Specifically, in the network of Oldenbourg for small query size (1% in each dimension), the break-even point is at about

300 moving objects, while for larger query sizes (20% in each dimension), the FNR-Tree shows better range query performance in all data sizes. The corresponding break-even point for the network of San Jose is at about 500 moving objects for 1% and 10% query size, while for 20% query size it decreases to 350. The main reason for the difference in the break-even point is that San Jose's road network is much larger than Oldenbourg's and consequently, the spatial search in the corresponding 2D R-Tree requires more node accesses - which remain stable regardless of the number of moving objects.



(a) (b) (c)

Fig. 10. Range queries for Oldenbourg: (a) 1%, (b) 10% and (c) 20% in each dimension

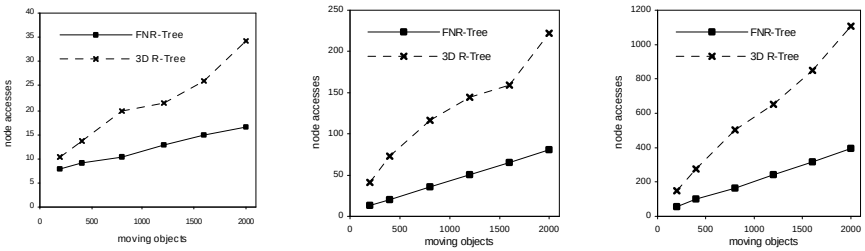


(a) (b) (c)

Fig. 11. Range queries for San Jose: (a) 1%, (b) 10% and (c) 20% in each dimension

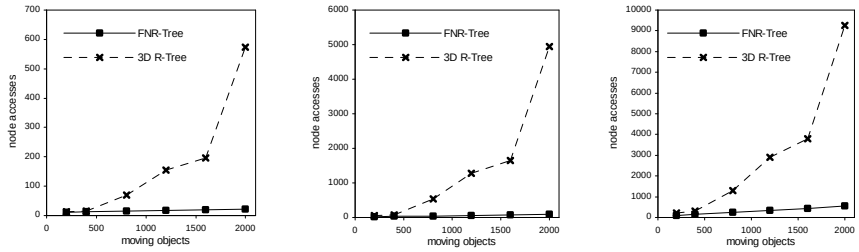
It is worth to note that on both networks, the ratio between node accesses of the FNR and the 3D R-Tree is conversely relative to the query size and the data size: as much as the data size grows, the ratio between the FNR and the 3D R-Tree node accesses gets smaller. In Oldenbourg, for 1% query size the ratio is 1.22 for 200 moving objects and 0.55 for 2000 moving objects. As the query window increases to 10% and 20%, the ratio decreases to 0.44 and 0.40 respectively. The same trend is illustrated in the results for San Jose, where the ratio from 1.31 for 1% query size and 200 moving objects, decreases to 0.15 for 2000 objects and for larger query size (20%).

Range Queries with Larger Temporal Extent. Figures 12 and 13 show the average number of node accesses for several datasets and queries having larger temporal extent. In all datasets and query sizes that we used on both networks, the FNR-Tree shows superior performance over the 3D R-Tree. Furthermore, if we compare the performance of the FNR-Tree in queries having larger temporal extent, with queries of equal spatial and temporal extent, we could see that its performance is much better on the first query type. Specifically, for 2000 moving objects and a query size of 1% in the two spatial dimensions and 10% in the temporal dimension, the ratio between the FNR and the 3D R-Tree node accesses is 0.49 for Oldenbourg's road network. As the temporal extent grows, this ratio gets smaller: for 2.000 moving objects, 1% query size in each spatial dimension and 100% in the temporal dimension, the ratio decreases to 0.36. The same trend is shown in the network of San Jose where the performance of the FNR-Tree is much better in all query and data sizes (Figure 13).



(a) (b) (c)

Fig. 12. Range queries for Oldenbourg: (a) 1% in each spatial dimension, 10% in the temporal dimension, (b) 1% in each spatial dimension, 100% in the temporal dimension and (c) 10% in each spatial dimension, 100% in the temporal dimension



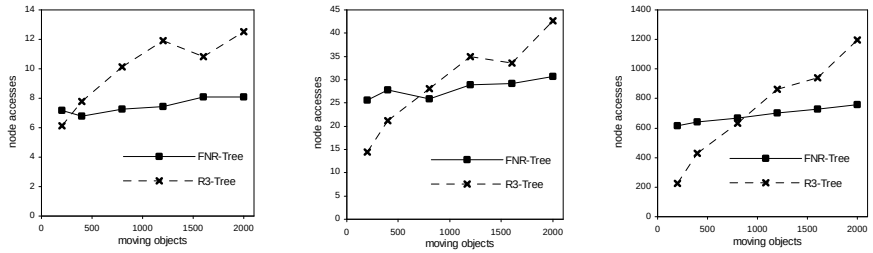
(a) (b) (c)

Fig. 13. Range queries for San Jose: (a) 1% in each spatial dimension, 10% in the temporal dimension, (b) 1% in each spatial dimension, 100% in the temporal dimension and (c) 10% in each spatial dimension, 100% in the temporal dimension

The fact that the FNR-Tree shows better performance in the queries with larger temporal extent can be explained below: When a query is executed against the FNR-Tree, the search algorithm starts with the spatial search in the 2D R-Tree. Then, having located the corresponding 1D R-Trees inside the FNR-Tree, the performance of a searching among them, requires minimum node accesses, because the inserted data

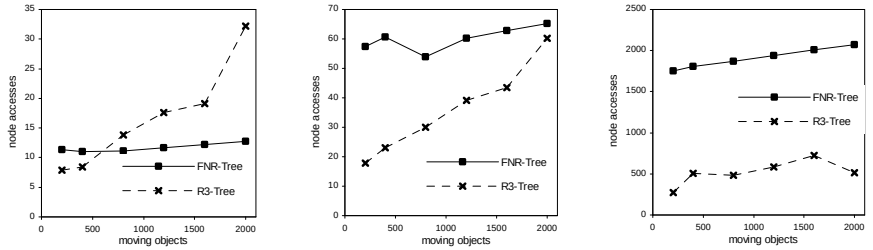
have been already sorted in ascending order (a consequence of the monotonicity of time) and the formed 1D R-Trees have very low overlap and high space utilization.

Time Slice Queries. On the contrary, the performance of the FNR-Tree in time slice queries is not that good. Figures 14 and 15 show the average number of node accesses for time slice queries with several datasets and spatial extents. In the road network of Oldenbourg, the FNR-Tree shows better performance for a number of moving objects and over; the break-even point depends on the query size and is at about 300 for 1% and 900 for 100% query size in each spatial dimension. In the road network of San Jose, the 3D R-Tree shows better performance in most cases, except for less than 600 moving objects and 1% query size in each spatial dimension.



(a) (b) (c)

Fig. 14. Timeslice queries for Oldenbourg: (a) 1%, (b) 10% and (c) 100% in each spatial dimension



(a) (b) (c)

Fig. 15. Timeslice queries for San Jose: (a) 1%, (b) 10% and (c) 100% in each spatial dimension

The main reason for this shortcoming of the FNR-Tree is that searching in a large spatial window leads searching many 1D R-Trees in the forest. Even if the cost of each search is the minimum of 2 node accesses (i.e., a typical height of the 1D R-tree), for a spatial window of 100% the minimum number of node accesses could not be less than *2*(Number of 1D R-Trees) + (Number of 2D R-Tree Nodes) = 2*(Number of 2D R-Tree Leaves) + (Number of 2D R-Tree Nodes).* This formula in the case of Oldenbourg results to *2*212+230=664* node accesses.

4.4 Summary of the Experiments

The experiments that were conducted in order to evaluate the performance of the FNR-Tree showed that it supports range queries much more efficiently than the 3D R-Tree. We found additionally, that in a special case of range query, where the temporal extent of the query is much larger than the spatial extent, the performance of the FNR-Tree increases significantly. On the contrary, in time slice queries the performance depends on the road network: For a relatively small network (Oldenbourg) the FNR-Tree shows better performance from a number of moving objects and over, while for a larger network (San Jose), the 3D R-Tree outperforms the FNR-Tree in most cases. Regarding the size of the FNR-Tree, its space utilization – depending on the network and the number of moving objects – can reach the 96%, and the average size per moving object can be 2 to 3 times less than the respective size of a 3D R-Tree. However, the average node accesses per insertion in the FNR-Tree can be 2 to 3 times more than the respective accesses for an insertion in the 3D R-Tree.

5 Conclusion and Future Work

Although the vast majority of spatiotemporal access methods (3D R-Tree [16], HR-Tree [6, 7], TB-Tree, STR-Tree [10], TPR-Tree [13], Mv3R-Tree [14], PPR-Tree [3]) index objects moving in the whole of the two dimensional space, in a great number of real-world applications the objects are moving in a constrained space (e.g. fleet management systems). These moving restrictions have recently been a subject of research [5, 8, 9, 11]. Though the development of a spatiotemporal access method suitable for objects moving on fixed networks seems challenging because of the great number of real-world applications, until recently there was no proposal.

For this purpose, we present a new indexing technique, called Fixed Network R-Tree (FNR-Tree); an extension of the R-Tree [2] designed to index moving objects constrained to move on fixed networks. The general idea that describes the FNR-Tree is a forest of several 1-dimensional (1D) R-Trees on top of a single 2-dimensional (2D) R-Tree. The 2D R-Tree is used to index the spatial data of the network (e.g. roads consisting of line segments), while the 1D R-Trees are used to index the time interval of each object's movement inside given link of the network.

In the conducted performance study, comparing this novel access method with the traditional 3D R-Tree [16] under various datasets and queries, the FNR-Tree was shown to outperform the R-Tree in most cases. The FNR-Tree has higher space utilization, smaller size per moving object and supports range queries much more efficiently. In general, we argue that the FNR-Tree is an access method ideal for fleet management applications.

The FNR-Tree showed a weakness in time-slice queries and, in general, is inefficient to retrieve object trajectories that intersect with a time interval without any spatial extend, because this would result to a search among all the 1D R-Trees contained in the FNR-Tree. Another shortcoming is the inability of the FNR-Tree to answer topological (trajectory-based) queries [10]. This kind of queries requires the preservation of each moving object's trajectory. Therefore, future work has to include the in-

tegration of the trajectory preservation as well as the improvement of the access method in order to efficient answer queries without spatial extent.

Acknowledgements

Part of this work was in a MSc. dissertation under the supervision of Professor Timos Sellis, for the MSc. program on GEOINFORMATICS at the National Technical University of Athens (NTUA). The author would like to thank his supervisor Timos Sellis, Yannis Theodoridis for discussions on the subject, and Thomas Brinkhoff for providing his network-based data generator and support. The author is a scholar under the Greek State Scholarship's Foundation.

References

1. Brinkhoff, T.: Generating Network-Based Moving Objects. In *Proceedings of the 12th Int'l Conference on Scientific and Statistical Database Management*, SSDBM'00, Berlin, Germany, 2000.
2. Guttman, A.: R-Trees: a dynamic index structure for spatial searching, *In Proceedings of the 13th Association for Computing Machinery SIGMOD Conference*, 1984, 47-57.
3. Hadjieleftheriou, M., Kollios, G., Tsotras, V.J., Gunopulos, D.: Efficient Indexing of Spatiotemporal Objects. *In Proceedings of 8th International Conference on Extending Database Technology (EDBT)*, Prague, Czech Republic, 2002.
4. Kamel, I., and Faloutsos, C.: On Packing R-trees. In *Proceedings of the 2nd Conference on Information and Knowledge Management*, 490-499, 1993.
5. Kollios, G., Gunopulos, D., and Tsotras, V.: On Indexing Mobile Objects. *In Proceedings of the 18th ACM Symposium on Principles of Database Systems*, Philadelphia, PA, USA, 261-272, 1999.
6. Nascimento, M.A., and Silva, J.R.O.: Towards historical R-trees. *In Proceedings of the 13th ACM Symposium on Applied Computing (ACM-SAC'98)*, 1998.
7. Nascimento, M.A., Silva, J.R.O., and Theodoridis, Y.: Evaluation for access structures for discretely moving points. *In Proceedings of the International Workshop on Spatio-Temporal Database Management (STDBM'99)*, Edinburgh, Scotland, 171-188, 1999.
8. Pfoser, D: Indexing the Trajectories of Moving Objects. *IEEE Data Engineering Bulletin*, 25(2):2-9, 2002.
9. Pfoser, D., and Jensen, C.S.: Querying the Trajectories of On-Line Mobile Objects. TIMECENTER Technical Report, TR-57, 2001.
10. Pfoser D., Jensen C.S., and Theodoridis, Y.: Novel Approaches to the Indexing of Moving Object Trajectories. *In Proceedings of the 26th International Conference on Very Large Databases*, Cairo, Egypt, 2000.
11. Papadias, D., Tao, Y., Kalnis., P., and Zhang, J.: Indexing Spatio-Temporal Data Warehouses. *In Proceedings of IEEE International Conference on Data Engineering (ICDE)*, 2002.
12. Saltenis, S., and Jensen, C.S.: Indexing of Moving Objects for Location-Based Services. TIMECENTER Technical Report, TR-63, 2001.
13. Saltenis, S., Jensen, C.S., Leutenegger, S.T., and Lopez, M.A.: Indexing the Positions of Continuously Moving Objects. TIMECENTER Technical Report, TR-44, 1999.

14. Tao, Y., and Papadias, D.: Mv3R-tree: a spatiotemporal access method for timestamp and interval queries. *In Proceedings of the 27th International Conference on Very Large Databases*, 2001.
15. Theodoridis, Y., Sellis, T., Papadopoulos, A.N., Manolopoulos, Y.: Specifications for Efficient Indexing in Spatiotemporal Databases. *In Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, Capri, Italy, 1998.
16. Theodoridis, Y., Vazirgiannis, M., and Sellis, T.: Spatio-temporal Indexing for Large Multimedia Applications. *In Proceedings of the 3rd IEEE Conference on Multimedia Computing and Systems*, Hiroshima, Japan, 1996.