Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information and Software Technology 46 (2004) 739–751
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Multi-way R-tree joins using indirect predicates

Ho-Hyun Park[a], Jun-Ki Min[b,*], Chin-Wan Chung[b], Tae-Gyu Chang[a]

[a]School of Electrical and Electronics Engineering, Chung-Ang University, 221, HukSuk-Dong, DongJak-Ku, Seoul 156-756, South Korea
[b]Department of Electrical Engineering and Computer Science, KAIST 373-1, Kusong-dong, Yusong-gu, Taejon 305-701, South Korea

Received 1 July 2003; revised 11 December 2003; accepted 12 December 2003
Available online 28 February 2004

Abstract

Since spatial join processing consumes much time, several algorithms have been proposed to improve spatial join performance. Spatial join has been processed in two steps, called *filter step* and *refinement step*. The *M-way R-tree join* (MRJ) is a filter step join algorithm, which synchronously traverses M R-trees. In this paper, we introduce *indirect predicates* which do not directly come from the multi-way join conditions but are indirectly derived from them. By applying indirect predicates as well as direct predicates to MRJ, we can quickly remove the minimum bounding rectangle (MBR) combinations which do not satisfy the direct predicates or the indirect predicates at the parent level. Hence we can reduce the intermediate MBR combinations for the input to the child level processing and improve the performance of MRJ. We call such a multi-way R-tree join algorithm using indirect predicates *indirect predicate filtering* (IPF). Through experiments using synthetic data and real data, we show that IPF significantly improves the performance of MRJ.

Keywords: Spatial databases; Spatial join; M-way R-tree join; Indirect predicates

1. Introduction

For the past several years, the research on spatial database systems has actively progressed because the applications using the spatial information such as geographic information systems, multimedia systems, satellite image database and location based service have increased.

The spatial join is a common spatial query type which requires high processing cost due to the high complexity and large volume of spatial data. The spatial join combines entities from data sets into a single entity set whenever the combination satisfies the join condition (e.g. intersect). In general, the join operation is an important and time consuming database query operation since it retrieves information from different data sets based on Cartesian product.

To reduce the overall processing cost, the spatial join is processed in two steps, called the *filter step* and the *refinement step* [5,12]. As shown in Fig. 1(a), the filter step evaluates tuples whether they satisfy the constraints of a given spatial query, using the MBR (Minimum Bounding Rectangle) approximation. We call the result of the filter step *candidate tuples*, which constitute a super set of the exact query result. The refinement step (Fig. 1(b)) examines the candidate tuples using exact computational geometric algorithms [19] to check whether the tuples really satisfy the constraints of the given spatial query. This paper, like most related spatial database literature, focus the query processing on the filter step [2,22].

Example 1. An example of a 3-way spatial join is 'Find all buildings which are adjacent to roads that intersect with boundaries of districts'.

As in Example 1, the multi-way spatial join combines $M (M > 2)$ spatial relations using $M - 1$ or more spatial predicates. Formally, an M-way spatial join can be expressed as follows [11,14]: given M relations $R_1, R_2, \ldots, R_M$ and a query Q, where Q_{ij} is the binary spatial predicate between R_i and R_j, find all M-tuples $\{\langle r_1, r_2, \ldots, r_M\rangle | \forall i, j : r_i \in R_i, r_j \in R_j \text{ and } r_i Q_{ij} r_j = \text{TRUE}\}$. An M-way spatial join can be modeled by a *query graph* whose vertices represent relations and edges represent spatial predicates.

* Corresponding author. Tel.: +82-42-869-5577; fax: +82-42-869-3577.
E-mail addresses: jkmin@islab.kaist.ac.kr (J.-K. Min), hohyun@cau.ac.kr (H.-H. Park), chungcw@islab.kaist.ac.kr (C.-W. Chung), tgchang@cau.ac.kr (T.-G. Chang).

0950-5849/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2003.12.007

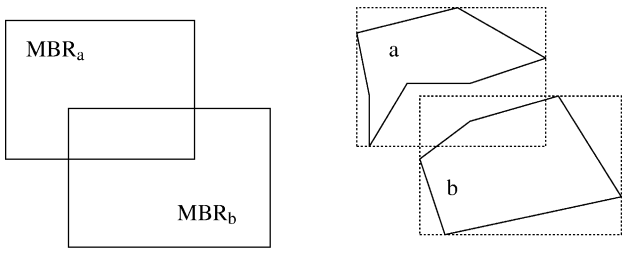


Fig. 1. Spatial join processing steps.

There have been several papers on the multi-way spatial join [10,11,14,15]. One way to process an M-way spatial join is as a sequence of 2-way joins [10]. Another possible way, when all join attributes have spatial indexes and each join attribute is shared among the join predicates connected to the relation (i.e. only one spatial attribute per relation participates in the join), is to scan the relevant indexes synchronously for all join attributes to obtain a set of spatial object identifier (oid) tuples. In the case when the R-trees are used, this is called the *M-way R-tree join* (MRJ) which is considered as a generalization of the *2-way R-tree join* [2,7].

MRJ is known to be more efficient than a sequence of 2-way joins in the case of a dense query graph, high density of data and a small range of M [11]. Furthermore, MRJ has several advantages over a sequence of 2-way joins. Among the following, the third advantage will be explained later in Section 2.2:

- It does not create intermediate results.
- It is appropriate to an interactive query and browsing environment because it generates the first output as soon as the algorithm starts [11].
- It avoids unnecessary refinement operations for some object pairs.

To improve the performance of MRJ, several algorithms have been proposed [10,11,14,15]. Generally, these techniques only considered the join ordering among relations or tuples. However, in this paper, we introduce *indirect predicates*, which do not directly come from the multi-way join conditions but are indirectly derived from them. By applying these extra join conditions (i.e. indirect predicates), we can quickly remove the false results. We call such a multi-way R-tree join algorithm using indirect predicates *indirect predicate filtering* (IPF). The contributions of our work are as follows:

- We propose three kinds of IPF called *domain max information* (DMAX), *node max information* (NMAX), and *entry max information* (EMAX).
- The dynamic maintenance algorithms of DMAX/NMAX/EMAX with the insertion and deletion of R-trees are introduced.
- We conduct experiments using synthetic data and real data to evaluate the query performance of IPF.

The experimental result shows that IPF significantly reduces the MRJ processing time, and among three IPF methods, the performance of EMAX outperforms those of DMAX, NMAX.

The remainder of this paper is organized as follows: Section 2 provides some background by briefly explaining the 2-way R-tree join and the MRJ. In Section 3, we introduce indirect predicates in MRJ. And in Section 4, we propose a method for processing MRJ using indirect predicates and new R-tree structures for IPF. In Section 5, we present experiments for a performance analysis of IPF using synthetic data and the TIGER data [23]. Finally in Section 6, we conclude this paper and suggest some future studies.

2. Related work

2.1. R-tree joins

R-trees were proposed by Guttman [3] as a direct extention of B$^+$-trees [8] to n-dimensions. An R-tree node is constructed by grouping MBRs of near spatial objects, and a parent node is again constructed by grouping MBRs of near nodes. Fig. 2 shows an example set of data MBRs and the corresponding R-trees built on these MBRs (assuming maximum node capacity is 3).

After Guttman's proposal, several researchers proposed their own variations of the R-tree such as R + -tree [20] and R*-tree [1]. We call all of them *R-tree family*. The R-tree family seems to be the most promising ones among many extent spatial data structures and is the one that most research efforts have concentrated on.

Assuming that R-trees exist for both join inputs, a join algorithm called *R-tree join* which synchronously traverses both R-trees using depth-first search was proposed [2].

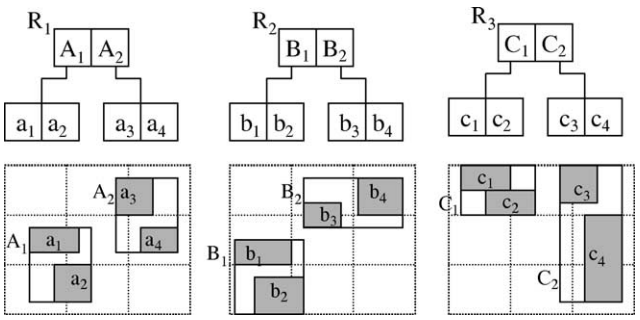


Fig. 2. Some MBRs and the corresponding R-trees.

```
PROCEDURE RJ (Rtree_Node R, S)
BEGIN
1.  FOR all E_S ∈ S DO
2.    FOR all E_R ∈ R DO
3.      IF E_R.rect intersects with E_S.rect THEN
4.        IF R is a leaf page THEN // assuming S is also a leaf page
5.          output (E_R, E_S)
6.        ELSE
7.          ReadPage(E_R.ref)
8.          ReadPage(E_S.ref)
9.          RJ (E_R.ref, E_S.ref)
END
```

Fig. 3. 2-way R-tree join algorithm.

The basic behavior of the algorithm is as follows: first, it reads the root nodes of the R-trees and checks if the rectangles of entries of both nodes mutually intersect. Next, only for intersected entry pairs, it traverses the child node pairs by depth-first search and continuously checks the intersection between the rectangles of entries of both child nodes. In this way, if the algorithm reaches the leaf nodes, it outputs the intersected entry pairs and returns to the parent nodes. The basic algorithm is shown in Fig. 3.

Example 2. Let us consider the join between two R-trees R_1 and R_2 in Fig. 2. First of all, the MBR intersections between entry sets $\{A_1, A_2\}$ and $\{B_1, B_2\}$ in both root nodes must be checked. Since $\langle A_1, B_1\rangle$ and $\langle A_2, B_2\rangle$ are intersected pairs at the root level, the child nodes of these pairs must be synchronously traversed. Let us assume that $\langle A_1, B_1\rangle$ will be first traversed. Now the intersections between entry sets $\{a_1, a_2\}$ and $\{b_1, b_2\}$ in nodes A_1 and B_1 are checked, and $\langle a_1, b_1\rangle$ and $\langle a_2, b_2\rangle$ are generated as output. And then, the synchronous traversal returns to the parent level. In this time, nodes A_2 and B_2 are visited. At the check between entry sets $\{a_3, a_4\}$ and $\{b_3, b_4\}$ in nodes A_2 and B_2, $\langle a_3, b_4\rangle$ are generated as the query result.

Two optimization techniques, called *search space restriction* and *plane sweep*, are used to reduce the CPU time. The search space restriction heuristic picks out the entries whose rectangles do not intersect with the rectangle enclosing the other node, before the intersection is actually checked between the rectangles of entries of both nodes. In the intersection check between nodes A_2 and B_2 of Example 2, since among entries in node A_2, a_4 does not intersect with the MBR of node B_2, a_4 cannot intersect with any entry in node B_2 and it can be removed from the check.

The plane sweep first sorts the rectangles of entries of both nodes for one axis, and then moves forward along the sweep line and checks the remaining intersection for the other axis. In the intersection check between nodes A_1 and B_1 of Example 2, if we sort the entries in each node along the x-axis, we obtain (a_1, a_2) and (b_1, b_2). After that, if we merge the two sorted sequences, we obtain (b_1, a_1, b_2, a_2) as a sweep line. Moving along the sweep line, we can check y-axis intersection for all entries of the other node which are located after the current sweep line and their x-axis' intersect with the current sweep line entry. In Example 2, for entry b_1, the y-axis' intersection will be checked with a_1 and a_2, for a_1, checked with b_2, for b_2, checked with a_2, and finally for a_2, the plane sweep ends because of no further entries after the sweep line.

Additionally, the algorithm applied the *page pinning* technique for I/O optimization. The algorithm used only a local optimization policy to fetch the child node pairs. Later, a global optimization algorithm by breadth-first search was proposed [7]. In this paper, we call both of the join algorithms *2-way R-tree join* or simply *R-tree join*. When R-trees exist for both join inputs, it has been shown that the R-tree join is most efficient [9,10,18].

2.2. M-way R-tree joins

Recently the MRJ algorithm was proposed as a generalization of the 2-way R-tree join [11,14,15]. As in the case of the 2-way R-tree join, the basic behavior is also the synchronous traversal of M R-trees. The synchronous traversal of M R-trees works as follows: it starts from the root nodes of M R-trees. For each M-combination from the entries (called *entry-tuple*) of the nodes, it checks all predicates defined by a query. If an entry-tuple satisfies all the predicates, one of the following occurs:

- If the *node-tuple* (an M-combination of the R-tree nodes) is at a nonleaf level, the algorithm is recursively called for the child node-tuple pointed by the entry-tuple.
- If the node-tuple consists of the leaf nodes, the algorithm outputs the entry-tuple and processes the next entry-tuple.

If an entry-tuple does not satisfy at least one predicate, the entry-tuple is pruned and the next entry-tuple is to be processed. If all entry-tuples have been checked in a node-tuple, the algorithm returns to the parent node-tuple. The above procedure is described in Fig. 4.

In the above algorithm, function *GetNextTuple* gets a combination from all the entries of nodes N[ ]. If there is no

```
PROCEDURE MRJ (Query_graph Q[][], Rtree_Nodes N[])
BEGIN
1.  WHILE ( τ = GetNextTuple(N[]) )
2.    IF τ satisfies Q[][] THEN
3.      IF N[] are leaf nodes THEN // assuming all tree heights are equal
4.        output (τ)
5.      ELSE
6.        FOR i = 0 to M-1 DO
7.          ReadPage(τ.ref[i])
8.        MRJ (Q[][], τ.ref[])
END
```

Fig. 4. M-way R-tree join algorithm.

further combination from N[ ], *GetNextTuple* returns NULL and the while loop ends.

Example 3. We will explain an example of MRJ using the R-trees again in Fig. 2. Let us consider a 3-way spatial join whose join predicates are 'R_1 intersect R_2 and R_2 intersect R_3'. At the root level, there are eight possible entry combinations because each node has two entries. Among these combinations, $\langle A_2, B_2, C_1\rangle$ and $\langle_2, B_2, C_2\rangle$ satisfy the join predicates. MRJ first reads the child nodes pointed by an entry-tuple $\langle A_2, B_2, C_1\rangle$ and check the join predicates for the entries among the child nodes. Since there is no entry-tuple among the nodes, MRJ returns to the parent level and processes the next entry-tuple $\langle A_2, B_2, C_2\rangle$. In this time, among the entry-tuples from the child nodes pointed by $\langle A_2, B_2, C_2\rangle$, $\langle a_3, b_4, c_3\rangle$ satisfies the join predicates and is generated as output.

Algorithm *MRJ* assumes that all join predicates are *intersect* (not disjoint) and the same predicate is applied for both leaf and nonleaf levels. As pointed out in Ref. [4], when general join predicates are used, a different predicate at nonleaf levels should be applied from the predicate at the leaf level. However, when the actual predicate is *intersect*, the leaf and nonleaf level predicates are the same. In a different manner, Theodoridis and Papadias classified the spatial relationships as topological, distance and direction relationships [21]. They have shown that all spatial relationships can be mapped to the *intersect* relationship in the R-tree operation. Thus, the general join predicates and the different predicates for nonleaf levels can be easily applied to Algorithm *MRJ*.

In Example 3, $\langle A_2, B_2, C_1\rangle$ satisfies the join predicates at the root level, but it generates no entry combination at the child level. Similarly, if an entry combination satisfies the query at the nonleaf level but it leads to generate no query result at the leaf level, we call it *false intersection*. The smaller the number of join predicates are, the more the false intersection occurs. In this paper, we use the indirect predicate concept to identify and prune the false intersection in advance.

The search space restriction and plane sweep heuristics of the 2-way join can be extended to the M-way join. As extensions of the search space restriction, some algorithms such as *static variable ordering* [11] and *space restriction ordering* (SRO) [15] were proposed. And as extensions of the plane sweep, some algorithms such as *multi-level forward checking* [13], *plane sweep ordering* [15] and *plane sweep forward checking* (PSFC) [11] were developed. For the details and performance evaluations about these algorithms, refer to Ref. [17]. This paper adopts the SRO–PSFC combination as a standard MRJ algorithm because the combination was evaluated to be most efficient in Ref. [17]. Therefore, we made implementations and experiments of IPF based on the SRO–PSFC combination.

In Section 1, we mentioned that MRJ can avoid unnecessary refinement operations compared to the sequence of 2-way joins. Let us consider again the above example of the 3-way spatial join. Let us assume that the above 3-way join is processed by a sequence of 2-way joins and the join ordering is determined to be $((R_1, R_2), R_3)$ by the query optimizer. According to Example 2, since entry pairs $\langle a_1, b_1\rangle$ and $\langle a_2, b_2\rangle$ are the filter step result of the join between R_1 and R_2, the refinement step operations should be performed on these pairs. However, if we examine Fig. 2, we come to know that both of b_1 and b_2 do not intersect with any object of R_3 and the intermediate results $\langle a_1, b_1\rangle$ and $\langle a_2, b_2\rangle$ cannot be contained in the final result of the 3-way join. Therefore, the refinement step operations on $\langle a_1, b_1\rangle$ and $\langle a_2, b_2\rangle$ in the sequence of the 2-way joins were performed unnecessarily. Since MRJ gets rid of such cases in the filter step, it can avoid such unnecessary refinement operations.

3. M-way R-tree joins using indirect predicates

In this section, we present our proposed method called *indirect predicates filtering*. In this work, following the standard approach in the spatial join literature, *intersect* (not disjoint) is considered as the default join predicate.

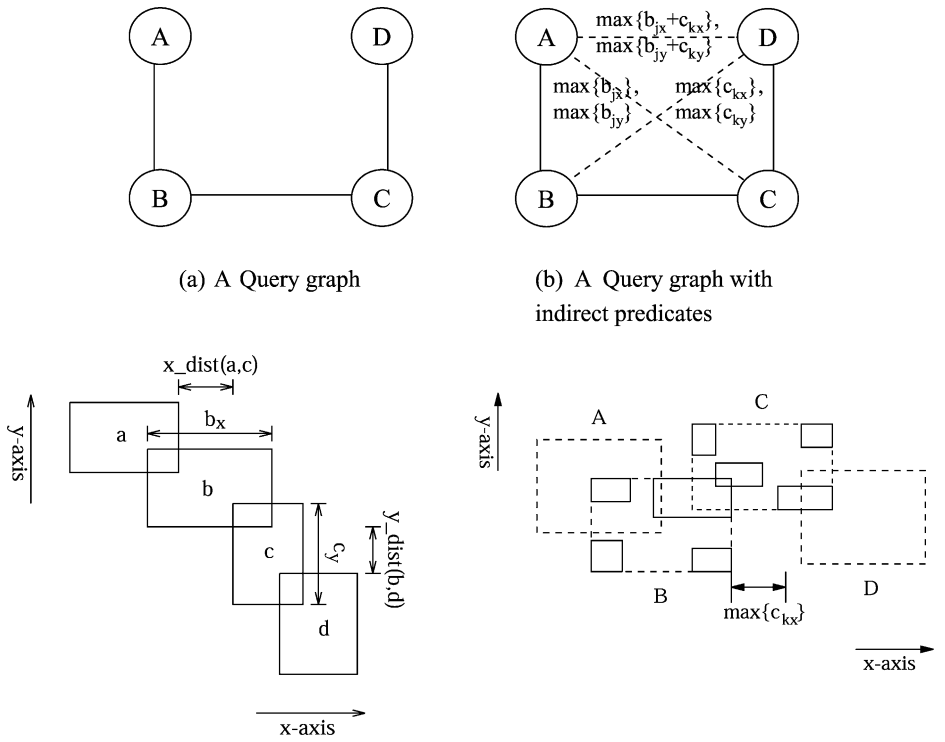
The maximum number of possible predicates in the M-way spatial join is $M(M-1)/2$, i.e. all relation pairs have join predicates. We call such a join *complete*. If a join is not complete, i.e. the number of predicates is less than $M(M-1)/2$, the join is *incomplete*.

As mentioned in Section 2.2, MRJ may generate many false intersections at nonleaf levels. Especially in an incomplete join, the possibility of a false intersection is high. In this case, if we can detect the false intersections before visiting the node-tuple, we can save I/O and CPU time. In this section, we propose a method which can detect the false intersections at nonleaf levels of R-trees.

3.1. Indirect predicates

In this section, we first address the concept of indirect predicates which do not directly come from the join conditions but are indirectly derived from them.

Example 4. Let us consider an example query representing a 4-way spatial join 'A *intersect* B and B *intersect* C and C *intersect* D'. Fig. 5(a) shows the query graph for this join. Fig. 5(c) shows an MBR intersection for a result tuple (i.e. a tuple from leaf node entries which satisfies the above query). It seems that there are no relationships between A and C, between B and D, and between A and D because there are no predicates between them. However, if b_x and c_x represent x-lengths for MBRs of b and c, respectively, the following relationships are satisfied on x-axis for a result



(a) A Query graph

(b) A Query graph with indirect predicates

(c) An MBR intersection for a result tuple

(d) An NBR intersection between R-tree nodes

Fig. 5. A query graph and MBR intersection for a 4-way join.

tuple $\langle a,b,c,d\rangle$

$$x\_\mathrm{dist}(a,c) \le b_x,\ x\_\mathrm{dist}(b,d) \le c_x,\ x\_\mathrm{dist}(a,d) \le b_x + c_x. \tag{1}$$

The same condition holds on y-axis. Since b_x and c_x are values per spatial object, MRJ cannot know the values during nonleaf level processing. However, since the maximum x-length per object set can be kept in advance as catalog information by the query optimizer, the query optimizer can use it. For the result tuple $\langle a,b,c,d\rangle$ which satisfies Eq. (1), the following relationships are also satisfied on x-axis

$$x\_\mathrm{dist}(a,c) \le \max\{b_{jx} | b_j \in \mathrm{dom}(B)\}, \tag{2}$$
$$x\_\mathrm{dist}(b,d) \le \max\{c_{kx} | c_k \in \mathrm{dom}(C)\},$$
$$x\_\mathrm{dist}(a,d) \le \max\{b_{jx}\} + \max\{c_{kx}\}.$$

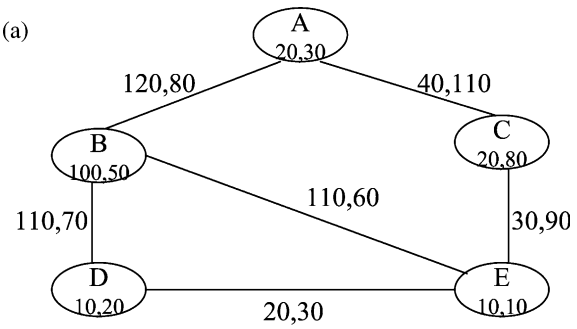
In Eq. (2), dom(B) represents the domain (i.e. relation) of data MBRs for variable B. The same condition holds on y-axis. The x-length and y-length of MBR at the nonleaf level of an R-tree may be longer than the max x-length and max y-length of the data MBRs in the domain. Therefore, Eq. (2) can be used to find false intersections in advance at the nonleaf level. Fig. 5(d) shows an MBR intersection between R-tree nodes for the above query. (We use the node name as the same as the variable name for convenience.) If $x\_\mathrm{dist}(B,D) > \max\{c_{kx}\}$, $x\_\mathrm{dist}(A,C) > \max\{b_{jx}\}$ or $x\_\mathrm{dist}(A,D) > \max\{b_{jx}\} + \max\{c_{kx}\}$, we do not have to visit the node-tuple $\langle A,B,C,D\rangle$ because the node-tuple and the descendent node-tuples will never satisfy the query. Therefore, the node-tuple shown in Fig. 5(d) can be pruned in advance at the entry-tuple level of the parent nodes.

We call the user predicates in the query such as 'A *intersect* B' and 'B *intersect* C' the *direct predicates* which correspond to the edges in Fig. 5(a), and the derived predicates such as $x\_\mathrm{dist}(A,C) \le \max\{b_{jx}\}$ and $x\_\mathrm{dist}(B,D) \le \max\{c_{kx}\}$ the *indirect predicates*. The dotted edges in Fig. 5(b) represent the indirect predicates. These indirect predicates can be used for pruning entry-tuples at nonleaf levels of MRJ. We call such pruning IPF. The max x-length and y-length can be obtained from the catalog information in the database schema.

3.2. Indirect predicate paths and lengths

A *path* from vertex v_p to vertex v_q in a graph G is a sequence of vertices, $v_p, v_{i_1}, v_{i_2}, \ldots, v_{i_n}, v_q$ such that $(v_p, v_{i_1}), (v_{i_1}, v_{i_2}), \ldots, (v_{i_n}, v_q)$ are edges in the graph. The *length* of a path is the sum of the weights of the edges on that path. In Fig. 5, we call the paths ABC, BCD and ABCD for indirect predicate pairs AC, BD and AD the *indirect predicate paths* (ipp), and the x-path lengths $\max\{b_{jx}\}$, $\max\{c_{kx}\}$, and $\max\{b_{jx}\} + \max\{c_{kx}\}$ the *indirect predicate* x_path lengths (x_ippl). The *indirect predicate* y_path lengths (y_ippl) are similarly defined. In this section,

(a)



(b)

pairs	x_ipp	x_ippl	y_ipp	y_ippl
AD	ACED	30	ABD	50
AE	ACE	20	ABE	50
BC	BEC	10	BEC	10
CD	CED	10	CED	10

Fig. 6. Maximum weighted query graph.

we will explain how to compute the indirect predicate paths and indirect predicate path lengths.

In Fig. 5, since the corresponding indirect predicate path for each indirect predicate pair is only one, it is easy to compute indirect predicate paths and indirect predicate path lengths. However, there can be several indirect predicate paths for an indirect predicate pair in a general M-way join because the general M-way join graph can contain cycles. Therefore, a systematic method to compute indirect predicate paths and their lengths is required.

As shown in Fig. 6(a), we first draw a query graph whose vertices represent relations and edges represent direct predicates. Then, we assign weights to vertices. The weight of a vertex is the maximum x-length (x_max) and y-length (y_max) in the relation which the vertex represents. Since there can be multiple paths between a vertex pair, we compute the ipp and ippl by using the all pair shortest path algorithm [6]. In order to get the shortest path between a vertex pair, we need edge weights but we have only vertex weights now. Therefore, we obtain edge weights from vertex weights. The weight of an edge is obtained by summing weights of the vertices on which the edge is incident. An example query graph having both vertex weights and edge weights for a 5-way join is shown in Fig. 6(a). We call this query graph *maximum weighted query graph*.

When there is no direct predicate between two vertices S and D in a maximum weighted query graph, the ipp and ippl between S and D can be obtained as follows:

- First, we calculate the shortest path and shortest path length per axis.
- Second, we subtract the weights of both S and D from the shortest path length and then divide the shortest path length by 2.

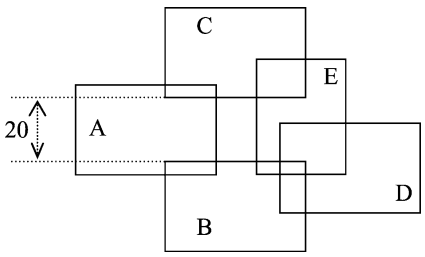


Fig. 7. An example of applying indirect predicates in a 5-way join.

In the first step, the weights of S and D are included in the edge weights of the shortest path length and the weights of the intermediate vertex are included twice. Then, in the second step, we obtain the sum of the weights of intermediate vertices in the shortest path. Therefore, the x_ippl between S and D can be calculated by Eq. (3)

$$x\_\text{ippl}(S,D) = (x\_\text{shortest\_path\_length}(S,D) - x\_\max(S) - x\_\max(D))/2. \quad (3)$$

The y_ippl is similarly defined. The ipps and ippls for all indirect predicate pairs in Fig. 6(a) are shown in Fig. 6(b). As shown in for indirect predicate pairs AD and AE of Fig. 6(b), the x_ipp and y_ipp between a vertex pair can be different because the shortest path per axis between the vertex pair can be different.

The indirect predicate paths and lengths can be used to find false intersections which could not be found only by direct predicates during the nonleaf level processing of MRJ.

Example 5. Let Fig. 7 be an MBR combination of nonleaf nodes of R-trees in a 5-way spatial join whose query graph is Fig. 6(a) and whose ipps and ippls are Fig. 6(b). Fig. 7 satisfies all the direct predicates in the query graph of Fig. 6(a), i.e. (A intersect B) and (A intersect C) and (B intersect D) and (B intersect E) and (C intersect E) and (D intersect E). However, since the y-distance between B and C is 20, it is larger than the y_ippl (10) between B and C in Fig. 6(b). Therefore, this node combination is pruned without further processing for their descendants.

In Fig. 6(b), BAC and BEC are candidates of the y_ipp between B and C. Since y_max (10) of D is smaller than y_max (30) of A, BEC is selected as the y_ipp by the shortest path algorithm.

4. Variations of indirect predicates

4.1. Node max information

Until now, we have used only one max x-length and y-length per relation. This is called *domain max information*.

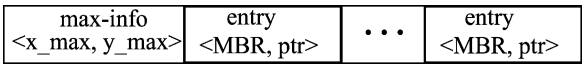


Fig. 8. Node structure of the *NMAX R-tree*.

Fig. 9. Nonleaf node structure of the *EMAX R-tree*.

In this case, if there are several extremely large objects in a relation although other objects are not so large, the effect of indirect predicates can be considerably degraded.

One possible solution for this is to have the max x-length and y-length per R-tree node. Each node keeps the length of the largest object per axis among the objects contained in all subtrees of the node. A leaf node has the max x-length and y-length for MBRs of all entries in the node, and a nonleaf node has the maximum value for the max x-lengths and max y-lengths of its child nodes. At the end, the root node has the max x-length and max y-length for the relation. The max x-length per R-tree node is recursively defined as in Eq. (4)

$$x\_\max(N) = \begin{cases} \max\{N_1\mathrm{rect}_x \ldots N_n\mathrm{rect}_x\} \\ \text{for leaf node} \\ \max\{x\_\max(N_1\mathrm{ref}) \ldots x\_\max(N_n\mathrm{ref}) \\ \text{for nonleaf node} \end{cases}, \quad (4)$$

where n is the number of entries in node N.

The max y-length is similarly defined. We call the max x-length and y-length per R-tree node *node max information*. By using the NMAX instead of the DMAX, we can improve pruning effects in IPF of MRJ.

As shown in Fig. 8, since only two max values are attached as *max-info* per R-tree node (one for x-length and the other for y-length), we can ignore the storage overhead due to the max lengths. Since the max lengths can be dynamically maintained with the R-tree insertion and deletion, we can always have exact max lengths per R-tree node. We call this R-tree the *NMAX R-tree*.

Since calculating the shortest path for every node-tuple needs a large CPU time overhead[1], we calculate the paths (i.e. ipps per axis) only once using the max information of the root nodes of R-trees. The reason for calculating the ipps only once is because the probability that the shortest path among the root nodes is equal to the shortest path among the nodes of the other nonleaf levels is high. However, we calculate the lengths (i.e. ippls) every time based on the ipps obtained from the root nodes by the following reasons: (1) although ipps per R-tree level are the same, the ippls may be significantly different. (2) If an ipp between two nodes are given, its ippl can be calculated in linear time.

4.2. Entry max information

In NMAX R-trees, we used the NMAX of the current level to check indirect predicates between entries each of which points to the child node. Since the NMAX of the current level is the maximum of the NMAX of all the child nodes (see Eq. (4)), the max lengths can be much longer than we need. And if the R-tree height is 2, there is no benefit from the NMAX because there is only one nonleaf node (root node). What we actually need is the NMAX of the child nodes. However, we cannot get the NMAX of the child nodes without visiting them. Therefore, the last strategy for IPF is that each entry of nonleaf nodes has the max information of its child node. We call the max information per entry EMAX and the R-tree which has the entry max information the *EMAX R-tree*.

As shown in Fig. 9, an entry of a nonleaf node in an EMAX R-tree consists of (*max-info, MBR, ptr*) where *MBR* is the minimum bounded rectangle of its child node, *ptr* is the pointer to the child node and *max-info* (i.e. EMAX) is the NMAX of the child node. The structure of leaf nodes is the same as that of the normal R-tree. *max-info* can also be dynamically maintained with the *MBR* field during insertion and deletion. An entry of a nonleaf node of the EMAX R-tree occupies 28 bytes, i.e. each 4 bytes for xmin, xmax, ymin, ymax of MBR, and each 4 bytes for x_max, y_max of max_info, and 4 bytes for ptr. The normal R-tree uses 20 bytes per entry because there is no max_info in its entry. Therefore, an EMAX R-tree uses storage 28/20 times of the normal R-tree in nonleaf nodes. However, in R-trees, since the most part of storage is used in leaf nodes, this is not a significant overhead. Since the number of entries which the EMAX covers is much smaller than what the NMAX covers (average $1/C$ times, C : the average number of entries of an R-tree node), we expect that although there is a little storage overhead we can get a considerable effect of IPF using the EMAX.

The MRJ algorithms using indirect predicates are the same as the normal MRJ algorithms except that the indirect predicates should be checked for all node pairs which have no direct predicates.

4.3. Dynamic maintenance of DMAX/NMAX/EMAX

In this section, we describe how DMAX, NMAX and EMAX are dynamically maintained with the insertion and deletion of an entry.

The dynamic maintenance of DMAX is straightforward. When an entry is inserted, if the x-length and/or y-length of the entry is greater than the current DMAX, it becomes the new DMAX. In the case of deletion, if the x-length and/or y-length of the deleted entry is equal to the current DMAX, the new DMAX must be recomputed by scanning all entries in the relation.

[1] The complexity of computing all pair's shortest paths is known to be $O(M^3)$ [6].

```
PROCEDURE AdjustXNMAX (Rtree_Nodes N[], x_length)
BEGIN
1.  Let P be the parent node of N
2.  IF N is not splitted THEN
3.     IF x_length > N.x_max THEN
4.        N.x_max := x_length
5.        AdjustXNMAX (P, N.x_max)
6.  ELSE IF N is splitted or newly created by split
       or some entries of N are deleted for re-insertion THEN
7.     IF N is at the leaf level THEN
8.        Compute new N.x_max by scanning all entries
9.     ELSE
10.       Compute new N.x_max by visiting all child nodes
11.    AdjustXNMAX (P, N.x_max)
END
```

Fig. 10. The algorithm of AdjustXNMAX.

```
PROCEDURE CondenseXNMAX (Rtree_Nodes N[], x_length)
BEGIN
1.  Let P be the parent node of N
2.  IF N is not deleted THEN
3.     IF x_length = N.x_max THEN
4.        IF N is at the leaf level THEN
5.           Compute new N.x_max by scanning all entries
6.        ELSE
7.           Compute new N.x_max by visiting all child nodes
8.        CondenseXNMAX (P, N.x_max)
9.  ELSE IF N is deleted THEN
10.    CondenseXNMAX (P, N.x_max)
END
```

Fig. 11. The algorithm of CondenseXNMAX.

The dynamic maintenance of NMAX and EMAX can be incorporated into the insertion and deletion algorithms of the R-trees. The insert algorithm of the R-tree executes Algorithm *AdjustTree* which adjusts all covering rectangles upwards along the insert and split path [3]. In an EMAX R-tree, since an entry of a nonleaf node directly has a *max-info* field (EMAX) together with a *MBR* field, EMAX can be maintained exactly in the same way as the *MBR* field in Algorithm *AdjustTree*.

Since NMAX exists not per entry but per node, it is maintained slightly differently from EMAX. The following algorithm *AdjustXNMAX* is invoked together with Algorithm *AdjustTree* of the R-tree (Fig. 10).

When an entry is inserted to a leaf node and a split does not occur, if x_length of the entry is greater than the current x_max (the x-component of *max-info* for NMAX) of the node, the x_length becomes the new x_max. If x_max of the leaf node is replaced, Algorithm *AdjustXNMAX* is invoked again for the parent node to adjust x_max of the parent node. If a split occurs, x_max's for the split node and the newly created node must be recalculated. For a leaf node, x_max is calculated by scanning all entries. If the split is propagated to a nonleaf node, for both the split node and the newly created node, all child nodes must be visited to recalculate the new x_max's. In this way, Algorithm *AdjustXNMAX* is propagated upward until no split occurs and x_max of a node is not changed, or it encounters the root node.

The R*-tree sometimes invokes the forced re-insert routine when an overflow occurs [1]. To perform the re-insertion, some entries must be deleted first. After the deletion, the new x_max is calculated by the same method as in the case of the split. The procedure for the re-insert is the same as that for the normal insert. In the case of the re-insert of a nonleaf level entry, the parameter x_length is the x_max of the child node of the entry. The algorithm for y_max, i.e. *AdjustYNMAX*, can be similarly defined (Fig. 11). When an entry of the R-tree is deleted, the deletion algorithm lastly invokes *CondenseTree*. As in the case of the insertion, the following algorithm *CondenseXNMAX* is invoked together with Algorithm *CondenseTree* of the R-tree.

When an entry is deleted from a leaf node, if the x_length of the entry is equal to the current x_max of the node, the new x_max must be calculated by scanning all entries of the node. The newly calculated x_max is propagated upward at the same time with *CondenseTree*. If *CondenseXNMAX* is invoked at nonleaf level as a result of the propagation and the x_length parameter is equal to the current x_max of the nonleaf node, all child nodes of the nonleaf node must be visited to calculate the new x_max. If the number of entries of a node is less than the low limit (i.e. m) as a result of the deletion, the node must be condensed.

The original condense algorithm *CondenseTree* of the R-tree deletes the entry pointing to the node from the parent node and re-inserts all remaining entries in the deleted node [3]. When a node is deleted, the x_max of the parent must be adjusted because an entry is also deleted from the parent node. In that case, the x_max of the deleted node is passed as a parameter. The maintenance of the x_max for the re-insert is the same as that for the forced re-insert in *AdjustXNMAX*.

The algorithm for y_max, i.e. *CondenseYNMAX*, can be similarly defined. As we have shown in the above two algorithms, the dynamic maintenance of NMAX has more overhead than that of EMAX because it sometimes needs scanning child nodes. This is also one reason we suggest EMAX in this paper.

5. Experiments

To measure the performance of IPF in the MRJ, we conducted some experiments using synthetic data and real data. The experiments were performed on a Sun Ultra II 170 MHz platform on which Solaris 2.5.1 was running with 384 MB of main memory. As we mentioned in Section 2.2, many heuristics were developed as extensions of the search space restriction and plane sweep heuristics of the 2-way join. Among them, we used the SRO–PSFC combination as

a standard MRJ algorithm in our experiments because it is known to be most efficient [17].

5.1. Experimental environments

Data sets. we extensively evaluated our proposed technique, IPF with synthetic and real-life data sets.

For a performance evaluation of IPF using synthetic data sets, we first generated several data sets which consist of 10,000 uniformly distributed rectangles for data densities 0.25 and 1.0 on domain size (100,000,100,000). For each data set, in order to easily maintain the data density, we generated the same sized rectangles (500 for density = 0.25 and 1000 for density = 1.0). And then, for each data set, we built the R*-trees [1] for node sizes 1K and 4K. For each node size, the heights of the R*-trees are 3 and 2, and the sizes of LRU buffers are 512 and 256 pages, respectively[2].

The real data sets in our experiments were extracted from the TIGER/Line data of US Bureau of the Census [24]. We used the road segment data of seven counties of the California State in the TIGER data. The statistical information of the California TIGER data is summarized in Table 1. In Table 1, L*i*_NMAX stands for the average NMAX in the *i*-th level of the R*-tree built on the data set. L0_NMAX is for the leaf level and L2_NMAX for the root level, i.e. DMAX. The original TIGER data of all counties were center-matched to join different county regions, i.e. the *x* and *y* coordinates of the original TIGER data were subtracted from those of the center point of each county.

We built the R*-trees for the above data sets for node size 4K. The heights of the R*-trees are equally 3. For this experiment, we randomly extracted the following three data combinations from the TIGER data shown in Table 1. An M-way join for each data combination was performed for the first M counties of the data combination for $M = 3, \ldots, 7$.

Data combi 1	Ala.	S.D.	Sac.	Ker.	Mon.	S.B.	Ora.
Data combi 2	Mon.	S.B.	S.D.	Sac.	Ker.	Ala.	Ora.
Data combi 3	Ora.	Ala.	S.B.	S.D.	Ker.	Sac.	Mon.

Query sets. We selected the following four query types as input queries: half, ring, chain and star. Example query graphs for each query type including the complete query type in a 5-way join are shown in Fig. 12. The spatial predicate used for our experiments is *intersect* (not disjoint).

5.2. Experimental results of IPF

In our experiments, we report the response time of each M-way spatial join technique. The response time consists of CPU and I/O portions. According to Refs. [11,15], however, the MRJ is a CPU-bound task. Since the presentation of the I/O is less important and it is well described in Ref. [15], we present only the total response time in our experimental results.

[2] We assume that an R*-tree node occupies one page.

Table 1
Statistical information of the California TIGER data

(a) *Basic statistics*

County	# of obj	Domain area	Avg. length	Density
Ala.	49,070	8,622,244,995	10,280	0.23
Ker.	113,407	257,781,100,758	212,169	0.26
Mon.	35,417	175,744,112,068	234,192	0.20
Ora.	91,970	6,999,955,588	8066	0.21
Sac.	46,516	7,577,171,218	11,186	0.24
S.D.	103,420	15,124,196,476	122,104	0.22
S.B.	64,037	9,930,158,696	10,081	0.22

(b) *max-info per R*-tree level*

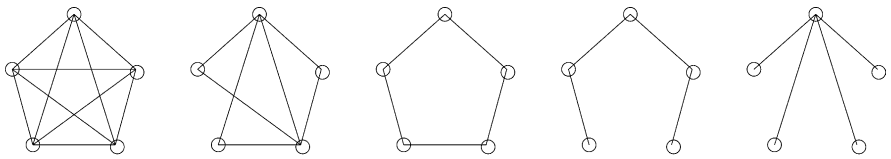
County	L2_NMAX	L1_NMAX	L0_NMAX
Ala.	46,623,940	36,912,640	676,550
Ker.	82,046,497	58,484,599	15,901,273
Mon.	90,856,194	66,084,504	16,751,351
Ora.	36,586,735	24,232,638	557,488
Sac.	64,424,103	46,423,223	788,604
S.D.	80,546,828	46,953,819	973,807
S.B.	45,416,460	29,312,845	660,540

Synthetic data sets. To measure the effect of IPF, we varied DMAX. Note that the change of DMAX does not incur the loss of accuracy of query results because indirect predicates are supplementary and the actual result is determined by only direct predicates. Fig. 13 shows the response time of NO IPF and IPF in the chain query type with varying DMAX (assuming *x*-length and *y*-length of DMAX are equal) and the performance rates between them. NO IPF in Fig. 13 stands for the MRJ algorithm which just uses the SRO and PSFC heuristics without indirect predicates while IPF literally stands for the MRJ algorithm which uses both heuristics with indirect predicates. Obviously, the performance of NO IPF is not affected by varying DMAX.

Since we generated only the same sized rectangles, the values of DMAX, NMAX and EMAX are actually the same as the size of the rectangles. However, for convenience of experiments, we assume that we can vary only DMAX in synthetic data sets. This corresponds to the case that we vary only one rectangle with the other rectangles unchanged.

As shown in Fig. 13, the effect of IPF increases as M increases and DMAX decreases. IPF performs better in the low density data (0.25) than in the high density data (1.0), especially in the node size 4K. In this experiment, IPF is faster about 5 times than NO IPF in the case of low density, high M and small DMAX. (See the case of density = 0.25, $M = 7$ and DMAX = 500 in Fig. 13(b).) However, there is no effect of IPF in the case of large DMAX. (See around 8000 of DMAX in node size 1K and from around 16,000 of DMAX in node size 4K in Fig. 13(b).)

We also measured the performance of IPF in other query types (Fig. 14). There is nearly no effect in the half query type but there is a large effect in the star query type. This



(a) complete (b) half (c) ring (d) chain (e) star

Fig. 12. Example query graphs in a 5-way join.

means that IPF has a large effect in the small number of direct predicates (and consequently numerous indirect predicates) but has a small effect in many direct predicates.

Real data sets. For the real data combinations of Section 5.1, we measured the performance of IPF using three kinds of max information such as DMAX, NMAX and EMAX. Fig. 15 shows the response time of NO IPF and IPF for various data combinations of real data sets under node size 4K, and Fig. 16 the relative performance rate of IPF to NO IPF.

0.25

1K

	3	4	5	6	7
NO IPF	13.9	29.6	77	238	873
500	13.3	23.6	43	81	162
1000	13.5	24.8	50	105	248
2000	13.6	25.3	59	146	401
4000	13.9	28.4	71	202	670
8000	13.9	29.6	78	242	883
16000	13.9	29.7	79	245	894

4K

	3	4	5	6	7
NO IPF	4.1	9.7	25	77	264
500	4.0	7.9	15	30	57
1000	4.0	8.2	17	37	78
2000	4.0	8.5	19	46	114
4000	4.0	9.0	21	58	165
8000	4.1	9.5	24	71	227
16000	4.1	9.7	25	78	265

1.0

1K

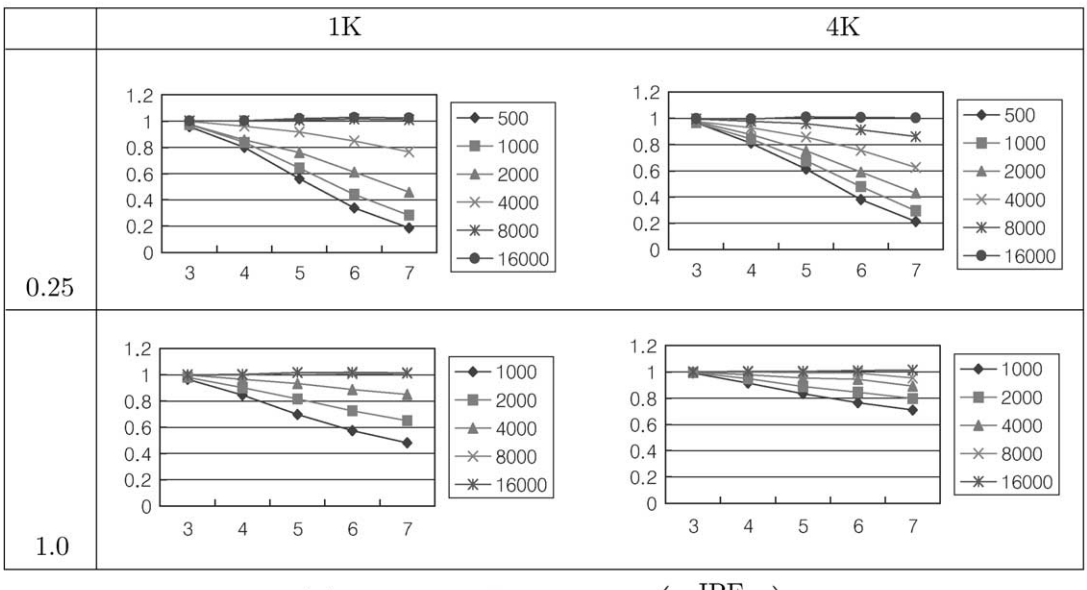
	3	4	5	6	7
NO IPF	17.5	50	187	775	3379
1000	16.9	43	130	445	1626
2000	17.2	45	152	562	2195
4000	17.5	49	174	687	2874
8000	17.5	50	189	781	3415
16000	17.5	51	190	790	3426

4K

	3	4	5	6	7
NO IPF	6.4	22.1	83	343	1385
1000	6.3	20.2	69	262	982
2000	6.3	20.9	74	290	1105
4000	6.3	21.6	79	323	1236
8000	6.4	22.1	83	340	1320
16000	6.4	22.2	84	346	1404

x-axis: the number of relations, y-axis: DMAX

(a) Response time (unit: sec)



(b) Relative performance rate $\left(\frac{\text{IPF}}{\text{NO IPF}}\right)$

Fig. 13. Performance of IPF in the chain query type on synthetic data sets.

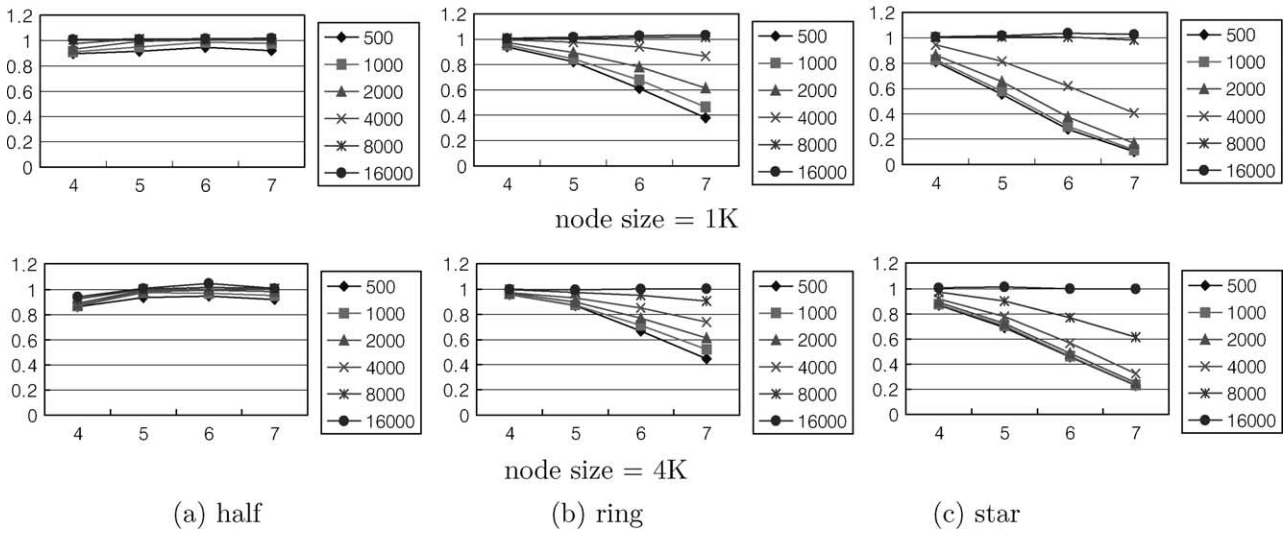


Fig. 14. Relative performance of IPF in other query types (density = 0.25).

Most behaviors are similar to the case of density = 0.25 of the synthetic data sets because the average density of the real data sets is around 0.25. Since the lengths of indirect predicates decrease along the sequence of DMAX, NMAX and EMAX, the performance of IPF increases along the sequence. From these results, we can see that EMAX clearly outperforms DMAX and NMAX.

In the star query, the performance of IPF is much influenced by the characteristics of the central node because the longest path in the query graph has length 2 and the central node is the intermediate node of all ipps. In our experiments, the central node of the star query represents the first element of each data combination. If there are many objects over the small domain area in the central node, the MBR sizes of the intermediate nodes of the R*-tree will be small and the effect of IPF will decrease. For this reason, we think, the effect of IPF for the star query in data combination 3 is small. From the above fact, we can also know that IPF has a large effect if the domain area is large.

As a general conclusion of this experiment, IPF has a considerable effect in higher Ms and sparser query types in terms of spatial query characteristics, and in lower density, larger R-tree node size, smaller max-info and larger domain area in terms of data characteristics. In most cases, IPF using EMAX is more efficient than those using DMAX, NMAX, and NO IPF.

5.3. Experimental results of dynamic maintenance of NMAX and EMAX

We also made an experiment to measure the overhead of the dynamic maintenance of NMAX and EMAX during

		Data Combi 1				Data Combi 2				Data Combi 3			
Half	M	4	5	6	7	4	5	6	7	4	5	6	7
	NO IPF	8.5	8.4	10.7	22.5	4.8	5.8	12.3	18.4	26.8	52.1	15.0	16.4
	DMAX	7.1	8.5	10.8	22.9	4.3	5.7	12.4	18.7	26.7	52.6	15.2	16.8
	NMAX	7.1	8.2	10.8	22.5	4.2	5.4	11.8	18.3	26.5	51.6	14.9	16.5
	EMAX	6.6	6.8	9.3	18.2	3.7	5.0	9.4	17.2	23.2	41.1	15.2	14.9
Ring	M	4	5	6	7	4	5	6	7	4	5	6	7
	NO IPF	7.3	13.4	42	165	3.4	10.6	40	209	27.9	70.9	63	256
	DMAX	7.2	13.1	36	143	3.4	10.1	36	164	27.8	69.8	55	213
	NMAX	7.2	12.7	35	136	3.4	9.8	35	152	27.8	67.7	53	196
	EMAX	6.9	10.0	22	77	3.2	8.4	26	106	25.9	59.3	45	125
Chain	M	4	5	6	7	4	5	6	7	4	5	6	7
	NO IPF	20.0	48	327	3128	6.6	33	207	1372	64.1	357	539	1855
	DMAX	16.4	42	203	1959	5.9	26	158	811	58.7	302	361	1044
	NMAX	15.9	39	175	1702	5.5	24	143	734	56.6	278	308	909
	EMAX	12.6	26	88	657	4.5	17	78	389	46.8	190	175	359
Star	M	3	4	5	6	3	4	5	6	3	4	5	6
	NO IPF	4.8	16.2	77	1657	22.1	11.4	175	1112	17.3	77.4	715	1611
	DMAX	4.7	10.5	46	706	21.5	7.9	99	575	17.2	77.1	717	1606
	NMAX	4.7	10.5	46	702	21.4	7.6	96	530	17.2	76.6	713	1600
	EMAX	4.4	9.6	43	680	19.8	5.8	41	260	17.0	70.6	695	1595

Fig. 15. Response time of IPF for various data combinations on real-life data sets (nodes size = 4K, unit: seconds).

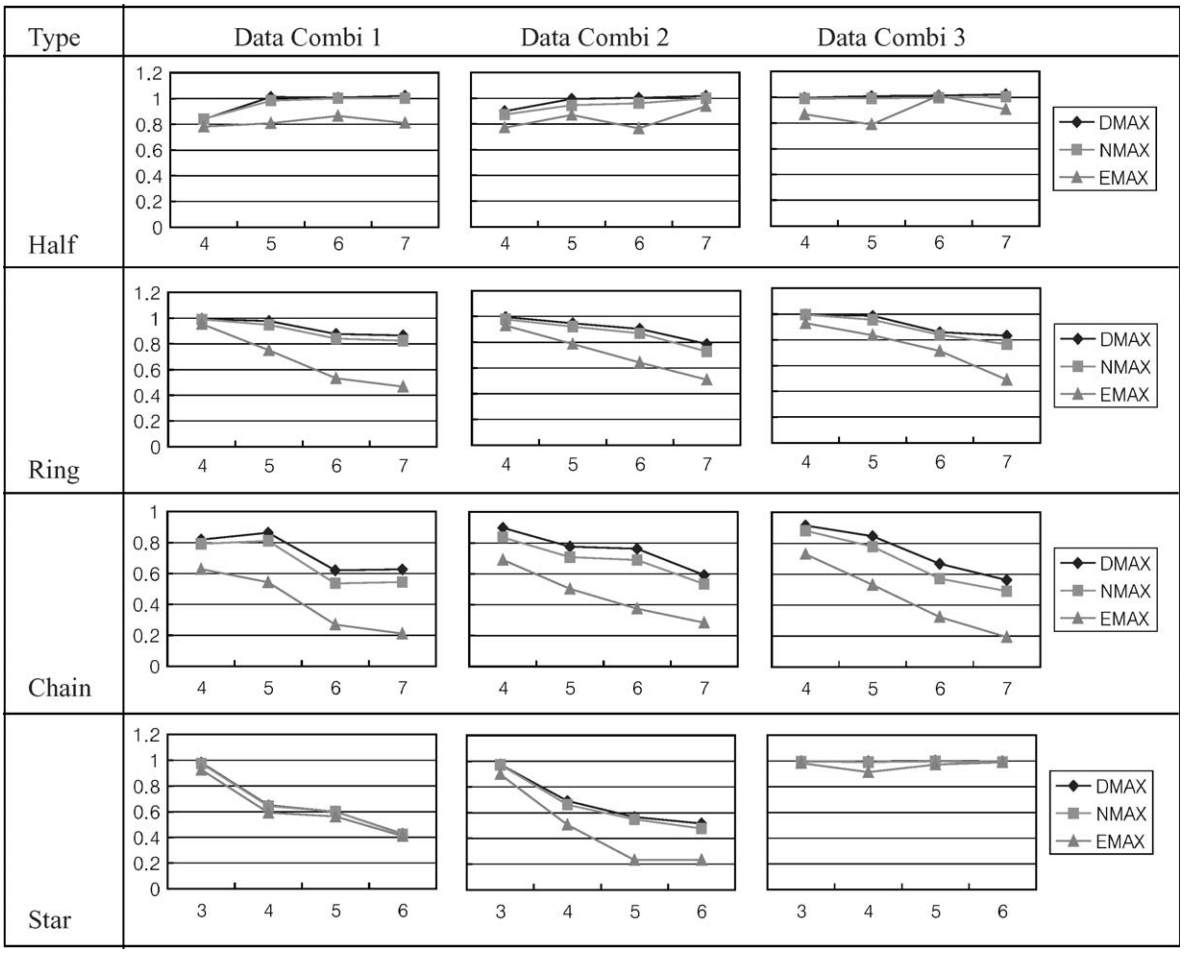


Fig. 16. Relative performance rate (IPF/NO IPF).

R-tree insertion and deletion. For this experiment, we implemented the dynamic maintenance algorithms of the NMAX and EMAX R*-trees presented in Section 4.3. The experimental data are the same as the real data, used in Section 5.2, which are the road data of seven counties in the California state. We generated the following three kinds of R*-trees for each data set:

- the original R*-tree (ORIG) which has no max-info,
- the NMAX R*-tree which has NMAX per node,
- the EMAX R*-tree which has EMAX per entry.

The generation time of each R*-tree per data set is shown in Fig. 17. The NMAX and EMAX R*-trees obviously consume more time than the original R*-tree for tree generation because they must dynamically maintain NMAX and EMAX whenever insertion occurs. According to Fig. 17, the EMAX R*-tree needs about 5% more time than the original R*-tree, and the NMAX R*-tree about 14% more time.

Since the max_info of the EMAX R*-tree can be adjusted at the same time as the MBR in *AdjustTree* which the insertion algorithm of the R*-tree finally invokes, the maintenance overhead of EMAX is not so big. However, according to Section 4.3, when insertion occurs in the NMAX R*-tree, *AdjustXNMAX* and *AdjustYNMAX* must be invoked along with *AdjustTree* to maintain NMAX dynamically. If split or forced re-insertion occurs during insertion, the *AdjustXNMAX* and *AdjustYNMAX* algorithms must visit all child nodes to calculate a new NMAX. Therefore, the overhead of NMAX is shown higher than the overhead of EMAX in Fig. 17.

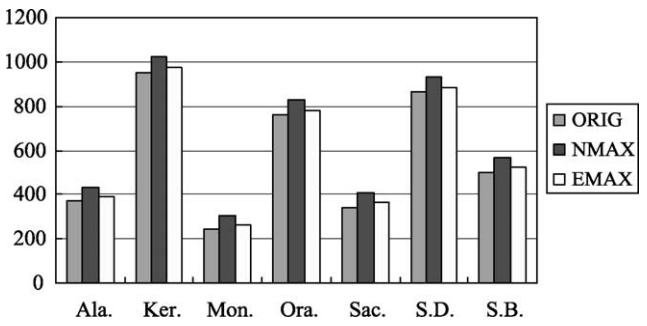


Fig. 17. The R*-tree generation time of ORIG, NMAX and EMAX.

In Section 5.2, we showed that the join performance using EMAX is much better than that using NMAX. And in this section we showed that the dynamic maintenance cost of EMAX is cheaper that that of NMAX. Therefore, we recommend EMAX rather than NMAX in processing multi-way spatial joins using indirect predicates.

6. Conclusions

In this paper, we introduced *indirect predicates* in the MRJ, and proposed an optimization technique called *indirect predicate filtering* to improve the performance of MRJ. For IPF, we introduced three kinds of max information called *domain max information, node max information* and *entry max information*. For DMAX, we used the catalog information of the query optimizer, and for NMAX and EMAX, we suggested new R-tree structures called the *NMAX R-tree* and the *EMAX R-tree*.

Through experiments using synthetic data and real data, we showed that IPF has a great impact on improving the performance of MRJ. From the viewpoint of query characteristics, the effect of IPF increases as M is higher and the number of direct predicates is smaller. From the viewpoint of data characteristics, the effect of IPF increases as the data density is lower and the node size of the R^*-tree is bigger and the domain area is larger and the max-info is smaller. Especially IPF using EMAX clearly outperforms the other max information DMAX and NMAX.

We also examined how NMAX and EMAX can be dynamically maintained in conjunction with the R-tree insertion and deletion algorithms. By experiments, we showed that the maintenance cost of EMAX is cheaper than that of NMAX. Therefore, this paper recommends EMAX rather than NMAX in processing multi-way spatial joins using indirect predicates because EMAX outperforms NMAX in both join and insertion.

Since MRJ is a filter step operation, the result is a set of oid-tuples. After completing MRJ, an oid-pair may appear several times in the resulting oid-tuples. If the oid-tuples are read in the refinement step without scheduling, it may access the same page several times and perform the same refinement operation several times. However, this can be solved by extending scheduling methods for oid pairs such as [24] to oid-tuples. In future studies, first, we will develop an efficient refinement algorithm for the M-way spatial join. Second, we will develop a cost model of IPF and combine the IPF algorithm with our rule-based optimization technique for spatial and non-spatial mixed queries called ESFAR (Early Separated Filter And Refinement) [16].

Acknowledgements

This work was supported by the Ministry of Information and Communication of Korea through the research grant of IITA.

References

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The R^*-tree: an efficient and robust access method for points and rectangles, Proc. ACM SIGMOD (1990) 322–331.
[2] T. Brinkhoff, H.-P. Kriegel, B. Seeger, Efficient processing of spatial joins using R-trees, Proc. ACM SIGMOD (1993) 237–246.
[3] A. Guttman, R-trees: a dynamic index structure for spatial searching, Proc. ACM SIGMOD (1984) 47–57.
[4] O. Günther, Efficient computation of spatial joins, Proc. IEEE ICDE (1993) 50–59.
[5] R.H. Güting, An introduction to spatial database systems, VLDB J. 3 (4) (1994) 357–399.
[6] E. Horowitz, S. Sahni, Fundamentals of Computer Algorithms, Computer Science Press, Rockville, MD, 1978.
[7] Y.-W. Huang, N. Jing, E.A. Rundensteiner, Spatial joins using R-trees: breadth-first traversal with global optimizations, Proc. VLDB (1997) 396–405.
[8] D. Knuth, The Art of Computer Programming, vol. 3. Sorting and Searching, Addison Wesley, Reading, MA, 1973.
[9] M.L. Lo, C.V. Ravishankar, Spatial joins using seeded trees, Proc. ACM SIGMOD (1994) 209–220.
[10] N. Mamoulis, D. Papadias, Integration of spatial join algorithms for processing multiple inputs, Proc. ACM SIGMOD (1999) 1–12.
[11] N. Mamoulis, D. Papadias, Multiway spatial joins, ACM Trans. Database Syst. (TODS) 26 (4) (2001) 424–475.
[12] J.A. Orenstein, Spatial query processing in an object-oriented database system, Proc. ACM SIGMOD (1986) 326–336.
[13] D. Papadias, N. Mamoulis, V. Delis, Algorithms for querying by spatial structure, Proc. VLDB (1998) 546–557.
[14] D. Papadias, N. Mamoulis, Y. Theodoridis, Constraint-based processing of multiway spatial joins, Algorithmica 30 (2) (2001) 188–215.
[15] H.-H. Park, G.-H. Cha, C.-W. Chung, Multi-way spatial joins using R-trees: methodology and performance evaluation, Proc. SSD (1999) 229–250.
[16] H.-H. Park, Y.-J. Lee, C.-W. Chung, Spatial query optimization utilizing early separated filter and refinement strategy, Inf. Syst. 25 (1) (2000) 1–22.
[17] H.-H. Park, Early Separated Filter/Refinement Strategies and Multi-way Spatial Joins for Spatial Query Optimization, PhD Thesis, KAIST, 2001.
[18] J.M. Patel, D.J. DeWitt, Partition based spatial-merge join, Proc. ACM SIGMOD (1996) 259–270.
[19] F.P. Preparata, M.I. Shamos, Computational Geometry: An Introduction, Springer, Berlin, 1985.
[20] T. Sellis, N. Roussopoulos, C. Faloutsos, The R^+-tree: a dynamic index for multidimensional objects, in: Proceedings of VLDB, 1987.
[21] Y. Theodoridis, D. Papadias, Range queries involving spatial relations: a performance analysis, Proc. COSIT (1995) 537–551.
[22] Y. Theodoridis, E. Stefanakis, T. Sellis, Cost models for join queries in spatial databases, Proc. IEEE ICDE (1998) 476–483.
[23] US Bureau of the Census, Washington, DC, TIGER/Line Files, Technical Documentation, 1995.
[24] P. Valduriez, Join indices, ACM Trans. Database Syst. 12 (2) (1987) 218–246.