Processing Distance Join Queries with Constraints

Apostolos N. Papadopoulos Alexandros Nanopoulos Yannis Manolopoulos
Data Engineering Research Lab., Department of Informatics
Aristotle University, Thessaloniki 54124, Greece
email: {*apostol,alex,manolopo*}*@delab.csd.auth.gr*

Abstract

Distance-join queries are used in many modern applications, such as spatial databases, spatiotemporal databases, and data mining. One of the most common distance-join queries is the closest-pair query. Given two datasets $\mathcal{D}_{\mathcal{A}}$ and $\mathcal{D}_{\mathcal{B}}$ the closest-pair query (CPQ) retrieves the pair (a,b), where $a \in \mathcal{D}_{\mathcal{A}}$ and $b \in \mathcal{D}_{\mathcal{B}}$, having the smallest distance between all pairs of objects. An extension to this problem is to generate the k closest pairs of objects (k-CPQ). In several cases spatial constraints are applied, and object pairs that are retrieved must also satisfy these constraints. Although the application of spatial constraints seems natural towards a more focused search, only recently they have been studied for the CPQ problem with the restriction that $\mathcal{D}_{\mathcal{A}} = \mathcal{D}_{\mathcal{B}}$. In this work, we focus on constrained closest-pair queries (CCPQ), between two distinct datasets $\mathcal{D}_{\mathcal{A}}$ and $\mathcal{D}_{\mathcal{B}}$, where objects from $\mathcal{D}_{\mathcal{A}}$ must be enclosed by a spatial region R. Several algorithms are presented and evaluated using real-life and synthetic datasets. Among them, a heap-based method enhanced with batch capabilities outperforms the other approaches as it is demonstrated by an extensive performance evaluation.

Keywords: *spatial data, closest-pair queries, spatial constraints*

1 Introduction

Research in spatial and spatiotemporal databases is very active in the last twenty years. The literature is rich in efficient access methods, query processing techniques, cost models and query languages, providing the necessary components to build high quality systems. The majority of research efforts aiming at efficient query processing in spatial and spatiotemporal databases, concentrated in the following significant query types:

- *Range Query*: is the most common topological query. A query area R is given and all objects that intersect or are contained in R are requested.
- *k Nearest-Neighbor Query*: Given a query point P and a positive integer k, the query returns the k objects that are closer to P, based on a distance metric (e.g., Euclidean distance).

- *Spatial Join Query*: It is used to determine pairs of spatial objects that satisfy a particular property. Given two spatial datasets $\mathcal{D}_\mathcal{A}$ and $\mathcal{D}_\mathcal{B}$ and a predicate θ, the output of the spatial join query is a set of pairs o_a,o_b such that $o_a \in \mathcal{D}_\mathcal{A}$, $o_b \in \mathcal{D}_\mathcal{B}$ and $\theta(o_a, o_b)$ is true.

- k *Closest-Pair Query* (k-CPQ): It is a combination of spatial join and nearest neighbor queries. Given two spatial datasets $\mathcal{D}_\mathcal{A}$ and $\mathcal{D}_\mathcal{B}$, the output of a k closest-pairs query is composed of k pairs o_a,o_b such that $o_a \in \mathcal{D}_\mathcal{A}$, $o_b \in \mathcal{D}_\mathcal{B}$. These k pair-wise distances are the smallest amongst all possible object pairs.

Spatial joins and closest-pair queries require significant computation effort and many more I/O operations than simpler queries like range and nearest neighbors. Moreover, queries involving more than one datasets are very frequent in real applications, and therefore, special attention has been given by the research community [12, 5, 6, 15, 18, 3].

In this study, we focus on the k-Semi-Closest-Pair Query (k-SCPQ), and more specifically, on an interesting variation which is derived by applying spatial constraints in the objects of the first dataset. We term this query k-Constrained-Semi-Closest-Pair Query (k-CSCPQ). In the k-SCPQ query we require k object pairs (o_a, o_b) with $o_a \in \mathcal{D}_\mathcal{A}$ and $o_b \in \mathcal{D}_\mathcal{B}$ having the smallest distances between datasets $\mathcal{D}_\mathcal{A}$ and $\mathcal{D}_\mathcal{B}$ such that each object o_a appears at most once in the final result. In the k-CSCPQ query, an additional spatial constraint is applied, requiring that each object $o_a \in \mathcal{D}_\mathcal{A}$ that appears in the final result must be enclosed by a spatial region R[1]. An example is given in Figure 1, illustrating the results of the aforementioned CPQ variations for $k = 2$.

Distance-based queries, such as nearest-neighbor and closest-pair queries, play a very important role in spatial and spatiotemporal databases. Apart from the fact that these queries compose an important family of queries on their own, they can be used as fundamental building blocks for more complex operations, such as data mining algorithms. Several data mining tasks require the combination of two datasets, to draw conclusions. A clustering algorithm based on closest pairs has been proposed in [13]. In [2, 3] the authors study applications of the k-NN join operation to knowledge discovery, which is a direct extension of the k-Semi-Closest-Pair query. More specifically, the authors discuss the application of k-NN join to clustering, classification and sampling tasks in data mining operations, and they illustrate how these tasks can be performed more efficiently. In [20] it is reported that the k-NN join can also be used to improve the performance of LOF algorithm, which is used for

[1]For the rest of the study we assume that both datasets $\mathcal{D}_\mathcal{A}$ and $\mathcal{D}_\mathcal{B}$ contain multidimensional points. The methods are also applicable for non-point objects.

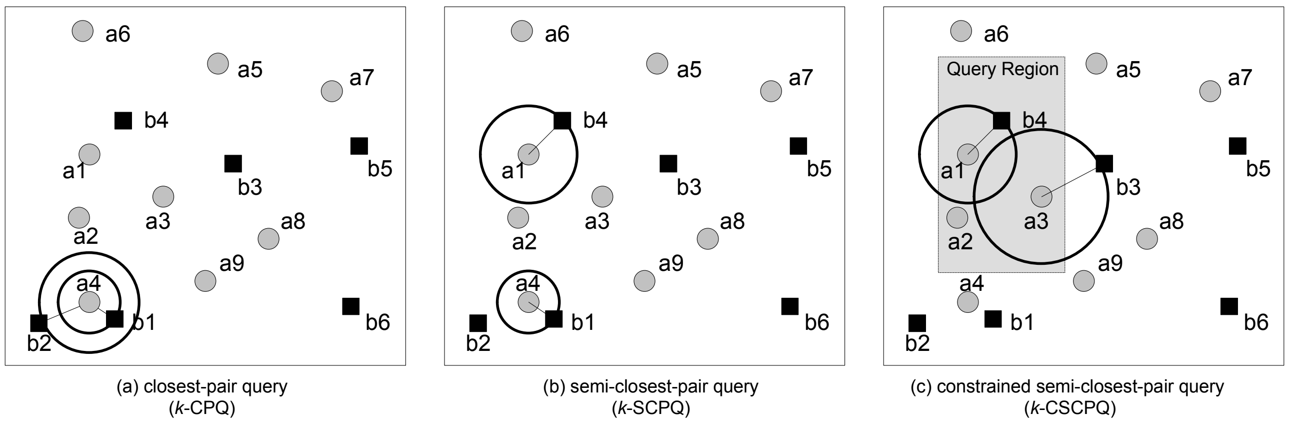


Figure 1: Results of CPQ variations for k=2.

outlier detection in a single dataset [4], and also to improve the performance of the Chameleon clustering algorithm [11]. The importance of dynamic closest-pair queries to hierarchical clustering has been studied in [8]. In the same paper the authors discuss the application of CPQ to other domains such as the Traveling Salesman Problem (TSP), non-hierarchical clustering, and greedy matching, to name a few.

Taking into consideration the significance of distance-based queries in several disciplines, in this work we focus on the Semi-Closest-Pair query with spatial constraints and study efficient algorithms for its computation. The motivation behind the current study is the fact that in many realistic cases the user focuses on a portion of the dataspace rather than in the whole dataspace. Although this sounds natural, specifically for large dataspaces and large populations, there is limited research work towards constrained spatial query processing. Moreover, spatial constraints may be applied implicitly by the system as a result of user query. For example, consider the query: "Find the three closest parks from all hotels located at the center of the city". The hotels located at the city center are usually enclosed by a polygonal region which defines the center of the city. Finally, many complex algorithms first perform a partitioning of the dataspace into cells, and then operate in each cell separately. Therefore, our methods can be used as of-the-self components for more complex operations, to speed up specific algorithmic steps.

The rest of the article is organized as follows. Section 2 presents the appropriate background, the related work and the main contributions of the paper. The query processing algorithms are presented and studied in Section 3. Section 4 contains the performance evaluation results, whereas Section 5 concludes the work and motivates for further research in the area.

2 Problem Definition and Related Work

Let $\mathcal{D}_\mathcal{A}$ and $\mathcal{D}_\mathcal{B}$ be two datasets of multi-dimensional points, each indexed by means of a spatial access method. We assume that the R*-tree [1] is used to index each dataset, although other variations could be applied equally well. Dataset $\mathcal{D}_\mathcal{A}$ is called the *primary dataset*, whereas dataset $\mathcal{D}_\mathcal{B}$ is called the *reference dataset*. We are interested in determining the k objects from $\mathcal{D}_\mathcal{A}$ that are closer to objects from the reference dataset $\mathcal{D}_\mathcal{B}$, under the constraint that all points from $\mathcal{D}_\mathcal{A}$ that are part of the answer must be enclosed by a spatial region R_q. If the number of objects from $\mathcal{D}_\mathcal{A}$ contained in R is less than k, then all objects are reported, ranked by their NN distance to the reference dataset $\mathcal{D}_\mathcal{B}$. For simplicity and clarity we assume that R_q is a rectangular region, although arbitrary query regions can be used as well.

The first method towards processing of constrained closest-pair queries has been proposed in [14], where it is assumed that $\mathcal{D}_\mathcal{A}$=$\mathcal{D}_\mathcal{B}$. Moreover, the authors assume that in order for a pair (o_1, o_2) to be part of the answer, both o_1 and o_2 must be enclosed by the query region R. In order to facilitate efficient query processing, the R-tree is used to index the dataset. The proposed method augments the R-tree nodes with auxiliary information concerning the closest pair of objects that resides in each tree branch. This information is adjusted accordingly during insertions and deletions. Performance evaluation results have shown that the proposed technique outperforms by factors existing techniques based on closest pairs. This method can not be applied in our case, since we assume that datasets $\mathcal{D}_\mathcal{A}$ and $\mathcal{D}_\mathcal{B}$ are distinct. This method can only be applied if for every pair of datasets we maintain a different index structure, which is not considered a feasible approach.

In [17] the authors study the processing of closest-pair queries by applying cardinality constraints on the result. For example, the query "determine objects from $\mathcal{D}_\mathcal{A}$ such that they are close to at least 5 objects from $\mathcal{D}_\mathcal{B}$", involves a distance join (closest-pair) and a cardinality constraint on the result. However, we are interested in applying spatial constraints on the objects of $\mathcal{D}_\mathcal{A}$.

Research closely related to ours include the work in [21] where the All-Semi-Closest-Pair query is addressed, under the term All-Nearest-Neighbors. The authors propose a method to compute the nearest neighbor of each point in dataset $\mathcal{D}_\mathcal{A}$, with respect to dataset $\mathcal{D}_\mathcal{B}$. They also provide a solution in the case where there are no available indexing mechanisms for the two datasets. The fundamental characteristics of these methods is the application of batching operations, aiming at reduced processing costs. Although the proposed methods are focused on evaluating the nearest-neighbor for every object in $\mathcal{D}_{\mathcal{A}}$, they can be modified towards reporting the best k answers, under spatial constraints. The details of the algorithms are given in the subsequent section.

Methods proposed for Closest-Pair queries [10, 6, 7] can be used in our case by applying the necessary modifications in order to: 1) process Semi-Closest-Pair queries and 2) support spatial constraints. Algorithms for Closest-Pair queries are either recursive or iterative and work by synchronized traversals of the two index structures. Performance is improved by applying plane-sweeping techniques and bidirectional node expansion [16]. The details of the Closest-Pair algorithm, which is used for comparison purposes, are given in the subsequent section.

Table 1 summarizes the symbols and the corresponding definitions that are used for the rest of the study.

Symbol	**Description**
$\mathcal{D}_{\mathcal{A}}$, $\mathcal{D}_{\mathcal{B}}$	the two datasets (primary and reference)
a_i, b_j	an object of $\mathcal{D}_{\mathcal{A}}$, an object of $\mathcal{D}_{\mathcal{B}}$
$T_{\mathcal{A}}$, $T_{\mathcal{B}}$	the two R-trees (primary and reference)
N_A, N_B	a node of $T_{\mathcal{A}}$, a node of $T_{\mathcal{B}}$
$N_A[i]$, $N_B[j]$	the i-th entry of N_A, the j-th entry of N_B
R	query region
$\|R\|$	objects from $\mathcal{D}_{\mathcal{A}}$ contained in query region R
$mindist(R_1, R_2)$	minimum distance between rectangles R_1 and R_2
$dist(a_i, b_j)$	distance between objects a_i and b_j (e.g., Euclidean)
k	number of answers required
d_k	the k-th best distance determined so far

Table 1: Basic notations used throughout the study.

3 Adapting Existing Algorithms

In this section we describe the way that existing algorithms can be used in order to answer k-CSCPQ queries. Three algorithms are investigated, namely the naive algorithm MNN, the batching algorithm for All-Nearest-Neighbor queries and the Semi-Closest-Pair algorithm which is adapted from the Closest-Pair algorithm. The proposed approach is studied in a subsequent section.

3.1 The Naive Algorithm (MNN)

The simplest approach for k-CSCPQ query processing is to apply the Multiple Nearest-Neighbor algorithm (MNN), which has been used in [21] as a straightforward solution to the All-NN problem. This method traverses the $T_{\mathcal{A}}$ recursively, by following branches with respect to the Hilbert values of MBR centroids, and only if the corresponding MBR intersects the query region R_q. When a leaf node is reached, a 1-NN query to $T_{\mathcal{B}}$ is executed for every object o_i contained in the leaf, to determine if o_i contributes to the best k answers. The k best answers are organized by means of a maxheap structure, and therefore the k-th best distance is available at the top of the heap. If the distance of a_i to its nearest neighbor is less than the top of the heap, then a_i (and the corresponding distance) is inserted into the maxheap. This process is continued until no branches of $T_{\mathcal{A}}$ are available for investigation.

It is evident that this method requires the execution of a large number of 1-NN queries, and therefore the computation cost is high due to excessive distance computations. However, it is expected that locality preservation will be high, since the pages required for each 1-NN will be close in space. This behavior of the MNN algorithm has been investigated in [21] for the All-NN problem. Here, we include MNN algorithm in our study for completeness and in order to test its behavior in k-CSCPQ query processing.

In summary, the MNN algorithm can be used for k-CSCPQ query processing, provided that a node is inspected only if its MBR intersects the query region R. Figure 2 presents the outline of the MNN algorithm.

```
Algorithm MNN ($N_A$, $T_B$, $k$, $R$)
Input: $N_A$ is a node of $T_A$ (initially is the root)
$T_{\mathcal{B}}$ is the reference tree,
$k$ is the requested number of closest pairs,
$R$ is the query region
Output: maxheap is the heap of the best $k$ answers

1.    if ($N_A$ is LEAF) then
2.        discard entries of $N_A$ that are not contained in $R$;
3.        foreach of the remaining entries $N_A[i]$ execute an 1-NN query on reference tree $T_{\mathcal{B}}$;
4.        update maxheap if necessary;
5.    elsif ($N_A$ is INTERNAL) then
6.        check which entries $N_A[i]$ intersect $R$;
7.        sort these entries wrt MBR centroid Hilbert value;
8.        for each entry call MNN recursively;
9.    endif
```

Figure 2: Outline of MNN algorithm.

3.2 The Batch Nearest-Neighbor Algorithm (BNN)

The BNN algorithm has been proposed in [21] as a solution for the All-NN problem, when the two datasets under consideration are indexed. The algorithm recursively visits nodes of $T_{\mathcal{A}}$ according to the Hilbert values of the MBR centroids. Evidently, a node will be inspected only if its MBR is intersected by the query region R_q. When a leaf node of $T_{\mathcal{A}}$ is reached, the algorithm applies a heuristic in order to decide if a batch query will be executed to $T_{\mathcal{B}}$ or not.

Objects of $\mathcal{D}_{\mathcal{A}}$ are collected and organized into groups as follows: Let G be an empty set of collected objects. When a leaf node of $T_{\mathcal{A}}$ is reached, its objects are sorted by means of their Hilbert values. Then, these objects are inserted one-by-one into G, until a condition is satisfied. The condition states that the next object will be inserted into G if the area of the MBR of G is less than or equal to $avgLeafArea_B$, the average area of the leaf nodes of $T_{\mathcal{B}}$ that will be accessed during a NN search. In order to determine $avgLeafArea_B$, a 1-NN query is executed for the first object that is inserted into G. If the area of the MBR of G exceeds $avgLeafArea_B$, then a batch query is executed. Otherwise, the object is inserted into G. If all objects in the current leaf have been inspected and the area criterion still holds, then the grouping continues with the next visited leaf node of $T_{\mathcal{A}}$. This grouping technique adapts the size of each group according to the sparseness of dataset $\mathcal{D}_{\mathcal{A}}$. In some cases, the objects of a leaf node will be split in several groups, whereas in other cases a group will be formed by objects of several leaf nodes.

It has been shown in [21] that this method achieves significant performance improvements in comparison to the MNN algorithm for the All-NN problem. Due to batching, the number of distance computations is decreased significantly in comparison to the MNN algorithm.

In summary, the BNN algorithm can be used for k-CSCPQ query processing, provided that a node of the primary R-tree $T_{\mathcal{A}}$ is inspected only if its MBR intersects the query region R. The outline of the BNN algorithm is illustrated in Figure 3.

3.3 The Semi Closest-Pair Algorithm (SCP)

Algorithms that process k-CPQ queries can be adapted in order to answer k-CSCPQ queries. In this study, we consider a heap-based algorithm proposed in [7], enhanced with plane-sweeping optimizations [16]. Moreover, the algorithm is enhanced with batching capabilities, towards reduced processing costs. Algorithm SCP is based on a bidirectional expansion of internal nodes which has been proposed in

```
Algorithm BNN ($N_A$, $T_B$, $k$, $R$)
Input: $N_A$ is a node of $T_A$ (initially is the root)
$T_B$ is the reference tree,
$k$ is the requested number of closest pairs,
$R$ is the query region
Output: $maxheap$ is the heap of the best $k$ answers

1.   if ($N_A$ is LEAF) then
2.       discard entries of $N_A$ that are not contained in $R$;
3.       foreach of the remaining entries $N_A[i]$, insert $N_A[i]$ into group $G$ until the area criterion is violated;
4.       if (area threshold is violated) then
5.           execute a batch 1-NN query using group $G$ on $T_{\mathcal{B}}$;
6.           update $maxheap$ if necessary;
7.       else
8.           continue;
9.       endif
9.   elsif ($N_A$ is INTERNAL) then
10.      check which entries $N_A[i]$ intersect $R$;
11.      sort these entries wrt MBR centroid Hilbert value;
12.      for each entry call BNN recursively;
13.  endif
```

Figure 3: Outline of BNN algorithm.

[16], in contrast to a unidirectional expansion [10]. A minheap data structure is used as a priority queue to keep pairs of entries of $T_{\mathcal{A}}$ and $T_{\mathcal{B}}$, which are promising to contain relevant object pairs from the two datasets. The minheap structure stores pairs of internal nodes only, keeping the size of the minheap at reduced levels. In addition, a maxheap data structure maintains the best k distances determined so far.

Algorithm SCP continuously retrieves pairs of entries from the minheap, until the priority of the minheap top is greater than the current d_k. Let (E_A, E_B) be the next pair of entries retrieved by the minheap. We distinguish the following cases:

- Both E_A and E_B correspond to internal nodes: in this case a bidirectional expansion is applied in order to retrieve the sets of MBRs of the two nodes pointed by E_A and E_B respectively. Then, plane-sweeping is applied in order to determine new entry pairs, which are either rejected or inserted into minheap according to their distance.

- Both E_A and E_B correspond to leaf nodes: in this case a batch operation is executed, by means of the plane-sweep technique, in order to determine object pairs (a_i, b_j) of the two datasets that may contribute to the final answer. If $dist(a_i, b_j) > d_k$ then the pair is rejected. If $dist(a_i, b_j) \le d_k$ and object a_i does not exist in maxheap, then the pair (a_i, b_j) is inserted into maxheap. However, if a_i is already in maxheap, we check if the new distance is smaller than the already recorded one. In this case, the distance of a_i is replaced in maxheap.

- One of the two entries corresponds to an internal node, and the other entry corresponds to a leaf node: in this case a unidirectional expansion is performed only for the entry which corresponds to an internal node. New entry pairs are either rejected or inserted into minheap.

In summary, the SCP algorithm can be used for k-CSCPQ query processing, provided that:

- a node of $T_{\mathcal{A}}$ is inspected only if its MBR intersects the query region R and
- during plane-sweeping operations each object from $\mathcal{D}_{\mathcal{A}}$ is considered only once

The outline of the SCP algorithm is illustrated in Figure 4.

Algorithm SCP (N_A, N_B, k, R)
Input: N_A is a node of T_A (initially is the root)
N_B is a node of T_B (initially is the root)
k is the requested number of closest pairs,
R is the query region
Output: *maxheap* is the heap of the best k answers
Local: *minheap* is the priority queue which stores pairs of entries of T_A and T_B

```
1.   inspect the two roots $N_A$ and $N_B$ using plane-sweep;
2.   store relevant entry pairs in minheap;
3.   while ((minheap is not empty) and (priority of top $\leq d_k$))
4.       get the top of minheap;
5.       if (both entries correspond to leaf nodes) then
6.           read contents of both leaf nodes;
7.           execute plane-sweep;
8.           update maxheap if necessary;
9.       elsif (both entries correspond to internal nodes) then
10.          read contents of both internal nodes; /* bidirectional expansion */
11.          execute plane-sweep;
12.          insert new entry pairs into minheap if necessary;
13.      else
14.          check the entry which corresponds to a leaf node against the other entries;
15.          insert new entry pairs into minheap if necessary;
16.      endif
17.  endwhile
```

Figure 4: Outline of SCP algorithm.

4 The Proposed Approach (The Probe-and-Search Algorithm)

In this section, we present a new algorithm for answering k-CSCPQ queries, when the two datasets under consideration are indexed by means of R*-trees or similar access methods (e.g., X-trees). The investigation of a new algorithm is motivated by the following characteristics of existing techniques:

- The MNN algorithm is expected to issue a significant number of 1-NN queries to T_B for each object retrieved by T_A enclosed by the query region R. Moreover, all nodes of T_A intersected by R will be inspected, since no other pruning is possible.

- The BNN algorithm is tailored to answer All-NN queries. If the primary dataset $\mathcal{D}_\mathcal{A}$ is dense in comparison to $\mathcal{D}_\mathcal{B}$, then many leaf nodes of T_A will participate in the batching operation, resulting in high CPU and I/O costs. Moreover, all nodes of T_A intersected by R will be inspected, since no other pruning is possible.

- The SCP algorithm has a pruning criterion, and therefore it is not necessary to inspect all nodes of T_A intersected by R. However, bidirectional expansion and synchronized tree traversal can lead to a large number of entries inserted into the heap structure, especially for datasets with a high degree of overlap.

According to the aforementioned characteristics of existing methods we would like to design an algorithm having the following properties:

- The algorithm should have reduced CPU cost, which is enabled by the use of batching operations,
- Buffer exploitation should be increased introducing as few buffer misses as possible,
- The working memory of the algorithm should be low, and
- Pruning of T_A should be enforced in order to avoid inspecting all tree nodes intersected by R.

In the sequel we present in detail the proposed algorithm, which is termed Probe-and-Search (PaS). It consists of three stages: a) searching the primary tree, b) pruning the primary tree and c) performing batching operations in the reference tree.

4.1 Searching the Primary Tree

Given the number of requested answers k and the query region R, the algorithm begins its execution by inspecting relevant nodes of the primary dataset, which is organized by T_A. Instead of using a recursive method to traverse the tree, a heap structure is used to accommodate relevant entries. The heap priority is defined by the Hilbert value of the MBR centroid of every inspected node entry, as it has been used in [21]. We call this structure *HilbertMinHeap*. When a new node of T_A is visited, we check which of its entries are intersected by the query region R. Then, the Hilbert value of each of these entries is calculated, and the pair (entry, HilbertValue) is inserted into *HilbertMinHeap*.

The use of the Hilbert value guarantees that locality of references is preserved, and therefore, nodes that are located close in the native space are likely to be accessed sequentially. The search is continued until a node is reached which resides in the level right above the leaf level.

4.2 Pruning the Primary Tree

In order to prune the primary tree T_A, the PaS algorithm should be able to determine whether a node of T_A cannot contribute to the result. Let N_A be a node examined by PaS (i.e., it has been inserted in *HilbertMinHeap*). For each N_A's entry, $N_A[i]$, PaS checks if there is an intersection of $N_A[i]$ with R. If this is true, then a 1-NN query is issued to T_B and the minimum distance *mindist* between $N_A[i].mbr$ and an object in T_B is determined. If the calculated *mindist* is larger than the current k-th best distance, then it is easy to see that the further examination of $N_A[i]$ can be pruned, because $N_A[i]$ will not contain any object whose distance from any object of T_B will be less than the currently found k-th distance. In this case, we avoid the fetching of the corresponding page and the examination of $N_A[i]$'s entries.

Since PaS uses the aforementioned pruning criterion, we would like to prioritize the examination of the entries of T_A according to their *mindist* distance from the entries of T_B. This way, the most promising entries of T_A are going to be examined first. Thus, the current k-th best distance will be accordingly small so as to prune many entries of T_A and the final result will be formed more quickly. PaS performs the required prioritization by placing the examined entries of T_A into a second heap structure. An entry in this heap comprises a pair ($N_A[i]$, *mindist*), where *mindist* is the result of the 1-NN query issued to T_B by $N_A[i].mbr$. The entries in this heap are maintained according to their *mindist* values.

It is easy to contemplate that the closer to the root of T_A a node is, the smaller its corresponding *mindist* from T_B will be. Therefore, the nodes of the upper levels of T_A are more difficult to be pruned. Moreover, we would spend considerable cost to issue 1-NN queries for such nodes, which will not payoff. For this reason, PaS uses the prioritization scheme only for the leaves of T_A. Since the number of leaves in the R*-tree is much larger than the number of internal nodes, the gain is expected to be significant. Consequently, once an internal node $N_A[i]$, which is a father of a leaf, is inspected (i.e., was previously an entry in the *HilberMinHeap*), then for all its children (leaves) $N_A[i]$ that intersect query region R, a 1-NN query is issued against T_B. As a result, pairs of the form ($N_A[i]$, *mindist*) are inserted into the second priority heap, which is denoted as *LeafMinHeap*. Evidently, a leaf node never enters the *HilberMinHeap*, thus no duplication incurs. The entries of *LeafMinHeap* are examined (in a batch mode) in the sequel, to find those that will contribute to the final result. This issue is considered in more detail in the following subsection.

Figure 5 illustrates a schematic description of the searching and pruning operations of the PaS algorithm. The figure also illustrates the separate parts of the tree, which populate the different heap structures that are maintained.

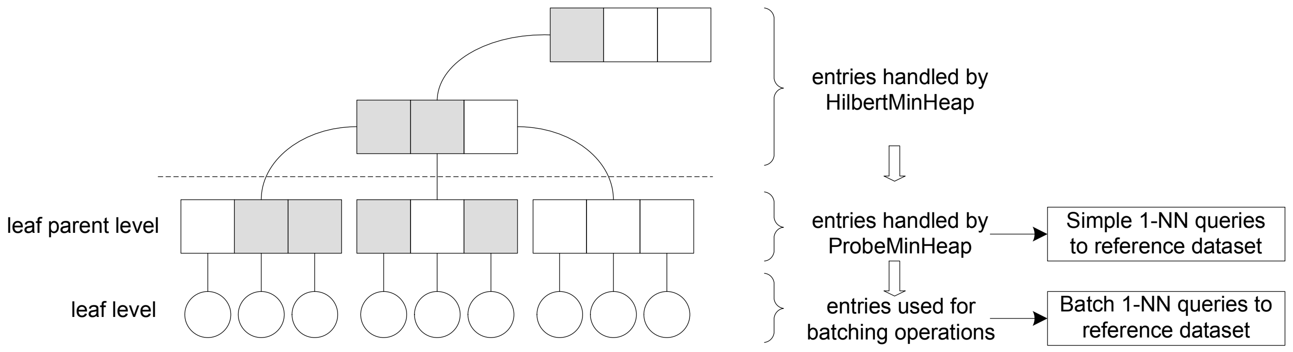

entries handled by HilbertMinHeap
leaf parent level
entries handled by ProbeMinHeap
Simple 1-NN queries to reference dataset
leaf level
entries used for batching operations
Batch 1-NN queries to reference dataset

Figure 5: Bird's eye view of the Probe-and-Search algorithm.

4.3 Enhancing the Algorithm with Batching Capabilities

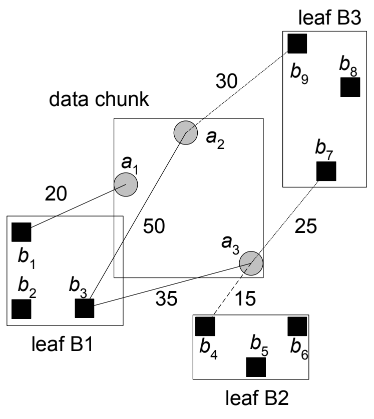
Pruning the primary tree is one direction towards reducing the number of distance calculations. Another direction to achieve this, is to apply batching operations during query processing. The basic idea is to perform multiple nearest-neighbor queries for a set of data objects, instead of calculating the nearest neighbor for each object separately.

The BNN algorithm uses batching operations in an aggressive way, to avoid individual 1-NN queries as much as possible. Recall that BNN focuses on All-NN queries instead of k-NN queries. Therefore, the size of each chunk can be quite large resulting in increased CPU and I/O costs. This effect is stronger when the primary dataset $\mathcal{D_A}$ is dense in comparison to $\mathcal{D_B}$. In this case, a large number of leaf nodes of T_A participate in the formulation of each chunk before the area criterion is violated.

Instead of relying on when the area criterion will be violated, we enforce that batching is performed for objects contained in a single leaf of T_A. The relevant leaf entries that may change the answer set are accommodated in the *LeafMinHeap* structure. These entries are inspected one-by-one by removing the top of the heap. For each such entry $N_A[i]$, the following operations are applied:

- The leaf node L pointed by $N_A[i].ptr$ is read into main memory.
- The MBR of all objects in L enclosed by R is calculated.
- If the area of the MBR is less than or equal to the average leaf area of all leaf nodes of T_B, a batch query is issued to T_B.
- Otherwise, objects are distributed to several chunks, and for each chunk a separate batch query is issued.

Let us describe in more detail the chunk generation process. Upon visiting a leaf node L of the primary tree T_A, we check if $area(L) \leq avgLeafArea_B$, where $avgLeafArea_B$ is the average area of leaf nodes in the reference tree T_B. In this case, a single chunk is generated, composed of all data objects in leaf L, and a batch query is executed for this chunk. Otherwise, data objects in L are partitioned into two or more chunks. Objects in L are sorted with respect to their Hilbert values. Then, we keep on including objects in the Hilbert order until the area criterion is violated. This signals the completion of a chunk and the initiation of another. The process continues, until all objects in L have been investigated. For each generated chunk a batch query is executed to T_B.



Hash table contents, after processing leaf B1		
ID	**ID NN**	**Dist**
*a*1	*b*1	20
*a*2	*b*3	50
*a*3	*b*3	35

Hash table contents, after processing leaf B3		
ID	**ID NN**	**Dist**
*a*1	*b*1	20
*a*2	*b*9	30
*a*3	*b*7	25

Hash table contents, after processing leaf B2		
ID	**ID NN**	**Dist**
*a*1	*b*1	20
*a*2	*b*9	30
*a*3	*b*4	15

Figure 6: Batch query processing by means of the *BatchHashTable* structure.

Each batch query is executed recursively by traversing nodes of T_B with respect to the *mindist* distance between the MBR of the chunk and the MBR of each visited node entry. Each time a leaf node is reached, the pairwise distances are calculated by using a plane-sweep technique, to avoid checking all possible pairs of objects. A hash table is maintained, called *BatchHashTable*, which stores the currently best distance for each object in the chunk. When another leaf node of T_B is reached, a test is performed, to determine if there is an object in the new leaf that change the best answers determined so far. Figure 6 illustrates an example of a batch query and the contents of the *BatchHashTable* structure after each leaf inspection. The best answers after each leaf process are shown shaded in the hash table.

The number of *BatchHashTable* elements is bounded by the maximum number of entries that can be accommodated in a leaf node, and therefore, its size is very small. After the completion of the batch query execution, the contents of *BatchHashTable* are merged with the globally determined k best answers, which are maintained in a heap structure, called *AnswersMaxHeap*. This structure accommodates the best k answers during the whole process.

4.4 Putting it All Together

In summary, the PaS algorithm operates by means of a sequence of search, probe and batch operations. Table 2 contains the auxiliary data structures that are required and the corresponding descriptions, whereas Figure 7 contains the outline of the PaS algorithm.

Data Structure	**Description**
AnswersMaxHeap	A priority queue which accommodates the best k answers during the whole execution.
HilbertMinHeap	A priority queue which contains entries of T_A prioritized with respect to the Hilbert value of MBR centroids.
LeafMinHeap	A priority queue where entries of T_A are prioritized according to *mindist* value between entries of T_A and objects in T_B.
BatchHashTable	A hash table used for each batch query operation, which contains the distances of each object in the data chunk.

Table 2: Auxiliary data structures utilized by PaS algorithm.

5 Performance Study

5.1 Preliminaries

The algorithms PaS, BNN, MNN and SCP have been implemented in C++, and the experiments have been conducted on a Windows XP machine with Pentium IV at 2.8GHz. The real datasets

Algorithm PaS (N_A, N_B, k, R)
Input: N_A is a node of T_A (initially is the root)
N_B is a node of T_B (initially is the root)
k is the requested number of closest pairs,
R is the query region
Output: $AnswersMaxHeap$ is the heap of the best k answers
Local: $HilbertMinHeap$, $LeafMinHeap$, $BatchHashTable$

```
1.   inspect the root $N_A$ and insert entries in $HilbertMinHeap$ if they intersect $R$;
2.   while ($HilbertMinHeap$ is not empty)
3.       get the top element of $HilbertMinHeap$;
4.       read node $N_A$ in memory;
5.       if entry does NOT point to a parent of a leaf then
6.           foreach entry $N_A[i]$ do
7.               if $N_A[i].mbr$ intersects $R$ then
8.                   calculate Hilbert value of $N_A[i].mbr$ centroid;
9.                   insert entry into $HilbertMinHeap$;
10.              endif
11.          endfor
12.      else
13.          foreach entry $N_A[i]$ do
14.              probe $T_B$ and calculate $mindist$ between $N_A[i].mbr$ and an object in $T_B$;
15.              if $mindist$ ¡= $d_k$ then insert entry into $LeafMinHeap$ endif
16.          endfor
17.          while ($LeafMinHeap$ is not empty)
18.              get the top element of $LeafMinHeap$;
19.              read leaf node $N_A$ in main memory;
20.              separate objects in $N_A$ into chunks;
21.              foreach chunk do
22.                  initialize $BatchHashTable$;
23.                  execute batch query to $T_B$;
24.                  update $AnswersMaxHeap$ from $BatchHashTable$ if necessary;
25.              endfor
26.          endwhile
27.      endif
28.  endwhile
```

Figure 7: Outline of PaS algorithm.

used for the experimentation are taken from TIGER [19] and are illustrated in Figure 8. The LA1 dataset contains 131,461 centroids of MBRs corresponding to roads in Los Angeles. Dataset LA2 contains 128,971 centroids of MBRs corresponding to rivers and railways in Los Angeles. Finally, dataset CA contains 1,300,000 centroids of road MBRs of California. These datasets are available from http://www.rtreeportal.org/spatial.html. In addition to the above datasets, we also use uniformly distributed points. All datasets are normalized to a square, where each dimension takes real values between 0 and 1023.

Notice that LA1 and LA2 are quite similar, having similar data distributions and populations. On the other hand, dataset CA shows completely different data distribution and population. The selection of these datasets have been performed, to test the performance of the algorithms in cases

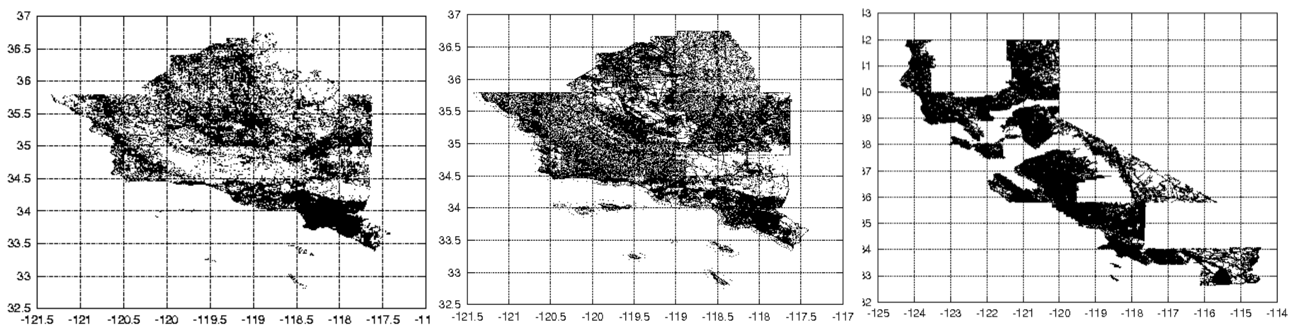

(b) LA1, 131461 (b) LA2, 128971 (c) CA, 1300000

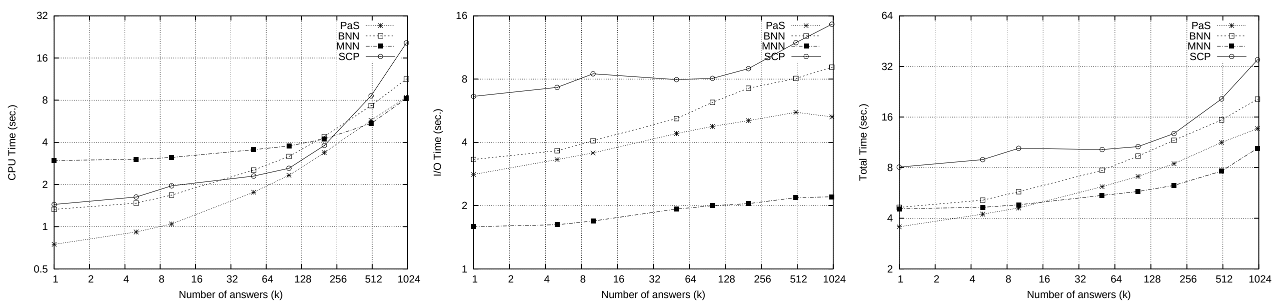
Figure 8: Datasets used in the experimentation.

where the primary and reference dataset follow the same distribution. In the sequel, we investigate the performance of the algorithms for three cases: 1) $\mathcal{D}_\mathcal{A}$=LA1 and $\mathcal{D}_\mathcal{B}$=CA, 2) $\mathcal{D}_\mathcal{A}$=CA and $\mathcal{D}_\mathcal{B}$=LA1, and 3) $\mathcal{D}_\mathcal{A}$=LA1 and $\mathcal{D}_\mathcal{B}$=LA2. The aforementioned cases correspond to the three different possibilities regarding the relative size between the primary and the reference dataset. In particular, in case (1) the primary dataset is significantly smaller than the reference dataset, in case (2) the primary dataset is significantly larger than the reference dataset, and in case (3) the primary and the reference datasets are of about the same size. In most application domains, the reference objects are much less than the objects in the primary dataset (e.g., authoritative sites are much smaller than domestic buildings). Therefore, (2) is the case of interest for the majority of application domains. Case (3) can also be possible in some applications. In contrast, one should hardly expect an application domain for case (1). Nevertheless, for purposes of comparison, we also consider this case, to examine the relative performance of the examined methods in all possible cases.

In each experiment, 100 square-like queries are executed following the distribution of the primary dataset $\mathcal{D}_\mathcal{A}$. CPU, I/O and total time correspond to average values per query. We note that if a selected region of interest does not contain at least k objects from the primary dataset, then a new region is generated to satisfy this constraint. The disk page size is set to 1024 bytes for all experiments conducted. An LRU page replacement policy is assumed for the buffer operation. The capacity of the buffer is measured as a percentage of the database size. In the sequel we present the results for different parameter values, i.e., the number of answers, the area of the query region, the size of the buffer, and the population of the datasets. Moreover, a discussion of the memory requirements of all methods is performed in a separate section.

5.2 Performance vs Different Parameter Values

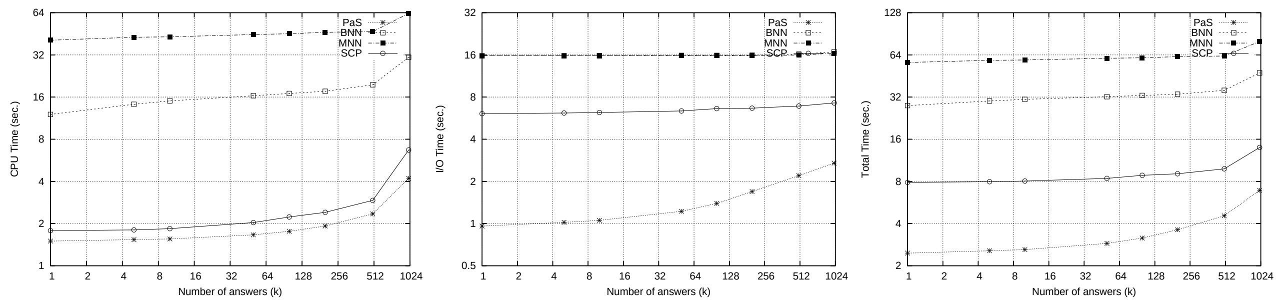
In this section we present representative experimental results which demonstrate the performance of each method under different settings.



(a) CPU time (b) I/O time (c) Total time

Figure 9: CPU time, I/O time and total time vs k for $\mathcal{D}_{\mathcal{A}}$=LA1 and $\mathcal{D}_{\mathcal{B}}$=CA (logarithmic scales).

We start by first testing case (1), that is, when the primary dataset is significantly smaller than the reference dataset. As mentioned, this case is only examined for purposes of comparison, since it does not constitute a case of interest for the vast majority of applications. Figure 9 illustrates the performance of the algorithms when $\mathcal{D}_{\mathcal{A}}$=LA1 and $\mathcal{D}_{\mathcal{B}}$=CA, by varying the number of answers k. Evidently, $\mathcal{D}_{\mathcal{B}}$ contains many more objects than $\mathcal{D}_{\mathcal{A}}$. The query region is set to 1% of the dataspace area, the buffer capacity is 10% of the total number of pages of both trees. PaS and SCP have the lowest CPU cost, whereas the CPU cost of MNN is significantly higher due to the excessive number of 1-NN queries executed, as it is illustrated in Figure 9(a). The above situation is maintained until k=500. Above that level we observe that the performance of BNN is better than that of MNN and SCP. PaS manages to keep the CPU cost at low levels for all values of k. With respect to I/O cost, which is depicted in Figure 9(b), the situation is quite different. Clearly, MNN has the lowest I/O cost due to the locality of references and the fact that every 1-NN query posed to the reference tree fetches less pages than the other algorithms. Therefore, although the number of page requests is large, most of them are absorbed by the buffer. The I/O cost of PaS is maintained at low levels, especially for k greater than 10. The I/O cost of SCP is quite significant, because it does not utilize the buffer adequately. More specifically, it does not preserve well the locality of references. On the other hand, PaS, BNN and MNN traverse the primary tree based on the Hilbert values of MBR centroids and therefore, locality of references is better preserved. The total running time of the algorithms is presented in Figure 9(c). As can been shown, PaS outperforms all other methods for smaller values of k, whereas for larger values of k MNN performs marginally better than PaS. The performance of BNN worsens with increasing k and SCP presents the larger execution times in all cases. Therefore, even for the extreme case when the reference dataset is significantly larger than the primary, the performance of PaS is reasonably good, whereas BNN and particularly SCP are not able to maintain a good performance. Notice that although MNN presents the smallest execution times for larger values of k, as it will be shown next, this happens only for case (1), since for the other cases (which are the cases of interest in most applications) MNN is the worst choice.

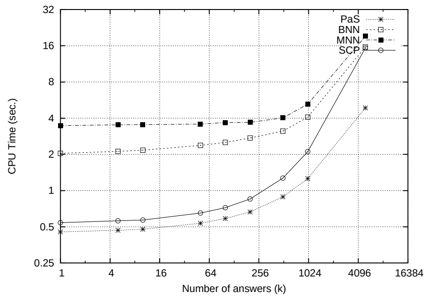


(a) CPU time (b) I/O time (c) Total time

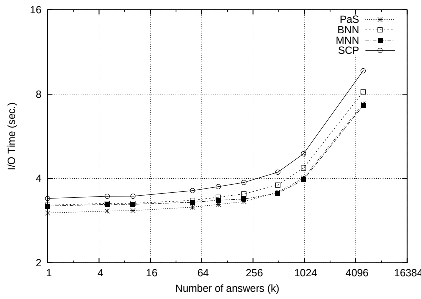
Figure 10: CPU time, I/O time and total time vs k for $\mathcal{D}_{\mathcal{A}}$=CA and $\mathcal{D}_{\mathcal{B}}$=LA1 (logarithmic scales).

Figure 10 depicts the performance of the algorithms vs k when $\mathcal{D}_{\mathcal{A}}$=CA and $\mathcal{D}_{\mathcal{B}}$=LA1. Again, the query region is set to 1% of the dataspace area and the buffer capacity is 10% of the total number of pages of both trees. It is evident that algorithm PaS shows the best performance over the other methods. MNN and BNN inspect all nodes in the primary tree that are intersected by the query region. This results in a large number of 1-NN queries performed by MNN, and a large number of batch queries performed by BNN. Moreover, since $\mathcal{D}_{\mathcal{A}}$ is quite dense, BNN groups several leaf nodes to construct each chunk for batching. This results in increased CPU and I/O cost. On the other hand, PaS is capable of pruning several nodes due to the probes performed on the reference tree. Page requests are absorbed by the buffer, resulting in significantly less I/O time with respect to the other methods.

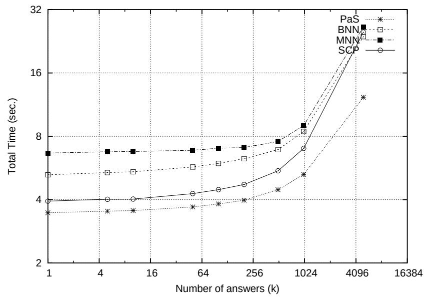
Figure 11 illustrates the performance of the algorithms under study for $\mathcal{D}_{\mathcal{A}}$=LA1 and $\mathcal{D}_{\mathcal{B}}$=LA2. These datasets follow similar distributions and they have similar populations. The query region is set to 1% of the dataspace area and the buffer capacity is 10% of the total number of pages of both trees. Again, PaS shows the best performance with respect to CPU time. Moreover, the I/O time of PaS is the smallest, for up to 256 nearest neighbors. After this threshold its performance is similar to that



(a) CPU time



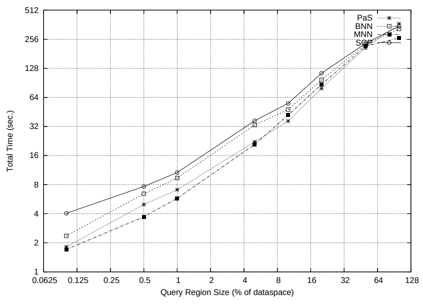
(b) I/O time



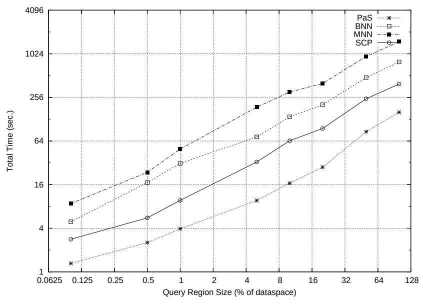
(c) Total time

Figure 11: CPU time, I/O time and total time vs k for $\mathcal{D}_{\mathcal{A}}$=LA1 and $\mathcal{D}_{\mathcal{B}}$=LA2 (logarithmic scales).

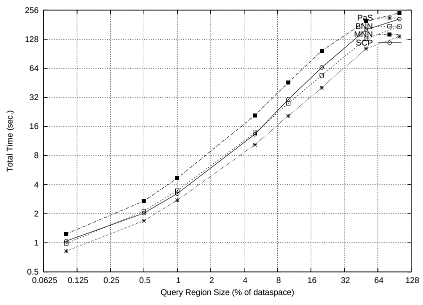
of MNN. With respect to the overall performance of the methods, PaS shows the best performance. For k=5000, the total time of the other methods converge to a large execution time, whereas that of PaS is maintained at a significantly lower level.



(a) $\mathcal{D}_{\mathcal{A}}$=LA1, $\mathcal{D}_{\mathcal{B}}$=CA



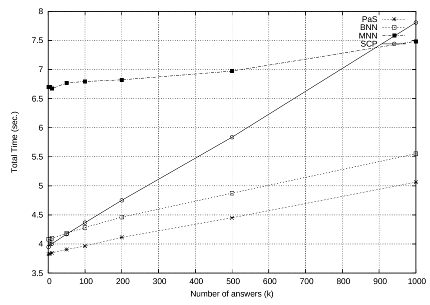
(b) $\mathcal{D}_{\mathcal{A}}$=CA, $\mathcal{D}_{\mathcal{B}}$=LA1



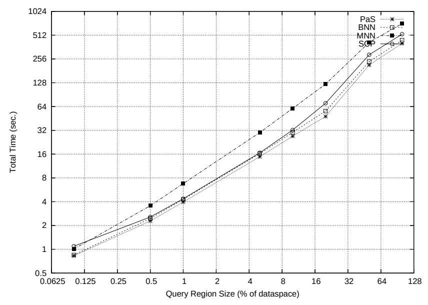
(c) $\mathcal{D}_{\mathcal{A}}$=LA1, $\mathcal{D}_{\mathcal{B}}$=LA2

Figure 12: Total time vs query region size (logarithmic scales).

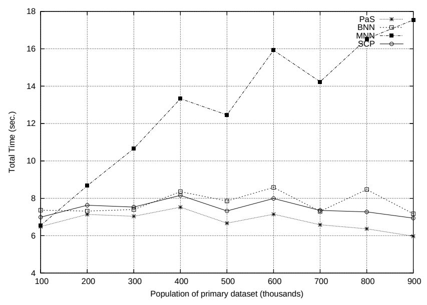
Figure 12 depicts the total running time of all methods, for all three dataset combinations, vs the area of the query region. The number of answers k is set to 100, whereas the buffer capacity is set to 10% of the total number of pages of both trees. When the reference dataset is larger than the primary dataset (Figure 12(a)) PaS MNN and show similar performance. In particular, MNN is slightly better for smaller query regions, since it manages to maintain locality of references. For larger query regions, PaS is slightly better. In contrast, BNN and SCP show similar performance (only for very small query regions BNN is better than SCP) and are clearly outperformed. It is interesting to notice that when the query region reaches 100% of the dataspace area, all methods converge to the same point. In the inverse case where the primary dataset is much larger than the reference dataset, PaS clearly shows the best performance, whereas MNN shows the worst. SCP shows better performance than BNN and MNN. Recall that PaS and SCP perform pruning in the primary R-tree, and therefore it is not necessary to visit all leaf nodes intersected by the query region. On the other hand, MNN and BNN do not have such a mechanism. Since the primary R-tree is quite large, the pruning ability of PaS and SCP helps in reducing the total running time of these methods (Figure 12(b)). Finally, when the populations of both datasets are similar (Figure 12(c)), PaS shows the best performance, although the performance difference is smaller than that of the previous case.



(a) variable k



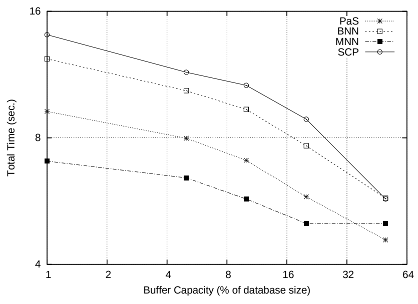
(b) variable R



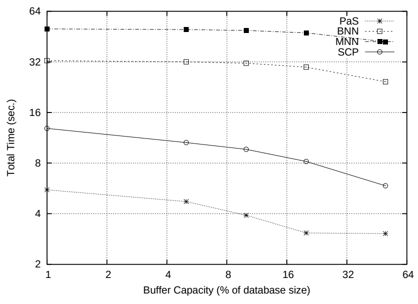
(c) variable population

Figure 13: Total time vs k, R and population for uniformly distributed datasets (logarithmic scales).

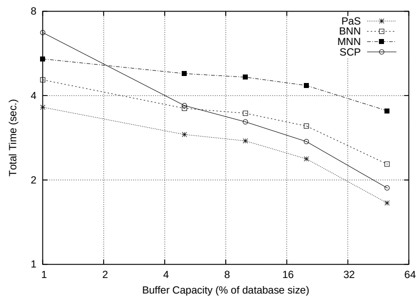
The performance of the methods vs k, R and dataset population is depicted in Figure 13 when both primary and reference datasets follow a uniform distribution. To generate Figure 13(a) the query region is set to 1% of the dataspace area whereas the buffer capacity is set to 10% of the database size. PaS shows the best performance for all values of k. The performance of BNN is close to that of PaS, whereas MNN shows very large total running time. It is interesting to note that for small values of k, SCP outperforms BNN. However, by increasing the number of nearest neighbors, SCP degenerates due to the significant overlap between the two datasets. Figure 13(b) depicts the performance of the methods vs the query region area. The number of answers k is set to 100, and the buffer capacity is again 10% of the database size. All methods are highly affected by R, and PaS shows the best performance for the whole range. Again, MNN gives high running times whereas the performance of BNN is similar to that of SCP. Finally, Figure 13(c) shows the performance for various dataset populations. We assume that both datasets contain 1,000,000 uniformly distributed points. The x axis depicts the population of the primary dataset in thousands of objects. Therefore, if the population of the primary dataset is 200,000 objects, then the population of the reference dataset is 800,000 objects. The same applies for the other values as well. The query region size is set to 1% of the dataspace area, the number of answers is 100 and the buffer capacity is set to 10% of the database size. Algorithm PaS shows the best total running time, for all combinations.



(a) $\mathcal{D}_{\mathcal{A}}$=LA1, $\mathcal{D}_{\mathcal{B}}$=CA



(b) $\mathcal{D}_{\mathcal{A}}$=CA, $\mathcal{D}_{\mathcal{B}}$=LA1



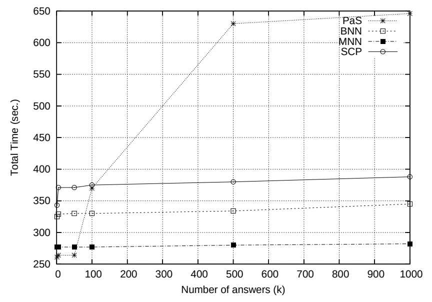
(c) $\mathcal{D}_{\mathcal{A}}$=LA1, $\mathcal{D}_{\mathcal{B}}$=LA2

Figure 14: Total time vs buffer capacity (logarithmic scales).

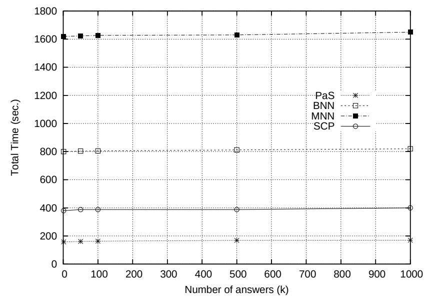
Another important parameter that may affect the performance of the methods drastically is the buffer capacity utilized. Figure 14 illustrates the total running time of the methods with respect to buffer capacity, for the three dataset combinations. The number of answers k is set to 100, and the query region area occupies 1% of the dataspace area. For $\mathcal{D}_{\mathcal{A}}$=LA1 and $\mathcal{D}_{\mathcal{B}}$=CA (Figure 14(a)), MNN shows the best performance, with PaS being competitive for buffer capacities larger than 1% of the database size. Evidently, for very large buffer capacities (>30%) the performance of the methods converge, since the number of buffer misses is reduced significantly. Another observation is that all methods are strongly dependent on the buffer capacity. Figure 14(b) illustrates the performance of the methods for $\mathcal{D}_{\mathcal{A}}$=CA and $\mathcal{D}_{\mathcal{B}}$=LA1. In this case, PaS shows the smallest total running time, with SCP following. Recall that PaS and SCP are the only methods with pruning capabilities during the search of the primary tree. Since $\mathcal{D}_{\mathcal{A}}$ is large, these two methods benefit more from the application of pruning. Finally, Figure 14(c) shows the total running time when $\mathcal{D}_{\mathcal{A}}$=LA1 and $\mathcal{D}_{\mathcal{B}}$=LA2. For buffer capacities larger than 0.5% of the database size, PaS shows the best performance.

Recall that PaS has been designed towards solving the k-Constrained-Semi-Closest-Pair problem. However, for completeness in the sequel we focus on two special cases. First, we examine the performance of the methods when the region of interest R covers 100% of the dataspace of $\mathcal{D}_{\mathcal{A}}$, and then we investigate if PaS is a good candidate for solving the All-Nearest-Neighbor problem, where again R covers 100% of the dataspace area of $\mathcal{D}_{\mathcal{A}}$ and $k = |\mathcal{D}_{\mathcal{A}}|$. In both cases the buffer capacity is set to 10% of the total storage requirements of both trees.

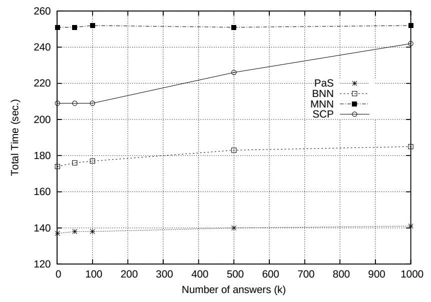
Figure 15 depicts the performance of all methods when R covers 100% of the area of $\mathcal{D}_{\mathcal{A}}$. It is evident that PaS has the best performance in the majority of cases. More specifically, it is the best



(a) $\mathcal{D}_{\mathcal{A}}$=LA1, $\mathcal{D}_{\mathcal{B}}$=CA



(b) $\mathcal{D}_{\mathcal{A}}$=CA, $\mathcal{D}_{\mathcal{B}}$=LA1



(c) $\mathcal{D}_{\mathcal{A}}$=LA1, $\mathcal{D}_{\mathcal{B}}$=LA2

Figure 15: Total time vs k for R covering 100% of $\mathcal{D}_{\mathcal{A}}$ area.

choice when we have a large primary dataset and a small reference dataset, and when both datasets have almost equal size (Figures 15(b) and (c) respectively). When the primary dataset is significantly larger than the reference one, then for small values of k PaS performs the best. However, for larger values of k its performance degenerates. This has been also noticed for cases where the region R does not cover the whole dataspace of $\mathcal{D}_{\mathcal{A}}$.

	PaS	BNN	MNN
Case (1) $\mathcal{D}_{\mathcal{A}}$=LA1, $\mathcal{D}_{\mathcal{B}}$=CA	1801	1591	616
Case (2) $\mathcal{D}_{\mathcal{A}}$=CA, $\mathcal{D}_{\mathcal{B}}$=LA1	1663	2402	3799
Case (3) $\mathcal{D}_{\mathcal{A}}$=LA1, $\mathcal{D}_{\mathcal{B}}$=LA2	226	315	319

Table 3: Total time (sec.) of Pas, BNN and MNN for All-NN queries.

Table 3 shows the performance of PaS, BNN and MNN for processing All-Nearest-Neighbor queries. Again, we conclude that for the two cases of interest PaS is the best choice. BNN is the best choice when both datasets have almost equal size and it is second best for small primary and large reference dataset. We have not included results for the SCP algorithm, because its CPU requirements are very high for solving the All-Nearest-Neighbor query, in comparison to the other methods.

In the sequel, we study the working memory requirements of the methods, which play an important role for the overall performance.

5.3 Results on Memory Requirements

By carefully studying the presented algorithms it is evident that PaS, BNN and SCP require that some pages in the buffer must be pinned to avoid additional I/O operations during query processing. Moreover, PaS and SCP require additional storage for the heap structures. In this section, we investigate the memory that each algorithm requires. The memory requirements of each method are as follows:

Memory requirements of PaS: Recall that PaS requires some storage to store the entries of the two heap structures (*HilbertMinHeap* and *LeafMinHeap*). It is interesting to investigate the amount of memory required to store these structures. Moreover, every time a batch operation is executed on the reference tree, the chunk composed of entries of the primary tree must be maintained in the buffer, to avoid extra I/O operations.

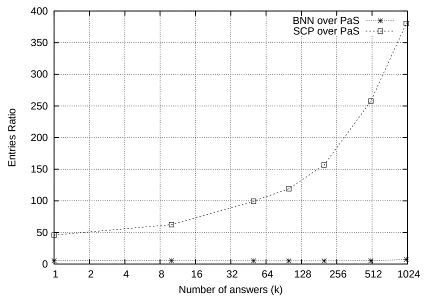
Memory requirements of BNN: BNN performs a DFS traversal of the primary tree, and therefore no heap structure is required to hold entries of the primary tree. However, the primary data entries that compose a chunk must be maintained in the buffer during a batch operation. The required memory has a direct impact on the performance of the algorithm.

Memory requirements of MNN: MNN does not require any additional data structure, and it does not perform batching operations. Therefore, no additional memory is required.

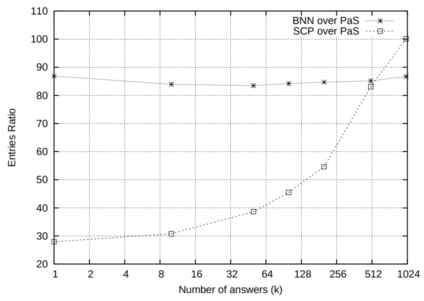
Memory requirements of SCP: Finally, SCP is a heap-based algorithm, and therefore additional memory is required to store heap elements. However, the size of the heap can be quite significant, requiring special treatment.

In the sequel, we present experimental results on the memory requirements of the algorithms. The required memory is given as the number of entries that must be hosted in the buffer during the processing of a query. For each experiment, 100 queries are executed by using the three real dataset combinations used in the previous section. Since MNN does not require any additional storage, it is excluded from our study. Moreover, since the structures *AnswersMaxHeap* and *BatchHashTable* are required by all algorithms, we do not take into account the required storage.

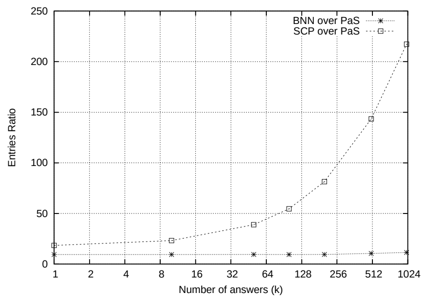
By inspecting Figures 16 and 17 it is evident that algorithm PaS requires considerably less memory than BNN and SCP. Therefore, PaS could be executed in less memory space. Moreover, the significant memory consumption of algorithms BNN and SCP could lead to large storage overhead. If this memory is part of the buffer used, then the buffer utilization would deteriorate.



(a) $\mathcal{D}_{\mathcal{A}}$=LA1, $\mathcal{D}_{\mathcal{B}}$=CA

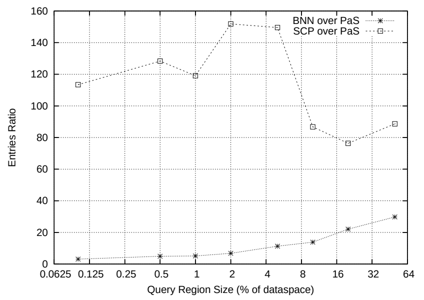


(b) $\mathcal{D}_{\mathcal{A}}$=CA, $\mathcal{D}_{\mathcal{B}}$=LA1

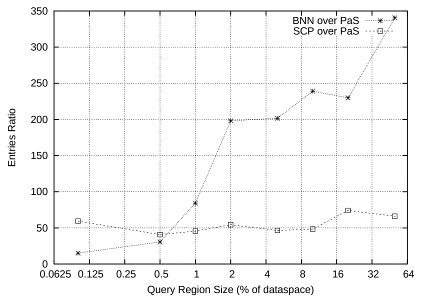


(c) $\mathcal{D}_{\mathcal{A}}$=LA1, $\mathcal{D}_{\mathcal{B}}$=LA2

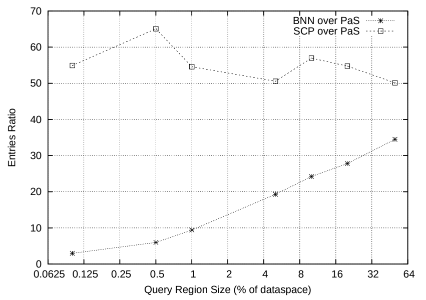
Figure 16: Working memory requirements (number of entries) vs k.



(a) $\mathcal{D}_{\mathcal{A}}$=LA1, $\mathcal{D}_{\mathcal{B}}$=CA



(b) $\mathcal{D}_{\mathcal{A}}$=CA, $\mathcal{D}_{\mathcal{B}}$=LA1



(c) $\mathcal{D}_{\mathcal{A}}$=LA1, $\mathcal{D}_{\mathcal{B}}$=LA2

Figure 17: Working memory requirements (number of entries) vs R.

5.4 Discussion

Herein, we give a synopsis of the previously described experimental results on execution times. Also, we combine the conclusions drawn from the results on main-memory requirements and remark on their impact on execution times. Finally, we give some theoretical aspects regarding the performance of PaS and justify its good behavior.

Table 4 illustrates the characterization of the performance of all examined algorithms with respect to the three cases (1: small primary and large reference dataset, 2: large primary and small reference dataset, 3: primary and reference dataset of similar size). Recall that the first case is examined only for purposes of comparison, since only the second (in particular) and third cases are of interest in most applications.

Evidently, the proposed algorithm (PaS) presents the best performance in cases of interest (second and third), whereas for the first case it still manages to maintain relatively good performance. In contrast, MNN is the worst choice in all cases except the first one. BNN is also a bad choice, since in

	PaS	SCP	BNN	MNN
Case (1)	Good	Bad	Bad	Best
Case (2)	Best	Bad	Bad	Bad
Case (3)	Best	Good	Bad	Bad

Table 4: Characterizing the performance of all algorithms wrt to the three examined cases.

all three cases it has the second worst execution time amongst all methods. SCP is second best in the second case, but in all other cases its performance is bad. Overall, PaS performs favorably against the other algorithms and has the appealing characteristic of presenting stably good performance in all cases.

Let us investigate in more detail, why PaS manages to maintain a good performance in comparison to the other approaches. Recall that, PaS differs from MNN and BNN in the fact that a pruning of the primary tree is performed, by issuing 1-NN queries to the reference tree when a node in the leaf-parent level of the primary tree is reached. This operation discards parts of the primary dataset that can not contribute to the final answer. Let $n_A = |\mathcal{D_A}|$ and $n_B = |\mathcal{D_B}|$ denote the number of objects in $\mathcal{D_A}$ and $\mathcal{D_B}$ respectively. Moreover, let f_A and f_B denote the average node utilization (number of objects per node) for the primary (T_A) and reference (T_B) tree respectively. The total number of leaf nodes in the primary tree is given by:

$$NL_A = \lceil \frac{n_A}{f_A} \rceil \tag{1}$$

Recall that, PaS performs a 1-NN query to T_B for every leaf node MBR intersected by the query region R. Let R_x and R_y be the extensions of R in the x and y axis respectively. Then, following [9] the number of leaf node MBRs intersected by R is given by the next equation, where le_x and le_y is the average leaf MBR extension in the x and y axis respectively:

$$NL_A(R) = \frac{n_A}{f_A} \cdot (le_x + R_x) \cdot (le_y + R_y) \tag{2}$$

By assuming a uniform distribution for $\mathcal{D_A}$, we expect square-like MBRs, and all leaf MBRs are expected to contain the same number of data objects. Therefore, $le_x = le_y = \sqrt{\frac{f_A}{n_A}}$ and each leaf node of T_A is expected to contain f_A objects. By substituting in the previous equation we get:

$$NL_A(R) = \frac{n_A}{f_A} \cdot \left(\frac{f_A}{n_A} + \sqrt{\frac{f_A}{n_A}} \cdot (R_x + R_y) + R_x \cdot R_y \right) \tag{3}$$

Equation (3) gives the number of 1-NN queries that will be executed to T_B to check if a leaf node of T_A is promising or not. A leaf node MBR of T_A is considered promising if after the execution of the 1-NN search the distance between this MBR and an object in D_B is less than or equal to the current best k-th distance determined so far. If a leaf node MBR survives the 1-NN probe, then the corresponding leaf contents are candidates for the final result. The final decision will be performed after the execution of the batching operations.

Recall that each leaf node of the primary tree T_A contains on the average f_A data objects. Therefore, the minimum number of batch queries for each leaf is 1 and the maximum is $\frac{f_A}{2}$. The worst case appears when we can pack only two objects at a time before the average leaf area criterion is violated. Again, assuming a uniform distribution for the objects in the reference dataset D_B, the average MBR area for the leaf nodes of T_B is $avg_B = \frac{f_B}{n_B}$. Therefore, the number of executed batch NN queries for every promising leaf node of T_A is given by $\lceil \frac{avg_A}{avg_B} \rceil$. This suggests that every time a leaf node is not considered promising (which is determined by a probe to T_B) we save $\lceil \frac{avg_A}{avg_B} \rceil$ batch operations. This is important, since the cost for a batch operation is more computationally intensive than that of a probe operation, because the former requires the examination of data objects by means of plane sweeping. In conclusion, the more leaf node MBRs are discarded from the set of $NL_A(R)$ MBRs, the more batching operations are saved in the subsequence step.

To illustrate the pruning effect, Figure 18 depicts the number of objects from $\mathcal{D}_{\mathcal{A}}$ that are discarded due to the series of 1-NN queries issued to T_B for the leaf-parent level of T_A, when R covers 1% and 10% of $\mathcal{D}_{\mathcal{A}}$. The parameter k has been set to 1, 20, 100 and 500, whereas the size of the buffer has been set to 10% of the total size of both trees. It is evident, that a significant number of objects are pruned, leading to a reduced number of batching operations. Note however, that when the reference dataset is much larger than the primary dataset, each 1-NN query does not come at free. In this case, the number of saved batch queries may not compensate the cost of the 1-NN queries for the leaf MBRs (as it has been demonstrated in the experimental results).

Regarding the impact of main-memory requirements on the performance, we showed that for the first and third cases, SCP requires significantly more memory than the others, whereas for the second case, the requirements of BNN are significantly larger than those of the others. In all cases, PaS needs the smallest amount of main-memory, to process the examined query. What can be inferred from the aforementioned conclusion is that we had an optimistic perspective when examined the performance of SCP and BNN, since their significant requirements for main-memory where not handled by a scheme

R covers	k	total	pruned
1%	1	1924	850
	20	1957	788
	100	2048	668
	500	2557	582
10%	1	16938	10352
	20	16938	10194
	100	16938	10003
	500	17535	9623

(a) $\mathcal{D}_{\mathcal{A}}$=LA1, $\mathcal{D}_{\mathcal{B}}$=CA

R covers	k	total	pruned
1%	1	26578	23676
	20	26578	23239
	100	26578	22577
	500	26578	20728
10%	1	207130	192773
	20	207130	191964
	100	207130	190764
	500	225069	201671

(b) $\mathcal{D}_{\mathcal{A}}$=CA, $\mathcal{D}_{\mathcal{B}}$=LA1

R covers	k	total	pruned
1%	1	1924	235
	20	1957	201
	100	2048	172
	500	2557	119
10%	1	16938	2310
	20	16938	2201
	100	16938	2035
	500	17535	1859

(c) $\mathcal{D}_{\mathcal{A}}$=LA1, $\mathcal{D}_{\mathcal{B}}$=LA2

Figure 18: Number of objects in R that are pruned by using simple 1-NN queries.

that stores their corresponding data structures partially on disk (when main-memory does not suffices). In more strict implementations, however, such an assumption may not be followed, thus the execution times of SCP and BNN will be further increased by the I/O cost of accessing the parts of the data structures that are held on disk. This clearly shows that PaS is expected to outperform the other methods at a higher degree.

6 Concluding Remarks and Further Research

Distance based queries are considered very important in several domains, such as spatial databases, spatiotemporal databases, data mining tasks, to name a few. An important family of distance-based queries involve the association of two or more datasets. In this paper, we focused on the k-Semi-Closest-Pair query with spatial constraints applied to the objects of the first dataset (primary dataset). We studied several existing algorithms that can solve the problem with necessary modifications. In addition, we proposed a new technique which has the following benefits: a) requires less memory for query processing in comparison to existing techniques, b) requires less CPU processing time and c) requires less total running time.

There are several directions for further research in the area that may lead to interesting results. We note the following:

- the application of the proposed method for k-NN join processing,
- the adaptation of the method for high-dimensional spaces (perhaps with the aid of more efficient access methods),
- the study of processing techniques when constraints are also applied to the reference dataset,
- the study of PaS variations when there are no available access methods for one or both datasets.

References

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger: "The R*-tree: an Efficient and Robust Access Method for Points and Rectangles", *Proc. ACM SIGMOD*, pp. 322-331, Atlantic City, NJ, May 1990.

[2] C. Bohm and F. Krebs: "Supporting KDD Applications by the K-Nearest Neighbor Join", *Proceedings of the 14th International Conference on Database and Expert System Applications (DEXA 2003)*, pp.504-516, Prague, Czech Republic, 2003.

[3] C. Bohm and F. Krebs: "The k-Nearest Neighbor Join: Turbo Charging the KDD Process", *Knowledge and Information Systems (KAIS)*, 2004.

[4] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander: "LOF: Identifying Density-Based Local Outliers", *Proceedings of the ACM International Conference on the Management of Data (SIGMOD 2000)*, pp.93-104, Dallas, TX, 2000.

[5] T. Brinkhoff, H. P. Kriegel, and B. Seeger: "Efficient Processing of Spatial Joins Using R-trees", *Proceedings of the ACM International Conference on Management of Data (SIGMOD 1993)*, pp.237-246, Washington, D.C., May 1993.

[6] A. Corral, Y. Manolopoulos, Y. Theodoridis and M. Vassilakopoulos: "Closest Pair Queries in Spatial Databases", *Proceedings of the ACM International Conference on the Management of Data (SIGMOD 2000)*, Dallas, TX, 2000.

[7] A. Corral, Y. Manolopoulos, Y. Theodoridis and M. Vassilakopoulos: "Algorithms for Processing K-Closest-Pair Queries in Spatial Databases", *Data and Knowledge Engineering (DKE)*, Vol.49, No.1, pp.67-104, 2004.

[8] D. Eppstein: "Fast Hierarchical Clustering and Other Applications of Dynamic Closest Pairs", *Journal of Experimental Algorithmics*, Vol.5, No.1, pp.1-23, 2000.

[9] C. Faloutsos and I. Kamel: "Beyond Uniformity and Independence: Analysis of R-trees Using the Concept of Fractal Dimension", *Proceedings of 13th ACM Symposium on Principles of Database Systems (PODS 1994)*, 1994.

[10] G.R. Hjaltason and H. Samet: "Incremental Distance Join Algorithms for Spatial Databases", *Proceedings of ACM SIGMOD Conference*, pp.237-248, 1998.

[11] G. Karypis, E.-H. Han, and V. Kumar: "Chameleon: Hierarchical Clustering Using Dynamic Modeling", *Computer*, Vol.32, No.8, pp.68-75, 1999.

[12] P. Mishra and M. H. Eich: "Join Processing in Relational Databases", *ACM Computing Surveys*, Vol.24, No.1, 1992.

[13] A. Nanopoulos, Y. Theodoridis and Y. Manolopoulos: "C^2P: Clustering Based on Closest Pairs", *Proceedings of the 27th International Conference on Very Large Databases (VLDB 2001)*, Roma, Italy, 2001.

[14] J. Shan, D. Zhang and B. Salzberg: "On Spatial-Range Closest-Pair Query", *Proceedings of the 8th International Symposium on Spatial and Temporal Databases (SSTD 2003)*, pp.252-269, Santorini, Greece, 2003.

[15] K. Shim, R. Srikant and R. Agrawal: "High-Dimensional Similarity Joins", *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, Vol.14, No.1, pp.156-171, 2002.

[16] H. Shin, B. Moon and S. Lee: "Adaptive Multi-Stage Distance Join Processing", *Proceedings of the ACM SIGMOD Conference*, pp.343-354, 2000.

[17] Y. Shou, N. Mamoulis, H. Cao, D. Papadias, and D. W. Cheung: "Evaluation of Iceberg Distance Joins", *Proceedings of the 8th International Symposium on Spatial and Temporal Databases (SSTD 2003)*, pp.270-278, Santorini, Greece, 2003.

[18] Y. Tao and D. Papadias: "Time-Parameterized Queries in Spatio-Temporal Databases" *Proceedings of the ACM International Conference on the Management of Data (SIGMOD 2002)*, pp. 334-345, 2002.

[19] TIGER/Line Files, 1994 Technical Documentation / prepared by the Bureau of the Census, Washington, DC, 1994.

[20] C. Xia, H. Lu, B. C. Ooi and J. Hu: "GORDER: An Efficient Method for KNN Processing", *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB 2004)*, pp.756-767, Toronto, Canada, 2004.

[21] J. Zhang, N. Mamoulis, D. Papadias and Y. Tao: "All-Nearest-Neighbors Queries in Spatial Databases", *Proceedings of the 16th International Conference on Scientific and Statistical Databases (SSDBM 2004)*, pp.297-306, Santorini, Greece, 2004.