Kalay U, Kalipsiz O. A comparison study of moving object index structures. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 24(6): 1098–1108 Nov. 2009

A Comparison Study of Moving Object Index Structures

Utku Kalay and Oya Kalipsiz

Computer Engineering Department, Yildiz Technical University, Besiktas, Istanbul, 34349, Turkey

E-mail: utku@ce.yildiz.edu.tr; kalipsiz@yildiz.edu.tr

Received September 10, 2008; revised August 17, 2009.

Abstract The task of selecting the most appropriate method for indexing the data according to application requires a careful comparison study of indices of interests. In particular, we consider object movements by tracing their trajectories within a predefined road network. MV3DR-tree and 3DR-tree constitute our first group indexing the objects moving in free movement scenarios. Besides, Mapping and MON-tree are the second group indexing the locations of objects moving over a network of road. Those access methods mainly organize a group of R-tree in order to index the underlying road network and the object movements. Our goal in this study is to evaluate existing proposals under fair circumstances with respect to storage consumption and spatio-temporal query execution performance. In our comparisons, we discuss the structure's sensibility to query's spatial and/or temporal extent as well as the tradeoff arising between two groups in terms of reliability and disk access performance. We believe that revealing the vulnerabilities of the selected structures, especially Mapping and MON-tree motivates us to design more robust organizations.

Keywords access methods, query processing, spatial databases, temporal databases

1 Introduction

Rapid innovations of wireless technologies introduced dynamic information tracking into many applications and research disciplines including database systems as well. Among those applications, tracking moving objects by storing the location information within a specialized database, namely moving object database (MOD), deserves special attention due to its high demand in real world applications. Modeling the movements and imprecise location information and efficiently querying either past or present and future positions of moving objects are only a few examples of the main interests of a MOD. It is obvious that diverse range of data and query types and new complex models make conventional aspects of MOD more sophisticated.

Indexing, for instance, is an important research topic in the area of moving object database. An index storing the locations of moving objects can be extended on the existing spatial indexing methods with some modifications or additional data structures to accommodate the specific dynamic requirements and/or new query types. For example, there are some robust organizations in the literature that use R-tree[1], a well-known spatial index structure, either as a skeleton or as a building block, with some in-node reorganizations.

In general, the problem of indexing moving object locations is grouped into two classes: one tackles indexing the past trajectories of objects and the other tackles the current and near-future positions of the objects. Significant organizations focusing on querying past trajectories can be listed as 3DR-tree[2], STR/TB-tree[3], HR-tree[4], MV3DR-tree[5], Mapping[6], FNR-tree[7], and MON-tree[8]. For querying current and near-future positions, [9–11] can be cited as sample milestones in this branch. The other important aspect of index design is the query type, since efficient execution of the query directly depends on the index main architecture. As an example, spatio-temporal range querying (timestamp or interval), kNN querying, topological querying, navigational querying and continuous querying are some of the query types that play important roles in the evaluation of the index structure's quality.

This paper discusses an overall comparison of some index structures proposed as a solution for efficiently storing and querying past trajectories of objects having moved over a predefined road network. Four of the index structures stated in the previous paragraph are selected for comparison: 3DR-tree, MV3DR-tree, Mapping and MON-tree. The first two are generic solutions to indexing objects moving without any constraint (i.e., free-movement). 3DR-tree is a quite naive solution, whereas MV3DR-tree is an elaborate organization that handles various types of range queries efficiently. The remaining two organizations are specifically designed for the motions over road-networks. First, we named the technique proposed in [6] as "Mapping"

Regular Paper

in this paper. Mapping is based on simple mappings of network, trajectories and spatio-temporal queries to lower-dimensional space. Lastly, MON-tree in [8] is a well-organized 2-level structure, so that the first level indexes the edges having traffic on, while the R-trees at the below level indexes the motions along each edge with 2D rectangles.

We eliminate the other index structures in the list, because they were evaluated in other studies in the literature. For example, FNR-tree is proved to be inefficient in terms of many aspects against MON-tree structure[8]. Additionally, we eliminate STR/TB-tree because it is designed specifically for navigational and other types of trajectory queries. Lastly, since MV3DR-tree emerges as superior version of HR-tree, we eliminate HR-tree in our comparisons as well. We think that the comparisons between the remaining items in the list would be valuable to discuss and put forward some future work. Because, for example, we are not aware of any comparison between MV3DR-tree and MON-tree and Mapping, which all are proved to be well-organized and robust organizations of their own in the literature.

Following this, the comparisons are based on the space utilization of index organizations as well as spatio-temporal range query execution performance in terms of the number of disk accesses. With these comparisons, our aim is to designate the pros and cons of index structures that are of interest and to point out common problems in this area. Eventually, we argue that the structures specifically designed for network-based movements do not always outperform the other generic solutions. The arguments are mostly based on the experimental study which evaluates performance of index structures under fair conditions, as far as possible. More specifically, all index implementations and query executions are carried out within SaIL (Spatial Index Library), which is introduced as a common application programming interface in [12].

The rest of the paper is organized as follows. Section 2 briefly discusses the main issues about indexing movements along the road network edges. In addition to basic approaches to modeling the movement and designing index structures, this section states the importance of network connectivity in relation to the quality of the index architecture. Section 3 explains the main aspects of selected index organizations that are compared. Section 4 describes the simulation environment and query execution tests comparing the quality of the selected index structures. Section 5 concludes the paper and presents ideas for future research work.

2 Indexing Movements over a Road Network

One key challenge is to provide the database with an ability to track the dynamics of the system. When we are interested in the past location records of an object, the term, "trajectory", is used to refer to the trace of its past locations. Since it is impractical to develop the complete trajectory precisely from a limited number of updates, we may attempt to predict the locations between update points. Although this is the way of access methods designed for near-future queries to predict the future locations of objects, it is still a necessity to derive approximate location information between consecutive update points of the past trajectories of objects.

Consider a trajectory of a car moving in the road-network of a city as a sequence of 3 dimensional reported data points, which is a combination of 2 spatial and 1 temporal coordinates. Being the case, the location information between sample points can be derived either with simply using linear interpolation or with applying polynomial *splines*. To derive such data, it is clear that the object should report at least its velocity and direction or the id of the road partition (i.e., edge) on which the motion occurs. The updated information actually depends on the objects' update policy as well as the movement scenario. As introduced in [6], the movement scenarios can be grouped into three classes: free-movement, constraint movement and network-constraint movement. For example, Fig.1 shows the trajectory of an object moving over a part of a road network. It starts its motion at t_1 on edge $S1$ and reports its location at t_2 when it is on $S3$ and eventually ends its travel at t_3 on edge $S4$.

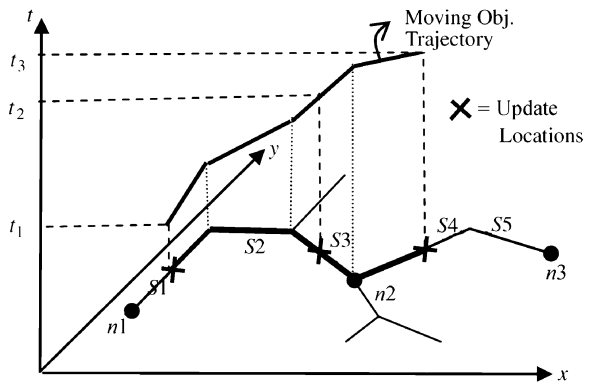


Fig.1. Trajectory of an object moving on a part of a road network and its update points at t_1, t_2 and t_3.

Interpolation seems to be a relatively easy and useful solution; however, it is obvious that the quality of derived information is traded off as the number of location updates decreases. On the contrary, it is beneficial to exploit information from movement scenario, if it is possible, in order to alleviate this quality degradation. This is especially true for constraint

movement and network-constraint movement scenarios. Free-movement scenario, in contrast, does not provide any additional information about the movement. The importance of exploiting road network information can be stated with a brief example as the following. Fig.2 illustrates the reflection of the trajectory in Fig.1 along spatial coordinates only.

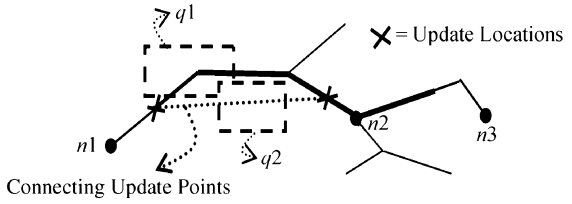


Fig.2. Connecting reported locations of a moving object with a single line and spatial parts of sample queries, $q1$ and $q2$.

Fig.2 additionally contains spatial parts of two spatio-temporal range queries. It is easily seen from Fig.2 that having connected two consecutive update points and applied linear interpolation, it is possible to have false hits in the result set of $q2$ and misses in the result set of $q1$. Rather, utilizing network connectivity and then linear interpolation along the connecting path dramatically increases the quality of the result set.

The example above also clarifies the need of specialized index organizations for network-based movements. As a consequence, the implied aspect of utilizing the network is to store the network location of object instead of absolute location (in the Euclidean space) in the index nodes (pages), since sole Euclidean location does not convey any information about the network. Network location can be thought of as the position of an object within the network partition, i.e., segment, edge or predetermined route. More specifically, the position can simply be the distance from the beginning of the partition. Therefore, network location information has two parts: one is the network partition and the other is the relative distance in this partition. For example, the first update point in Fig.1 can be represented as ($S1S3$, 0.3), which means that $S1S3$ is an edge and the object is on the position that is 0.3 of the whole length away from the beginning of edge $S1S3$. Hence, the proposed index is supposed to store this network location information somehow. Mapping and MON-tree both utilize the network connectivity in similar fashion which will be discussed in the following section.

One drawback of utilizing network in index organizations is the additional R-tree (or other auxiliary organization) indexing the network partitions. This is necessary for both spatio-temporal querying and movement insertions into the organizations. Because, both query execution and movement insertion, in the first place, have to access the relevant network partition. As a consequence, the additional network index naturally increases the storage consumption and the overall execution time performance of the index. The network index storage overhead can be alleviated by using more suitable R-tree models (such as Hilbert R-tree[13]) or excluding the network partitions on which any traffic has not occurred yet. Since network index keeps only the network partitions' id and the surrounding MBR within index leaf nodes, we should somehow store the details of network partitions, such as coordinates and even connectivity. It is worth noting that storing connectivity may be more important for other query types such as kNN and requires more complex and carefully designed organizations. Moreover, storing the network partitions on disk is a preferable method in terms of scalability as well. Otherwise, the index solutions would be restricted to only movements within small networks, which can fit into the allocated main memory.

To sum up, utilizing network seems a necessity for reliability of query result set and other query types such as kNN, while it brings out crucial problems such as complexity and storage overhead. In this study, we implemented the network-based index structures with the integration of network partitions stored in the secondary storage.

3 Main Aspects of Selected Organizations

This section covers the basic functionalities of the selected index organizations, namely 3DR-tree[2], MV3DR-tree[5], Mapping[6] and MON-tree[8]. More elaborate discussions can be accessed from the references.

3.1 3DR-Tree

3DR-tree, a hierarchical and height-balanced index, is the simplest way to index moving objects, since time axis is added to spatial axes as if it is the third spatial axis. Certainly, this naive approach causes some important problems. First, 3DR-tree severely experiences dead-space problem, because a minimum bounding box (MBB) has larger dead-space than a minimum bounding region (MBR) with the same spatial extensions. The new update point determines the extent of each dimension of the new MBB entry. As a consequence, as time passes, dead-spaces, particularly long-term objects, cover large part of space and eventually lead to high overlap and small discrimination capability of overall structure. At worst, for most queries, all leaf nodes may be visited, which makes the index structure useless.

A solution is to minimize the dead-space by shortening the time intervals between consecutive updates. To

shorten this interval, one naive attempt is to decrease the maximum update interval of moving objects, while a more autonomous attempt is to cut a large MBB into a number of small MBBs and then insert them, instead of one large MBB, into 3DR-tree. In the latter solution[14], determining the number of cutting points and their places along the time interval may require complex and time-consuming algorithms with greedy characteristics. It is obvious that both attempts would increase the number of insertions and eventually lead to higher storage consumption. For the purpose of achieving the equilibrium, the number of artificial splits can be determined by means of experimental studies.

The second problem of 3DR-tree is as the following. At first glance, it seems good packing all data (without any redundancy) into a single well-organized tree in terms of storage consumption; however, searching the whole single tree for each type of query is not reasonable. For example, the execution of the timestamp queries or short-length time interval queries against the whole tree dramatically degrades the performance. Rather, if the tree were organized according to temporal extent, the query would be directed to the corresponding part of the tree. This is actually the main strategy of HR-tree and MV3DR-tree organizations.

3.2 MV3DR-Tree

MV3DR-tree[5] combines MVR-tree (multi version R-tree) and 3DR-tree that is built on the leaves of MVR-tree. The goal is to answer short-term and timestamp queries by using MVR-tree, as well as to answer long-term queries by using 3DR-tree. MVR-tree consists of multiple "logical" 2DR-trees, each of which manipulates records within a time interval determined by the version splits of root nodes. Version split is the foundation of multi-version infrastructure and is based on the spatial extents of the nodes. Version splits prevent long bounding boxes by splitting them into 2 entries. As a side effect, this introduces data redundancy. Fig.3 illustrates the basic structure of MV3DR-tree.

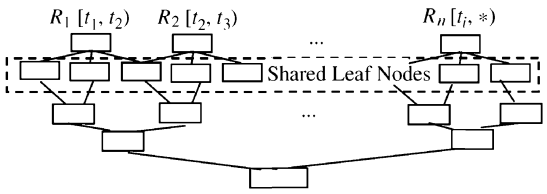


Fig.3. MV3DR-tree: combination of MVR-tree at the upper side and 3DR-tree at the lower side.

Although the size of the 3DR-tree is very small since it shares the leaf nodes of the MVR-tree, the overall storage consumption of MV3DR-tree is still almost 1.5 times larger than that of 3DR-tree storing the same amount of data. The main reason for this increase is the duplicate data from version splits. The other disadvantage of MV3DR-tree is the update cost since it has to modify 2 structures simultaneously during insertions.

3.3 Mapping

Multi-version approach is a solution to alleviating the large dead space problem of 3DR-tree, but is not the only solution. The study in [6] proposes a transformation technique that is able to represent 3D bounding boxes with 2D rectangles. In this paper, we name this organization as "Mapping". Mapping is originally designed for network constraint movement, thus both network and trajectories (and queries) are mapped to lower dimensional data. Fig.4(a) shows a 2D raster network mapped to 1D representation and Fig.4(b) shows trajectories mapped on the new representation of the network.

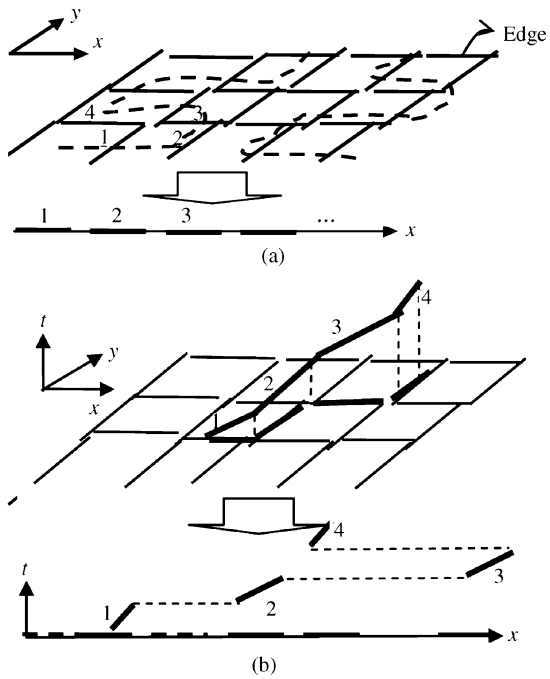


Fig.4. (a) Illustration of network mapping. (b) Illustration of trajectory mapping.

At the first mapping, all network edges are sorted according to the Hilbert value of their center points, the resulting 1D sorted intervals instead of 2D road-network. Secondly, trajectories are mapped to new space based on calculating how much of this edge traversed since the last update (or since the time at beginning of the edge). It is also important to note that we still need to index roads of network with an additional 2DR-tree. Eventually, instead of a single

3DR-tree, after the mappings we have two 2DR-trees; network2DR-tree and trajectory2DR-tree. It is evident that the spatio-temporal query is also mapped to a number of queries that are then processed against the trajectory2DR-tree.

Fig.5 shows a spatio-temporal query, $\boldsymbol{Q}$ and a number of 2D queries that result from the mapping. The size of this query set is actually the number of roads inside the spatial extent of the query. This number is an important parameter influencing the query performance and directly depends on the complexity of the road-network.

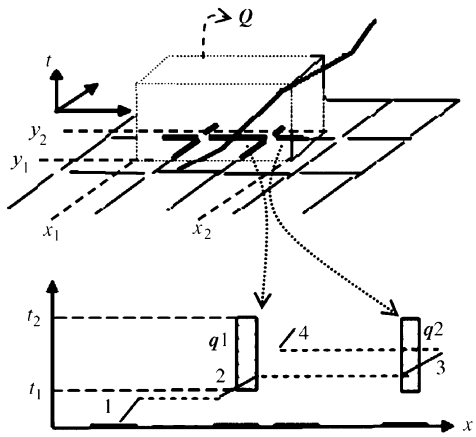


Fig.5. Mapping of spatio-temporal query, $\boldsymbol{Q}(x_1, y_1, x_2, y_2, t_1, t_2)$.

3.4 MON-Tree

MON-tree is a 2-level organization of a group of R-trees. The first level includes an R-tree for indexing network roads that had traffic on it and the second level includes R-trees, each for indexing motions on a single network path (i.e., edge). Modeling movements with 2D rectangles are similar to Mapping technique except that the movements are stored in their original spaces. Fig.6 shows the main structure of MON-tree and insertion of a movement on e_3.

The number of low-level R-trees is equal to the total number of edges at leaves of the top level R-tree. Each R-tree at the lower level stores the movements along the corresponding edge with 2D rectangles. In addition, a hash table in the main memory can be used as an auxiliary structure to directly access the movements on a specific edge. An object movement is represented as a rectangle (p_1, p_2, t_1, t_2), which means that an object motion starts at p_1 part of an edge, e_i, at time t_1, and ends its movement at p_2 part of e_i at time t_2, while $0.0 \leqslant p_1 < 1.0$ and $0.0 < p_2 \leqslant 1.0$ are hold. For example, Fig.6 shows the insertion of a movement starting at 0.3 of the edge at t_1 and ending at 0.6 of the edge at t_2.

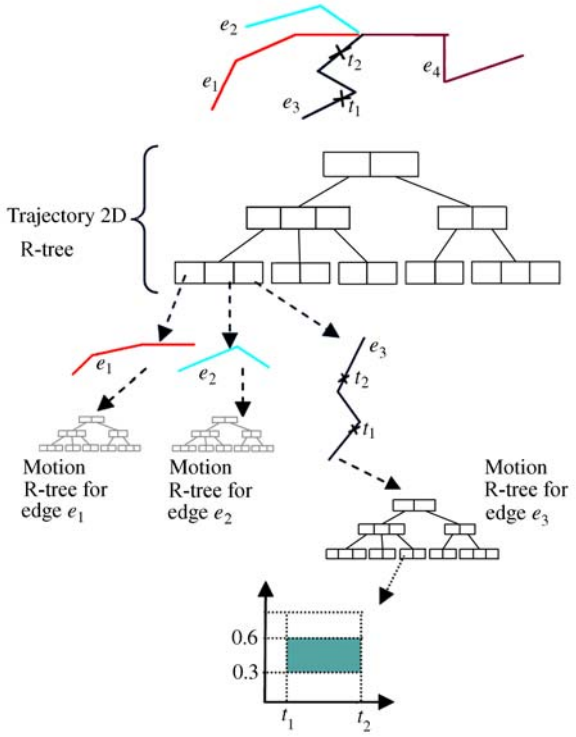


Fig.6. Storing a motion on a small network in the MON-tree architecture.

Now, consider a spatial-temporal range query, $\boldsymbol{Q}(q_{\text{spatial}}, t_1, t_2)$, where q_{spatial} is a spatial range query that is valid for the duration from t_1 to t_2. In the first step, q_{spatial} is the static range query over the top-level R-tree that finds the edges that are covered by the query. In fact, since the edges may partially be within the query's spatial region, the covered parts of the edge (*qlist*) are selected from the main memory with a minimal execution overhead. Secondly, we access the corresponding low-level R-trees to retrieve the motions within the requested time interval. Therefore, the output of the second step is the objects of whose past movement regions are intersecting any query region in *qlist*. That is actually done by other range query executions over the low-level R-trees found in the first step.

4 Experimental Studies

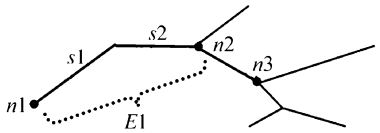
For retrieving the past locations of dependently moving objects, particularly moving over network of city roads, we implement and compare 4 different index methods. As it is previously pointed out, these organizations are selected based on the observation that some has no performance comparison against others. For example, the study in [8] states that Mapping organization was not expected to be better than FNR-tree and MON-tree, without presenting any experimental comparison. Additionally, we think that comparing

MV3DR-tree — the generic solution for animated objects — and the network-specific solutions such as MON-tree would be interesting.

This section includes the basic principles of our model of road networks, traffic generation and update policy of moving objects. A brief description of the library that we used to implement index structures is presented as well.

4.1 Network Model

The network, representing a roadmap, consists of nodes that are connected to each other with a weighted line-segment. The weight is related to the length of line-segment. We define edges as the road section between connected cross-points or open-ended points. The cross-point is the intersecting point of more than two line-segments. Fig.7 depicts a small portion of a network of roads and the definitions of node, line-segment and edge.



$E1$ is an edge between the nodes $n1$ and $n2$.
$E1$ consists of $s1$, $s2$ line-segments.
$n1$ is open-ended, $n2$ is cross-point.

Fig.7. Elements of road network.

In our experiments, we used road networks with different capacity. Table 1 shows the total number of cross-points and total number of edges within the cities of Chicago and North Central area.

Table 1. Capacity of Road Networks

	No. Cross Points	No. Edges
Chicago	943	1412
North Central	2294	3333

The edges, the cross-points and their interconnectivity are stored in random access files, namely "Poly-Line Component" and "Adjacency Component". Poly_line component is accessed during traffic generation and index construction while the adjacency component is accessed only at generating traffic paths with using Dijikstra's shortest path algorithm. The connectivity representation in adjacency component is implemented as proposed in [15].

4.2 Traffic Generation and Update Policy

We modeled our moving object simulation environment as a discrete-time animated movie sequence, so that an object changes its velocity at each time instant while remaining within a velocity range. The velocity range is simply based on the fact that the object moves faster when it is on longer roads. More specifically, once all edges in road-network are grouped into 3 classes based on their lengths and assigned a velocity range to each group (i.e., long roads are assigned high velocities), an object velocity is generated based on the predetermined velocity range for the edge on which the object is currently traveling.

All initial values and following motion characteristics during network traffic generation mostly fit uniform distribution. Each moving object travels along a predefined route within a road network. The routes are the shortest paths between randomly selected nodes (cross-points) in the network. The shortest paths are determined by applying Dijikstra's shortest path algorithm to the network connectivity and polylines stored in hard disk as shown in Fig.8. Therefore, after determining the routes by using complete network information, the location and velocity reports are generated based on a policy presented in the following paragraphs.

Our goal in this study is to load all indices with the same updates generated from the same traffic, such that the numbers of entries inserted to each index are equal. After reviewing [16–17], we decided to implement a common point policy. The speed dead-reckoning update policy suggested for free-movement scenarios in [16] is similar to Distance-Threshold-Triggered Location Update (DTTLU) within the Net_LUM[17]. Therefore such a policy seems suitable for our scenario combining different index structures.

Updates are reported to database (in our case to index structures) at time steps, whenever the distance between the object's real position (usually retrieved by GPS on the vehicle in real life) and the predicted location (calculated by using the last update position and velocity stored in the database) exceeds a predetermined threshold value. This threshold value apparently determines the quality of structure as well as the index loading performance. The experiments include some results about the affect of threshold variation on the index performance.

The last point is the updates at the cross-points (at the end of edges). For Mon-tree's internal workings and the implementation of edge-based network representation, we have to generate this update, even though the object is still inside the allowed threshold area. Thus we do not miss the existence of the object movement on each edge throughout the trajectory. Otherwise we would need to utilize the network node connectivity to fill the gaps throughout the trajectory. This observation is valid for both Mapping and MON-tree structures, because they are organized on top of the edges of the trajectories.

A more elaborate model would partition the network into routes which are the groups of consecutive edges. As pointed out in [8], the route model enables the storage consumption of MON-tree to decrease because of the elimination of the end-of-edge updates throughout the trajectories. However, the query execution performance would degrade with the larger MBRs of routes because of its interference with the selectivity of the structure. We defer a similar performance study on such a route model. Defining the routes, revising MON-tree and network-storage implementation accordingly may be the main aspects of the future work. Lastly, we exclude the uncertainty issues discussed in [16–17] and data errors in [17].

4.3 Spatial Index Library: SaIL

We utilized Spatial Index Library (SaIL) in JAVA for our implementations and tests. The library is available at [12]. As stated previously, we stored the polyline representations — edges of the road network — and network connectivity within random access files. The polylines_file are referred to by the Mapping and Mon-tree index structures.

Fig.8 depicts the basic modules of our experimental study. Both access methods and real hard disk are interfaced with appropriate interfaces. Taking scalability into consideration, the architecture enables new access methods and/or buffer managements to be integrated with the existing system.

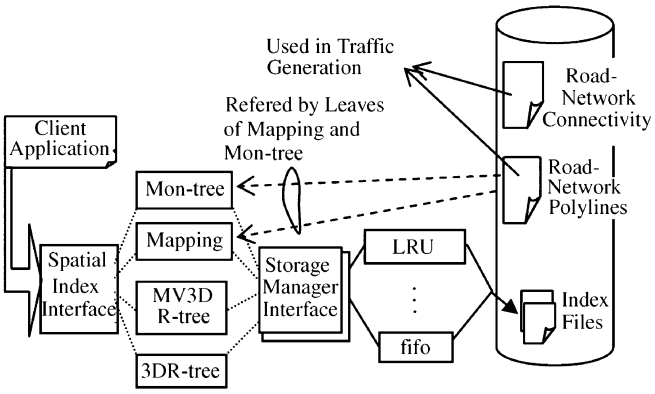


Fig.8. Connection of network representation and main modules of extended SaIL. (Traffic generation uses both the network connectivity and polylines.)

4.4 Issues in Index Design

As opposed to statement in [8], Mapping organization does not suffer in terms of network model, although all the edges are needed to be inserted into network R-tree. The reason is that since the road-network is stable (which is a reasonable assumption), it is easy in practice to decrease the height of network R-tree with bulk-loading organizations, such as Hilbert R-tree[13]. On the other hand, network R-tree section of MON-tree cannot be loaded beforehand because it totally depends on the traffic distribution characteristics.

System characteristics such as page size and buffer size and buffer type are the same for all the indices and kept stable in all the experiments. In our experiments, both at data loading and querying, disk page size in all the components including poly-line component is set to 4 KB. The replacement policy randomly selects the victim page in the buffer. Furthermore, the branching factor of each structure which is the maximum possible number of descendents of its nodes differs, since the internal node structure differs from each other. More specifically, the branching factor for each index is set to its maximum value so that the page overflow is avoided with a minimum internal fragmentation at the same time. After a couple of experiments the branching factor has been determined as shown in Table 2. (Remember the network R-tree section of Mapping is Hilbert R-tree and the branching factor is out of consideration.)

Table 2. Branching Factors of Each Structure

3DR-tree	60
MV3DR-tree	72 (for MVR-tree)
	60 (for 3DR-tree)
Mapping	102 (for trajectory 2DR-tree)
Mon-tree	85 (for network R-tree)
	102 (for motion R-trees)

4.5 Index Loading

The main operation of index loading is to insert information generated from two consecutive update points into 4 different indices. Each update is represented as path in Mapping and MON-tree organizations and as a single line in 3DR-tree and lastly as just 2 update points in MV3DR-tree. Fig.9 illustrates a sample path, $P1$ in edge $E1$ and how it is stored in Mapping and MON-tree. (Insertions to others are omitted because it is straightforward.)

As shown in Fig.9, Mapping and MON-tree preserve the path between update points, by means of accessing additional component, namely Poly-Line Component, in which the detailed poly-line representation of each edge in the network is stored. On the contrary, the location information in 3DR-tree and MV3DR-tree stores absolute locations in 2D coordinate system.

Fig.10 shows that the overall behavior of structures in terms of storage consumption exponentially increases with larger data set size (DS) values. The slope is

not dramatic and changes moderately with the type of structure.

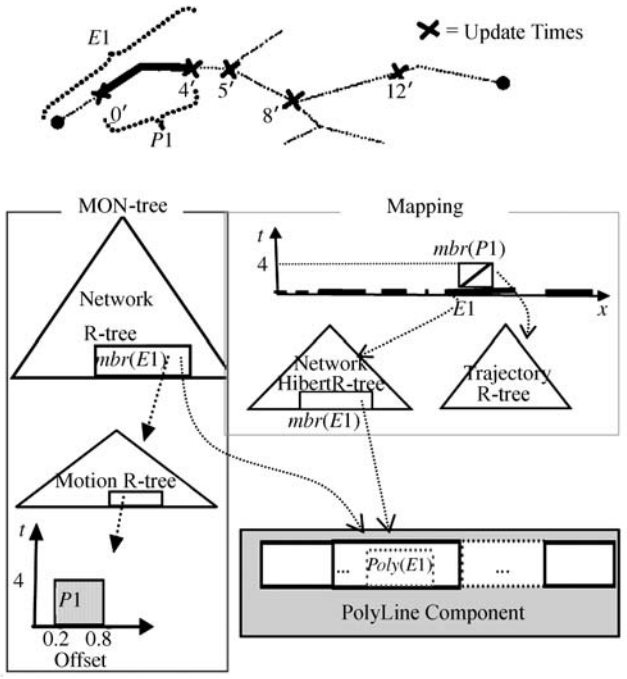


Fig.9. Overview of loading MON-tree and Mapping indices.

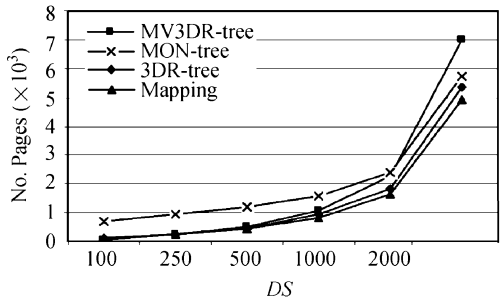


Fig.10. Overall index size with varied data set size (DS) values. (Note that Polyline Component's pages are excluded.)

First observation is about the MON-tree. It is seen from Fig.10, in case of scarce traffic characteristics (with small DS), MON-tree storage consumption is considerably high in relation to others, since scarce traffic causes most motion pages to remain nearly empty throughout the entire simulation. More specifically, consider a traffic scenario having edges with just a few updates on them. In such cases, the motion R-trees corresponding to those edges have only nearly empty roots, which lowers the storage utilization under 50%. The unnecessary allocation of a whole R-tree for a few updates gives its place to moderate storage consumption as increasing DS fills the motion R-trees.

The undesirable behavior mentioned above does not occur in Mapping, since motions in trajectory 2DR-tree are not scattered over R-trees, rather are accumulated into a single 2DR-tree. In addition to that, utilizing Hilbert R-tree for network storage makes the Mapping best in terms of storage consumption.

To get a sense of how index storage consumptions were really influenced by traffic load from a different perspective, we attempted to show the affect of the update threshold during traffic generation. Since the decreasing threshold value results in a loaded traffic for the same data set, it seems interesting to show the index's behaviors on this parameter.

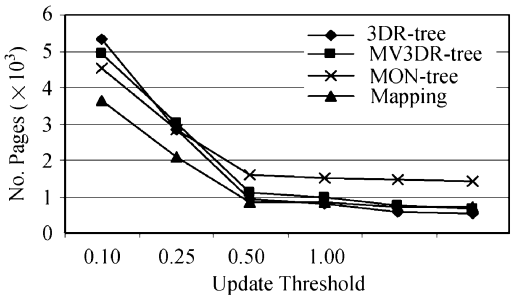


Fig.11. Overall index sizes with varied update threshold values on the same traffic data set, $DS = 1000$. (Note that Polyline Component's pages are excluded.)

Fig.11 shows that 3DR-tree and MV3DR-tree are the most sensible to the update threshold value because the traffic load increase arises from the updates which are exclusive of the *end_of_edge* updates. Another point in Fig.11 is MON-tree's weakness for scarce traffic as observed in the previous experiment. Since high update threshold values cause scarce traffic, MON-tree unnecessarily allocates disk pages with low utilization.

Fig.11 provides us another threshold value for the update threshold. Setting the update threshold as 0.5 in the following experiments seems reasonable in terms of reliability and index performance.

4.6 Spatio-Temporal Query Execution Performance

This subsection covers the spatio-temporal query performance in terms of disk access. A number of workloads are executed against each index structure storing motions of 1000 objects traveling along Chicago and North Central area road map. A workload includes 100 uniformly distributed random queries, each of which has the same spatial and temporal extents. To evaluate the disk access performance, we counted the cache misses which precisely reflects the number of disk access, rather than the number of disk pages read. The random eviction buffer for accessing each structure

is set to hold 30 disk pages, which is almost 1% of each index capacity storing 1000 object's traffic. More specifically, we access the whole MON-tree, including network R-trees and motion R-trees, over the same 30-length-buffer; similarly we access the whole Mapping organization, including Hilbert R-tree and trajectory 2DR-tree over another 30-length-buffer. In addition to that, we access Poly-Line Component over a buffer with a capacity of 5, which does not significantly affect the overall access performance of Mapping and MON-tree.

Executing workloads with different *query_spatial_extent* and different *query_temporal_extent* characteristics gives an idea about the index's behavior against varying *spatio-temporal* range query characteristics. More specifically, *spatial_extent* is the percentage of area covered by the query to complete spatial area, and *temporal_extent* is the percentage of the query's duration to the whole simulation length. Based on the previous setup environment, we test the impact of query's *spatial_extent/temporal_extent* variation on the disk access performance by means of calculating the accumulated cache misses while executing each query.

Since the behavior of index structures against query characteristics does not change significantly with different road networks, our test results include only the outcomes from the North Central case.

In Figs. 12 and 13, it is seen that MV3DR-tree ratifies its typical characteristic such as its small dependency on *spatial_extent* variation and large dependency on *temporal_extent* variation. In addition to that, we observed that with increasing *temporal_extent*, auxiliary 3DR-tree of MV3DR-tree does not alleviate high disk access as expected, because the query execution algorithm still needs to access a large number of MV-tree leaf nodes which dominate the number of accessed auxiliary 3DR-tree index nodes. Interestingly, with increasing *temporal_extent*, pure 3DR-tree gives performance results a number of times better than 3DR-tree of MV3DR-tree, because pure 3DR-tree clusters the

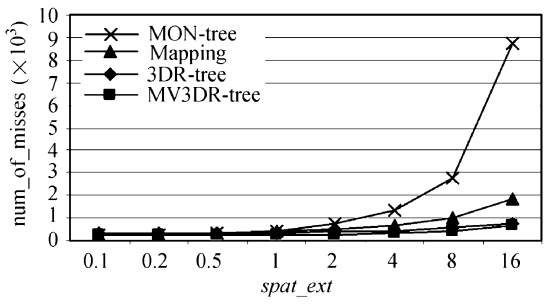


Fig.12. Query execution performance against varying spatial_extent when $DS = 1000$, *update_threshold* $= 0.5$, *temp_extent* $= 1\%$ and *network* $=$ North Central.

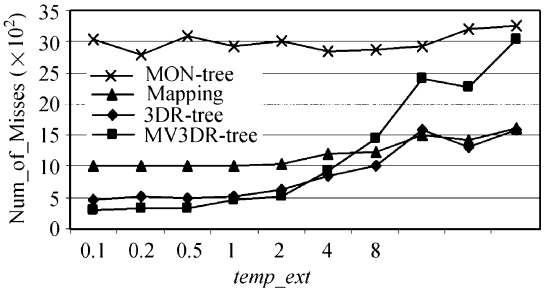


Fig.13. Query execution performance against varying *temporal_extent* when $DS = 1000$, *update_threshold* $= 0.5$, *spatial_extent* $= 8\%$ and *network* $=$ North Central.

motions based on both spatial and temporal characteristics. At this point, in order to improve the MV3DR-tree performance on *large temporal_extent* condition, it would be beneficial to utilize a specialized buffer that has a policy favoring MV-tree leafs to keep them in the buffer.

One of the interesting observations that may be derived from the graphs is the fact that although 3DR-tree is not the best at any time, it is the only one that protects its moderate success against the changes of query window's characteristics. Moreover, the behavior of 3DR-tree to change *spatial_extent* and *temporal_extent* is almost the same, because 3DR-tree directly combines temporal dimension with spatial dimensions. However, as is pointed out in Section 2, quality deterioration is traded off against this performance success.

We now concentrate on network-based index solutions, Mapping and MON-tree. Mapping structure shows a dependency to *spatial_extent* variation while it remains almost stable with the increasing temporal extent of the queries in comparison to MV3DR-tree. As the *spatial_extent* of query increases, the number of edges covered by the query increases and eventually results in higher number of queries over the transformed trajectory 2DR-tree. However, the variation of *temporal_extent* does not affect the number of covered edges by the query. The expected deterioration of Mapping for window queries with large *spatial_extent* is already depicted in Fig.5. Furthermore, the MON-tree degrades with increasing *spatial_extent* much faster than Mapping. This vulnerability is inevitable in part because MON-tree searches all motion R-trees corresponding to edges inside the query's spatial area, whereas Mapping technique sends a number of queries that equal the number of edges inside the query's spatial area to only one index (trajectory 2DR-tree) holding the mapped motions. Another reason is about the network R-tree of MON-tree and Hilbert R-tree of Mapping as stated previously. Hilbert R-tree with its high space utilization and shallow architecture effectively minimizes the

spatial section of query, especially for highly loaded indices. Consequently, this empirical study proves that loaded MON-tree, even with an average traffic, is most dependent to *spatial_extent* variation.

On the contrary, like the Mapping structure, MON-tree performance is almost stable against the *temporal_extent* variations. This is reasonable because MON-tree's main structure is based on road-network spatial data. The temporal data does only shape relatively small motion R-trees; therefore query's temporal section has almost no effect on the number of accessed motion R-trees.

Although the MON-tree's behavior against the variations of query characteristics has been as expected, its performance is the worst for almost all conditions. The main reason is that the underlining architecture is based on the edges and this increases the MON-tree's network R-tree load dramatically. It is expected that with route model as proposed in original paper[8], MON-tree will show better results on query execution and storage consumption while keeping its behaviors against the *spatial_extent/temporal_extent* variations.

5 Conclusion

In this study, we compared the performance of the 4 well-known index structures, storing efficiently the moving object locations over a road network. We achieved outcomes that have either affirmative or complementary aspects. In general, the results of performance studies show that none of the organizations in the experiments is preferable for all the circumstances, nevertheless the success of each organization always changes with varying the data set size and query workload characteristics. We observed the deficiency of network-based index organizations. More specifically, we include 3DR-tree that is expected to be the worst case in our experiments; on the contrary, in most circumstances we found out 3DR-tree shows more stable results than the other organizations in terms of query execution performance. This success comes by compromising the reliability of query results set.

As future work, we may reiterate the storage consumption and query execution performances with a more generic network model, namely route model. It is evident that applying a more generalized network model would positively affect network-specialized index solutions, specially the MON-tree structure, while the generic solutions would remain unaffected.

In addition to route model solution, MON-tree storage consumption can be reduced by integrating all header pages into a single configuration page (or into consecutive pages). This attempt reduces not only the disk access but also the storage consumption. The low utilization problem of MON-tree leaf nodes can be solved by ignoring updates on these edges.

The number of cache misses (the overall disk access) can be improved significantly with better buffer replacement policies. The cache performance (defined as the number of all hits to overall number of reads) varies significantly from index to index. For example after executing workload with 100 query having average spatial/temporal extents, we merely recorded 5% cache performance for querying MV3DR-tree and 14% for querying 3DR-tree and 21% for querying MON-tree and lastly 62% for querying Mapping organization. This observation proves that specialized buffer organizations considering the index structure and/or contents of pages can be valuable for designing robust index structures. Similar attempts such as in [18–20] have already emerged recently.

References

[1] Guttman A. R-trees: A dynamic index structure for spatial searching. In *Proc. the ACM SIGMOD Conference on Management of Data*, Boston, USA, June 18–21, 1984, pp.47–57.

[2] Theodoridis Y, Vazirgiannis M, Sellis T. Spatio-temporal indexing for large multimedia applications. In *Proc. the 3rd IEEE Conference on Multimedia Computing and Systems* (*ICMCS 1996*), Hiroshima, Japan, June 17–23, 1996, pp.441–448.

[3] Pfoser D, Jensen C S, Theodoridis Y. Novel approaches to the indexing of moving object trajectories. In *Proc. the 26th International Conference on Very Large Databases*, Cairo, Egypt, Sept. 10–14, 2000, pp.395–406.

[4] Nascimento M A, Silva J R O. Towards historical R-trees. In *Proc. ACM Symposium on Applied Computing* (*ACM-SAC 1998*), Atlanta, Georgia, Feb. 27–Mar. 1, 1998, pp.235–240.

[5] Tao Y, Papadias D. MV3DR-tree: A spatiotemporal access method for timestamp and interval queries. In *Proc. the 27th International Conference on Very Large Databases*, Roma, Italy, Sept. 11–14, 2001, pp. 431–440.

[6] Jensen C S, Pfoser D. Indexing of network constrained moving objects. In *Proc. the 11th Int. Symp. Advances in Geographic Information Systems* (*ACM-GIS 2003*), New Orleans, USA, Nov. 7–8, 2003, pp.1–8.

[7] Frentzos E. Indexing objects moving on fixed networks. In *Proc. the 8th Int. Symp. Spatial and Temporal Databases* (*SSTD 2003*), Santorini Island, Greece, July 24–27, 2003, pp.289–305.

[8] Almeida V T, Güting R H. Indexing the trajectories of moving objects in networks. *GeoInformatica*, 2005, 9(1): 33–60.

[9] Kollios G, Gunopulos D, Tsotras V J. On indexing mobile objects. In *Proc. ACM Symp. Principles of Database Systems* (*PODS 1999*), Philadephia, USA, May 31–June 2, 1999, pp.261–272.

[10] Saltenis S, Jensen C S, Leutenegger S T, Lopez M A. Indexing the positions of continuously moving objects. In *Proc. the SIGMOD Int. Conf. Management of Data*, Dallas, USA, May 16–18, 2000, pp.331–342.

[11] Tao Y, Papadias D, Sun J. The TPR*-tree: An optimized spatio-temporal access method for predictive queries. In *Proc. the Int. Conf. Very Large Data Bases* (*VLDB 2003*), Berlin, Germany, Sept. 9–12, 2003, pp.790–801.

[12] Spatial Index Library (SaIL). http://u-foria.org/marioh/spatialindex/ index.html.
[13] Kamel I, Faloutsos C. Hilbert R-tree: An improved R-tree using fractals. In *Proc. the 20th VLDB Conf.*, Chile, Sept. 12–15, 1994, pp.500–509.
[14] Kollios G, Gunopulos D, Tsotras V, Delis A, Hadjieleftheriou M. Indexing animated objects using spatiotemporal access methods. *IEEE TKDE*, 2001, 13(5): 758–777.
[15] Papadias D, Zhang J, Mamoulis N, Tao Y. Query processing in spatial network databases. In *Proc. the 29th Int. Conf. Very Large Data Bases* (*VLDB 2003*), Berlin, Germany, Sept. 9–12, 2003, pp.802–813.
[16] Wolfson O, Sistla A P, Chamberlain S, Yesha Y. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 1999, 7(3): 257–387.
[17] Ding Z, Zhou X. Location update strategies for network-constrained moving objects. In *Proc. 13th Int. Conference on Database Systems for Advance Applications* (*DASFAA 2008*), New Delphi, India, March 19–21, 2008, pp.644–652.
[18] Leutenegger S T, Lopez M A. The effect of buffering on the performance of R-trees. *Transaction on Knowledge and Data Engineering*, 2000, 12(1): 33–44.
[19] Brinkhoff T. A Robust and self-tuning page-replacement strategy for spatial database systems. In *Proc. Conference on Extending Database Technology* (*EDBT 2002*), Prague, Czech Republic, March 25–27, 2002, pp.533–552.
[20] Sacco G M. Index access with a finite buffer. In *Proc. the Very Large Data Bases Conf.*, Athens, Greece, Aug. 25–29, 1997, pp.301–309.



Utku Kalay received the B.S. degree in electrical engineering from Istanbul Technical University in 1997 and M.S. degree in computer science from University of Southern California, Los Angeles in 2000. He received the Ph.D. degree in computer engineering from Yildiz Technical University. He is currently working as an associate professor and lecturer. His main research interests are robust index organizations, moving object databases and information retrieval.



Oya Kalipsiz received her M.S. degree in computer engineering from Istanbul Technical University in 1984. She received her Ph.D. degree from Istanbul University in 1989 with a study on Hospital Information Systems. She is currently working as the head of Computer Engineering Department of Yildiz Technical University. Her main research interests are software engineering, database systems, data mining, system analysis, and management information systems.