GeoInformatica 2:2, 149–174 (1998)

Symbolic Intersect Detection: A Method for Improving Spatial Intersect Joins

YUN-WU HUANG
IBM T.J. Watson Research Center, Hawthorne, NY 10532
ywh@us.ibm.com

MATTHEW JONES
University of Michigan, Electrical Eng. and Computer Science Dept, Ann Arbor, MI 48109
mjones@eecs.umich.edu

ELKE A. RUNDENSTEINER
Worcester Polytechnic Institute, Department of Computer Science, 100 Institute Road, Worcester, MA 01609
rundenst@cs.wpi.edu

Received January 14, 1998; Revised March 16, 1998; Accepted March 17, 1998

Abstract

Due to the increasing popularity of spatial databases, researchers have focused their efforts on improving the query processing performance of the most expensive spatial database operation: the spatial join. While most previous work focused on optimizing the filter step, it has been discovered recently that, for typical GIS data sets, the *refinement step* of spatial join processing actually requires a longer processing time than the *filter step*. Furthermore, two-thirds of the time in processing the refinement step is devoted to the computation of polygon intersections. To address this issue, we therefore introduce a novel approach to spatial join optimization that drastically reduces the time of the refinement step. We propose a new approach called Symbolic Intersect Detection (SID) for early detection of true hits. Our SID optimization eliminates most of the expensive polygon intersect computations required by a spatial join by exploiting the symbolic topological relationships between the two candidate polygons and their overlapping minimum bounding rectangle. One important feature of our SID optimization is that it is complementary to the state-of-the-art methods in spatial join processing and therefore can be utilized by these techniques to further optimize their performance. In this paper, we also develop an analytical cost model that characterizes SID's effectiveness under various conditions. Based on real map data, we furthermore conduct an experimental evaluation comparing the performance of the spatial joins with SID against the state-of-the-art approach. Our experimental results show that SID can effectively identify more than 80% of the true hits with negligible overhead. Consequently, with SID, the time needed for resolving polygon intersect in the refinement step is improved by over 50% over known techniques, as predicted by our analytical model.

Keywords: true hit detection, spatial joins, spatial databases, spatial query processing

1. Introduction

The requirement for spatial data management in Geographic Information Systems (GIS), Cartography, image processing, VLSI, and CAD/CAM has driven the fast-increasing market demand for spatial databases. Spatial join processing is critical in spatial databases because it is commonly used for many classes of queries required by the above-mentioned

applications. One query class is map overlay [4], [13] which merges two map layers into a single layer consisting of the overlapping regions. The second is intersection detection which returns all objects in one target data set that share a spatial relation such as *intersect* or *contain* with at least one object from a second target data set. A query such as ''*Find all counties that intersect some areas flooded last year*'' is an example of such a query based on the *intersect* relation.

Computing spatial joins is very expensive in terms of both CPU and I/O costs because:

1. Spatial objects are typically represented by structures that require extensive storage. For example, a high-resolution vector representation of a polygon may store thousands of points where each point is represented by an x-coordinate and a y-coordinate value.
2. A spatial join operation requires multiple scans of often large data sets.
3. Determining a spatial relation such as *intersect* between two objects is very compute-intensive, namely, it requires super-linear time complexity as a function of the number of points used to represent each object. The most widely used algorithm is called the *plane sweep* algorithm [3], [18], [22] whose lower bound in computation complexity is $O(n \times \log(n))$ where n is the number of points representing the objects.

The first two factors contribute to high I/O costs, whereas the third results in high CPU costs. As a result, spatial join queries over large data sets usually incur a long response time. To minimize cost, strategies for spatial join processing typically execute in two steps [16]:

- The *filter step* uses a conservative geometric approximation, such as the MBR (Minimum Bounding Rectangle), to represent each object. It then computes a set of object-pairs comprised of one object from each target data set, such that the geometric approximations of the two objects share the constrained relation. This resulting set of object-pairs is called the candidate set.
- The *refinement step* retrieves the full spatial data representation of the two objects in each object-pair of the candidate set and runs an algorithm such as *plane sweep* to determine if the spatial relation (specified in the join query) exists between them.

The advantage of the two-step approach is that the filter step eliminates many object-pairs that *cannot* satisfy the spatial predicate without having to retrieve their storage-intensive spatial object representations. More importantly, the CPU-intensive refinement step processing is avoided for the candidates eliminated in the filter step.

Very recently, it has been discovered, based on experiments conducted over GIS map data [17], that the refinement step is more expensive (in terms of overall processing time) than the filter step. This makes the optimization of the refinement step in spatial join processing especially critical. Furthermore, approximately two-thirds of the time in processing the refinement step is devoted to resolving *intersect* relation while the other one-third of the processing time is used to retrieve the spatial representation of the objects [17]. Therefore, spatial join computation is a very CPU-intensive process in which the

intersect test in the refinement step, based on current technology, is likely to be a bottleneck. The situation is even worse for fine-grained (high-resolution) map data[1] because while the filter step is independent of map resolution, the cost of the refinement step increases super-linearly as the resolution increases. To account for the emergence of higher map resolution (e.g., images generated from fine-grained NASA measurement instruments), we are investigating improvements on spatial join processing that are scalable in handling fine-grained data sets.

While most recent spatial join research [2], [9]–[11], [17] presented improvements on the filter step, this paper focuses on the optimization of the refinement step. State-of-the-art approaches in spatial join processing employ the *plane sweep* algorithm [3], [18], [22] to detect intersections between object-pairs during the refinement step. We propose a screen-test procedure to be executed before the *plane sweep* algorithm that substantially reduces the computation required during refinement. We call this procedure Symbolic Intersect Detection (SID). For each object-pair in the candidate set, SID uses the overlapping MBR (OMBR)[2] to clip all the segments of the two polygons which overlap the area bounded by the OMBR. During SID optimization, we represent each clipped segment in a compact symbolic form that identifies where the segments intersect the OMBR. SID generates two sets of such symbolic clipped segments, one for each object in the object-pair. Next, based on the symbolic representation of the clipped segments, SID efficiently determines when any two clipped segments, one from each set, intersect each other, and as a consequence, SID eliminates the expensive *plane sweep* computation for all these detected true hits.

In this paper, we present an analytical model as well as an experimental evaluation over fine-grained real GIS data sets, which confirm that a very high percentage ($> 80\%$) of true hits can be identified by SID. As a result, the overall performance of the refinement computation of spatial join processing is greatly improved by our proposed SID optimization.

This paper makes the following contributions:

1. We propose a promising new approach for the optimization of the refinement step of spatial intersect[3] joins, called SID.
2. We present an analytical cost model characterizing SID's effectiveness under various conditions.
3. Based on real map data, we run experiments comparing the performance of the state-of-the-art spatial join approach with the spatial join using the SID optimization. Our results show that our SID optimization effectively detects more than 80% of the true hits with negligible overhead. Consequently, with the SID optimization, the time *intersect* computation in the refinement step is improved by over 50%, as predicted by the analytical model.
4. Our SID optimization is independent of the access data structures deployed in the filter step which is the focus of many recent research efforts. Consequently, our SID optimization can be directly applied by practically all of the state-of-the-art spatial join methods [2], [9]–[11], [17], therefore further optimizing the performance of these known techniques.

A preliminary version of this paper was presented in [8]. In contrast to the previous version, this paper provides a more detailed presentation of the SID method. Furthermore, in addition to an experimental evaluation based on real GIS maps, a significant portion of this paper now focuses on a thorough performance study of SID based on a new analytical cost model (Section 4) not available in the preliminary version [8].

The outline of the remainder of this paper is as follows: In Section 2, we describe the background and related work on spatial join processing. Our proposed Symbolic Intersect Detection approach is introduced in Section 3, followed by an analytical cost model in Section 4. We present the experimental evaluation in Section 5 and conclude this paper in Section 6.

2. Background on spatial joins

2.1. Related work

The literature has been endowed with a plethora of recent research focusing on spatial join processing. In [15], the z-ordering technique is used to transform multi-D data into the 1-D domain. A spatial join is then conducted on the B^+-tree structures that store z-ordering values of the spatial data. In [20], spatial join indexes are computed using Grid files [14] to index the spatial data. In [12], spatial join is conducted to create the distance-associated join indexes for spatial range queries. In [6], a model of the generalization tree is proposed to compare the tree-based spatial joins with the alternative approaches using cost estimation. Spatial joins based on depth-first traversal of R-trees were proposed in [2]. In [9], we developed an alternative R-tree join technique based on breadth-first traversal which was shown to have a better overall performance than the depth-first approach.

Some spatial join research has focused on joining spatial data when the associated spatial indexes do not exist for both data sets. In [10], a seeded tree is constructed for the data set without an index in order to join it with the R-tree of the other data set. When indexes do not exist for both data sets, a spatial hash join is proposed in [11] that uses spatial partitioning as the hash function. Parallel to the work of [11], a partition-based spatial-merge join is proposed in [17].

All above mentioned research focused mainly on developing a more efficient filtering step (i.e., to optimize false hit reduction). To our knowledge, the only related work that achieves a significant percentage of true hit reduction is the progressive approximation techniques presented in [3]. Their approach achieves roughly 35% true hit detection in the filter step. The advantage of detecting true hits in the filter step is that the vectors for the identified true-hit candidates need not be retrieved in the refinement step. However, their approach incurs additional pre-computation and storage costs in order to create and store the additional approximation for each object. Work in [19] evaluated topological relationship between MBRs to identify a subset of those relationships for which the refinement step is not needed. While it is clear that this approach can reduce the load on the

refinement phase for the *overlaps* predicate, the work does not present true hit detection rates for real maps.

Work in [3] proposed another approach using trapezoid decomposition as the spatial representation for polygons. Although very efficient in spatial join processing, such an approach incurs even more extensive pre-computation and storage costs. Furthermore, it requires creating a spatial representation for polygons that is incompatible with the vector format most commonly used in spatial data applications.

In contrast, our SID approach not only achieves an impressively higher true hit detection rate ($> 80\%$, see Section 5), but it is also independent of the access structures deployed in the filter step and is based on the vector representation of the polygons. Therefore, SID is compatible with the popular spatial data (index and storage) structures and can be used in conjunction with almost all spatial join optimizations of the filter step, particularly those techniques that focus on early detection of false hits [2], [3], [9]–[11], [17], [19].

Since there exist many strategies for optimizing the filter step, our focus in this paper is on the computationally more expensive refinement step. Therefore, for the purpose of this paper, we assume a *candidate set* has already been computed (by any of the popular recently emerged techniques) and is available as the input to the refinement step. The remainder of this section gives a background on the refinement step processing adopted by most recent spatial join techniques [3], [18], [22].

2.2. The two-phase approach in state-of-the-art refinement processing

Upon retrieving from the candidate set a polygon-pair and their vector representations, the state-of-the-art refinement approach determines the *intersect* relation between the two polygons by processing the *restricting* phase and the *sweeping* phase (see figure 1a).

2.2.1. The restricting phase. If a polygon-pair in the *candidate set* is a true hit (its two polygons intersect), the area of intersection must be enclosed by their OMBR. Therefore, any line segments of one polygon that do not overlap[4] the area bounded by the OMBR will not intersect any line segments in the other polygon. Therefore, to determine the *intersect* relation between the two candidate polygons, the *restricting* phase eliminates the line segments that do not overlap the OMBR. Only the ones that overlap the OMBR are retained as input to the next phase: the *sweeping phase*.

If no line segment in either polygon is marked, the two polygons do not intersect, and this candidate can be eliminated from further consideration. Therefore, in addition to reducing the number of line segments, the *restricting* phase can also potentially reduce the number of candidate pairs passed on to the *sweeping phase*.

2.2.2. The sweeping phase. During the *sweeping phase*, the query processor runs the *plane sweep* algorithm [3], [18], [22] to determine whether or not the candidate polygons intersect. The *plane sweep* algorithm sorts the marked line segments from both vectors by the x-coordinate values of their left-most vertex. Next, the algorithm ''sweeps'' a vertical

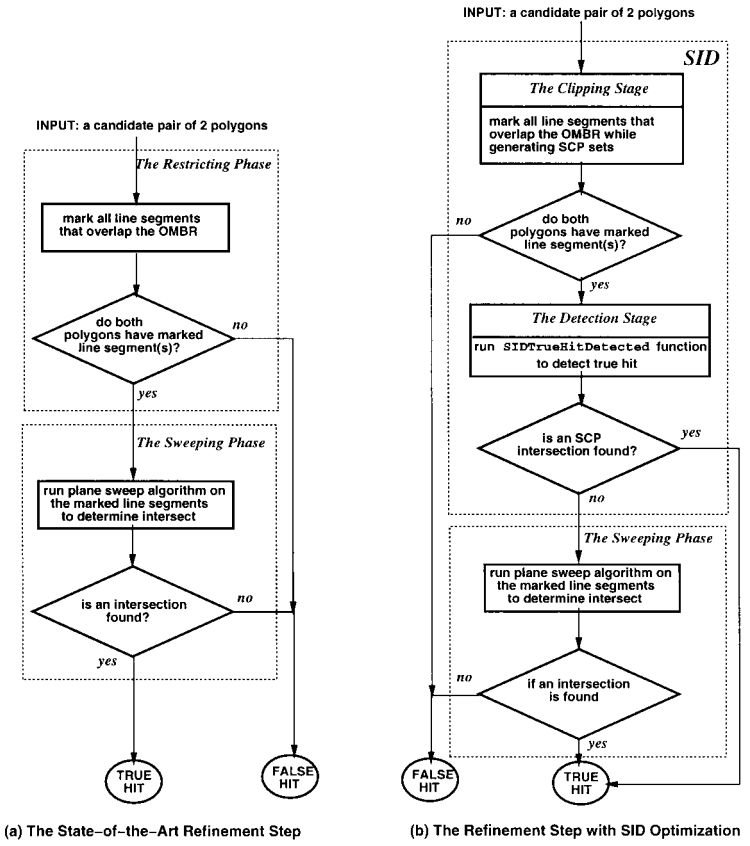

Figure 1. The state-of-the-art approach vs. the SID approach in refinement step.

line across all sorted line segments from left to right. The sweep process evaluates the sorted line segments by stopping at the left-most vertex of each segment it encounters. At each of these vertices, the algorithm conducts an *intersect* test between the encountered line segment and all line segments from the other polygon whose MBRs intersect that of this line segment. If at any time an intersection is found, the sweep process terminates and the candidate satisfies the *intersect* relation. If the sweep process exhausts all line segments and does not find an intersection, the candidate is discarded as the two candidate polygons cannot intersect.

A key performance consideration of the *plane sweep* algorithm is how many iterations of the inner loop the algorithm requires. While the worst case can require as many as n iterations for the inner loop (n is the number of line segments passed on by the *restricting* phase), for GIS map data sets, our experiment results show that the average number of iterations for the inner loop is a small constant (close to 3 with our test data). The average

time complexity to perform the *sweeping phase* on real map data therefore is bounded by the cost to sort the line segments $(O(n \times \log(n)))$ [3] passed on by the *restricting* phase.

3. The Symbolic Intersect Detection (SID) optimization

We now introduce a novel approach to improve the performance of the refinement step called Symbolic Intersect Detection (SID). As is commonly assumed, the two target data sets contain polygon objects and each object is represented by a vector of topologically ordered 2-D boundary points or line segments. For the discussion in this paper, we focus on the *boundary intersect* predicate. However, a generalized intersect predicate such as *area intersect* can be easily extended from the *boundary intersect* computation.[5]

3.1. Overview

Figure 1 contrasts the state-of-the-art refinement step (the state-of-the-art approach) with the refinement that incorporates our SID optimization (the SID approach) presented in the remainder of this section. As shown in figure 1, the SID approach improves on the state-of-the-art approach by directly replacing the *restricting* phase with our proposed SID optimization.

The basis for SID stems from our observation that clipping the two candidate polygons using their OMBR as a window creates two sets of polygon segments, one for each polygon. By examining the topological relationship between these polygon segments and the OMBR, in many cases, we can detect without ambiguity that two polygons intersect. This technique improves performance during the refinement step by reducing the number of polygons that must be passed on to *plane sweep* algorithm. Our SID optimization is comprised of two stages, the *clipping* stage and the *detection* stage.

3.2. The clipping stage of SID

3.2.1. Clipped polygon segments. An ordered traversal of all the line segments in one candidate polygon may cross the boundary of the OMBR zero or more times. By clipping a candidate polygon with the OMBR, we create a set of polygon segments (in the shape of an *arc*), bounded by the OMBR, that are parts of the boundaries of this polygon. Each polygon segment is entirely enclosed by the OMBR with its two end points on the OMBR boundary.

For example, in figure 2, during the *clipping* stage while evaluating the candidate polygons r and s, three clipped polygon segments are created. They are x and y of polygon r, and z of polygon s.

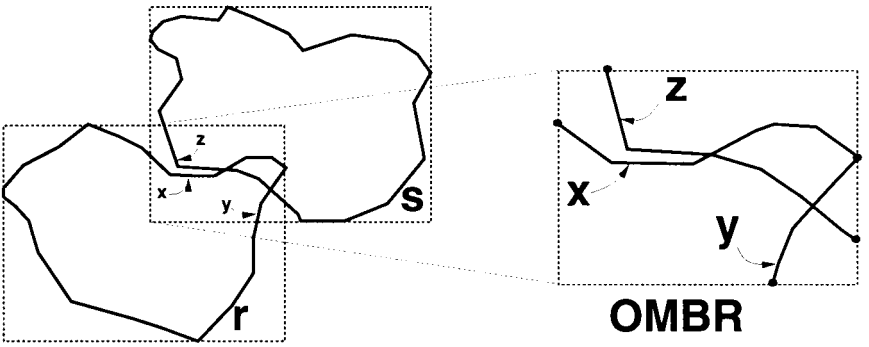

Figure 2. Clipping the polygon segments using OMBR.

3.2.2. Classification of clipped polygon segments. Based on the topological relationship between the OMBR and the two end points of the clipped polygon segments, we categorize these clipped polygon segments into 10 classes. These 10 classes are **Vertical**, **Horizontal**, **Left**, **Up**, **Right**, **Down**, **UpperLeft**, **UpperRight**, **LowerRight**, and **LowerLeft**. Figure 3 illustrates each of these classes. This classification is complete in the sense that all possible cases of clipped polygon segments are covered by this set.

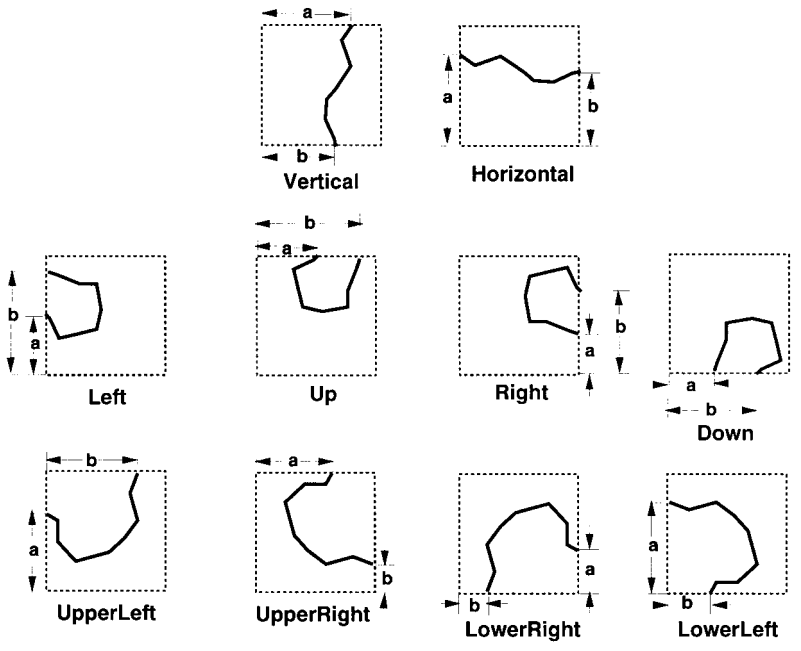

Figure 3. The ten classes of clipped polygon segments.

3.2.3. Symbolic Clipped Primitives (SCP). In SID, each clipped polygon segment is modeled by a symbolic representation we call a Symbolic Clipped Primitive (SCP). Each SCP is represented by a 3-tuple $\langle class, a, b\rangle$, where *class* is one of the ten classes (shown in figure 3) to which the polygon segment modeled by this SCP belongs. The values a and b in $\langle class, a, b\rangle$ are offsets from the corners of the OMBR that identify the location of the segment endpoints. For each SCP, the locations of the corners, from which the offsets a and b are measured from, are determined by the class of the SCP. The relevant corners and offsets for each SCP class are illustrated in figure 3.

3.2.4. The clipping stage creates two SCP sets. During the *clipping* stage, an SCP is generated for each clipped polygon segment. All the SCPs for a candidate polygon are then stored in a set we call the SCP set. The result of the *clipping* stage therefore is two SCP sets, one for each candidate polygon.

3.3. The detection stage of SID

3.3.1. Detecting intersection using SCPs. After the two SCP sets are created for the candidate polygons during the *clipping* stage, SID performs true hit detection by a simple comparison of SCPs from the two sets. To illustrate how many true hits can be detected by comparing SCPs we use the examples depicted in figure 4. Let r and s be the two polygons in a candidate. Then SCP(r) represents a 3-tuple $\langle class, a_r, b_r\rangle$ in the SCP set of r whereas SCP(s) represents a 3-tuple $\langle class, a_s, b_s\rangle$ in the SCP set of s.

Figure 4a shows that if SCP(r) $= \langle$**Vertical**,-,-$\rangle$ and SCP(s) $= \langle$**Horizontal**,-,-$\rangle$ (''-,-'' means don't-care), then the polygon segments represented by SCP(r) and SCP(s) must intersect. As shown in figure 4a, SID requires in this case only the classes of the two SCPs to determine unambiguously that r and s intersect.

In figure 4b, SCP(r) $= \langle$**Vertical**, 3, 4$\rangle$ and SCP(s) $= \langle$**Vertical**, 4, 3$\rangle$. Because $a_r = 3 < 4 = a_s$ and $b_r = 4 > 3 = b_s$, the SCPs must intersect. Therefore, r and s intersect

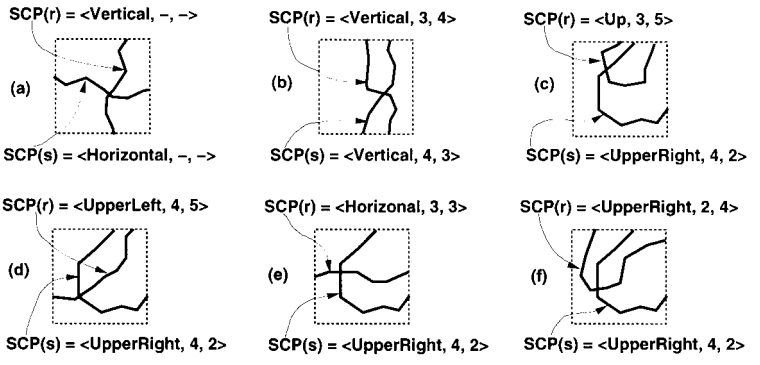

Figure 4. Six examples of intersection detected by SID.

in this case. More generally, if both classes are **Vertical** and we have $(a_r \leq a_s$ and $b_4 \geq b_s)$ or $(a_r \geq a_s$ and $b_r \leq b_s)$, then r and s intersect. Figure 4c is similar to 4b except that the intersection is assured because of the condition $a_r = 3 < a_s = 4 < b_r = 5$. The offset b_s is irrelevant for the intersect detection. Figures 4d, 4e, and 4f show additional examples where SID detects intersections.

In contrast, figure 5 illustrates some of the cases where SID cannot detect an intersection. Note that an intersection does occur in figures 5b and 5c. However, the symbolic information kept by the SCPs is non-resolvable. For instance, figures 5a and 5b have the same SCPs, but figure 5b corresponds to an intersection whereas figure 5a does not.

The examples in figure 4 are a small subset of cases where SID can determine polygon intersection. We present all such cases using a two-D table (Symbol Intersect Resolution Table, or SIRT) in table 1. Each entry in SIRT is a predicate evaluated by the function `SymbolIntersect` at the *detection* stage. `SymbolIntersect` uses the two SCP classes to index into SIRT, evaluates the predicate based on the offsets of the two SCPs, and returns a boolean value. If `SymbolIntersect` returns `TRUE`, it means the two SCPs intersect, whereas a `FALSE` means that SID cannot determine whether or not the two SCPs intersect.

3.3.2. SID performs pair-wise evaluation of the two SCP sets. SID conducts polygon intersect detection by calling the `SymbolIntersect` function for all pairs of SCPs in the two SCP sets created during the *clipping* stage. This process corresponds to a nested-loop over the two SCP sets as illustrated by the function `SIDTrueHitDetected` in figure 6. Once an intersection between two SCPs is detected (line 4 in figure 6), a candidate is known to be a true hit. Only if SID exhausts all SCP pairs without finding an intersection (line 5 in figure 6), SID will pass this candidate to the *sweeping* phase to perform the traditional *intersect* testing using *plane sweep* (as done by the state-of-the-art-approach).

3.4. The SID approach: Restricting phase plus true hit detection

The SID approach improves the refinement processing by deploying the techniques in the *clipping* and *detection* stages as a replacement for the *restricting* phase in the state-of-the-art approach (figure 1). Because the SID *clipping* stage needs to generate the SCP sets as well as perform the state-of-the-art restricting process, it incurs a slightly higher

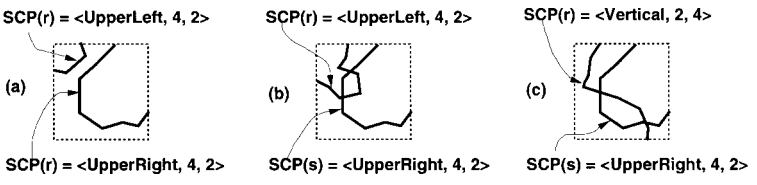

Figure 5. Three examples of non-resolvable cases.

Table 1. Symbol Intersect Resolution Table (SIRT).

SCP(r) = $\langle class, a_r, b_r \rangle$ (row)
SCP(s) = $\langle class, a_s, b_s \rangle$ (column)

Class	*Vertical*	*Horizon*	*Left*	*Up*	*Right*	*Down*	*UpperL*	*UpperR*	*LowerR*	*LowerL*
Vertical	*	T	F	$a_r \le a_s$ $b_r \ge a_s$	F	$a_r \le b_s$ $b_r \ge b_s$	$b_r \ge a_s$	$a_r \le a_s$	$b_r \le b_s$	$b_r \ge b_s$
Horizon	T	*	$a_r \le a_s$ $b_r \ge a_s$	F	$a_r \le b_s$ $b_r \ge b_s$	F	$a_r \le a_s$	$b_r \le b_s$	$a_r \ge b_s$	$a_r \ge a_s$
Left	F	$a_r \le b_s$ $a_r \ge a_s$	*	F	F	F	$a_r \le b_s$ $a_r \ge a_s$	F	F	$a_r \le b_s$ $a_r \ge a_s$
Up	$a_r \le b_s$ $a_r \ge a_s$	F	F	*	F	F	$b_r \le b_s$ $b_r \ge a_s$	$a_r \le b_s$ $a_r \ge a_s$	F F	F F
Right	F	$b_r \le b_s$ $b_r \ge a_s$	F	F	*	F	F	$b_r \le b_s$ $b_r \ge a_s$	$a_r \le b_s$ $a_r \ge a_s$	F
Down	$b_r \le b_s$ $b_r \ge a_s$	F	F	F	F	*	F	F	$b_r \le b_s$ $b_r \ge a_s$	$b_r \le b_s$ $b_r \ge a_s$
UpperL	$a_r \le b_s$	$a_r \ge a_s$	$a_r \le a_s$ $b_r \ge a_s$	$a_r \le b_s$ $b_r \ge b_s$	F	F	*	$a_r \le b_s$	F	$a_r \ge a_s$
UpperR	$a_r \ge a_s$	$b_r \ge b_s$	F	$a_r \le a_s$ $b_r \ge a_s$	$a_r \le b_s$ $b_r \ge b_s$	F	$b_r \ge a_s$	*	$a_r \ge b_s$	F
LowerR	$b_r \ge b_s$	$b_r \le a_s$	F	F	$a_r \le a_s$ $b_r \ge a_s$	$a_r \le b_s$ $b_r \ge b_s$	F	$b_r \le a_s$	*	$b_r \ge b_s$
LowerL	$b_r \le b_s$	$a_r \le a_s$	$a_r \le a_s$ $b_r \ge a_s$	F	F	$a_r \le b_s$ $b_r \ge b_s$	$a_r \le a_s$	F	$b_r \le b_s$	*

Key:
1. ''T'' for TRUE means intersect detected without conditions.
2. ''F'' for FALSE means intersect cannot be determined.
3. ''*'' = $((a_r \le a_s) \wedge (b_r \ge b_s)) \vee ((a_r \ge a_s) \wedge (b_r \le b_s))$.
4. An entry is TRUE if the conjunction of all its clauses, if any, evaluates to TRUE.

```
    Boolean SIDTrueHitDetected(SCP set: x, SCP set: y)
    {
1       for each SCP r in x do
2           for each SCP s in y do
3               if (SymbolIntersect(r, s)) then
4                   return TRUE; // polygons intersect
                end if;
            end do;
        end do;
5       return FALSE; // pass this candidate-pair to plane sweep
    }
```

Figure 6. Function for early true hit detection using SCPs.

computation cost than the *restricting* phase by itself. Our analysis and experiments show that this cost (in terms of processing time) in general is 25% higher than that of the *restricting* phase (see Section 5). However, the time complexity of both approaches are bounded linearly by the number of line segments in the candidate polygons. Our evaluation confirms, however, that the 25% increase in the linear cost component is insignificant compared to the super-linear complexity of *plane sweep* that can potentially be saved by SID.

The time complexity for `SIDTrueHitDetected` is bounded by $O(n \times m)$ where n and m are the cardinality of the two SCPs sets, respectively. In Section 5, our experimental results based on real map data show that the size of an SCP set is typically very small (the average is less than 5). Therefore, the cost of processing the *detection* stage in SID is also very small in practice.

4. Cost analysis

This section presents an analytical cost comparison between the state-of-the-art approach and the SID approach to the refinement step processing. Table 2 is the list of the parameters used in the cost model.

4.1. Cost of processing the state-of-the-art refinement step

4.1.1. Restricting phase. During the *restricting* phase, each line segment in both polygons of the candidate is tested to see if it overlaps the OMBR. The cost to perform the *restricting* phase on a polygon-pair is given by:

Table 2. Parameters used in the cost model.

Parameter	*Description*
n	average number of line segments in one candidate polygon
k	number of candidates in the *candidate set*
d	average number of the inner loops of Plane Sweep computation
s	average number of SCPs in the SCP set for a candidate polygon
$i_{l,m}$	time to compute the intersection between a line segment and an MBR in the *restricting* phase
$\widehat{i_{l,m}}$	time to compute the intersection between a line segment and an MBR while generating an SCP in the *clipping* stage
$i_{i,l}$	time to compute the intersection between two line segments
i_{scp}	time to compute `SymbolIntersect`
p_e	probability that a false hit is eliminated from the candidate set in the *restricting* phase
p_o	probability that a line segment overlaps the OMBR
p_f	probability that an element in the initial *candidate set* is a false hit
p_r	probability that a candidate is a false hit after the *restricting* phase
p_s	probability that a candidate is a false hit after the SID optimization
p_t	probability that a true hit is detected in the *detection* stage

$$C_r = 2 \times n \times i_{l,m}. \tag{1}$$

4.1.2. Sweeping phase. The *plane sweep* algorithm is performed on the reduced sets of line segments for the two candidate polygons in order to determine precisely whether or not the two polygons intersect. The cost of an average *plane sweep* test on two candidate polygons is:

$$C_{ps} = 2 \times n \times p_o \times \log(n \times p_o) + 2 \times n \times p_o \times d \times \frac{1 + p_r}{2} \times i_{l,l}. \tag{2}$$

The term $2 \times n \times p_o \times \log(n \times p_o)$ is the cost to sort the line segments selected during the *restricting* phase separately for both polygons. The term $2 \times n \times p_o$ is the number of outer loops and d is the average number of inner loops for the *plane sweep* algorithm. Therefore, the product of the two is the average number of *intersect* tests required for one full *plane sweep* on one candidate (a polygon-pair).

However, for a true hit we assume that only half of the line segments need to be evaluated before an intersect is found, whereas, for a false hit, the sweeping process must test all line segments. Therefore, the average number of *intersect* tests conducted during a full *plane sweep* run must be adjusted to account for the early termination when an intersection is found. Such an adjustment is accomplished as follows. A false hit incurs a full cost, therefore its probability weighted cost is $1 \times p_r$, where p_r is the probability that a candidate is a false hit. A true hit, on average, incurs only half of the cost, and therefore its probability weighted cost is $\frac{1}{2} \times (1 - p_r)$. The total (probability weighted) cost is the sum of the two, hence the value $(1 + p_r/2)$.

4.1.3. The total state-of-the-art refinement cost. The total state-of-the-art *refinement* cost is given by:

$$C_{ref} = C_r \times k + C_{ps} \times k \times (1 - p_f \times p_e). \tag{3}$$

The parameter k is the number of candidates in the *candidate set*, therefore $C_r \times k$ is the total cost of processing the *restricting* phase for all k candidates. The term $k \times (1 - p_f \times p_e)$ is the number of candidates remaining after the *restricting* phase, therefore $C_{ps} \times k \times (1 - p_f \times p_e)$ is the total cost of processing the *sweeping* phase.

4.2. Cost of the refinement step with SID optimization

The refinement step with SID optimization has two phases: the SID phase and the *sweeping* phase (see figure 1). The SID can be further divided into two stages: the *clipping* stage and the *detection* stage.

4.2.1. The clipping stage of SID. In addition to performing the tasks similar to those performed by the *restricting* phase of the state-of-the-art approach, SID generates the

SCPs for the two candidate polygons. The average cost to perform *clipping* on a candidate-pair is given by:

$$C_c = 2 \times n \times \widehat{i_{l,m}}. \tag{4}$$

The quantity $\widehat{i_{l,m}}$ includes the cost to test if a line segment overlap the OMBR ($i_{l,m}$) and the cost of keeping track of the status of the line segment in order to generate an SCP for each clipped polygon segment.

4.2.2. The detection stage of SID. During the *detection* stage, SID calls the function `SIDTrueHitDetected` (figure 6) to detect true hits. The cost for one execution of `SIDTrueHitDetected` is given by:

$$C_d = s^2 \times \left(1 - \frac{(1-p_r)p_t}{2}\right) \times i_{scp} \tag{5}$$

where s is the average number of items in an SCP set (and therefore defines the number of iterations for each of the loops the `SIDTrueHitDetected` function). Again, as in the analysis of the state-of-the-art approach, we assume early termination half-way through the nested comparison loop for the true hits, and we require all comparison for the undetected ones. The term $1 - ((1-p_r)pt/2)$ constitutes the expected fraction of the full iterations of the intersection loop. It consists of three terms, the number of true hits detected by the SID optimization with an average of 1/2 of the number of iterations, i.e. $\frac{1}{2} \times (1-p_r) \times p_t$, the false hits with a full cost p_r, and the cost for true hits not detected by the SID optimization $(1-p_r) \times (1-p_t)$, also assessed at full cost. These combine to form the expected cost for a single candidate pair: $((1-p_r) \times p_t/2) + p_r + (1-p_r) \times (1-p_t) = 1 - ((1-p_r)p_t/2)$.

4.2.3. The sweeping phase with SID optimization. The average cost of a *plane sweep* on a candidate-pair with the SID optimization is as follows:

$$\widehat{C_{ps}} = 2 \times n \times p_o \times \log(n \times p_o) + 2 \times n \times p_o \times d \times \frac{1+p_s}{2} \times i_{l,l}. \tag{6}$$

We use a probability weighted cost $((1+p_s)/2)$ to account for early termination on hits.

4.2.4. The total refinement cost with SID optimization. The total refinement cost with the SID optimization for the entire candidate set therefore is:

$$\widehat{C_{ref}} = C_c \times k + C_d \times k \times (1 - p_f \times p_) + \widehat{C_{ps}} \times k \times (1 - p_f \times p_e) \times (1 - p_t + p_r p_t). \tag{7}$$

The parameter k stands for the number of items in the *candidate set*. The term $k \times (1 - p_f \times p_e)$ is the number of candidates remaining after the *clipping* stage, and $k \times (1 - p_f \times p_e) \times (1 - p_t + p_r p_t)$ is the number of candidates remaining after the SID optimization.

4.3. *Performance comparison based on the cost model*

To practically use our cost model to compare the performance between the state-of-the-art and the SID approaches, we must reduce the number of variables by giving reasonable values to as many parameters as possible. We achieve this by assigning to all parameters one of the following three types of values: controlled, approximated, and derived.

4.3.1. Parameters with controlled values. In our performance evaluation below, we assume the number of tuples in the candidate set k is 2,000. The parameter n represents the resolution of the data set, and is set to 500, 1,000, 1,500, and 2,000 in our evaluation. The parameter p_f is the probability that a candidate is a false hit, therefore its value depends on how effective the filter step eliminates false hits. We control p_f by assigning to it a range of values from $[0, 1]$, namely, 0.2, 0.4, 0.6, and 0.8. The parameter s is the average number of SCPs in the SCP set for a candidate polygon. Our controlled value of s ranges from 0 to 120.

4.3.2. Parameters with approximated values. As in many complex cost models, some parameters can be assigned an approximated value in order to simplify the analysis. The approximation is typically done by taking measurements gathered from experiments or from historical data. Along these same lines, we conducted an extensive set of experiments on real GIS map data, and measured the values of some parameters as displayed in table 3. Note that the values of $i_{l,m}, \widehat{i_{l,m}}, i_{l,l}$, and i_{scp} are CPU cost relative to the unit cost which is the time needed to perform a compare/swap operation during sorting.[6]

4.3.3. Parameters with derived values. The values of some parameters in our model can be derived from those of the other parameters. In particular, we derive $p_r = (p_f \times (1 - p_e)/1 - p_f p_e)$ and $p_s = (p_r/1 - p_t + p_r p_t)$. We define TDR (*true hit detection rate*), used in subsequent discussions, as the average percentage of true hits that can be detected by SID optimization. TDR is derivable as TDR $= ((1 - p_r \times p_t)/1 - p_f)$ where $(1 - p_r) \times p_t$ is the probability that a candidate is a true hit detected by SID and $1 - p_f$ is the probability that a candidate is a true hit. For ease of discussion (and reading),

Table 3. Parameters with approximated values.

Parameter	d	p_e	p_o	$i_{l,m}$	$\widehat{i_{l,m}}$	$i_{i,l}$	i_{scp}
Approximated value	3	0.5	0.3	1.2	1.5	70	5

we also use CFR (*candidate set false hit rate*) as a percentage representation of the probability parameter p_f (CFR $= p_f$).

4.3.4. Evaluation 1: The effect of the percentage of true hits detected by SID. In this evaluation, we set $n = 1{,}000$ and vary TDR from 0% to 100%. Figures 7 and 8 show the results of our cost model based on equations (3) and (7). The two figures are plotted with s (the number of items per SCP set) set to 10 and 50, respectively. The solid horizontal line in each of the two figures depicts the cost of the state-of-the-art approach. The other non-solid lines represent, in percentage to the cost of the state-of-the-art approach, the costs of the SID approach with CFR set to 20%, 40%, 60%, and 80%, respectively. The case when CFR $= 0\%$ or 100% is very unlikely and therefore is omitted.

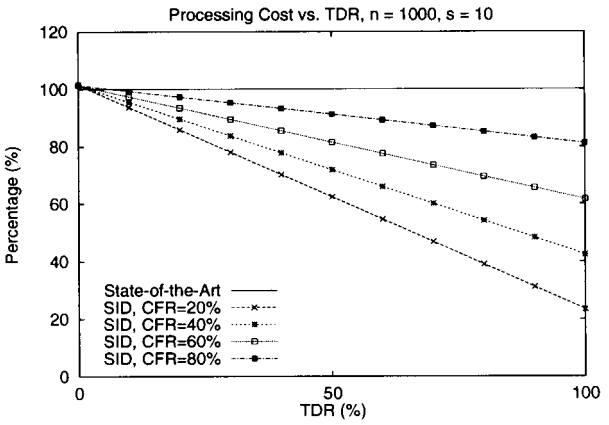

Figure 7. Processing cost vs. TDR ($s = 10$).

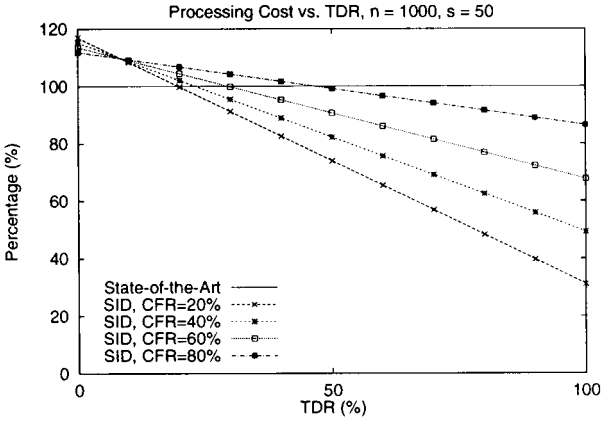

Figure 8. Processing cost vs. TDR ($s = 50$).

From figures 7 and 8 we see that a higher TDR achieved by SID results in a greater performance improvement of our proposed refinement step over the state-of-the-art approach. Among the various CFR values, the performance improves most drastically with a small CFR when the TDR increases. This is because when the CFR is small, most of the candidates in the *candidate set* are true hits. A higher TDR achieved by SID on such a *candidate set* eliminates the need for processing the *sweeping* phase for a large number of the candidates, thus improving the overall performance more significantly.

Contrasting figure 7 with figure 8, we can see that the cost savings achieved by SID is less significant when $s = 50$ than when $s = 10$. In other words, when s is larger (i.e., $s = 50$ in figure 8), the overhead of processing the *detection* stage (equation (5)) increases enough so that performance advantages are only gained when TDR exceeds 50%. However, when s is small (i.e., figure 7), SID incurs minimum overhead and promises performance improvements even for a modest TDR (such as 20%). Based on our experimental studies over a set of real map data in Section 5, we discover that the average s is between 3 and 5, and therefore drastic improvements can be expected by our technique in real-life applications.

4.3.5. Evaluation 2: The effect of the average number of items in an SCP set. We first set the TDR to 60%, CFR to 40%, and vary s from 0 to 120 with n set to 500, 1,000, 1,500, and 2,000. The results in figure 9 show that when s is small (i.e., $s < 10$), the performance improvement achieved by SID is very significant (40% improvement), and is insensitive to n. This means that when s is small, the improvement achieved by SID is very significant across maps of various resolutions (i.e., values of n). Only when s is very large (> 60), will SID optimization become ineffective, particularly for less fine-grained map data. However, since in real-life GIS data, the value of s is typically very small, therefore SID is likely to achieve significant improvement (roughly 40%) for maps of various resolutions when TDR = 60% (SID detects 60% true hits).

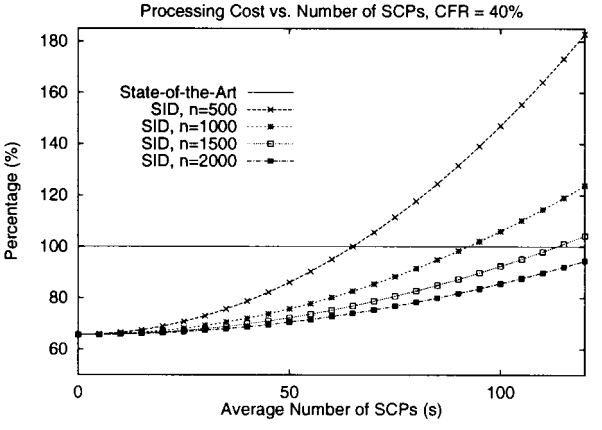

Figure 9. Processing cost vs. number of SCPs (CFR = 40%).

Next, we set TDR to 60%, $n = 1{,}000$, and vary s from 0 to 120 with values of CFR set to 20%, 40%, 60%, and 80%. The results in figure 10 show that, with a small s, the higher the CFR the greater the performance improvement SID achieves. This is because when the CFR is high, most tuples in the candidate set are true hits. Because SID detects true hits (in this evaluation, 60% of true hits) with only negligible overhead, the total refinement cost improves as CFR increases. From the two sets of evaluation based on our cost model, we conclude that a small s value and a high TDR combined can provide a dramatic performance improvement in the refinement step. In the next section, we investigate the values of s and TDR for real data sets and demonstrate that such a favorable combination indeed occurs in real map data.

5. Experimental evaluation

We conduct experimental evaluation based on real maps for two purposes. First, we measure the performance improvement of SID against the state-of-the-art refinement processing. Second, we compare the results with those predicted by our cost model in order to verify the accuracy of the cost model.

5.1. Experimental design

5.1.1. Testbed. All experiments were conducted on a SUN Sparc-20 workstation running the Unix operating system. The implementation of the filtering step in the spatial join is based upon optimization techniques proposed in [2]. For the refinement step, we implemented the traditional refinement step as well as the refinement step that incorporates

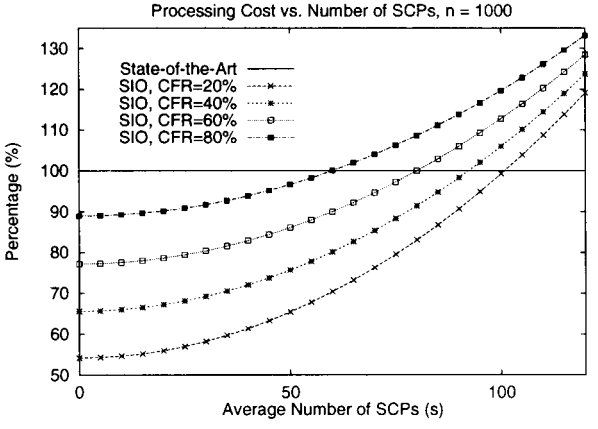

Figure 10. Processing cost vs. number of SCPs ($n = 1{,}000$).

the SID optimization technique presented in this paper. All programs were written in C++.

5.1.2. Maps. The map data used in our experiments were extracted from the polygon data sets in the Sequoia 2000 Storage Benchmark [23]. In particular, they are regions of various land use and land cover classifications in the State of California and Nevada. The number of line segments per polygon range from tens to thousands. Because our SID optimization is targeted to handle complex spatial data, we extracted several sets of polygons and created eight classes of data with the minimum number of line segments set to 100, 200, 300, 400, 500, 600, 700, and 800. This resulted in eight groups of data sets with an average number of line segments of 850, 950, 1,100, 1,300, 1,450, 1,650, 1,750, and 1,850, respectively (see table 4).

5.1.3. Spatial join experiments. For each map class, we selected a pair of polygon data sets, performed the (intersect) spatial join between the two sets and recorded the CPU usage for both versions of the refinement step.

5.2. Experiments measuring the size of SCP sets

Based on our analytical model presented in Section 4, the two most important factors that determine the performance of SID are the cardinality of the SCP set (s) and the true hit detection rate (TDR). Particularly, if s is small (< 10), the performance gain of SID is assured, and is proportional to the value of TDR. Intuitively, if we randomly lay a rectangle over a smooth-shaped polygon, then the number of the line segments of the polygon that cross the rectangle boundary is likely to be very small. Because the number of crossings determines the cardinality of the SCP set, the s value for a smooth-shaped polygon is also likely to be very small. Conversely, if the polygon has many zigzag-shaped components, then it is possible that more of the polygon's line segments cross the boundary of an overlaid rectangle, thereby contributing to a higher s value. We now present the s values measured from real map data during the *clipping* stage of SID.

Figure 11 shows the distribution of s values for maps with polygons of various sizes, namely classes 1 ($n = 850$), 5 ($n = 1,450$), and 8 ($n = 1,850$). We can see by the results in figure 11 that maps with more segments per polygon (higher-classes) tend to have larger SCP sets. Nevertheless, our measurements show that no map produces an s value that is greater than 16 during a SID enhanced join, and most of the s values are smaller than 5 for maps in all three classes. In fact, more than 95% of the MBR overlapping cases have fewer

Table 4. Map classifications.

Map Class Number	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Min. no. of line segments per polygon	100	200	300	400	500	600	700	800
Avg. no. of line segments per polygon	850	950	1,100	1,300	1,450	1,650	1,750	1,850

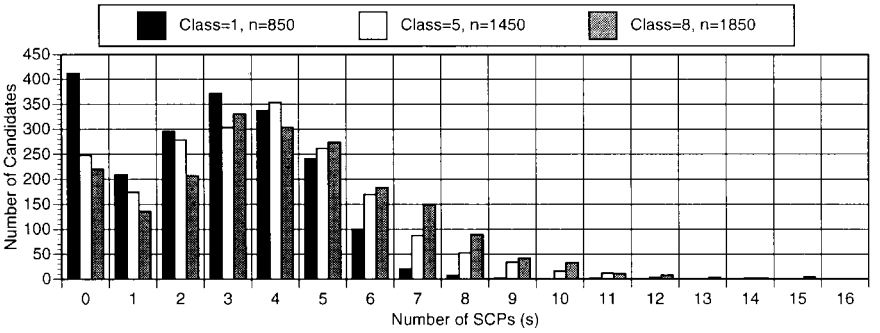

Figure 11. Distribution of number of SCPs per polygon.

than 10 SCPs per polygon for all map classes. These results indicate that it is very likely that MBR overlapping between the majority of polygons in typical GIS maps will result in small SCP sets (small s). Based on our analytical model, this finding assures that our proposed SID optimization will be effective in reducing the cost of the refinement step during spatial join processing.

5.3. *Experiments measuring the True Hit Detection Rate (TDR)*

Figure 12 records the TDR achieved by SID for maps of all classes (1–8). The results show that there is no notable correlation between the TDR and the complexity (class) of the

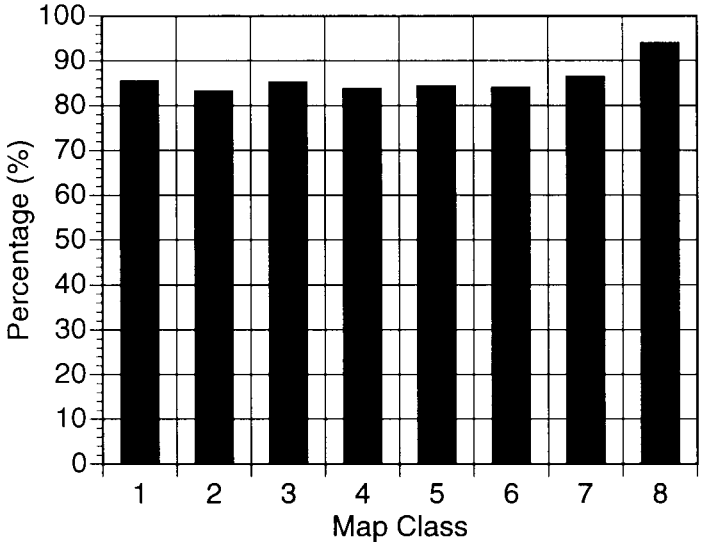

Figure 12. True Hit Detection Rates achieved by SID.

maps. In fact, in all map classes, the true hit detection rates are consistently between 82 and 94%. This finding indicates that our proposed SID optimization technique effectively eliminates the need of expensive plane-sweep intersect operations for more than 80% of the true hits in the *candidate set* for maps with a wide variety of complexities.

5.4. *Experimental results comparing performance of the refinement steps*

In figure 13, we compare the CPU usage times between SID and the state-of-the-art approach for all map classes (1–8). The results show that SID achieves a significant performance improvement for all map classes. In figure 14, we show the percentage of the CPU costs of SID over the state-of-the-art approach. The results show that the performance gain by SID map classes 2 $(n = 950)$ to 4 $(n = 1,300)$ is above 40% whereas an over 50% improvement is achieved for map classes higher than 4 $(n > 1,300)$.

5.5. *Comparing experimental results with analytical results*

Based upon the analytical model that we presented in Section 4.3, and depicted in figure 10, we predict that for $n = 1,000$, $s = 10$, CFR $= 40\%$, and TDR $= 80\%$, the CPU time for refinement in the SID approach is slightly higher than 50% of CPU time for refinement in the state-of-the-art approach. This means that a performance gain of more than 40% is expected in the refinement step. The experimental results in figure 14 show that the SID

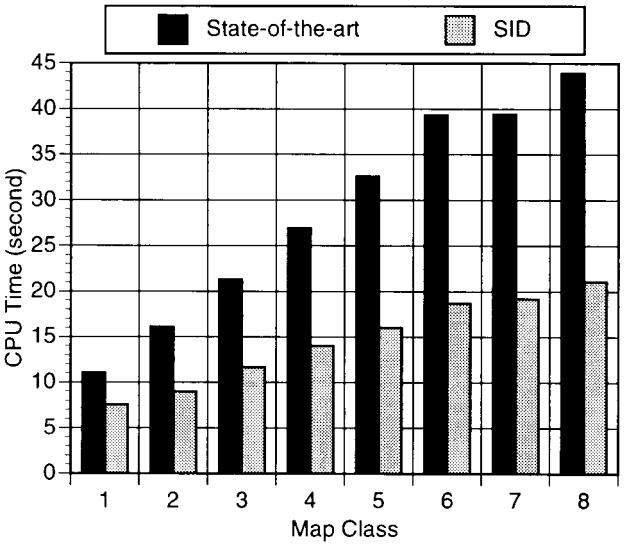

Figure 13. CPU usage time: SID vs. state-of-the-art.

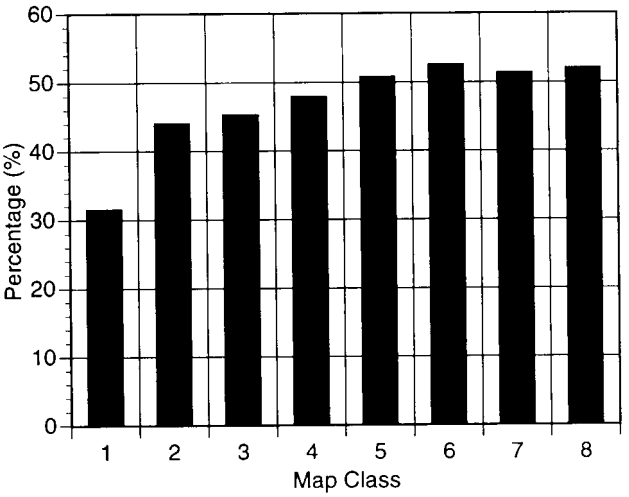

Figure 14. Percentage of the CPU time achieved by SID of that achieved by the state-of-the-art approach.

approach outperforms the state-of-the-art refinement step by more than 40% for map classes 2 and 3 (with 950 and 1,100 line segments, respectively). The two results, experimental and analytical, agree with a very small discrepancy ($< 10\%$). This demonstrates that our analytical cost model is accurate in predicting the effectiveness of our proposed SID technique.

6. Conclusion

Efficient processing of spatial joins is crucial for many applications such as GIS, Cartography, CAD/CAM, etc. Because spatial join computation has been recognized as a CPU-intensive process [17], we consider the optimization of the most computation-expensive component in spatial join processing: the refinement step. To this end, we present the Symbolic Intersect Detection (SID) technique that significantly improves the efficiency of the refinement step. SID performs efficient true hit detection in two stages. First, to determine if two candidate polygons intersect, SID uses their OMBR to clip all segments of the two polygons which overlap the OMBR. SID abstracts each segment by a compact symbolic representation using only offsets of the sides of the OBMR. Next, based on this abstract information, SID efficiently detects situations under which two clipped segments cross each other deterministically. When such a crossing is determined between two clipped segments, their association polygons therefore are guaranteed to intersect. As a result, performance is improved because further *intersect* computation such as the computationally intensive *plane sweep* algorithm is not needed for the true hit candidates detected by SID.

In this paper, in addition to presenting the SID approach, we also describe an analytical cost model for studying the performance of SID. Our model identifies that SID is most effective when the number of clipped segments is small and when polygon representation is fine-grained. In addition, we conduct an experimental evaluation of SID and its state-of-the-art competitor based on real GIS maps with complex polygons. Our experimental results on real GIS maps show that the average number of clipped segments per polygon is indeed small (< 5). Furthermore, an impressive percentage (greater than 80%) of the true hits are shown to be detected by SID with only negligible overhead. Consequently, with the SID optimization, the time to resolve polygon intersection in the refinement step is improved by over 50%, as predicted by our analytical model. Because SID optimizes the refinement step, it is complementary to many state-of-the-art spatial join techniques [2], [9]–[11], [17] that have focused on improving the filter step. The combination of an impressive performance gain achievable by SID and its compatibility with other approaches therefore promises a significant improvement for existing solutions to spatial join processing.

Acknowledgments

This work was supported in part by the University of Michigan ITS Research Center of Excellence grant (DTFH61-93-X-00017-Sub) sponsored by the U.S. Dept. of Transportation and by the Michigan Dept. of Transportation.

This work was performed while Yun-Wu Huang was a Ph.D. Student at the University of Michigan and while Elke A. Rundensteiner was a faculty member of the University of Michigan.

The authors thank students in the University of Michigan Database Group (UMDG) and also in the Database Systems Research Group (DSRG) at WPI for their valuable feedback on this work.

Notes

1. A fine-grained map means its objects are represented by large numbers (hundreds or more) of points.
2. The OMBR of an object-pair is the rectangle which is the overlap between the MBRs of the two objects.
3. For the discussion in this paper we focus on the *boundary intersect* predictate.
4. We use *overlap* to mean *intersects* or *contains* or *is contained by.*
5. The object-pairs that satisfy the *area intersect* predicate are the union of those that satisfy the *boundary intersect* predicate and those that do not satisfy the *boundary intersect* predicate but satisfy the *contain* predicate. The most common technique in resolving the *contain* predicate is called the point test [18] which has a linear time complexity in the length of the vector representation. Therefore, *contain* test can be much less expensive than *intersect* test which dominates the cost in *area intersect* joins.
6. It is much cheaper to compute $i_{l,m}$ than $i_{l,l}$ because determining whether or not a segment intersects an orthogonal rectangle is usually a simple comparison, whereas determining if two segments intersect requires floating point division, a notoriously expensive operation.

References

1. N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. ''The R*-tree: An Efficient and Robust Access Method for Points and Rectangles,'' in *Proc. of the 1990 ACM SIGMOD Int. Conf. on Management of Data*, 322–332, 1990.
2. T. Brinkhoff, H. Kriegel, and B. Seeger. ''Efficient Processing of Spatial Joins Using R-trees,'' in *Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of Data*, 237–246, 1993.
3. T. Brinkhoff, H. Kriegel, R. Schneider, and B. Seeger. ''Multi-Step Processing of Spatial Joins,'' in *Proc. of the 1994 ACM SIGMOD Int. Conf. on Management of Data*, 197–208, 1994.
4. P.A. Burrough. Principles of Geographic Information Systems for Land Resources Assessment. Oxford University Press, 1986.
5. C. Faloutsos and I. Kamel. ''On Packing R-tree,'' in *Proc. of the Conference on Information and Knowledge Management*, 490–499, 1993.
6. O. Gunther. ''Efficient Computation of Spatial Joins,'' in *Proc. of the 9th Int. Conf. on Data Eng.*, 50–59, 1993.
7. A. Guttman. ''R-tree: a dynamic index structure for spatial searching,'' in *Proc. of the 1984 ACM SIGMOD Int. Conf. on Management of Data*, 45–57, 1984.
8. Y.W. Huang, M. Jones, and E.A. Rundensteiner. ''Improving Spatial Intersect Joins Using Symbolic Intersect Detection,'' in *Proc. of the 5th International Symposium on Spatial Databases*, 165–177, 1997.
9. Y.W. Huang and E.A. Rundensteiner. ''Spatial Joins Using R-trees: Breadth-First Traversal with Global Optimizations,'' in *Proc. of the 23rd International Conference on Very Large Data Bases*, 396–405, 1997.
10. M.L. Lo and C.V. Ravishankar. ''Spatial Join Using Seeded Trees,'' in *Proc. of the 1994 ACM SIGMOD Int. Conf. on Management of Data*, 209–220, 1994.
11. M.L. Lo and C.V. Ravishankar. ''Spatial Hash-Joins,'' in *Proc. of the 1996 ACM SIGMOD Int. Conf. on Management of Data*, 247–258, 1996.
12. W. Lu and J. Han. ''Distance-associated Join Indices for Spatial Range Search,'' *IEEE 8th Int. Conf. on Data Engineering*, 284–292, 1992.
13. D.J. Maguire, M.F. Goodchild, and D.W. Rhind. Geographic Information Systems, volume 1. John Wiley & Sons, Inc.: New York, 1991.
14. J. Nievergelt and H. Hinterberger. ''The Grid File: An Adaptable, Symmetric Multikey File Structure,'' *ACM Transactions on Database Systems*, 9(1):39–71, 1984.
15. J.A. Orenstein. ''Spatial Query Processing in an Object-Oriented Database System,'' in *Proc. of the 1986 ACM SIGMOD Int. Conf. on Management of Data*, 1986.
16. J.A. Orenstein. ''A Comparison of Spatial Query Processing Techniques for Native and Parameter Spaces,'' in *Proc. of the 1990 ACM SIGMOD Int. Conf. on Management of Data*, 343–352, 1990.
17. J.M. Patel and D.J. DeWitt. ''Partition Based Spatial-Merge Join,'' in *Proc. of the 1996 ACM SIGMOD Int. Conf. on Management of Data*, 259–270, 1996.
18. F.P. Preparata and M.I. Shamos. Computation Geometry. Springer, 1985.
19. D. Papadias, Y. Theodoridis, T. Sellis, and M.J. Egenhofer. ''Relations in the World of Minimum Bounding Rectangles: A Study with R-trees,'' in *Proc. of the 1995 ACM SIGMOD Int. Conf. on Management of Data*, 92–103, 1995.
20. D. Rotem. ''Spatial Join Indices,'' *IEEE 7th Int. Conf. on Data Engineering*, 500–509, 1991.
21. T. Sellis, N. Roussopoulos, and C. Faloutsos. ''The R^+-Tree: A Dynamic Index for Multi-dimensional Objects,'' in *Proc. of the International Conference on Very Large Data Bases*, Brighton, England, 3–17, 1987.
22. M.I. Shamos and D.J. Hoey. ''Geometric Intersection Problems,'' in *Proc. 17th Annual Conf. on Foundations of Computer Science*, 208–215, 1976.
23. M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. ''The SEQUOIA 2000 Storage Benchmark,'' in *Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of Data*, 1993.

Yun-Wu Huang received the B.S. degree in Management Science from National Chiao-Tung University in 1982, and the M.S. degree in Computer Science from Indiana University in 1989. He had worked as a computer professional in the areas of databases and computer networks between 1989 and 1995. He graduated with a Ph.D. in Computer Science from the University of Michigan in 1997. Presently, he is a research staff at the IBM T.J. Watson Research Center. His current research interests include spatial databases, Geographic Information Systems, data mining, and distributed systems.

Matthew Jones has a BSEE from Rice University, and a MSCS from the University of Michigan. In the intervening 5 years, he wrote commendable design software to support his efforts as a digital logic designer. He is currently ABD in Computer Science in the Software Systems Research Lab at the University of Michigan, and he works full time developing internet routing software with IEng. His interests include special applications of databases, design patterns for software development, and balancing the demands of career with the exhilarations of fatherhood.

Elke Angelika Rundensteiner is currently Associate professor of the Department of Computer Science at the Worcester Polytechnic Institute, after having been a faculty member in the Department of Electrical Engineering and Computer Science at the University of Michigan. She has received a B.S. degree (Vordiplom) from the Johann Wolfgang Goethe University, Frankfurt, West Germany, an M.S. degree from Florida State University, and a Ph.D. degree from the University of California, Irvine. Dr. Rundensteiner has been active in the database research community for over 10 years now, and her current research interests include object-oriented databases, data warehousing, database and software evolution, multi-media databases, distributed and web database applications, and information visualization. She has more than 100 publications in these areas. Her research has been funded by government agencies including NSF, ARPA, NASA, CRA, DOT; and by industry including IBM, AT&T, Informix and GE. She is on Program Committees of the key conferences in the database field such as IEEE ICDE, CIKM, ACM SIGMOD, VLDB, as well as others. Dr. Rundensteiner has received numerous honors and awards, including a Fulbright Scholarship, an NSF Young Investigator Award, an Intel Young Investigator Engineering Award, and an IBM Partnership Award.