Transform-Space View: Performing Spatial Join in the Transform Space Using Original-Space Indexes

Min-Jae Lee, *Student Member*, *IEEE*, Kyu-Young Whang, *Senior Member*, *IEEE*, Wook-Shin Han, *Member*, *IEEE*, and Il-Yeol Song, *Member*, *IEEE Computer Society*

Abstract—Spatial joins find all pairs of objects that satisfy a given spatial relationship. In spatial joins using indexes, original-space indexes such as the R-tree are widely used. An *original-space index* is the one that indexes objects as represented in the original space. Since original-space indexes deal with extents of objects, it is relatively complex to optimize join algorithms using these indexes. On the other hand, *transform-space indexes*, which transform objects in the original space into points in the transform space and index them, deal only with points but no extents. Thus, optimization of join algorithms using these indexes can be relatively simple. However, the disadvantage of these join algorithms is that they cannot be applied to original-space indexes such as the R-tree. In this paper, we present a novel mechanism for achieving the best of these two types of algorithms. Specifically, we propose the new notion of the *transform-space view* and present the *transform-space view join algorithm*. The transform-space view is a virtual transform-space index based on an original-space index. It allows us to "interpret" or "view" an existing original-space index as a transform-space index with no space and negligible time overhead and without actually modifying the structure of the original-space index or changing object representation. The transform-space view join algorithm joins two original-space indexes in the transform space through the notion of the transform-space view. Through analysis and experiments, we verify the excellence of the transform-space view join algorithm. The transform-space view join algorithm always outperforms existing ones for all the data sets tested in terms of all three measures used: the one-pass buffer size (the minimum buffer size required for guaranteeing one disk access per page), the number of disk accesses for a given buffer size, and the wall clock time. Thus, it constitutes a lower-bound algorithm. We believe that the proposed transform-space view can be applied to developing various new spatial query processing algorithms in the transform space.

Index Terms—Transform-space view, adaptive row major order, spatial join, corner transformation, databases.

1 Introduction

Spatial joins (or simply *joins*) find all pairs of objects that satisfy a given spatial relationship. An example of a spatial join is "finding all the roads that overlap with rivers." Since spatial joins require a significant cost of processing, much effort has been made to develop efficient algorithms [4], [12], [20], [25], [28].

Existing spatial join algorithms can be classified into those that use indexes on both files and those that do not. Join algorithms that do not use indexes on both files include the Partition-Based Spatial Merge Join [25], Spatial Hash Join [20], Seed Join [19], Sort/Sweep Spatial Join [8], Unified Approach for Indexed and Non-Indexed Spatial Joins [1], Iterative Spatial Join [13], and Slot Index Spatial Join [21]. These join methods are best used when there is no preexisting index on the files to be joined or there exists an index on only one file. However, when there are indexes on both files, these algorithms are slower than those that use indexes. Therefore, in this paper, we limit our discussion to join algorithms that use indexes on both files (we call them *index join algorithms*).

Index join algorithms are classified into those whose indexes are created in the original-space (*o-space*) and those whose indexes are created in the transform-space (*t-space*). *O-space index join algorithms* use indexes that consider the extents of objects in the o-space. On the other hand, *t-space index join algorithms* use indexes in which objects with extents in the o-space have been transformed into points with no extents in the t-space.

O-space index join algorithms include the Depth-First Traversal R-tree Join [4] and the Breadth-First Traversal R-tree Join [12]. Both are based on the R-tree family [2], [9]. The Depth-First Traversal R-tree Join is a representative algorithm widely used for comparing the performances of other join algorithms. It performs the join by traversing two R-trees in the depth-first order. Since it optimizes only a part of the sequence of accessing disk pages (*page access sequence*) for joining two R-trees, it only achieves local optimization [12], [28]. The Breadth-First Traversal R-tree Join achieves global optimization by optimizing the page

- *M.-J. Lee is with Research Center, Neowiz, Co., Ltd., 159-1 Assem Tower 6th fl., Samsung Dong, Gangnam Gu, Seoul, 135-798, Korea. E-mail: mjlee@mozart.kaist.ac.kr.*
- *K.-Y. Whang is with the Department of Computer Science and Advanced Information Technology Research Center (AITrc), Korea Advanced Institute of Science and Technology (KAIST), 373-1 Koo-Sung Dong, Yoo-Sung Ku, Daejeon, 305-701, Korea. E-mail: kywhang@mozart.kaist.ac.kr.*
- *W.-S. Han is with the Department of Computer Engineering, Kyungpook National University, 1370 Sankyuk-Dong, Puk-Gu, DaeGu, 702-701, Korea. E-mail: wshan@knu.ac.kr.*
- *I.-Y. Song is with the College of Information Science and Technology, Drexel University, Philadelphia, PA 19104. E-mail: song@drexel.edu.*

Manuscript received 29 Mar. 2005; revised 22 July 2005; accepted 27 July 2005; published online 19 Dec. 2005.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0117-0305.

access sequence for the entire join. This is done by traversing the two R-trees in the breadth-first order. However, it has a drawback of incurring disk or memory overhead to store the entire page access sequence before performing join.

T-space index join algorithms include the *Transformation-Based Spatial Join* [28]. It is based on the Multi-Level Grid File (MLGF) [17], [29]. This algorithm achieves global optimization by utilizing special characteristics of the t-space. Due to this global optimization, it shows performance comparable to or better than that of the Depth-First Traversal R-tree Join. Besides, since the algorithm does not store the entire page access sequence, unlike the Breadth-First Traversal R-tree Join, it does not incur as much disk or memory overhead. Thus, it shows a performance comparable to or better than that of the Breadth-First Traversal R-tree Join. But, it has a drawback in that it cannot be applied to joining two o-space indexes or joining a t-space index with an o-space index.

Each class of index join algorithms has some advantages and disadvantages. O-space index join algorithms have an advantage in that they can be applied to widely used index structures such as the R-tree family in the o-space. The disadvantage is that, since the algorithms deal with the extents of the objects, it is relatively complex to optimize them. In particular, doing global optimization requires significant disk or memory overhead. In contrast, since t-space index join algorithms deal with only points but no extents, optimization is relatively straightforward and a global optimization is possible without significant overhead. However, the disadvantage is that they cannot be applied to widely used index structures such as the R-tree.

In this paper, we propose a novel mechanism for achieving the best of these two types of approaches. Specifically, we propose a technique that can apply a t-space index join algorithm to an o-space index. Our approach is based on the new notion of the *transform-space view* (*t-space view*), which is a virtual t-space index for an o-space index. The importance of the t-space view is that it allows us to dynamically "interpret" or "view" an existing o-space index as a t-space index with no space and negligible time overhead and without modifying the structure of the o-space index or changing the representation of objects stored in it. Thus, efficient t-space index join algorithms can be applied to o-space indexes through the t-space view. We also propose a t-space index join algorithm based on the t-space view (we call it the *t-space view join algorithm*). This t-space view method is applicable to tree-structured o-space indexes, where regions and objects stored in the index are represented as minimum bounding rectangles (MBRs).[1]

In the t-space index, join algorithms, including the Transformation-Based Spatial Join [28]—the page access sequence for which various space filling curves could be used—makes a significant impact on the performance. In [18], the authors have presented a formal analysis of the effect of the space filling curves on the performance of the Transformation-Based Spatial Join algorithm and proposed a new space filling curve called the *adaptive row major order* (*ARM order*). The ARM order significantly enhances the performance of the algorithm by controlling the order of accessing pages for a given buffer size adaptively. Here, we adapt this analysis and the ARM order to the t-space view join algorithm and show that they are also feasible for the t-space view join algorithm. Experimental results show that the analysis is highly accurate and the ARM order significantly improves the performance of the proposed algorithm.

Through extensive experiments, we show that the t-space view join algorithm always outperforms existing join algorithms that use R-trees in the o-space for all the data sets tested. As the measures of performance, we use the one-pass buffer size, the number of disk accesses, and the wall clock time. Here, the *one-pass buffer size* is the minimum buffer size necessary for processing joins while accessing the pages to be joined only once. Smaller one-pass buffer sizes allow more joins to be done with optimal disk accesses given limited buffer.

The contributions of our paper are summarized as follows: First, we propose the new notion of the t-space view that allows us to dynamically interpret an o-space index as a t-space index without modifying its original structure and with no space and negligible time overhead. This allows us to apply conventional t-space index join algorithms to widely used R-trees. Second, we propose a new join algorithm based on the t-space view. Third, we show that the formal analysis and the ARM order for the Transformation-Based Spatial Join algorithm [18] are also usable for the proposed algorithm. Fourth, through experiments, we show that applying a t-space index join algorithm to R-trees through the t-space view is faster than applying o-space join algorithms to R-trees in the o-space for all the data sets tested, thus rendering it a lower-bound algorithm.

The rest of this paper is organized as follows: Section 2 describes related work covering o-space and t-space index join algorithms. Section 3 defines the concept of the t-space view and Section 4 presents our t-space view join algorithm. Section 5 explains a formal analysis of the effect of the space filling curve on the one-pass buffer size and Section 6 introduces the ARM order. Section 7 presents the experimental results for performance evaluation of the algorithm. In Section 8, we summarize and conclude the paper.

2 Related Work

In this section, we describe representative join algorithms that use indexes. Section 2.1 describes o-space index join algorithms based on R-trees. Section 2.2 explains t-space index join algorithms.

2.1 O-Space Index Join Algorithms

We consider two R-tree based join algorithms[2] depending on the traversal method used: the *Depth-First Traversal R-tree Join* algorithm proposed by Brinkhoff et al. [4] and the *Breadth-First Traversal R-tree Join* algorithm proposed by Huang and Jing [12]. Here, we briefly explain the structure of the R-tree [9] and define some notation. The R-tree is a height balanced tree structure storing multidimensional

1. Examples are R-tree [9], R*-tree [2], X-tree [3], and SKD-Tree [22]. Examples to which the t-space view method cannot be applied are P-Tree [14] and Cell-Tree [7].

2. To the extent of the authors' knowledge, these are the only R-tree-based spatial join algorithms reported in the literature. Other algorithms are their variants or do not require indexes on both files.

rectangles. A nonleaf page of an R-tree contains entries of the form $< mbr, ref >$, where *mbr* is the minimum bounding rectangle (MBR) containing all MBRs of the entries in a child page and *ref* is the pointer to the child page. A leaf page contains entries of the form $< mbr, oid >$, where *mbr* is the MBR of a spatial object and *oid* refers to the spatial object in the database.

The Depth-First Traversal R-tree Join finds all pairs of MBRs that overlap with each other from nonleaf pages of the two given R-trees by traversing depth first. The algorithm minimizes the search space by using the property that, when a pair of MBRs do not overlap with each other, their child pages will not overlap either and it thereby excludes them from further traversal.

It is necessary to control the page access sequence to prevent pages from being reread frequently from the disk. The Depth-First Traversal R-tree Join uses some heuristics, such as local plane-sweeping and local Z-ordering, to minimize disk accesses. Since these heuristics optimize the page access sequence of the child pages of only one pair of nonleaf pages that overlap with each other at a time, they tend to find a local optimum rather than finding the global optimum [12], [28].

If we generate and optimize the page access sequence of the child pages of as many pairs of overlapping nonleaf pages as possible simultaneously, then a near-global optimum can be found. Based on this observation, the Breadth-First Traversal R-tree Join generates the page access sequence of the entire set of pages by traversing breadth-first, optimizing the sequence, and performing the join. However, because this algorithm incurs a significant overhead for keeping track of the entire sequence, the algorithm has a lower performance than the Depth-First Traversal R-tree Join in small or large buffers depending on the techniques of managing the sequence. That is, when the page access sequence is stored in main memory (*Combo2 strategy*), it, in effect, causes a reduction in the buffer size. Since an insufficient buffer significantly drops the performance of join, the algorithm performs worse for small buffers. When the page access sequence is stored on the disk (*Combo1 strategy*), a performance drop caused by insufficient buffer does not occur. However, this strategy always causes extra disk accesses to store and access the entire page access sequence to the disk; thus, the algorithm performs worse for large buffers.

2.2 T-Space Index Join Algorithms

T-space index join algorithms use t-space indexes that manage points. The points are transformed from the objects with extents in the o-space by using a transformation technique. A typical transformation technique used is corner transformation [24], [27]. *Corner transformation* transforms each object in the n-dimensional o-space into a single point in the $2n$-dimensional t-space by using $2n$ parameters. These points are subsequently indexed by a multidimensional point access method. The coordinates of a point in the $2n$-dimensional space are determined by the minimum and maximum values of the MBR on each of the n axes in the o-space. For example, a one-dimensional object whose minimum and maximum values on the x-axis are lx and rx, respectively, is transformed into the point $< lx, rx>$ in the two-dimensional t-space.

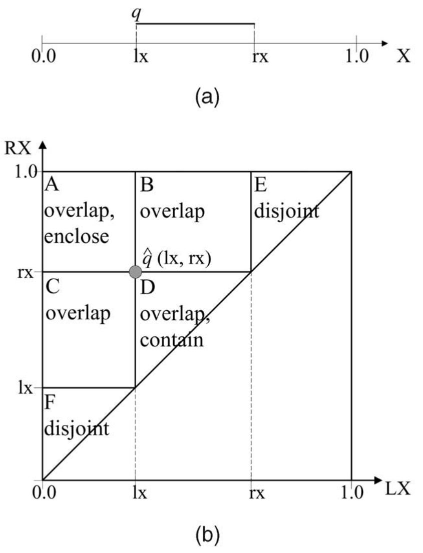


Fig. 1. T-space regions having different spatial relationships with the query region q in the o-space. Here, $\hat{q}$ is the transformed point of q. (a) Query region q in the o-space. (b) Six regions in the t-space according to spatial relationships to q.

When we use the t-space index to process an o-space query, we must transform the o-space query into a t-space query. Fig. 1 shows t-space regions used for transforming o-space queries into t-space queries [24], [28]. Fig. 1a shows a query region q in the o-space and Fig. 1b shows six t-space regions A to F having different spatial relationships with the query region q. In Fig. 1b, region A contains all the points that enclose q in the o-space; region D contains those that are contained by q; regions A, B, C, and D contain those that overlap with q; regions E and F contains those that do not overlap with q. Using these t-space regions, an o-space query that finds objects overlapping with q in the o-space is transformed into a t-space query that finds points in the union of t-space regions A, B, C, and D. Since finding points with no extents in a region is conceptually more straightforward than finding objects with extents that satisfy a spatial relationship with a region, processing queries in the t-space can be more concisely done than in the o-space [6], [28]. We can also process spatial join more concisely in the t-space.

Some transformation techniques [26] transform n-dimensional objects into one-dimensional points. These points are then indexed by a one-dimensional access method. These methods first partition the space into grid cells. Each grid cell is labeled with a unique number given by the Z-order or the Hilbert order. Objects are then indexed by a B^+-tree according to the label of the grid cell in which the objects are contained. It is known that these techniques are not favorable for spatial join since they have an obvious disadvantage in that, if the grid structures of the two indexes are not compatible, they cannot be joined unless the labels of one of them are recomputed [6].

The *Transformation-Based Spatial Join* [28] uses the *Multi-Level Grid File* (*MLGF*) [17], [29] as the index. This algorithm joins two MLGF indexes R and S that contain points that have been transformed by corner transformation. Here, we briefly explain the structure of the MLGF and define some

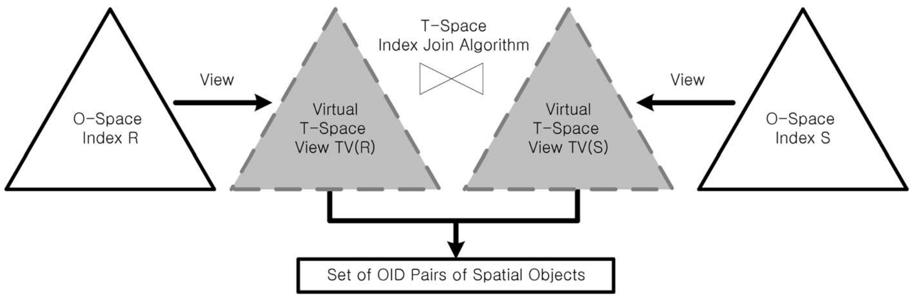


Fig. 2. The process of performing a join using t-space views.

notation. The MLGF is a height-balanced index tree that stores multidimensional point data. A nonleaf page in an MLGF contains entries of the form $< region, ref >$, where *ref* is the pointer to the child page and *region* contains all the regions represented by the entries of the child page pointed by *ref*. A leaf page contains entries of the form $< point, oid >$, where *point* is the coordinate of the points in the t-space and *oid* is the identifier of the object stored in the database.

The Transformation-Based Spatial Join finds a pair of points whose original objects overlap by comparing the points in all the leaf-pages of indexes R and S. Here, the two t-space regions of S that are to be joined with two adjacent t-space regions of R have considerably overlapping areas. By taking advantage of this characteristic, together with the LRU buffer policy, the algorithm globally optimizes the number of disk accesses. On the other hand, the drawback of the algorithm is that it cannot be directly applied to the data stored in and managed by an o-space index.

3 T-Space View

A *t-space view* is a virtual t-space index mapped from an o-space one by the transformation technique. Using the t-space view, an o-space index can be interpreted or viewed as a t-space index without modifying the structure of the o-space index. Thus, the t-space join algorithms can be directly applied to the t-space views of the o-space index. Fig. 2 illustrates the process of performing a join using t-space views. In the figure, two o-space indexes R and S are interpreted as virtual t-space views TV(R) and TV(S), respectively, and a t-space index join algorithm is applied to TV(R) and TV(S).

The t-space view is interpreted by mapping the structure of the o-space index into that of the t-space index. We first informally describe the mapping using the R-tree for the o-space index and the MLGF for the t-space index. As shown in Section 2, an entry in a nonleaf page of the R-tree has the form $< mbr, ref >$ and the one in the MLGF has $< region, ref >$. Here, *mbr* represents an n-dimensional o-space region and *region* represents a $2n$-dimensional t-space region. Thus, we map an *mbr* of the R-tree into a *region* of the MLGF. Similarly, an entry in a leaf page of the R-tree has the form $< mbr, oid >$ and the one in the MLGF has $< point, oid >$. Here, *mbr* represents an n-dimensional o-space object and *point* represents a $2n$-dimensional t-space point. Thus, in this case, we map an *mbr* of the R-tree into a *point* of the MLGF. Using these mappings, we can interpret an R-tree as an MLGF, without physically materializing a new index.

We now formally define the mapping of the $2n$-dimensional t-space view for an n-dimensional o-space index. We use corner transformation for the mapping. Here, we assume that the o-space index is tree-structured and regions and objects stored in the index are represented as MBRs.

Definition 1. *Suppose objects and regions in the n-dimensional o-space are represented in the form*

$$[l_1, r_1] \times [l_2, r_2] \times \cdots \times [l_n, r_n],$$

where l_i is the minimum value and r_i is the maximum value on the ith axis. The $2n$-dimensional t-space view for an n-dimensional o-space index is interpreted as follows:

a. ***Mapping of o-space objects**. An object $[l_1, r_1] \times [l_2, r_2] \times \cdots \times [l_n, r_n]$ in the n-dimensional o-space is mapped into the point $< l_1, r_1, l_2, r_2, \cdots, l_n, r_n >$ in the $2n$-dimensional t-space.*
b. ***Mapping of o-space regions**. A region $[l_1, r_1] \times [l_2, r_2] \times \cdots \times [l_n, r_n]$ in the n-dimensional o-space is mapped into the region $[l_1, r_1] \times [l_1, r_1] \times [l_2, r_2] \times [l_2, r_2] \times \cdots \times [l_n, r_n] \times [l_n, r_n]$ in the $2n$-dimensional t-space.*

The mapping of o-space objects in Definition 1 follows the definition of corner transformation. The mapping of o-space regions is an extension of the corner transformation modified in such a way that the containment relationships between objects and regions in the o-space are preserved in the t-space. That is, the mapping guarantees that the t-space region corresponding to an o-space region is the region that contains all the t-space points corresponding to the objects contained in the o-space region. We need the preservation of the containment relationship to maintain a proper search structure in the t-space view. Suppose a one-dimensional o-space object $[lx', rx']$ contained in the one-dimensional o-space region $[lx, rx]$ (i.e., $lx \leq lx' \leq rx' \leq rx$) is mapped into the two-dimensional t-space point P $< lx', rx' >$. Then, P exists in the t-space region $[lx, rx] \times [lx, rx]$ by the relationship $lx \leq lx' \leq rx$ and $lx \leq rx' \leq rx$. Therefore, the relationships between the objects and regions in the o-space are preserved in the t-space as well after mapping.

Example 1. Fig. 3 illustrates an example of a one-dimensional R-tree and its two-dimensional t-space view. Fig. 3a shows a one-dimensional R-tree in the o-space. Fig. 3b shows the two-dimensional t-space

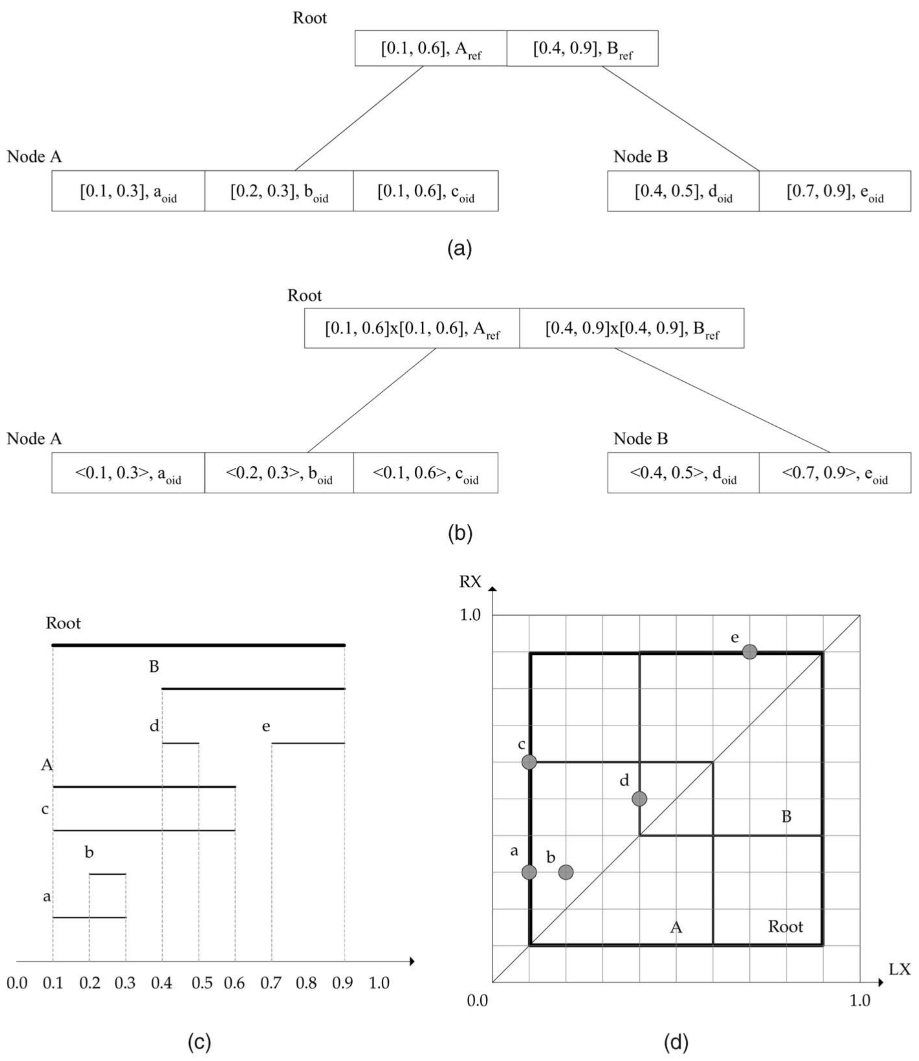


Fig. 3. An example of the interpretation of the t-space view for an o-space index. (a) One-dimensional R-tree. (b) Two-dimensional t-space view corresponding to the R-tree in (a). (c) Representation of objects and regions of the R-tree in (a) in the one-dimensional o-space. (d) Representation of points and regions in the two-dimensional t-space view.

view mapped from the R-tree in Fig. 3a. Fig. 3c shows the representation of o-space objects and regions of Fig. 3a in the one-dimensional o-space. Fig. 3d shows the representation of the corresponding point objects and regions in the two-dimensional t-space. The R-tree in Fig. 3a consists of two leaf pages, A and B, and one nonleaf page, Root. Leaf pages store o-space objects a, b, c, d, and e. The mapping is done as follows: By Definition 1a, the o-space object a, represented as $[0.1, 0.3]$ in Fig. 3c, is interpreted as the t-space point $< 0.1, 0.3>$ in Fig. 3d. Other o-space objects b, c, d, and e can be interpreted similarly. By Definition 1b, the o-space region A, represented as $[0.1, 0.6]$ in Fig. 3c, is interpreted as the t-space region $[0.1, 0.6] \times [0.1, 0.6]$ in Fig. 3d. The region B, represented as $[0.4, 0.9]$ in Fig. 3c, is similarly interpreted as the t-space region $[0.4, 0.9] \times [0.4, 0.9]$ in Fig. 3d.

Since the t-space view is defined in the t-space, all the properties of the t-space are applied to the t-space view. Thus, the relationships for the t-space in Fig. 1b are also effective for the t-space view. In addition, the t-space view has the following properties: The first property is that the mapping from an o-space index to a t-space view has no space and negligible time overhead since each entry of an o-space index is mapped into an entry of a t-space view through mere "interpretation" or "viewing." Thus, we can apply efficient t-space join algorithms to o-space indexes through the t-space view without additional cost. The second property is that the t-space view does not generate horizontally adjacent regions of pages because regions in a t-space view are always squares and on the diagonal from Definition 1. We take advantage of this property in the t-space view join algorithm that we present in Fig. 9. Using this property, we omit the process of grouping horizontally adjacent pages needed to perform spatial join in the t-space. Details are explained in Section 4.

4 T-SPACE VIEW JOIN ALGORITHM

The *t-space view join algorithm* spatially joins two t-space views TV(R) and TV(S) of o-space indexes R and S. For ease of explanation, we first explain the algorithm in the

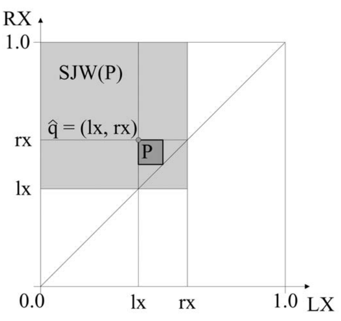


Fig. 4. A spatial join window.

two-dimensional t-space defined from the one-dimensional o-space. We then extend it to handle the $2n$-dimensional t-space. We assume the LRU buffer replacement policy commonly used in DBMSs.

We use the notion of the *spatial join window* [28]. We first describe its properties and, then, we present the t-space view join algorithm that takes advantage of these properties.

Definition 2 [28]. *Let t-spaces of indexes R and S to be joined be TS(R) and TS(S). The* Spatial Join Window SJW(P) *for a rectangular region P in TS(R) is the minimum region in TS(S) where all the objects intersecting with the objects in P can reside.*

The region of SJW(P) in TS(S) is obtained as follows: To intersect with some objects in P, objects in TS(S) must intersect with the object $\hat{q} = < lx, rx >$ that corresponds to the upper-left corner point of P since $\hat{q}$ is the largest o-space object containing any o-space object residing in P. According to Lemma 1, SJW(P) is derived as a region $[0, rx] \times [lx, 1]$.

Lemma 1 [28]. *The minimum region in TS(S) where all the t-space points whose o-space objects intersect with the o-space object $\hat{q} = < lx, rx >$ in two-dimensional TS(R) can reside is $[0, rx] \times [lx, 1]$.*

Proof. See Appendix A, which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm. □

Example 2. Fig. 4 shows SJW(P) in TS(S) for a rectangular region P in TS(R). In Fig. 4, P is shown as a dark-gray rectangle and SJW(P) is shown as a light-gray rectangle.

The two SJW's in TS(S) for two adjacent regions in TS(R) significantly overlap due to two properties. This overlap allows us to minimize the number of disk accesses by retaining the pages of the SJW to be reread in the buffer.

The first property is that SJWs for two horizontally adjacent regions have the containment relationship. In Fig. 5a, illustrating two adjacent regions, let the one far from the diagonal line be P_1 and the one near the diagonal line be P_2. Then, as shown in Fig. 5b, the relationship $SJW(P_1) \supset SJW(P_2)$ holds. If we join P_2 and $SJW(P_2)$ after we join P_1 and $SJW(P_1)$, $SJW(P_2)$ has already been read into the buffer and, thus, extra disk access can be avoided. Therefore, we use the notion of the *horizontal strip* enclosing horizontally adjacent regions and those regions in a horizontal strip are joined consecutively.

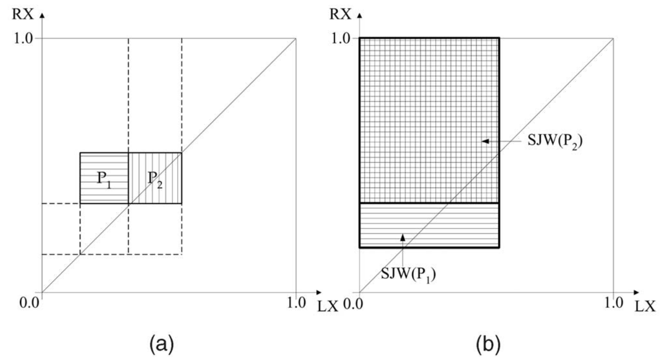


Fig. 5. Spatial join windows for horizontally adjacent regions. (a) Horizontally adjacent regions P_1 and P_2. (b) $SJW(P_1)$ and $SJW(P_2)$.

The second property is that, between two vertically adjacent horizontal strips $HSTR_1$ and $HSTR_2$, $SJW(HSTR_1)$ and $SJW(HSTR_2)$ significantly overlap. In Fig. 6, if we join $HSTR_2$ and $SJW(HSTR_2)$ after we join $HSTR_1$ and $SJW(HSTR_1)$, the former join can be processed with only incremental disk accesses because most of $SJW(HSTR_2)$ has already been read into the buffer. Therefore, we perform the join by selecting vertically adjacent horizontal strips consecutively.

Using these two properties, we define the t-space view join algorithm as in Fig. 7. The algorithm takes two t-space views TV(R) and TV(S) and a space filling curve SFC as the input. Since a t-space view is a virtual structure defined in the t-space, all the properties of the t-space, including the SJW and its two properties, are also preserved in the t-space view. In the explanation of the algorithm, we assume that TV(R) and TV(S) have the same properties of TS(R) and TS(S). The *space filling curve* [15] is a way of linearly ordering regions in a space. Here, the curve is used to select horizontal strips in a sequence. In a two-dimensional t-space, the SFC is the vertical axis RX. The case of $2n$-dimensional t-space will be discussed at the end of this section.

In Fig. 7, the t-space view join algorithm consists of two loops. The first loop in Line 1 selects the region *r_region* of a leaf page in TV(R) using the function *Next_Leaf_Region* in Fig. 8. The function *Next_Leaf_Region* returns the region corresponding to each leaf page in TV(R) following the order of SFC. The function discards the regions in TV(R)

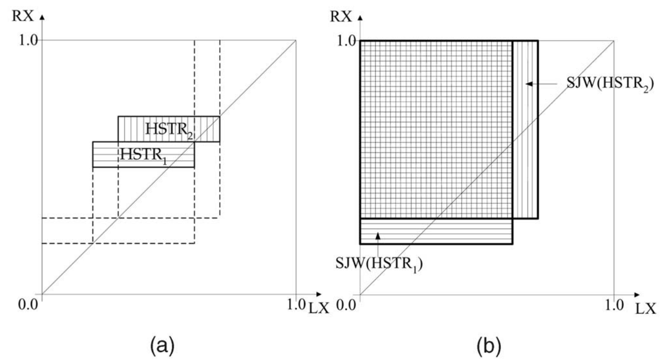


Fig. 6. Spatial join windows for vertically adjacent horizontal strips. (a) Vertically adjacent horizontal strips $HSTR_1$ and $HSTR_2$. (b) $SJW(HSTR_1)$ and $SJW(HSTR_2)$.

```
Algorithm Transform_Space_View_Join(TV(R), TV(S), SFC)
1  for each region r_region returned from Next_Leaf_Region (TV(R), TV(S), SFC) do
2      let r_page be the page corresponding to r_region
3      for each page s_page from the set of pages in SJW(r_region) in TV(S) do
4          Join_Pages(r_page, s_page)
5      end
6  end
```

Fig. 7. The t-space view join algorithm.

that are not to be joined with any region in TV(S) as soon as possible. By doing this, it prevents the algorithm from reading unnecessary pages from TV(R). For SFC, the function can use any space filling curve defined recursively. In the first loop, we use each region as a horizontal strip because 1) there are no horizontally adjacent pages in a t-space view according to the properties identified from Definition 1 and 2) objects are not duplicated in the intersecting regions according to the properties of o-space indexes like the R-tree.[3] Line 2 sets *r_page* to the page corresponding to *r_region*. The second loop in Line 3 selects each leaf page *s_page* from the set of pages in SJW(*r_region*) in TV(S). Line 4 joins the two selected leaf pages, *r_page* and *s_page*, using the function *Join_Pages*.

Fig. 8 shows the function *Next_Leaf_Region*. It takes two t-space views TV(R) and TV(S) and a space filling curve SFC as the input and returns the region corresponding to each leaf page in TV(R) following the order of SFC. By exploiting the hierarchical structure of the t-space view, the function can select regions of the leaf pages in a sorted order according to SFC with minimal overhead. In the hierarchical structure of the t-space view, the region of the parent page always comes before the region of a child page. That is, the region of the page will not be selected by SFC until the region of the parent page is selected. Using this property, we can significantly reduce the number of the regions to be sorted at a time. Since the regions to be sorted are dynamically changing as the selection proceeds, we use a heap[4] to maintain the SFC-sorted order and *heap sort* [16] to incrementally add and subtract regions maintaining the sorted order. We call this technique *hierarchical region-based sorting* and elaborate on this in Fig. 8. The concept of this technique has been derived from a similar technique used by Hjaltason and Samet [11] to find k-nearest neighbors incrementally using R-trees. Here, pages are selected incrementally in the order of the distance between the query point and the region of each page.

In Fig. 8, the function *Next_Leaf_Region* initializes *heap* with the region of the root page in TV(R) (Lines 1, 2). For each subsequent *Next_Leaf_Region* call, the function subtracts a region *r_region* (Lines 4, 5) and checks whether any page exists in SJW(*r_region*) in TV(S). If no page exists in SJW(*r_region*), we simply discard the *r_region* (Line 7). By doing this, we can discard any *r_region*s that are not going to participate in the join, thus avoiding unnecessary page accesses. If the *r_region* is a leaf region, we simply return the page for *r_region* (Lines 8, 9). If it is not a leaf region, we add all the child regions of *r_region* (Lines 10, 11). The size of *heap* is very small since each region in the heap does not expand into regions of the child pages until it is selected by SFC. In our extensive experiments, the size of the heap has been less than 4 Kbytes in all the cases.

We have so far explained the two-dimensional t-space view join based on the two-dimensional spatial join windows, horizontal strips, and space filling curves. We now extend them to $2n$-dimensional cases.

We define the *spatial join window* in the $2n$-dimensional space as the Cartesian product of n number of two-dimensional spatial join windows. That is, for a hyper-rectangular region P in the $2n$-dimensional TS(R), let $\hat{q}$ be $< ld_1, rd_1, ld_2, rd_2, \ldots, ld_n, rd_n >$, where ld_i is the minimum value on the $(2i-1)$th axis and rd_i is the maximum value on the $2i$th axis. Then, the $2n$-dimensional SJW(P) is the region $[0, rd_1] \times [ld_1, 1] \times [0, rd_2] \times [ld_2, 1] \times ... \times [0, rd_n] \times [ld_n, 1]$.

We define the $2n$-dimensional *horizontal strip* as the Cartesian product of n number of two-dimensional horizontal strips. Then, the *space filling curve* for ordering the horizontal strips is defined in the n-dimensional subspace consisting of the Cartesian product of n number of *vertical*

```
heap: a global variable where elements are maintained to be sorted by SFC
Function Next_Leaf_Region (TV(R), TV(S), SFC)
1   if heap has not been initialized then
2       add the region of the root page of TV(R) into heap
3   end
4   while heap is not empty do
5       let r_region be a region subtracted from heap
6       let r_page be the page corresponding to r_region
7       if the set of pages in SJW(r_region) in TV(S) is not empty then
8           if r_page is a leaf page then
9               return r_region
10          else /* if r_page is a non-leaf page then */
11              add all the regions of the children in r_page into heap
12          end
13      end
14  end
15  return null
```

Fig. 8. The Next_Leaf_Region function using hierarchical region-based sorting.

3. In the case of joining o-space indexes where objects are duplicated, duplicated join pairs can occur in the result. Thus, we need to eliminate them after the join is completed. However, we note that this step is also needed even if we use an o-space join algorithm.

4. The heap is implemented by using a complete binary tree. It has $O(\log_2 n)$ time complexity for adding and subtracting regions.

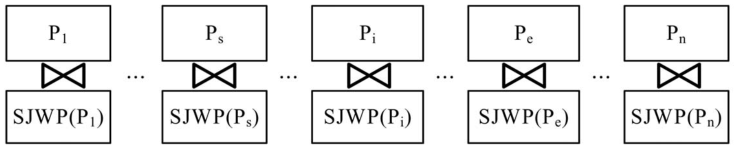


Fig. 9. The process of joining pages in the t-space join algorithm.

axes of two-dimensional spaces. This is an extension of the space filling curve for the two-dimensional case, which is simply the (one-dimensional) vertical axis. We can use any space filling curve that is able to order the strips in the n-dimensional space. Examples of such space filling curves are the row major order, Z-order [23], and Hilbert order [10]. In Sections 5 and 6, through a formal analysis, we propose a new space filling curve that minimizes the one-pass buffer size and the number of disk accesses.

The t-space view join achieves global optimization by considering the entire page access sequences of two t-space views, TV(R) and TV(S). Here, we take advantage of the properties of the spatial join window and the LRU buffer replacement policy. By exploiting the hierarchical structure of the TV(R) and the heap whose elements are ordered by SFC, the ordering of the page access is done with little overhead, resulting in global optimization in the t-space view join. Therefore, it always outperforms the Breadth-First Traversal R-tree Join that needs significant overhead for global optimization. It also outperforms the Depth-First Traversal R-tree Join that only does local optimization.

We also have solved the problems of the Transformation-Based Spatial Join algorithm—an earlier t-space index join algorithm. The t-space view join has the following advantages: First, it can be applied to o-space indexes, while the Transform-Based Spatial Join cannot. Second, the t-space view join does not access unnecessary leaf pages of the outer index, while the Transform-Based Spatial Join does.

5 Space Filling Curves and One-Pass Buffer Size[5]

The order of accessing disk pages makes a significant impact on the performance of joins [4], [12], [18], [28] in terms of the one-pass buffer size and the number of disk accesses. Here, the *one-pass buffer size* is the minimum buffer size necessary for processing joins while accessing the pages to be joined only once. In this section, we adapt the analysis [18] of the effect of the space filling curve on the one-pass buffer size, when used with the Transformation-Based Spatial Join algorithm [28], to the t-space view join algorithm. We first explain a method that calculates the one-pass buffer size for a given space filling curve and calculate one-pass buffer sizes for typical space filling curves by using this method. We then conclude which space filling curve is preferable for the t-space view join.

To make analysis easy, we make a few assumptions for data distributions, which we call *uniform data and page distribution assumptions*. We assume the distribution of data is uniform in terms of the positions and sizes of objects in the two-dimensional o-space, that the sizes of the regions corresponding to the leaf pages of the o-space index are equal, and that their center positions have a uniform distribution. Finally, we use the LRU buffer replacement policy commonly used in DBMSs.

Fig. 9 shows the process of joining the pages of TV(R) and TV(S) in the t-space view join algorithm. Here, the pages in TV(R) are read consecutively following the order of SFC. $\mathrm{P_i}$ represents the ith page read from TV(R) and $\mathrm{SJWP(P_i)}$ represents the set of pages corresponding to the region $\mathrm{SJW(P_i)}$ in TV(S).

Among the pairs of pages whose SJWPs overlap in Fig. 9, let $\mathrm{P_s}$ and $\mathrm{P_e}$ be the pair whose difference between the two values s and e is the maximum. Then, to guarantee one-pass, the t-space view join algorithm needs an LRU buffer whose size is equal to the total number of pages read in when doing the join starting from $\mathrm{P_s}$ to $\mathrm{P_e}$. This constitutes the one-pass buffer. If the buffer is smaller than the one-pass buffer, due to LRU buffer replacement, not all the pages that have been read in when the join is processed for $\mathrm{P_s}$ are guaranteed to remain in the buffer until the join is processed for $\mathrm{P_e}$. Equation (1) calculates the average one-pass buffer size:

$$(1 + |\mathrm{SJWP(P_s)}|) + \sum_{i=1}^{\mathrm{MAXLDIST}-1} (1 + AVG(|\Delta\mathrm{SJWP}|)). \quad (1)$$

In (1), $(1 + |\mathrm{SJWP(P_s)}|)$ represents the number of pages read in for $\mathrm{P_s}$ and $\mathrm{SJWP(P_s)}$. $(1 + AVG(|\Delta\mathrm{SJWP}|))$ represents the average number of pages read in for each page P_i processed after P_s and the incremental $\mathrm{SJWP}(\mathrm{P}_i)$ that is newly read in. MAXLDIST is the maximum linear distance between any two pages whose SJWPs overlap. In Fig. 9, MAXLDIST is $|e - s + 1|$.

According to (1), MAXLDIST is the most dominant factor of the one-pass buffer size. To obtain MAXLDIST, we first define in Fig. 10 a scheme for identifying pages in the four-dimensional t-space by two-dimensional coordinate values. Fig. 10a and Fig. 10b show the LX-RX and LY-RY planes that are t-spaces of the X and Y axes, respectively. Here, a rectangle represents the region corresponding to a disk page. In Fig. 10c, we project the four-dimensional space onto the RX-RY plane (the Cartesian product of two vertical axes as described in Section 4), where the regions are to be selected according to the space filling curve. For simplicity, Fig. 10c represents these regions projected onto the RX-RY plane in an array-like fashion in order to identify them

5. We prominently cite our earlier work in the following: The analysis in Section 5 and the ARM order in Section 6 have appeared in an earlier paper [18] in the context of the Transformation-Based Spatial Join algorithm [28]. The main contribution of the earlier paper is that it presents a formal analysis of the effect of the space filling curves on the performance of the Transformation-Based Spatial Join algorithm and proposes a new space filling curve called the ARM order. The current paper applies them to the t-space view join algorithm and verifies whether they work properly in the context of the t-space view join algorithm. Experimental results in Section 7.1 show that the analysis is highly accurate and the ARM order significantly improves the performance of the t-space view join algorithm.

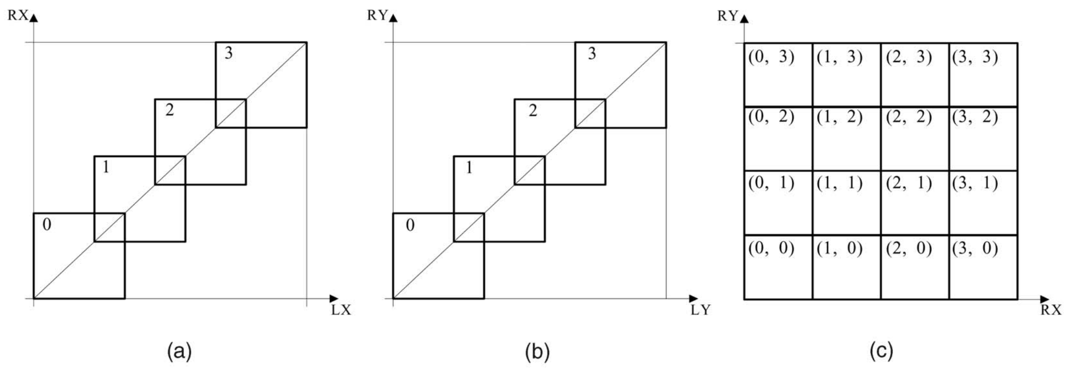


Fig. 10. A four-dimensional t-space. (a) LX-RX plane. (b) LY-RY plane. (c) RX-RY plane.

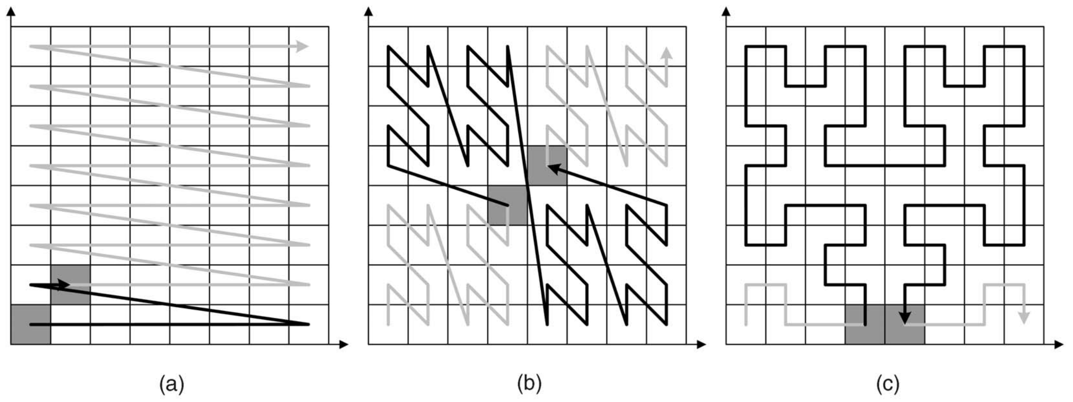


Fig. 11. MAXLDISTs for typical space filling curves. (a) Row major order. (b) Z-order. (c) Hilbert order.

clearly. Next, we assign the two-dimensional coordinate value (u, v) to the region in the RX-RY plane whose coordinate in the RX axis is u and in the RY axis is v. Since each region corresponds to a page, a page is identified by the coordinate of the region.

A space filling curve C represents the order using a *linearization function* L_C. The linearization function $\mathrm{L_C}$ is a function that maps the page whose coordinate is (u, v) to the order number i. This is denoted as $L_C(u, v) = i$. When the coordinates of two pages $\mathrm{P_1}$ and $\mathrm{P_2}$ are (u_1, v_1) and (u_2, v_2), the distance between the two pages for a space filling curve C is calculated as $|\mathrm{L_C}(u_2, v_2) - \mathrm{L_C}(u_1, v_1) + 1|$.

To obtain MAXLDIST, we now define the notion of page adjacency.

Definition 3. *The* page adjacency *of an index is defined as the minimum constant* c *satisfying* $|u_1 - u_2| \leq c$ *and* $|v_1 - v_2| \leq c$ *for all pairs of pages* (u_1, v_1) *and* (u_2, v_2) *whose SJWPs overlap, i.e., if* $|u_1 - u_2| > c$ *or* $|v_1 - v_2| > c$*, then the SJWPs of the pages* (u_1, v_1) *and* (u_2, v_2) *do not overlap.*

Using the page adjacency defined above, we calculate MAXLDIST's for typical space filling curves: the row major order, Z-order, and Hilbert order. For other space filling curves, MAXLDIST can be calculated similarly. We first explain the cases when the page adjacency is 1. We then explain them when the page adjacency is greater than 1.

Lemma 2. *Under uniform data and page distribution assumptions, for the row major order,* $\mathrm{MAXLDIST_{RM}} = \lceil\sqrt{n}\rceil + 2$ *when the page adjacency = 1 and* $n \geq 4$.

Proof. See Appendix B, which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm. □

Lemma 3. *Under uniform data and page distribution assumptions, for the Z-order,* $\mathrm{MAXLDIST_Z} = \frac{n}{2} + 2$ *when the page adjacency = 1 and the number of pages* $n \geq 4$.

Proof. See Appendix C, which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm. □

Lemma 4. *Under uniform data and page distribution assumptions, for the Hilbert order,* $\mathrm{MAXLDIST_{Hilbert}} = \frac{10n+8}{12}$ *when the page adjacency = 1 and the number of pages* $n \geq 4$.

Proof. See Appendix D, which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm. □

Example 3. Fig. 11 shows MAXLDISTs for the row major order, Z-order, and Hilbert order when the number of pages $n = 64$ and the page adjacency = 1. Two pages with the linear distance between them being MAXLDIST are represented as gray rectangles and the path of the space filling curve between the two pages as black curves. In each figure, the length of the curve is MAXLDIST. For the row major order, MAXLDIST is $\lceil\sqrt{64}\rceil + 2 = 10$, for Z-order, $\frac{64}{2} + 2 = 34$, and for Hilbert order, $\frac{10 \cdot 64 + 8}{12} = 54$.

Now, we consider the case where the page adjacency $c > 1$. For the row major order, $\mathrm{MAXLDIST_{RM}} = c \times \lceil\sqrt{n}\rceil + c + 1$ from (4) in Appendix B, which can be found on the

Computer Society Digital Library at http://computer.org/tkde/archives.htm, since $max(|u_1 - u_2|)$ and $max(|v_1 - v_2|)$ are c. For the Z-order and Hilbert order, their MAXLDISTs are independent of the page adjacency c within the error of $4^{\lfloor \log_2 c \rfloor}$ from Lemma 5.

Lemma 5. *For the space filling curves Z-order and Hilbert order,*[6] *the value of MAXLDIST is the same regardless of the page adjacency c within the error of $4^{\lfloor \log_2 c \rfloor}$.*

Proof. See Appendix E, which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm. □

In Lemmas 2, 3, and 4, we have formally derived MAXLDISTs of typical space filling curves. We observe that MAXLDIST of the row major order is $O(\sqrt{n})$ while the others are $O(n)$. Therefore, the row major order has the smallest one-pass buffer size when $n \gg 1$. Hence, to minimize the one-pass buffer size, using the row major order is preferable. However, when the given buffer is smaller than the one-pass buffer, the row major order abruptly incurs a lot of disk accesses. In the next section, we analyze this problem and present a solution. As the solution, we introduce a new space filling curve, called the *adaptive row major order*, that has been proposed in [18].

6 Adaptive Row Major Order

As shown in Section 5, the row major order is the space filling curve that has the smallest one-pass buffer size. But, it has the problem of reading some pages more than once when the buffer is smaller than the one-pass buffer. We analyze this problem as follows.

In the row major order, when the given buffer is even slightly smaller than the one-pass buffer, we may have to reread SJWPs of all the pages that have been processed. When page adjacency = c, if we let (u, v) be the ith page, then $(u + c, v + c)$ is identified as the $(i + \mathrm{MAXLDIST_{RM}})$th page. Since the linear distance between them is $\mathrm{MAXLDIST_{RM}}$ and the buffer is smaller than the one-pass buffer determined by $\mathrm{MAXLDIST_{RM}}$ from (1), the SJWP of (u, v) must have been swapped out by the time the page $(u + c, v + c)$ is processed. However, since the SJWP of $(u + c, v + c)$ overlaps that of (u, v) (note that page adjacency = c), we need to reread the SJWP of the page (u, v) from disk. In this way, for each i, we may have to reread the SJWP of the ith page (u, v) from the disk when joining the $(i + \mathrm{MAXLDIST_{RM}})$th page $(u + c, v + c)$ if the LRU buffer given is smaller than the one-pass buffer determined by $\mathrm{MAXLDIST_{RM}}$.

The linear distance between (u, v) and $(u + c, v + c)$ is proportional to $\lceil \sqrt{n} \rceil$, which is the horizontal width of the row major order. Thus, we can solve the problem by controlling the horizontal width. For this purpose, we use a new space filling curve, the *adaptive row major order* (*ARM order*), proposed in [18]. The formal definition is stated in Definition 4.

6. This lemma can be applied to any space filling curve defined recursively. The details are not discussed here since it is beyond the scope of this paper.

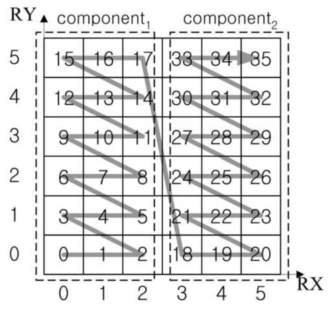


Fig. 12. Adaptive row major where $n = 36, k = 2$.

Definition 4. *The linearization function of the ARM order* $\mathrm{L_{ARM}}$ *is defined as (2):*

$$\mathrm{L_{ARM}}(u,v) = \begin{cases} v \times \frac{\lceil \sqrt{n} \rceil}{k} + u \text{ } if \text{ } u < \frac{\lceil \sqrt{n} \rceil}{k} \text{ } and \text{ } k > 0 \\ \cdots \\ v \times \frac{\lceil \sqrt{n} \rceil}{k} + (u - \frac{i\lceil \sqrt{n} \rceil}{k}) + \frac{i \cdot n}{k} \text{ } for \text{ } 1 \le i < k-1 \text{ } if \\ \frac{i\lceil \sqrt{n} \rceil}{k} \le u < \frac{(i+1)\lceil \sqrt{n} \rceil}{k} \text{ } and \text{ } k > 2 \\ \cdots \\ v \times \frac{\lceil \sqrt{n} \rceil}{k} + (u - \frac{(k-1)\lceil \sqrt{n} \rceil}{k}) + \frac{(k-1)n}{k} \text{ } if \\ u \ge \frac{(k-1)\lceil \sqrt{n} \rceil}{k} \text{ } and \text{ } k > 1. \end{cases} \quad (2)$$

Example 4. Equation (3) and Fig. 12 show the ARM order when $n = 36$ and $k = 2$. This ARM order consists of two components, each traversed in the row major order. In general, the ARM order consists of k number of components of width $\frac{\lceil \sqrt{n} \rceil}{k}$, each traversed in the row major order. The left part $(u < 3)$ has a width of 3 and is described by the linearization function $v \times 3 + u$. The right part $(u \ge 3)$ also has a width of 3 and is described by the linearization function $v \times 3 + (u - 3) + 18$.

$$\mathrm{L_{ARM}}(u,v) = \begin{cases} v \times 3 + u, & if \ u < 3 \\ v \times 3 + (u-3) + 18 & if \ u \ge 3. \end{cases} \quad (3)$$

In the ARM order, we determine k in such a way that, given a buffer of a specific size B and page adjacency c, the one-pass buffer size determined by MAXLDIST within one component ($\mathrm{MAXLDIST_{component}}$) is equal to or smaller than B. Here, $\mathrm{MAXLDIST_{component}} = c \times \frac{\sqrt{n}}{k} + c + 1$. From (1), we determine k as the minimum one satisfying $(1 + |\mathrm{SJWP(P_s)}|) + (1 + AVG(|\Delta\mathrm{SJWP}|)) \times (\mathrm{MAXLDIST_{component}} - 1) \le \mathrm{B}$.[7]

7 Experiments

In this section, we first compare the performances of the t-space view joins using different space filling curves and

7. The parameters, $|\mathrm{SJWP(P_s)}|$, $AVG(|\Delta\mathrm{SJWP}|)$, and c, can be measured or predicted by sampling nonleaf pages. To handle the worst case in the nonuniform distribution, we sample some nonleaf pages in the densest region identified from the index with a small overhead. Nevertheless, these nonleaf pages tend to remain in the buffer without (or with only a few for small buffers) extra disk accesses. The extra cost for measuring these parameters has been added in the experimental results in Section 7.

TABLE 1
Algorithms Compared

Name	Description
TSVJ-RM	T-Space View Join using the Row Major Order
TSVJ-Z	T-Space View Join using the Z-Order
TSVJ-Hilbert	T-Space View Join using the Hilbert Order
TSVJ-ARM	T-Space View Join using the ARM Order
DFRJ	Depth-First Traversal R-tree Join [4]
BFRJ-Combo1	Breadth-First Traversal R-tree Join: Combo1 [12]
BFRJ-Combo2	Breadth-First Traversal R-tree Join: Combo2 [12]

verify the analytic results in Sections 5 and 6. We then compare the performance of the t-space view join with those of o-space index join algorithms—the Depth-First Traversal R-tree Join and the Breadth-First Traversal R-tree Join (Combo1 and Combo2). Table 1 summarizes the algorithms compared.

We perform experiments using the three measures: the one-pass buffer size, the number of disk access, and the wall clock time. We use three distributions of data: uniform, exponential, and real distributions. The data and distributions are identical to those used in [28]. Data sets U1 and U2 have uniform distributions; data sets E1 and E2 exponential distributions. Each data set has 130,000 MBRs (3.5 MBytes). The center points of the MBRs in U1 and U2 are uniformly distributed in the o-space, and those in E1 and E2 are exponentially distributed with an average value of 1/4 along each axis in the o-space. The number of join results for U1 and U2 is 87,434 and that for E1 and E2 is 87,346. The real data set consists of Tiger1 and Tiger2, which are identical to those used by Brinkhoff et al. [4] and Huang and Jing [12]. Tiger1 contains 131,461 MBRs of streets in an area of California (3.8 MBytes); Tiger2 contains 128,971 MBRs of rivers and railway tracks (3.4 MBytes).[8] The number of join results of Tiger1 and Tiger2 is 86,094. All the data sets used for the experiments are in the two-dimensional o-space. For spatial join algorithms, performances in the two-dimensional o-space are most important since almost all spatial applications deal with two-dimensional data—for example, maps.

We conduct all the experiments on a Sun Ultra 60 workstation. To avoid the buffering effect of the file system and to guarantee actual disk I/Os, we use raw disks for indexes. Indexes used in the experiments are R*-trees, which are identical to those used by Brinkhoff et al. [4]. The page size of the R*-trees is 4 Kbytes. We use the LRU buffer replacement policy.

7.1 Performance Comparison among T-Space View Join Algorithms Using Different Space Filling Curves

We verify the analysis in Sections 5 and 6 on the one-pass buffer size of the t-space view join algorithms using different space filling curves for uniform data sets. We also compare the results for other data sets: exponential and real ones. Fig. 13 summarizes the one-pass buffer sizes. The results show that TSVJ-RM and TSVJ-ARM have the smallest one-pass buffer size as we have analytically predicted in Sections 5 and 6. Compared to TSVJ-Z, TSVJ-RM and TSVJ-ARM reduce the one-pass buffer size by up to 14.0 times[9] for different data sets. Compared to TSVJ-Hilbert, they reduce the one-pass buffer size by up to 21.3 times.

For uniform data sets, the results show that the ratio of three one-pass buffer sizes for TSVJ-RM, TSVJ-Z, and TSVJ-Hilbert is 71:885:1480 = 0.04:0.59:1.00. This ratio is very close to the ratio 69:973:1615 = 0.04:0.60:1.00 calculated from (1) and Lemmas 2, 3, and 4 when using the parameters $|\mathrm{SJWP}(\mathrm{P_s})| = 4$, $AVG(|\Delta\mathrm{SJWP}|) = 1.12$,[10] the page adjacency $c = 1$,[11] and the number of pages $n = 913$. These parameters have been measured by sampling pages of the t-space view. This result indicates that the analysis for uniform data sets is highly accurate. We also observe that the results are similar for exponential and real ones, indicating that the analysis based on uniform data sets is a reasonable approximation.

Fig. 14 shows the number of disk accesses the t-space view join algorithms incur as the buffer size is varied for the uniform, exponential, and real data sets. In the figure, the bottom horizontal line represents the optimal number of disk accesses obtained with the one-pass buffer.

In Fig. 14, we observe that TSVJ-ARM outperforms other TSVJ algorithms regardless of the buffer size. TSVJ-RM incurs the same number of disk accesses as TSVJ-ARM when the buffer is larger than the one-pass buffer. TSVJ-RM, however, has the problem of causing excessive disk accesses when the buffer is smaller than the one-pass buffer. This experimental result matches the analytic result in Sections 5 and 6. To summarize the performance improvement of TSVJ-ARM compared to the others, we distinguish two cases: those using a small buffer and those using a large buffer. A *small buffer* has a size of $1 \sim 2$ percent of the combined size of the two indexes joined. A *large buffer* has a size of $5 \sim 6$ percent of the same. We observe that TSVJ-ARM outperforms other TSVJ algorithms by up to 3.95 times[12] for the small buffer and by up to 1.31 times for the large buffer.

7.2 Performance Comparison of T-Space View Join Algorithm with O-Space Index Join Algorithms

In this section, we compare our TSVJ-ARM with DFRJ and BFRJ. We choose TSVJ-ARM because it has the best performance among TSVJ algorithms.

7.2.1 Comparison of TSVJ-ARM with DFRJ

Figs. 15, 16, and 17 summarize the results of experiments for TSVJ-ARM and DFRJ using the uniform, exponential, and real data sets. The results show that TSVJ-ARM always

8. Although this size of the real data set is relatively small, we choose it since it has been used in the literature for many other spatial join algorithms to evaluate performance. We also note that the t-space view join only needs small memory (buffer) whose size is $5 \sim 6$ percent of the size of indexes to be joined for guaranteeing optimal disk performance. Thus, using modern computer systems, we can join fairly large indexes optimally. For example, with 60 MBytes of buffer, we can optimally join the indexes of 1 GBytes.

9. $\frac{\text{one-pass buffer size (other algorithm)}}{\text{one-pass buffer size (TSVJ-ARM or TSVJ-RM)}}$.

10. Experiments show that $AVG(|\Delta\mathrm{SJWP}|)$ varies very little, having nearly the same value for different space filling curves.

11. We note that the one-pass buffer sizes for the space filling curves determined from (1) and Lemmas 3 and 4 are independent of the page adjacency c within the error of $4^{\lfloor \log_2 c \rfloor}$ from Lemma 5.

12. $\frac{\text{number of disk accesses (other algorithm)}}{\text{number of disk accesses (TSVJ-ARM)}}$.

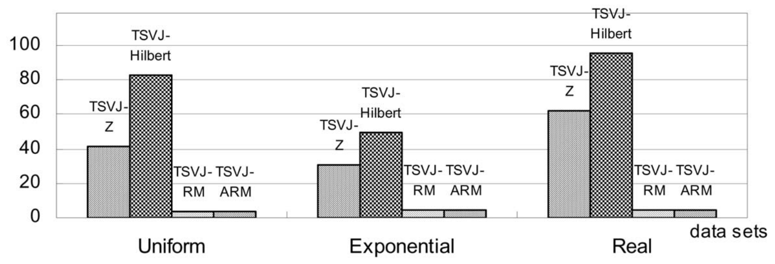


Fig. 13. One-pass buffer sizes of TSVJs when using different space filling curves.

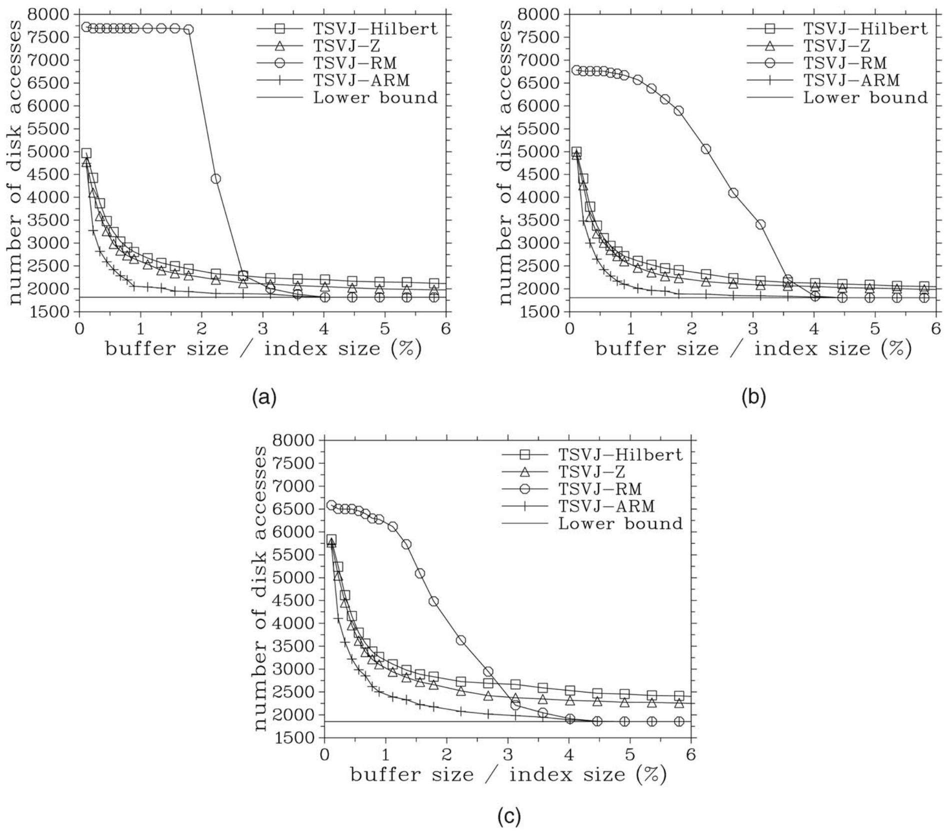


Fig. 14. The number of disk accesses of TSVJs with various space filling curves for uniform, exponential, and real data sets. (a) Uniform data set. (b) Exponential data set. (c) Real data set.

outperforms DFRJ in terms of the three measures for all the data sets tested.

Fig. 15 shows that, compared to DFRJ, TSVJ-ARM reduces the one-pass buffer size by up to 15.7 times. Fig. 16 compares the numbers of disk accesses as the buffer size varies. Fig. 16 shows that, with the small buffer, TSVJ-ARM reduces the number of disk accesses by up to 1.55 times and, with the large buffer, by up to 1.16 times. Fig. 17 shows the result of the wall clock time with curves similar to those in Fig. 16. The results using the uniform and exponential data sets are omitted since they are similar to those using the real data sets.

The reason why TSVJ-ARM has a better performance than DFRJ is that TSVJ-ARM formally controls the order of page accesses by employing global optimization that takes advantage of the characteristics of the spatial join windows in the t-space. On the other hand, DFRJ controls the order of page accesses by using heuristics such as local plane-sweeping and local Z-ordering, resulting in local optimization.

7.2.2 Comparison of TSVJ-ARM with BFRJ

BFRJ-Combo2 controls the LRU buffer by using the method of pin/unpin, which helps the buffering efficiency. The pin operation fixes the pages that are expected to be reread in the buffer and assures that they remain in the buffer until they are reread. The unpin operation undoes the pin

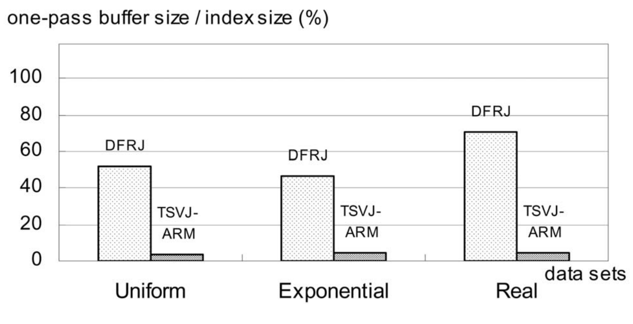
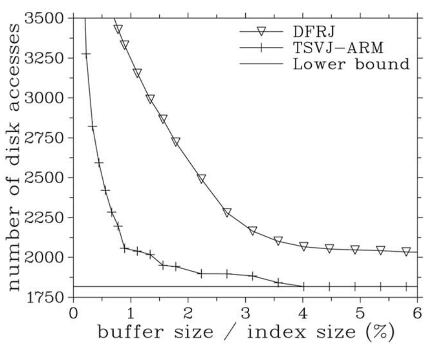
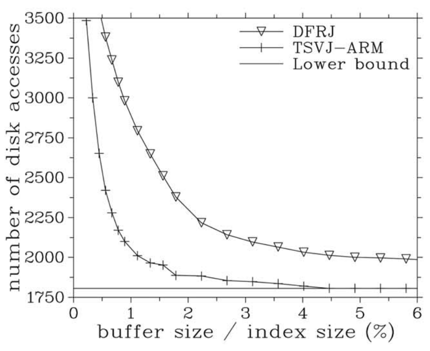


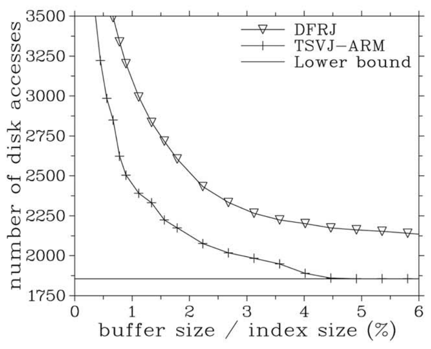
Fig. 15. One-pass buffer sizes of DFRJ and TSVJ-ARM.



(a)



(b)



(c)

Fig. 16. The number of disk accesses of DFRJ and TSVJ-ARM for uniform, exponential, and real data sets. (a) Uniform data set. (b) Exponential data set. (c) Real data set.

operation. This method can be applied to any algorithms that do *global optimization* since, in these algorithms, we can predict which pages are to be pinned or unpinned. This method can also be applied to TSVJ-ARM. But, since TSVJ-ARM carefully reads pages to assure that they are not unnecessarily reread from the disk, TSVJ-ARM with and without this method show almost the same performance. Therefore, we DO NOT apply this method to TSVJ-ARM. Fig. 18 compares the one-pass buffer sizes of TSVJ-ARM and BFRJ-Combo2 for the uniform, exponential, and real data sets. Since BFRJ-Combo1 uses external disk sort, it always reads disk pages more than once. Thus, we have not been able to obtain the one-pass buffer size of BFRJ-Combo1 and compare it with that of TSVJ-ARM. Fig. 18 shows that,

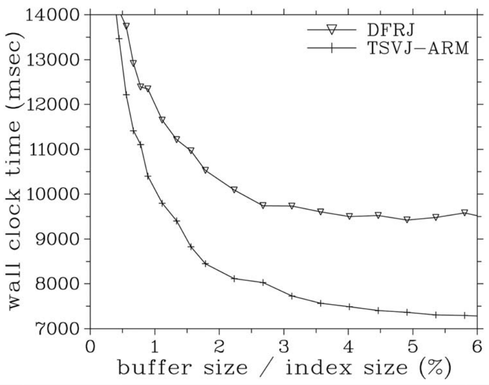


Fig. 17. The wall clock time of DFRJ and TSVJ-ARM for real data sets.

compared to BFRJ-Combo2, TSVJ-ARM reduces the one-pass buffer size by up to 1.14 times.

Figs. 19 and 20 show the numbers of disk accesses and the wall clock times of BFRJ-Combo1, BFRJ-Combo2, TSVJ-RM, and TSVJ-ARM as the buffer size varies. Fig. 19 additionally shows the results of TSVJ-RM since they have some similarities compared to those of BFRJ-Combo2. These similarities are due to the ordering methods used in these algorithms. BFRJ-Combo2 orders the regions primarily by x coordinate value of the o-space regions and TSVJ-RM orders regions primarily by the ry coordinate value and secondarily by the rx coordinate value of the t-space regions. Since the regions in both algorithms are primarily ordered by one coordinate value of the regions, they show similar performance.[13]

Fig. 19 shows that TSVJ-ARM always outperforms BFRJ-Combo1 and BFRJ-Combo2. Compared to BFRJ-Combo1, with the small buffer, TSVJ-ARM reduces the number of disk accesses by up to 1.87 times; with the large buffer, by up to 1.31 times. Compared to BFRJ-Combo2, with the small buffer, TSVJ-ARM reduces the number of disk accesses by up to 2.89 times; with the large buffer, the two algorithms

13. Recently proposed spatial join algorithms, the Sort/Sweep Spatial Join [8] and the Unified Approach for Indexed and Non-Indexed Spatial Joins [1], also use a similar ordering to optimize the sequence of accessing pages globally. Thus, they are expected to have a similar performance with TSVJ-RM and BFRJ-Combo2. But, we did not perform a direct comparison of them since the type of the algorithms [1], [8] is different from that of TSVJs and BFRJs: These algorithms [1], [8] join a nonindexed file with an indexed file; TSVJs and BFRJs join two indexed files. We also note that the plane sweeping technique uses a similar ordering.

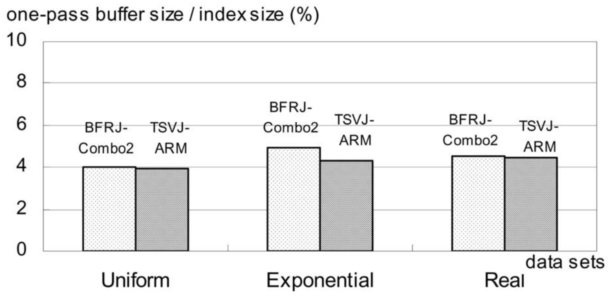


Fig. 18. One-pass buffer sizes of BFRJ-Combo2 and TSVJ-ARM.

show almost the same performance since the buffer size is close to the one-pass buffer sizes.

Fig. 20 shows the result of the wall clock time with curves similar to those in Fig. 19. We note that the number of disk accesses in Fig. 19 for BFRJs is similar with TSVJ-ARM, but the wall clock time in Fig. 20 for BFRJs is worse than TSVJ-ARM. The reason is that TSVJ-ARM uses less CPU time than BFRJ: TSVJ takes advantage of the partial sort result that the index already contains; on the other hand, BFRJs do not use such pre-sorted result. But, we note that overall trends in Figs. 19 and 20 are similar. The results using the uniform and exponential data sets are omitted since they are similar to those using the real data sets.

TSVJ-ARM outperforms both BFRJ-Combo1 and BFRJ-Combo2 regardless of the buffer size because it does not need extra disk or memory overhead for global optimization. In contrast, BFRJ-Combo1 needs extra disk accesses to store the join sequence of the entire pages, causing extra disk access overhead. Thus, it performs worse for large buffers. BFRJ-Combo2 needs memory overhead to store the join sequence, effectively reducing the buffer size. Thus, it performs worse for small buffers. It is a disadvantage of BFRJs that any single algorithm is not efficient for all the range of buffer sizes. TSVJ-ARM solves this problem. In addition, TSVJ-ARM reduces the number of disk accesses by adaptively controlling the order for a given buffer size, while BFRJs do not.

8 Conclusions

In this paper, we first have proposed the new notion of the t-space view. A *t-space view* is a virtual t-space index of an o-space index. The novel aspect of this notion is that it allows us to dynamically interpret an existing o-space index such as the R-tree as a t-space index with no space and negligible time overhead and without modifying its original structure. The idea of the t-space view can be applied to any o-space index where regions and objects stored in the index are represented as minimum bounding rectangles (e.g., R-tree [9], R*-tree [2], X-tree [3], and SKD-Tree [22]).

Next, we have presented the *t-space view join algorithm* based on the notion of the t-space view. The algorithm

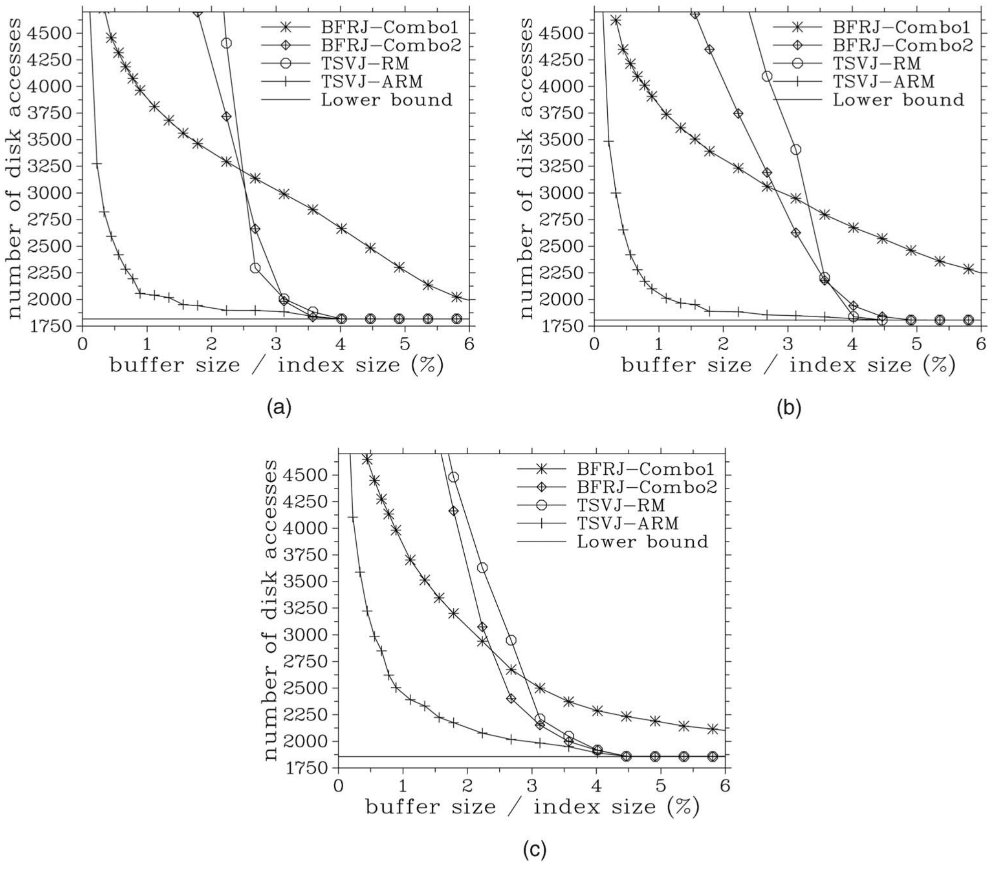


Fig. 19. The number of disk accesses of BFRJ-Combo1, BFRJ-Combo2, TSVJ-RM, and TSVJ-ARM for uniform, exponential, and real data sets. (a) Uniform data set. (b) Exponential data set. (c) Real data set.

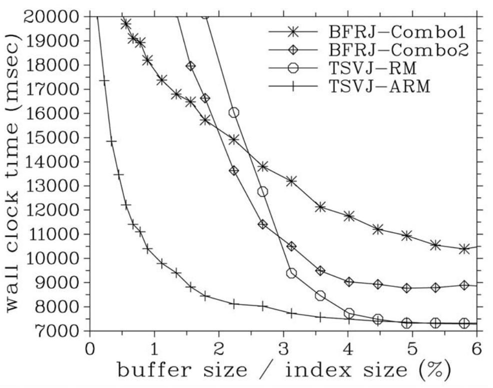


Fig. 20. The wall clock time of BFRJ-Combo1, BFRJ-Combo2, TSVJ-RM, and TSVJ-ARM for real data sets.

globally optimizes the order of accessing the entire pages of indexes by utilizing the special characteristics of the t-space without much extra overhead in terms of memory or disk access time, providing advantages over existing o-space index join algorithms. It also solves the drawbacks of the Transformation-Based Spatial Join algorithm [28], which is a t-space index join algorithm.

Next, we have adapted the formal analysis [18] of the effect of the space filling curves on the performance of the Transformation-Based Spatial Join algorithm [28] to the t-space view join algorithm. Experimental results show that the analysis is highly accurate. We have also applied the ARM order [18] to the t-space view join algorithm. The ARM order adaptively controls the order of accessing pages for a given buffer size. Experiments show that the ARM order also works well (or even better) with the t-space view join algorithm, significantly reducing all three measures used: the one-pass buffer size, the number of disk accesses for a given buffer size, and the wall clock time.

Last, through extensive experiments, we have verified the excellence of the t-space view join. It has been shown that the t-space view join always outperforms the existing spatial join algorithms that use R-trees in the o-space (the Breadth-First Traversal R-tree Join and the Depth-First Traversal R-Tree Join) for all data sets tested. Thus, it constitutes a lower-bound algorithm. Specifically, it reduces the one-pass buffer size by up to 1.14 times for the Breadth-First Traversal R-tree Join and 15.7 times for the Depth-First Traversal R-Tree Join. It reduces the number of disk accesses by up to 2.89 times for the Breadth-First Traversal R-tree Join and 1.55 times for the Depth-First Traversal R-Tree Join. We also have shown that the wall clock time has a similar trend. The reason why the proposed algorithm shows a superior performance is that it performs a global optimization without significant overhead while the others either do only a local optimization or do a global optimization with certain overhead.

The most important contribution of this paper is to show that we can use o-space indexes such as the R-tree in the t-space through the notion of the t-space view. As further study, based on this new notion, we are expecting to develop various new spatial query processing algorithms in the t-space using conventional o-space indexes.

ACKNOWLEDGMENTS

This work was supported by the Ministry of Science and Technology (MOST)/Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc) and by the BK21 project. The original version of this work was done as part of Min-Jae Lee's PhD dissertation while he was at the Computer Science Department of KAIST. Kyu-Young Whang was the corresponding author.

REFERENCES

[1] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, J. Vahrenhold, and J.S. Vitter, "A Unified Approach for Indexed and Non-Indexed Spatial Joins," *Proc. Seventh Int'l Conf. Extending Database Technology (EDBT)*, pp. 413-429, 2000.

[2] 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 (ACM SIGMOD)*, pp. 322-331, 1990.

[3] S. Berchtold, D. Keim, and H.-P. Kriegel, "The X-Tree: An Index Structure for High-Dimensional Data," *Proc. 22nd Int'l Conf. Very Large Data Bases*, pp. 28-39, 1996.

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

[5] C. Faloutsos, "Multiattribute Hashing Using Gray-Codes," *Proc. Int'l Conf. Management of Data (ACM SIGMOD)*, pp. 227-238, 1986.

[6] V. Gaede and O. Günther, "Multidimensional Access Methods," *ACM Computer Surveys*, vol. 30, no. 2, pp. 170–231, 1998.

[7] O. Günther, "The Cell Tree: An Object-Oriented Index Structure for Geometric Databases," *Proc. 15th IEEE Int'l Conf. Data Eng.*, pp. 598-605, 1989.

[8] C. Gurret and P. Rigaux, "The Sort/Sweep Algorithm: A New Method for R-Tree Based Spatial Joins," *Proc. 12th Int'l Conf. Scientific and Statistical Database Management (SSDBM)*, pp. 153-165, 2000.

[9] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. Int'l Conf. Management of Data (ACM SIGMOD)*, pp 47-57, 1984.

[10] D. Hilbert, "Über die stetige Abbildung einer Linie auf Flächenstück," *Annals of Math.*, vol. 38, pp. 459-460, 1891.

[11] G.R. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," *ACM Trans. Database Systems*, vol. 24, no. 2, pp. 265-318, June 1999.

[12] 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.

[13] E.H. Jacox and H. Samet, "Iterative Spatial Join," *ACM Trans. Database Systems*, vol. 28, no. 3, pp. 230-256, Sept. 2003.

[14] H.V. Jagadish, "Spatial Search with Polyhedra," *Proc. 16th IEEE Int'l Conf. Data Eng.*, pp. 311-319, 1990.

[15] H.V. Jagadish, "Linear Clustering of Objects with Multiple Atributes," *Proc. Int'l Conf. Management of Data (ACM SIGMOD)*, pp. 332-342, May 1990.

[16] D.E. Knuth, *The Art of Computer Programming, Vol. III: Sorting and Searching.* Addison Wesley, 1973.

[17] J. Lee, Y. Lee, K. Whang, and I. Song, "A Region Splitting Strategy for Physical Database Design of Multidimensional File Organizations," *Proc. 23rd Int'l Conf. Very Large Data Bases*, pp. 416-425, 1997.

[18] M. Lee, K. Whang, W. Han, and I. Song, "Adaptive Row Major Order: A New Space Filling Curve for Efficient Spatial Join Processing in the Transform Space," *J. Systems and Software*, Oct. 2004.

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

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

[21] N. Mamoulis and D. Papadias, "Slot Index Spatial Join," *IEEE Trans. Knowledge and Data Eng.*, vol. 15, no. 1, pp. 211-231, Jan./Feb. 2003.

[22] B.C. Ooi, K.J. Mcdonell, and R. Sacks-Davis, "Spatial kd-Tree: An Indexing Mechanism for Spatial Databases," *Proc. IEEE Computer Software and Applications Conf.*, pp. 433-438, 1987.

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

[24] B.-U. Pagel, H.-W. Six, and H. Toben, "The Transformation Technique for Spatial Objects Revisited," *Proc. Third Int'l Symp. Spatial Databases (SSD),* pp. 73-88, 1993.

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

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

[27] 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.

[28] J. Song, K. Whang, Y. Lee, M. Lee, and S. Kim, "Spatial Join Processing Using Corner Transformation," *IEEE Trans. Knowledge and Data Eng.,* vol. 11, no. 4, pp. 688-695, July/Aug. 1999.

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

Min-Jae Lee received the BS degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST) in 1995 and the MS and PhD degrees in computer science from KAIST in 1997 and 2004, respectively. Until November 2004, he was a postdoctoral fellow at the Advanced Information Technology Information Center, KAIST. In December 2004, he joined Neowiz, Co., Ltd., in Korea as a research staff member. His research interests include spatial databases, access methods, information retrieval, query processing, database systems, and storage systems. He is a student member of the IEEE and a member of the ACM.

Kyu-Young Whang graduated (summa cum laude) from Seoul National University in 1973 and received MS degrees from the Korea Advanced Institute of Science and Technology (KAIST) in 1975 and Stanford University in 1982. He received the PhD degree from Stanford University in 1984. From 1983 to 1991, he was a research staff member at the IBM T.J. Watson Research Center, Yorktown Heights, New York. In 1990, he joined KAIST, where he is currently a full professor in the Department of Computer Science and director of the Advanced Information Technology Research Center (AITrc). His research interests encompass database systems/storage systems, object-oriented databases, multimedia databases, geographic information systems (GIS), data mining/data warehouses, and XML databases. He is an author of more than 90 papers in refereed international journals and conference proceedings and more than 150 domestic ones. He served as an IEEE Distinguished Visitor from 1989 to 1990, received the Best Paper Award from the Sixth IEEE International Conference on Data Engineering (ICDE) in 1990, served seven times as a program cochair and vice chair for ICDE from 1989 to 2006, and served on program committees of more than a hundred international conferences, including VLDB, ACM SIGMOD, and ICDE. He was the program chair (Asia and Pacific Rim) for COOPIS '98 and the program chair (Asia, Pacific, and Australia) for VLDB2000. He is the general chair of VLDB2006 and was the general chair of PAKDD2003 and DASFAA2004. He received the External Honor Recognition from IBM twice. He is an editor-in-chief of the *VLDB Journal*, having served on the editorial board as a founding member for 15 years. He was an associate editor of the *IEEE Data Engineering Bulletin* from 1990 to 1993 and an editor of the *Distributed and Parallel Databases Journal* from 1991 to 1995. He is currently on the editorial boards of the *IEEE Transactions on Knowledge and Data Engineering*, the *International Journal of GIS*, and the *World Wide Web Journal*. He served as a trustee of the VLDB Endowment from 1998 to 2004 and is a steering committee member of the DASFAA and PAKDD conferences. He was a member of the 10-year best paper award committee of VLDB2003 and VLDB2005 and is currently a member of the awards committee of IEEE ICDE. He is a senior member of the IEEE, a member of the ACM, and a member of IFIP WG 2.6.

Wook-Shin Han received the BS degree in computer engineering from Kyungpook National University in 1994 and the MS and PhD degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST) in 1996 and 2001, respectively. He is currently an assistant professor in the Department of Computer Engineering at Kyungpook National University. His research interests include object-oriented/object-relational databases, XML databases, and information retrieval. He is a member of the IEEE and the ACM.

Il-Yeol Song received the MS and PhD degrees in computer science from Louisiana State University in 1984 and 1988, respectively. He is a professor in the College of Information Science and Technology at Drexel University, Philadelphia, Pennsylvania. His research focuses on the practical application of modeling and design theory to real-world problems. His current research areas include database modeling and design, design and performance optimization of data warehouses and OLAP, database systems for Web-based systems, bioinformatics, and object-oriented analysis and design with UML. He has published more than 120 refereed technical articles in various journals and international conferences. He is a coauthor of the ASIS Pratt_Severn Excellence in Writing Award at National ASIS meeting (1997) and received the Best Paper Award from the 2004 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (IEEE CIBCB 2004). He received a Research Scholar Award from Drexel University in 1992. He has also won 12 research awards from the annual Drexel Sigma Xi Scientific Research Competitions or annual Drexel Research Days. He has won three teaching awards from Drexel University: the Exemplary Teaching Award in 1992, the Teaching Excellence Award in 2000, and the Lindback Distinguished Teaching Award in 2001. He served as a program cochair of CIKM '99, DOLAP '98, DOLAP '99, ER '03, and DGOV '04. He is an associate editor for the *Journal of Database Management* and the *International Journal of E-Business Research*. He is a member of the ACM, the IEEE Computer Society, KSEA, and KOCSEA.