Topological Relationship Query Processing for Complex Regions in Oracle Spatial

Ying Hu, Siva Ravada, Richard Anderson, Bhuvan Bamba
Oracle Spatial Technologies
One Oracle Drive
Nashua, NH 03062, USA
{ying.hu, siva.ravada, richard.anderson, bhuvan.bamba}@oracle.com

ABSTRACT
Although geographic information systems (GIS) and spatial database communities have extensively studied topological relationships for more than two decades, there is little literature describing how to efficiently implement them in GIS and spatial database systems. This is rather surprising considering that topological relationship queries are supported in many GIS and spatial database systems including IBM Informix Spatial and Geodetic DataBlades, ESRI SDE, Microsoft SQL server 2008, Oracle Spatial and PostGIS. In order to bridge this gap, we report our experience with implementing several optimization techniques in Oracle Spatial to speed up topological relationship query processing for query windows represented by complex regions (such as polygons or multi-polygons). Our experiments, utilizing real-world data sets, demonstrate that topological relationship query performance can be significantly improved using the proposed techniques.

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

General Terms
Algorithms, Management, Performance, Design, Experimentation.

Keywords
Topological relationships, 9-intersection model, in-memory R-tree, spatial query processing.

1. INTRODUCTION
Topological relationships have been studied by GIS and spatial database communities for more than two decades. Seminal works include the 9-intersection model [6, 8, 24] and the Region Connection Calculus (RCC) model [7]. These models can be used to define different topological relations such as *touch*, *overlap*, *inside*, and *contains*. However, as described in [24], literature on the actual implementation of these topological relationships, especially in the context of commercial GIS and spatial database systems, is rare. This is very surprising as many spatial database and GIS vendors [9, 12, 17, 20, 23] already support topological relationship queries. In order to resolve this discrepancy, we present in this paper our experience with implementing several optimization techniques in Oracle Spatial to speed up topological relationship query processing when complex regions are used as query windows.

ACM SIGSPATIAL GIS '12, November 6–9, 2012. Redondo Beach, CA, USA

Topological relationship queries with complex regions are very important. For example, an insurance company may need to search for properties within a flooded region. This is often called a point-in-polygon query [16]. Other examples include searching for land parcels touching a lake, highways crossing a city, and numerous similar spatial queries. These complex regions (like a flood zone, a lake or a city) are represented as polygons or multi-polygons. In the rest of this paper, we simply use the term "query polygon" for both the polygon and multi-polygon geometries that are used to run queries against test geometries. The performance of these topological relationship queries is one of the most important aspects of GIS and spatial database systems. To demonstrate the effectiveness of our optimization techniques for these topological relationship queries, we utilize real-world data sets to conduct an experimental study.

The rest of this paper is organized as follows: Section 2 reviews related work. Section 3 discusses some preliminary concepts including geometry validation and topological relationships in Oracle Spatial. Section 4 reviews the in-memory R-tree techniques. Sections 5 and 6 describe some optimization techniques used for minimal bounding rectangles (MBRs) and for different test geometries, respectively. Some extensions such as Open Geospatial Consortium (OGC) Model [19] are discussed in Section 7. Section 8 presents results of an experimental study using real-world data sets. Section 9 concludes the paper.

2. RELATED WORK
Sweep line-based line intersection algorithms are well known: a vertical line moves from left to right, intersecting the line segments in sequence [2, 21]. Several implementations of topological relationship query processing are based on the sweep line technique [5, 13]. However, in enterprise GIS and spatial database systems two factors can cause performance problems for these sweep line-based algorithms: (1) the number of test geometries; and (2) the complexity of test geometries and query geometries.

To address the problem of a large number of geometries, spatial indexes can be built to reduce the search scope. This is the two-step query processing method: (1) the filter step utilizes spatial indexes and returns a candidate set; (2) the refinement step is a test on exact geometries in the candidate set [3]. In commercial database systems two classes of spatial indexes, — Quad-tree indexes and R-tree indexes, are typically supported. For example,

Microsoft SQL server supports multiple-level Quad-tree indexes [10, 17] and Oracle Spatial supports both Quad-tree and R-tree indexes [15]. Since Quad-tree indexes typically need more fine-tuning to set an appropriate tessellation level, and they can only be used to index 2-dimensional (2D) non-geodetic geometries in Oracle Spatial, this paper will focus on the use of R-tree indexes. For R-tree indexes, topological relationship with MBRs has been well studied [22].

Previous works, which address the second problem of geometry complexity, include interior approximations [1, 14] and TR*-tree [4]. The interior approximations work [14] discusses tessellation of non-geodetic query polygons. If a point or an MBR is inside an interior tile, then this point or MBR is also inside the query polygon. 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. For geodetic geometries, Voronoi tessellations can also be used to speed up geodetic computations [12]. Recently, [11] discussed how an in-memory R-tree, built on top of boundary line segments of a polygon, can be used to speed up both geodetic and non-geodetic point-in-polygon queries. In PostGIS [23] and JTS [13], a similar concept of "prepared geometry" can be used for the INTERSECTS and CONTAINS operations. An in-memory interval tree can be used for point-in-polygon computations and an in-memory R-tree can be used for line-line intersections.

This paper extends the techniques presented in our previous work [11] to different types of test geometries, such as multi-point, line, multi-line, polygon and multi-polygon, and discusses the processing of different topological relationship queries for query polygons in Oracle Spatial. In addition, these techniques have been extended to other query geometries like query points, query lines, and collections of query points, query lines and query polygons in Oracle Spatial. However, we will focus primarily on query polygons in this paper, due to their extensive usage with respect to other query geometry types.

3. KEY CONCEPTS

In this section, we review some basic concepts that are used in the rest of this paper.

3.1 Two-dimensional Geometry Validation

A point is a 0-dimensional geometry, and its boundary is NULL. A multi-point is also a 0-dimensional geometry and its boundary is also NULL.

Both a line and a multi-line are a 1-dimensional geometry and they can be self-crossing. In Oracle Spatial, the boundary of a line or multi-line is determined by checking if the end points touch any point of this line or multi-line. Note that this is different from the OGC's "mod 2" union rule [19]: a point is in the boundary of a line or multi-line if it is in the boundaries of an odd number of elements of the line or multi-line. For example, in Figure 1, case (1) displays a single line; point B is in the boundary while point A is not in the boundary according to Oracle Spatial's rules. Case (2) displays a multi-line composed of two lines EF and CD, where line CD is closed and point E is the same as points C and D. According to OGC's "mod 2" union rule, point E, C, or D is in the boundary. But in Oracle Spatial, point E, C, or D is not included in the boundary. Note that both case (1) and case (2) are equivalent from the perspective of topological relationships, and in both cases, points A, and C, D or E respectively are considered as interior points in Oracle Spatial.

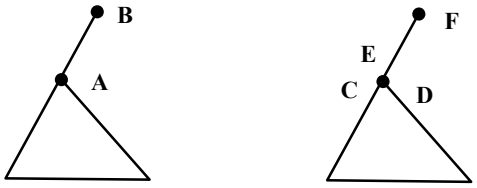

Figure 1. Determination of boundary of a line or a multi-line in Oracle Spatial.

Both a polygon and a multi-polygon are a 2-dimensional geometry. Restrictions imposed on both polygon and multi-polygon geometries include:

- They must not be self-crossing.
- They must be oriented correctly. That is, exterior ring boundaries must be oriented counter-clockwise and interior ring boundaries must be oriented clockwise.
- The interior of each polygon is connected.

These restrictions allow us to determine if a point or a line resides inside or outside a query polygon locally.

3.2 Topological Relationships

Although Oracle Spatial supports the OGC topological relationships including EQUALS, DISJOINT, INTERSECTS, TOUCHES, CROSSES, WITHIN, CONTAINS and OVERLAPS [19], we would prefer to use the original 9-intersection model in [6] because it separates the WITHIN operation into two: INSIDE and COVEREDBY; this makes it easier to explain the implementation details. Therefore, we will focus on the original 9-intersection model and postpone the discussion of OGC's topological relationships to Section 7.4.

The 9-intersection model can be described as a matrix, as shown in Figure 2. We use the following notion to indicate which bit is set. When boundary (*b*), interior (*i*), exterior (*e*) of test geometry B intersects boundary (*b*), interior (*i*), exterior (*e*) of query geometry A, *bit*_[*b*|*i*|*e*]_[*b*|*i*|*e*] (the first [*b*|*i*|*e*] is for test geometry B and the second [*b*|*i*|*e*] is for query geometry A) is set. For example, *bit_b_b*, *bit_b_e*, *bit_i_e*, *bit_e_b*, *bit_e_i* and *bit_e_e* are set in Figure 2.

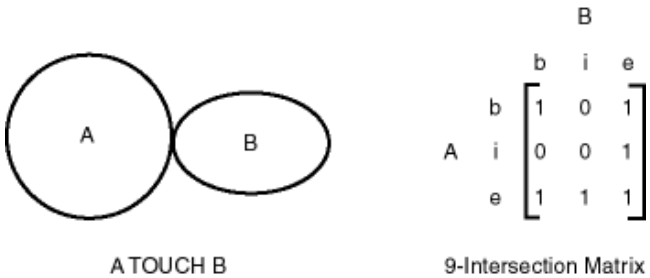

Figure 2: 9-intersection matrix.

In Oracle Spatial, two enhancements are added on the original 9-intersection model: 1) OVERLAPBDYDISJOINT and OVERLAPBDYINTERSECT are treated separately. This separation is useful to distinguish some cases as shown in Figure 3 and Figure 4. 2) ON is added for cases where a point, multi-point, line, or multi-line is completely on the boundary of a polygon. In this scenario, ON is a special case of TOUCH. The ON relationship is added because many Oracle Spatial users

would like to distinguish it from the TOUCH relationship. For example, some users may use "INSIDE + COVEREDBY + ON" (a composite topological relationship that will be discussed in Section 7.3) to retrieve all roads completely in the boundary and interior of a city. In addition, we also support the converse relationship of ON internally, which is also a special case of TOUCH.

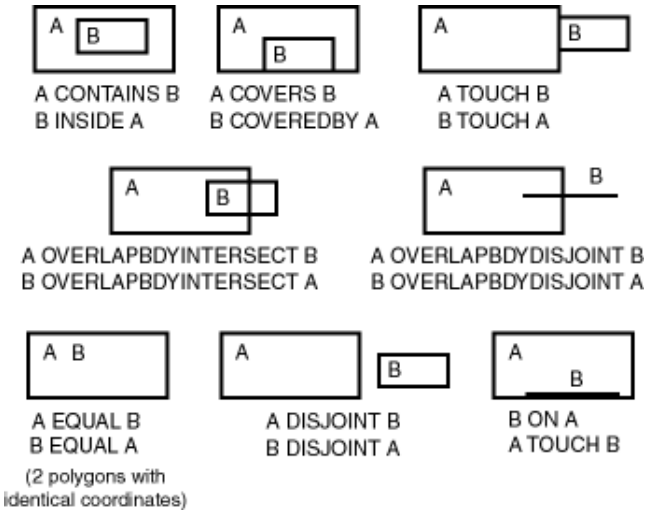

Figure 3. Topological relationships in Oracle Spatial.

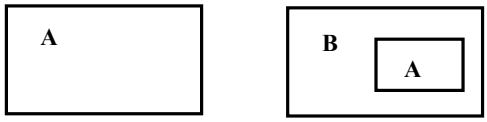

Figure 4. Another OVERLAPBDYDISJOINT case: query geometry A is a multi-polygon and test geometry B is a polygon.

The definition of topological relationships in Oracle Spatial and usage of the 9-intersection bits to identify or rule out existing topological relationships are as follows:

DISJOINT: The boundaries and interiors do not intersect. This is the same as DISJOINT in the OGC model. If any of *bit_b_b*, *bit_b_i*, *bit_i_b* and *bit_i_i* is set, it is not DISJOINT.

TOUCH: The boundaries intersect but the interiors do not intersect. This is the same as TOUCHES in the OGC model. If *bit_i_i* is set, it is not TOUCH.

OVERLAPBDYDISJOINT: The interior of one object intersects the boundary and interior of the other object, but the two boundaries do not intersect. If *bit_b_b* is set, it is not OVERLAPBDYDISJOINT.

OVERLAPBDYINTERSECT: The boundaries and interiors of the two objects intersect. If all *bit_b_b*, *bit_i_i*, *bit_i_e* and *bit_e_i* are set, it is OVERLAPBDYINTERSECT.

EQUAL: The two objects have the same boundary and interior. This is the same as EQUALS in the OGC model. If any of *bit_b_i*, *bit_b_e*, *bit_i_b*, *bit_i_e*, *bit_e_b* and *bit_e_i* is set, it is not EQUAL.

CONTAINS: The interior and boundary of one object is completely contained in the interior of the other object. If any of *bit_b_b* , *bit_b_i* , *bit_e_b* and *bit_e_i* is set, it is not CONTAINS.

COVERS: The interior of one object is completely contained in the interior or the boundary of the other object and their boundaries intersect. If any of *bit_e_b* and *bit_e_i* is set, it is not COVERS.

INSIDE: The opposite of CONTAINS. A INSIDE B implies B CONTAINS A. If any of *bit_b_b*, *bit_b_e*, *bit_i_b* and *bit_i_e* is set, it is not INSIDE.

COVEREDBY: The opposite of COVERS. A COVEREDBY B implies B COVERS A. If any of *bit_b_e* and *bit_i_e* is set, it is not COVEREDBY.

ON: The interior and boundary of one object is on the boundary of the other object. This is a special case of TOUCH. If any of *bit_b_i*, *bit_b_e*, *bit_i_i* and *bit_i_e* is set, it is not ON.

ANYINTERACT: The objects are non-disjoint. This is the same as INTERSECTS in the OGC model. If any of *bit_b_b*, *bit_b_i*, *bit_i_b* and *bit_i_i* is set, it is ANYINTERACT.

Note that *bit_e_e* is always set for all of the above topological relationships because we assume that no geometry can span the whole universal space.

4. IN-MEMORY R-TREE ON QUERY POLYGON

In [11], we described an in-memory R-tree built on top of boundary line segments of a query polygon. Unlike a TR*-tree that is built on top of trapezoids [4], we utilize an in-memory R-tree, built on top of boundary line segments of a query polygon, because we can easily extend this technique to geodetic polygons, internally represented as geocentric 3D surfaces in Oracle Spatial. Furthermore, we have extended this technique to include other query geometries like query points, query lines, and collections of query points, query lines, and query polygons. Therefore, in an in-memory R-tree, leaf entries can be made up of points or line segments (either boundary line segments of a query polygon, or just line segments of a query line). However, we will focus primarily on query polygons in this paper.

To simplify our discussion, we use non-geodetic query polygon examples in this paper, except in Section 7.1 where geodetic query polygons are briefly discussed, and Section 8. Figure 5 shows a query polygon comprising 14 line segments (shown in blue). Each line segment is used to obtain its respective MBR, which corresponds to an entry in a leaf node. An in-memory R-tree can be built on the 14 MBRs in Figure 5, marked with dashed lines; and four MBRs marked with dotted lines represent the leaf nodes of the in-memory R-tree. Note that some lines in these MBRs are both dashed and dotted. The root MBR is not shown in order to make the four MBRs for the leaf nodes more distinct. Note that because the in-memory R-tree is built once per query polygon, the cost of building it is amortized over a large number of test geometries.

Figure 5 also shows a green MBR (M), from a disk-based R-tree index, inside the query polygon. It is obvious that M's descendants will also be inside the query polygon.

To decide if an MBR is inside a query polygon, we use a two-step procedure:

(1) Search the in-memory R-tree to determine if there is any intersection between the MBR and line segments of the query polygon. Note that, although the MBR (M) intersects with the entry MBR (N) in Figure 5, it does not intersect

with any line segments in the query polygon. If there is any intersection, child MBRs of the MBR (M) will be fetched from disk or buffer cache for further processing recursively. Otherwise, go to the next step.

(2) Choose the right-top corner point of the MBR (M) and draw a horizontal line segment L (shown in red) from this point to cross the maximum X value of the query polygon. Then the line segment L can be used to search the in-memory R-tree again and get how many line segments of the query polygon intersect with it. If the number of intersections is odd it is guaranteed that the corner point is inside the query polygon, and thus the whole MBR and its descendants are also completely inside the query polygon. Otherwise, the MBR and its descendants can be skipped.

Note that in step (2), we can also use the R-tree Nearest Neighbor (NN) search method to identify the point on the query polygon that is the closest to the right-top corner point of the MBR (M) and determine if the right-top corner point of the MBR (M) is inside the query polygon. More details can be found in [11]. The above two steps are also applied to a point: the first step deals with the case of point-on-polygon, while the second step deals with the case of point-in-polygon.

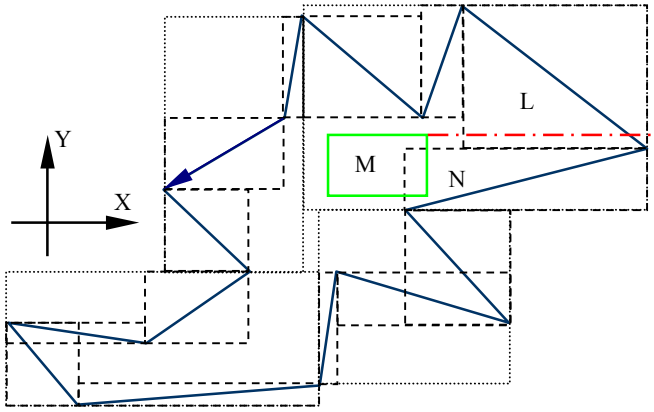

Figure 5. An in-memory R-tree is built on a query polygon.

In practice, however, we limit the memory usage associated with the in-memory R-tree structure by grouping several line segments into a single entry to construct the in-memory R-tree structure. For example, assume that a query polygon has 50,000 line segments. If we can group 10 continuous line segments into one entry, there will be only 5,000 entries instead of 50,000 entries originally.

5. MBR OPTIMIZATIONS

In the filter step of the two-step query processing [3], topological relationships with MBRs have been well studied for R-trees [22]. This section will focus on some new optimization techniques added on top of the previous work [22]. These new techniques are based on the in-memory R-tree on query polygon and they apply to all topological relationships except CONTAINS, COVERS, and EQUAL. For example, if the MBR of a test geometry already contains the MBR of a query polygon, the MBR of the test geometry will also contain the entire query polygon. Thus, CONTAINS, COVERS, and EQUAL do not require these new MBR optimizations.

5.1 Non-Leaf MBR Optimizations

The in-memory R-tree for the query polygon allows us to quickly determine if a non-leaf MBR is inside, intersects or is outside the query polygon. The following optimizations can be used to further determine the topological relationships.

- For ANYINTERACT and INSIDE, if a non-leaf MBR is inside the query polygon, its descendant test geometries can be returned without further checking.
- For all topological relationships except DISJOINT, if a non-leaf MBR is outside the query polygon, its descendant test geometries can be discarded.
- For COVEREDBY, ON, OVERLAPBDYDISJOINT, OVERLAPBDYINTERSECT and TOUCH, if a non-leaf MBR does not intersect the query polygon ring, its descendant test geometries can be discarded.

5.2 Leaf MBR Optimizations

For a leaf MBR, we can quickly determine if it is inside, intersects, or is outside the query polygon. We can then use the following optimizations similar to those previously presented in Section 5.1.

- For ANYINTERACT and INSIDE, if a leaf MBR is inside the query polygon, its test geometry can be returned without further checking.
- For all topological relationships except DISJOINT, if a leaf MBR is outside the query polygon, its test geometry can be discarded.
- For COVEREDBY, ON, OVERLAPBDYDISJOINT, OVERLAPBDYINTERSECT and TOUCH, if a leaf MBR does not intersect the query polygon ring, its test geometry can be discarded.

When a leaf MBR intersects a query polygon, we will perform additional optimizations for ANYINTERACT, COVEREDBY, INSIDE, ON and TOUCH by checking if each of the four boundary edges of the leaf MBR is inside or outside the query polygon:

- For COVEREDBY, INSIDE and ON, if any of the four boundary edges is outside the query polygon, its test geometry can be discarded.
- For ON and TOUCH, if any of the four boundary edges is inside the query polygon, its test geometry can be discarded.
- For ANYINTERACT, if any of the four boundary edges is inside the query polygon, its test geometry can be returned.

Figure 6 explains the above three optimizations: 1) The test geometry enclosed by leaf MBR (M') cannot be COVERED, INSIDE or ON the query polygon because the right boundary edge is totally outside the query polygon. 2) The test geometry enclosed by leaf MBR (M') cannot TOUCH the query polygon because the left boundary edge is totally inside the query polygon. 3) For ANYINTERACT, we can return the test geometry enclosed by leaf MBR (M') without further computation because the left boundary edge is totally inside the query polygon and the test geometry must intersect the query polygon. Furthermore, in Figure 6, if the test geometry enclosed by leaf MBR (M') is a single polygon, OVERLAPBDYINTERSECT will hold true. If the test geometry is not a single polygon, then either OVERLAPBDYINTERSECT or OVERLAPBDYDISJOINT will hold true.

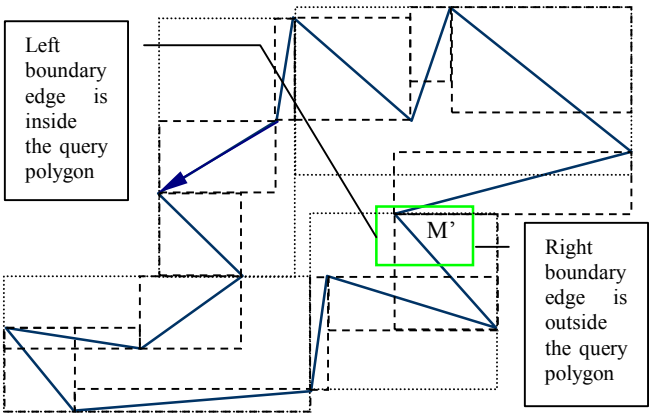

Figure 6. A boundary edge of a leaf MBR can be inside or outside a query polygon.

6. TEST GEOMETRY OPTIMIZATIONS

The filter step, including the MBR optimizations in Section 5, discards some test geometries while some are returned without further computation. The remaining test geometries are candidates for further processing. In this section, we will discuss a method to determine the topological relationship between a candidate test geometry and a query polygon.

In Section 3.2 we discussed the 9-intersection bits used to identify or rule out different topological relationships. For example, when *bit_i_i* is set, we can immediately rule out the TOUCH relationship. In this section, we discuss the method for determining the 9-intersection bits. Note that we assume that test geometries are valid, and we do not sort the coordinates in test geometries. In other words, we use the original sequence of points in a test geometry to set the 9-intersection bits sequentially, locally and independently, and we do not need to carry much state information. This is different from sweep line-based algorithms where coordinates in both query geometries and test geometries have to be sorted and an event queue has to be maintained.

Furthermore, it is possible that some coordinates or points in a test geometry are not accessed at all. For example, for a TOUCH operation, when a point or a line segment from a test geometry is already determined to be inside a query polygon, or *bit_i_i* is set, we can stop processing the rest of coordinates or points in this test geometry and discard it.

We will split this section into three subsections based on the test geometry types and explain different optimization techniques.

6.1 Test Point Geometry Optimizations

When the test geometry is a single point, there are only three topological relationships to a query polygon: INSIDE, ON (a special case of TOUCH) and DISJOINT, which correspond to *bit_i_i*, *bit_i_b* and *bit_i_e* being set, respectively. As these can be done by the techniques previously discussed in Section 4, we will skip them.

For a multi-point test geometry, we use the same techniques described in Section 4 to determine if each point is inside, on or outside a query polygon (*bit_i_i*, *bit_i_b* and *bit_i_e* is set correspondingly).

Note that 1) as a point or multi-point geometry have no boundary, *bit_b_i*, *bit_b_b* and *bit_b_e* are not set; 2) as the exterior of a point or multi-point geometry intersects boundary, interior and exterior of a query polygon, *bit_e_i*, *bit_e_b* and *bit_e_e* can be set for all cases. So the only difference among the different cases is how we set *bit_i_i*, *bit_i_b* and *bit_i_e*. In other words, setting *bit_i_i*, *bit_i_b* and *bit_i_e* allows us to determine the topological relationship. For instance,

- If only *bit_i_i* is set, the topological relationship is INSIDE.
- If only *bit_i_b* is set, the topological relationship is ON. Note that ON is a special case of TOUCH. So for a TOUCH query, a test geometry, whose topological relationship is ON, is also returned.
- If only *bit_i_e* is set, the topological relationship is DISJOINT.
- If only *bit_i_b* and *bit_i_i* are set, the topological relationship is COVEREDBY.
- If only *bit_i_b* and *bit_i_e* are set, the topological relationship is TOUCH.
- If *bit_i_i* and *bit_i_e* are set, the topological relationship is OVERLAPBDYDISJOINT, regardless of *bit_i_b* being set or not.
- If any of *bit_i_b* and *bit_i_i* is set, the topological relationship is ANYINTERACT.

6.2 Test Line Geometry Optimizations

For a query polygon, when the test geometry is a line or multi-line, the topological relationships CONTAIN, COVERS, and EQUAL cannot hold true, but all other topological relationships are possible. First, we determine which end points are in the boundary as described in Section 3.1. If some end points are in the boundary, we check if they are inside, on or outside the query polygon, which corresponds to setting *bit_b_i*, *bit_b_b* and *bit_b_e* respectively. In addition, we can make the following inferences: 1) when *bit_b_i* is set, *bit_i_i* can also be set; 2) when *bit_b_b* is set, *bit_e_b* can also be set; 3) when *bit_b_e* is set, *bit_i_e* can also be set. For the remaining end points that are not in the boundary, we also check if they are inside, on or outside the query polygon, which corresponds to setting *bit_i_i*, *bit_i_b* and *bit_i_e*. This can be accomplished in a manner similar to the discussion in Section 4 because the end points are still points.

We can now focus on the manner in which the other portion of line or multi-line (i.e. except its end points) intersects the query polygon, which also corresponds to how the bits *bit_i_i*, *bit_i_b* and *bit_i_e* are set. As a line or multi-line can contain many line segments, we describe how the three bits are set for each of the line segments. Figure 7 shows such a line segment SE. This function is performed in two major steps:

(1) Use the line segment SE to search the in-memory R-tree to determine whether or not there is any intersection or collinearity between the line segment SE and boundary line segments of the query polygon. For example, in Figure 7, there are 3 intersection points A, B and C, and line segment AB is collinear with the boundary of the query polygon. We can view the line segment SE as being split into four newly generated line segments: SA, AB, BC and CE. Hence, *bit_i_b* can be set if there is any intersection or collinearity.

(2) For each newly generated line segment that is not collinear, namely SA, BC and CE in Figure 7, we use an intersection point to search the in-memory R-tree to determine if this newly generated line segment is either inside or outside the query polygon. For example, for line segment BC, either B or C can be used, as both are intersection points. The question arises as to why use an intersection point to search the in-memory R-tree. This is done because a line segment can intersect the boundary of a polygon or multi-polygon at the same intersection point many times and we need to scan all intersecting boundary edges and return the closest one. For example, in Figure 8, the query region is a multi-polygon with three exterior rings and one interior ring meeting at point D. Line segment S'E' also intersects the multi-polygon at point D. After scanning all intersecting boundary edges and measuring the angle of intersection, we determine that line segment DX is the closest to line segment S'D and line segment YD is the closest to line segment DE'. Note that, in Oracle Spatial, polygons are oriented: Exterior ring boundaries must be oriented counterclockwise and interior ring boundaries must be oriented clockwise. Therefore, we can easily determine that newly generated line segment S'D is inside and DE' is outside the query polygon, which corresponds to setting *bit_i_i* and *bit_i_e*.

When the test geometry is a line or multi-line and query geometry is a polygon or multi-polygon, setting *bit_b_b*, *bit_b_i*, *bit_b_e*, *bit_i_b*, *bit_i_i* and *bit_i_e* is sufficient and the topological relationship can be determined in a similar fashion as for point data geometry. For example, if only *bit_b_i* and *bit_i_i*, or only *bit_i_i* of the above six bits are set, topological relationship INSIDE holds true. So we don't need *bit_e_b*, *bit_e_i* and *bit_e_e*, although *bit_e_i* and *bit_e_e* can always be set.

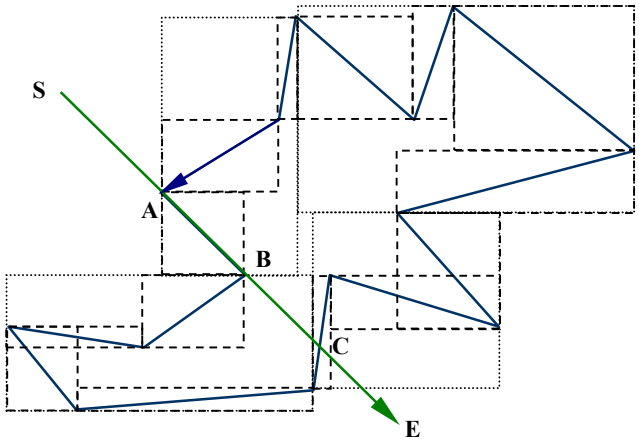

Figure 7. Line segment intersecting a query polygon.

6.3 Test Polygon Geometry Optimizations

For a query polygon, when the test geometry is either a polygon or multi-polygon, all topological relationships except ON are possible, as shown in Figures 3 and 4. Similar to the procedure for a test line geometry, we take the first point in each polygon ring of a test polygon geometry and determine if this point is inside, on or outside the query polygon, which corresponds to setting *bit_b_i*, *bit_b_b* or *bit_b_e* respectively. In addition, we can make the following inferences: 1) when *bit_b_i* is set, *bit_i_i* and *bit_e_i* can also set; 2) when *bit_b_e* is set, *bit_i_e* can also be set. In other words, if a boundary point of a test polygon is inside a query polygon, there must be an exterior point of the test polygon and an interior point of the test polygon inside the query polygon; if a boundary point of a test polygon is outside a query polygon, there must be an interior point of the test polygon outside the query polygon. This step is necessary because it is possible that polygon rings in both a test polygon and a query polygon do not intersect each other.

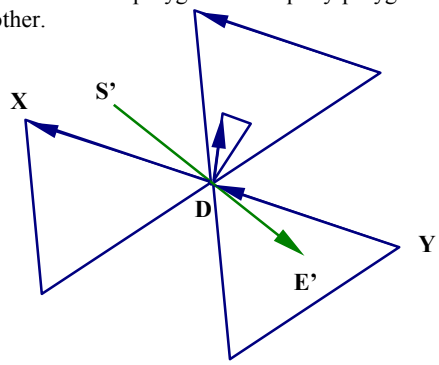

Figure 8. A line segment intersects a query polygon at the same point more than once.

We can now focus on the case where polygon rings in both a test polygon and a query polygon intersect each other. This is also similar to what we have done for test line geometries. Assume that line segment SE in Figure 7 is from a test polygon, and its interior is on its left side while its exterior is on its right side, as exterior ring boundaries must be oriented counterclockwise and interior ring boundaries must be oriented clockwise. The 9-intersection bit determination is also performed in two major steps:

(1) Use the line segment SE to search the in-memory R-tree of the query polygon to determine if there is any intersection or collinearity between the line segment SE and line segments of the query polygon. If there is such a case, *bit_b_b* can be set. The intersection points can split a line segment into several newly generated line segments. When there is a collinear case, we can determine if the two collinear line segments have the same direction. For example, in Figure 7 line segment AB is collinear with the boundary of the query polygon and they also have the same direction. If the line segments are in the same direction, *bit_b_b* and *bit_i_i* can be set. If they are not in the same direction, then *bit_b_b*, *bit_i_e*, and *bit_e_i* can be set.

(2) For each newly generated line segment that is not collinear, we use an intersection point to search the in-memory R-tree of the query polygon to determine if the newly generated line segment is inside or outside the query polygon. For example, in Figure 7, when line segment SA is found to be outside the query polygon, *bit_b_e* and *bit_i_e* can be set; when line segment BC is found to be inside the query polygon, *bit_b_i*, *bit_i_i* and *bit_e_i* can be set. This is similar to what we have done for test line geometries. In addition, we can set *bit_i_b*, *bit_i_i* and *bit_i_e* under certain conditions. For example, in Figure 7, at the intersection points A or C, we can set *bit_i_b*, *bit_i_i* and *bit_i_e* as follows.

When a query polygon ring is to the left of a test polygon ring, we cannot guarantee that *bit_i_b*, *bit_i_i* and *bit_i_e* will be set. For example, for line segment S"E" of a test polygon in Figure 9, we find that line segments WF and FZ of a query polygon are to the left of line segment S"E". At the intersection point F, we can set *bit_b_e* and *bit_i_e*, but we cannot set *bit_i_b* and *bit_i_i* because line segments WF and FZ are not necessarily inside the test

polygon. For example, if there is an interior ring comprising line segments PF and FQ in Figure 9, line segments WF and FZ will be outside the test polygon.

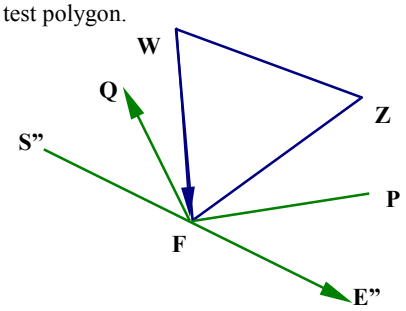

Figure 9. How *bit_i_b* can be set.

The question arises as to how *bit_i_b* and other bits can be set without considering the presence of another ring, like line segments PF and FQ in Figure 9. We rely on the following theorem.

Theorem 1: For two valid polygons P_1 and P_2, if P_1's polygon ring (PR_1) intersects P_2's polygon ring (PR_2) at point I, consider two line segments HI and IJ on PR_1, where one of HI and IJ is completely inside PR_2 while the other is not completely inside PR_2 (i.e. either completely outside PR_2 or on PR_2), then one boundary line segment of P_1 must be inside P_2.

Proof Sketch: For valid polygons, the interior of each polygon is connected. If we assume that there is no boundary line segment of P_1 inside P_2 under the above conditions (i.e. one of HI and IJ on PR_1 is completely inside PR_2 while the other is not completely inside PR_2), we will reach a contradiction - the interior of a polygon is not connected.

By using Theorem 1, we can set *bit_i_b*, *bit_i_i* and *bit_i_e* in many cases as discussed for the intersection points A and C in Figure 7. If we assume there is no interior ring (i.e. line segments PF and FQ) in Figure 9, how do we set *bit_i_b* and *bit_i_i* in this case? Or if we assume there is such interior ring (i.e. line segments PF and FQ) in Figure 9, how do we set *bit_e_b* and *bit_e_i* instead of *bit_i_b* and *bit_i_i*?

Before working with each line segment of a test polygon, we consider its first line segment from each ring of a query polygon, and insert this line segment and its associated data structure into an array. For each line segment of a test polygon, after we finish the two steps as described above, we determine if this line segment (of the test polygon) is closer to each line segment's first point in the array (for the query polygon). If it intersects the first point, then we determine how close the intersecting line segments are. For example, in Figure 9, assume that in the query polygon WFZ the first line segment is FZ and the first point is F. When we are working with line segment S"E" of the test polygon, we find it intersects point F. Then we measure the angle between line segments FZ and FE" and the angle between line segments FZ and FS", to determine that FE" is closer to FZ. Later when we are working with line segment PF, we find it also intersects point F. Then we measure the angle between line segments FZ and FP and obtain that FP is closer to FZ.

Once all line segments of the test polygon are scanned, we know that the following cases are possible for each line segment in the array (for the query polygon).

A) The line segment is collinear with a line segment of the test polygon. Since this is a collinearity case, all necessary bits are already set in the step (1), as previously discussed.

B) If the first point of the line segment is intersected by one or more line segments in the test polygon, we determine which line segment of the test polygon is the closest to the line segment of the query polygon. In addition, we can also determine if this *open* line segment of the query polygon is inside or outside the test polygon. By considering the *open* line segment we can exclude intersection points like point F in Figure 9. If it is inside the test polygon, *bit_i_b*, *bit_i_i* and *bit_i_e* can be set. If it is outside the test polygon, *bit_e_b*, and *bit_e_i* can be set. For example, in Figure 9, the first point F is intersected by line segments S"E", PF and FQ, and the open line segment FZ (i.e. without point F) is outside the test polygon, as it is to the right of line segment PF which is the closest to line segment FZ.

C) The first point of the line segment is either completely outside or completely inside the test polygon. For example, in Figure 4, the query polygon A has two rings; one ring is outside the test polygon B and *bit_e_b* and *bit_e_i* can be set, while the other ring is inside the test polygon B and *bit_i_b*, *bit_i_i* and *bit_i_e* can be set.

Therefore, for a test polygon, the 9-intersection bits can also be set sequentially, locally and independently. If any trigger bit for a certain topological relationship (e.g. *bit_i_i* for TOUCH and *bit_b_b*, *bit_b_i*, *bit_i_b* or *bit_i_i* for ANYINTERACT) is set, the test polygon can be discarded or returned.

7. EXTENSIONS

In this section, we briefly discuss a few extensions to the above optimization techniques.

7.1 Supporting Geodetic Geometries

Oracle Spatial converts geodetic coordinates to 3D Earth-centered coordinates and builds 3D minimal bounding boxes (MBBs) on them. Consequently, while 2D geodetic geometries have two dimensions (longitude, latitude) associated with them, any R-tree indexes built on them are 3D (Geocentric 3D) in Oracle Spatial. Some similarities and differences when processing geodetic and non-geodetic queries can be found in [11]. This subsection provides an example of extending our optimization techniques to geodetic geometries. For instance, we discuss some leaf MBR optimizations in Figure 6. Now MBRs become 3D MBBs. We will determine if each of the six boundary faces in a leaf MBB is inside, intersects or is outside the query polygon. For COVEREDBY, INSIDE and ON, if any boundary face is outside the query polygon, its test geometry can be discarded.

7.2 Supporting Arc Geometries

In Oracle Spatial, circular arcs are supported for non-geodetic geometries. Both test geometries and query geometries may contain arcs as a component of line or polygon geometry types. For query polygons, we need to insert arcs into the in-memory R-tree. This can be easily done, as R-tree is versatile for approximating different shaped geometries. But the arc-arc and arc-line intersections are more complex than a line-line intersection. For example, there can be two possible intersection points for arc-arc and arc-line intersections. Hence, the optimization techniques need to be extended to consider arcs. An arc will be split into several arcs at intersection points and each of the newly generated arcs is used to determine if it is inside, outside or on the query polygon. In this subsection, we focus on a method to determine which arc or line is closest to another arc or line. For each newly generated arc, we first obtain the tangent line

from the intersection point. The angle between this tangent line and another tangent line of an arc or a line can be measured and used for comparison. However, if the tangent line is collinear with another tangent line of an arc or a line, then we can use the radius as follows. In Figure 10, two arcs and one line are tangential at the same intersection point. In order to measure how close they are, we consider the parameter *alpha* as follows: *alpha* = {1/radius, if it is an arc and the arc center is to the left of the arc; 0, if it is a line; -1/radius, if it is an arc and the arc center is to the right of the arc}. By considering the difference between the alpha values, we can decide which pair is the closest when the arcs and lines are tangential at a point. Hence, we can also determine if an arc is inside, outside or on the query polygon.

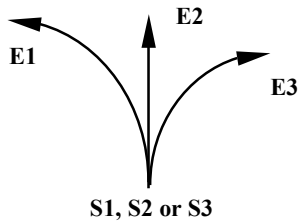

Figure 10. Two arcs and one line are tangential at the same point (S1, S2 or S3).

7.3 Composite Topological Relationships

Composite topological relationships can be formed by combining multiple relationships with the logical OR operator. For example, "INSIDE + COVEREDBY" is widely used in Oracle Spatial. As these composite topological relationships are OR based, we need to perform some adjustments.

Consider "INSIDE + COVEREDBY" as an example. If we have already determined that a non-leaf MBR is INSIDE a query polygon, its descendant test geometries can also be returned for "INSIDE + COVEREDBY". But if we have already determined that a non-leaf MBR cannot be COVEREDDBY a query polygon, its descendant test geometries will not be discarded for "INSIDE + COVEREDBY" relationship. Furthermore, "INSIDE + COVEREDBY" can be quickly ruled out if any of *bit_b_e* and *bit_i_e* is set for a test geometry. This is the same as COVERED described in Section 3.2 and it also applies for the relationship "INSIDE + COVEREDBY + EQUAL". For "OVERLAPBDYDISJOINT + OVERLAPBDYINTERSECT" relationship, we can verify that *bit_i_i*, *bit_i_e* and *bit_e_i* are all set. If they are set, "OVERLAPBDYDISJOINT + OVERLAPBDYINTERSECT" will hold true. In addition, for the case shown in Figure 6, "OVERLAPBDYDISJOINT + OVERLAPBDYINTERSECT" will hold true without retrieving the test geometry.

7.4 OGC Model

As EQUALS, DISJOINT, TOUCHES, and INTERSECTS operators in OGC model can be mapped to EQUAL, DISJOINT, TOUCH, and ANYINTERACT in Oracle Spatial, we just need to discuss WITHIN, CONTAINS, CROSSES, and OVERLAPS.

Based on the OGC definition of WITHIN[1] [19], we can translate OGC's WITHIN to "INSIDE + COVEREDBY + EQUAL", and translate OGC's CONTAINS to "CONTAINS + COVERS + EQUAL". So these can be handled as discussed in Section 7.3.

[1] The OGC's WITHIN is defined as $(a \cap b) \wedge (I(a) \cap E(b) = \phi)$.

For OGC's CROSSES with a query polygon, a test geometry can be a multi-point, line or multi-line. Thus, OGC's CROSSES can be translated to "OVERLAPBDYDISJOINT + OVERLAPBDYINTERSECT", when a test geometry is a multi-point, line or multi-line.

For OGC's OVERLAPS with a query polygon, a test geometry can be a polygon or multi-polygon. Thus, OGC's OVERLAPS can be also translated to "OVERLAPBDYDISJOINT + OVERLAPBDYINTERSECT", when a test geometry is a polygon or multi-polygon.

Thus OGC's CROSSES and OVERLAPS with a query polygon are also handled as discussed in Section 7.3.

7.5 WITHIN_DISTANCE

Although it is not a topological relationship operation, WITHIN_DISTANCE is widely used and can be translated to ANYINTERACT with a new query polygon, BUFFER(query polygon, distance). In Oracle Spatial, WITHIN_DISTANCE is not handled this way because BUFFER(query polygon, distance) computation can be costly. In fact, WITHIN_DISTANCE operation is performed in an approximately reverse manner. The in-memory R-tree is still built on top of boundary line segments of the original query polygon. While performing the MBR optimizations, we can enlarge an MBR by the distance and use the new MBR to search the in-memory R-tree of the query polygon. For test geometries, we split a complex test geometry into simple primitives, point, line and arc, and use these primitives to search the in-memory R-tree of the query polygon. If any primitive (point, line, or arc) of a test geometry is already within the distance of a query polygon, we can stop and the rest of the test geometry need not be accessed at all.

The query polygon's in-memory R-tree can also be used for other distance-related queries such as nearest neighbor (NN) queries, and for computing maximum distance between a query polygon and a test geometry, which has been shown to be infeasible for the sweep line-based algorithm in [5]. Further details are beyond the scope of this paper.

8. EXPERIMENTS

In this section, we discuss experiments using real-world data sets. To emphasize that the in-memory R-tree is used not only in the refinement step, but also in the filter step, we compare results from the following three configurations:

- In Configuration (A), the filter step only uses MBRs (or MBBs for geodetic geometries) and the refinement step uses the techniques discussed in Section 6.
- In Configuration (B), the filter step also uses non-leaf MBR optimization techniques discussed in Section 5.1. The refinement step is the same as that in Configuration (A).
- In Configuration (C), the filter step uses both non-leaf and leaf MBR optimization techniques, discussed in Section 5.1 and Section 5.2 respectively. The refinement step is the same as that in Configuration (A).

We use 50 US states and 1061 local regions including "Manhattan NY" as query polygons. The 50 US states have an average of 1,755 line segments with "Alaska" containing the most line segments (18,603). The 1061 local regions have an average of 520 line segments with "Long Island" containing the most line

segments (6,915). 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.

To limit the memory usage associated with the in-memory R-tree, we use the default number of leaf entries (4096) in our experiments. In other words, even if a query polygon has 20,000 line segments, we will group continuous line segments into 4096 leaf entries. Experiments in [11] have shown that results with lower limits (such as 4,096 entries) are almost as good as those without a limit.

We split this section into two subsections based on their geometry types – test lines and test polygons, since experiments for test points can be found in [11].

8.1 Test Line Geometry Experiments

For test lines, we use two line data sets from NAVTEQ [18]: one is the HIGHWAY data comprising about 1 million highway segments in North America and the other is the EDGE data containing 66 million road segments in North America. Both data sets have geodetic coordinates (longitude, latitude) in the World Geodetic System (WGS 84).

For the HIGHWAY data, we use 50 US states and 1061 local regions as query polygons to retrieve test lines with the following topological relationships: "ANYINTERACT", "INSIDE + COVEREDBY", "TOUCH" and "OVERLAPBDYDISJOINT + OVERLAPBDYINTERSECT". In Figure 11, we report the total execution time (in seconds) for these geodetic query polygons under the three different configurations (A), (B), and (C). It is clear that the in-memory R-tree is needed not only in the refinement step, but also in the filter step.

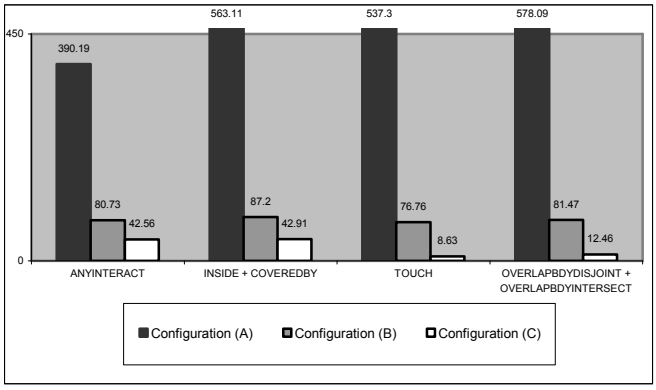

Figure 11. HIGHWAY in geodetic coordinates.

For the EDGE data set, we randomly select six local regions as query polygons to retrieve test lines with the same four topological relationships. The query performance results (in seconds) are shown in Figure 12.

We also transform the above geodetic data set to projected (or non-geodetic) geometries to show the non-geodetic query performance in Figure 13 and Figure 14. In general, non-geodetic computations have less overhead than geodetic computations. For example, geodetic coordinates need to be converted to 3D Earth-centered coordinates and computations on these 3D coordinates are also more complex than 2D projected coordinates. The geodetic query performance results (in seconds) shown in Figure 11 and Figure 12 are comparable to the non-geodetic query performance results shown in Figure 13 and Figure 14.

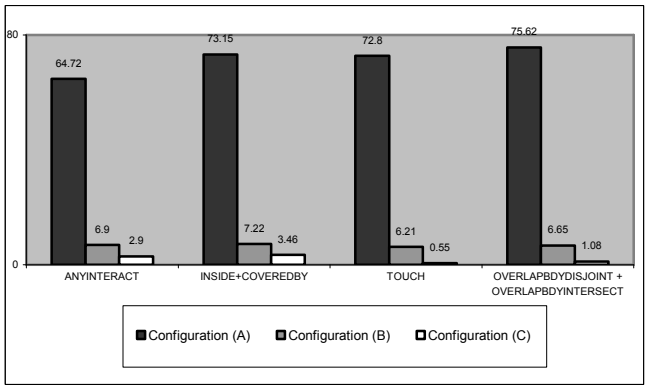

Figure 12. EDGE in geodetic coordinates.

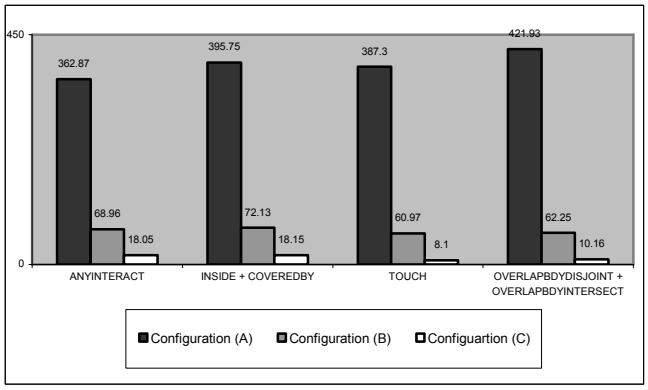

Figure 13. HIGHWAY in non-geodetic coordinates.

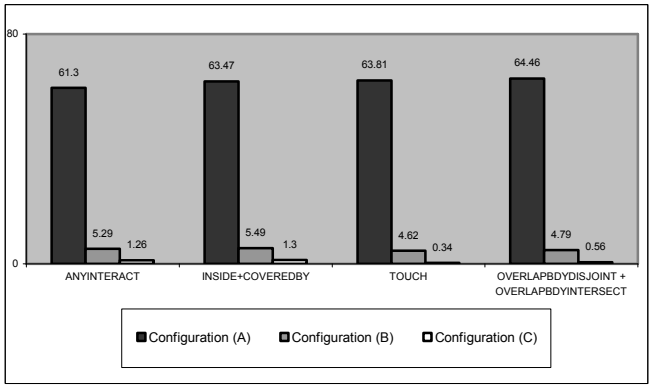

Figure 14. EDGE in non-geodetic coordinates.

8.2 Test Polygon Geometry Experiments

For test polygons, we use one polygon data set from NAVTEQ [18]: the BLOCKS data containing around 11 million US census blocks. The same 50 US states and 1061 local regions are used as query polygons to retrieve test polygons with the same four topological relationships. The geodetic and non-geodetic query performance results (in seconds) are shown in Figure 15 and Figure 16 respectively. Again, it is clear that the in-memory R-tree is needed not only in the refinement step, but also in the filter step. As shown in Figure 15, for the TOUCH relationship, the non-leaf MBR optimizations in the filter step reduce execution time by 87%. Usage of the leaf MBR optimizations as well results in further improvement in execution time.

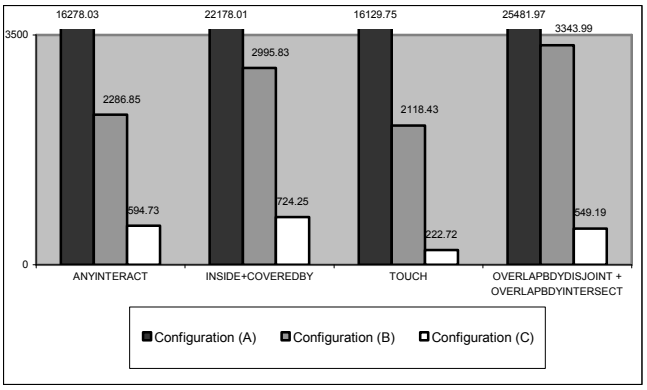

Figure 15. BLOCKS in geodetic coordinates.

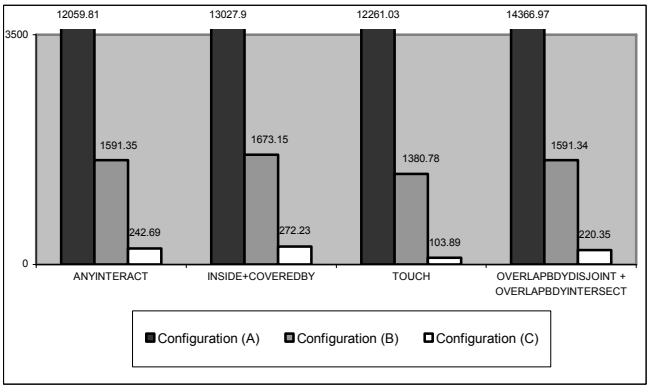

Figure 16. BLOCKS in non-geodetic coordinates.

9. CONCLUSIONS

This paper presents optimization techniques implemented in Oracle Spatial to speed up topological relationship query processing for complex regions. These optimization techniques improve topological relationship query performance significantly without additional user tuning. In future, we plan to extend these optimization techniques to other spatial queries, including distance-based spatial queries, and long-running spatial operations like UNION and INTERSECTION.

10. REFERENCES

[1] Wael M. Badawy, Walid G. Aref: On Local Heuristics to Speed Up Polygon-Polygon Intersection Tests. ACM-GIS 1999: 97-102

[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] Edward P. F. Chan, Jimmy N. H. Ng: A General and Efficient Implementation of Geometric Operators and Predicates. SSD 1997: 69-93

[6] Eliseo Clementini, Jayant Sharma, Max J. Egenhofer: Modeling topological spatial relations: Strategies for query processing. Computers & Graphics 18(6): 815-822 (1994)

[7] Zhan Cui, Anthony G. Cohn, David A. Randell: Qualitative and Topological Relationships in Spatial Databases. SSD 1993: 296-315

[8] Max J. Egenhofer, John R. Herring: Categorizing Binary Topological Relations Between Regions, Lines, and Points in Geographic Databases. Technical Report. Department of Surveying Engineering, University of Maine, 1991

[9] ESRI. 1995. ESRI Spatial Database Engine (SDE). Environmental Systems Research Institute, Inc., Redlands, CA: http://edndoc.esri.com/arcsde/9.1/

[10] Yi Fang, Marc Friedman, Giri Nair, Michael Rys, Ana-Elisa Schmid: Spatial indexing in Microsoft SQL server 2008. SIGMOD Conference 2008: 1207-1216

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

[12] IBM Informix, Informix Geodetic and Spatial DataBlades, http://www-01.ibm.com/software/data/informix/blades/

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

[14] Ravi Kanth Kothuri, Siva Ravada: Efficient Processing of Large Spatial Queries Using Interior Approximations. SSTD 2001: 404-424

[15] 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

[16] Paul A. Longley, Michael F. Goodchild, David J. Maguire, David W. Rhind: Geographic Information Systems and Science. John Wiley & Sons Ltd, West Sussex, UK. (2005)

[17] Microsoft SQL Server, Spatial Indexing Overview, http://technet.microsoft.com/en-us/library/bb964712.aspx

[18] NAVTEQ, http://www.navteq.com/

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

[20] Oracle Spatial, Oracle® Spatial Developer's Guide 11g Release 2 (11.2), Part Number E11830-07, 2010

[21] Joseph O'Rourke: Computational Geometry in C. Cambridge University Press, Cambridge, UK (1998)

[22] Dimitris Papadias, Yannis Theodoridis, Timos K. Sellis, Max J. Egenhofer: Topological Relations in the World of Minimum Bounding Rectangles: A Study with R-trees. SIGMOD Conference 1995: 92-103

[23] PostGIS: http://postgis.refractions.net/

[24] Markus Schneider, Thomas Behr: Topological relationships between complex spatial objects. ACM Trans. Database Syst. 31(1): 39-81 (2006)