# Closest Pair Queries with Spatial Constraints ⋆

Apostolos N. Papadopoulos Alexandros Nanopoulos Yannis Manolopoulos

Department of Informatics, Aristotle University, GR-54124 Thessaloniki, Greece.
E-mail: {apostol,alex,manolopo}@delab.csd.auth.gr

**Abstract.** 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 $R$. A new algorithm is proposed, which is compared with a modified closest-pair algorithm. The experimental results demonstrate that the proposed approach is superior with respect to CPU and I/O costs.

## 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, $k$ nearest-neighbor query, spatial join query and closes-pair query. t 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-pair 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 [11, 5, 6, 14, 17, 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

⋆ Research supported by ARCHIMEDES project 2.2.14, “Management of Moving Objects and the WWW”, of the Technological Educational Institute of Thessaloniki (EPEAEK II), and by the 2003-2005 Serbian-Greek joint research and technology program.

$\mathcal{D}_{\mathcal{A}}$ 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 $k = 2$.

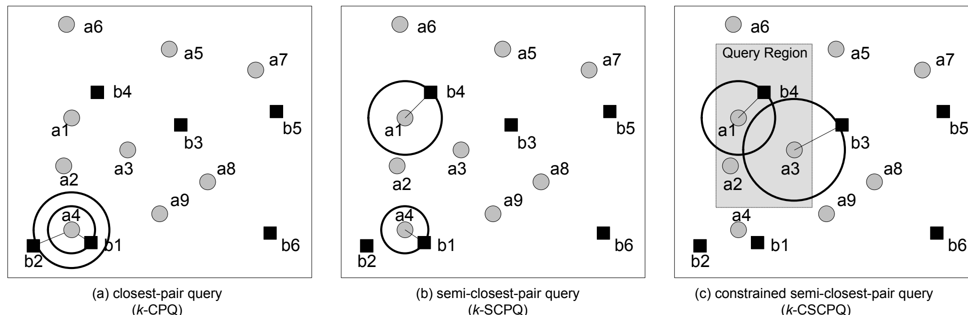



**Fig. 1.** Results of CPQ variations for $k$=2.

Distance-based queries, such as nearest-neighbor and closest-pair, 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 in order to draw conclusions. A clustering algorithm based on closest pairs has been proposed in [12]. 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 [19] it is reported that the $k$-NN join can also be used to improve the performance of LOF algorithm, which is used for outlier detection in a single dataset [4], and also to improve the performance of the Chameleon clustering algorithm [10]. 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

[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.

region which determines 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 of more complex operations, in order 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 [13], 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 [16] 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 [20] 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 [9, 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 [15]. The details of the Closest-Pair algorithm, which is used for comparison purposes, are given in the subsequent section.

# 3 Processing Techniques

## 3.1 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 [15]. 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 [15], in contrast to a unidirectional expansion [9]. 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 $dist(a_i, b_j) \leq d_k$ 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.

## 3.2 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. We would like to devise 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.

**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 [20]. 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.

**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 access to 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 shaped 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 expected gain

is still 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 *HilbertMinHeap*), 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 *HilbertMinHeap*, thus no duplication incurs. The entries of *LeafMinHeap* are examined (in a batch mode) in the sequel, in order to find those that will contribute to the final result. This issue is considered in more detail in the following subsection.

Figure 2 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.

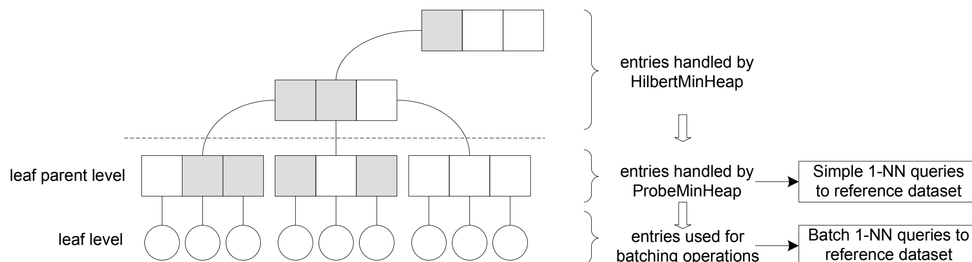


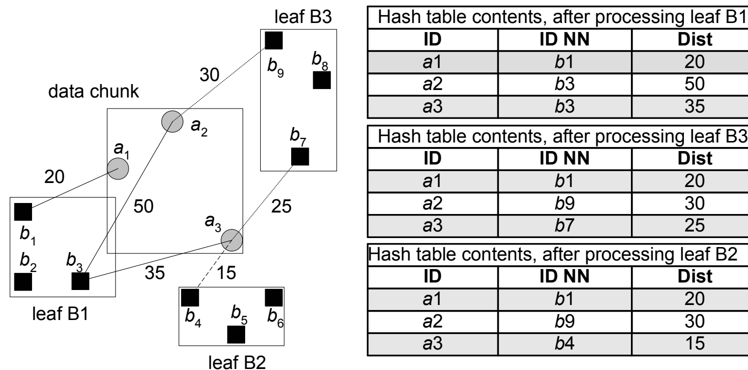

**Fig. 2.** Bird's eye view of the Probe-and-Search algorithm.

**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 which has been proposed in [20] uses batching operations in an aggressive way, in order 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 $\mathcal{D}_\mathcal{A}$ is dense in comparison to $\mathcal{D}_\mathcal{B}$. 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 ProbeMinHeap 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.




| Hash table contents, after processing leaf B1 | | |
|---|---|---|
| ID | ID NN | Dist |
| a1 | b1 | 20 |
| a2 | b3 | 50 |
| a3 | b3 | 35 |

| Hash table contents, after processing leaf B3 | | |
|---|---|---|
| ID | ID NN | Dist |
| a1 | b1 | 20 |
| a2 | b9 | 30 |
| a3 | b7 | 25 |

| Hash table contents, after processing leaf B2 | | |
|---|---|---|
| ID | ID NN | Dist |
| a1 | b1 | 20 |
| a2 | b9 | 30 |
| a3 | b4 | 15 |

**Fig. 3.** 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, in order 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 in order to determine if there is an object in the new leaf that change the best answers determined so far. Figure 3 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 Performance Study

### 4.1 Preliminaries

The algorithms PaS 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 used for the experimentation are taken from TIGER [18]. 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.

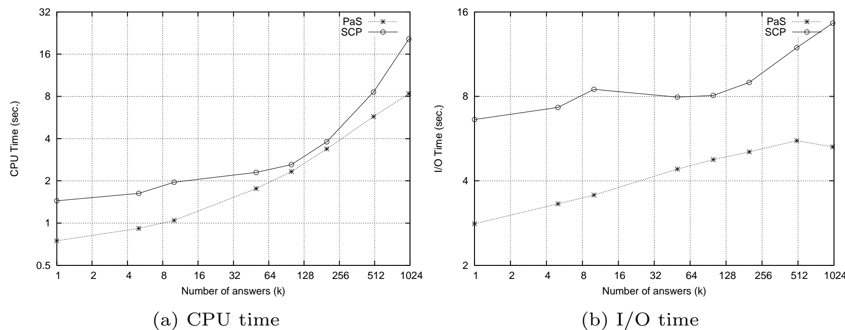
LA1 and LA2 are quite similar, having similar data distributions and populations. On the other hand, dataset CA shows completely different data distribution and population. The selection of these datasets have been performed in order to test the performance of the algorithms in cases where the primary 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, in order 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 and I/O time correspond to average values per query. 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.

## 4.2 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

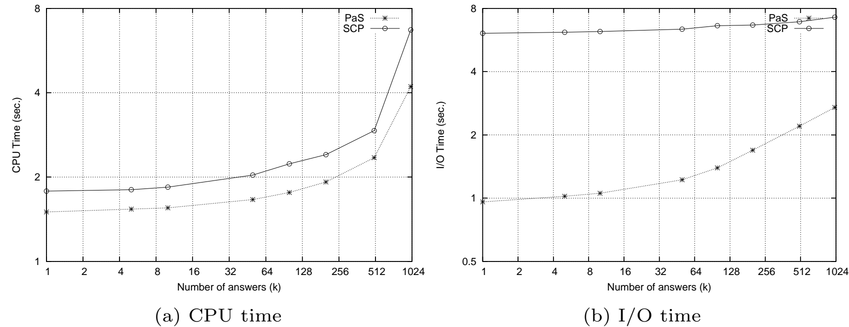
(b) I/O time

**Fig. 4.** CPU and I/O 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 4 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 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 4(b), the situation is quite different.

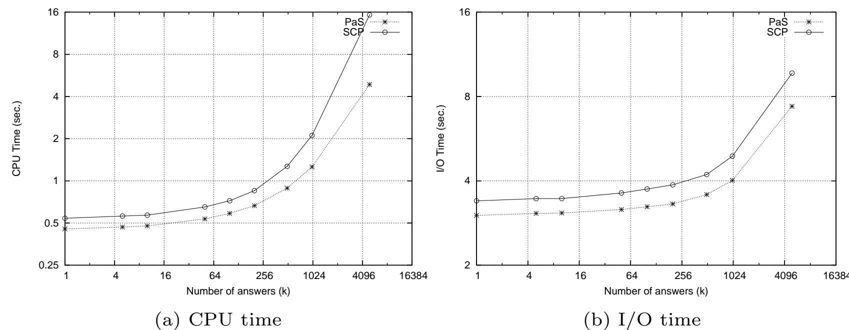
The I/O cost of PaS is maintained at low levels, especially for $k$ greater than 10. It is evident that PaS outperforms SCP Therefore, even for the extreme case when the reference dataset is significantly larger than the primary, the performance of PaS is reasonably good, whereas SCP is not able to maintain a good performance.




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

**Fig. 5.** CPU and I/O time vs $k$ for $\mathcal{D}_{\mathcal{A}}$=CA and $\mathcal{D}_{\mathcal{B}}$=LA1 (logarithmic scales).

Figure 5 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. 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 SCP.

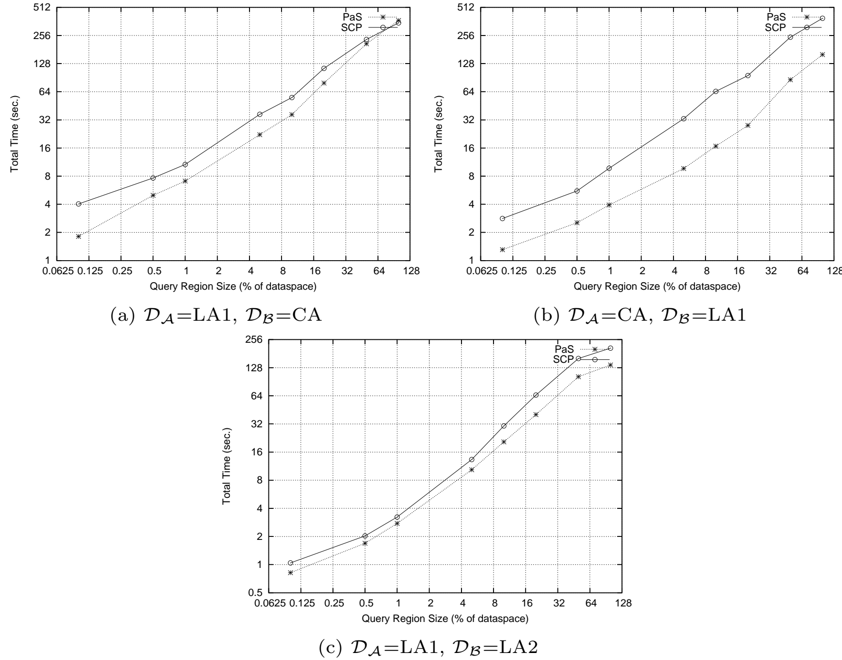



(a) CPU time (b) I/O time

**Fig. 6.** CPU and I/O time vs $k$ for $\mathcal{D}_{\mathcal{A}}$=LA1 and $\mathcal{D}_{\mathcal{B}}$=LA2 (logarithmic scales).

Figure 6 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. With respect to the overall performance of the methods, PaS shows the best performance.




(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

**Fig. 7.** Total time vs query region size (logarithmic scales).

Figure 7 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. Evidently, PaS shows the best performance and outperforms SCP significantly.

## 5 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 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), and
– the study of processing techniques when constraints are also applied to the reference dataset.

## 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. G.R. Hjaltason and H. Samet: “Incremental Distance Join Algorithms for Spatial Databases”, *Proceedings of ACM SIGMOD Conference*, pp.237-248, 1998.
10. G. Karypis, E.-H. Han, and V. Kumar: “Chameleon: Hierarchical Clustering Using Dynamic Modeling”, *Computer*, Vol.32, No.8, pp.68-75, 1999.
11. P. Mishra and M. H. Eich: “Join Processing in Relational Databases”, *ACM Computing Surveys*, Vol.24, No.1, 1992.
12. 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.
13. 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.
14. 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.
15. H. Shin, B. Moon and S. Lee: “Adaptive Multi-Stage Distance Join Processing”, *Proceedings of the ACM SIGMOD Conference*, pp.343-354, 2000.
16. 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.
17. 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.
18. TIGER/Line Files, 1994 Technical Documentation / prepared by the Bureau of the Census, Washington, DC, 1994.
19. 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.
20. 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.