

On Multi-Way Spatial Joins with Direction Predicates^{*}

Hongjun Zhu, Jianwen Su, and Oscar H. Ibarra

Department of Computer Science
University of California
Santa Barbara, CA 93106
USA
{hongjunz,su,ibarra}@cs.ucsb.edu

Abstract. Spatial joins are fundamental in spatial databases. Over the last decade, the primary focus of research has been on joins with the predicate “region intersection.” In modern database applications involving geospatial data such as GIS, efficient evaluation of joins with other spatial predicates is yet to be fully explored. In addition, most existing join algorithms were developed for two-way joins. Traditionally, a multi-way join is treated as a sequence of two-way joins. The goal of this paper is to study evaluation of multi-way spatial joins with direction predicates: complexity bounds and efficient algorithms. We first give I/O efficient plane sweeping based algorithms for 2-way direction joins and show that by combining the plane sweeping technique with external priority search trees, a 2-way direction join of N -tuple relations can be evaluated in $O(N \log_b \frac{N}{M} + k)$ I/Os in the worst case, where M is the size of the memory, b is the page size and k is the result size. The algorithms are then extended to perform a subclass of multi-way direction joins called “star joins”. We show that the I/O complexity of evaluating an m -way star join of N -tuple relations is $O(mN \log_b \frac{N}{M} + K + k)$, where $K \leq mN^2$ is the size of the intermediate result, M , b and k ($\leq N^m$) are the same as above. We also apply the algorithm for star joins to evaluate a more general case of multi-way joins, which are star connections of star joins and show that this can be done in polynomial time. In the general case, we show that testing emptiness of a multi-way direction join is NP-complete. This lower bound holds even when in the join predicate (1) only one attribute for each relation is involved, and (2) each spatial attribute occurs a bounded number of times. It implies that join evaluation in these cases is NP-hard.

1 Introduction

Similar to the relational join operations, spatial joins play an important role in spatial databases since they can connect different datasets through *spatial predicates*. Unfortunately, spatial joins are very expensive to evaluate. Although

^{*} Support in part by NSF grants IRI-9700370 and IIS-9817432.

one could treat spatial joins as user-defined external functions in the efficient evaluation of joins, it is generally believed that more efficient algorithms may be developed by exploiting the semantics of spatial predicates. Unlike the relational case, there is a rich set of spatial predicates, ranging from topological properties (e.g., intersection), (Euclidean) distances, to directions. Directions are fundamental in especially geospatial database applications. The focus of this paper is to study algorithms for spatial joins with direction predicates.

A multi-way direction join involves more than two relations. Little has been done for multi-way spatial join evaluation. In relational databases, a multi-way join is processed by combining multiple 2-way joins. This method was used for multi-way spatial join in [MP99,PRS99]. Their algorithms require that the input relations have R -tree indexes. First, two relations are joined together using R -tree based 2-way spatial join algorithm and the result is joined with a third relation, and then the fourth relation, and so on. In [MP99], the 2-way join between an intermediate result and an input relation is evaluated by a 2-way spatial join algorithm for the case where one of the input relations (the intermediate result) does not have R -tree index. In [PRS99], a spatial index structure is constructed for the intermediate result and is used in the join with the input relation. In [MP98], search algorithms for Constraint Satisfaction Problem (CSP) are combined with R -tree based 2-way spatial join algorithms to solve a special case of multi-way spatial join where there exists a join condition between each pair of input relations. Their method was further extended to evaluate multi-way spatial joins in [PMT99]. The key technique in the extended method is to traverse several R -trees simultaneously to achieve I/O efficiency and use techniques in CSP to speed up the join with the remaining relations. In the present paper, we investigate multi-way spatial joins with direction predicates with a focus on I/O complexity in the worst case.

A lot of work has been done on 2-way spatial joins. Most previous work focused on joins with the predicate *region intersection*. The intersection spatial join algorithms can be roughly divided into two categories: ones that use spatial index structures and ones that do not. Algorithms of [BKS93,HJR97] use spatial index structures such as R -trees, R^+ -trees, R^* -trees. Interval B-trees [ZSI99] and external segment trees [Arg95] can also be used. Algorithms without using index structures were reported in [PD96], [LR96], and [APR⁺98,Vit98]. [PD96] and [LR96] used a partition based method, and [APR⁺98,Vit98] applied the distributes-sweeping technique developed in [GTVV93]. In [Rot91,SA97], spatial joins with a more general predicate *within* was considered, which returns objects within some distance. Hjaltason and Samet [HS98] developed a distance spatial join algorithm that uses R^* -tree like index structure and priority queue to compute tuples whose spatial attributes are within a given range. The worst case I/O complexity of all above algorithms is unfortunately $O(N^2)$, except that algorithms of [Arg95,APR⁺98,ZSI99] have a $O(N \log N + k)$ I/O complexity for intersection spatial join.

Günther [Gün93] proposed a general method using *generalization trees* as index structures for evaluating joins with many predicates including intersection,

direction, distance, etc. Becker, Hinrichs, and Finke used *grid file* to compute spatial joins [BHF93]. These algorithms use data structures that cannot guarantee better I/O complexity in the worst case. In fact, in the worst case the I/O cost of these join algorithms is still $O(N^2)$. There is no other work on spatial joins with direction predicates to the best of our knowledge.

Spatial objects are potentially very large. Approximations are frequently used to reduce the access to the spatial objects. A popular approximation method is minimum bounding rectangles (mbr's). Typically, the evaluation of a spatial join employs two steps: (1) Filter step that applies spatial join on the mbr's of the objects; and (2) Refinement step that checks whether the objects discovered by the filter step satisfy the given predicate. Most work on spatial joins focuses on the filter step and mbr's, (although some work on other approximation methods has been done [BKSS94,ZS98,ZSI00a,ZSI00b]). This is also the focus of the present paper. Specifically, we define and study direction predicates between rectangles.

This paper makes the following three contributions.

1. We develop an I/O efficient algorithm for 2-way direction joins with one predicate. The algorithm uses the plane sweeping technique and I/O efficient data structures such as external priority search trees [ASV99] and B-trees. We show that each 2-way direction join with one predicate can be computed within $O(N \log_b \frac{N}{M} + k)$ I/Os, where N is the size of the relations, M is the size of the memory, b is the page size, and k is the number of rectangle pairs satisfying the direction predicate (the result size).
2. We extend our algorithm for a subclass of multi-way direction joins, "star joins", in which all join predicates involve one common relation (called the "center relation"). The idea is to decompose star joins into a sequence of 2-way direction joins with one predicate and merge their results together. For the case where the center relation of the star join is always the right relation in a join predicate, we use the 2-way direction join algorithms to evaluate each 2-way join and store their results in one B-tree on tuples in the center relation. When the center relation also occurs as left relations in the join condition, we develop new techniques to ensure I/O efficiency. Let M be the size of the memory. We show that each m -way star join of N -tuple relations can be evaluated in $O(mN \log_b \frac{N}{M} + K + k)$ I/Os, where $K \leq mN^2$ is the size of the intermediate result, and $k \leq N^m$ the size of the join result. We then extend the evaluation algorithm for star joins to evaluate multi-way direction joins that are star connections of d star joins. We show that the I/O complexity in the worst case is $O(dN^2 \log_b \frac{dN}{M} + mN \log_b \frac{N}{M} + dmN + K + k)$ where m is the total number of relations (occurrences) involved.
3. We also study the complexity of evaluating a multi-way direction join. Specifically, we study the problem of testing if the result of a given multi-way join with direction predicates of given relations is empty. We show that this problem is NP-complete in general and surprisingly even for the case when (1) only one (spatial) attribute from each relation is involved in the join condition, and (2) each (spatial) attribute occurs a bounded number of times

in the join condition. The result immediately implies that the multi-way direction join evaluation problem is intractable.

This paper is organized as follows. Section 2 defines a spatial model and direction predicates used in the paper. Section 3 presents the algorithms for evaluating 2-way direction joins with single predicate. Section 4 discusses evaluation of star joins and extensions. Section 5 studies the complexity of evaluating multi-way direction joins. Section 6 gives the conclusions.

2 Direction Joins

In this section, we briefly introduce the spatial data model used in this paper, define the direction predicates and multi-way direction join operations, i.e., spatial joins with direction predicates.

We consider spatial regions that are (*filled*) rectangles with non-zero area in the (real) plane. Two rectangles are said to *interior-intersect* (*i-intersect*) if their interiors intersect. A region is *bounded* if it is contained in the rectangular region $a \leq x \leq b \wedge c \leq y \leq d$ for some real numbers a, b, c, d . A spatial *object* in a database is a bounded rectangle.

A fundamental geospatial properties, direction predicates are an important class of spatial predicates in GIS applications. Unfortunately, there does not seem to be a single standard way to define directions. Some treat direction predicates based on regions induced by horizontal and vertical lines which are projections of the given object [Fra92,PS93,PTS94]. Others use angular directions between objects [Dut89,Her93]. In this paper, we adopt the former and our definitions are close to the ones in [PTS94] and focus on join evaluations.

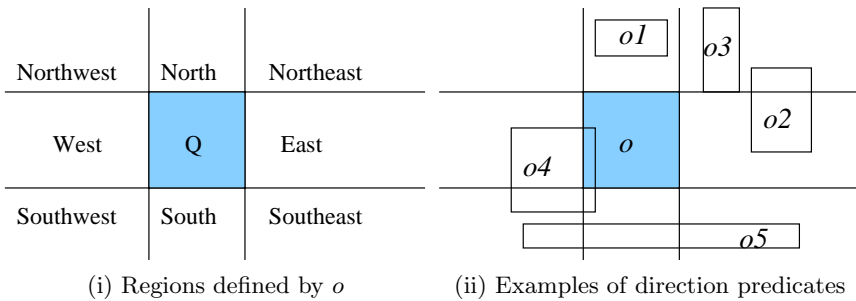


Fig. 1. Direction Predicates and Examples

In our model, we only consider directions between two objects whose interiors do not intersect. (Their boundaries may intersect.) Specifically, a spatial object o induces eight (unbounded) regions as shown in Fig. 1(i). We introduce the following three classes of direction predicates with respect to o using the regions.

- *Uni-direction predicates*: E (east), S, W, N, SE (southeast), SW, NE, and NW. The semantics of all predicates is defined similarly as the following example

illustrates. Two objects o, o' satisfy $(o' N o)$ if o' i-intersects only the *north* region of o and o, o' do not i-intersect. For instance, o_1 in Fig. 1(ii) is in the north of o and $(o_1 N o)$ is true. But $(o_2 E o)$ is false since o_2 also i-intersects the northeast region of o .

- *Bi-direction predicates*: NE-N, NE-E, SE-E, SE-S, SW-S, SW-W, NW-W, and NW-N. A bi-direction predicate is true if one object i-intersects exactly the two specified regions of the other and the two objects do not i-intersect. For example, the object o_2 in Fig. 1(ii) satisfies NE-E with respect to o .
- *Tri-direction predicates*: *weak-p*, where p is one of E, W, S, and N. Two objects o', o satisfy $(o' \text{weak-}p o)$ if o' i-intersects three consecutive regions that are (1) induced by o and (2) centered at the region corresponding to p , and the two objects do not i-intersect. In Fig. 1(ii), $(o_5 \text{weak-S } o)$ is true. (Note that *weak-SW* is not meaningful since it is not possible to have two objects non-i-intersect but one intersects three consecutive regions centered at the southwest, e.g. o_4 . Three other predicates are similar.)

Our definitions are close to the ones in [PTS94], and some of predicates defined above turn out to be the same as theirs. For example, in our definition the predicate N is the same as their “strong_bounded_north.” However, there are some major differences between our definitions and theirs. First, we define direction predicates using unbounded regions defined by the referenced object, while in [PTS94], they define the predicates using points of the two objects. Second, we consider only rectangle objects and assume that the interiors of two objects do not intersect. Third, the predicates we defined here are pairwise exclusive, meaning that if two spatial objects satisfy one predicate, they do not satisfy all other predicates. However, in [PTS94], two objects may satisfy several predicates at the same time.

A direction join is a join with a conjunction of direction predicates as the join condition.

Example 2.1 Consider a database with the following relations: *beaches*, *hotels*, and *camp_sites*. Each relation has a spatial attribute specifying the actual locations. Then the query “find all hotels in the west of a beach with rate lower than 100 dollars” requires a join of *hotels* with *beaches* based on their locations and relative directions. As another example, the query “find all beaches with camp site in the east and hotel in the north” would require a join of all three relations and direction specifications of all three locations. The first query involves a two-way direction join with predicate W, while the second involves a 3-way direction joins with three relations and predicates N and E. □

Definition 2.2 Let r_1, \dots, r_m be a sequence of (not necessarily distinct) relations for some positive integer m and φ a conjunction of direction predicates over spatial attributes in r_i 's. An m -way φ -join of r_1, \dots, r_m , denoted by $\bowtie_{\varphi}[r_1, \dots, r_m]$, consists of all tuples (t_1, \dots, t_m) such that $\forall i, t_i \in r_i$ and φ is true.

For example, the direction joins of the two queries in Example 2.1 are expressed as $\bowtie_{H.loc W B.loc} [H, B]$ and $\bowtie_{(C.loc E B.loc) \wedge (H.loc N B.loc)} [H, B, C]$.

An important subclass of 2-way direction joins consists of direction joins with a single direction predicate such as the first example above. In the remainder of the paper, unless otherwise specified, we use the term “2-way direction join” to mean joins in this subclass.

We also assume that b is the page size throughout the paper as our interest is on I/O complexity.

3 A Sweeping Technique for 2-Way Direction Join Evaluation

In this section, we develop a technique for 2-way direction joins with one predicate. The technique combines plane sweeping [SH76] and I/O efficient data structures such as B-trees and external priority search trees [ASV99]. Using this technique we show that each 2-way direction join can be computed within $O(N \log_b \frac{N}{M} + k)$ I/Os, where N is the size of the relations, M is the size of the memory, and k is the number of rectangle pairs satisfying the direction predicate (the result size).

Let r, s be two relations with spatial attributes A, B (resp.) and θ a direction predicate. The 2-way direction join $\bowtie_{r.A\theta s.B}[r, s]$ returns all pairs of tuples $t \in r, t' \in s$ such that $(t.A\theta t'.B)$ is true. The main idea of evaluating the join is to use the plane sweeping technique. The technique is to sweep a horizontal (or vertical) line across the plane in a predetermined direction (depending on the predicate θ). At each position of the sweep line, a rectangle can be in exactly one of the following three states: *sleeping* if the sweep line has not reached it yet, *active* if the sweep line intersects it, and *dead* when the sweep line has passed it. During the join evaluation, according to the direction predicate, dead objects will be maintained and appropriate operations are performed to find all pairs of rectangles that satisfy the direction predicate.

In the following, we illustrate this join evaluation technique with the predicate \mathbf{N} , and then generalize the technique to handle other predicates.

For simplicity, let r and s be two sets of rectangles. We compute $\bowtie_{r, \mathbf{N}_s}[r, s]$ as follows. A horizontal sweep line is used in this case. Initially the line is located above all rectangles in r and s . Since the predicate is \mathbf{N} , we keep track of all rectangles in r that are above the current sweep line, i.e., all dead rectangles in r , in the *dead set* of r . During the computation, the sweep line moves down. When it leaves the lower boundary of a rectangle t in r , the state of t changes from active to dead and t is inserted into the dead set. When the sweep line encounters the upper boundary of a rectangle t' in s , a search is performed to find all rectangles in r that are in the north of t' . Clearly, a rectangle in r is in the north of t' if it is currently dead and its projection on the x axis (x -projection) is contained in the x -projection of t' . Therefore, the search to be performed is an interval containment search on the x -projections of all rectangles in the dead set. This interval containment search problem can be reduced into a 2-sided 2-dimensional range search problem as follows. We view each x -projection (an interval) as a 2-dimensional point with the lower, upper bounds as the x, y

coordinates (resp.). Then finding all intervals contained in an interval $[u, v]$ can be done by finding all points in the right half plane of $x = u$ and the bottom half plane of $y = v$ (including the lines). The sweep line keeps moving down and the insertion and search operations are performed for rectangles in r and s (resp.) until the sweep line passes all upper boundaries of rectangles in s .

Example 3.1 Two rectangle sets $r = \{t_1, t_2, t_3, t_4\}$ and $s = \{t'_1, t'_2\}$ are shown in Fig. 2. At the beginning, the sweep line is located above the upper boundary of t_4 . When the line reaches the upper bound of t'_2 , t_2, t_3, t_4 are dead and thus in the dead set. The search for intervals contained in the x -projection of t'_2 reports t_3 and t_4 which are in the north of t'_2 . □

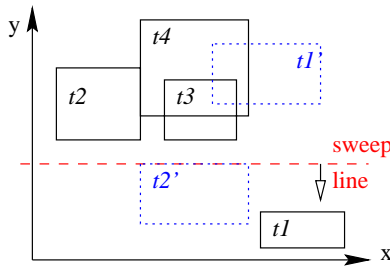


Fig. 2. Direction join r north s

The evaluation of direction join with other predicates are similar. Basically, they all use plane sweeping technique and perform insertion or search operations when the sweep line encounters the boundary of an input rectangle. However, there are some key differences summarized in the following.

Sweep line moving direction Depending on the predicates, the sweep line can move horizontally from top down or bottom up, or it can move vertically from left to right or from right to left. For example, in the evaluation for join with the predicate W (west) the sweep line is a vertical line and moves from left to right.

Search operations In order to find join result, certain search operations are performed. For predicates N, S, E, W, and all tri-direction predicates, these search operations are interval containment searches which can be transformed into 2-sided 2-dimensional range searches. For predicates NW, NE, SW, and SE, the search operations look for all x -projections of dead rectangles that lie in the left of the x -projection of the rectangle under consideration. These can be easily implemented as 1-dimensional range searches on the right endpoints of x -projections of all dead rectangles. For bi-direction predicates, the search operations are more complicated. These operations search for all x -projections of dead rectangles whose right endpoints are within the x -projection of the rectangle t under consideration, and whose left endpoints are in the left of the x -projection of t . We can view x -projections $[u, v]$ of

Table 3. Summary of 2-way direction join algorithms

<i>predicates</i>	<i>sweep line movement</i>	<i>range search</i>		<i>data</i>
		dimension	sides	structure
NE	↓ or ←	1	1	B-tree
SE	↑ or ←			
SW	↑ or →			
NW	↓ or →			
E, weak-E	←	2		external priority search tree
S, weak-S	↑			
W, weak-W	→			
N, weak-N	↓			
NE-E, SE-E	←	2	3	
SE-S, SW-S	↑			
NW-W, SW-W	→			
NE-N, NW-N	↓			

dead rectangles as 2-dimensional points (u, v) , and transform the search operation into a 3-sided 2-dimensional search for $u' < v < v' \wedge u < u'$, where $[u', v']$ is the x -projection of the rectangle under consideration.

In summary, the search operations in all evaluations can be transformed into either 1 sided range searches for 1-dimensional points, or 2 sides or 3 sided range searches for 2-dimensional points.

The direction joins algorithms for different predicates are summarized in Table 3. In the table, column 2 shows the sweep line movements, where ↓ means top-down movement, ↑ means bottom-up movement, ← means left-to-right movement, and → means right-to-left movement. Column 3 provides the dimension and side information of the range search operations in the algorithms. Column 4 shows the temporary data structures used in the algorithms which will be explained later in this section.

We now consider the I/O complexity of these join algorithms. Again we use the join algorithm for predicate N as an example. A key component affecting the analysis is the insertion and the search operations on the dead set of r . To achieve I/O efficiency, the data structure for the dead set must be carefully designed. In order to search, the dead set must maintain the x -projections of dead rectangles in r . As we discussed before, the search operation can be reduced into a 2-sided 2-dimensional range search problem. Fortunately, the external priority search tree of [ASV99] provides an I/O efficient solution for the 2-sided searching problem. The external priority search tree is an external version of priority search tree [McC85]. It is used to store 2-dimensional points. An external priority search tree consists of a weight-balanced B-tree [AV96] built on the x -coordinates of all points and auxiliary data structures associated with nodes in the tree which are based on y -coordinates of the points. It was shown in [ASV99] that an external priority search tree occupies $O(\frac{N}{b})$ pages and it performs insertions in $O(\log_b N)$ I/Os and 2-sided range searches in $O(\log_b N + k)$ I/Os, where N is the number of points, and k is the number of points in the search result. We can further show that with a memory of size M , the top $O(\log_b M)$ levels of the tree can be

stored in the memory. Thus the I/O complexity of insertion operations can be lowered to $O(\log_b N - \log_b M) = O(\log_b \frac{N}{M})$ and the ones for range search can be reduced to $O(\log_b \frac{N}{M} + k)$.

For evaluations with other predicates, if the evaluation requires 2-sided or 3-sided 2-dimensional range search, external priority search trees are used to maintaining the temporary data structure. If the evaluation needs only 1-dimensional range searches, B-trees are sufficient for the temporary data structure. Column 4 of Table 3 shows the data structures used in the evaluation of a direction join.

Theorem 3.2 Let r, s be N -tuple relations with spatial attributes A, B (resp.), θ a direction predicate, and M the size of the memory. The join $\bowtie_{r.A\theta s.B} [r, s]$ can be computed in $O(N \log_b \frac{N}{M} + k)$ I/Os, where k is the number of tuples in the result.

Proof. (Sketch) We prove for the case of 2-way direction joins with the predicate N ; cases for other predicates are similar.

We use the algorithm described above to evaluate the join $\bowtie_{r.AN s.B} [r, s]$. The algorithm sweeps through all rectangles in $r.A$ and $s.B$ and perform actions when each rectangle encountered. The sweep accesses each rectangle in $r.A$ and $s.B$ only once, The number of I/Os for this phase is $O(\frac{N}{b})$. For each rectangle in $r.A$ (or $s.B$), an insertion operation (resp. a search operation) on the dead set for r is performed. The dead set contains at most N points during the execution, one for each dead rectangle in $r.A$. The dead set is stored in an external priority search tree. It is shown in [ASV99] that the insertion operation on an external priority search tree takes $O(\log_b N)$ I/Os and a range search operation takes $O(\log_b N + k')$ I/Os, where k' is the number of rectangles in $r.A$ that are in the north of the current rectangle in $s.B$ (encountered by the sweep line). Since the memory size is M , we store the top levels of the tree in memory, the number of I/Os required for an insertion can then be reduced to $O(\log_b \frac{N}{M})$ and the I/Os for range search can be reduced to $O(\log_b \frac{N}{M} + k')$. Let k be the size of the join result. All executions of insertion and search operations thus take $O(N \log_b \frac{N}{M} + k)$ I/Os.

In conclusion, the join $\bowtie_{r.AN s.B} [r, s]$ can be evaluated in $O(N \log_b \frac{N}{M} + k)$ I/Os. □

There is an important property of the algorithms for 2-way direction join with one predicate. This property is critical for the algorithms to be described in the next section. We state the property below.

Lemma 3.3 Let r, s be two relations with spatial attributes A, B (resp.), and $\bowtie_{r.A\theta s.B} [r, s]$ a 2-way direction join. The result of the join (evaluated by the algorithms described above) is grouped by tuples in s , the relation in the right argument of the join predicate.

For example, let $r = \{t_1, t_2, t_3\}$ and $s = \{t'_1, t'_2\}$ be two sets of rectangles. Assume that t_1 and t_2 are in the north of t'_1 and t_1, t_3 are in the north of t'_2 . The

result of $\bowtie_{rN_s} [r, s]$ generated by the algorithm described above is an ordered list grouped by tuples in s . For instance, it could be $\{(t_1, t'_2), (t_3, t'_2), (t_1, t'_1), (t_2, t'_1)\}$.

In a join predicate, we term the object (relation) occurring before the direction predicate symbol as the *left* object (relation) and the one after as the *right* object (relation). In the algorithms (e.g., for N), we can see that during the evaluation of 2-way direction joins, the search operations are only performed when the sweep line reaches the rectangles in the right relation. Indeed, for every tuples t in the right relation, the search operation find *all* tuples in the left relation that satisfy the direction predicate with t . Thus the join result is grouped by tuples in the right relation. This is an important observation because it is used in the next section to improve I/O efficiency of a subclass case of multi-way direction joins.

4 An Efficient Algorithm for Star Joins

In this section, we consider a subclass of multi-way direction joins, called “star joins”, in which all join predicates involve one common relation (called the “center relation”). We show that each m -way star join of N -tuple relations can be evaluated in $O(mN \log_b \frac{N}{M} + K + k)$ I/Os, where M is the size of the memory, $K (\leq mN^2)$ is the size of the intermediate result, and $k (\leq N^m)$ the size of the join result.

The primary difficulty of evaluating a multi-way join is to efficiently manage intermediate result and control its size. Since in a star join the center relation connects to every other relation in the join condition, this suggests the approach of decomposing a star join into a collection of 2-way joins followed by a merge join (non spatial) of all results. However, a direct application of the sort-merge method will raise the I/O complexity to $O(mN^2 \log N)$ since the result of a 2-way join may have a size up to N^2 . This is significantly higher (by a factor of N) than the cost of our algorithm. A key step in our algorithm is to guarantee that the results of 2-way joins are always sorted using several new techniques so that the results of two-way joins can be pipelined to the merge phase.

Definition 4.1 Let r_1, \dots, r_m be a sequence of relations and $1 \leq j \leq m, m > 2$. An m -way φ -join $\bowtie_{\varphi}[r_1, \dots, r_m]$ is a *star join with center relation* r_j if each join predicate in φ involves a spatial attribute of r_j and a spatial attribute of r_i ($i \neq j$), and each relation r_i ($i \neq j$) appears only once in φ .

The second query in Example 2.1 contains a star join with *beaches* as the center relation. The main result of this section is now stated below.

Theorem 4.2 Each star join $\bowtie_{\varphi}[r_1, \dots, r_m]$ with center relation r_j ($1 \leq j \leq m$) can be evaluated in $O(mN \log_b \frac{N}{M} + K + k)$ I/Os, where N is the number of tuples in relations r_i 's, M is the size of the memory, $K \leq mN^2$ is the size of the intermediate result, and $k \leq N^m$ is the size of the join result.

To compute a star join we decompose the join into a collection of 2-way direction joins and then perform a (non-spatial) merge join of all the results. A

naive way is to use the nested loop method for each 2-way join with the center relation as the outer relation in the nested loops. The intermediate results will be sorted by tuples in the center relation and a straightforward merge will produce the final result. However, for each 2-way join it takes $O(N^2)$ I/Os no matter what the size of the intermediate result is. Obviously, this is not desirable. To improve the I/O performance, we apply and extend the plane sweeping based technique developed in Section 3 to evaluate those 2-way joins. We also develop techniques to guarantee that the intermediate results are sorted based on the center relation. This eliminates sorting.

In Section 4.1, we first consider a special case where the center relation of the star join is always the right relation in the join condition. In this case, the 2-way direction join evaluation strategy presented in Section 3 is used to evaluate each 2-way join and the intermediate results are stored as sorted on the tuple identifiers (tids) of tuples in the center relations before the merge. In Section 4.2 we further develop several techniques to eliminate this limitation and thus prove Theorem 4.2. Finally, in Section 4.3 we extend the algorithm for star joins to evaluate multi-way direction joins that are star connections of star joins.

4.1 Star Joins with Center as the Right Relation

Lemma 4.3 Let $\bowtie_\varphi[r_1, \dots, r_m]$ be a star join with center r_j such that r_j is always the right relation in φ . Then the join can be computed in $O(mN \log_b \frac{N}{M} + K + k)$ I/Os, where N is the maximal number of tuples in each r_i , M is the size of the memory, $K \leq mN^2$ the intermediate result size, and $k \leq N^m$ the join result size.

Let r_1, \dots, r_m be a sequence of relations with spatial attributes, and $\bowtie_\varphi[r_1, \dots, r_m]$ a star join with center r_j . Without loss of generality, let $\varphi = \bigwedge_{1 \leq i \leq m, i \neq j} (r_i.A_i \theta_j r_j.B_i)$, $1 \leq i \leq m, i \neq j$, where θ_j is a direction predicate, and A_i, B_i spatial attributes of relations r_i and r_j , resp.

To evaluate the star join, we first decompose it into $(m-1)$ 2-way direction joins $\bowtie_{r_i.A_i \theta_j r_j.B_i}[r_i, r_j]$, $1 \leq i \leq m, i \neq j$. Each of the 2-way direction joins is evaluated using an appropriate algorithm in Section 3. Let K_i be the number of pairs of tuples in r_i and r_j that satisfy the direction predicate, i.e., the intermediate result generated by the 2-way join, and let $K = \sum_{1 \leq i \leq m, i \neq j} K_i$. Let M be the size of the memory. We use $\frac{M}{2}$ amount of the memory for the evaluation of each 2-way join. By Theorem 3.2, each 2-way direction join takes $O(N \log_b \frac{N}{M} + K_i)$ I/Os. The evaluation of all 2-way joins takes $O(mN \log_b \frac{N}{M} + K)$ I/Os. Here $K_i \leq N^2$ and thus $K \leq mN^2$.

To reduce the I/O cost in merging the intermediate results, we want the intermediate result to be sorted based on the center relation. By Lemma 3.3, the results of these 2-way direction joins are grouped by tuples in the right relation, i.e., the center. Therefore, we can maintain each 2-way join result as sorted on the tids of tuples in the center r_j . To do this, we store all intermediate results of the 2-way joins in a single B-tree, denoted by T , based on tids of tuples in r_j . Each entry for tuple t in the leaf nodes of T is associated with $(m-1)$ lists L_i , $1 \leq i \leq m, i \neq j$. Each list L_i contains tids of tuples in r_i whose

spatial attributes satisfy the corresponding direction predicate with respect to t . During the evaluation of a 2-way direction join between r_i and r_j , for every tuple t in r_j , a search operation is performed to find all tuples t' in r_i such that the spatial attributes of t' and t (resp.) satisfy the direction predicate. The result of the search operation is inserted into list L_i associated with t in T . Because there are only N tuples in r_j , the tree part of T takes at most $O(N)$ pages. We store the top $\log_b \frac{M}{2}$ levels of the tree in the remaining $\frac{M}{2}$ memory. An insertion for tuple t in r_j on T then takes $O(\log_b \frac{N}{M} + k_t)$ I/Os, where k_t is the number of tuples in r_i that satisfy the direction predicate with respect to t . There are totally $O(mN)$ search operations for all 2-way joins. Thus all executions of the insertion operations on T requires $O(mN \log_b \frac{N}{M} + K)$ I/Os.

When all 2-way direction joins are evaluated, the intermediate results are stored in the lists associate with leaf entries in T . We sequentially scan all these entries. For each entry t , a cross product of tuples in the $(m-1)$ nonempty lists associated with t is executed to form tuples in the join result that contains t . Let k be the size of the join result. It is easy to see that only $O(K + k)$ I/Os are required for this step. In the worst case, k could be N^m . Therefore, the total number of I/Os required by the star join evaluation is $O(mN \log_b \frac{N}{M} + K + k)$, where $K \leq mN^2$ and $k \leq N^m$.

4.2 Star Joins

We now consider star joins in general where the center may occur as the left relations. Let r_1, \dots, r_m ($m > 1$) be a sequence of relations with spatial attributes, and $\bowtie_{\varphi}[r_1, \dots, r_m]$ a star join with center r_j . Similar to the preceding case, we decompose the star join into $(m-1)$ 2-way direction joins and then merge their results. In order to reduce the cost of sorting on the intermediate results, it is desirable to group all intermediate results based on tuples in r . However, when the center occurs as the left relation, this is not guaranteed. We introduce below two techniques to handle cases where r_j is the left operand in a join predicate so that the result is grouped by tuples in r_j again.

1. If there is an expression $r_j.B_i \theta r_i.A_i$ in φ and θ is NW, NE, SW, or SE, we convert the expression $r_j.B_i \theta r_i.A_i$ to an equivalent one $r_i.A_i \theta' r_j.B_i$ by swapping the arguments. We can then evaluate the 2-way join $\bowtie_{r_i.A_i \theta' r_j.B_i}[r_i, r_j]$. The result is grouped by tuples in the center relation r_j and can be used in the merging stage.
2. For other predicates (N, S, W, E, weak-N, weak-S, weak-W, and weak-E), if there is an expression $r_j.B_i \theta r_i.A_i$ in φ , we use modified versions of the algorithms in Section 3 to evaluate the corresponding 2-way joins. The results of the modified algorithms are grouped based on tuples in r_j (the center) in stead of the right relation.

Below we discuss each technique in detail.

Let $r_j.B_i \theta r_i.A_i$ ($1 \leq i \leq m, i \neq j$) be an predicate in φ , where B_i and A_i are spatial attributes of r_j and r_i , resp. If θ is one of NW, NE, SW, or SE, we

transform $r_j.B_i\theta r_i.A_i$ into an equivalent expression $r_i.A_i\theta' r_j.B_i$ based on the following observations. We can then use the algorithms in Section 3 to evaluate the 2-way join $\bowtie_{r_i.A_i\theta' r_j.B} [r_i, r_j]$. The results will be grouped by tuples in r_j .

Lemma 4.4 For all rectangles o and o' , $(o \text{ NW } o') \equiv (o' \text{ SE } o)$, and $(o \text{ NE } o') \equiv (o' \text{ SW } o)$.

If in the join predicate φ there is an expressions $r_j.B_i\theta r_i.A_i$, where B_i and A_i are spatial attributes of r_j and r_i (resp.), and θ is one of the direction predicates N, S, W, E, weak-N, weak-S, weak-W, or weak-E, we evaluate the corresponding 2-way direction joins using modified versions of the algorithms in Section 3. The approach is still based on plane sweeping, but the results of the join will be grouped by tuples in r_j (the left relation) instead of tuples in r_i (the right relation). We use the predicate N as an example to highlight the necessary modifications.

The modified evaluation of the 2-way direction join $\bowtie_{r_j.B_i \mathbf{N} r_i.A_i} [r_j, r_i]$ differs from the one we described in Section 3 in three parts. For simplicity, let r and s be two sets of rectangles corresponding to r_j and r_i , resp. The differences are summarized in the following.

1. Although the sweep line is still a horizontal line, it moves from bottom up in stead of top down.
2. During the sweeping, instead of maintaining rectangles in r (the left relation) in an intermediate data set, rectangles in s (the right relation) are maintained.
3. Although a 2-d 2-sided range search operation is performed when the sweep line reaches the lower boundary of a rectangle t in r (the left relation), the search is different from the search performed in the original algorithm. The modified search looks for rectangles t' in the dead set of s whose x -projections containing the x -projection of t .

The following example illustrates the modifications.

Example 4.5 Consider two sets of rectangles r, s and the same join $\bowtie_{r \mathbf{N} s} [r, s]$ as in Example 3.1. In the modified algorithm, a horizontal sweep line is used and it moves from bottom up. When the sweep line reaches the upper boundary of $t'_2 \in s$, t'_2 is inserted into the dead set for s . When the sweep line moves to the lower boundary of $t_3 \in r$, t_3 is in the north of t'_2 , which is found by the search operation performed at this point. □

We can easily show that the modified algorithm still has I/O complexity $O(N \log_b \frac{N}{M} + k)$, where N is the size of r_j , M the size of the available memory, and r_i , and k is the size of the join result. More importantly, the join result is now grouped based on tuples in the left relation. Algorithms of 2-way direction joins with predicates S, E, W, weak-N, weak-S, weak-W, and weak-E can be modified quite similarly. The I/O complexity of these modified algorithms remains the same as that of the original ones and the results of the joins become grouped based on tuples in the left relation (i.e., r_j).

Now we summarize the evaluation of a general star join. Let r_1, \dots, r_m ($m > 1$) be a sequence of relations with spatial attributes, and $\bowtie_\varphi[r_1, \dots, r_m]$ a star join with center r_j . Using the techniques discussed above, we evaluate the star join in the following three steps:

1. If there are subexpressions of form $r_j.B_i\theta r_i.A_i$ in φ , where θ belongs to $\{\text{NW, NE, SW, SE}\}$, we convert them into equivalent ones using Lemma 4.4. Let the resulting join condition be φ' .
2. We then decompose the star join with condition φ' into 2-way direction joins, and evaluate them using either the algorithms presented in Section 3 or their modified versions. The intermediate results of these 2-way direction joins are stored in a B-tree on tids of tuples in the center relation, as in the special case.
3. Finally, we merge all results of the 2-way joins according to the center relation.

Similarly, we can show that this algorithm requires $O(mN \log_b \frac{N}{M} + K + k)$ I/Os, where $K \leq mN^2$ is the size of the intermediate result, and $k \leq N^m$ is the size of the join result. This concludes the discussion on the proof of Theorem 4.2.

In the above star join evaluation, the center relation is accessed $(m-1)$ times in all 2-way direction joins. This can be further improved when these 2-way joins involves the same spatial attribute of the center relation. For example, we can perform several 2-way joins in a single plane sweeping, with the sweep line moving in the same direction. This is possible since we can rotate the plane(s) properly to have a common sweep line orientation and moving direction. However, this improvement has the same big- O complexity.

4.3 Star Connections of Star Joins

The techniques for computing star joins we described above can be further extended to evaluate a more general subclass of multi-way direction joins.

Let Q be an m -way direction join ($m > 2$), Q is a d -connection of star joins ($d < m$) if Q consists of $(d + 1)$ pairwise disjoint (except the centers) star joins with the $(d + 1)$ centers forming another $(d + 1)$ -way star join. Suppose that each relation in the join Q has N tuples. Let M be the size of the memory. We show below that the join Q can be evaluated in polynomial time using an extension of the algorithms for star join evaluation. And the I/O complexity of the evaluation is $O(dN^2 \log_b \frac{dN}{M} + mN \log_b \frac{N}{M} + dmN + K + k)$ where $K \leq mN^2$ is the size of the intermediate result and $k \leq N^m$ the size of the join result.

The idea is to evaluate the join Q as $(d + 1)$ star joins and combine their results together. From the discussions in Sections 4.1 and 4.2, we know that for each star join, it takes $O(m_iN \log_b \frac{N}{M} + K_i + k_i)$, $0 \leq i \leq d$, where K_i is the size of the intermediate result of the star join and m_i is the number of relations. In the worst case, K_i could be m_iN^2 . All these intermediate results are stored in B-trees as we described in Sections 4.1 and 4.2. Let them be T_0, \dots, T_d , resp.

The next step of the evaluation will merge these intermediate results. A naive approach would generate the result of each star join and merge them to form the

final result of Q . However, this approach requires the results of star joins to be stored somewhere. Since the result of a star join may be exponential, i.e., N^{m_i} , $0 \leq i \leq d$, it is not feasible. To overcome this problem, we use the intermediate results of star joins (of size up to $m_i N^2$) directly to form the final result of Q , thus avoiding the generation of the star join results. In the following we sketch the method.

Suppose T_0, \dots, T_d are the intermediate trees for the $(d + 1)$ star joins in Q and w.l.o.g. let r be the center of the star join over the centers and T_0 the tree for r . To generate the final join result, we sequentially scan all entries in the leaf nodes of T_0 . Each entry corresponding to t in r_0 is associated with lists L_i 's containing tuples in r_i 's (resp.) that satisfy the join predicates with respect to t . According to the algorithm for star joins, a cross product of t and all tuples in these lists will general tuples in the star join result that contains t .

In order to generate tuples for the final result of Q , for each tuple t' in some list L_i , if r_i is a center of another star join, there is a tree, say T_i , storing the intermediate result of this star join. We search t' in the B-tree T_i . Similarly, in T_i there are lists $L_{i,j}$'s that represents the resulting tuples of the star join. By scanning through these lists (if they are not empty) portions of the final results of Q with respect to tuples t, t' can be generated. In this way, the merging does not need any sorting.

We now consider the I/O complexity of the evaluation. Let $K \leq mN^2$ be the size of the intermediate results of all star joins. The evaluation of $(d + 1)$ star joins without generating their results takes $O(mN \log_b \frac{N}{M} + K)$ I/Os. We can then divide the memory into d parts, each part storing the top $\log_b \frac{M}{d}$ of the intermediate trees for the d star joins T_1, \dots, T_d in Q . Then the merge step for each tuple t in the center relation r takes $O(dN \log_b \frac{dN}{M} + dm + k_t)$ I/Os, where k_t is the number of tuples in the final join result that contains t . The entire merge step (of all tuples) then takes $O(dN^2 \log_b \frac{dN}{M} + dmN + k)$ I/Os, where $k \leq N^m$ is the size of the join result.

Therefore, the I/O complexity of evaluating an m -way d -connection of star joins is

$$O(dN^2 \log_b \frac{dN}{M} + mN \log_b \frac{N}{M} + dmN + K + k)$$

where $K \leq mN^2$ is the size of total intermediate results, and $k \leq N^m$ the size of the join result.

5 Hardness of Multi-way Direction Joins

In this section, we consider the complexity of evaluating a multi-way direction join. Specifically, we study the problem of testing if the result of a given multi-way join with direction predicates of given relations is empty. We show that this problem is NP-complete in general and surprisingly even for the case when (1) only one (spatial) attribute from each relation is involved in the join condition, and (2) each (spatial) attribute occurs a bounded number of times in the

join condition. The result immediately implies that the multi-way direction join evaluation problem is intractable.

Let r_1, \dots, r_m be a sequence of relations and $Q = \bowtie_{\varphi}[r_1, \dots, r_m]$ an m -way join with direction predicates. Q is *single attribute* if for each $1 \leq i \leq m$, r_i has at most one attribute participating in the join condition φ . Q has *degree* ℓ for some positive integer ℓ if for each $1 \leq i \leq m$, each attribute from r_i occurs at most ℓ times in φ . A collection of multi-way joins is *bounded* if all joins in it have a constant degree. The *join emptiness* problem is to test if the result of Q on r_1, \dots, r_m is empty.

Theorem 5.1 The join emptiness problem is NP-complete for m -way direction joins. Furthermore, it remains NP-complete even for m -way direction joins that are single attribute and bounded.

It is obvious that the join emptiness problem is in NP. To establish the NP-hard result, we first show that the join emptiness problem is NP-hard for single attribute m -way joins, and then extend the result to single attribute bounded m -way joins.

Let G be an (undirected) graph and k a positive integer as the input to the k -clique problem. Without loss of generality, we assume that each edge in G is incident to two distinct vertices and the vertices in G are integers $1, 2, \dots, n$. Suppose $0 \leq i, j \leq n$ are two integers. We denote by $\square_{(i,j)}$ a square of some small, fixed size (e.g. 0.1×0.1) centered at the point (i, j) . We construct four unary relations R, V, B, L with a single spatial attribute A as follows:

$$\begin{aligned}
 R &= \{\square_{(i,j)} \mid i < j, (i, j) \in G \text{ or } (j, i) \in G\} & B &= \{\square_{(i,0)} \mid 1 \leq i \leq n\} \\
 V &= \{\square_{(i,i)} \mid 1 \leq i \leq n\} & L &= \{\square_{(0,i)} \mid 1 \leq i \leq n\}
 \end{aligned}$$

Fig. 4 (left) illustrates the relations R, V, B, L on the plane.

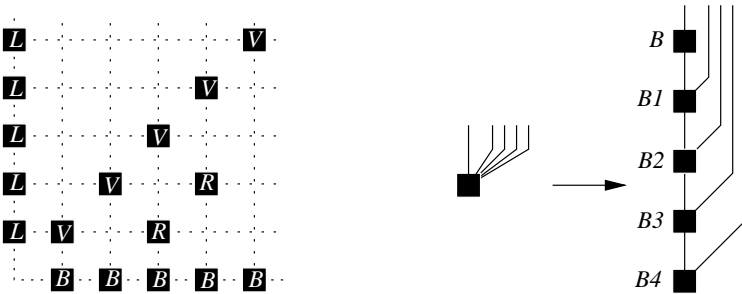


Fig. 4. Illustration of the reduction

We now construct the join expression Q . Essentially, Q joins R for $\frac{k(k-1)}{2}$ times (the number of edges in a k -clique), and joins each of V, B, L for k times (the number of vertices in a k -clique). For simplicity, we use subscripts V_i, B_i, L_i for occurrences of V, B, L and subscripts $R_{i,j}$ for occurrences of R where $1 \leq i \leq n$ and $1 \leq j < i$. We describe the join condition φ of Q as follows.

1. For each $1 \leq i \leq n$, φ contains the condition $(B_i.A \text{ S } V_i.A) \wedge (L_i.A \text{ W } V_i.A)$. Intuitively, V_i represents a vertex in a k -clique and if this condition is true, $B_i.A$ and $L_i.A$ all represent the same vertex.
2. φ also contains the join predicate $(L_i.A \text{ S } L_{i+1}.A)$ for each $1 \leq i \leq (n - 1)$. These conditions are to ensure that the vertices represented by L_i 's (and thus V_i 's and B_i 's) are pairwise distinct.
3. For each $1 \leq i \leq n$ and each $1 \leq j < i$, the join condition φ contains the condition $(R_{i,j}.A \text{ N } B_i.A) \wedge (R_{i,j}.A \text{ E } L_j.A)$. Roughly, $R_{i,j}$ is to ensure the existence of an edge between vertices i and j . The above condition does this by checking if $R_{i,j}$ lines up vertically with B_i and horizontally with L_j .

To complete the reduction, we need to show that G has a clique of size k if and only if Q is not empty on the input. For the only if direction, suppose G has a k -clique with vertices ℓ_i ($1 \leq i \leq k$). We pick the tuples $\square_{(\ell_i, \ell_i)}$ from V_i , $\square_{(0, \ell_i)}$ from L_i , $\square_{(\ell_i, 0)}$ from B_i , and $\square_{(\ell_i, \ell_j)}$ from $R_{i,j}$. It is easy to verify that these tuples will generate an output tuple for the join. For the other direction, suppose the join Q contains one tuple t in the output. From the construction of the join condition φ , it must be the case that V_i 's must be distinct and along with B_i, L_i represent k distinct x, y coordinates (vertices). The conditions on $R_{i,j}$ will ensure that these vertices are pairwise connected by an edge. This establishes the single attribute case.

For the bounded case, we note that each of the relations V_i 's and $R_{i,j}$'s occurs twice in the join condition of the above reduction. Only B_i 's and L_i 's may occur up to $k + 1$ times. Let $c \geq 3$ be the bound on the number of occurrences of an attribute in a join condition. For each $1 \leq i \leq k$, we use $\frac{k+1}{c-1}$ different relations B_i^j to represent the same vertex i so that each attribute of each relation (occurrence) does not appear more than 3 times in the join condition. These relations contain small squares in the south of B_i . Fig. 4 (right) shows an example of applying this technique, where each line represents an occurrence of B in the join condition. Similar steps are done for L_i 's. It is clear that in the result join condition, each attribute occurs for no more than 3 times. It is easy to argue that this is indeed a reduction.

6 Conclusions

In this paper, we study multi-way spatial joins with direction predicates. The NP-complete lower bound results reported in this paper show that multi-way spatial joins are at least as hard as the relational version of the problem, even in very simple cases. It is conceivable that the complexity lower bound for spatial joins with other predicates (such as region intersection etc.) is not lower. We show that for a subclass of multi-way joins the worst case upper bound can be lowered to polynomial time. However, it remains an interesting open problem to find other subclasses of spatial joins in polynomial time.

References

- [APR⁺98] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. Vitter. Scalable sweeping-based spatial join. In *Proc. Int. Conf. on Very Large Data Bases*, 1998.
- [Arg95] L. Arge. The buffer tree: A new technique for optimal I/O-algorithms. In *Proc. Workshop on Algorithms and Data structures*, 1995.
- [ASV99] L. Arge, V. Samoladas, and J. S. Vitter. On two-dimensional indexability and optimal range search indexing. In *Proc. ACM Symp. on Principles of Database Systems*, 1999.
- [AV96] L. Arge and J. S. Vitter. Optimal dynamic interval management in external memory. In *Proc. IEEE Symp. on Foundations of Computer Science*, 1996.
- [BHF93] L. Becker, K. Hinriches, and U. Finke. A new algorithm for computing joins with grid files. In *Proc. Int. Conf. on Data Engineering*, pages 190–197, 1993.
- [BKS93] T. Brinkhoff, H-P. Kriegel, and B. Seeger. Efficient processing of spatial join using R-trees. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1993.
- [BKSS94] T. Brinkhoff, H-P. Kriegel, R. Schneider, and B. Seeger. Multi-step processing of spatial joins. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1994.
- [Dut89] S. Dutta. Qualitative spatial reasoning: a semi-qualitative approach using fuzzy logic. In *Proc. Symp. on Large Spatial Databases*, 1989.
- [Fra92] A. U. Frank. Qualitative spatial reasoning about distances and directions in geographic space. *Journal of Visual Languages and Computing*, 3:343–371, 1992.
- [GTVV93] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. External-memory computational geometry. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 714–723, 1993.
- [Gün93] O. Günther. Efficient computation of spatial joins. In *Proc. Int. Conf. on Data Engineering*, pages 50–59, 1993.
- [Her93] D. Hernandez. Maintaining qualitative spatial knowledge. In *Proceedings of the European Conference on Spatial Information Theory*, 1993.
- [HJR97] Y.-W. Huang, N. Jing, and E. A. Rundensteiner. Spatial joins using R-trees: Breadth-first traversal with global optimizations. In *Proc. Int. Conf. on Very Large Data Bases*, 1997.
- [HS98] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 237–248, 1998.
- [LR96] M-L. Lo and C. V. Ravishankar. Spatial hash-joins. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1996.
- [McC85] E. M. McCreight. Priority search trees. *SIAM Journal of Computing*, 14:257–276, 1985.
- [MP98] N. Mamoulis and D. Papadias. Constraint-based algorithms for computing clique intersection joins. In *Proc. ACM Symp. on Advances in Geographic Information Systems*, 1998.
- [MP99] N. Mamoulis and D. Papadias. Integration of spatial join algorithms for processing multiple inputs. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1999.

- [PD96] J. M. Patel and D. J. DeWitt. Partition based spatial-merge join. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1996.
- [PMT99] D. Papadias, N. Mamoulis, and Y. Theodoridis. Processing and optimization of multiway spatial joins using R-trees. In *Proc. ACM Symp. on Principles of Database Systems*, 1999.
- [PRS99] A. Papadopoulos, P. Rigaux, and M. Scholl. A performance evaluation of spatial join processing strategies. In *Proc. Symp. on Large Spatial Databases*, pages 286–307, 1999.
- [PS93] D. Papadias and T. Sellis. The semantics of relations in 2D space using represectative points: spatial indexes. In *Proceedings of the European Conference on Spatial Information Theory*, 1993.
- [PTS94] D. Papadias, Y. Theodoridis, and T. Sellis. The retrieval of direction relations using r-trees. In *Proc. 5th Int. Conf. Database and Expert Systems Applications*, pages 173–182, 1994.
- [Rot91] D. Rotem. Spatial join indices. In *Proc. Int. Conf. on Data Engineering*, pages 500–509, 1991.
- [SA97] J. C. Shafer and R. Agrawal. Parallel algorithms for high dimensional similarity joins for data mining applications. In *Proc. Int. Conf. on Very Large Data Bases*, pages 176–185, 1997.
- [SH76] M. I. Shamos and D. Hoey. Geometric intersection problems. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 208–215, 1976.
- [Vit98] J. Vitter. External memory algorithms: Dealing with massive data. In *Proc. ACM Symp. on Principles of Database Systems*, pages 233–243, 1998.
- [ZS98] G. Zimbrão and J. M. Souza. A raster approximation for processing of spatial joins. In *Proc. Int. Conf. on Very Large Data Bases*, pages 558–569, 1998.
- [ZSI99] H. Zhu, J. Su, and O. H. Ibarra. An index structure for spatial joins in linear constraint databases. In *Proc. Int. Conf. on Data Engineering*, 1999.
- [ZSI00a] H. Zhu, J. Su, and O. H. Ibarra. Extending rectangle join algorithms for rectilinear polygons. In *Proc. Int. Conf. on Web-Age Information Management*, 2000.
- [ZSI00b] H. Zhu, J. Su, and O. H. Ibarra. Towards spatial joins for polygons. In *Proc. Int. Conf. on Statistical and Scientific Database Management*, 2000.