ELSEVIER

Data Knowledge Engineering 41 (2002) 67–83

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

Tie-breaking strategies for fast distance join processing ☆

Hyoseop Shin [a,*], Bongki Moon [b], Sukho Lee [a]

[a] *School of Computer Science and Engineering, Seoul National University, Seoul, Republic of Korea*
[b] *Department of Computer Science, University of Arizona, Tucson, AZ 85721, USA*

Received 21 March 2001; received in revised form 21 July 2001; accepted 14 September 2001

Abstract

The distance join is a spatial join that finds pairs of closest objects in the order of distance by associating two spatial data sets. The distance join stores node pairs in a priority queue, from which node pairs are retrieved while traversing R-trees in top-down manners in the order of distance. This paper first shows that a priority strategy for the tied pairs in the priority queue during distance join processing greatly affects its performance. Then it proposes a probabilistic tie-breaking priority method. The experiments show that the proposed method is always better than alternative methods in the performance perspectives. © 2002 Published by Elsevier Science B.V.

Keywords: Distance join; Probabilistic tie-breaking priority; Distance distribution function

1. Introduction

The distance join is a spatial join which retrieves data pairs in the order of distance by associating two or more data objects in a multi-dimensional space [11,20]. An SQL example of distance join query is as follows:

```
Select *
From A, B
Order By distance (A.object, B.object)
Stop After k;
```

☆ This work was sponsored in part by Brain Korea 21. It was also sponsored in part by National Science Foundation CAREER Award (IIS-9876037) and Research Infrastructure program EIA-0080123. The authors assume all responsibility for the contents of the paper.

* Corresponding author. Tel.: +82-822-3416-0316; fax: +82-822-3416-0303.
E-mail addresses: hsshin@db.snu.ac.kr (H. Shin), bkmoon@cs.arizona.edu (B. Moon), shlee@comp.snu.ac.kr (S. Lee).

0169-023X/02/$ - see front matter © 2002 Published by Elsevier Science B.V.
PII: S0169-023X(02)00021-6

In the example above, the Order By clause forces the query results to be sorted in the increasing order of Euclidean distance, while the Stop After [6,7] clause limits the cardinality of the query results to k.

Applications of distance join include various areas in need of multi-dimensional data such as spatial databases, geographic information systems (GIS), multimedia and image databases, airplane navigation control systems. For example, in spatial database systems, users can request a specific number of spatially closest pairs like "retrieve 5 hotel and restaurant pairs which are closest to each other in this city". In image database systems, a distance join can be used to retrieve the most similar image pairs when associating the two image data sets. In airplane navigation control systems, it can be periodically monitored for safety which airplanes are the closest to others at the moment by processing distance joins queries.

1.1. Overview of distance join using R-Trees

A distance join can be efficiently processed when associated data sets are indexed by spatial data structures such as R-Trees [2,3,10,19], because the search space of a distance join algorithm can be considerably reduced if data nodes are paired by traversing the indexes in top-down manners. It is also desirable to maintain node pairs, which are generated while traversing R-Trees, in a priority queue. The priority of the queue is set to the distance between two nodes of a pair. This makes it possible to examine node pairs by order of distance. We define this priority queue as a main queue (MQ). The main queue is initialized with a pair which consists of two root nodes, one from each R-Tree, at the beginning of the algorithm.

When a pair is fetched from the main queue, it is examined to see whether it is an object pair or a non-object pair. If it is an object pair, it is returned immediately as the next answer. If it is a non-object pair, it is expanded into child node pairs each of which consists of two child nodes, one from each parent node. For example, in Fig. 1, the R-Tree node pair $\langle r, s\rangle$ is dequeued from the main queue and expanded into nine node pairs, $\{\langle r_1, s_1\rangle, \langle r_1, s_2\rangle, \langle r_1, s_3\rangle, \langle r_2, s_1\rangle, \langle r_2, s_2\rangle, \langle r_2, s_3\rangle, \langle r_3, s_1\rangle, \langle r_3, s_2\rangle, \langle r_3, s_3\rangle\}$. After that, these node pairs are inserted into the main queue. Note that the distance of a parent node pair is less than or equal to the distance of any of its child node pairs. This is due to the spatially hierarchical containment property of the spatial indices such that the area represented by a node in those indices include all the areas represented by its child nodes.

If the number of object pairs to be returned as results, k, is known a priori, the performance of the algorithm can be improved further. To make use of this information, k minimum distances are

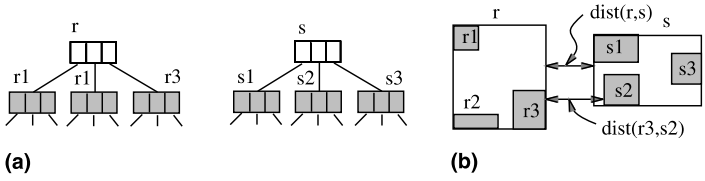


Fig. 1. Relationship between child and parent nodes in R-trees: (a) tree-structured spatial index, (b) spatial containment.

maintained among the distances of the object pairs which have been inserted into the main queue so far. The maximum distance value among the k smallest distances is set to a cut-off distance, $q\mathscr{D}_{max}$. We use $q\mathscr{D}_{max}$ to prune the node pairs that have a distance larger than this value. To maintain the k smallest distances, a max-heap, called distance queue ($\mathscr{Q}_{\mathrm{D}}$), is introduced. $q\mathscr{D}_{max}$, the header value of the distance queue, is initialized to an infinity at the beginning of the algorithm, and gets smaller as the algorithm proceeds, until it becomes equal to the real cut-off value, $\mathscr{D}_{max}$, for k. Note that $\mathscr{D}_{max}$ is not known a priori. Shin et al. [20] defined the distance join using the distance queue as the k-distance join. Fig. 2 represents the framework of k-distance join which will be used in this paper.

1.2. Problem definition

Among the node pairs generated during node expansions, only the node pairs that have a distance smaller than $q\mathscr{D}_{max}$ are inserted into a main queue, and the other node pairs are discarded. The value of $q\mathscr{D}_{max}$ is initialized to an infinity at the start of the algorithm and becomes smaller as the algorithm proceeds. Note that the $q\mathscr{D}_{max}$ value becomes smaller only when an *object pair* whose distance is smaller than $q\mathscr{D}_{max}$ is inserted into the main queue. Its implication is that the more object pairs with short distances are inserted into the main queue in the early stages of the algorithm, the more quickly the value of $q\mathscr{D}_{max}$ is decreased. This in turn will reduce the processing time of distance join queries.

Consider two pairs of objects shown in Fig. 3. The distance between objects r_1 and s_1 is shorter than that between objects r_2 and s_2 (that is, $dist(r_1, s_1) < dist(r_2, s_2)$). Nonetheless, the distances of their parent node pairs are identical (that is, $dist(parent(r_1), parent(s_1)) = dist(parent(r_2), parent(s_2))$). This is because the parent nodes of r_1 and s_1 overlap each other, and so do the parent

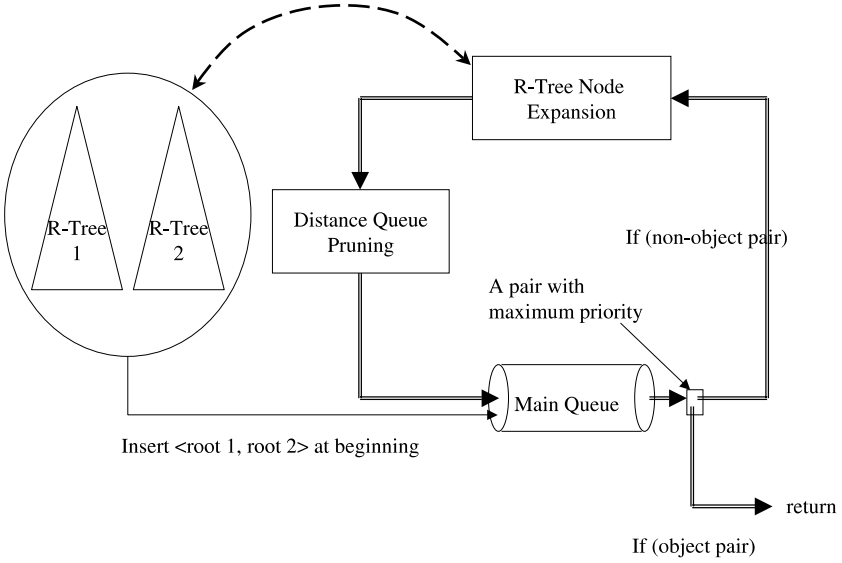


Fig. 2. The framework of k-distance join.

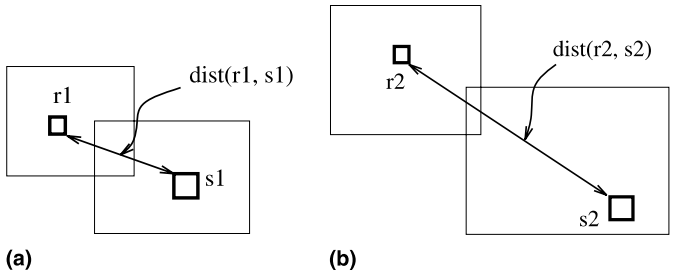


Fig. 3. Motivating examples to break ties: (a) a pair of objects with a short distance, (b) a pair of objects with a long distance.

nodes of r_2 and s_2. When such node pairs of an equal distance are inserted into a main queue, they are indistinguishable in terms of priority and the relative order of node expansions between the two pairs will be arbitrary (or somewhat affected by the way of implementing a max-heap).

In fact, the chance that a pair of non-object nodes overlap is expected to be higher than that of a pair of objects, because a non-object node tends to occupy a larger space than an object does. In our experiments, the distance was zero for more than 98% of non-object node pairs dequeued from a main queue. Consequently, it can have a critical impact on the processing speed of distance join queries to determine the order of such a large number of zero-distance node pairs in the main queue. Therefore, it is not a sufficient solution to rely solely on node distances in order to determine the order of node pairs in the main queue. Instead, it is desirable to employ a secondary priority for breaking the ties among node pairs.

In this paper, we present the optimized priority measure to order node pairs of the same distance to further improve processing distance joins. In Section 2, two viable priority methods are introduced, and the probabilistic priority method is proposed in Section 3. An approximate method of the probabilistic priority measurement is presented in Section 4. Related work is presented in Section 5. Performance evaluation of our method is shown in Section 5. Finally, we conclude this paper in Section 6.

2. Related work

Studies [8,11,20] on the distance join queries are found in the recent literature. Shin et al. [20] discriminated between incremental distance joins and k-distance joins. Incremental distance joins can be used if k is not known a priori, otherwise, k-distance joins can be used to accelerate the performance of the incremental distance joins using a distance queue and $q\mathscr{D}_{max}$. The secondary priority methods described in this paper are mainly applied to k distance joins.

The tie-breaking problem in distance join processing has been mentioned in the previous work. However, the ultimate goal of the problem was not defined clearly and accordingly the solutions were restrictive. Hjaltason and Samet [11] proposed to first select a node pair, among the tied node pairs in a main queue, which has a node with the deepest depth in the R-Tree. This method is based on a heuristic that it is desirable to produce object pairs earlier than non-object pairs.

Corral et al. [8] proposed to first select a node pair, among the tied node pairs in a main queue, which has a node with the largest area.

A tie-breaking method helps reduce the number of insertions of node pairs into a main queue by smartly choosing a node pair among the zero-distant node pairs in the main queue. Apart from this problem addressed in this paper, once a node pair is selected among the node pairs in the main queue, Shin et al. [20] mentioned a problem how to smartly expand a node pair into their child node pairs to reduce the number of distance computations during each node expansion. In that paper, a novel strategy, defined as the *sweeping index* method in which sweeping axis and direction are selected on a node pair basis, was proposed for optimizing plane-sweeping techniques [17]. The tie-breaking and node expansion strategies are the two key leverages for accelerating the distance join algorithms.

It may be argued that a spatial distance join query can be processed by a spatial join operation [1,4,5,9,12–14,16,18] followed by a sort operation. Specifically, if a cut-off distance value, $\mathscr{D}_{max}$, can be predicted precisely for a given stopping cardinality k, we can use a spatial join algorithm with a `within` predicate instead of an `intersect` predicate to find the k nearest pairs of objects. In practice, however, it is almost impossible to estimate an accurate $\mathscr{D}_{max}$ value for a given stopping cardinality k, and, to the best of our knowledge, no method for estimating such a cutoff value has been reported in the literature. If the $\mathscr{D}_{max}$ value is overestimated, then the results from a spatial join operation may contain too many candidate pairs, which may cause a long delay in a subsequent stage to sort all the candidate pairs. On the other hand, if the $\mathscr{D}_{max}$ value is underestimated, a spatial join operation may not return a sufficient number of object pairs. Then, the spatial join operation should be repeated with a new estimate of $\mathscr{D}_{max}$, until k or more pairs are returned. This may cause a significant amount of waste in processing time and resources.

3. Secondary priorities for breaking ties

Although the primary priority of the main queue is already fixed as the distance, in the performance perspective of the distance join, a secondary priority for breaking tied pairs should be given to decrease $q\mathscr{D}_{max}$ as quickly as possible. In other words, it is desirable to force node pairs with smaller distances to be generated earlier than ones with larger distances.

3.1. Maximum distance priority method

In this method, the maximum distance between the two nodes of each pair is used as a secondary priority of the main queue; among the tied node pairs in the main queue, those who have smaller maximum distances are supposed to get higher priorities than those who have larger maximum distances. It is reasoned that if the maximum distance of a node pair is smaller, the distances of its child node pairs are distributed in a smaller interval of distances, and thus the expansion of the node pair may possibly generate more child node pairs of smaller distances.

This method does not always show desirable results. For instance, consider pair A and B in Fig. 4. According to this method, node pair A will have higher priority than node pair B, as the maximum distance of A is smaller than the maximum distance of B. However, the two nodes of pair B overlap each other more and accordingly will generate more child node pairs with smaller

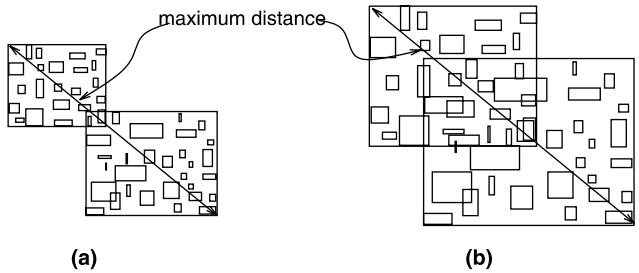


Fig. 4. A counter example of maximum distance priority method: (a) pair A, (b) pair B.

distances than A. It is therefore desirable to give higher priority to pair B. As seen in this example, this method does not reflect how much the two nodes of each pair are overlapped.

3.2. Relative overlap priority method

The relative overlap priority method aims at alleviating the problem raised by the maximum distance priority method. This method assigns higher priorities to the node pairs which have higher relative overlap values. The relative overlap of a node pair is obtained by dividing its overlapped area by the sum of the areas of the two nodes as follows:

$$RelativeOverlap(\langle r, s\rangle) = \frac{area(r \cap s)}{area(r) + area(s)}. \quad (1)$$

If the relative overlap of a node pair is higher than that of another node pair, it is likely to generate a larger number of child node pairs which have relatively small distance values. However, like the maximum distance priority method, this method does not always show desirable results, either. For instance, let's consider pair A and B in Fig. 5.

According to this method, node pair B will have higher priority than node pair A, because the overlap rate of B is higher than the overlap rate of A. However, in reality, node pair A will still generate a larger number of child node pairs with relatively small distances compared to node pair B. As seen in this example, this method cannot reflect how wide the child nodes of each pair spread.

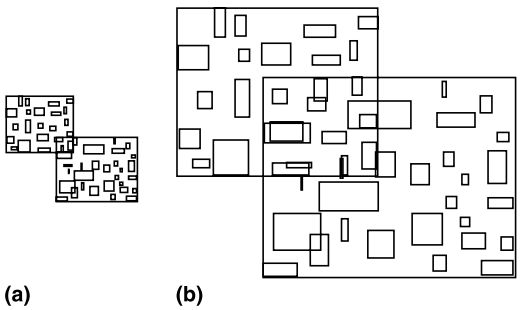


Fig. 5. A counter example of overlap rate priority method: (a) pair A, (b) pair B.

4. Probabilistic tie-breaking priority method

Each priority method described in the previous section does not guarantee the optimal ordering of node pairs due to its own limitations. The goal of the secondary priority of the main queue is to decrease $q\mathscr{D}_{max}$ as quickly as possible, so that node pairs with relatively small distances are generated in the early stages of the distance join. To be more specific, it aims at making the $q\mathscr{D}_{max}$ value approach the real cut-off distance, $\mathscr{D}_{max}$, as quickly as possible. In this section, we propose a novel probabilistic method for finding a near-optimal secondary priority to fulfill the goal.

Note that the $q\mathscr{D}_{max}$ value is always no smaller than $\mathscr{D}_{max}$. Conceptually, an optimal secondary priority method must select a node pair among those of zero-distance from the main queue, in a way that the selected node pair is expected to generate a maximum number of the child node pairs that have distances smaller than $\mathscr{D}_{max}$.

Fig. 6(a) shows an example of an overlapped node pair and Fig. 6(b) shows the distance distribution of the child node pairs. Let $f(t)$ represent a distance distribution function. Then, the ratio of the number of the child node pairs which have distances smaller than $\mathscr{D}_{max}$ value, among all the child node pairs that are generated from the node pair, is formulated as

$$CandidateRatio(\langle r, s\rangle) = \frac{\int_0^{\mathscr{D}_{max}} f(t)\mathrm{d}t}{\int_0^{d_{parent}} f(t)\mathrm{d}t}, \tag{2}$$

where d_{parent} is the maximum distance of the parent node pair.

Then, the *probabilistic secondary priority method* will select a node pair, among the zero-distant node pairs in the main queue, which is an object node pair or has the largest value of *Candidate Ratio*.

4.1. Estimation of $\mathscr{D}_{max}$

It is impossible to know the cut-off distance value, $\mathscr{D}_{max}$, for the specific k before the processing of a given distance join query. Following the method proposed by Shin et al. [20], we estimate $e\mathscr{D}_{max}$ based on the assumption that the given datasets are uniformly distributed.

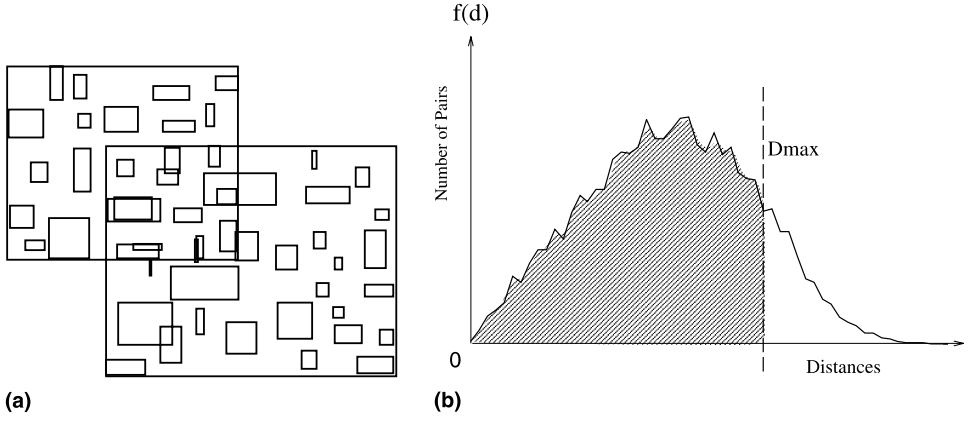


Fig. 6. Distance distribution of child node pairs for an overlapped node pair: (a) an overlapped node pair, (b) distance distribution of its child node pairs.

Let $|R|$ and $|S|$ be the number of data objects in sets R and S, respectively. Then, the number of data objects in S within a distance d from a data object in R is approximated by $|S| \times \frac{\pi \times d^2}{area(R \cap S)}$. Therefore, the total number of object pairs (k) within a distance d is given by

$$k = |R| \times |S| \times \frac{\pi \times d^2}{area(R \cap S)}.$$

For a given k value as the number of requested query results, an estimation of $e\mathscr{D}_{max}$ can be obtained from the above equation as follows:

$$e\mathscr{D}_{max} = \sqrt{k \times \rho} \quad \left(\text{where } \rho = \frac{area(R \cap S)}{\pi \times |R| \times |S|}\right). \tag{3}$$

4.2. Distance distribution function

In this section, we derive a distance distribution function for a given pair of nodes that overlap each other. The distance distribution function $f(d)$ is the expected number of object pairs whose distance is d. We derive a formula of $f(d)$ for a one-dimensional space, and then extend it to higher-dimensional spaces.

4.2.1. Distance distribution function for one-dimensional node pair

In a one-dimensional space, a pair of nodes are represented as two line segments, as shown in Fig. 7. The expected number of object pairs at distance zero (i.e., $d = 0$) is the length of overlap of the two line segments. That is,

$$f(0) = Overlap. \tag{4}$$

If a pair of objects are apart by a non-zero distance d, they can always be positioned at the same location by shifting either of the line segments to left or right direction by distance d. Thus, for a non-zero distance (i.e., $d \neq 0$), the expected number of object pairs at distance d is the length of overlap of the two line segments with one of them shifted to left and right by d. That is,

$$f(d) = f_l(d) + f_r(d), \tag{5}$$

where $f_l(d)$ is the length of overlap with a line segment shifted to the left by d, and $f_r(d)$ is the length of overlap with the same line segment shifted to the right by d.

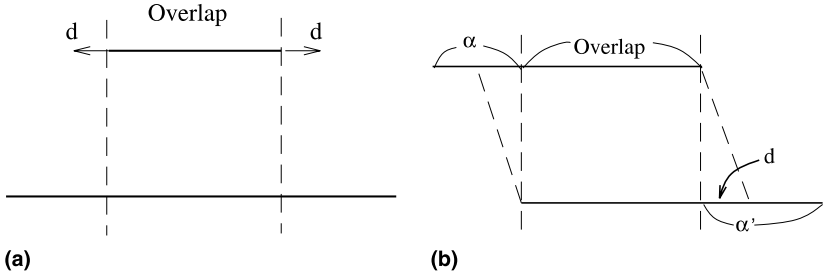


Fig. 7. Distance distribution for one-dimensional node pair: (a) expansions in two directions, (b) example of an expansion.

The formula of $f_r(d)$ for $d > 0$ can be obtained from Fig. 7(b) as follows:

$$f_r(d) = \max\{\min(\alpha, d) + Overlap - \max(d - \alpha', 0), 0\}, \tag{6}$$

where α is the left margin of a node of the given pair, α' is the right margin of the other node of the pair, and *Overlap* is the overlapped interval of the pair.

4.2.2. Distance distribution function for two-dimensional node pair

In a two-dimensional space, a pair of nodes are represented as two rectangles, as shown in Fig. 8. The expected number of object pairs at distance zero (i.e., $d = 0$) is the intersected area of the two rectangles. That is,

$$f(0) = O_x \times O_y, \tag{7}$$

where O_x and O_y represent the side lengths of the intersected area along x and y axes, respectively.

For a non-zero distance (i.e., $d \neq 0$), the notion of shifting a line segment in a one-dimensional space can be extended to the two-dimensional space. The expected number of object pairs at distance d is the intersected area of the two rectangles with one of them drifted to any direction by d as shown in Fig. 8(a). That is,

$$f(d) = f_{lu}(d) + f_{ld}(d) + f_{ru}(d) + f_{rd}(d), \tag{8}$$

where $f_{lu}(d), f_{ld}(d), f_{ru}(d)$ and $f_{rd}(d)$ are the intersected areas with a rectangle drifted by d to left and upward, left and downward, right and upward, and right and downward directions, respectively.

The formula of $f_{lu}(d)$ for $d > 0$ can be obtained from Fig. 8(b) as follows:

$$f_{lu}(d) = \int_{x^2+y^2=d^2} \max\{(O_x + x - x'), 0\} \times \max\{(O_y + y - y'), 0\} \mathrm{d}(x, y), \tag{9}$$

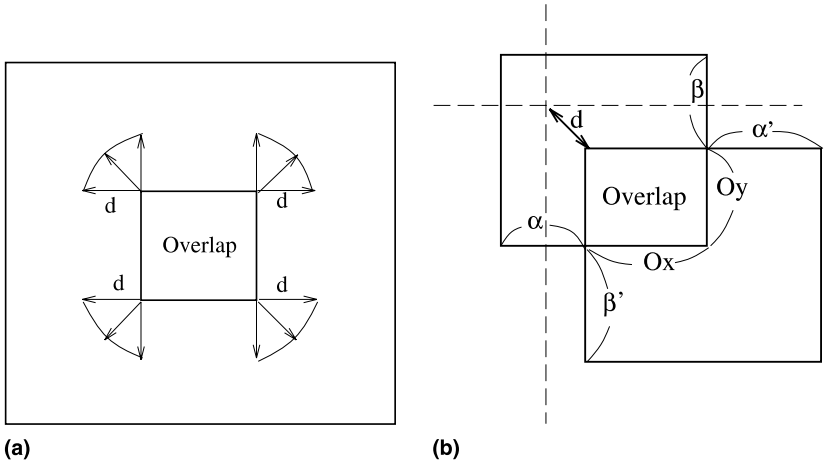


Fig. 8. Distance distribution for two-dimensional node pair: (a) expansions in four directions, (b) example of an expansion.

where α and β represent the margin of a node of the given pair along the x and y axis each, α' and β' represent the margin of the other node of the pair along the x and y axis each, and $0 \leqslant x \leqslant d$, $x = Min\{d, \alpha\},\ y = Min\{d, \beta\},\ x' = Max\{d - \alpha', 0\},\ y' = Max\{d - \beta', 0\}$.

4.2.3. Distance distribution function for N-dimensional node pair

The distance distribution function for an N-dimensional space can be obtained the same way as in the previous sections. In an N-dimensional space, an N-dimensional hyper-rectangle can be drifted in 2^N directions. Therefore, $f(d)$ is formulated by the following equations:

$$f(0) = \prod_{i=1}^{N} O_i \tag{10}$$

and, for a non-zero d,

$$f(d) = \sum_{k=1}^{2^N} f_k(d), \tag{11}$$

$$f_k(d) = \int_{\sum_{i=1}^{N} x_i^2 = d^2} \prod_{i=1}^{N} \max\{O_i + x_i - x_i', 0\} \mathrm{d}(x_1, \ldots, x_N), \tag{12}$$

where α_i represents the margin of a node of the given pair along the ith axis, α_i' represents the margin of the other node of the pair along the ith axis, O_i represents the overlapped interval of the pair along the ith axis, and $0 \leqslant x_i \leqslant d,\ x_i = Min\{d, \alpha_i\},\ x_i' = Max\{d - \alpha_i', 0\}$.

5. Approximation of distance distribution

The distance distribution function described in the previous section provides conceptual formulae based on the uniformity assumption of given datasets. In practice, we use an approximation of the formulae because of the high computational complexity of the formulae. As seen in Fig. 9, the distribution of child node pairs of a node pair generally follows a biased normal distribution, or a Gaussian distribution. Therefore, if the average distance value (DA) in the distribution is available, the probabilistic distribution can be simplified to a triangular one as in Fig. 9.

The approximate value of the average distance can be obtained by computing the distance between the center points of the two nodes in the given node pair. To increase the correctness of the average distance, more than one points can be selected from each node. For example, in Fig. 10, four points are selected from each node and the average distance value is obtained from the distances of 16 point pairs. Once the average distance, DA, is given, the distance distribution function, $f(d)$, is abbreviated as follows:

$$f(d) = \begin{cases} \frac{\Delta}{DA} \times d & \text{if } d \leqslant DA, \\ \frac{\Delta}{DA - d_{parent}} \times (d - d_{parent}) & \text{if } DA < d \leqslant d_{parent}, \\ 0 & \text{if } d > d_{parent}, \end{cases} \tag{13}$$

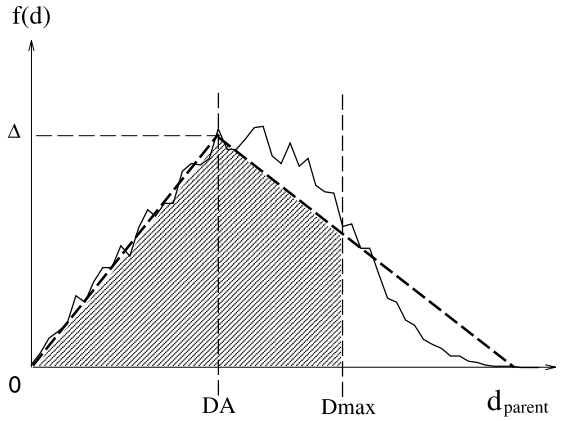


Fig. 9. Approximation of distance distribution function.

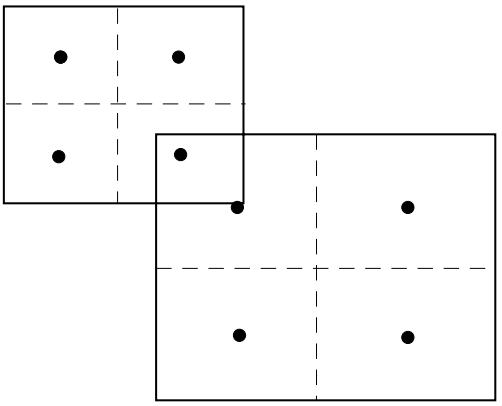


Fig. 10. Sampling of multiple center points to compute average distance.

where Δ represents the $f(d)$ value when $d = DA$, and d_{parent} represents the maximum distance of the parent pair.

6. Performance evaluation

In this section, we evaluate the proposed methods empirically and compare with the previous work. In particular, the maximum distance priority, overlap rate priority, probabilistic priority methods (denoted as *MaxDist, Overlap*, and *Prob*, respectively) were compared with the depth priority [11] and area priority [8] methods (denoted as *Depth* and *Area*, respectively) which were described in the related work.

Experiments were performed on a Sun Ultrasparc-II workstation running on Solaris 2.7. This workstation has 256 MBytes of memory and 9 GBytes of disk storage (Seagate ST39140A) with Ultra 10 EIDE interface. The disk is locally attached to the workstation and used to store

databases, queues and any temporary results. We used the direct I/O feature of Solaris for all the experiments to avoid operating system's cache effects, and the average disk access bandwidth was about 0.5 MBytes/s for random accesses and about 5 MBytes/s for sequential accesses.

To evaluate distance join algorithms, we used real-world data sets in TIGER/Line97 from the US Bureau of Census [15]. The particular data sets we used were 633,461 streets and 189,642 hydrographic objects from the Arizona state. These data sets were indexed by R*-trees [2].

To compare the performances of these methods, we enhanced $\mathscr{B}$-**KDJ** [20] distance join algorithm with a tie-breaking mechanism. We call it $\mathscr{TB}$-**KDJ**. The cardinality k of distance join query results were varied from 1 to 100,000.

6.1. $\mathscr{TB}$-*KDJ*: Tie-breaking K-distance join algorithm with bi-directional expansion

$\mathscr{TB}$-**KDJ** is same as $\mathscr{B}$-**KDJ** except that $\mathscr{TB}$-**KDJ** selects a node pair following a tie-breaking priority for the equal-distant node pairs in a main queue (Algorithm 1 – line 5), while $\mathscr{B}$-**KDJ** arbitrarily selects a node pair among those tied node pairs.

The original $\mathscr{B}$-**KDJ** algorithm uses $q\mathscr{D}_{max}$ from the distance queue $\mathscr{Q}_D$ as a cutoff value to examine node pairs. If a pair of nodes $\langle r, s\rangle$ removed from the main queue are a pair of objects, then the object pair is returned as a query result. Otherwise, the pair is expanded by the *Plane-Sweep* procedure for further processing.

Algorithm 1.

$\mathscr{TB}$-KDJ: Tie-Breaking k-Distance Join Algorithm with Bi-directional Expansion and Plane Sweep

1: set *AnswerSet* $\leftarrow$ an empty set;
2: set $\mathscr{Q}_M, \mathscr{Q}_D \leftarrow$ empty main and distance queues;
3: insert a pair $\langle R.root, S.root\rangle$ into the main queue $\mathscr{Q}_M$;
4: **while** $|AnswerSet| < k$ and $\mathscr{Q}_M \neq \emptyset$ **do**
5: select c in $\mathscr{Q}_M$ which has the highest tie-breaking priority among the node pairs with the smallest distance;
6: **if** c *is an* $\langle object, object\rangle$ **then** $AnswerSet \leftarrow \{c\} \cup AnswerSet$;
7: **else** *PlaneSweep*(c);
end
procedure PlaneSweep($\langle l, r\rangle$)
8: set L $\leftarrow$ sort_axis({child nodes of l}); // Sort the child nodes of l by axis values.
9: set R $\leftarrow$ sort_axis({child nodes of r}); // Sort the child nodes of r by axis values.
10: **while** $L \neq \emptyset$ *and* $R \neq \emptyset$ **do**
11: $n \leftarrow$ a node with the min axis value $\in L \cup R$; // n becomes an anchor.
12: **if** $n \in L$ **then**
13: $L \leftarrow L - \{n\}$; *SweepPruning*(n, R);
else
14: $R \leftarrow R - \{n\}$; *SweepPruning*(n, L);
end
end

```
    procedure SweepPruning(n, List)
15: for each node m ∈ List in an increasing order of axis value do
16:   if axis_distance(n, m) > qD_max then return;        // No more candidates.
17:   if real_distance(n, m) ⩽ qD_max then
18:     insert ⟨n, m⟩ into Q_M;
19:     if ⟨n, m⟩ is an ⟨object, object⟩ then insert real_distance(n, m) into Q_D;    //qD_max modified.
      end
    end
```

Assume that a sweeping axis (i.e., x- or y-dimensional axis) and a sweeping direction (i.e., forward or backward) are determined. Then, the child nodes of r and s are sorted by x or y coordinates of one of the corners of their MBRs in an increasing or decreasing order, depending on the choice of sweeping axis and sweeping direction. Each node encountered during a plane sweep is selected as an *anchor*, and it is paired up with child nodes in the other group.

Since an axis distance between any pair $\langle r, s\rangle$ is always smaller than or equal to their real distance (i.e., $axis\_distance(r, s) \leqslant real\_distance(r, s)$), real distances are computed only for nodes whose axis distances from the anchor are within the current $q\mathscr{D}_{max}$ value (line 17). Given that a real distance is more expensive to compute than an axis distance, it may yield non-trivial performance gain. Then, each pair whose real distance is within $q\mathscr{D}_{max}$ is inserted into the main queue $\mathscr{Q}_{\mathrm{M}}$ (line 18). If it is a pair of objects, then update the current $q\mathscr{D}_{max}$ value by inserting the real distance of the object pair into the distance queue $\mathscr{Q}_{\mathrm{D}}$ (line 19).

6.2. Experimental results

A tie-breaking priority measure of a main queue mainly affects the operation of the main queue. The task of managing a main queue is largely I/O intensive as well as CPU intensive. In implementing a main queue, a hybrid memory/disk scheme [11] and a technique based on range partitioning [7] are used. A partition in the shortest distance range is kept in memory as a heap structure, while the rest of partitions are stored on disk as merely unsorted piles. Shin et al. [20] proposed to use Eq. (3) to determine the boundary distance values of the partitions to minimize the required number of partition movements between main memory and disk. In this scheme of managing a main queue, inserting a node pair into the in-memory portion of the queue is CPU intensive, while inserting into the disk resident portion is I/O intensive. Thus, the tie-breaking method can directly affect the total response time of distance join algorithms.

First, we measured the number of queue insertions when each method was applied to $\mathscr{TB}$-**KDJ**. Table 1 shows the number of queue insertions for the each method when k is varied from 1 to 100,000.

Area method [8] showed very poor performance compared to other methods for every k value. For better presentation, we do not include *Area* method in the following experiment sets. Fig. 11 shows the queue insertion ratio of each method in percentage against *None* method in which no tie-breaking priority is applied. As shown in Fig. 11, our proposed methods, *MaxDist, Overlap* and *Prob* methods showed better performances than the other methods for every k value. Especially, *Prob* method was always better than the other methods. It reduced the queue insertions

Table 1
Number of queue insertions of $\mathscr{TB}$-**KDJ**

Stopping cardinality k	Tie-breaking methods					
	None	Depth	Area	Overlap	MaxDist	Prob
1	17,359	9835	505,377	12,401	7698	6753
10	23,040	15,627	506,259	14,186	12,237	11,529
100	24,583	17,369	511,789	15,641	14,390	12,675
1000	35,472	31,392	539,457	28,102	27,323	23,893
10,000	122,566	130,641	642,479	126,333	113,210	109,883
100,000	913,360	922,115	1,450,896	862,049	813,015	756,118

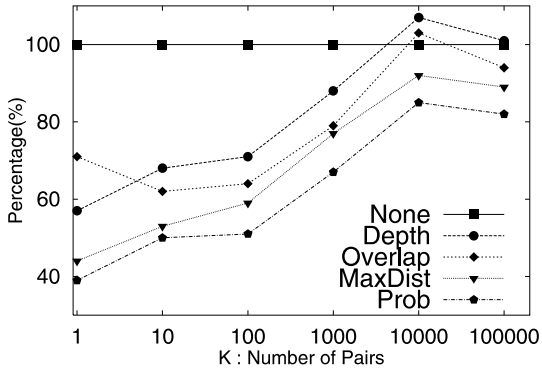


Fig. 11. Queue insertions ratio of $\mathscr{TB}$-**KDJ**.

of $\mathscr{TB}$-**KDJ** algorithm by 61 (when $k = 1$) to 15 (when $k = 10{,}000$) percent. It was also noted that tie-breaking methods are more efficient in small k values than in large k values.

Fig. 12 shows the distance computation ratios of the various methods to the *None* method. Although the tie-breaking methods are aiming at reducing the number of insertions of node pair into a main queue, the number of distance computations is also reduced in proportion to the number of queue insertions. In our experiments, 27 (when $k = 1$) to 13 (when $k = 10{,}000$) percent amount of distance computations was saved by the *Prob* method.

The number of R-tree node accesses, which constitutes important performance measures of distance join algorithms together with the number of distance computations and the number of queue insertions, is invariable regardless of any tie-breaking method. Nevertheless, Given a limited R-tree buffer size, the number of R-tree nodes which are actually accessed from the disk may be varied with each tie-breaking method. Thus, an efficient R-tree buffer management policy for each tie-breaking method can help reduce the actual number of R-tree node accesses. The issue of the R-tree buffer management for distance join processing is beyond the scope of this paper. Fig. 13 shows the total response time ratio of each method to the *None* method. Again, our proposed methods were better than the other methods. *Prob* showed the best result by reducing the response time of $\mathscr{TB}$-**KDJ** by 45% (when $k = 1$) to 13% (when $k = 10{,}000$).

Meanwhile, the *Prob* method proposed in this paper requires to sample center points for any given node pair in order to approximate the average distance of the node pair. Generally, the

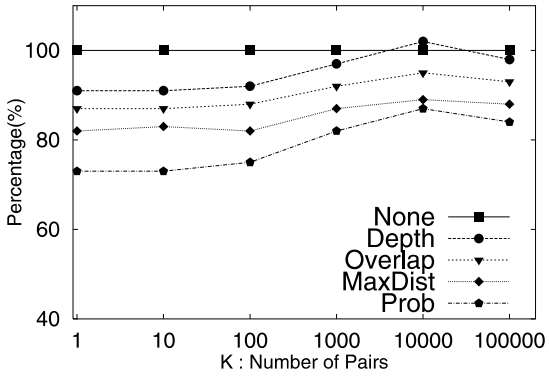


Fig. 12. Distance computations ratio of $\mathcal{TB}$-**KDJ**.

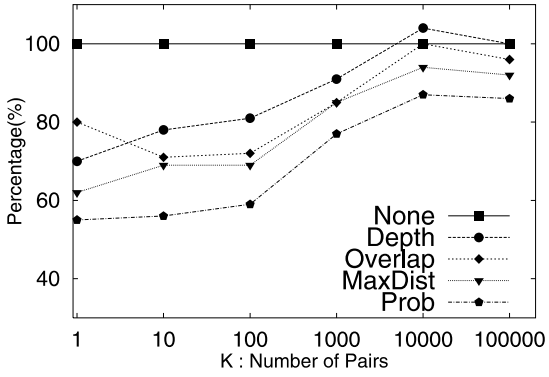


Fig. 13. Response times ratio of $\mathcal{TB}$-**KDJ**.

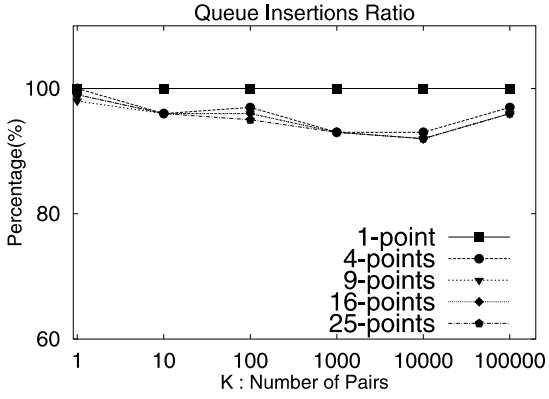


Fig. 14. Effect of sampling numbers of center points on approximated probabilistic method.

more number of center points are sampled, the more correctly the average distance will be approximated. To figure out the adequate number of samplings, we measured the performance of $\mathscr{TB}$-**KDJ** each for the varying number of samplings from 1 to 25. Fig. 14 shows their relative response time ratios when 1-point case was set to 100%. The result implied that 4-point sampling was enough to estimate the average distance for every k between 1 and 100,000.

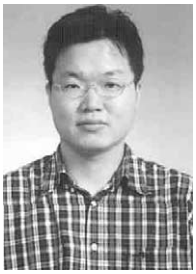
7. Conclusions

This paper proposed tie-breaking priority methods to order node pairs of the same distance in a main queue for efficient processing of distance join queries. Since a large number of overlapping node pairs are generated during a distance join processing, a priority measure to order them has a great impact on the performances of distance join algorithms. Especially, a good tie-breaking priority can greatly reduce the number of insertions of node pairs into the main queue. In this paper, we first presented two heuristic approaches to breaking methods, *MaxDist* and *Overlap*. These methods, however, did not always show desirable results in all cases. To fully cover these problems in a systematic way, we proposed a probabilistic tie-breaking method and its approximated version to be practical. Our proposed method enhanced the overall performances of distance join queries up to 44% in the experiments.

References

[1] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, J.S. Vitter, Scalable sweeping-based spatial join, in: Proceedings of the 24th VLDB Conference, New York, USA, June 1998, pp. 259–270.
[2] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The R*-tree: An efficient and robust access method for points and rectangles, in: Proceedings of the 1990 ACM-SIGMOD Conference, Atlantic City, NJ, May 1990, pp. 322–331.
[3] S. Berchtold, D.A. Keim, H.-P. Kriegel, The X-tree: An index structure for high-dimensional data, in: Proceedings of the 22nd VLDB Conference, Bombay, India, September 1996.
[4] T. Brinkhoff, H.-P. Kriegel, R. Schneider, B. Seeger, Multi-step processing of spatial joins, in: Proceedings of the 1994 ACM-SIGMOD Conference, Minneapolis, Minnesota, May 1994, pp. 197–208.
[5] T. Brinkhoff, H.-P. Kriegel, B. Seeger, Efficient processing of spatial joins using R-trees, in: Proceedings of the 1993 ACM-SIGMOD Conference, Washington, DC, May 1993, pp. 237–246.
[6] M.J. Carey, D. Kossmann, On saying "enough already!" in SQL, in: Proceedings of the 1997 ACM-SIGMOD Conference, Tucson, AZ, May 1997, pp. 219–230.
[7] M.J. Carey, D. Kossmann, Reducing the braking distance of an SQL query engine, in: Proceedings of the 24th VLDB Conference, New York, NY, August 1998, pp. 158–169.
[8] A. Corral, Y. Manolopoulos, Y. Theodoridis, M. Vassilakopoulos, Closest pair queries in spatial databases, in: Proceedings of the 2000 ACM-SIGMOD Conference, Dallas, TX, May 2000, pp. 189–200.
[9] O. Günther, Efficient computation of spatial joins, in: Proceedings of the 9th International Conference on Data Engineering, 1993, pp. 50–60.
[10] A. Guttman, R-Trees: A dynamic index structure for spatial searching, in: Proceedings of the 1984 ACM-SIGMOD Conference, Boston, MA, June 1984, pp. 47–57.
[11] G.R. Hjaltason, H. Samet, Incremental distance join algorithms for spatial databases, in: Proceedings of the 1998 ACM-SIGMOD Conference, Seattle, WA, June 1998, pp. 237–248.
[12] Y.-W. Huang, N. Jing, E. Rundensteiner, Spatial joins using R-Tree: Breadth-first traversal with global optimization, in: Proceedings of the 23rd VLDB Conference, 1997, pp. 396–405.

[13] M.-L. Lo, C.V. Ravishankar, Spatial joins using seeded trees, in: Proceedings of the 1994 ACM-SIGMOD Conference, Minneapolis, Minnesota, May 1994, pp. 209–220.
[14] M.-L. Lo, C.V. Ravishankar, Spatial hash-join, in: Proceedings of the 1996 ACM-SIGMOD Conference, Montreal, Canada, June 1996, pp. 247–258.
[15] Bureau of the Census, Tiger/Line Precensus Files: 1997 technical documentation, Washington, DC, 1997.
[16] J.M. Patel, D.J. DeWitt, Partition based spatial-merge join, in: Proceedings of the 1996 ACM-SIGMOD Conference, Montreal, Canada, June 1996, pp. 259–270.
[17] F.P. Preparata, M.I. Shamos, Computational Geometry: an Introduction, Springer, New York, 1991.
[18] D. Rotem, Spatial join indices, in: Proceedings of the 7th International Conference on Data Engineering, 1991, pp. 500–509.
[19] T.K. Sellis, N. Roussopoulos, C. Faloutsos, The R+-Tree: A dynamic index for multi-dimensional objects, in: Proceedings of the 13th VLDB Conference, 1987, pp. 507–518.
[20] H. Shin, B. Moon, S. Lee, Adaptive multi-stage distance join processing, in: Proceedings of the 2000 ACM-SIGMOD Conference, Dallas, TX, May 2000, pp. 343–354.



Hyoseop Shin is a PhD student in the School of Computer Science and Engineering, Seoul National University, Seoul, Korea. His current research interests include high-dimensional databases, geographic information systems, XML, embedded databases. He received his MS and BS degrees in the Department of Computer Engineering from Seoul National University, Seoul, Korea, in 1996 and 1994, respectively. He visited the Department of Computer Science, University of Arizona, during 1999–2001 each with the support of Korea Science and Engineering Foundation and Brain Korea 21.

Bongki Moon is an Assistant Professor in the Department of Computer Science, University of Arizona. His current research interests include high performance spatial and temporal databases, scalable web servers, data mining and warehousing, and parallel and distributed processing. He received his PhD degree in Computer Science from University of Maryland, College Park, in 1996, and his MS and BS degrees in Computer Engineering from Seoul National University, Korea, in 1985 and 1983, respectively. He was a member of the research staff at Communication Systems Division, Samsung Electronics Corp., Korea, from 1985 to 1990. He is the recipient of the National Science Foundation CAREER Award.

Sukho Lee received his BA degree in Political Science and Diplomacy from Yonsei University, Seoul, Korea, in 1964 and his MS and PhD in Computer Sciences from the University of Texas at Austin in 1975 and 1979, respectively. He is currently a professor of the School of Computer Science and Engineering, Seoul National University, Seoul, Korea, where he has been leading the Database Research Laboratory. He has served as the president of Korea Information Science Society. His current research interests include database management systems, spatial database systems, and multimedia database systems.