

DISTANCE-ASSOCIATED JOIN INDICES FOR SPATIAL RANGE SEARCH

Wei Lu and Jiawei Han[†]
School of Computing Science
Simon Fraser University
British Columbia, Canada V5A 1S6
e-mail: {weilu, han}@cs.sfu.ca

Abstract

Spatial join indices are join indices constructed for spatial objects. Similar to join indices for relational database systems, spatial join indices improve efficiency of spatial join operations. In this paper, a distance-associated join index structure is developed to speed up spatial queries especially for spatial range queries. Three distance-associated join indexing mechanisms: basic, ring-structured and hierarchical, are presented and studied. Our analysis and performance study shows that distance-associated spatial join indices substantially improve the performance of spatial queries, and different structures are best suited for different applications.

1. Introduction

Spatial databases have been widely used in geographical and CAD/CAM applications. Spatial indexing mechanisms have been studied extensively. R+-trees [10], Quad-trees [8], K-D-B trees [6], Grid files [4] and other structures have been popularly used as indexing structures for spatial object retrieval [1-3, 5, 9, 11].

Join indices were first developed by Valduriez to enhance the performance of join operations in relational databases [12]. Instead of computing spatial joins at query processing time, join indices are precomputed, stored as an index file and retrieved at the query processing time. Each record of the join index file contains a matched object identifier pair. Join indices reduce the number of I/O's needed and thus improve the performance of join operations.

As an extension to join indices for spatial database applications, Rotem [7] proposed a spatial join index structure which converts geometric computation for certain spatial relationship into a simple spatial join index

file. Join indices store object identifier pairs for those objects having such spatial relationship. Certain queries can be processed by retrieving spatial join indices rather than performing geometric algorithms. A spatial join index can be built based on a matching predicate, such as *intersect* and *contain*. Furthermore, queries related to certain distance can be processed by constructing spatial join indices based on ϵ -overlap (overlapping within a disk with a radius of ϵ) where ϵ is a fixed distance defined by a database designer.

Spatial join indexing is a promising approach for answering queries involving intersection, containment, etc. However, ϵ -overlap is a distance-fixed mapping. It is difficult to construct a large number of spatial join index files corresponding to different query distance values. Moreover, many queries are related to a distance range which forms a ring surrounding the object. It is impossible for a database designer to anticipate and enumerate all kinds of query ranges. Therefore, such spatial join indices may not be effective for queries involving various distances or distance ranges among objects. For example, when a range query is to find all the objects with a distance to a given object between 1000 and 2000 meters, the ϵ -overlap approach becomes difficult since its indexing structure can not express the ring distance constraint.

In this paper, we propose a flexible spatial join index facility for dynamic spatial range queries. The idea is to associate a distance measurement with each join index record so as to reduce or even to eliminate geometric computation at the retrieval time. By organizing index records into B+-trees, spatial range queries as well as other distance-related queries can be processed efficiently. Based on this basic distance-associated join index structure, two structured distance join indices, *ring-structured* and *hierarchical*, are proposed to enhance search performance in more sophisticated geometric environments. Ring-structured distance-associated join indices partition the join index file into several index files based on certain distance ranges. A query related to a given spatial range needs only to access those ring index

[†] The work was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grant A-3723 and a research grant from Centre for Systems Science of Simon Fraser University.

structures which overlap with the inquired range. Hierarchical distance-associated join indices resemble multiple-scaled maps where smaller scaled objects such as houses and buildings are associated with nearby objects such as highway intersections or major buildings. This method reduces the overall join index file size, and it can be used for hierarchically organized spatial environments. In general, distance-associated join index structures reduce or avoid geometric computation at query processing time by simply searching the index files. Therefore, they reduce the cost of I/O and CPU computation and facilitate spatial query processing. The precomputation of the distance information and the storage of them for efficient retrieval may substantially improve the performance of query processing. This is the motivation of the construction of distance-associated join indices.

The rest of the paper is organized as follows. In section 2, three types of distance-associated join indices are proposed and studied. In section 3, simulation results are presented and their performance is analyzed. In section 4, we discuss further improvement of spatial join indexing mechanism and summarize our study.

2. Distance-Associated Join Indices for Dynamic Spatial Range Search

Spatial joins are very common in databases which store images, pictures, maps and drawings. These joins are especially costly to perform, hence the use of spatial join indices can be valuable, if they can be created and maintained efficiently. Since many spatial joins are the joins among spatial objects within certain distance ranges, and other distance-related joins can be considered as special cases of spatial range joins, our design pays special attention on *spatial range queries* (the queries which inquire about certain spatial objects related to other spatial objects within a certain distance range).

As an example of range queries, consider a region map database scenario where schools, galleries and regional parks, etc. are marked as points or small regions. The following kinds of spatial range queries could be inquired frequently.

- (1) Given a location, find regional parks that are beyond 30 miles but are within 60 miles.
- (2) Find all gallery and school pairs which are within 1 mile.

The distance-related predicate of these queries can be abstracted into the following form,

$$D_{\min} < distance(A, B) \leq D_{\max},$$

where D_{\min} and D_{\max} are variables. Notice that $distance(A, B) \leq D_{\max}$ can be considered as a special case where $D_{\min} = 0$.

To facilitate spatial joins in complex environments, three kinds of distance-associated spatial join indices: *basic*, *ring-structured* and *hierarchical*, are proposed and studied in the following three subsections.

2.1. Basic Distance-Associated Join Indices

Basic distance-associated join index (*Basic DJI*, or *BDJI*) is an indexing mechanism which associates two spatial objects (i.e., their identifiers) with a piece of distance information. It optimizes distance-related queries by computing the distances between two static spatial objects at index construction time rather than at query processing time. Therefore, information about the distance between two spatial objects is available in the spatial join index without any geometric computation at query processing time.

2.1.1. Definition and Construction

Given two spatial object relations R_1 and R_2 , the basic DJI records are generated by coupling object identifier pairs in R_1 and R_2 respectively with the distance between the objects.

$$BDJI = \{ \langle o_i, o_j, d_{i,j} \rangle \mid o_i \in R_1 \wedge o_j \in R_2 \wedge d_{i,j} = distance(o_i, o_j) \}.$$

where $distance(o_i, o_j)$ is a function which takes the two object identifiers and returns the distance between objects.

Notice that $distance(o_i, o_j)$ is usually defined according to specific applications. For example, the distance between two buildings can be defined as the distance from the center of one building to the center of the other, the shortest distance between them, or their Mahattan distance by following their corresponding city streets, etc. We consider that the computation of distance may involve I/O and complex geometric computation and is thus a relatively expensive process. The distance attribute in the join index record can be used to directly answer queries about the distance between two spatial objects or answer spatial range queries by satisfying the distance constraints provided in the queries.

The basic distance-associated join indices (Basic DJI's) are constructed as follows. Join indices are sorted first by the first attribute, so that queries related to a specified object can be answered efficiently. A B+-tree is built on the primary index. Then records with the same first attribute value are sorted by the distance attribute. To speed up search for range queries, the secondary index is created based on the value of the distance attribute. The algorithms for creation, retrieval, and maintenance of basic DJI are presented below.

Algorithm-1. Creation of Basic DJI.

Input. Spatial object relations R_1 and R_2 .

Output. Construction of Basic DJI.

Method.

- (1) For each pair of spatial objects o_i and o_j , where $o_i \in R_1$ and $o_j \in R_2$, compute their distance and generate an index record $\langle o_i, o_j, d_{i,j} \rangle$.
- (2) Sort the DJI records by the first attribute, construct the primary index for the DJI.
- (3) For indices with the same value in the first attribute, Sort the records according to the distance attribute, and construct a secondary index. \square

To simplify our discussion, we assume that R_1 is identical to R_2 , that is, the DJI's are constructed to reflect the spatial distance relationship within the same set of spatial objects. It is straightforward to generalize the results to DJI's between two distinct sets of spatial objects.

To illustrate the algorithm, the indices for three simple spatial objects are demonstrated in Figure 1, in which indices are first sorted by the first attribute, that is, o_1, o_2 and o_3 . Records with the same first attribute value are then sorted by their distance value. For example, for the same first attribute o_1 , the second attribute o_3 is before o_2 because o_1 is closer to o_3 than to o_2 .

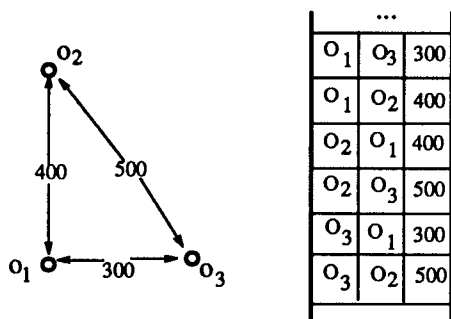


Figure 1. Indices for three spatial objects.

2.1.2. Retrieval

Suppose that the basic DJI's are constructed on N spatial objects. The following presents a retrieval algorithm for a typical spatial range query: *finding all the objects whose distance from a given object is between D_{min} and D_{max} .*

Algorithm-2. Data retrieval for a typical spatial range query.

Input. (i) an object id o_i , (ii) the lower bound of the range, D_{min} , and (iii) the upper bound of the range, D_{max} .

Output. Every object (its identifier) in the database whose distance from o_i is within the specified spatial range (D_{min}, D_{max}].

Method.

- (1) Search for object o_i using the primary index,
- (2) Search along the secondary index until the distance value reaches D_{min} ,
- (3) Read the leaf index records until the distance value is greater than D_{max} . \square

The first step and second step both take $O(\log N)$ I/O time where N is the number of objects. The I/O cost in the third step is proportional to number of results in the query divided by the number of index records stored in one data page.

Such an index structure also facilitates other kinds of queries:

- (1) the search for the closest spatial objects with the computation complexity $O(\log N)$;
- (2) the search for all the pairs between spatial objects satisfying a given distance constraint in $O(N \log N)$ time.

Example 1. (Range search using basic DJI). Given an object o_2 in Figure 1, the query is to find the objects within the distance range between 300 and 450. Figure 2 shows a portion of the B+-tree. The search proceeds as follows. Searching o_2 by the primary index, record R_1 is selected. Searching for 300 using the secondary index, we obtain record r_1 which satisfies the distance constraint. By following the linked list at the leaf-level, we record r_2 which is beyond D_{max} . Then the retrieval terminates with one resulting object o_1 . The search path is shown by the dark arrowed lines.

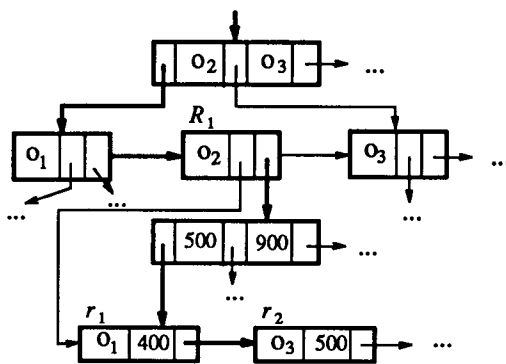


Figure 2. Processing a spatial range query using the Basic DJI.

2.1.3. The Update Algorithms

The above two algorithms have shown that the basic DJI's can be created and used quite efficiently. The following algorithm shows that it is also quite easy to maintain the indices on updates. Only the insertion algorithm is presented here, however, it is easy to work out the deletion algorithm following the similar approach.

Algorithm-3. Insertion of an object to the basic DJI.

Input. A new object with the identifier o_i .

Output. An updated basic DJI after o_i is inserted into the database.

Method.

- (1) For each spatial object o_j in the database, compute the basic DJI record $\langle o_i, o_j, d_{i,j} \rangle$. Symmetrically, we obtain $\langle o_j, o_i, d_{j,i} \rangle$;
- (2) Cluster the new index records with those whose first attribute being o_i and sort them by their distances;
- (3) Insert each record $\langle o_j, o_i, d_{j,i} \rangle$ into the existing DJI's using the B+-tree insertion algorithm;
- (4) Insert the set of new index records with the first attribute being o_i into the B+-tree. \square

The first step takes $O(N)$ time where N is the number of existing spatial objects in the database. The second step takes $O(N \log N)$ time to sort N records. The third step takes $O(N \log N)$ time to insert N records. Finally, the fourth step takes $O(\log N)$ to find the right place to insert the set of new index records. Therefore, the algorithm complexity is $O(N \log N)$. Similarly, the deletion of a spatial object from the database will also take $O(N \log N)$ time.

Since the basic DJI provides a reasonable cost for index maintenance as well, it is expected that it will be an interesting candidate to replace the run time geometric computation of distances between sets of objects.

However, since distance-associated join indices provide association among all the spatial objects with different distances, the total number of index records to be maintained in the basic DJI file will be the number of records in the cross-product set of spatial objects in the database. As the number of objects in the database increases, the size of the basic DJI file increases quadratically. It will be impractical to construct and store such a huge index file in a relatively large size of spatial database.

In the practical applications, most range searches are confined to the vicinity of a spatial object. It is natural to specify a cutting radius or a scope value for most spatial objects. For example, a fire station 60 miles away should be ruled out for being used for an emergency purpose. In

this case, 60 miles may be set as its cutting radius. Two objects are related if and only if the distance between them is within the specified cutting radius.

2.2. Ring-Structured Distance-Associated Join Indices

Following the same philosophy for reducing the size of distance-associated join index files, a ring-structured distance-associated join index (*Ring-structured DJI* or *RDJI*) can be constructed. The ring structure partitions one DJI file into several files based on different distance ranges. For example, the objects with the distance within 100 meters are partitioned into one ring, those with the distance between 100 to 500 meters are partitioned into the second ring, etc. For a query with a specified spatial range, search can be confined to only those ring-structured DJI files which overlap the specified range.

Different standards can be used as the criteria in the partition or creation of ring-structured DJI's, such as equal-distance, equal-area, progressively increased distance range, etc. For simplicity, we examine a ring structure partitioned by equal distance. Given a set of n equal distance radii, $r_0 = 0, r_1 = r, r_2 = 2 \times r, \dots, r_n = n \times r$, a set of RDJI files can be constructed which correspond to the rings specified by the radii.

$$RDJI_k = \{ \langle o_i, o_j, d_{i,j} \rangle \mid r_{k-1} < d_{i,j} \leq r_k \}, \text{ where } k = 1, \dots, n.$$

The index construction algorithm is similar to Algorithm-1. Figure 3(a) shows a set of concentric rings centered at o_1 with equal-distance radii: $r_1 = 10, r_2 = 20, r_3 = 30$. Figure 3(b) illustrates a portion of the ring-structured DJI files.

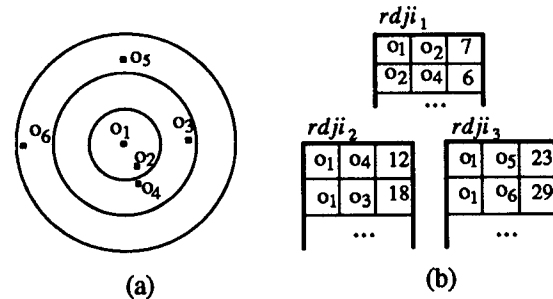


Figure 3. An example of ring-structured DJI.

For this example, if a spatial range query inquires spatial objects with the distance between 10 and 20, only the file $rdji_2$ is searched. Next, we present a range query algorithm on the ring-structured DJI.

Algorithm-5. Range query search using the ring-structured index structure.

Input. (i) an object id o_i ; (ii) the lower bound of the range, D_{\min} , and (iii) the upper bound of the range, D_{\max} .

Output. Spatial objects within the range.

Method.

- (1) Select the ring files which overlap with the query range. A ring index file should be searched if its lower and upper distance bound is within D_{\min} and D_{\max} , or D_{\min}/D_{\max} is within its lower and upper distance bound.
- (2) For each selected ring index file,
 - [1] Search for the inquired object id by the primary index.
 - [2] For rings covered by the query range, Collect all objects in the index files related to object id.
 - [3] For rings on the query boundaries
 - (i) Search along the secondary index until the distance value reaches D_{\min} , and
 - (ii) Read the leaf index records until the distance value is greater than D_{\max} . \square

The computational complexity of solving a spatial range query using the ring DJI structure should be the same as that using the basic DJI. However, since the ring structure partitions the basic DJI file into several smaller files, and the files which should be searched can be determined before database accessing if the distance or distance range values are provided in the query. Therefore, it is expected that the ring structure may improve the performance of query processing in comparison with the basic DJI.

The ring structure has obvious advantages over basic DJI when the spatial range in the query covers only one or a small number of rings. If the full range of a ring is completely within the query range, there is no need for the comparison with the values in the distance attribute since all the objects with the first object identifier in the ring satisfy the query. Moreover, if the query is to find all the pairs within a certain range which fully covers the range of the ring, all the object pairs in the ring are returned as the answers.

When the spatial range in a query involves many rings, primary index search is required for each ring. This may add overhead and complexity to memory and buffer management. Therefore, it is suggested to partition the distance-associated join indices wisely in the construction of ring DJI files. For example, one may choose the radius for the innermost ring to be 100 meters for those interested in neighborhood, and choose the second ring to be 100 to 1000 meters for those interested in shopping, schooling, bus-stops, etc. By doing so, frequent inquiries would likely to be involved in only one or a small number of rings.

2.3. Hierarchical Distance-Associated Join Indices

Since both basic DJI and ring-structured DJI store all the spatial object pairs in a spatial data relation, the size of the total index file(s) should be in proportion to the size of the cross-product of the relation on itself (although it stores only their keys rather than the full tuples). In most reasonably large spatial data relations, it is impractical and unnecessary to relate all the spatial objects. For example, it is rarely useful to relate one school building in one city to individual houses in another suburban city.

Hierarchical views are commonly taken in solving spatial problems in a complex world. When scheduling a flight from one continent to another, most small cities are ignored in the study. When driving a car to work, most individual houses are omitted in the calculation. Based on the similar point of view and assumption, spatial objects can be partitioned and classified correspondingly to fit into maps with different scales.

Analogous to multi-scaled maps, a *hierarchical distance-associated join index* (*Hierarchical DJI* or *HDJI*) can be constructed to organize spatial objects into different levels. Within one city block, it could be useful to construct distance-associated join indices to represent distances between individual houses and street intersections. In a larger scale, only highway intersections or major buildings in the city will be represented in the join indices. Queries about the distance between your house and your friend's in another suburban city can still be answered by referring more than one hierarchical join index files.

Suppose a spatial data relation consists of n interrelated object sets R_1, R_2, \dots, R_n , with different scalar scope values S_1, S_2, \dots, S_n respectively, where the scope value of S_i is an order of magnitude larger than that of S_{i-1} . For example, the distance between two houses is at an order of 10 meters but that between highway intersections is at an order of 1000 meters. An object in R_i is at a *higher level* than the one in R_j when $i > j$. Each object at level i has at least one parent object at level $i+1$. Object classes are constructed as follows.

$$C_n = R_n, \text{ and } C_i = R_i \cup R_{i+1}, \text{ where } i = 1, \dots, n-1.$$

Distance-associated join indices are constructed on the object classes with different scalar scope values S_1, S_2, \dots, S_n respectively. The join index on C_k is constructed based on the following formula,

$$HDJI_k = \{ \langle o_i, o_j, d_{i,j} \rangle \mid o_i \in C_k \wedge o_j \in C_k \wedge d_{i,j} \leq S_k \},$$

where $k = 1, \dots, n$.

A range query is to find all the objects within a certain distance range, D_{\min} and D_{\max} , from a given object o_i . In most cases, solving a range query using the hierarchical DJI needs the search on the hierarchical DJIs by climbing

up and stepping down the hierarchy. Thus a search can be partitioned into two phases: ascending phase and descending phase. First, find the class level i such that o_i is in R_i . Then, collect all the objects whose distance from o_i is between D_{\min} and D_{\max} . This is accomplished by joining all the partitioned index files whose objects are located possibly within the range. Suppose the current scope is s_1 . If $s_1 < D_{\max}$, climb up the hierarchy by joining the upper level spatial join index file. In the ascending phase, an object with a distance from o_i less than D_{\min} subtracting lower level scope value will not be included in the intermediate relation T for later descending because its descendants will always have distances less than D_{\min} . However, it should be included in the temporary relation $Temp$ for further ascent since its further ascent may generate satisfiable answers. In the descending phase, an object with a distance from o_i less than D_{\min} subtracting lower level scope value or greater than D_{\max} will not be collected in T since its descendants cannot satisfy the query. At the end, only the objects whose distance from o_i are between D_{\min} and D_{\max} are included in the result relation. Notice that newly generated index with a shorter distance will replace index of the same object pair with a longer path distance. In other words, the object distance stored in every intermediate or final result relation is measured by the shortest path distance.

Algorithm-6. Range query processing using the hierarchical DJI.

Input. (i) an object o_i , and (ii) the spatial range bounds D_{\min} and D_{\max} .

Output. All the spatial objects o_j such that $D_{\min} < distance(o_i, o_j) \leq D_{\max}$.

Method.

- (1) [Initialization]
 - Find $level$ such that $o_i \in R_{level}$.
 - $level_i := level$.
 - $Temp := \{ \langle o_i, o_j, d_{i,j} \rangle \mid \langle o_i, o_j, d_{i,j} \rangle \in HDJI_{level} \wedge d_{i,j} < D_{\max} \}$.
 - $T := Temp - \{ \langle o_i, o_j, d_{i,j} \rangle \mid d_{i,j} + S_{level-1} < D_{\min} \}$.
 - (assuming $S_0 = 0$)
- (2) [Ascending phase]
 - While $S_{level} < D_{\max}$ Do {
 - $level := level + 1$.
 - $Temp := \{ \langle o_i, o_k, d_{i,k} \rangle \mid \langle o_i, o_j, d_{i,j} \rangle \in Temp \wedge \langle o_j, o_k, d_{j,k} \rangle \in HDJI_{level} \wedge d_{i,k} = d_{i,j} + d_{j,k} \wedge d_{i,k} < D_{\max} \}$.
 - $T := T \cup (Temp - \{ \langle o_i, o_j, d_{i,j} \rangle \mid d_{i,j} + S_{level-1} < D_{\min} \})$.
 - }
- (3) [Descending phase]
 - While $level > level_i$ Do {

$level := level - 1$.

$$T := T \cup \{ \langle o_i, o_k, d_{i,k} \rangle \mid \langle o_i, o_j, d_{i,j} \rangle \in T \wedge \langle o_j, o_k, d_{j,k} \rangle \in HDJI_{level} \wedge d_{i,k} = d_{i,j} + d_{j,k} \wedge d_{i,k} + S_{level-1} > D_{\min} \wedge d_{i,k} < D_{\max} \}$$

- (4) Return $\{ o_j \mid \langle o_i, o_j, d_{i,j} \rangle \in T \wedge d_{i,j} > D_{\min} \}$. \square

Example-2. (A range query on the hierarchical index). Figure 4 shows a two-level hierarchical DJI where $S_1 = 1000$ and $S_2 = 10$ for a simple object setting. Given an inquired object o_{22} , find all the objects whose distance from o_{22} is between 4 and 450.

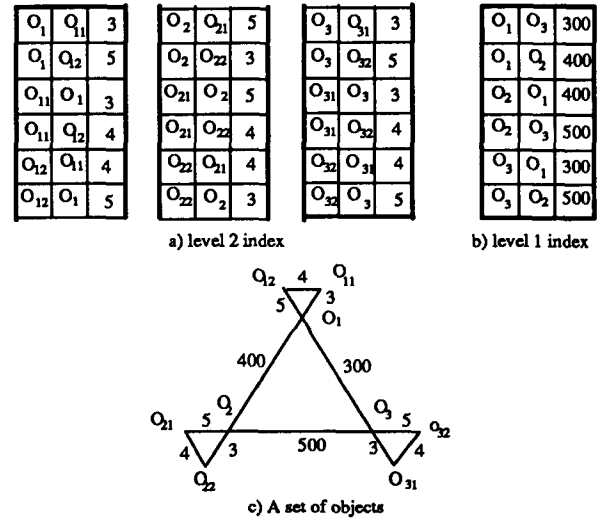


Figure 4. A simple two-level DJI and the index graph.

The search sequence is as follows.

- (1) First, search for object o_{22} in the HDJI.
- (2) By hierarchy ascent, we have $T = \{ \langle o_{22}, o_{21}, 4 \rangle, \langle o_{22}, o_1, 403 \rangle \}$.
- (3) By hierarchy descent, we have $T = \{ \langle o_{22}, o_{21}, 4 \rangle, \langle o_{22}, o_1, 403 \rangle, \langle o_{22}, o_{11}, 406 \rangle, \langle o_{22}, o_{12}, 408 \rangle \}$.

At the first iteration in the ascending phase, $\langle o_{22}, o_2, 3 \rangle$ is in temporary relation $Temp$ but not in the relation T because the sum of 3 and 0 is less than the lower bound 4. It is used as a bridge for the shortest distance to object o_1 . The records related to o_3 are not generated because its accumulated distance to o_{22} has exceeded the upper bound of the range. Therefore, the answer to the query is $\{ o_1, o_{11}, o_{12}, o_{21} \}$.

Interestingly, hierarchical DJI is quite efficient for finding the shortest distance between two spatial objects, for example, the shortest path from a person's house to his/her friend's house. The idea of the algorithm is as follows. Start from the leaf level of the hierarchy. If object o_i and o_j are directly related, return the distance

registered. Otherwise, climb up the hierarchy by joining the HDJI file at one level higher than the current level until they both reach a common node. The distances involved in the search path are accumulated along the climbing and the sum is reported as the distance between the two objects. The search path can be returned when inquired.

3. Analysis and Simulation Results

Three kinds of distance-associated join indices are proposed in the previous section. It is important to study their performance in a relatively large spatial database environment. In this section, we present our simulation results and analyze the results in order to compare the performance among four indexing schemes: (1) NDJI: *no distance-associated join indices* are used in the processing, that is, only regular join indexing structures are used and the distance information is computed at the query processing time using geometric operators; (2) BDJI: *Basic distance-associated join indices* are used; (3) RDJI: *ring-structured distance-associated join indices* are used; and (4) HDJI: *hierarchical distance-associated join indices* are used in the processing.

3.1. Analytical Model

An analytical model is constructed to compare the performance of different schemes. The following parameters are used in our analysis.

Parameter	Explanation
N	number of objects
N_i	average number of indices per page
N_b	maximum B-tree branches at any node
N_g	average number of geo-objects per page
N_{bf}	number of buffer pages
C_{io}	cost for one I/O operation
C_{dist}	cost for computing distance
C_{comp}	cost for one comparison operation

3.1.1. Storage Requirement

First, we analyze how much storage space to be used in the processing.

- (1) NDJI: It requires no space for the join index.
- (2) BDJI: Let S be the scope distance for the distance-associated join index method and a maximum object density D over the area. The BDJI file page size in page should be as follows.

$$N \times D \times S^2 \times \pi / N_i$$

When S is small, the size of the index file grows linearly with respect of number of objects in the file.

- (3) RDJI: Given n rings, a file corresponding to a individual ring has a size $1/n$ of that of the basic DJI in average. The ring-structured DJI requires the same order amount of space as BDJI in total.
- (4) HDJI: Given a leaf level scope value S_1 , leaf level index size is a dominating $c \times N \times S_1^2 \times D \times \pi / N_i$ pages, where c is a constant overlapping factor. The size of the hierarchical DJI is more manageable than that without the hierarchy, i.e., N^2 .

3.1.2. Processing Cost

Range query is chosen as an example for analysis. All pair query processing cost curves will be presented later. The total cost is the sum of CPU cost which is time consumed operations on data, such as comparison and I/O cost which is take for fetching data from secondary storage to the main memory.

- (1) NDJI: Two objects have to be in the memory at the same time for computing the distance. When number of buffer pages is smaller than number of the data pages, total number of the IO is greater than number of the data pages. In fact, let $N_p = N / N_g$, i.e. the number of pages of geometric objects and apply the *most-recently-used* page replacement strategy, number of pages involved is

$$N_{io} = \frac{N_p \times (N_p + N_{bf})}{2 \times N_{bf}}$$

For example, with $N_p = 100$ and $N_{bf} = 10$, the total number of I/O is 550.

$$\text{CPU cost} = N \times (N - 1) \times (C_{dist} + C_{comp}).$$

- (2) BDJI: With the distance-associated join index where related objects are paired up, query processing never needs to read the same page into the buffers twice for single spatial join operation. Retrieval requires $O(\log N)$ I/O for primary index search and the secondary index search respectively. The sequential reading cost is number of results divided by N_i which is usually a very small number.

$$\text{CPU cost} = 2 \times \log N \times C_{comp}$$

where the base of the logarithm function is N_b .

- (3) RDJI: If the query involves N_r rings, the ring method takes $N_r \times \log(N)$ I/O for primary index search and $O(\log(N / N_r))$ for secondary index search in average. Generally speaking, the ring-structured DJI takes more I/O than the basic DJI. CPU cost for a range query on the ring-structured DJI is $2 \times \log(N / N_r) \times C_{comp}$, which is usually a smaller number than that of BDJI.
- (4) HDJI: $O(\log N)$ I/Os are needed for searching the leaf level primary index. The maximum number of

join iteration is twice as the number of the hierarchy levels, which usually is a small number. Because object connections are established only at the consecutive levels, we can assume that number of higher level objects related to any object is bounded by a constant. Hence, each join iteration takes $O(\log N)$. Therefore, the overall complexity is $O(\log N)$.

In the query for the distance between two given objects, object search by primary index takes $O(\log N)$ at the leaf level. The cost for hierarchy climbing is at most linear to the number of the hierarchy levels.

Overall, by using distance-associated join indices, object retrieval for spatial range queries takes logarithmic time to the number of objects in the setting.

3.2. Simulation Results

The simulation is performed on the SPARC-workstation under the UNIX system. The simulation program is written in C. A set of randomly generated points are used for simulation. The simulation is performed for two types of queries: (i) range queries, and (ii) queries for all pairs of the objects with a specified distance constraint. In Figure 5(a), given a fixed number (4000) of objects, the curves represent the processing cost as the query range increases for different methods respectively. In Figure 5(b), the curves illustrate the relationship between the processing cost and the number of objects in the spatial database. A three-level hierarchy is built for the simulation where the scope values are specified as $S_1 = 25000$, $S_2 = 2500$ and $S_3 = 250$, respectively. Some other simulation parameter values are set as follows:

$$N_{bf} = 40, \text{ ring radii } r_k = k \times 2000, k = 1, \dots, 6$$

$$N_g = 50, N_b = 50, C_{io} = 4000, C_{dist} = 500, C_{comp} = 1.$$

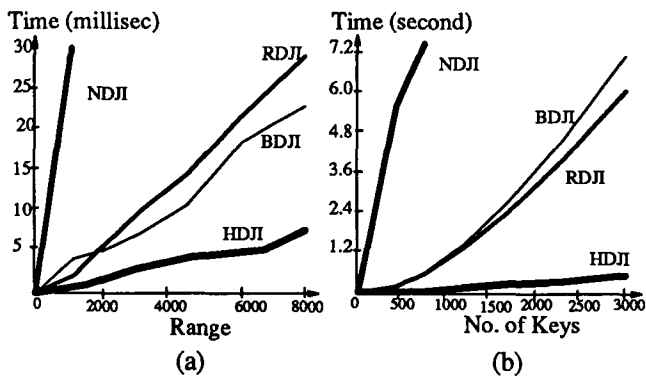


Figure 5. Cost curves for a) spatial range query, and b) all-pairs query.

The curves show the effectiveness of the distance-associated join indexing mechanism for reducing query processing cost in spatial queries.

3.3. Analysis of Simulation Results

As shown by the simulation results, the distance-associated join index improves the range query performance significantly. The three methods have advantages in different situations. The basic DJI is simple and derives reasonably good performance when the world is relatively small and simple. However, when the size of the data relation grows, the performance of the BDJI degrades.

The ring-structured DJI decomposes one BDJI file into several ring index files. For queries relevant to one particular ring, the ring structure reduces the search space and therefore enhances the performance. For range queries, ring-structured DJI reduces search cost when the range is confined to a small number of rings. When the range expands, that is, when it requires to reference many ring structures, ring DJI structure performs a little weaker than basic DJI structure. For all pair queries (queries inquiring all the pairs of the spatial objects satisfying a distance constraint), ring structure performs slightly better than basic DJI because less comparison operations are required in the ring structure.

Since the hierarchical indexing structure substantially reduces the number of index records which must be directly associated in the database, it substantially reduces the storage space and hence the accessing cost in a relatively large database. Although spatial objects should reference their higher level spatial objects for some spatial queries, the cost of accessing several higher level reference points will only increase the accessing cost linearly. Since such an organization may substantially reduce the storage and accessing cost, hierarchical DJI has the best performance in most cases.

4. Discussion and Conclusion

We have developed three distance-associated join indexing mechanisms for efficient spatial search. Distance-associated join indices is an extension to spatial join indices [7] by associating distance information with the join index records. By association of distance information with join indices, geometric computation of spatial objects at the query processing time can be reduced or eliminated by performing such computations at the index construction time and registering them in the join index records.

Distance-associated join index mechanism can be extended to other kinds of queries. Many queries are not only relevant to distance computation, but also relevant to the computation of directions or orientations between two

spatial objects. If such queries are frequently encountered in the system, a further extension to the distance-associated join indices is to associate one more piece of information *direction*. In such cases, an index record could be in the form of $\langle o_i, o_j, \text{distance}, \text{angle} \rangle$ where *angle* is, for example, the angle formed between the vector from o_i to o_j and *x*-axis. Similar join index construction mechanisms can be applied to such distance-and-direction-associated join indices. A query such as "finding all the restaurants within 1000 meters located to the east of the conference center" can be answered efficiently. Furthermore, spatial range queries, direction range queries, and queries asking about the distances and directions from certain locations can be answered by examining such distance-and-direction-associated join indices.

It is an interesting idea to precompute some geometric/geographic information in the static environment and store them in data relation or some index structures for later efficient retrieval. However, such an idea should not be pushed to extreme. First, in the world where the locations of objects changing from time to time, it is expensive to recompute such indices and keep all the associations consistent and up-to-date. Moreover, there are many geometric operations which may involve combinations of different objects. There are so many different combinations and there is no way to explore and store all such combinations. For example, a geometric constructor *union* may involve different combinations and create new geometric objects. There is no way to explore all such combinations and precompute them before query processing.

In this paper, we studied distance-associated join index mechanism and developed three kinds of distance-associated join indices for dynamic spatial range query optimization and the optimization of other spatial queries. Each kind of distance-associated structure has its own application domain. The basic DJI is concise and efficient in a simple and small environment. The ring-structured DJI performs well when the query references only a small spatial range in a moderate sized database. By adjusting ring radii, reduction of the size of the file to be processed and improvement of I/O cost can be both achieved. The hierarchical DJI reforms best among the three in a complex and large spatial database. For a city map database, the hierarchical DJI can facilitate the finding of the shortest path between objects naturally with a reasonable processing cost. All these three join index mechanisms are simple, flexible, and easy to create and maintain. Our preliminary simulation-based performance study demonstrates the high promise of this approach. We are currently implementing the distance-associated join indexing mechanism in a relatively large database. More results will be reported in the future.

References

1. D. Greene, An Implementation and Performance Analysis of Spatial Data Access Method, *Proc. 5th Int. Conf. Data Engineering*, Los Angeles, CA, Feb. 1989, 606-615.
2. O. Gunther, The Design of Cell Tree: An Object-Oriented Structure for Geometric Databases, *Proc. 5th Int. Conf. Data Engineering*, Los Angeles, CA, Feb. 1989, 598-605.
3. Henrich, H. Six and P. Widmayer, The LSD tree: spatial access to multidimensional point and non-point objects, *Proc. 15th Int. Conf. Very Large Data Bases*, Amsterdam, Aug. 1989, 45-53.
4. J. Nievergelt, H. Hinterberger and K. C. Sevcik, The Grid File: An Adaptable, Symmetric Multikey File Structure, *ACM Trans. Database Systems*, 9(1), 1984, 38-71.
5. J. A. Orenstein and F. A. Manola, PROBE Spatial Data Modeling and Query Processing in an Image Database Application, *IEEE Trans. Software Engineering*, 14(5), May 1988, 611-629.
6. J. T. Robinson, The K-D-B tree: A Search Structure for Large Multidimensional Dynamic Indexes, *Proc. 1981 ACM-SIGMOD Int. Conf. Management of Data*, Ann Arbor, MI, April 1981, 10-18.
7. D. Rotem, Spatial Join Indices, *Proc. 7th Conf. Data Engineering*, Kobe, Japan, 1991, 500-509.
8. H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1990.
9. B. Seeger and H. Kriegel, Techniques for Design and Implementation of Efficient Spatial Access Methods, *Proc. 13th Int. Conf. Very Large Data Bases*, Brighton, England, 1988, 360-371.
10. T. Sellis, N. Roussopoulos and C. Faloutsos, The R+-Tree: A Dynamic Index for Multi-Dimensional Objects, *Proc. 13th Int. Conf. Very Large Data Bases*, Brighton, England, 1987, 3-11.
11. H. Six and P. Widmayer, Spatial Searching in Geometric Databases, *Proc. 4th Int. Conf. Data Engineering*, Los Angeles, CA, February 1988, 496-503.
12. P. Valduriez, Join Indices, *ACM Trans. Database System Vol 12, No 2*, (June 1987), 218-246.