Trajectory Queries and Octagons in Moving Object Databases*

Hongjun Zhu Jianwen Su Oscar H. Ibarra

Department of Computer Science
University of California at Santa Barbara

ABSTRACT

An important class of queries in moving object databases involves trajectories. We propose to divide trajectory predicates into topological and non-topological parts; extend the 9-intersection model of Egenhofer-Franzosa to a 3-step evaluation strategy for trajectory queries: a filter step, a refinement step, and a tracing step.

The filter and refinement steps are similar to region searches. As in spatial databases, approximations of trajectories are typically used in evaluating trajectory queries. In earlier studies, minimum bounding boxes (MBBs) are used to approximate trajectory segments which allow index structures to be built, e.g., TB-trees and R*-trees. The use of MBBs hinders the efficiency since MBBs are very coarse approximations especially for trajectory segments. To overcome this problem, we propose a new type of approximations, "minimum bounding octagon prism" (MBOP). We extend R*-tree to a new index structure "Octagon-Prism tree" (OP-tree) for MBOPs of trajectory segments. We conducted experiments to evaluate efficiency of OP-trees in performing region searches and trajectory queries. The results show that OP-trees improve region searches significantly over synthetic trajectory data sets to TB-trees and R*-trees and can significantly reduce the evaluation cost of trajectory queries compared to TB-trees.

1. INTRODUCTION

A growing number of new database applications involve access and management of moving object trajectories. Such applications include traffic control and fleet management etc. Although moving object trajectories are chains of line (or curve) segments, they have very different characteristics than typical spatial data such as the asymmetry of time and space. As a result, queries on trajectories tend to focus on properties involving motions (e.g., directions and speed) that do not arise in spatial database queries. An interesting and urgent issue is to develop efficient algorithms for trajectory query evaluation. In this paper, we consider trajectories of 2-dimensional (2-d) moving points that are chains of line segments and study efficient indexing and query evaluation techniques for trajectory queries.

In [19, 15], different types of queries were studied for 2-d moving points. [19] considered predicates to express queries involving trajectories of moving objects on road system. In [15], trajectory queries involving spatio-temporal predicates such as "enter", "leave", and navigation related ones such as "travel distance" were introduced. Trajectory queries are different from traditional spatial database queries. Predicates on trajectories may not even be spatial or temporal (e.g., current moving speed). In this paper, we consider trajectory queries that can be expressed using binary spatio-temporal predicates.

We argue that evaluation of a binary predicate over trajectories can be decomposed into topological predicates and additional conditions on their motions. The former can be dealt with using spatial database techniques while the latter may require "tracing" related trajectories. In particular, we extend the 9-intersection model of [5] to handle topological predicates.

Trajectories of moving objects may be very complex, naive evaluation of trajectory queries can be very expensive. Similar to spatial databases, simple objects can be used as approximations of trajectories and index structures can then be constructed using the approximations. Minimum bounding boxes (MBBs) are the most commonly used approximation method in spatial databases. [15] introduced two index structures for MBBs trajectories. STR-tree is an extension of R*-tree in which each line segment of a trajectory is treated individually with some consideration to group segments of the same trajectory. Another index structure is TB-tree which indexes MBBs of fixed size segments of trajectories. In [8], a splitting strategy was developed to split the input trajectories into fixed number of segments such that the total volume of the resulting MBBs of these segments is minimized. PPR-trees by [13] are used to organize the MBBs of the trajectory segments. [17] studied past movement of discretely moving objects,

*Support in part by NSF grants IIS-9817432 and IIS-0101134. Email: {hongjunz, su, ibarra}@cs.ucsb.edu

CIKM'02, November 4–9, 2002, McLean, Virginia, USA.

and introduced MV3R-tree—a combination of a multi-version R*-tree and a 3-d R*-tree. MV3R-trees can also be used to index MBB approximations of moving object trajectories.

In this paper, we consider indexing the approximations of trajectories, possibly with splitting each trajectory into segments. A starting point is the previous work [3, 21, 22] showing that MBBs are coarse approximations for 2-d spatial objects and that better approximations have the potential of improving the efficiency. Such approximations include circles, ellipses, rectilinear polygons, rotated MBBs, trapezoids, k-corner bounding convex polygons, and convex hull. Indexing techniques including sphere trees [14] and SS-Trees [20] for circles, cell trees [7] and polyh-edratree [10] for convex polygons, and extended R*-trees for different approximation methods [3] were developed for some of these approximations. It is not hard to believe that finer approximations than MBBs can also be used for trajectories in order to improve query performance.

The main technical contributions of this paper are summarized below. We develop a 3-step strategy for evaluating a trajectory predicate as follows.

1. Filtering: perform a region search on approximations of trajectory segments.
2. Refinement: evaluate the region search on trajectory segments in the filtering result.
3. Tracing: follow each trajectory in the refinement result to check additional conditions (e.g., related to motion).

We also introduce a new approximation called "minimum bounding octagon prism" (MBOP) that approximates each trajectory segment by cutting out four triangular prisms from the four corners of its MBB. Based on MBOP, We develop a new index structure Octagon-Prism tree (OP-tree) for MBOPs that extends R*-tree.

Experimental results show that OP-trees reduces the evaluation time for trajectories by more than 50% compared to TB-trees. Unlike TB-trees, the region search time with OP-tree is 20% to 40% less than that for R*-trees in our experiments using 1-d, 2-d, and 3-d region search instances. In particular, experiments show when the time interval in the region search increases, the saving by OP-trees also increases. However, when the size of spatial region in the region search increases, the saving by OP-trees will decrease. We also studied different splitting strategies to approximate trajectories and our preliminary results show that the splitting strategies do not have significant impact on relative performance of OP-trees and the other index structures based on MBBs.

This paper is organized as follows. §2 discusses trajectory queries. § 3 introduces the new approximation method for trajectories. §4 describes the data structure OP-tree. §5 shows how to evaluate trajectory query and region searches using OP-trees. §6 presents the experimental results of performance evaluation. Finally, §7 discusses future work.

2. TRAJECTORY QUERIES

An important class of queries in moving object databases involve trajectories of moving objects. Examples of queries on trajectories include: "find all trucks that were in the area of Santa Barbara county within the last 24 hours", "find the total number of cars passing through the intersection of highways 101 and 154 between 8am and 10am today".

Consider trajectories of 2-d moving points. Following the framework of [19, 15], we assume that each trajectory p is a chain of line segments. Fig. 2 shows a trajectory with 7 line segments.

[19] considered predicates to express queries involving trajectories of moving objects on road system. For example, one query is to find every car that can always reach point p along its route within 5 minutes at any time between 1:00pm and 5:00pm. In [15], trajectory based query involving spatio-temporal predicates such as "enter", "leave", and navigation related ones such as "travel distance" were introduced.

Predicates in trajectory queries may or may not be spatial, temporal, or spatio-temporal. For example, the query to find all cars on Highway 10 that are currently moving faster than some truck is essentially not spatial nor temporal. We consider trajectory queries that are expressed using binary spatio-temporal predicates. We argue that the evaluation of a binary predicate between two trajectories can be decomposed into topological predicates and conditions on their motions:

I. Evaluate topological relationships between (the spatial projections of) the trajectories.
II. Examine additional conditions concerning motions such as moving direction and speed.

For examples, the predicate "collision" consists of intersection of two trajectories (at the same or different times), a time and a moving direction conditions. The above query on cars can be evaluated by first finding all cars that reach p between 1:00pm and 1:35pm and then examining the trajectories.

In relational queries, binary predicates can be used as conditions in selections or in joins. Algorithms for the two cases use different techniques. The situation for trajectory predicates is similar. The focus of the paper is on selection queries using binary trajectory predicates with one trajectory specified by the user.

Before we discuss in detail the evaluation strategies, we make two assumptions that aim at addressing the most common trajectory queries in practice. First, it is often that trajectory predicates are further constrained by a time interval (or instant). Second, the trajectory provided by the user in a selection condition is a spatial region. We note here that our algorithms in this paper will work without the time interval assumption, and can be easily extended to predicates over two trajectories, i.e., without the second assumption.

Decomposition of a trajectory predicate into topological and motion predicates allows us to use techniques for spatial query evaluation in developing efficient algorithms for trajectory queries. In particular, the 9-intersection topological relationship model of [5] naturally leads to a method for the topological part (part I in the above) of trajectory predicates.

Let R be a spatial region in the plane, and I the time interval of interest. We define $bd(R)$, $in(R)$, $ex(R)$ to be the boundary, interior, and exterior of R (resp.).

Let T be a trajectory. We categorize binary trajectory predicates involving T and R during the interval I using the 9-intersection topological relationship model of [5] into the following.

- I: T intersects only $in(R) \times I$.
- B: T intersects only $bd(R) \times I$.
- IB: T intersects only $in(R) \times I$ and $bd(R) \times I$.
- EB: T intersects only $ex(R) \times I$ and $bd(R) \times I$.
- IEB: T intersects each of $in(R) \times I$, $ex(R) \times I$, and $bd(R) \times I$.

For the two remaining classes, interior-exterior is not meaningful since trajectories are continuous, and exterior can be decomposed into multiple regions with IB.

Let Q be a trajectory query with a binary predicate $\phi(R, I)$ where R is a spatial region and I a time interval. We decompose $\phi(R, I)$ into two parts: one involving only topological relationships in one of the categories listed above, and the other concerning other aspects (e.g., motion). In this framework, the trajectory query T consists of two parts: the first part is a region search on trajectories using R and I, and the second part involves a further examination following the chain of lines segments of the trajectory depending on the query predicate.

A more complex trajectory query may involve several (binary) predicates as shown in Example 2.1.

EXAMPLE 2.1. *The query, "find all cars that meets around 7pm the buses that left Santa Barbara station at 5pm", can be evaluated as a two stage selection query:*

1. *Find the locations at 7pm of all buses that left Santa Barbara station at 5pm.*
2. *For each location, find all cars that pass through it around 7pm.* ∎

3. OCTAGON APPROXIMATION

In order to efficiently evaluate trajectory queries, we consider approximation for trajectories and index structures on the approximations. One way to approximate a trajectory is to split it into small groups of line segments and approximate each group with an MBB. However, MBBs are very coarse approximations especially for trajectories and it is very likely that an MBB satisfies the query condition but not the actual trajectory. In this section, we introduce a new approximation called "minimum bounding octagon prism" (MBOP). MBOP approximations are generated by cutting the "corners" of MBBs. We consider "approximation quality" and number of "false hits" in comparing MBOPs with other approximations including MBBs and minimum bounding k-corner convex prisms (k-CN prisms). We observe that the quality of MBOP is close to that of 4-CN prism.

3.1 Approximation of trajectories

Trajectories of moving objects can have complex shapes and large sizes since one trajectory may consist of thousands of line segments. Indexing trajectories can effectively improve trajectory query evaluation. Using techniques from spatial databases, we can approximate trajectories and construct indexes on the approximations. With the indexes, the region search part of trajectory query can be evaluated by first searching through the indexed approximations of trajectories, and then examining the trajectories whose approximations are in the search result. This will reduce accesses to the actual trajectories in the region search evaluation and therefore the cost of the trajectory query evaluation.

MBBs are widely used for multidimensional data. It is straightforward to approximate trajectories with MBBs and to organize the MBBs with index structures such as R*-trees. However, for trajectories that are chains of line segments, it is very inefficient to approximate by a single MBB (1) an entire trajectory, or (2) each line segment of a trajectory [15, 8]. Indeed, (1) would result in large MBBs and (2) would drastically increase the number of MBBs. Consequently, either a significant amount of the false hits would be likely or the index becomes excessively large. [15, 8] suggested to split a trajectory into "segments"[1] of reasonable size and approximate each by an MBB. [15] used a pre-determined size to split each trajectory into equal size segments called "bundles". A Trajectory-Bundle tree (TB-tree) was developed to organize the MBBs of all trajectory segments, with all segments of the same trajectory linked together. [8] developed algorithms to distribute a predefined total number of splits among a set of trajectories and an optimal algorithm and a heuristic are developed to split each trajectory based on the distribution generated. Furthermore, PPR-trees (partially persistent R-trees) [13] were used to index the resulting MBBs of trajectory segments.

It is clear that MBBs are coarse approximations for trajectories. The approach in this paper is to replace MBBs by the finer MBOP approximation and organize MBOPs into a new index structure to achieve better overall performance of trajectory query evaluation.

In our experiments (Section 6), we used the algorithms in [15, 8] to divide trajectories into fixed size segments [15] or variable size segments [8]. Obviously and verified by experiments, our approach is independent of and complementary to splitting strategies: fixing a splitting strategy, algorithms based on the new approximation are more efficient than that for MBBs.

Previous study in spatial databases focused on approximating 2-d spatial objects. Different approximations have been proposed and data structures were developed for them: e.g., sphere trees [14] and SS-Trees [20] for circles, Cell trees [7] and polyhedra-tree [10] for convex polygons. [3] compared MBB, rotated MBB, minimum bounding circle, minimum bounding ellipse, convex hull (CH), and minimum bounding k-corner convex polygon (k-CN) based on point location region queries for 2-d region data. R*-trees augmented by approximations in the leaf nodes were used as indexes.

Most index structures for complex approximations are complex and hard to maintain. In addition, search may consuming significant more CPU cycles. Using R*-trees to organize complex approximations seems simple but it suffers from large number of false hits. Because finer approximations than MBBs require more storage space, the resulting R*-trees are larger and taller which then causes increase of the filter cost. The increased cost may likely wipe out any savings from using the

[1] A trajectory segment is a chain of line segments.

finer approximations. It is desirable to develop an approximation that is of high quality and easy to index.

3.2 Minimum bounding octagon prisms

In this section, we describe the MBOP approximation. MBOP approximations are "conservative" in the sense that the MBB of an object coincides with the MBB of the MBOP of the same object.

The MBOP approximation of a trajectory segment (a 3-d polyline) T is generated by cutting off the corners of the MBB of T. There are many ways to cut an MBB. For example, one can treat T simply as a 3-d object and cut all 8 corners of the MBB of T with 8 planes. The planes used could be arbitrary or they can be normal to some specified vectors as in [10]. However, trajectories are very different from 3-d polylines. The time dimension and spatial dimensions behave differently. For example, a moving point can be at the same location at different time instants, but at most one location at each time instant. It seems natural to cut the MBB based on the projection of the trajectory segment on the spatial dimensions. In particular, an MBOP approximation is the Cartesian production of a spatial approximation and a time interval, where the spatial approximation is a "minimum bounding octagon" (defined below) of the spatial projection of T.

A *bounding octagon* (BO) of a 2-d spatial object S is an octagon that (1) contains S and (2) has two pairs of boundary lines (in the opposite sides) parallel to the two axises, resp. Fig.1(a) shows the MBB (dashed box) and a smallest BO of a 2-d polyline.

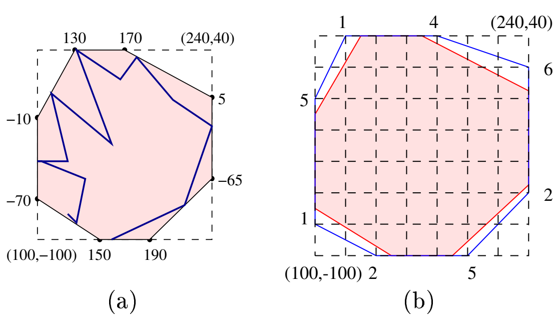


Figure 1: A smallest BO and an MBO

Given a 2-d polyline S, we can construct its BO by trimming four corners of the MBB of S. Clearly the smallest BO of S is always contained in the MBB of S. To represent a smallest BO, we use the MBB augmented by the intersection points of the "corner cutting" lines with MBB. There are at most two such intersection points on each side of the MBB. Let k>1 be an integer. A BO is k-*normalized* if the intersection points can be represented as a k-bit integer from the lower or left boundary of the MBB. For example, if the left, right boundaries are at x_0, x_1 and an intersection point is at x then $\frac{x-x_0}{x_1-x_0}=\frac{i}{2^k-1}$ for some integer i between 0 and 2^k-1.

Definition. Let S be a spatial object and $k > 1$ an integer. A *minimum bounding octagon* of S, denoted as MBO(S), is the smallest k-normalized BO of S.

Fig.1(b) shows the MBO of the polyline in (a) (where $k = 3$) and the smallest BO of the same polyline. Using the results in [4], we developed an algorithm that computes the MBO for a polygon or polyline S. The complexity of the algorithm is $O(n \log n)$, where n is the number of boundary points of S.

For the space requirement of an MBO, 4 real numbers are for the MBB. Each MBO boundary point can be represented as a relative value using k bits, and a total of $8k$ bits are needed. In all of our experiments (Section 6), we use k=8 so each relative value takes 1 byte.

Definition. If T is a trajectory segment with spatial and temporal projections S and I (resp.), the *minimum bounding octagon prism* of T, MBOP(T), is the Cartesian product of MBO(S) and I.

Fig.2 shows the MBOP of a trajectory segment.

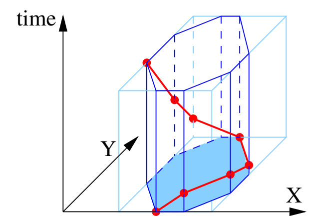


Figure 2: A trajectory segment and its MBOP

In order to understand potential impact of MBOPs in the performance of trajectory queries, we compare MBOPs with MBBs and other approximations using "approximation quality" and the number of false hits.

The *quality* of an approximation of a 3-d polyline T is defined as the ratio of the volume of the convex hull prism of T (a prism whose projection on the spatial dimension is the convex hull of the projection of T) to that of the approximation[2]. The closer the quality of an approximation is to 1, the finer it is.

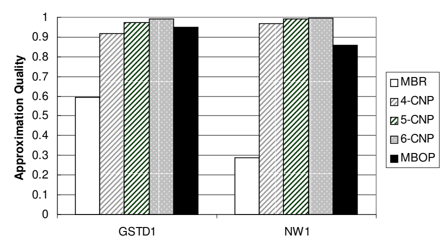


Figure 3: Approximation quality

Fig.3 shows the average approximation quality of MBB, 4-CN prism, 5-CN prism, 6-CN prism, and MBOP for two data sets. The two data sets, GSTD1 and NW1, are generated by the GSTD generator [18] and Brinkhoff's generator [2], resp., where the number of trajectories per set is 1000. Each trajectory is divided into fixed size segments with each group contains at most 41 line segments. Obviously, the approximation quality of MBOP is much better than that for MBBs and close to that for 4-CN prisms. This is mostly expected, except that the approximation quality of MBB (close to 0.6) for GSTD1 is unusually high than that (around 0.37) of MBB for random polygons in our earlier study [9]. This is because the data in GSTD1 is randomly generated and

[2]This concept is modified from a similar one for 2-d region objects in [3].

consequently the convex hull of the projection of a trajectory segments is relatively "fat". Results for variable size segments are similar.

The *number of false hits* of a region search is the number of trajectory segments that do not intersect the search region but whose approximations do. [9] studied the relationship between approximation quality and number of false hits for 2-d polygon data and show that the higher the approximation quality, the smaller the number of false hits in the case of spatial joins. Based on the approximation quality, we expect that the number of false hits for MBOPS in region searches should be better than that for MBBS and close to that for 4-CN prisms. Fig.4 shows the average number of false hits (the Y axis) over 1000 randomly generated region searches (where projections of the search regions on the spatial dimensions are windows) on the two data sets. The results confirm our predictions. The number of false hits for MBOPS is indeed close to that for 4-CN prisms, and is about 50% less than that for MBBS.

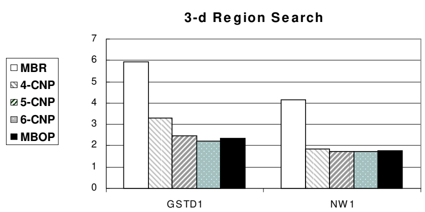


Figure 4: False hits of region search

In summary, MBOP approximations are simple and of hight quality and thus have a great potential in improving the overall performance of region searches on trajectories and trajectory queries.

4. OCTOGON-PRISM TREES

In this section, we present an index structure "Octagon-Prism tree" (OP-tree) for MBOPS of 2-d moving point trajectories. Roughly speaking, OP-trees are extensions of R*-trees. The keys in OP-trees are MBOPS of 2-d moving point trajectory segments.

Similar to R*-trees, entries in the leaf nodes of an OP-tree store pointers to actual objects, i.e., trajectory segments (generated using a splitting algorithm). Each leaf node entry is of the form $(addr, mbp)$, where *addr* is the address of the page that storing a trajectory segment t and *mbp* is the MBOP of t. All pages storing the segments of the same trajectory are linked together by a doubly linked list, so that it is possible to follow the links to access all segments belonging to the same trajectory. Entries in non-leaf nodes are of the form (ptr, mbp), where *ptr* is a pointer to a child node, and *mbp* is the MBOP that contains all the MBOPS stored in the child node pointed by *ptr*. Let M be the maximum number of entries that will fit in one node, and m $(2 \leq m \leq M)$ the minimum number of entries in a node. An OP-tree must satisfy all of the following conditions:

- The root has at least two child nodes unless it is a leaf node.
- Every leaf node, if it is not the root, contains k entries for some $m \leq k \leq M$.
- Every non-leaf node contains k entries, $m \leq k \leq M$.
- All leaf nodes are on the same level.

Fig.5(a) shows an OP-tree where $M{=}4, m{=}2$ with height 2. Fig.5(b) illustrates the MBOs of the MBOPS in the entries of leaf nodes and the ones in the root.

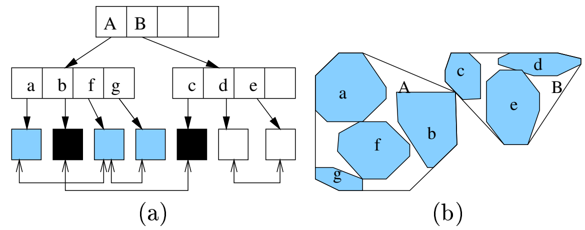


Figure 5: An example OP-tree and MBOs in it

The main difference between an OP-tree and an R*-tree is that in an OP-tree, the keys in each entry are MBOPS instead of MBBS. As we discusses in Section 3, an MBOP includes an MBB and 8 additional bytes for relative positions. If each of the *addr* and *ptr* parts takes 4 bytes, and the node size is 1K byte, the maximum fanout of an OP-tree is 17, while the maximum fanout of an R*-tree is 19.

The insertion and deletion operations of an OP-tree use the technique similar to these operations for R*-trees, except that MBOPS are used instead of MBBS.

Similar to region search on R*-trees, when performing a region search on an OP-tree, we start the search from the root. For every entry in the root, we check if the search region intersects the MBOP in the entry. If the MBOP of an entry intersects the search region, we either repeat the same process for the subtree pointed by the entry or report the address stored in the entry.

Testing intersection of a search region and an MBOP consists of (1) checking the time constraint, and (2) checking the spatial constraint in terms of a spatial region (eg. point or line) intersecting an MBO of the MBOP. We take advantage of the MBB in the MBO and first test if the search region intersects the MBB. If it does, we then test if the MBO intersects the search region. This will help reduce the CPU cost of the region search.

Remark. In [12], a compression technique is developed to reduce the space requirements for MBBS in R*-trees. After the compression, most part of (sometimes the entire) R*-tree can be stored in main memory and the search performance of R*-trees can be improved. The same compression technique can also be applied to the MBB part within each MBOP of OP-trees. Although the fanout gap between the compressed versions of R*-tree and OP-tree will slightly increase. using OP-trees are still expected to cost less in trajectory query evaluation. This is because with the compressed versions, the filter step can mostly be done in main memory and the dominating factor in the overall cost is the cost of the refinement step. With OP-trees, the refinement step costs significantly less than with R*-trees.

5. TRAJECTORY QUERIES AND REGION SEARCHES

In this section, we consider the evaluation of trajectory queries. In Section 2, we argue that evaluation of a trajectory predicate can be decomposed into evaluation of topological predicates and conditions related to the motions of trajectories. In this section, we give technical details of converting the former into region search and the latter into tracing trajectories in the result of the region search. Based on this, we present a 3-step trajectory predicate evaluation strategy based on OP-trees: "filter", "refinement", and "tracing".

As shown in Section 2, a trajectory predicate can be decomposed into topological predicates and motion related conditions. Substantial amount of work has been done on evaluating topological relationships in spatial databases. The conventional approach is to employ region search. In our context concerning trajectory predicates, we found that the following three types of region search of particular importance, depending on the spatial projection S of the search region.

- *1-d region search:* S is a point.
- *2-d region search:* S is a line segment.
- *3-d region search:* S a polygon. In this paper, we only consider the case of a box.

By a careful examination of the classes of topological relationships defined in Section 2, we map the classes to the types of region search that are needed. The following table shows the mapping between the classes in Section 2 and region search types. 1-d region search is also needed for the degenerated cases.

class	I	B	IB	EB	IEB
region search	3-d	2-d	3-d	2-d & 3-d	3-d

Using MBOPs of trajectory segments and an OP-tree index on the MBOPs, region search can be accomplished by the typical filter-refinement combination. A trajectory predicate can be evaluated in the following steps:

1. *Filtering:* a region search on the OP-trees of the input trajectory segments.
2. *Refinement:* for every trajectory segment in the result of the filter step, check if it satisfies the region search condition.
3. *Tracing:* for every trajectory in the result of the refinement step, follow the trajectory to determine whether it satisfies the additional condition in the trajectory predicate.

The tracing step depends on the trajectory predicate. However, once the necessary part (e.g., only one direction) or the entire trajectory is obtained, the truth of the predicate can be determined.

The complex query in Example 2.1 can then be evaluated with the following steps:

1. Perform a 2-d region search to find all bus trajectory segments intersecting the boundaries of Santa Barbara at 5pm.
2. For each trajectory reported by the previous step, check its direction. Tracing every trajectory that left Santa Barbara to find its location at 7pm.
3. For each location found, perform a 1-d region search to find all car trajectory segments that intersect one of these locations at 7pm.

6. EXPERIMENTAL RESULTS

In this section, we present the experimental results on the performance of evaluation trajectory queries using OP-trees. The results show that OP-trees improve trajectory queries and region searches significantly over trajectories generated by the GSTD algorithm and Brinkhoff's algorithm. In particular, the overall I/O cost for trajectory queries is less than half of the cost by using TB-trees, a significant improvement. The I/O and CPU costs for region search are 50% less than that of TB-trees and is 30% less compared with R^*-trees in most cases in our study. When the spatial projection of the search region is fixed, the savings of OP-trees over TB-trees and over R^*-trees increases when the time interval in the region search condition increases. However, if the time interval is fixed, the savings by OP-tree decreases when the size of the spatial search region increases. We also show that no matter what splitting strategy is used, OP-trees outperform the other trees.

6.1 Setup

In our experiments, we compare two performance measurements: the number of I/O accesses and the number of CPU instructions. The I/O cost is calculated by counting the number of page reads. An 8 page LRU buffer is used in all experiments. The CPU cost is measured by the total number of floating point operations. For simplicity, we only consider the number of floating point arithmetic/comparison operations as the number of CPU instructions.

We use synthetic data sets generated by the GSTD generator [18] and Brinkhoff's generator [2]. We show the results from four data sets. *GSTD1* and *GSTD5* were generated using the GSTD generator, which generates uniformly distributed moving points whose movements are also uniformly distributed. *NW1* and *NW5* were generated using Brinkhoff's generator for the road map of German city Oldenburg (NW1), and the road information for San Joaquin County, California, USA (NW5, obtained from TIGER/Line2000 files), resp. The following table shows for each dataset, the total number of trajectories and the total number of line segments.

Data set	GSTD1	NW1	GSTD5	NW5
#Trajctories	1000	1000	5000	5000
Total #lines	100,425	512,330	54,574	229,501

When approximating the trajectories, we first use the fixed size splitting strategy that was introduced in [15]. Pages storing line segments of the same trajectory are linked together. The trajectory segment in each page is approximated by an MBOP and an MBB, then an OP-tree, a TB-tree and a 3-d R^*-tree are constructed on the resulting MBOPs or MBBs (resp.). In our experiments, we consider a page size of 1024, therefore each trajectory segment contains at most 41 line segments. We also used the splitting algorithm of [8].

We did not compare with the tree structures such as PPR-tree [13] or MV3R-tree[17]. Since OP-trees are extensions of R^*-trees, it is possible to extend PPR-trees or MV3R-tree by replacing the MBBs in the trees with MBOs. It would be interesting to compare the performance of an persistent version of OP-tree with these trees. [8] compared PPR-tree and R*-trees, however the data sets they used are different from ours. The data sets in [8] are moving regions and the average changes of movements per trajectory is very small. Since the trajectories of moving points we considered change their movements very frequently and consist of large number of line segments, it is unclear whether PPR-trees still win over R*-trees.

6.2 Region search

We first consider the performance of OP-trees for region search. Using each of the OP-trees, TB-trees, and R*-trees constructed, we perform 1-d, 2-d, and 3-d region searches. For each type of search, we used a set of 1000 search uniformly distributed (in the data space) instances and compute the average performance.

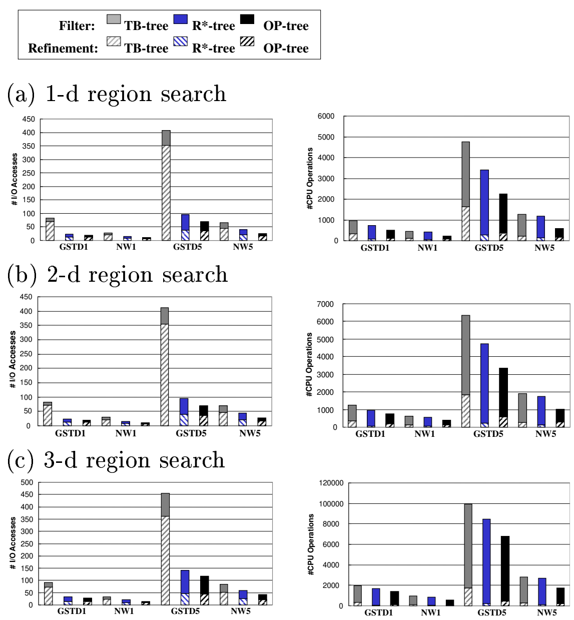


Figure 6: Region search performance

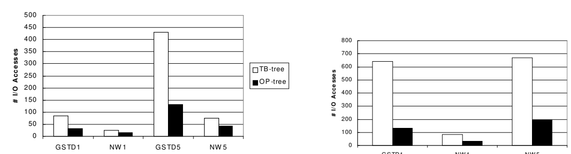
Fig.6(a) shows the I/O (left) and CPU cost (right) for 1-d region search on all datasets. The average length of time intervals in search conditions is 10% of the entire time duration of all input trajectories.

The figure shows that the overall I/O and CPU cost for OP-trees is less than half of the cost for TB-trees, and is about 20% to 40% less than that for R*-trees. In particular, for the filter step, TB-tree performs worst because it organizes data primarily based on their time duration. OP-tree performs the same or slightly worse than R*-trees. This is due to the smaller fan-out of OP-tree. Even though MBOPs are finer approximations than MBBs, it is possible that more nodes in OP-tree need to be accessed because each node contains less number of entries. The filter step CPU cost for OP-trees is generally higher than that for R*-trees since intersection test with MBOPs costs more CPU than that with MBBs. The CPU and I/O cost of the refinement step for TB-tree and R*-tree are the same since they both use MBB approximations. For OP-trees, both the I/O and CPU cost for the refinement step are significantly smaller compared to R*-trees and TB-trees. This is due to the reduction of false hits by using MBOPs. Since the cost of the refinement step is larger than the cost of the filter step in most cases, the overall I/O cost and CPU cost for 1-d region searches using OP-trees are less than those of R*-trees and TB-trees.

Fig.6(b) and (c) show the I/O and CPU costs for 2-d and 3-d region searches, resp. The average search window size is 0.1% of the data space. Time constraints are the same as in 1-d region searches. These results are very similar to the ones for 1-d region search. Note that the improvement of the performance of OP-trees over the other trees actually increase when the size of the data set increases.

6.3 Trajectory queries

We now present experimental results of OP-tree for trajectory queries. We consider both simple and complex queries and compare OP-tree with TB-tree [15] since TB-trees outperform R*-trees when line segments are not organized according to their trajectories. Even with proper grouping of line segments by trajectories, R*-trees are not as efficient as OP-trees in our experiments (not provided in this paper).



(a) Simple query (b) Complex query

Figure 7: Performance of trajectory queries

Fig.7(a) presents the average I/O cost of TB-tree and OP-tree for a set of 1000 simple trajectories queries of the type "find the positions of all the points at time t which previously enters a given window within $[t_1, t_2]$." The overall cost of OP-trees is only half of the cost of TB-trees. The results are consistent with the results for region search. This is because both index structures uses the same split algorithm and the I/O cost for tracing step is determined by the number of results reported by the region search. Since MBOPs can reduce false hits in region search, it also reduces the cost in the tracing stage. Therefore, with OP-trees, the saving in I/O cost for trajectory queries to TB-trees is even more that the savings in region queries.

A complex query similar to the one in Example 2.1 requires more than one region search. Fig.7(b) presents the average I/O cost of TB-trees and OP-trees for a set of complex trajectory queries similar to the one in Example 2.1. The results show even more savings by OP-trees than the simple query case because for each region search, OP-trees cost less than TB-trees. [3]

For complex trajectory queries, we might encounter different combinations of region searches. Both the time and the spatial constraints can vary depending on the trajectory query and trajectories in the database. In fact, the results in Fig.7(b) is the aggregated behavior

[3]The results from data set GSTD5 are similar but are omitted because the cost is much larder and hard to shown with others.

for the evaluation. In order to understand the sensitivity of time and spatial constraints, we investigate the performance behavior of OP-trees under when only the time constraint or spatial constraint (but not both) in the region search changes.

6.4 Sensitivity of time constraints

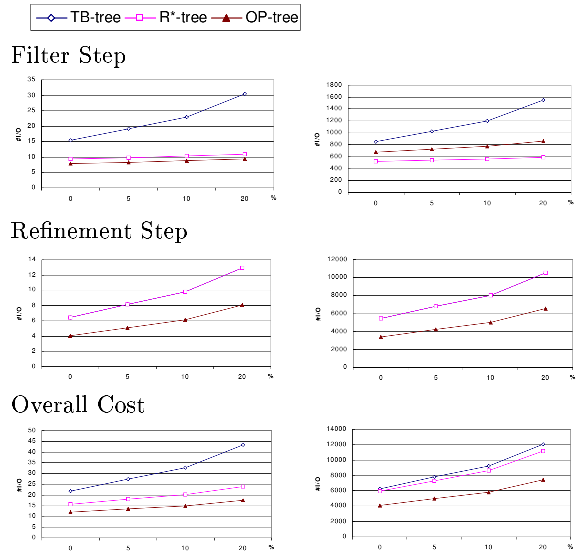
We first consider the time constraints. We vary the average length of the time interval in the search condition. Fig.8 shows the CPU and I/O costs for 3-d region searches on NW1 using both OP-tree, TB-tree, and R*-tree when the average length of time intervals are 0%, 5%, 10%, and 20% (resp) of the time range of input trajectories. Experimental results for other datasets are similar. Note that the cost of refinement step are the same for TB-trees and R*-trees, thus only one is shown.

The figure shows that the CPU and I/O costs for all three trees increases when the length of time range increases. This is straightforward because there are more answers in queries with larger time interval. However, the gap between OP-trees and R*-trees remains almost the same. One possible explanation for this is that both OP-trees and R*-trees group MBOP and MBBs (resp.) more according to their overall locality than their temporal locality. The trees are thus more sensitive to spatial constraint changes in query condition than changes in time constraints. However, The gap between TB-trees and the other two trees increases (especially for the filter step). This is because for TB-trees, time has a larger impact on the query performance. The MBBs stored in the trees are grouped by their closeness in the time dimension. When the time interval in the search condition increases, many more nodes will be visited.

6.5 Sensitivity of spatial constraints

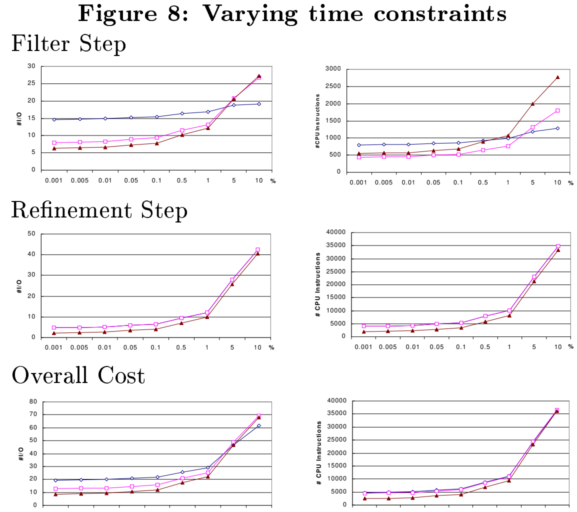
Now we fix the time constraint in the region search and change the spatial constraint. Fig.9 shows the results of 3-d region search on dataset NW1, where the average search window size varies from 0.001% to 10% of the entire data space. Results for other datasets are similar. The cost for all trees increase when window size increases due to larger number of results. However the savings of OP-tree and R*-trees over TB-trees becomes less when window size increases. For TB-trees, the filter step cost does not change a lot when window size increases because they are less sensitive to spatial changes in the search conditions. Although the overall cost for TB-trees increases when window size increases, it is mainly due to the cost increase in the refinement step. This is different from OP-trees and R*-trees, for which the cost for both the filter and refinement steps increases when the window size increases.

We also notice that the gap between OP-trees and R*-trees decreases when window size increases. We suspect that the reason for this scenario to happen is that when search window size increases, the size of the answer increases quadratically, however, the saving of false hit gained by using MBOP approximations only increases linearly because most of the savings are from intersections of search point and the boundaries of the MBBs.



(a) IO cost (b) CPU cost

Figure 8: Varying time constraints



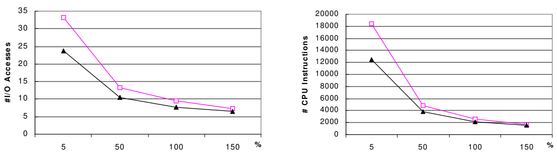
(a) IO cost (b) CPU cost

Figure 9: Varying spatial constraints

6.6 Impact of splitting

In the previous experiments, we use a fix size splitting strategy for grouping and approximating trajectories. We also considered different splitting strategies. We apply the LAGreedy algorithm of [8] to split the trajectories. In particular, we use the algorithm to distribute various number of splits (from 5%,to 150% of the total number of objects). Fig.10 presents the overall I/O and CPU costs of OP-tree and R*-trees for 3-d region search. It is obvious that for all cases, OP-tree consumes less I/O and CPU because it uses MBO approximations instead of MBBs.

Note that the cost of both trees decreases when the split percentage increases. This is because the approximation quality increases when more splits are used. However, one should expect a cost increase after the split percentage increases to a certain point because



(a) I/O cost (b) CPU cost

Figure 10: Impact of splitting

the total number of approximations becomes very large causing the increase of the cost of the filter step. This does not show in our experiments since the data set used is still rather small.

7. FUTURE WORK

A number of questions raised from the study need to be further explored. For example, the performance improvements of OP-trees over R*-trees shrinks when search window size increases. Although we suspect that this is due to the reduction of the false hits ratio, it is unclear if there are other factors.

One remaining question is how to extend MBOP and OP-trees to higher dimensions. Similar approach to MBOP can be used, However, for higher dimensions, the cutting will become too complex and it is not clear whether this method will indeed wins over MBBs.

8. REFERENCES

[1] B. Becker, P. G. Franciosa, S. Gschwind, T. Ohler, G. Thiemt, and P. Widmayer. Enclosing many boxes by an optimal pair of boxes. In *Proc. STACS*, 1992.

[2] T. Brinkhoff. Generating network-based moving objects. In *Proc. SSDBM*, 2000.

[3] T. Brinkhoff, H-P. Kriegel, and R. Schneider. Comparison of approximations of complex objects used for approximation-based query processing in spatial database systems. In *Proc. ICDE*, 1993.

[4] D. Dori and M. Ben-Bassat. Circumscribing a convex polygon by a polygon of fewer sides with minimal area addition. *Computer Vision, Graphics, and Image Processing*, 24, 1983.

[5] M. J. Egenhofer and R. Franzosa. Point-set topological spatial relations. *Int. Journal of Geo. Info. Systems*, 5(2):161–174, 1991.

[6] Y. J. Garcia, M. A. Lopez, and S. T. Leutenegger. On optimal node splitting for R-trees. In *Proc. VLDB*, 1998.

[7] O. Günther. The design of the cell tree: An object-oriented index structure for geometric databases. In *Proc. ICDE*, 1989.

[8] M. Hadjieleftherious, G. Kollios, and V. J. Tsotras. Efficient indexing of spatialtemporal objects. In *Proc. EDBT*, 2002.

[9] L. Han, H. Zhu, and J. Su. Experimental evaluation of filter effectiveness. In *Proc. ACM GIS*, 2000.

[10] H. V. Jagadish. Spatial search with polyhedra. In *Proc. ICDE*, 1990.

[11] T. L. Kay and J. T. Kajiya. Ray tracing complex scenes. In *Proc. ACM SIGGRAPH*, 1986.

[12] K. Kim, S. K. Cha, and K. Kwon. Optimizing multidimensional index trees for main memory access. In *Proc. ACM SIGMOD*, 2001.

[13] A. Kumar, V. J. Tsotras, and C. Faloutsos. Designing access methods for bitemporal databases. *IEEE Trans. Knowledge and Data Engineering*, 10(1), 1998.

[14] P. J. M. Oosterom. *Reactive Data Structures for Geographic Information Systems*. PhD thesis, Dept. of Computer Science, Leiden University, Netherlands, 1990.

[15] D. Pfoser, C. Jensen, and Y. Theodoridis. Novel approaches to the indexing of moving object trajectories. In *Proc. VLDB*, 2000.

[16] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[17] Y. Tao and D. Papadias. Mv3r-tree: a spatio-temporal access method for timestamp and interval queries. In *Proc. VLDB*, 2001.

[18] Y. Theodoridis, J. R. O. Silva, and M. A. Nascimento. On the generation of spatio-temporal datasets. In *Proc. SSD*, 1999.

[19] M. Vazirgiannis and O. Wofson. A spatiotemporal model and language for moving objects on road networks. In *Proc. SSTD*, 2001.

[20] D. White and R. Jain. Similarity indexing with the SS-tree. In *Proc. ICDE*, 1996.

[21] H. Zhu, J. Su, and O. H. Ibarra. Extending rectangle join algorithms for rectilinear polygons. In *Proc. WAIM*, 2000.

[22] H. Zhu, J. Su, and O. H. Ibarra. Towards spatial joins for polygons. In *Proc. SSDBM*, 2000.