Concise Papers

Spatial Join Processing Using Corner Transformation

Ju-Won Song, *Member*, *IEEE*,
Kyu-Young Whang, *Senior Member*, *IEEE*,
Young-Koo Lee, *Student Member*, *IEEE*,
Min-Jae Lee, *Student Member*, *IEEE*,
and Sang-Wook Kim, *Member*, *IEEE*

Abstract—Spatial join finds pairs of spatial objects having a specific spatial relationship in spatial database systems. Since spatial join is a fairly expensive operation, we need an efficient algorithm taking advantage of the characteristics of available spatial access methods. In this paper, we propose a spatial join algorithm using corner transformation and show its excellence through experiments. To the extent of authors' knowledge, the spatial join processing using corner transformation is new. In corner transformation, two regions in one file joined with two adjacent regions in the other file share a large common area. The proposed algorithm utilizes this property in order to reduce the number of disk accesses for spatial join. Experimental results show that the performance of the algorithm is generally better than that of the R*-tree based algorithm proposed by Brinkhoff et al. This is a strong indication that corner transformation is a promising category of spatial access methods and that spatial operations can be performed better in the transform space than in the original space. This reverses the common belief that transformation will adversely effect the clustering. We also briefly mention that the join algorithm based on corner transformation has a nice property of being amenable to parallel processing. We believe that our result will provide a new insight towards transformation-based processing of spatial operations.

Index Terms—Spatial join, GIS, spatial databases, corner transformation.

1 Introduction

New database applications such as geographic information systems (GISs) require efficient management of *spatial objects* (hereafter, we simply call them *objects*). Recently, there have been a number of research efforts on spatial database systems for managing these objects [4]. An excellent survey on spatial database systems is presented in [5].

In spatial database systems, an efficient spatial join operation should be provided [5]. *Spatial join* (we simply call it *join*) of two files R and S is defined as $R \bowtie_{i\theta j} S$ where i and j are columns of R and S having spatial data types, and θ is a spatial operator [4]. Since spatial join is very costly [2], [4], spatial join algorithms should be carefully designed by taking advantage of the characteristics of the underlying spatial access methods (*SAMs*).

Spatial join algorithms recently developed have focused on the cases where both files are indexed by SAMs [5]. Orenstein [10] proposed an algorithm for joining two object sets transformed by the z-order and then indexed by one-dimensional access methods such as the B-tree. Gunther [4] presented an algorithm using the *generalization tree*, an abstraction of the R-tree, and derived its cost model. Brinkhoff et al. [2], [3] proposed algorithms using the R*- tree [1]. Lo and Ravishankar [8] proposed an algorithm that constructs a SAM called the *seeded tree* on the fly in case one object set is not indexed by a SAM. Huang and Jing [7] proposed another R-tree based algorithm that adopts the breadth-first tree traversal technique. Their Combo 1 strategy performs better than the Brinkhoff's algorithm for small buffers, but worse for large buffers; Combo 2 strategy the other way around.

Some algorithms are not SAM-based. Examples are Partition Based Spatial-Merge Join [12] and Spatial Hash-Join [9]. These methods are best used when there are no pre-existing indexes for the spatial data. When indexes exist for both files, however, the Parition Based Spatial-Merge Join is slower than the Brinkhoff's algorithm [12]. Spatial Hash-Join is claimed to be faster than the Brinkhoff's algorithm in terms of the weighted I/O cost, which assumes that the random disk access cost is five times the sequential access cost. If this ratio were one, however, the algorithm would be at least three times slower than the Brinkhoff's algorithm [9].

The transformation technique [14], a category of SAMs, transforms objects in the original space (*o-space*) into points in the transform space (*t-space*) using parameters that represent the shape of each object. The transformed points are then managed by a multidimensional point access method (*PAM*). A typical transformation technique is *corner transformation* [14]. In this technique, the minimum bounding rectangle (*MBR*) of each object is used for the key of the object. The *MBR* of an object in the o-space is the rectangle having sides parallel to the axes that minimally encloses the object. Corner transformation transforms each object in the two-dimensional o-space into a single point in the four-dimensional t-space by using the four parameters that are the coordinates of the MBR's lower-left and upper-right points.

It has been believed that transformation techniques degrade the performance of spatial query processing since they cannot retain o-space's spatial proximity among objects in the t-space [16]. However, a recent study [11] have refuted the belief, and we also provide a positive result in this direction. In this paper, we propose an efficient spatial join algorithm using corner transformation and show its excellence through experiments. We also show that corner transformation is a promising category of SAMs by comparing its performance for join with that of the R*-tree. In our spatial join algorithm using corner transformation, the two regions of the file S that are to be joined with two adjacent regions of the file R have considerably overlapping areas. The algorithm exploits this property to speed up the join processing by determining the order of accessing data pages in the overlapping area assuming LRU buffer replacement. Since most algorithms published have certain advantages and disadvantages compared with the R-tree based algorithm by Brinkhoff et al., in this paper, we use the Brinkhoff's algorithm as the standard one with which we compare the performance of our algorithm.

The rest of this paper is organized as follows. Section 2 describes the characteristics of corner transformation. Section 3 explains how corner transformation can be used for spatial join processing and proposes a new spatial join algorithm based on corner transformation. Section 4 defines the MBR-MLGF, an enhancement of the Multi-Level Grid File [18], [19] and presents the experimental results for performance evaluation of our algorithm that uses the MBR-MLGF as the underlying PAM. In Section 5, we summarize and conclude the paper.

- *J.-W. Song, K.-Y. Whang, Y.-K. Lee, and M.-J. Lee are with the Department of Computer Science and Advanced Information Technology Research Center, Korea Advanced Institute of Science and Technology, South Korea.*
  E-mail: {jwsong, kywhang, yklee, mjlee}@mozart.kaist.ac.kr.
- *S.-W. Kim is with the Department of Information and Telecommunications Engineering, Kangwon National University, South Korea.*

Manuscript received 25 Aug. 1997; revised 20 Nov. 1997.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 105544.

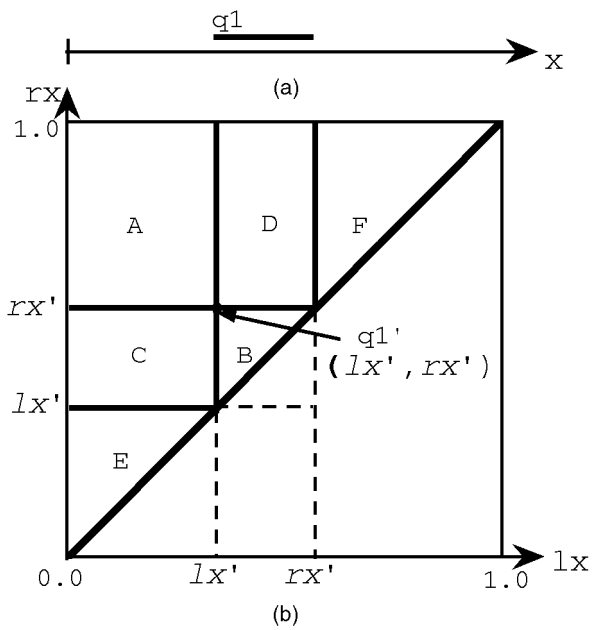

Fig. 1. Partitioning areas according to spatial relationships: (a) query region q1 in the o-space; (b) area partition of the t-space.

2 Corner Transformation

We introduce corner transformation in this section. For simplicity, we first discuss the case of the two-dimensional t-space transformed from a one-dimensional normalized o-space whose size is one.

Corner transformation maps an object in the one-dimensional o-space into a point in the two-dimensional t-space. The left end of an object is mapped to the horizontal (lx) axis and the right end to the vertical (rx) axis in the t-space. We define the area in the t-space where the objects can be mapped as the *Transformed Objects Placing Area* (TOPA). Transformed objects are placed in the upper part of the diagonal in the t-space since the *rx-value* (the coordinate of the right end) of an object is larger than the *lx-value* (that of the left end). Therefore, the TOPA is contained in the upper half triangle of the t-space.

Since most objects managed by a GIS are much smaller than the o-space, transformed objects are typically distributed in the narrow strip above the diagonal. Therefore, the distribution of the transformed objects is skewed with high correlation between the lx and rx axes [11]. Thus, to use corner transformation, we need a robust PAM that can handle the skewed distribution efficiently.

The Multilevel Grid File (MLGF) [18], [19], LSD-tree [6], and Buddy-tree [15] are examples of such PAMs. Due to their local splitting strategy [20], their directory sizes increase almost linearly in the data file size regardless of data distribution. In Section 4, we use the MBR-MLGF, an enhancement of the MLGF, as the underlying PAM for performance evaluation.

We now describe a method of transforming region queries in the o-space into range queries in the t-space. A *region query* selects the objects that have a specific spatial relationship, such as containment and intersection, with the given query region. To process a region query we should identify the area in the t-space containing the objects that have a specific spatial relationship with the given query region in the o-space. Fig. 1a and Fig. 2b show a query region q1 in the o-space and its corresponding point $q1'$ in the t-space. The upper triangle, which includes all the transformed objects, is partitioned into six areas A to F according to the spatial relationships with respect to the query region. Boundary values of these areas are determined by the coordinates of $q1'$. These areas have the following characteristics:

- Area A contains all the objects that enclose q1 in the o-space since their lx-values are less than that of $\mathrm{q1'(lx')}$ and the rx-values greater than that of $\mathrm{q1'(lx')}$.

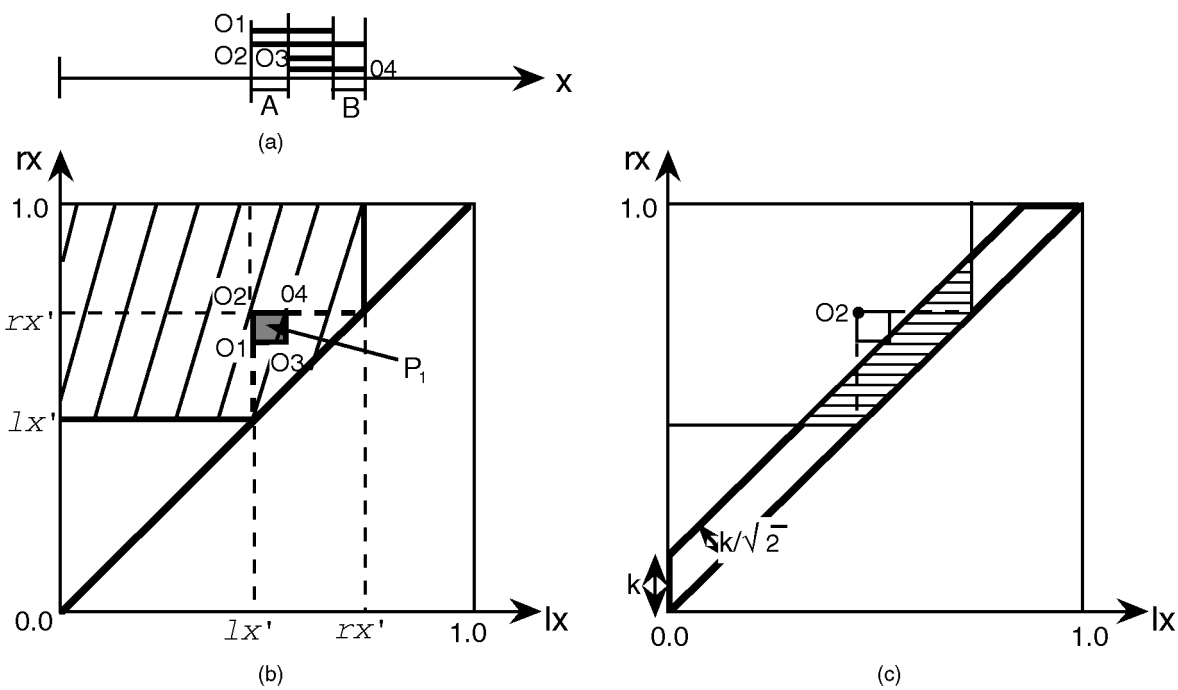

Fig. 2. A spatial join window: (a) objects in the o-space; (b) region $\mathrm{P_1}$ in the file R and $SJW(\mathrm{P_1})$ in file S; (c) $SJW(\mathrm{P_1})$ restricted by the TOPA, k being the size of the largest object in the o-space.

- Area B contains all the objects that are enclosed by q1 since their lx-values are greater than lx' and the rx-values less than rx'.
- Area C contains all the objects that only intersect with the left end of q1 since their lx-values are less than lx' and the rx-values between lx' and rx'.
- Area D contains all the objects that only intersect with the right end of q1 since their lx-values are between lx' and rx' and the rx-values greater than rx'.
- Area E contains all the objects that reside to the left of q1 since their rx-values are less than lx'.
- Area F contains all the objects that reside to the right of q1 since their lx-values are greater than rx'.

Region queries can be transformed into range queries in the t-space using this area partition. For example, a *region intersection query* selects all the objects that intersect with any part of a given query region. This query with the query region q1 in the o-space can be processed by accessing the union of the areas A, B, C, and D in the t-space. These areas can be accessed by processing the range query $(\mathrm{lx} \leq rx' \text{ and } \mathrm{rx} \geq lx')$ in the upper triangle of the t-space.

The space partition introduced here is a variation of the scheme proposed by Seeger and Kriegel. Seeger and Kriegel [14] partitioned the t-space for center transformation into six areas. The result is identical to Fig. 1b except it is rotated by $\pi/4$. The boundaries of the partitioned areas in corner transformation are parallel to the axes, whereas those in center transformation are parallel to the diagonals [14]. Since region splitting in most PAMs is done parallel to the axes, we can reduce the number of regions that cause false-drops if the boundaries of the search region are parallel to the axes. Corner transformation has the advantage for spatial join processing in this regard.

3 Spatial Join Algorithm

We now propose a new spatial join algorithm using corner transformation. For the convenience of explanation, we tentatively assume that the TOPA consists of grid-like cells of the same size, each of which corresponds to a data page of the PAM. However, the proposed algorithm is applicable to all PAMs regardless of their space partition strategies. We also assume LRU buffer replacement.

We first introduce a new notion of the *spatial join window* and then identify relationships among spatial join windows for adjacent regions. The algorithm takes advantage of these relationships to optimize the join in the number of disk accesses. We then propose a join algorithm for the one-dimensional o-space and extend it for the two-dimensional o-space. The algorithm only considers intersection—the most commonly used operator for spatial join. However, the basic ideas of the algorithm can be applied to other spatial operators such as enclosure and containment.

3.1 Spatial Join Window

Let the t-spaces of files R and S be TS(R) and TS(S), respectively. We define the *Spatial Join Window* for a rectangular region P_1 in TS(R), *SJW*(P_1), as the minimum region in TS(S) where all the objects intersecting with the objects in P_1 can reside. We define the *Spatial Join Window Pages* for P_1, $SJWP(\mathrm{P}_1)$, as the set of pages corresponding to $\mathrm{SJW}(\mathrm{P}_1)$.

Fig. 2a shows the objects O1, O2, O3, and O4, which have end points of the range A as the lx-values and those of the range B as the rx-values. Fig. 2b shows the transformed objects represented as four corner points of P_1. We know that O2, representing the upper-left point of P_1, is the largest object in P_1. If the lx-value of an object is in A and the rx-value in B, the transformed object is located in P_1.

In Fig. 2, $\mathrm{SJW}(\mathrm{P}_1)$ can be obtained as follows. To intersect with some objects in P_1, objects in the file S must intersect with the object O2 since it is the largest object enclosing all the other objects in P_1. Fig. 2b shows $\mathrm{SJW}(\mathrm{P}_1)$ (here, the P_1 in TS(R) and $\mathrm{SJW}(\mathrm{P}_1)$ in TS(S) are depicted together). If the coordinates of O2 are (lx', rx'), $\mathrm{SJW}(\mathrm{P}_1)$ is the intersection area between the range '(lx $\leq rx'$ **and** rx $\geq lx'$)' that is above the diagonal of the t-space. Since the TOPA has a narrow trapezoid shape as explained in Section 2, $\mathrm{SJW}(\mathrm{P}_1)$ becomes smaller than the one shown in Fig. 2b. Fig. 2c shows $\mathrm{SJW}(\mathrm{P}_1)$ in the TOPA when the size of the largest object is k.

3.2 Relationships Among Spatial Join Windows

A *strip STR* in the TOPA is defined as the minimum rectangle that contains all the grid cells with the same projection range for an axis. A *horizontal strip* has the same projection range for the rx axis, and a *vertical strip* for the lx axis. The set of pages corresponding to the cells in a strip $\mathrm{STR}_l(l = 1, 2, ..., \mathrm{N}, \mathrm{N}: \text{the number of strips})$ are defined as *strip pages*, $SP(\mathrm{STR}_l)$. In Fig. 3a, the strip consisting of cells R1, R2, and R3 is a horizontal strip, and the one consisting of cells R7, R5, and R3 a vertical strip. We now identify relationships among SJWs in TS(S) for adjacent regions in TS(R).

First, the SJWs for two adjacent strips have a large common area since the upper-left points of the strips are also adjacently placed. Fig. 3b represents the relationship between two adjacent horizontal strips HSTR_1 and HSTR_2. Here, $\mathrm{SJW}(\mathrm{HSTR}_1)$ is the union of regions A and B, and $\mathrm{SJW}(\mathrm{HSTR}_2)$ the union of B and C. Hence, the common area between the SJWs for the two adjacent strips is B. The dashed line parallel to the diagonal line denotes the upper boundary of the TOPA.

Second, the SJW for a cell farer from the diagonal in a strip contains the SJW for a cell nearer since one of the coordinate values for the upper-left point of the former is farther from the diagonal than that of the latter, and the other is the same. For example, in Fig. 3b, SJW(R1), the union of A and B, contains SJW(R2), which is B.

A *common-SJW*$(\mathrm{STR}_l, \mathrm{STR}_{l+1})$ and an *added-SJW*$(\mathrm{STR}_l, \mathrm{STR}_{l+1})$ are common and added areas of the SJW for STR_{l+1} compared with the one for STR_l. Likewise, SJWPs corresponding to these types of SJWs are *common-SJWP*$(\mathrm{STR}_l, \mathrm{STR}_{l+1})$ and *added-SJWP*$(\mathrm{STR}_l, \mathrm{STR}_{l+1})$. In Fig. 3b, common-SJW$(\mathrm{HSTR}_1, \mathrm{HSTR}_2)$ is B, and added-SJW$(\mathrm{HSTR}_1, \mathrm{HSTR}_2)$ is C. In Fig. 3c, common-SJWP$(\mathrm{HSTR}_1, \mathrm{HSTR}_2)$ is {S5, ..., S15, S17, S18, S20}, and added-SJWP$(\mathrm{HSTR}_1, \mathrm{HSTR}_2)$ is {S22, S21, S19, S16}.

3.3 Join Algorithm for One-Dimensional O-Space

Here, we first introduce a skeleton join algorithm and then propose the complete algorithm that further optimizes page accesses by using the relationships among SJWs identified above.

The skeleton algorithm performs a join by partitioning the TOPA of the file R into N disjoint strips and doing all the subjoins between strips $\mathrm{STR}_l(l = 1, 2, ..., N,$ where N is the number of strips) and $\mathrm{SJW}(\mathrm{STR}_l)$. Fig. 4 shows the skeleton join algorithm *s_join*1. We assume that the TOPA is partitioned into horizontal strips and the strips are processed bottom up, i.e., in the *upward order*.[1] The algorithm consists of three loops: The outer loop (loop (1)) that selects a strip STR_l, the middle loop (loop (2)) that selects a page in SJWP(STR_l), and the inner loop (loop (3)) that selects a page in SP(STR_l). A subjoin of STR_l finds all the pairs

1. There may be four different processing orders: The upward order and downward orders based on horizontal strips, and the rightward order and leftward orders based on vertical strips. We simply choose the upward order because these methods have the same performance if the objects are uniformly distributed in the TOPA.

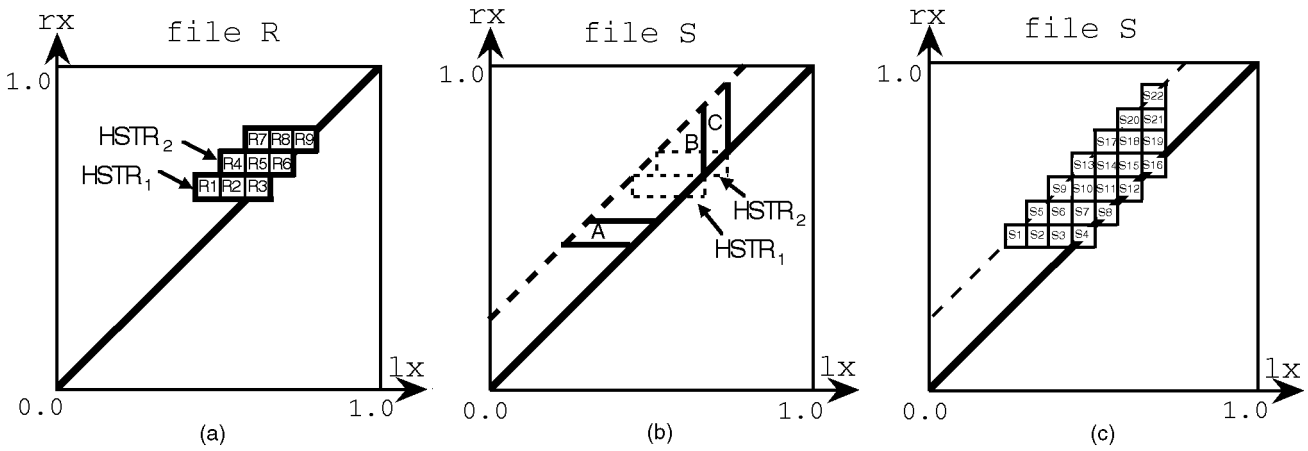

Fig. 3. Relationships among the SJWs: (a) strips of file R; (b) SJWs of file S; (c) SJWPs of file S.

```
s_join1(file_R, file_S)
{
(1)    for each horizontal strip in file_R, STR_1, in the upward order
(2)          for each page of SJWP(STR_1) in file_S, o_page
(3)                for each page of SP(STR_1), i_page
                         if o_page is a page of SJWP(i_page)
                                call join_pages(o_page, i_page);
                   end_for /* for SPs */
              end_for /* for SJWPs */
       end_for /* for strips */
}
```

Fig. 4. The skeleton algorithm *s_join*1.

of intersecting objects from SP(STR_l) of the file R and SJWP(STR_l) of the file S. The outer loop first selects a strip. Since the number of pages in an SP is usually smaller than that in the corresponding SJWP, the inner loop selects a page of the SP after the middle loop selects a page of the SJWP. As a result, each page in SP(STR_l) is accessed only once for the subjoin if SP(STR_l) can be retained in the buffer. Since a page in an SP of the file R is selected in the inner loop and a page in an SJWP of the file S in the middle loop, we define the file R as the *inner file* and the file S as the *outer file*.

All the pages in SP(STR_l) do not join with all the pages in SJWP(STR_l) since some page of SJWP(STR_l) may not belong to the SJWP that corresponds to a page of SP(STR_l). If *o_page* is a page in the SJWP for a page *i_page* in the SP, the two pages are joined by using *join_pages()*. Here, we omit detailed description of join_pages() since the refinement algorithms in [2], [3], [12] are directly applicable to this algorithm.

For the example in Fig. 3, s_join1 is processed as follows. During the subjoin for the strip $HSTR_1$, each page in SJWP($HSTR_1$) ({S1, S2, ..., S15, S17, S18, S20}) and each page in SP($HSTR_1$) ({R1, R2, R3}) are joined in the inner loop using join_pages () if S_i(i = 1, 2, ..., 15, 17, 18, 20) is a page in the SJWP for R_j(j = 1, 2, 3). Doing subjoins for all the remaining strips in the same manner completes the join.

If the given buffer size is not large enough to retain all the pages of common-SJWP(STR_l, STR_{l+1}) or all the pages of SP(STR_l), we need to use specific orders for accessing the pages in the SJWP or SP to reduce the number of disk accesses. Our strategy is to access first the pages of common-SJWP or SP that remain in the buffer. We will explain how this strategy is implemented in the complete algorithm.

The complete algorithm accesses pages in SJWP(STR_{l+1}) in the reverse order of of accessing those in SJWP(STR_l). This reduces the number of disk accesses since we first process the pages of common-SJWP(STR_l, STR_{l+1}) remaining in the buffer. Similarly, during a subjoin for STR_l, we access pages in SP(STR_l) to join with a page in SJWP(STR_l) in the reverse order of accessing them to join with the prior page in SJWP(STR_l). This order allows utilizing the pages in SP(STR_l) remaining in the buffer.

Fig. 5 shows the complete algorithm *s_join*2. As in s_join1, we assume that horizontal strips are processed in the upward order. The algorithm consists of the initialization part and three loops. In initialization (step (1)), *i_order* and *o_order* are initialized. The *i_order* controls the access order of pages in an SP, and the *o_order* in an SJWP. In s_join2, the initial order of accessing the pages in an SP is the far-near order,[2] and that of accessing the pages in an SJWP is the row-major order. The *far-near order* accesses pages from the furthest cell to the nearest one in a strip. In the *row-major order*, the SJWP is divided into multiple rows; the pages in each row are accessed in the far-near order, and the rows in the upward order. The remaining parts of the algorithm is the same as s_join1.

2. Since the performance of the algorithm is independent of any initial access order for pages in an SP, we simply use the far-near order.

```
(1)     i_order = FORWARD;
        o_order = FORWARD;

s_join2(file_R, file_S)
{
(2)     for each horizontal strip in file_R, STR1, in the upward order
              if o_order == FORWARD
                    begin
(3)                       for each page of SJWP(STR1) in file_S, o_page,
                               in the row-major order
                                 call internal_loop(i_strip, o_page);
                          end_for /* for SJWP */
                          o_order = BACKWARD;
                    end_begin
              else
                    begin
(4)                       for each page of SJWP(STR1) in file_S, o_page,
                               in the reverse of the row-major order
                                 call internal_loop(i_strip, o_page);
                          end_for  /* for SJWP */
                          o_order = FORWARD;
                    end_begin
        end_for /* for strip */
}

internal_loop(STR1, o_page)
{
        if i_order == FORWARD
              begin
(5)                 for each i_page that belongs to SP(STR1),
                         in the far-near order
                           if o_page is a page of SJWP(i_page)
                                 call join_pages(o_page, i_page);
                    end_for
                    i_order = BACKWARD;
              end_begin
        else
              begin
(6)                 for each i_page that belongs to SP(STR1),
                         in the reverse of the far-near order
                           if o_page is a page of SJWP(i_page)
                                 call join_pages(o_page, i_page);
                    end_for
                    i_order = FORWARD;
              end_begin
}
```

Fig. 5. The complete algorithm s_join2.

3.4 Extension for Two-Dimensional O-Spaces

We now extend the algorithm for the two-dimensional o-space since many real applications such as GISs typically process objects in the two-dimensional o-space. The basic structure of the extended algorithm is similar to that of the one-dimensional algorithm. The important issues to be addressed for the extended algorithm are: 1. how to define strips in the four-dimensional t-space, 2. how to determine the order of processing the strips, 3. how to determine the order of accessing the pages in an SJWP, and 4. how to determine the order of accessing the pages in an SP.

Let the two-dimensional o-space *x-y* be transformed into the four-dimensional t-space *lx-rx-ly-ry*. Fig. 6a and Fig. 6b show the *lx-rx* and *ly-ry* planes that are t-spaces of *x* and *y* axes, respectively. We assume that horizontal strips are selected in the *lx-rx* plane and the *ly-ry* plane of the inner file as in Fig. 6a and Fig. 6b. Then, when the four-dimensional t-space is projected onto *ry-rx* plane, the plane has a grid-like partition as in Fig. 6c. A grid cell in Fig. 6c represents the projected region of a *hyperstrip*, which consists of a pair of horizontal strips in the *lx-rx* and *ly-ry* planes. Each hyperstrip has adjacent hyperstrips in two directions along the *ry* and *rx* axes. For example, in Fig. 6c, a hyperstrip $STR_{i,j}$ consists of a pair of two horizontal strips ($HSTR_{xi}$, $HSTR_{yj}$).

Now, we need to determine the order of processing subjoins for these hyperstrips. This order has a major effect on the performance of the join. Suppose we process the subjoins for hyperstrips in the row-major order. Then, we have the following performance problem. In Fig. 6c, if subjoins for hyperstrips in each row of *ry-rx* plane are processed from left to right, the size of the added-SJW is very small since the SJWs for the two adjacent hyperstrips have different range values only in the *ly-ry* plane. However, processing a subjoin for the first hyperstrip of a row right after the last hyperstrip of the previous row incurs a sudden jump in the added-SJW since the SJWs for the two distant hyperstrips have different range values for both the *lx-rx* and *ly-ry* planes. As a result, the performance of the join degrades.

To resolve this problem, we use the order such that the hyperstrips to be processed consecutively change the range values only in one of the *lx-rx* or *ly-ry* planes. Space filling curves that

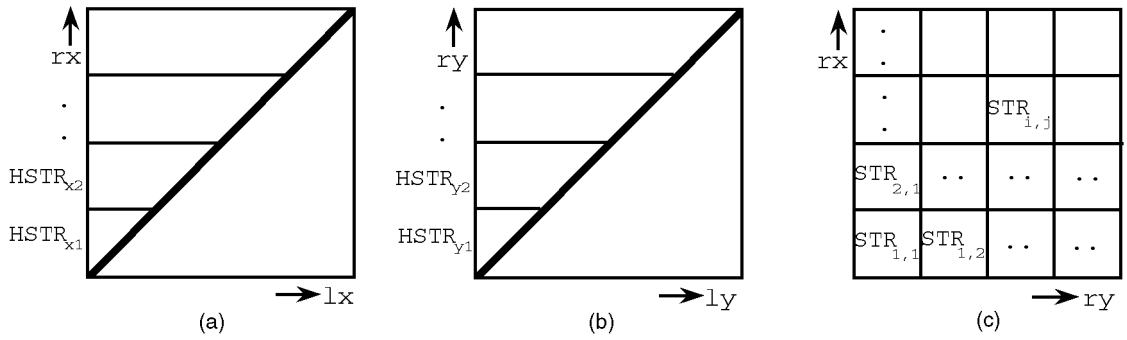

Fig. 6. Four-dimensional t-space: (a) lx-rx plane. (b) ly-ry plane. (c) ry-rx plane.

change the direction only in one axis at a time can be used for this order; some examples are the row-prime order and the Peano-Hilbert order [13]. We use the Peano-Hilbert order in the extended algorithm since it can be recursively applied to all levels of the MBR-MLGF that is used for our experiment.

For the orders of accessing pages in an SJWP and these in a hyperstrip in four-dimensional t-space, we directly extend the row-major order and far-near order used in the one-dimensional algorithm. The row index of a plane, *rx* of the *lx-rx* plane or *ry* of the *ly-ry* plane, changes first initializing the outer loop, and then the row index of the other plane changes initializing the inner loop. Let the coordinates of the upper-left point of a hyperstrip STR$_{i,j}$ be (lx_i, rx_i, ly_j, ry_j) in the inner file R. SJW(STR$_{i,j}$) can be obtained as the range (lx $\leq rx_i$ **and** rx $\geq lx_i$ **and** ly $\leq ry_j$ **and** ry $\geq ly_j$) in TS(S) by extending the one-dimensional case. Similarly, for accessing pages in a hyperstrip, we extend the one-dimensional case by using an inner loop and an outer loop.

4 Performance Evaluation

We have described the algorithm assuming a grid-like partition of the space. However, in practice, a specific PAM with its own space partitioning strategy should be used for the algorithm. Such a PAM should be able to handle skewed data efficiently. For this purpose, we use the MBR-MLGF. We now explain the characteristics of the MLGF, extend it to the MBR-MLGF, and present experimental results.

The MLGF is a balanced tree and consists of a multilevel directory and data pages. Each directory level reflects the status of space partition. A directory entry consists of a *region vector* and a *pointer* to a data page or to a lower-level directory page. A region vector in an *n*-dimensional MLGF consists of *n*-hash values that uniquely identify the region. The position, shape, and size of a region are reflected in the region vector. The *i*th hash value of a region vector is the common prefix of the hash values for the *i*th attribute of all the possible records that belong to the region. A region corresponding to a higher-level directory entry contains all the regions in the subtree whose root is the page pointed by the entry.

By splitting and merging pages, the MLGF adapts to dynamic environments where record insertions and deletions occur. If the data page overflows, the corresponding region is split into two equal-sized regions. The MLGF employs the *local splitting strategy* [20] that splits a region locally, rather than across the entire hyperplane, when the corresponding page overflows. The local splitting strategy maintains the policy of having one directory entry correspond to one page; thus, the strategy prevents the MLGF from creating unnecessary directory entries. The MLGF does not create directory entries for empty regions. As a result, the directory size of the MLGF is linearly dependent on the number of objects inserted regardless of data distribution or correlation between the attributes [20]. Thus, the MLGF gracefully adapts to highly skewed and correlated distributions that frequently occur in corner transformation.

The MBR-MLGF is an extension of the MLGF for managing spatial objects. Each directory entry of the MBR-MLGF maintains the minimum of lx-values (*min-lx*) and the maximum of rx-values (*max-rx*) of the objects (without additional storage overhead) in the corresponding region. These values represent the MBR containing the objects in the o-space.

By maintaining the min-lx and max-rx in the directory entries, the MBR-MLGF has the following two benefits for join processing. First, we can reduce the size of the SJW for a strip since the upper-left point of the strip for the MBR-MLGF is nearer to the diagonal than that for the MLGF. Second, the number of pages in an SJWP is reduced. Details of these benefits are referred to in [17].

Since space partition in each directory level of the MBR-MLGF is not always grid-like due to nonuniformity of data distribution, we need to modify our strip selection algorithm. We also need to handle multilevel nature of the directory. The recursive definition of Peano-Hilbert order helps solve this problem. Details of these modifications can be found in [17].

4.1 Experimental Results

We now perform experiments to evaluate the performance of the algorithm s_join2 extended for two-dimensional cases using the MBR-MLGF as the underlying PAM. We then compare the results with those of the R*-tree based join algorithm proposed by Brinkhoff et al. [2]. For all the experiments, we use the number of disk accesses for data pages as the performance measure. We do not consider CPU cost since we focus on the filter step, which is I/O-intensive, rather than the refinement step, which generally is known to be CPU-intensive [12].

For our experiment, we use real data in the two-dimensional o-space. The data sets for the two files joined are identical to those used by Brinkhoff et al. [2]. The first file contains the MBRs of the streets in an area of California whereas the second those of the rivers and railway tracks. They consist of 131,461 and 128,791 MBRs, respectively, and have 86,094 intersecting MBR pairs. We set the maximum blocking factor of a data page for the MBR-MLGF and R*-tree to the same value (=36) for a fair comparison. The MBR-MLGF shows a somewhat lower storage utilization making the file larger than that of the R*-tree. The storage utilization becomes worse as the dimensionality increases. For the experiment, in order to maintain the storage utilization similar to that of the R*-tree, we use cyclic splitting only for two out of four axes: rx and ry.

Fig. 7 and Fig. 8 show the results of the experiment. For the R*-tree, we choose the algorithm using *plane-sweeping with pinning* that has the best performance among those

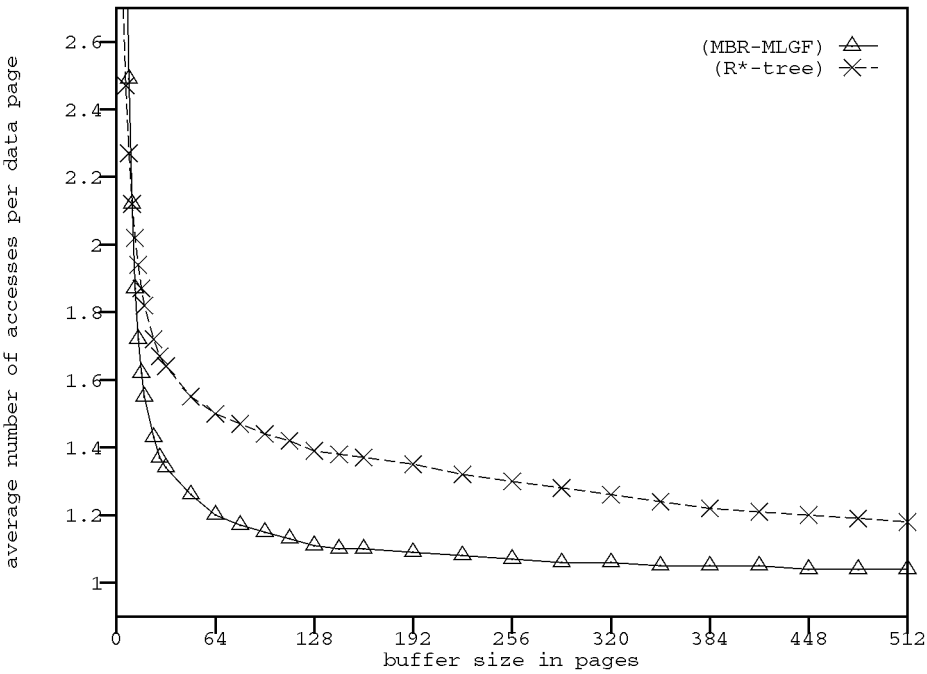

Fig. 7. The number of page accesses for the real data normalized by the total number of data pages in the file.

proposed in [2]. The numbers of data pages in the MBR-MLGF and R*-tree are 11,699 and 10,359, and their storage utilizations about 65 and 70 percent, respectively. Fig. 7 shows the number of page accesses normalized by the total number of data pages in the files. Thus, the number represents average accesses for each data page. An average access of 1.0 represents the theoretically optimal performance. Fig. 8 shows the absolute numbers of page accesses.

The result in Fig. 7 shows that the proposed algorithm has a better performance over the entire range of the buffer size with small exceptions when the buffer is extremely small. This result indicates that corner transformation and the proposed join algorithm performed in the t-space handle spatial join more efficiently than the R*-tree based algorithm performed in the o-space. The result in Fig. 8 shows a slight degradation in the performance of the MBR-MLGF. The reason for performance degradation is lower storage utilization. However, the R*-tree incurs a large processing cost to increase the storage utilization by employing a complicated splitting strategy and the forced reinsert policy [1]. For the proposed algorithm, the 1.1, 1.15, and 1.2-times buffer sizes [2] are 158, 95, and 65 pages, respectively, which are 1.35, 0.81, and 0.56 percent, respectively, of the number of the MBR-MLGF's data pages for the two files. In contrast, for the R*-tree case, they are 1,300,

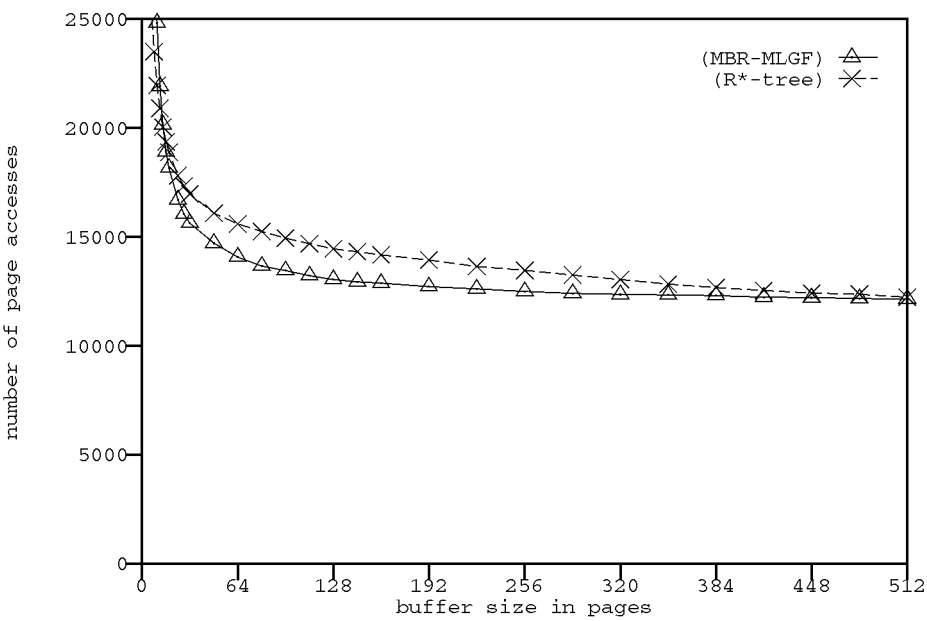

Fig. 8. The number of page accesses for the real data (unnormalized).

667, and 441 pages, respectively, which are 12.55, 6.44, and 4.26 percent, respectively, of the number of the R*-tree's data pages for the two files.

The principal reason why the proposed algorithm has a performance better than that of the R*-tree based join algorithm is as follows. As discussed in Section 3.3, in order to reduce the number of disk accesses, s_join2 determines the order of accessing the pages in SPs and SJWPs by taking advantage of the relationships among SJWs and the property of the LRU buffer replacement policy. This is a kind of global optimization. In contrast, the R*-tree based join algorithm determines the order of accessing the pages pointed by directory entries in the two nodes to be joined at the same level by employing the concepts of the plane-sweeping, local z-ordering, and plane-sweeping with pinning. These are largely local optimization based on *spatial locality* [2]. Thus, our algorithm has some advantage over the other. We also have performed comparatative experiments with synthetic two-dimensional data sets that have controlled distributions. They have tendencies similar to those in Experiment 2. Because of space limitation, we omit the detailed results, which are referred to [17].

5 Conclusions

Spatial join, which finds object pairs that have a specific spatial relationship, is a fairly expensive operation in spatial database systems. In this paper, we have proposed a new spatial join algorithm using corner transformation and have verified its excellence through experiments.

We first have introduced the new concept of the spatial join window, and then identified useful relationships among the spatial join windows for adjacent regions and adjacent strips. We have then proposed a spatial join algorithm that optimizes the performance taking advantage of the relationships under the LRU buffer replacement policy. We also have extended the algorithm to the two-dimensional o-space. Finally, we have proposed the MBR-MLGF as an enhancement of the MLGF and adapted the algorithm to use it as the underlying point access method.

To prove the viability of the proposed algorithm, we have performed a comparative experiment with the R*-tree using the real and synthetic data sets. The results show that our algorithm has a performance generally better than that of the R*-tree based one. This is a strong indication that corner transformation is a promising category of spatial access methods, and the spatial operations can be performed better in the transform space than in the original space. This reverses the common belief that transformation will adversely affect the clustering.

The advantage of the proposed algorithm is less marked when the results are not normalized because of lower storage utilization of the MBR-MLGF compared with that of the R*-tree. However, the storage utilization is a characteristic of the MBR-MLGF and should not be counted as a characteristic of corner transformation.

Finally, we point out the spatial join based on corner transformation has an additional benefit that it can easily be parallelized by dividing the trapezoidal Transformed Object Placing Area (TOPA) into possibly multiple overlapping segments. The narrower the area is, the easier it is to divide. We note that for a large map to be joined in a GIS, the TOPA will be quite narrow.

To the extent of our knowledge, spatial join algorithms using corner transformation is new, and we believe that our result will provide a new insight towards transformation-based processing of spatial operations.

Acknowledgments

This work was supported partially by the Korean Ministry of Science and Technology (MOST) through the National GIS Project—Development of a Spatial Object Storage System, and partially by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITRC). We are grateful to Thomas Brinkhoff, Daniel Keim, and Hans-Peter Kriegel for providing data and the program for the R*-tree based join algorithm.

References

[1] N. Beckmann, H.-P. Kriegel, and R. Schneider, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. Int'l Conf. Management of Data,* pp. 322–331, ACM SIGMOD, 1990.

[2] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, "Efficient Processing of Spatial Joins Using R-Trees," *Proc. Int'l Conf. Management of Data,* pp. 237–246, ACM SIGMOD, May 1993.

[3] T. Brinkhoff, H.-P. Kriegel, R. Schneider, and B. Seeger, "Multi-Step Processing of Spatial Joins," *Proc. Int'l Conf. Management of Data,* pp. 197–208, ACM SIGMOD, May 1994.

[4] O. Günther, "Efficient Computation of Spatial Joins," *Proc. Ninth Int'l Conf. Data Eng.,* pp. 50–59, 1993.

[5] O. Güting, "An Introduction to Spatial Database Systems," *VLDB J.,* vol. 3, no. 4, pp. 357–399, Oct. 1994.

[6] A. Henrich, H.W. Six, and P. Widmayer, "The LSD Tree: Spatial Access to Multidimensional Point and Nonpoint Objects," *Proc. 15th Int'l Conf. Very Large Data Bases,* pp. 45–53, 1989.

[7] Y.W. Huang and N. Jing, "Spatial Joins Using R-Trees: Breadth-First Traversal with Global Optimizations," *Proc. 23rd Int'l Conf. Very Large Data Bases,* pp. 396–405, 1997.

[8] M.L. Lo and C.V. Ravishankar, "Spatial Joins Using Seeded Trees," *Proc. Int'l Conf. Management of Data,* pp. 209–220, ACM SIGMOD, May 1994.

[9] M.L. Lo and C.V. Ravishankar, "Spatial Hash-Joins," *Proc. Int'l Conf. Management of Data,* pp. 247–258, ACM SIGMOD, June 1996.

[10] J. Orenstein, "Spatial Query Processing in an Object-Oriented Database System," *Proc. Int'l Conf. Management of Data,* pp. 326–336, ACM SIGMOD, 1986.

[11] B.U. Pagel, H.W. Six, and H. Toben, "The Transformation Technique for Spatial Objects Revisited," *Proc. Third Int'l Symp. Spatial Databases, SSD '93,* 1993.

[12] J.M. Patel and D.J. Dewitt, "Partition Based Spatial-Merge Join," *Proc. Int'l Conf. Management of Data,* pp. 259–270, ACM SIGMOD, June 1996.

[13] H. Samet, *The Design and Analysis of Spatial Data Structures,* Addison-Wesley, 1990.

[14] B. Seeger and H.-P. Kriegel, "Techniques for Design and Implementation of Efficient Spatial Access Methods," *Proc. 14th Int'l Conf. Very Large Data Bases,* pp. 360–371, 1988.

[15] B. Seeger and H.-P. Kriegel, "The Buddy-Tree: An Efficient and Robust Access Method for Spatial Database Systems," *Proc. 16th Int'l Conf. Very Large Data Bases* pp. 590–601, 1990.

[16] H.W. Six and P. Widmayer, "Spatial Searching in Geometric Databases," *Proc. Fourth Int'l Conf. Data Eng.,* pp. 496–503, 1988.

[17] J.W. Song, K.Y. Whang, and S.W. Kim, "Spatial Join Processing Using Corner Transformation," Technical Report CS/TR-96-107, Dept. of Computer Science, KAIST, Dec. 1996.

[18] K.Y. Whang and R. Krishnamurthy, "Multilevel Grid Files," IBM Research Report RC 11516, 1985.

[19] K.Y. Whang and R. Krishnamurthy, "The Multilevel Grid File—A Dynamic Hierarchical Multidimensional File Structure," *Proc. Second Int'l Conf. Database Systems for Advanced Applications,* pp. 449–459, 1991.

[20] K.Y. Whang, S.W. Kim, and G. Wiederhold, "Dynamic Maintenance of Data Distribution for Selectivity Estimation," *VLDB J.,* vol. 3, no. 1, pp. 29–51, Jan. 1994.