INDEXING AND QUERY PROCESSING SPATIOTEMPORAL DATA TRAJECTORIES
A spatiotemporal database consists of data that are related to both time and space.

Problem arises while indexing and querying these type of data. Processing queries in this type of data is more demanding due to the extra semantics involved.

Mostly, work in this field is guided by spatial and temporal domains without looking at the spatiotemporal aspect as a whole.

The proposed method explains these types of data along with the queries involving them. It also proposes novel access methods and finally a comparison between them to come up with the best solution.
Spatiotemporal Data Representation

- A point is represented in 3D space, two of the dimensions refer to the space while the other dimension represent the time component.
- The movement of the point from one area in a particular time to another area in different time is shown by the polyline. This movement is known as the trajectory of the data.
- To record the movement of objects, position samples are stored and interpolated linearly.
- Sampled positions become the endpoints of the line segments of polylines and the movement of a point object is represented by the entire polyline in 3D space.
- In classical spatial or temporal databases, only positions at particular area or time is available. The research considers other information such as speed, acceleration etc.
Past Work

- In most of the past related work, it deals with spatial data changing discreetly over time and do not take continuous change into account.
- The problem that isn’t addressed in any of these work is the preservation of trajectories.
- The previous work treats the data as normal line segments and group them together with respect to only spatial properties such as proximity.
- The proposed work preserves the trajectory by not only looking at the spatial aspects but also by grouping the line segments according to the trajectories they belong to.
The research focuses on three types of spatiotemporal queries:

1. Coordinate Based Queries: Point, Range, Nearest-Neighbor queries in 3D space.
2. Trajectory Based Queries: is based on the topology of the trajectory (topological queries) and also looks at derived information such as speed (navigational queries)
3. Combined Queries: Considers spatiotemporal range, topological and derived information
Topological Queries

- This involves the whole or part of the trajectory of an object.
- It’s rather expensive
- It extends the SQL basic spatial predicates, *disjoint, overlap, covers*, etc.
- Also, composite predicates based on the basic ones, for example, *enter, cross, bypass*, etc.
- Example: *Did Bus A enter the college at 8am?*
Navigational Queries

- These are derived from the trajectory information.
- For example, speed is the fraction of travel distance over time while heading can be determined by a vector between two specified positions.
- Speed and heading are both very important in real life applications as it can either refer to the current instance or an aggregation of a longer period of time.
- Example: What is Bus A’s top speed?

<table>
<thead>
<tr>
<th>Query Type</th>
<th>Operation</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinate-based Queries</td>
<td>overlap, inside, etc.</td>
<td>range × {segments} → {segments}</td>
</tr>
<tr>
<td>Topological Queries</td>
<td>enter, leave, cross, bypass</td>
<td>range × {segments} → {segments}</td>
</tr>
<tr>
<td>Trajectory-based Queries</td>
<td>Navigational Queries</td>
<td>traveled distance, covered area (top or average), speed, heading, parked</td>
</tr>
</tbody>
</table>
Combined Queries

- It extracts information related to partial trajectories.
- It combines both topological and navigational queries to extract information.
- For example, What were the trajectories of objects after they left Tucson between 7 a.m. and 8 a.m. today, in the next hour?
Access Methods

- Data **access methods** are used to process queries and **access** data.
- This research proposes two new access methods along with modifying the classic R tree and then comes up with fair comparison and evaluation between the three.
R Tree

- It's an height balanced tree with the index records in its leaf nodes containing pointers to actual data. Leaf nodes are in the form (id, MBB) and non leaf nodes are of the form (ptr, MBB).

- A node in the tree corresponds to a disk page and every node contains between m and M entries.

- Insertion is done by traversing a single path from root to leaf level and the path is chosen with respect to the least enlargement criterion. Covering MBBs are adjusted accordingly.

- In case a split is caused, the new entries are reassigned to old nodes and a newly created one according to either Exhaustive, QuadraticSplit, etc.

- To delete, a reverse insertion method is applied, incase it causes an underflow in the node, the node is deleted and its entries re-inserted.

- While searching, it is checked whether the given node entry overlaps a search window. If so, the child node is visited by recursively traversing the tree.

- At each level there might be several entries that overlap the search window, since overlapping MBBs are permitted.
R Tree

- R tree proves to be inefficient because, it introduces a lot of dead space. This leads to reduced prioritization of the index structure capability.
- R trees also fail to capture knowledge about the specific trajectory a line segment belongs to.
- To smoothen these inefficiencies, a small change is made, which is that a line segment can only be contained in four different ways in MBB. This information is stored at the leaf level as (id, MBB, orientation). If trajectories are numbered, a leaf node entry is then stored in the form, (id, trajectory#, MBB, orientation)
This is an extension of the R tree and is called spatiotemporal-R tree.

Insertion differs from that of R tree, it not only considers spatial closeness but also looks at the partial trajectory preservation. (Line segments belonging to the same trajectory are kept together).

This involves a new algorithm, FindNode which returns the node that contains the predecessor.

For insertion, if there is room in the node the new segment is inserted there. Otherwise node split strategy is used.
STR Tree- Insert Algorithm

- It includes an additional parameter called the preservation parameter $p$, that indicates the number of levels that are reserved for preservation of trajectories.

- When a leaf node returned by the `FindNode` is full, the algorithm checks whether $p-1$ parent nodes are full. In case of the them is not full the leaf node is split, otherwise this algorithm invokes `ChooseLeaf` on the subtree including all the nodes further to the right of the current insertion path.

- It is experimentally established that the best choice of a preservation parameter is $p = 2$. A smaller $p$ decreases the trajectory preservation and increases the spatial discrimination capabilities of the index.
STR Tree

- Since the goal is to preserve trajectories in the index, analysis is required while splitting nodes to see what kinds of segments are stored in the node.
- Two segments may belong to the same trajectory or not and may have common endpoints or not. Thus a node can contain four different type of segments:
  a. Disconnected Segments
  b. Forward/Backward connected segments
  c. Bi-connected Segments
STR Tree Split

- Where all segments are disconnected the QuadraticSplit algorithm is invoked.
- Where only one segment is disconnected, the disconnected segment is put in the newly created node.
- In case, where no disconnected segments exits, the most recent backward connected segment is put in the newly connected node.
- Using this insertion and split algorithm, index is obtained that preserves trajectories and considers time as the dominant dimension
TB Tree

- An underlying assumption behind R tree is that all geometries are independent. However, while looking at line segments, it is known that these are all parts of trajectories and this information is only implicitly maintained in both R tree and STR tree.

- TB tree strictly preserves the trajectories such that leaf nodes only contain segments belonging to the same trajectory.

- The R tree property of spatial discrimination is modified here. This has limitations as well that line segments in different trajectories but with spatial closeness will be kept on separate nodes.

- However, this has a huge gain on trajectory preservation and space discrimination would decrease.
TB Tree Insertion

- For insertion, the goal is to cut the whole trajectory into pieces with each of them containing M line segments.
- Firstly, a leaf node that contains its predecessors is found by traversing the tree from the root to each child nodes that overlaps with the MBB of each line segments. Segments are found according to FindNode algorithm.
- Splitting node in this case would violate trajectory preservation. Thus a new leaf node is created if a non full parent node is found. Otherwise, the new node is created at non leaf level that has this new node as the only descendent.
- TB tree grows from left to right.
TB Tree

- Structure of a TB tree is actually a set of leaf nodes each containing a partial trajectory.
- A doubly linked list is introduced that connects leaf nodes in a way that preserves trajectory evolution. This is done so that retrieval of segments later on can be done easily.
- The trajectory is fragmented across 6 nodes and these are connected through a linked list.
Three types of queries were mentioned, coordinate, trajectory and combined queries.

This research focuses on combined queries, not only entries are retrieved according to a given sub space (range query), it also retrieves entries belonging to the same query.
Combined Query in R Tree and STR Tree

- The first step is to retrieve initial set of segments based on the spatiotemporal range which is done by using range-search algorithms.
- Secondly, partial trajectories are extracted from the found segments. Same leaf nodes are checked first and then the other nodes are checked as range search.
Combined Search in TB Tree

■ It is similar to the previous method but differs in retrieving trajectories. While R and STR tree structures only offer modified range search algorithms, TB tree allows to retrieve connected segments without searching, due to the linked lists.

■ Finding segment in the same leaf node is similar to previous example while for different node, the next (previous) pointer to the next (previous) leaf node is followed.

■ In all the mentioned methods, to avoid retrieving same trajectories more than once, the trajectory id is stored and checked before retrieving a new trajectory.
Performance Comparison

- The access methods are studied under varying sets of data and queries and the study was conducted under C implementation.
- Page size of the leaf and non leaf are set to 1024 bytes.
- R tree and STR tree fanout is 28 and 36 respectively for leaf and non leaf nodes.
- TB tree fanout is 31 and 36 for leaf and non leaf nodes respectively.
- Due to lack of real data, synthetic data was generated using GSTD generator of spatiotemporal dataset which creates trajectories of moving data under various distributions.
- Initial points follow Gaussian distribution, and the movement is ruled by random distribution to give unbiased spread of points.
- The temporal resolution is held constant at 100k and the number of trajectories varies between 10 to 1000 resulting in datasets consisting of between 15k-1500k entries.
Space Utilization and Index Size

- R tree construction does not take the unilateral growth of the data in the temporal dimension into account.
- R tree also utilizes space less than the other two tree. TB tree has the smallest space utilization due to large fanout.
- The table below is for dataset of 1000 objects (1500K line segments)

<table>
<thead>
<tr>
<th></th>
<th>R-tree</th>
<th>STR-tree</th>
<th>TB-tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index size</td>
<td>~95 KB per object</td>
<td>~57 KB per object</td>
<td>~51 KB per object</td>
</tr>
<tr>
<td>Space utilization</td>
<td>55%-60%</td>
<td>~100%</td>
<td>~100%</td>
</tr>
</tbody>
</table>
Range Queries

- The three access methods are used for processing range queries. Dataset ranges from 10-1000 as mentioned before and query windows range from 1%, 10%, 20% of total range with respect to each dimension.

- Each query set includes 1000 query windows.

Figure 15: Range queries: varying range, (a) 1%, (b) 10% and (c) 20% in each dimension
Both TB tree and STR tree are trajectory oriented. For a smaller number of trajectories it is more orientated along time than space and this is known as temporal discrimination.

However, when the number of trajectories increase, more segments exists in a given point of time so spatial discrimination becomes more important.

Thus, for a smaller number of trajectories, TB and STR tree performs better as they do not consider the spatial discrimination.

However, TB tree performs well in all three cases compared to R tree and STR tree. STR tree doesn’t perform well in most cases as it contains R tree properties that does spatial discrimination primarily and doesn’t preserve the trajectories.
Time Slice Queries

- This query time is known as zero extent at the temporal dimension, meaning it only focuses on a certain time frame.

- The size of the query window can be arbitrary and 1%, 10% and 100% of the respective range in each spatial dimension is chosen.

![Diagrams](image)

Figure 16: Time slice queries: varying spatial range, (a) 1%, (b) 10% and (c) 100% in each dimension
Combined Queries

- The queries have inner and outer range as 1% and 10% and 1% and 20% in each dimension.
Combined Queries

- TB tree is superior to the other two under all circumstances.
- Apart from the trajectory preservation at each node, it also has the additional data structure for retrieving neighbor nodes that contribute to result. Thus the node access in case of the TB tree is only slightly larger in range query.
- STR tree and R tree only differ in index structure but have the same search algorithm.
The TB-tree supports trajectory-based queries much more efficiently than the R-tree does. At the same time, its performance on typical range queries is competitive to the R-tree.

STR-tree, although designed to combine the benefits of the TB-tree and the R-tree, it usually performs worse than the TB-tree, with the only exception being time slice queries.
Conclusion

- This research presents a set of pure spatiotemporal queries, trajectory based queries and combined queries.

- Simple modification to R tree as well as two new access methods, STR tree and TB tree is presented for indexing the trajectories of moving point object by preserving trajectories.

- Trajectory data is obtained by discreetly sampling movement of the point objects in time and then doing linear interpolation in between the samples.

- STR tree and R tree can also be implemented on top of the R tree and is already being adopted in commercial extensible database systems.

- The performance study presents experiments related to spatial range queries, navigational and combined queries.

- TB tree proves to be an access method well suited for trajectory based queries and also has a good spatial search performance.
Thank You