Distance Queries for Complex Spatial Objects in Oracle Spatial

Ying Hu, Siva Ravada, Richard Anderson, Bhuvan Bamba
Oracle Spatial and Graph
{ying.hu, siva.ravada, richard.anderson}@oracle.com, bhuvanbamba@gmail.com

ABSTRACT
With the proliferation of global positioning systems (GPS) enabled devices, a growing number of database systems are capable of storing and querying different spatial objects including points, polylines and polygons. In this paper, we present our experience with supporting one important class of spatial queries in these database systems: distance queries. For example, a traveler may want to *find hotels within 500 meters of a nearby beach*. In addition, this paper presents new techniques implemented in Oracle Spatial for some distance-related problems, such as the maximum distance between complex spatial objects, and the diameter, the convex hull and the minimum bounding circle of complex spatial objects. We conduct our experiments by utilizing real-world data sets and demonstrate that these distance and distance-related queries can be significantly improved.

Categories and Subject Descriptors
H.2.8 [**Database Management**]: Database Applications – *spatial databases and GIS.*

Keywords
Distance query, within-distance query, convex hull, maximum distance, diameter, minimum bounding circle.

1. INTRODUCTION
With the proliferation of global positioning systems (GPS) enabled devices such as mobile devices, a wide range of location-based applications require that backend database systems are capable of storing and querying different spatial objects including points, polylines and polygons. Not only commercial database systems including Oracle [28], IBM DB2 [18] and Microsoft SQL Server [23], but also open source database systems including PostgreSQL [29], MySQL [25] and MongoDB [24], provide support for spatial query processing. One important class of spatial queries in these database systems is *distance queries*, which include *nearest-neighbor queries*, and *within-distance queries*. For a large number of spatial objects, some of which can be very complex, supporting high-performance distance queries can be challenging. As nearest-neighbor queries have been well studied in the research community, we focus on within-distance queries in this paper, especially when the given spatial objects (or query spatial objects) are complex geometries, including complex polylines (such as coastlines), complex polygons (such as congressional districts) and complex heterogeneous collections (such as composite hydrography features that consist of complex polylines for narrow portions of rivers and complex polygons for wide portions of rivers and lakes), and briefly discuss nearest-neighbor queries. Within-distance queries are used to find all spatial objects within a given distance of a given spatial object. As within-distance queries can be translated to topological relationship *INTERSECTS* queries [27] with a new spatial object, *BUFFER*(given spatial object, given distance), within-distance queries are also called *buffer queries* [7]. For example, a traveler may want to *find hotels within 500 meters of a nearby beach*. Other examples include determination of the set of houses within 5 miles of U.S. Highway 101, the set of hydrography features within 1 mile of the region of the recent Yosemite fire, and so on. We discuss in-memory R-tree based optimization techniques implemented in Oracle Spatial to speed up these distance queries. We also discuss in-memory R-tree based implementations for some distance-related problems, such as computing the maximum distance between complex spatial objects, and the diameter, the convex hull and the minimum bounding circle of a complex spatial object. We make the following contributions to support these distance and distance-related queries in Oracle Spatial:

- We discuss in-memory R-tree based optimization techniques that are used in the distance queries. These in-memory R-tree techniques are extended from our previous works on topological relationship queries [16, 17].
- We present in-memory R-tree based implementations for some distance-related problems such as computing the maximum distance between complex spatial objects, and the diameter, the convex hull and the minimum bounding circle of a complex spatial object. To the best of our knowledge, this paper is the first work to show that hierarchical tree structures, such as the in-memory R-tree structure, can help optimize the above mentioned problems by speeding up the computation of angle-related minima/maxima.
- We utilize real-world data sets to conduct an experimental study. The experimental results demonstrate that the in-memory R-tree techniques are effective for distance and distance-related queries.

The rest of this paper is organized as follows: Section 2 reviews related work. Section 3 presents an overview of within-distance query processing. Sections 4 and 5 describe in-memory R-tree based optimization techniques used in distance query processing for minimal bounding rectangles (MBRs) and actual spatial objects, respectively. In section 6, we extend the in-memory R-tree techniques to some distance-related problems. Section 7 presents results of an experimental study using real-world data sets. Section 8 concludes the paper.

SIGSPATIAL '14, November 04 - 07 2014, Dallas/Fort Worth, TX, USA
Copyright 2014 ACM 978-1-4503-3131-9/14/11…$15.00
http://dx.doi.org/10.1145/2666310.2666385

2. RELATED WORK

Nearest neighbor queries are widely used in many location-based applications, and have been well studied for two decades [33, 13, 14, 5, 6, 22]. Spatial indexes such as R-tree indexes [12, 22] can be built to speed up nearest neighbor queries against a large number of spatial objects. Two basic approaches, Depth-First Search (DFS) and Best-First Search (BFS), can be used in these spatial indexes to find nearest neighbors. In addition, several spatial pruning rules have been proposed to further speed up nearest neighbor queries. For example, *minimax distance* (MINMAXDIST), which was introduced to specify the minimum of the maximum possible distances, can be used in these pruning rules [33]. Recently, Emrich et al. [8] gave an optimal pruning criterion for minimal bounding rectangles (MBRs).

Although within-distance query processing [7, 6] is not studied as much as nearest neighbor query processing in the research community, within-distance queries are also widely used in many location-based applications. For instance, one of geo-fencing test cases in the ACM SIGSPATIAL GIS Cup 2013 programming contest [31] required evaluations of the within-distance query performance for qualified point and polygon pairs. Note that in some complex cases of within-distance queries, both given spatial objects and test spatial objects can be complex geometries including polylines, polygons and even heterogeneous collections. Spatial indexes such as R-tree indexes are normally built on the MBRs for test spatial objects. The two-step query processing mechanism is as follows: First, the *filter* step utilizes the spatial index to determine a candidate set. Next, the *refinement* step examines test spatial objects in the candidate set using computational geometry algorithms [3, 11, 22].

For complex spatial objects, brute-force computational geometry operations, such as distance operations, can be expensive. For example, the distance between two complex spatial objects is defined as the *minimum distance* between the two spatial objects. The brute-force distance computation has O(*m***n*) time complexity, where *m* and *n* are the number of line segments (or points) in each of the two spatial objects. To avoid these expensive operations, *0-Object filtering* and *1-Object filtering* techniques was proposed [7]. The 0-Object filtering technique is used for two MBRs, and similar to the MINMAXDIST technique used in the nearest neighbor query processing. The 1-Object filtering technique is extended from the 0-Object technique, and instead of two MBRs, one actual spatial object and the MBR of the other spatial object are used. In addition, there are several techniques that were introduced to approximate complex spatial objects [34]. For example, the *strip tree* [1] is a hieratical structure that can represent both open curves and closed curves. The bounding boxes in a strip tree do not require that the sides are parallel to the coordinate axes. The strip tree was shown to be useful in some operations such as point-in-polygon operations, intersection operations, and so on [1]. The *sphere tree* [30] that is built with hierarchical bounding spheres can be used to efficiently compute distance between complex spatial objects. The *TR*-tree* [4] is a representation of polygonal objects. A polygon is decomposed into trapezoids and an R*-tree is built on top of these trapezoids to speed up the *INTERSECTS* operation.

Like the hierarchical tree structures described above, our previous works [16, 17] also use a hierarchy structure (more specifically an in-memory R-tree) in which leaf entries can be made up of points, and line segments, including boundary line segments of a polygon element and line segments of a polyline element. With experiments on real-world data sets, we have demonstrated that the performance of topological relationship queries for different complex spatial objects can be significantly improved by using the in-memory R-tree techniques not only in the refinement step but also in the filter step. In PostGIS [29] and JTS [19], a similar concept of "prepared geometry" can be used in *INTERSECTS* and *CONTAINS* operations. Furthermore, in-memory R-tree structures on line segments of complex polygons were used by two of the top three ranked competitors in ACM SIGSPATIAL GIS Cup 2013 programming contest [40, 21, 31].

As Earth is not flat, Euclidean distance is not an adequate measure over a large area, such as North America or the whole world. Several techniques used in geodetic distance queries for points [35] have been discussed. In Oracle Spatial, both great-circle distance and more accurate Vincenty's formulae [37] are supported. In addition, 2D geodetic coordinates (longitude, latitude) are internally converted to 3D Earth-centered coordinates in Oracle Spatial. Consequently, any in-memory and on-disk R-tree structures built on them are also 3D. However, to simplify our description, we will mainly use 2D non-geodetic or projected examples in this paper.

Besides the distance (or the minimum distance) between complex spatial objects, some of other distances have been used in spatial applications. For example, the *Hausdorff distance* was introduced to measure the similarity between trajectories with the help of R-tree structure based techniques [26]. In addition, the *maximum distance* between complex spatial objects can be used in some complex spatial analysis applications, such as complete-linkage clustering or farthest neighbor clustering [9]. One approach to the maximum distance problem is to obtain a convex hull [10, 2] for each of complex spatial objects and then use a rotating caliper algorithm [36] to find the maximum distance between these convex hulls. Similarly, the *diameter* that is the maximum distance between two points in a complex spatial object can also be computed by a rotating caliper algorithm [36]. Finally, another distance-related problem -- the *minimum bounding circle* (MBC) of a complex spatial object has been solved by different algorithms [38, 32, 2]. The MBC can also be used in some complex spatial analysis applications. For example, in a facility location planning application, a new hospital may be located in the center of the MBC of a region to better serve all households in the region.

3. OVERVIEW OF WITHIN-DISTANCE QUERY PROCESSING

Because within-distance queries can be translated to topological relationship *INTERSECTS* queries with a new spatial object, *BUFFER*(given spatial object, given distance), a straightforward solution is to create a buffer spatial object from the given spatial object, and use the new buffer spatial object to execute the *INTERSECTS* query. As topological relationship queries including *INTERSECTS* queries have been studied in our previous works [16, 17], we focus on an alternative approach in this section, in which an expensive buffer operation on the given spatial object is not needed. We assume that an R-tree index has been built on a large number of test spatial objects [20]. An in-memory R-tree is built on a given spatial object, and for example, an in-memory R-tree is shown in Figure 1. Note that the in-memory R-tree structure for a given spatial object is different from the R-tree index for test spatial objects, as the former is built on top of line segments or points in the given spatial object and resides only in memory while the latter is built on top of MBRs of test spatial objects and resides on disk and in memory. An overview of the

within-distance query processing is given by using a pseudo code description, as follows.

Algorithm: Within-Distance Query Processing

```
   input : given spatial object (Q), test spatial
           objects (TS), R-tree index on test
           spatial objects (RI), distance (D)
   output: set of test spatial objects within
           distance (D) of Q
 1 build in-memory R-tree (IMR-tree) for Q;
 2 push the root of RI into stack;
 3 while (stack is not empty)
 4   pop one R-tree index entry (RIE) from stack;
 5   if (RIE is a non-leaf entry in RI)
 6     if (RIE's inclusion flag is set to TRUE)
 7       for (each child entry of RIE)
 8         set its inclusion flag to TRUE;
 9         push it into stack;
10     else
11       determine if non-leaf RIE's MBR is within
         distance (D) of Q using IMR-tree;
12       if (RIE can be discarded)
13         continue; /* go back to line 3 */
14       else if (RIE can be included)
15         for (each child entry of RIE)
16           set its inclusion flag to TRUE;
17           push it into stack;
18       else
19         for (each child entry of RIE)
20           push it into stack;
21   else  /* a leaf entry in RI */
22     if (RIE's inclusion flag is set to TRUE)
23       put this TS into result set;
24     else
25       determine if leaf RIE's MBR is within
         distance (D) of Q using IMR-tree;
26       if (RIE can be discarded)
27         continue; /* go back to line 3 */
28       else if (RIE can be included)
29         put this TS into result set;
30       else
31         determine if TS is within distance (D) of
           Q using IMR-tree;
32         if (TS can be included)
33           put this TS into result set;
```

As shown in the above pseudo code, the R-tree index component and the computational geometry component are tightly integrated in Oracle Spatial. Lines 2-30 outline the filter step where both the R-tree index for test spatial objects and the in-memory R-tree for the given spatial object are used. Lines 31-33 describe the refinement step where exact test spatial objects and the in-memory R-tree for the given spatial object are used. In Sections 4 and 5, we will discuss the usage of in-memory R-tree in the filter step (for MBRs) and in the refinement step (for actual test spatial objects), respectively.

Note that in some spatial database systems, the filter step uses MBRs only, because their R-tree index component for the filter step and their computational geometry component for the refinement step are loosely coupled. The loose-coupling approach is easy to implement, because the filter step or the R-tree index component only deals with MBRs and does not take complex spatial objects into account. However, the loose-coupling approach can be inefficient to process within-distance queries, especially when given spatial objects are large and complex, like state regions, interstate highways and long rivers. This is because a large number of false-positives can be filtered out if the given spatial object or its in-memory R-tree structure can be used in the filter step, as shown in our experiments in Section 7.1.

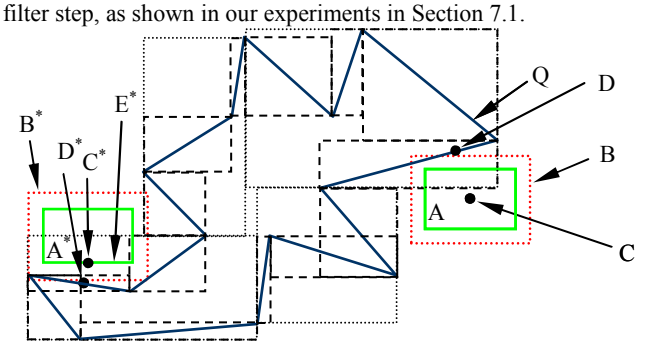

Figure 1. MBR optimizations for within-distance queries, with an in-memory R-tree built on a given polygon (Q).

4. MBR OPTIMIZATIONS

This section discusses the optimization techniques in the filter step. We use the term "non-leaf MBR" to denote a MBR from a non-leaf entry in an R-tree index, and the term "leaf MBR" to denote a MBR from a leaf entry in an R-tree index. We discuss non-leaf MBR optimizations in Section 4.1 and leaf MBR optimizations in Section 4.2, and then briefly discuss how these MBR optimization techniques can also be used in nearest neighbor queries in Section 4.3.

4.1 Non-Leaf MBR Optimizations

Lines 6-20 of the pseudo code description in Section 3 are used for non-leaf MBR optimizations. The basic idea is to apply the following three steps.

- Step 1. If the distance between the given spatial object and a non-leaf MBR is larger than the given distance, the non-leaf MBR and its descendants can be discarded.
- Step 2. If the non-leaf MBR is fully within the given distance of the given spatial object, all its descendant test spatial objects can be included in the final result.
- Step 3. Otherwise, the child MBRs of the non-leaf MBR are fetched from disk or buffer cache for further processing.

Step 1 for non-leaf MBR optimizations can be achieved by using the non-leaf MBR to search the in-memory R-tree of the given spatial object. Assume that MBR (A) is a non-leaf MBR in Figure 1. We can increase MBR (A) by the given distance to obtain another MBR (B), and use MBR(B) as a query window to search the in-memory R-tree of the given spatial object (Q).

Ideally, step 2 for non-leaf MBR optimizations is equivalent to determining if the Hausdorff distance from the non-leaf MBR (A)

to the given spatial object (Q) or max{min{distance(a, q) : q ∈ Q} : a ∈ A} is less than the given distance. However, to the best of our knowledge, it is still an open research problem to design an efficient R-tree based Hausdorff distance algorithm for any non-discrete spatial objects (such as lines or polygons), although several R-tree based Hausdorff distance algorithms for discrete points have been proposed [26]. To approximate step 2 for non-leaf MBR optimizations, we use the center point (C) of the non-leaf MBR (A) to find its closest point in spatial object (Q) (i.e. point D in Figure 1) by searching the in-memory R-tree of spatial object (Q). We further determine if all points in MBR (A) are within the given distance of point D. If so, all the descendant test spatial objects of non-leaf MBR (A) can be included in the final result. Note that the above approximation is based on the following inequations.

max{distance(a, D) : a ∈ A} ≥
min{max{distance(a, q) : a ∈ A} : q ∈ Q} ≥
max{min{distance(a, q) : q ∈ Q} : a ∈ A}.

We can obtain max{distance(a, D) : a ∈ A} ≥ min{max{distance(a, q) : a ∈ A} : q ∈ Q}, simply by using the definition of min{max{distance(a, q) : a ∈ A} : q ∈ Q}. Note that point D is on spatial object (Q). The rest is based on the following theorem.

Theorem 1: If there are two spatial objects A and Q,

min{max{distance(a, q) : a ∈ A} : q ∈ Q} ≥
max{min{distance(a, q) : q ∈ Q} : a ∈ A}.

Proof Sketch: To prove the above theorem, we can assume that point a' in A and point q' in Q are two points having min{max{distance(a, q) : a ∈ A} : q ∈ Q}, and point a" in A and point q" in Q are two points having max{min{distance(a, q) : q ∈ Q} : a ∈ A}. It is not difficult to prove that distance(a', q') ≥ distance(a", q') ≥ distance(a", q"). Thus, min{max{distance(a, q) : a ∈ A} : q ∈ Q} ≥ max{min{distance(a, q) : q ∈ Q} : a ∈ A}.

When the above two steps are completed, a non-leaf MBR may be left undecided if the non-leaf MBR is partially within the given distance of the given spatial object. Child MBRs of the non-leaf MBR will have to be fetched in step 3 for further processing.

4.2 Leaf MBR Optimizations

Lines 22-30 of the pseudo code description in Section 3 are used for leaf MBR optimizations, which are similar to the non-leaf MBR optimizations described in Section 4.1.

- Step 1. If the distance between the given spatial object and a leaf MBR is larger than the given distance, the test spatial object enclosed by the leaf MBR can be discarded.
- Step 2. If any one of the four edges in the leaf MBR is fully within the given distance of the given spatial object, the test spatial object enclosed by the leaf MBR can be included in the final result.
- Step 3. Otherwise, the test spatial object enclosed by the leaf MBR will be fetched from disk or buffer cache for further processing, which will be discussed in detail in Section 5.

Step 1 for leaf MBR optimization is similar to step 1 for non-leaf MBR optimizations. For example, assume that MBR (A*) in Figure 1 is a leaf MBR. MBR (A*) is increased by the given distance to obtain another MBR (B*), and then MBR (B*) is used as a query window to search the in-memory R-tree of the given spatial object (Q).

Since a leaf MBR encloses only a single test spatial object, there must be a point in each of four edges of the leaf MBR that is common to the test spatial object. Therefore in step 2 for leaf MBR optimizations, if any one of four edges in the leaf MBR is fully within the given distance of the given spatial object, the test spatial object enclosed by the leaf MBR can be returned in the final result. For example, if the bottom edge (E*) of the leaf MBR (A*) (shown in Figure 1) is fully within the given distance of the given spatial object (Q), the test spatial object enclosed by the leaf MBR (A*) can be returned in the final result. This is the same as determining if the Hausdorff distance from the bottom edge (E*) of the leaf MBR (A*) to the given spatial object (Q) or max{min{distance(e, q) : q ∈ Q} : e ∈ E*} is less than the given distance. As described in Section 4.1, an efficient R-tree based Hausdorff distance algorithm for any non-discrete spatial objects (such as lines or polygons) is still an open research problem. Thus to approximate step 2 for leaf MBR optimizations, we use the center point (C*) of the bottom edge (E*) to find its closest point in spatial object (Q) (i.e. point D* in Figure 1) by searching the in-memory R-tree of spatial object (Q), and then determine if all points in the bottom edge (E*) are within the given distance of point D*. If so, the test spatial object enclosed by the leaf MBR (A*) can be included in the final result. According to Theorem 1 in Section 4.1, max{distance(e, D*) : e ∈ E*} ≥ max{min{distance(e, q) : q ∈ Q} : e ∈ E*}.

Note that in the 2D non-geodetic case shown in Figure 1, the four boundary edges can be checked in step 2 for leaf MBR optimizations. In the 2D geodetic case where 3D minimum bounding boxes (MBBs) are used to accommodate 3D Earth-centered coordinates, the six boundary faces can be checked in step 2 for leaf MBB optimizations.

4.3 Usage in Nearest Neighbor Queries

The techniques in Sections 4.1 and 4.2 can also be adopted in nearest neighbor queries, especially when the given spatial object is complex. For example, MINDIST, or simply the distance between the given spatial object and an MBR can be computed by using the MBR to search the in-memory R-tree of the given spatial object, find the nearest neighbor point to the MBR, and calculate the minimum distance. To get MINMAXDIST between the given spatial object (Q) and an MBR, which corresponds to min{max{min{distance(a, q) : q ∈ Q} : a ∈ E_i} : E_i ∈ four boundary edges (E_1 –E_4) of MBR} for 2D non-geodetic cases, we reapply the technique that is based on Theorem 1 in Section 4.1. In other words, for each edge E_i in the MBR, we can compute max{distance(a, D_i) : a ∈ E_i}, where D_i is the point on the spatial object (Q) that is closest to the center of boundary edge E_i. This is done in a similar manner to step 2 for non-leaf MBR optimizations described in Section 4.2. When max{distance(a, D_i) : a ∈ E_i} is computed for each of four boundary edges in the MBR, their minimum, or min{max{distance(a, D_i) : a ∈ E_i} : E_i ∈ four boundary edges (E_1 –E_4) of the MBR} can be used to approximate MINMAXDIST between the given spatial object (Q) and the MBR. Then, both MINDIST and MINMAXDIST between the given spatial object and the MBR can be used in nearest neighbor query's different pruning rules, which have been well studied [33, 5, 22].

5. DISTANCE BETWEEN COMPLEX SPATIAL OBJECTS

As described in Section 2, the brute-force distance computation between two spatial objects has O(*m***n*) time complexity, where *m* and *n* are the number of line segments or points in each of the two

spatial objects. In other words, each pair of line segments or points in the two spatial objects is checked. Obviously, the brute-force approach can be expensive when *m* and *n* are large. Because the in-memory R-tree for the given spatial object is already built, we can use the following two approaches to determine if a test spatial object is within the given distance of a given spatial object.

The first approach, which does not build an in-memory R-tree for the test spatial object, can be described as follows.

Algorithm: Within-Distance Approach 1

```
   input : given spatial object (Q), in-memory R-
           tree (IMR-tree) for Q, test spatial
           object (TS), distance (D)
   output: TS, if TS is within distance (D) of Q
 1 for (each line segment or point (E) in TS)
 2 |  determine the minimum distance between E and
   |  Q using IMR-tree;
 3 |  if (E is within distance (D) of Q)
 4 |  └return TS;
 5 if (TS has polygons)
 6 |  check if any point of Q is in polygons of TS;
 7 |  if (one point of Q is in polygons of TS)
 8 |  └return TS; /* distance = 0 */
```

The first approach is used if the test spatial object is not as complex as the given spatial object, e.g., if the number of vertices in the test spatial object is much less than that in the given spatial object. The second approach can be used if the test object is as complex as the given spatial object. The second approach builds an in-memory R-tree for the test spatial object, and it can be described as follows.

Algorithm: Within-Distance Approach 2

```
   input : given spatial object (Q), in-memory R-
           tree for Q, test spatial object (TS),
           distance (D)
   output: TS, if TS is within distance (D) of Q
 1 build in-memory R-tree for TS;
 2 use two in-memory R-trees to find the nearest
   neighbor pair (NNP), and get min_distance;
 3 if (min_distance <= D)
 4 └return TS;
```

Note that in line 2 of the second approach, a priority queue can be used in the Best-First Search (BFS) manner to find the nearest neighbor pair from the two in-memory R-trees [13]. Furthermore, when one polygon geometry object can fully contain the other geometry, the distance between the two spatial objects will be zero. This case can be quickly determined by using one point from the second spatial object to run in-memory R-tree based point-in-polygon methods that have been described [15]. Since there could be many test spatial objects in the refinement step, the cost of building in-memory R-trees for them in the second approach cannot be neglected. In practice, several neighbor line segments or points can be grouped into a single leaf entry in order to limit the number of leaf entries in an in-memory R-tree. As the number of leaf entries in an in-memory R-tree is limited, the cost of building an in-memory R-tree is also limited. In addition, lower memory usage makes the whole system more scalable.

The above two approaches can also be adopted to compute the distance between two spatial objects. For example, in Approach 1, lines 3-4 are replaced with checking if the minimum distance is found, and finally the minimum distance is returned. In Approach 2, lines 3-4 are simply replaced with returning the distance between the nearest neighbor pair.

6. EXTENSIONS

Section 6 discusses extensions of the proposed in-memory R-tree techniques to other distance-related problems such as convex hull, maximum distance, diameter, and minimum bounding circle.

6.1 Convex Hull

As the convex hull of a spatial object has many practical applications, such as the rotating caliper algorithms for maximum distance and diameter described in the next two subsections, this subsection discusses the usage of an in-memory R-tree for a spatial object when computing the convex hull of the spatial object. Note that several convex hull algorithms have been proposed [2]. For example, the Graham scan algorithm is one of the well-known algorithms with O(*n*log n*) time complexity [10].

To simplify our description of the in-memory R-tree based convex hull algorithm, we assume that a spatial object comprises discrete points as shown in Figure 2. Note that the algorithm can be extended to other spatial objects such as polylines and polygons. In Figure 2, there are four leaf nodes (I, J, K, and L) and each leaf node has two entries. The eight leaf entries are points A, B, C, D, E, F, G, and H. Node M is the parent of nodes I and J, node N is the parent of nodes K and L, and nodes M and N are also two children in the root node.

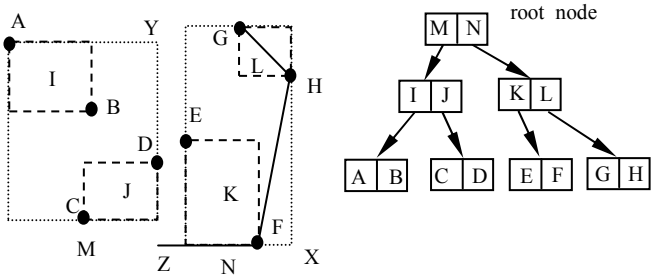

Figure 2. An in-memory R-tree is used to find the convex hull.

Since either of the x-axis or y-axis extreme points must be on the convex hull, we can take the y-axis minimum point (i.e. point F in Figure 2) as the starting point in the convex hull. Point F can be easily found from the in-memory R-tree. We also assume that the convex hull is counter-clockwise. To find the next point that is on the convex hull from point F, we take a horizontal line from point F, i.e. line ZF in Figure 2. Then we use the in-memory R-tree to find point (*) that will have the maximum angle ∠ZF(*). We start with the root node. There are two entries in the root node: node M and node N. The maximum angle from node N is angle ∠ZFX that is 180°. The maximum angle from node M is angle ∠ZFY. We push them into a priority queue based on a descending order of the angle values. Then we dequeue to get node N, and for the two entries (i.e. node K, node L) in node N, compute their maximum angle. The maximum angle from node K is 90° and the maximal angle from node L is angle ∠ZFH. We push them into the priority queue, and dequeue to get node L. For node L, we compute the maximum angles from point H and point G and push

them into the priority queue. Finally, we obtain point H as the point where the maximum angle ∠ZFH occurs. The usage of the priority queue here is similar to that for the incremental nearest neighbor (NN) algorithm [13]. However, the angle, instead of the distance, is used in the priority queue. Note that the maximum angle can only be found at one of the four corner points of an MBR, and any other angles constructed from other points in the MBR are not larger than the maximum angle.

Once point H is found, line segment FH will be on the convex hull. Then from point H, we can proceed to use the in-memory R-tree to determine that the next point is point G, where the maximum angle ∠FHG occurs. The algorithm completes once point F is found to be on the convex hull again.

The following pseudo code describes the complete algorithm.

Algorithm: RTREE_CONVEX_HULL

```
   input : in-memory R-tree (IMR-tree)
   output: point list (L) of convex hull
 1 find the y-axis minimum point (F) using
   IMR-tree, and build a horizontal line (ZF);
 2 Q = F;  /* the y-axis minimum point */
 3 P = Z;  /* PQ is the horizontal line now */
 4 R = NULL; /* for next point on convex hull */
 5 add Q into L;
 6 while (R != F)  /* if point F found again */
 7 |  R = FIND_CONVEX_HULL_NEXT_POINT(IMR-tree, P,
   |  Q);
 8 |  P = Q; /* prepare for the next one */
 9 |  Q = R;
10 |  add R into L;
11 return L;
```

Algorithm: FIND_CONVEX_HULL_NEXT_POINT

```
   input : in-memory R-tree (IMR-tree), point P,
           point Q
   output: point R /* next point on convex hull */
 1 initialize a descending priority queue (DPRIQ);
 2 for (each entry in root node of IMR-tree)
 3 |  compute the maximum angle constructed by any
   |  one of 4 corner points in this entry, and P
   |  and Q;
 4 |  put this entry with its maximum angle into
   |  DPRIQ;
 5 while (DPRIQ is not empty)
 6 |  dequeue from DPRIQ into entry (EN);
 7 |  if (EN is a leaf entry)
 8 |  |  return EN; /* this is the point */
 9 |  else  /* EN is a non-leaf entry */
10 |  |  for (each child entry in EN)
11 |  |  |  compute the maximum angle constructed by any
   |  |  |  one of 4 corner points in this entry, and P
   |  |  |  and Q;
12 |  |  |  put this entry with its maximum angle into
   |  |  |  DPRIQ;
```

The above algorithms can also be extended to 2D geodetic spatial objects, where 2D geodetic coordinates are converted to 3D Earth-centered coordinates. We assume the geodetic convex hull is fully inside a hemisphere. Two modifications are needed. The first modification is that an angle on a sphere or more accurately a spheroid (approximating Earth) is used. The angle is constructed by two geodetic lines. Thus, in line 1 of Algorithm RTREE_CONVEX_HULL, a horizontal line (ZF) will be replaced with a geodetic line, which has bearing = 90° to the true north at point F, if the geodetic convex hull is inside the Northern Hemisphere. The second modification is in lines 3 and 11 of Algorithm FIND_CONVEX_HULL_NEXT_POINT. The maximum angle in a 3D MBB can occur on the 12 boundary edges of the 3D MBB. Furthermore, we need to consider only points that are on the surface of Earth. Note that not all points in the 3D MBB are on the surface of Earth. Some points are under the surface of Earth, some points are above the surface of Earth, and the rest points are on the surface of Earth. When the 12 boundary edges are examined, the maximum angle is found. Any other feasible angles from the 3D MBB are not larger than the maximum angle.

6.2 Maximum Distance and Diameter

The maximum distance between two spatial objects can be computed by a rotating caliper algorithm when two convex hulls (one for each of the two spatial objects) are obtained [36]. With the in-memory R-tree being built and used in the queries such as distance computation in previous sections, we can extend the in-memory R-tree techniques to compute the maximum distance between two spatial objects. One approach is to use the in-memory R-trees to build convex hulls as described in Section 6.1 and then use the rotating caliper algorithm. Another approach is to modify the algorithms in Section 5 for the maximum distance. For example, in Approach 1 of Section 5, the steps performed in lines 5-8 are not required, lines 2-4 can be replaced with determining the maximum distance between each line segment or point in the test spatial object and the given spatial object using the in-memory R-tree for the latter, and finally the maximum distance can be returned. In Approach 2 of Section 5, nearest neighbor pair in line 2 can be replaced with farthest neighbor pair, and finally the distance between the farthest neighbor pair is returned.

Like the maximum distance, the diameter of a spatial object can also be computed by a rotating caliper algorithm when the convex hull of the spatial object is obtained [36]. Thus, the in-memory R-tree based convex hull algorithm described in Section 6.1 can be used. Similarly, Approach 2 described in Section 5 can also be adopted to compute the diameter that is the maximum distance between two points in a complex spatial object.

6.3 Minimum Bounding Circle

The minimum bounding circle (MBC) computation has been shown to be related to the problem of locating a third point (R) in order to minimize the angle ∠PRQ, given that points P and Q are on the MBC [32].

Before we proceed with the complete algorithm for the MBC problem, we explain how to use the in-memory R-tree to quickly find the third point, given two points on the bounding circle. In Figure 3, we assume that point F and point H are on a bounding circle. We also assume that the third point is on the left of line

segment FH. Note that three non-collinear points can define a circle. To find the third point on the bounding circle, we need to find the point (*) such that the angle ∠F(*)H is minimum. There are two cases, where the angle ∠F(*)H can be minimum in an MBR.

Case 1: Point (*) is one of the four corner points. For example in Figure 3, angle ∠FAH is minimum for all points in node M, and point A is the top-left corner point of the MBR (node M).

Case 2: Point (*) is the intersection point between line FH and the MBR, and the intersection point is outside line segment FH. In this case angle ∠F(*)H is 0°. Figure 4 shows that case 2 (where the minimum angle = 0°) occurs in (c), but not in (a) and (b).

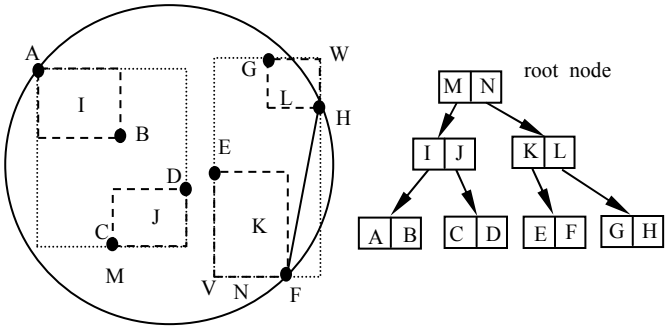

Figure 3. How to find the third point, given that point F and point H are on a bounding circle.

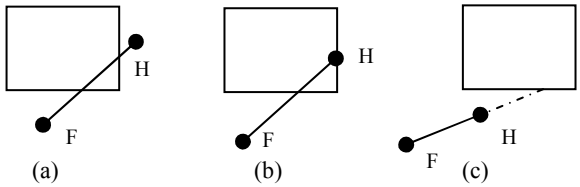

Figure 4: The minimum angle = 0° case occurs in (c), not in (a) and (b).

The angle ∠FWH is the minimum for points in node N. For the two entries of the root node in Figure 3, both angle ∠FAH and angle ∠FWH are pushed into an ascending order priority queue. As angle ∠FWH is less than angle ∠FAH, the top entry in the priority queue is node N. When the top entry (node N) is dequeued, node L and node K will be used to compute their minimum angle, and they will be pushed into the priority queue. For node L, it is still the angle ∠FWH. For node K, it is the angle ∠FVH. Because the angle ∠FWH is the minimum in the priority queue, the top entry that is node L is dequeued. Points G and H are entries in node L. As point H has been used as one of two points (F and H) on the bounding circle, we only need to push the angle ∠FGH into the priority queue. Now the angle ∠FAH is the top entry in the priority queue. Similarly, the search will continue from node M, to node I, and finally to point A, where the MBC is found. The following pseudo code shows the above algorithm to find the third point.

Algorithm: FIND_MBC_THIRD_POINT

```
   input : in-memory R-tree (IMR-tree), point P,
           point Q
   output: point R /* the third point */
 1 initialize an ascending priority queue (APRIQ);
 2 for (each entry in root node of IMR-tree)
 3 |  compute the minimum angle constructed by any
   |  one of 4 corner points, or the intersection
   |  point (the minimum angle = 0° case) in this
   |  entry, and P and Q;
 4 |  put this entry with its minimum angle into
   |  APRIQ;
 5 while (APRIQ is not empty)
 6 |  dequeue from APRIQ into entry EN;
 7 |  if (EN is a leaf entry)
 8 |  |  return EN; /* this is the point */
 9 |  else  /* EN is a non-leaf entry */
10 |  |  for (each child entry in EN)
11 |  |  |  compute the minimum angle constructed by any
   |  |  |  one of 4 corner points, or the intersection
   |  |  |  point (the minimum angle = 0° case) in this
   |  |  |  entry, and P and Q;
12 |  |  |  put this entry with its minimum angle into
   |  |  |  APRIQ;
```

A line segment on the convex hull can be used to get the first two points [32]. Thus, the complete algorithm to compute the MBC is described as follows.

Algorithm: RTREE_MBC

```
   input : in-memory R-tree (IMR-tree)
   output: MBC
 1 find the y-axis minimum point (F) using
   IMR-tree, and build a horizontal line (ZF);
 2 P = F;  /* the y-axis minimum point */
 3 S = Z;  /* the horizontal line */
 4 Q = FIND_CONVEX_HULL_NEXT_POINT(IMR-tree, S,
   P);
 5 while (1)
 6 |  R = FIND_MBC_THIRD_POINT(IMR-tree, P, Q);
 7 |  if (∠PRQ >= 90°)
 8 |  |  return the circle constructed by P and Q;
   |  |  /* the case of two points: PQ is the diameter
   |  |  of this circle */
 9 |  else if ((∠PQR <= 90°) && (∠RPQ <= 90°))
10 |  |  return the circle constructed by P, Q and R;
   |  |  /* the case of three points */
11 |  else if (∠PQR > 90°)
12 |  |  Q = R;  /* move on, use Q side */
13 |  else
14 |  |  P = R; /* move on, use P side */
```

Note that the logic in lines 7-14 of Algorithm RTREE_MBC was also described in the previous work [32]. The above algorithms can also be extended to 2D geodetic spatial objects, where 2D geodetic coordinates are converted to 3D Earth-centered coordinates. First, we assume that Earth's surface can be approximated to a sphere, so that three points on Earth's surface

can also construct a circle. We also assume the geodetic MBC is fully inside a hemisphere. In line 2 and line 11 of Algorithm FIND_MBC_NEXT_POINT, we need to figure out where the minimum angle occurs in a 3D MBB. Note that those points are in 3D Cartesian space, and an angle is also a 3D angle constructed by two 3D straight lines. It can be shown that the minimum angle can occur at one of the 8 corner points of the 3D MBB, or at the intersection point, which is similar to the minimum angle = 0° case in Figure 4 (c). Thus, once these points are examined, the minimum 3D angle is found in the MBB and any other 3D angles in the 3D MBB must be not less than the minimum 3D angle. Note that the logic in lines 7-14 of Algorithm RTREE_MBC also applies to the geodetic MBC.

7. EXPERIMENTS

In this section, we discuss experiments conducted using real-world data sets. We are mainly concerned with complex spatial objects as they are present frequently in expensive distance queries. We use three complex spatial object data sets including polygons, polylines, and heterogeneous collections. For complex polygons, we have 50 US states and 1061 local regions including Manhattan, NY and Long Island, NY. The areas of the 1061 local regions range from 5.83 km^2 to 78,200 km^2 with an average area of 4,330 km^2. The average number of vertices in the 50 US states and the 1061 local regions is 1,755 and 520, respectively. For complex polylines, we take 239 US interstate highways. The 239 US interstate highways have an average length of 296.5 km, and the longest one I-90 is 4,290.6 km. The average number of vertices in the 239 US highways is 1,075. For complex heterogeneous collections, we consider the 38 longest rivers of the United States [39], the longest of which is the Missouri River. The average number of vertices in the 38 US rivers is 31,952. Note that the wide portions of these rivers are represented as polygons while the narrow portions are represented as polylines. We also take two large spatial data sets, and use the above three complex spatial object data sets to run queries against them. For a large point data set, we use the US Business Area (ABI) data set consisting of over 10 million locations. For a large polygon data set, we use the BLOCKS data set consisting of around 11 million US census blocks. The BLOCKS data set (about 9 GB) is the largest of the above data sets. All experiments were conducted on an Intel Xeon server with 6GB of RAM. All these data sets have geodetic coordinates (longitude, latitude) in the World Geodetic System (WGS 84).

7.1 Compare Different MBR Optimizations

In this subsection, we conduct experiments to measure the following within-distance query performances.

```
SELECT COUNT(*)
FROM BLOCKS a
WHERE SDO_WITHIN_DISTANCE(a.geometry,
 :given_spatial_object, 'distance=4 unit=km') = 'TRUE';
```

All three complex spatial object data sets are used as given spatial objects, or query spatial objects. Five different configurations, where different MBR optimizations described in Section 4 can be enabled individually, are used.

- In Configuration A, the filter step only uses MBRs (or more specifically MBBs for geodetic spatial objects) and the refinement step uses the techniques discussed in Section 5.
- In Configuration B, the filter step uses step 1 and step 3 for non-leaf MBR optimizations, which are discussed in Section 4.1. The refinement step is the same as that in Configuration A.
- In Configuration C, the filter step uses all three steps for non-leaf MBR optimizations discussed in Sections 4.1. The refinement step is the same as that in Configuration A.
- In Configuration D, the filter step use not only the three steps for non-leaf MBR optimizations discussed in Section 4.1, but also step 1 and step 3 for leaf MBR optimizations discussed in Section 4.2. The refinement step is the same as that in Configuration A.
- In Configuration E, the filter step uses the three steps for non-leaf optimizations in Sections 4.1 as well as the three steps for leaf optimizations in Sections 4.2. The refinement step is also the same as that in Configuration A. This is the default configuration in Oracle Spatial.

Note that Configuration A can be used to simulate a spatial database system, where the R-tree index component and the computational geometry component are loosely coupled, so that the R-tree index component only uses the MBR of a given spatial object to examine the MBRs from the R-tree index on test spatial objects. In Figure 5, we report the total execution time for each of three complex spatial object data sets under the above five configurations. Note that because the query execution for the heterogeneous collections (i.e. US longest rivers) under Configuration A takes longer than 10 hours or 36,000s, it was terminated. It is clear that Configuration A is the most inefficient, as many false-positives have to be examined in the refinement step. The experimental results suggest that the spatial index component and the computational geometry component should be tightly integrated in spatial database systems. Although some approximations are used to obtain Hausdorff distances in both step 2 for non-leaf MBR optimizations and step 2 for leaf MBR optimizations, these approximations are relatively efficient.

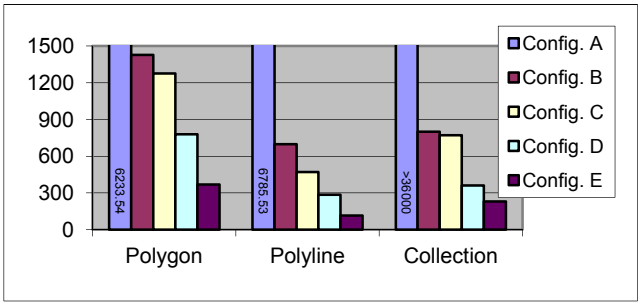

Figure 5. Execution time (in seconds) of within-distance queries under different configurations.

7.2 Comparison of Within-Distance Queries and Intersects-Buffer Queries

As we have described, within-distance queries can be translated to topological relationship *INTERSECTS* queries with a new spatial object, *BUFFER*(given spatial object, given distance). In Oracle Spatial, *SDO_ANYINTERACT* is the operator that is equivalent to Open Geospatial Consortium (OGC)'s *INTERSECTS* operation [27]. For example, the within-distance query in Section 7.1 can be rewritten as the following query with a buffer operation on the given spatial object.

```
SELECT COUNT(*)
FROM BLOCKS a
WHERE SDO_ANYINTERACT(a.geometry,
 SDO_GEOM.SDO_BUFFER(:given_spatial_object, 4,
 :tolerance, 'unit=km')) = 'TRUE';
```

In Section 7.2, we compare the within-distance queries with their equivalent intersects-buffer queries. We use all three complex

spatial object data sets to run both within-distance queries and intersects-buffer queries against the ABI point data set, with distance values ranging from 0.125km to 128km. We also pre-compute these buffer objects, and use them directly as given spatial objects to run intersects queries, so that expensive *BUFFER* operations are not taken into account.

Figures 6, 7 and 8 show experimental results for complex polygons, polylines and collections, respectively. Note that the "intersects" line in the figures is for intersects queries with pre-computed buffer objects. The results show that because the buffer operations can be expensive, within-distance queries can be faster than their equivalent intersects-buffer queries. However, if buffer objects have been pre-computed, the equivalent intersects queries can be slightly faster than within-distance queries. In some cases, it is possible that the equivalent intersects queries are much faster than within-distance queries as visible for the given distance = 128km in Figure 8. This occurs due to two reasons. Firstly, when the given distance is large, the buffer objects can be simpler than the original complex objects, so that distance computations described in Section 5 can be relatively more expensive than operations used in corresponding intersects queries. For example, when the given distance = 128km, the average number of vertices in 38 buffer objects for US rivers is only 5044. Secondly, since the Hausdorff distance calculations described in Section 4 use approximations, some MBRs cannot be determined fully within the given distance of given spatial objects early in the filter step. For example, the height of the R-tree index on the ABI point data set is 5. None of non-leaf MBRs at level = 4 can be determined to lie completely within 128km of the 38 longest rivers using Hausdorff distance approximations, but some non-leaf MBRs at level = 4 are actually completely within 128km of the 38 longest rivers.

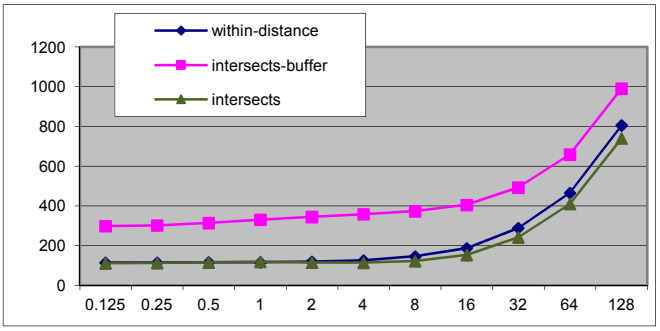

Figure 6. Execution time (in seconds) of within-distance queries vs. intersects-buffer and intersects queries with distances from 0.125km to 128km, given polygons.

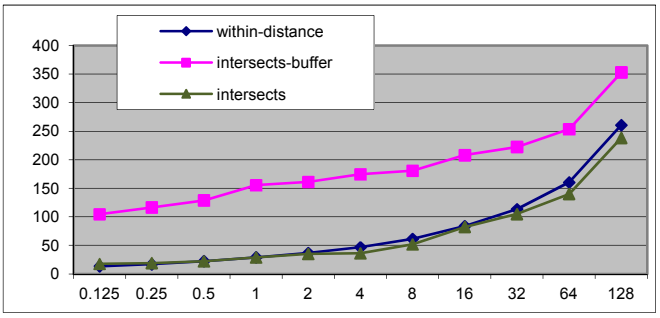

Figure 7. Execution time (in seconds) of within-distance queries vs. intersects-buffer and intersect queries with distances from 0.125km to 128km, given polylines.

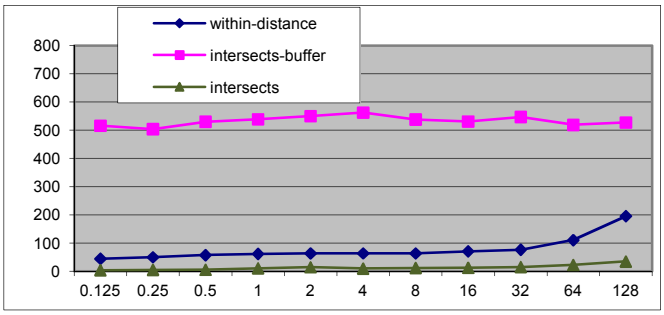

Figure 8. Execution time (in seconds) of within-distance queries vs. intersects-buffer and intersects queries with distances from 0.125km to 128km, given collections.

7.3 Convex Hull

In Section 7.3, we conduct experiments to compare the in-memory R-tree based convex hull algorithm described in Section 6.1 with the Graham scan algorithm [10, 2]. We use 38 longest rivers of the United States, because they are more complex than 50 US states, 1061 local regions and 239 US interstate highways so that it takes longer to complete the same query on average. For example, the average number of vertices in the 38 rivers is much higher than other spatial objects. In order to easily compare the original 2D Graham scan algorithm [10, 2], we also transform the geodetic spatial objects to projected spatial objects, which are flat in 2D Cartesian space. For the 38 US longest rivers, the total execution time of the Graham scan algorithm is 0.952s, while the total execution time of the in-memory R-tree based convex hull algorithm is 0.16s, which includes 0.083s for building in-memory R-trees. Note that as described in Section 5, several neighbor line segments are grouped into a single leaf entry, and the cost of building in-memory R-trees can be limited.

7.4 Maximum Distance and Diameter

In Section 7.4, the same projected data set of 38 US longest rivers in Section 7.3 is also used to compare two algorithms that compute maximum distance and diameter. The first algorithm uses the in-memory R-tree based convex hull algorithm before running the rotating caliper algorithm. The second algorithm is obtained by modifying Approach 2 of Section 5. For maximum distance computation, the total execution time of the first algorithm is 1.83s, which includes 0.16s for building convex hulls, while the total execution time of the second algorithm is 38.72s. For diameter calculation, the total execution time of the first algorithm is 0.23s, which also includes 0.16s for building convex hulls, while the total execution time of the second algorithm is 2.80s. This means that to determine the farthest neighbor pairs, the second algorithm is not as efficient as the first algorithm that applies the rotating caliper technique on convex hulls.

7.5 Minimum Bounding Circle

In Section 7.5, the same projected data set of 38 US longest rivers is used to compare the in-memory R-tree based MBC algorithm described in Section 6.3 with the randomized MBC algorithm [38, 2]. For the 38 US longest rivers, the total execution time of the randomized MBC algorithm is 0.132s on average. Note that because this is a randomized algorithm, there could be some rare runs in which it takes significantly longer. In contrast, the in-memory R-tree based MBC algorithm is deterministic, and it takes 0.098s, which also includes 0.083s for building in-memory R-trees. If the time of building in-memory R-trees is not taken into account, the total execution time of the in-memory R-tree based

MBC algorithm is 0.015s. Thus, when the in-memory R-trees are already built, the in-memory R-tree based MBC algorithm can be much faster than the randomized MBC algorithm.

8. CONCLUSIONS

In this paper, we discussed some in-memory R-tree based optimization techniques in Oracle Spatial to speed up distance queries for complex spatial objects. We also discussed distance-related problems, such as the maximum distance between complex spatial objects, and the diameter computation, the convex hull and the minimum bounding circle computation for a complex spatial object, can also be solved by using the in-memory R-tree structure. Finally, we have conducted experiments by utilizing real-world data sets to demonstrate that the performance of these distance and distance-related queries can be significantly improved by using the in-memory R-tree techniques.

9. REFERENCES

[1] Dana H. Ballard: Strip Trees: A Hierarchical Representation for Curves. *Commun. ACM (CACM)* 24(5):310-321 (1981)

[2] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars: *Computational Geometry: Algorithms and Applications*, Springer-Verlag, New York (2008)

[3] Thomas Brinkhoff, Holger Horn, Hans-Peter Kriegel, Ralf Schneider: A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems. *SSD* 1993: 357-376

[4] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger: Multi-Step Processing of Spatial Joins. *SIGMOD Conference* 1994: 197-208

[5] Antonio Corral, Yannis Manolopoulos, Yannis Theodoridis, Michael Vassilakopoulos: Closest Pair Queries in Spatial Databases. *SIGMOD Conference* 2000: 189-200

[6] Antonio Corral, Jesús Manuel Almendros-Jiménez: A performance comparison of distance-based query algorithms using R-trees in spatial databases. *Inf. Sci.* 177(11): 2207-2237 (2007)

[7] Edward P. F. Chan: Buffer Queries. *IEEE Trans. Knowl. Data Eng.* 15(4): 895-910 (2003)

[8] Tobias Emrich, Hans-Peter Kriegel, Peer Kröger, Matthias Renz, Andreas Züfle: Boosting spatial pruning: on optimal pruning of MBRs. *SIGMOD Conference* 2010: 39-50

[9] Brian S. Everitt, Sabine Landau, Morven Leese, Daniel Stahl: *Cluster Analysis*. Wiley, UK, 2011

[10] Ronald L. Graham: An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Inf. Process. Lett.* 1(4): 132-133 (1972)

[11] Güting, R.H., An Introduction to Spatial Database Systems. *VLDB Journal 3, 4* (1994) (Special Issue on Spatial Database Systems), 357-399.

[12] Antonin Guttman: R-Trees: A Dynamic Index Structure for Spatial Searching. *SIGMOD Conference* 1984: 47-57

[13] Gísli R. Hjaltason, Hanan Samet: Incremental Distance Join Algorithms for Spatial Databases. *SIGMOD Conference* 1998:237-248

[14] Gísli R. Hjaltason, Hanan Samet: Distance Browsing in Spatial Databases. *ACM Trans. Database Syst.* 24(2): 265-318 (1999)

[15] Ying Hu, Siva Ravada, Richard Anderson: Geodetic Point-In-Polygon Query Processing in Oracle Spatial. SSTD 2011: 297-312

[16] Ying Hu, Siva Ravada, Richard Anderson, Bhuvan Bamba: Topological relationship query processing for complex regions in Oracle Spatial. *SIGSPATIAL/GIS* 2012: 3-12

[17] Ying Hu, Siva Ravada, Richard Anderson, Bhuvan Bamba: Supporting Topological Relationship Queries for Complex Line and Collection Geometries in Oracle Spatial. *SIGSPATIAL/GIS* 2013: 94-103

[18] IBM DB2 Spatial Extender: http://www-03.ibm.com/software/products/en/db2spaext/

[19] JTS Topology Suite: http://tsusiatsoftware.net/jts/main.html

[20] Ravi Kanth Kothuri, Siva Ravada, Daniel Abugov: Quadtree and R-tree Indexes in Oracle Spatial: A Comparison Using GIS Data. *SIGMOD Conference* 2002: 546-557

[21] Suikai Li, Weiwei Sun, Renchu Song, Zhangqing Shan, Zheyong Chen, Xinyu Zhang: Quick Geo-Fencing Using Trajectory Partitioning and Boundary Simplification. *SIGSPATIAL/GIS* 2013: 590-593

[22] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos, Yannis Theodoridis: *R-Trees: Theory and Applications*. Springer, London, 2006

[23] Microsoft SQL Server Spatial Data: http://technet.microsoft.com/en-us/library/bb933790.aspx

[24] MongoDB Geospatial Indexes and Queries: http://docs.mongodb.org/manual/applications/geospatial-indexes/

[25] MySQL Spatial Extensions: http://dev.mysql.com/doc/refman/5.0/en/spatial-extensions.html

[26] Sarana Nutanong, Edwin H. Jacox, Hanan Samet: An Incremental Hausdorff Distance Calculation Algorithm. *PVLDB* 4(8):506-517 (2011)

[27] Open Geospatial Consortium Inc.: OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture

[28] Oracle Spatial and Graph Developer's Guide 12c Release 1 (12.1): http://docs.oracle.com/cd/E16655_01/appdev.121/e17896/toc.htm

[29] PostGIS: http://postgis.net/

[30] Sean Quinlan: Efficient Distance Computation Between Non-Convex Objects. *ICRA* 1994: 3324-3329

[31] Siva Ravada, Mohamed H. Ali, Jie Bao, Mohamed Sarwat: ACM SIGSPATIAL GIS Cup 2013. *SIGSPATIAL/GIS* 2013: 574-577

[32] Jon Rokne: An Easy Bounding Circle. *Graphics Gems II* 1991: 14-16.

[33] Nick Roussopoulos, Stephen Kelley, Frédéic Vincent: Nearest Neighbor Queries. *SIGMOD Conference* 1995: 71-79

[34] Hanan Samet: *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, San Francisco, CA, 2006

[35] Erich Schubert, Arthur Zimek, Hans-Peter Kriegel: Geodetic Distance Queries on R-Trees for Indexing Geographic Data. *SSTD* 2013: 146-164

[36] Godfried T. Toussaint: Solving geometric problems with the rotating calipers. *Proc. MELECON '83*, Athens

[37] Thaddeus Vincenty: Direct and Inverse Solutions of Geodesics on the Ellipsoid with application of nested equations". *Survey Review.* XXIII (176) : 88–93 (1975)

[38] Emo Welzl: Smallest enclosing disks (balls and ellipsoids). *New Results and New Trends in Computer Science, Lecture Notes in Computer Science* 555, Springer-Verlag, 359–370 (1991)

[39] Wikipedia: List of longest rivers of the United States (by main stem). http://en.wikipedia.org/wiki/List_of_longest_rivers_of_the_United_States_%28by_main_stem%29

[40] Tianyu Zhou, Hong Wei, Heng Zhang, Yin Wang, Yanmin Zhu, Haibing Guan: Point-Polygon Topological Relationship Query using Hierarchical Indices. *SIGSPATIAL/GIS* 2013: 582-585