An approximate algorithm for top-\(k\) closest pairs join query in large high dimensional data

Fabrizio Angiulli *, Clara Pizzuti

ICAR-CNR Instituto di Calcolo e Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche 87030 Rende, CS, Italy

Received 16 August 2004; accepted 16 August 2004
Available online 19 October 2004

Abstract

In this paper we present a novel approximate algorithm to calculate the top-\(k\) closest pairs join query of two large and high dimensional data sets. The algorithm has worst case time complexity \(O(d^2nk)\) and space complexity \(O(nd)\) and guarantees a solution within a \(O(d^{1+\frac{1}{t}})\) factor of the exact one, where \(t \in \{1, 2, \ldots, \infty\}\) denotes the Minkowski metrics \(L_t\) of interest and \(d\) the dimensionality. It makes use of the concept of space filling curve to establish an order between the points of the space and performs at most \(d + 1\) sorts and scans of the two data sets. During a scan, each point from one data set is compared with its closest points, according to the space filling curve order, in the other data set and points whose contribution to the solution has already been analyzed are detected and eliminated. Experimental results on real and synthetic data sets show that our algorithm behaves as an exact algorithm in low dimensional spaces; it is able to prune the entire (or a considerable fraction of the) data set even for high dimensions if certain separation conditions are satisfied; in any case it returns a solution within a small error to the exact one.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Technologies of databases; Applications of data and knowledge engineering; High dimensional data; \(k\)-closest pairs; Space filling curves

* Corresponding author. Tel.: +39 0984 831724; fax: +39 0984 839054.
E-mail addresses: angiulli@icar.cnr.it (F. Angiulli), pizzuti@icar.cnr.it (C. Pizzuti).
1. Introduction

Algorithms for extracting knowledge from large multidimensional data sets often involve queries relating data coming from multiple tables, a join query in database terms. A join query finds pairs of data points satisfying a particular property. Examples of join queries are similarity join query and top-k closest pairs join query (k-CPQ). Given two \(d\)-dimensional data sets \(B\) and \(R\), the former kind of query searches for the pairs of points \(p = (p_1, \ldots, p_d)\) from \(B\) and \(q = (q_1, \ldots, q_n)\) from \(R\) such that the distance between them is less than an input value \(\epsilon\), that is such that

\[
\text{dist}(p, q) = \left(\sum_{i=1}^{d} |p_i - q_i|^t\right)^{1/t} \leq \epsilon,
\]

where \(t \in \{1, 2, \ldots, \infty\}\) identifies the metrics of interest. Top-k closest pairs join query, instead, searches for the \(k\) pairs from the two data sets \(B\) and \(R\) having the \(k\) smallest distances between them. This last problem has been largely studied in computational geometry \([28,32]\) and it is known as the bichromatic \(k\) closest pairs problem \([4,18,21]\).

Recently, join queries received a lot of attention in the data mining field, mainly because algorithms for several knowledge discovery tasks relies on various kinds of similarity queries \([25,5]\), and in the information retrieval field to support query refinement \([26,27]\).

In this paper we deal with the top-k closest pairs query, though one of the two join operators could be used to find a suitable solution of the other. Approaches to solve similarity join query can be found in \([20,6,17]\).

Top-k closest pairs query is a useful operator in many data mining applications. Given two collections of objects, for example two repositories of documents or two files of patient analysis data, this operator permits to find the most similar objects coming from these collections, thus allowing to reveal the most related couples of objects contained in the groups. For example, consider a case where one data set represents the patients with specific pathologies, while the other collection contains patients under observation. A \(k\)-CPQ will discover which patients under observation are most similar to patients of the other group and thus likely to have an higher predisposition to develop a specific disease. This kind of information can be utilized for prevention purposes. In agglomerative hierarchical clustering algorithms using the single linkage strategy, the sum of the distances of the top-k closest pairs of objects from two partial clusters could be used as a more robust measure of similarity than the distance of the closest pair of objects. In spatial databases the importance of such an operator has been pointed out in \([10,34,11]\).

1.1. Related work

The (bichromatic) \(k\) closest pairs problem is a classical problem in computational geometry \([4,28,18,21]\). A comprehensive overview on algorithms regarding closest pairs and related problems can be found in \([28,32]\). When the dimension is two this problem can easily be solved by using the Voronoi diagrams \([32]\). However, when \(d > 2\) the problem becomes more difficult. Agarwal et al. \([1]\) gave an algorithm to compute the bichromatic closest pair by exploiting the relationship they showed to exist between the bichromatic closest pair problem and the minimum spanning tree problem.

Katoh and Iwano \([18]\) presented algorithms for finding the \(k\) closest/farthest bichromatic pairs, that iteratively reduces the search space by a half and uses higher order Voronoi diagrams. The authors found that their algorithms were very fast but they were defined only for \(d = 2\).
When the dimensionality of the data is low or fixed, such algorithms are very efficient. However, a thorough analysis reveals that their time requirements grow exponentially with the dimension.

In the context of spatial databases the top-k similarity closest pairs join problem was considered for the first time as a novel problem in [14,15]. In these works the authors introduced the spatial join operations distance join and distance semi-join and different incremental algorithms implemented using R-trees [9]. These algorithms provide an answer as soon as new data is available, and, after $k$ elements of the result have been obtained, to have the $(k + 1)$th it is not necessary to restart the method. They showed that these algorithms outperform non-incremental methods only if $k$ is small.

In [31] Shin and Moon enhanced the techniques proposed in [14] for the $k$-distance join and incremental distance join by using multistage and plane-sweep techniques.

In [10,11] Corral et al. presented a number of different algorithms for discovering the $k$ closest pairs between two spatial data sets stored in two different R-trees. Basically, these algorithms traverse the two trees and try to avoid the comparison of the points stored in all the possible pairs of nodes by using several strategies. In particular they propose a pruning heuristic and two updating strategies for minimizing the pruning distance and use them to design branch-and-bound algorithms that solve the $k$ closest pairs query problem. They reported experiments on real and uniformly distributed two-dimensional data, and studied the scalability of the methods when the data set size and the number of pairs required increases.

In [34] Yang and Lin proposed a new index structure, the b-Rdnn tree, to solve different join queries, as the $k$ closest pairs query. The b-Rdnn tree is an R-tree like data structure augmented with pre-computed nearest-neighbor information w.r.t. another b-Rdnn tree, such as the distance of each point from its nearest neighbor in the other tree. Thus, when a point is inserted in one of a pair of the b-Rdnn trees, a nearest-neighbor query must be performed. They showed the superiority of their method w.r.t. the two mentioned above when the two data sets present a high degree of overlap, where the overlap of two data sets is related to the overlapping area of the bounding rectangles of the two data sets, and thus to the number of overlapping bounding rectangles associated with the nodes of the R-trees storing the two data sets. In particular, they reported a poor performance of the two previous algorithms when the overlap is 100%. They presented experimental results up to four-dimensional data sets.

We note that, although these algorithms can be used in any dimension, when the dimensionality increases, the number of intersecting bounding rectangles associated with the nodes of the R-trees of two overlapping data sets grows, while nearest-neighbor queries degenerate in a visit of all the nodes of the R-tree. This means that, when the dimensionality is a parameter, a naive nested-loop algorithm enumerating all the possible pairs of points from the two data sets can outperform existing methods in the computation of the top-$k$ closest pairs join, even for very small values of the dimensionality. Then, it makes sense to search for an approximate solution in a reasonably amount of time. In [23,2] two approximate algorithms for the computation of the top-$k$ closest pairs self-join of an input data set are described.

1.2. Contribution

In this paper we present a novel approximate algorithm to calculate the top-$k$ closest pairs join, according to one of the Minkowski metrics, of two large and high dimensional data sets. The
algorithm is an extension of that presented in [2] to efficiently manage points coming from two distinct data sets. The algorithm is particularly suitable in all those applications in which the total number \( n \) of points from the two data sets overcomes the product \( dk \), where \( d \) denotes the dimensionality of the data sets.

The algorithm has worst case time complexity \( O(d^2nk) \) and space complexity \( O(nd) \) and guarantees a solution within a \( O(d^{1+t}) \) factor of the exact one, where \( t \in \{1, 2, \ldots, \infty\} \) denotes the Minkowski metrics \( L_t \) of interest.

It makes use of the concept of space filling curve to establish an order between the points of the space. Space filling curves [29] are mappings from the \( d \)-dimensional space \( D = [0,1]^d \), to the one-dimensional space \( I = [0,1] \) having the property that if two points from the unit interval \( I \) are close then the corresponding images are close too in the hypercube \( D \). The reverse statement, however, is not true because two close points in \( D \) can have non-close inverse images in \( I \). The ordering induced by the space filling curve thus allows to obtain the approximate nearest neighbors of each point by just considering its predecessors and its successors along the unit interval. To guarantee a solution with a good degree of approximation, the algorithm performs at most \( d+1 \) sorts and scans of shifted versions of the two data sets.

During a scan, each point from one data set is compared with its closest points, according to the space filling curve order, in the other data set, and the \( k \) closest pairs of points encountered are stored in a priority queue like data structure. This corresponds to exploring a certain region of the search space containing the point. If this region contains entirely the \( d \)-dimensional neighborhood of radius \( r \) of the point, with \( r \) the distance associated with the \( k \)th closest pair stored in the priority queue, then the point is deleted and it is not considered any more in the following iterations. Thus, if all the points are pruned during a certain iteration, then the algorithm stops reporting the exact solution.

We show that the pruning ability of the algorithm is related to the nearest neighbor distribution of the two data sets. Experimental results on real (up to 581,012 points in 60-dimensional) and synthetic (up to 200,000 points in 50-dimensional) data sets show that our algorithm:

- behaves as an exact algorithm, i.e. it prunes all the points during its execution, in low dimensional spaces (up to 6–7 dimensions);
- is able to prune the entire (or a considerable fraction of the) data set even when high dimensional data sets, respecting certain separation conditions, are considered;
- in any case returns a solution within a small error (up to 1% in the experiments reported) to the exact one.

A preliminary version of this work appeared in [3]. The rest of the paper is organized as follows. In the following section we give some definitions that will be used throughout the paper. In Section 3 we give a detailed description of the approximate algorithm. In Section 4 we describe experimental results obtained with our algorithm both on real and synthetic data and we draw our conclusions.
2. Preliminaries

In this section we give preliminary definitions, notations and properties used in the paper.

Space filling curves [29] are mappings from the $d$-dimensional space $D = [0,1]^d$ (or any other hypercube $[0,\ell]^d$ with $\ell > 0$), the original space, to the one-dimensional space $I = [0,1]$, the embedded space. The construction of a space filling curve can be viewed as a recursive partitioning of the original space in $2^{hd}$ equal-sized sub-hypercubes among which a particular order is established. The positive integer $h$ identifies the resolution, or approximation, of the partitioning. This order induces a mapping between one of the $2^{hd}$ equally-sized sub-intervals of $I$ and the coordinates of the $d$-dimensional points. Fig. 1 shows the first four approximations of the $Z$-order space filling curve.

Given a set of points $S$ from $D$ and a point $p$ in $S$ we denote by SFC($p$) the Space Filling Curve (SFC) key of $p$, i.e. the value of the sub-interval of $I$ associated with $p$ by the space filling curve mapping, and by SFC$_p(p,m)$ and SFC$_p(p,m)$ the $m$th predecessor and successor of $p$ in $S$ according to their SFC key. Space filling curves have been studied and used in several fields [12,13,16,24,33].

A useful property of such a mapping is that if two points from the unit interval $I$ are close then the corresponding images are close too in the hypercube $D$. The reverse statement, however, is not true because two close points in $D$ can have non-close inverse images in $I$. This implies that the reduction of dimensionality from $d$ to one can provoke the loss of the property of nearness. In order to preserve the closeness property, approaches based on the translation and/or rotation of the hypercube $D$ have been proposed [30,23,33,22]. Such approaches assure the maintenance of the closeness of two $d$-dimensional points, within some factor, when they are transformed into one-dimensional points.

In this work we use the sequence $0, 1/d+1, \ldots, d/d+1$ of shifts along the main diagonal of $D$, defined in [8], to preserve the closeness property. Without loss of generality, we assume that the two given

![Fig. 1. The first four approximations of the two dimensional Z-order space filling curve.](image-url)
data sets have been normalized so that they are constituted by points in $[0, 1]^d$. The original and the shifted data points thus belong to $[0, 2)^d$, hence in the following we consider data sets on $[0, 2)^d$.

**Definition 2.1.** An $r$-region is an open ended hypercube in $[0, 2)^d$ with side length of $r = 2^{1-s}$ having the form $\prod_{i=0}^{d-1} [a_i r, (a_i + 1)r)$, where each $a_i$, $0 \leq i < d$, and $s$ are in $\mathbb{N}$. The order of an $r$-region of side $r$ is the quantity $-\log_2 r$.

Notice that every $r$-region contains one and only one contiguous segment of a space filling curve.

**Definition 2.2.** Given two distinct points $p$ and $q$ we denote by $MinReg(p, q)$ the order of the smallest $r$-region containing both $p$ and $q$.

The function $MinReg$ can be calculated in time $O(d)$ working on the bit-string representation of $SFC(p)$ and $SFC(q)$. We note that the value $MinReg(p, q) + 1$ represents the order of the greatest $r$-region containing $p$ but not $q$.

**Definition 2.3.** Let $p$ be a point, and let $r$ be the side of a $r$-region. We denote by $MinDist(p, r)$ the value $\min_{i=1}^{d} \{ \min \{ \text{mod}(p_i, r), r - \text{mod}(p_i, r) \} \}$ where $\text{mod}(x, r) = x - \lfloor x/r \rfloor r$, and $p_i$ denotes the value of $p$ along the $i$th coordinate.

Hence $MinDist(p, r)$ is the perpendicular distance from the point $p$ to the nearest face of the $r$-region of side $r$ containing $p$, i.e. a lower bound to the distance between $p$ and a point lying out of this region.

Given a set of points $S$, a point $p$ of $S$ and the set $I$ of the closest points of $S$ according to the space filling curve order, the following property permits to compute a lower bound to the distance from $p$ to any other point in $S - I$.

**Proposition 2.1.** Given a set of points $S$, a point $p$ of $S$, and two integers $a$ and $b$, let $I = \{ x_i = SFC_p(p, i) | 1 \leq i \leq a + 1 \} \cup \{ y_j = SFC_p(p, j) | 1 \leq j \leq b + 1 \}$, let $s_n$ be the maximum between $MinReg(p, x_{a+1})$ and $MinReg(p, y_{b+1})$, let $r_n = MinDist(p, 2^{-(s_n+1)})$, and let $S_n = \{ q \in I | \text{dist}(p, q) \leq r_n \}$. Then the distance between $p$ and any point in $S - S_n$ is greater than $r_n$.  

![Fig. 2. An example of application of Proposition 2.1.](image-url)
Proof. First, we note that, for each $r$-region, the intersection of the space filling curve with the $r$-region results in a connected segment of the curve. Hence, to reach the points $SFC_p(p, m)$ and $SFC_c(p, m)$ from $p$ following the curve, we surely walk through the entire $r$-region of side $s_n$ containing $p$. As the distance from $p$ to the nearest face of its $s_n$-region is $r_n$, then the $d$-dimensional neighborhood of radius $r_n$ of $p$ is entirely contained in that region. It follows that the points in $S_n$ are all and the only points of $S$ placed at a distance not greater than $r_n$ from $p$. Obviously, the $(|S_n| + 1)$th nearest-neighbor of $p$ has a distance greater that $r_n$ from $p$. □

Fig. 2 shows an example of application of Proposition 2.1, and shows the $s_n$-region, the distance $r_n$, and the neighborhood of radius $r_n$ of $p$, for $a = b = 2$.

3. Algorithm

In the following the two data sets from which the $k$ pairs having the smallest distance are to be extracted are denoted by $B$, named the blue data set, having size $n_B$, and $R$, named the red data set, having size $n_R$, and we call blue (red, respectively) point a point coming from the blue (red, respectively) data set. Before starting with the description of the algorithm, we define the concept of point feature. A point feature is a record containing the fields point, col, key, lev, succelev, and $lb$, where: point is a point from the blue or red data set, col denotes the data set from which the point comes, i.e. the blue data set (in this case col is set to 0) or the red data set (in this case it is set to 1), key is the SFC key of point, lev and succelev are the order of an $r$-region, succ is an integer, and $lb$ is a distance representing the radius of a neighborhood of point.

The algorithm employs two main data structures: an array of point features $f$, having size $n = n_B + n_R$, in which both the blue and red points are stored, and a priority queue $CPQ$ of size $k$, whose elements are triples of the form $\langle p, q, \delta \rangle$, where $p$ is a blue point, $q$ is a red point, and $\delta$ is the distance dist($p, q$) from $p$ to $q$. Every triple contained in $CPQ$ represents one of the nearest pairs met during the execution of the algorithm. In the following we will denote with $Pairs(CPQ)$ the set $\{\langle p, q, \delta \rangle | \langle p, q, \delta \rangle \in CPQ\}$, with $Pairs(f)$ the set $\{\langle f[i].point, f[j].point \rangle | 1 \leq i, j \leq n, f[i].col = 0, f[j].col = 1\}$, and with $J_k(B, R)$ the $k$ closest pairs in the set $B \times R$.

To manage the priority queue $CPQ$ the procedure $CPQUpdate$ and the function $CPQMax$ are employed. The procedure $CPQUpdate(CPQ, p, q, \delta)$ modifies $CPQ$ as follows: unless the triple $\langle p, q, \delta \rangle$ is already present in $CPQ$, if the size of $CPQ$ is less than $k$, then the triple $\langle p, q, \delta \rangle$ is inserted in $CPQ$. Otherwise, if $\delta$ is less than the maximum distance associated with a triple in $CPQ$, than this triple is erased from $CPQ$ and the triple $\langle p, q, \delta \rangle$ is inserted in $CPQ$. The function $CPQMax(CPQ)$ returns the greatest distance between two pairs of points from $CPQ$, if $CPQ$ contains $k$ elements, and $\infty$, if $CPQ$ contains less than $k$ elements. Thus, its value is a lower bound to the distance of the $k$th closest pair of points from the two data sets.

Let $\langle p, q \rangle$ be a pair in $B \times R$. We say that $\langle p, q \rangle$ has been processed, if the function with parameters $CPQUpdate(CPQ, p, q, \text{dist}(p, q))$ has been executed by the algorithm. In particular, we point out that the field $lb$ of each point feature $f[i]$ stored in $f$ satisfies the following property.

Proposition 3.1. Let $p = f[i].point$ be a blue (red, respectively) point, then, for each red (blue, respectively) point $q$ such that $\text{dist}(p, q) \leq f[i].lb$, either (1) the pair $\langle p, q \rangle$ has been processed, or (2) the pair $\langle p, q \rangle$ does not belong to $J_k(B, R)$. 


The algorithm Approx-SFC-CP-Join, shown in Fig. 3 (left) receives in input the two data sets \( B \) and \( R \), and the number \( k \) of top closest pairs from \( B \times R \) to find. After the initialization of the point feature array \( f \), the procedure Normalize scales the two data sets so that they fit into the unit hypercube. The priority queue \( CPQ \) is built and the main cycle of the algorithm, consisting of at most \( d + 1 \) steps, starts. During each step the algorithm works on a shifted version of the original data set. The shift value is stored in the global variable \( \sigma \) and is equal to \( \frac{l}{d+1} \), where \( l \) is the iteration number, \( 0 \leq l \leq d \). We explain the single operations performed during each step of the main cycle.

**Linearize.** The procedure Linearize calculates the SFC key of each point \( f[i] \cdot point + \sigma \), stores this value in \( f[i] \cdot key \), and sorts the array w.r.t. the \( key \) field. Thus, it performs the SFC mapping of a shifted version of the two input data sets. After sorting, the procedure Linearize sets the values of the fields \( lev \), \( succ \), and \( succlev \) of each point feature \( f[i] \) in the following way:

- the value \( f[i] \cdot lev \) represents the order of the smallest \( r \)-region containing both \( f[i] \cdot point + \sigma \) and \( f[i + 1] \cdot point + \sigma \), i.e. it is set to \( \text{MinReg}(f[i] \cdot point + \sigma, f[i + 1] \cdot points + \sigma) \);
- if \( f[i] \cdot point \) is a blue (red, respectively) point, then \( f[f[i] \cdot succ] \cdot point \) is the first red (blue, respectively) point following \( f[i] \cdot point \), according to the order induced by the SFC keys above calculated;

```plaintext

For (i = 1; i ≤ n; i++) { // main cycle
    c = f[i] \cdot col;
    last[c] \cdot [laststop[c] + 1] = i;
    laststop[c] = (laststop[c] + 1) \% (m + 1);
    lastend[c]++;
    if (f[i] \cdot lb ≤ CPQMax(CPQ)) {
        j = f[i] \cdot succ;
        lev = GetSuccLev(i, j);
        count = 0;
        stop = false;
        while (count < m \&\& !stop) { //inner cycle
            count++;
            if (f[j] \cdot lb < CPQMax(CPQ)) {
                \( \delta = \text{dist}(f[i] \cdot point + \sigma, f[j] \cdot point + \sigma) \);
                CPQUpdate(CPQ, f[i] \cdot point, f[j] \cdot point, \delta);
                levnext = GetSuccLev(j, GetSucc(j));
                if (levnext < lev) {
                    \( \delta = \text{MinDist}(f[i] \cdot point + \sigma, 2^{-(lev+1)}) \);
                    if (\( \delta > CPQMax(CPQ) \)) stop = true;
                    j = GetSucc(j);
                }
            }
        }
        c = 1 - f[i] \cdot col; // updates the field lb
        if (lastend[c] > m) lev =
            \( \max(\text{MinReg}(f[last[c] \cdot laststop[c] + points + \sigma, f[l] \cdot points + \sigma), lev) \);
        if (lev ≥ 0) f[i] \cdot lb = \max(f[i] \cdot lb, MinDist(f[i] \cdot point + \sigma, 2^{-(lev+1)}));
        else f[i] \cdot lb = \infty;
    }
}
```

Fig. 3. Algorithm Approx-SFC-CP-Join (left) and procedure Approx-Join (right).
the value \( f[i].succelev \) represents the order of the smallest \( r \)-region containing both \( f[i].point + \sigma \) and \( f[f[i].succelev].point + \sigma \).

We note that, given a sequence of points \( p_1, \ldots, p_m \) ordered according to their SFC keys, then for each pair \( p_i, p_j, 1 \leq i < j \leq m \), it holds that \( \text{MinReg}(p_i, p_j) = \min_{k \leq j-1} \text{MinReg}(p_k, p_{k+1}) \). Thus, instead of calculating \( f[i].succelev \) by calling the function \( \text{MinReg} \) with the parameters \( f[i].point + \sigma \) and \( f[f[i].succelev].point + \sigma \), we can compute \( f[i].succelev \) in the following more efficient way: for \( i = n, \ldots, 1 \) (scan the array \( f \) in the reverse order), if \( f[i].col \) is different from \( f[i+1].col \), then set \( f[i].succelev \) to \( f[i].lev \), otherwise, set \( f[i].succelev \) to \( \min(f[i].lev, f[i+1].succelev) \).

**Approx-Join.** The procedure Approx-Join is reported in Fig. 3 (right). This procedure has the parameter \( m \), whose value is set to \( \min(n, \lceil \frac{m}{k} \rceil) \), where \( n_0 \) denotes the sum of the blue and red input data set sizes, while \( n \) is the current size of the array \( f \), that is the number of non-pruned points. Thus \( m \) is inversely proportional to \( n \). The value of \( m \) is the size of the one-dimensional neighborhood, according to the space filling order, that, for each non-pruned point, will be considered in the procedure. Thus, increasing the value of \( m \) is possible to examine a wider neighborhood of each point, maintaining constant the number of distance computations per iteration. Note that the maximum value \( m \) can assume is \( \sqrt{n_0} k \), which is reached when \( n = \frac{n_0}{n} k \).

This procedure performs a sequential scan of the array \( f \). To store the last \( m + 1 \) blue and red points encountered during the main cycle of the procedure, the arrays \( \text{last}, \text{lasttop}, \text{lastcnt} \) are employed. In particular, \( \text{last} \) is a \( 2 \times (m + 1) \) array (with row indices 0 and 1, and column indices from 0 to \( m \)), while \( \text{lasttop} \) and \( \text{lastcnt} \) are two arrays of two elements (having indices 0 and 1, and whose element values are initially set to zero). These arrays are managed so that \( \text{last}[0][\text{lasttop}[0]] \), \( \text{last}[1][\text{lasttop}[1]] \), respectively stores the index of the element of the array \( f \) containing the \( (m + 1) \)th blue (red, respectively) point preceding the current point \( f[i].point \).

For each point feature \( f[i] \), the distances among the blue (red, respectively) point \( f[i].point \) and the first \( m \) red (blue, respectively) points \( f[j_1].point, \ldots, f[j_m].point \) \((i < j_1 < \cdots < j_m \leq n)\) following \( f[i].point \) are calculated in the inner cycle. These points can be efficiently retrieved using the field \( \text{succelev} \) and the function \( \text{GetSucc} \) defined as follows: let \( \text{GetSucc}(j) \) be equal to \( j + 1 \), if \( f[j].col = f[i+1].col \), and equal to \( f[j+1].succelev \), otherwise. Then \( j_1 \) is equal to \( f[i].succelev \), while each \( j_{u+1}, 1 \leq u < m \), is equal to \( \text{GetSucc}(j_u) \).

We note that, according to Proposition 3.1, the procedure Approx-Join considers only the points \( f[u].point, 1 \leq u \leq n \), such that \( f[u].lb \leq \text{CPQMax}(CPQ) \).

In practice, the inner cycle could stop before comparing the point \( f[i].point \) with all the points \( f[j_1].point, \ldots, f[j_m].point \). Indeed, this cycle terminates after \( u' \in \{1, \ldots, m\} \) iterations, with \( u' < m \) if the minimum distance between \( f[i].point + \sigma \) and the nearest face of the greatest \( r \)-region containing this point but not \( f[j_{u' + 1}].point + \sigma \), given by \( \text{MinDist}(f[i].point + \sigma, 2^{-k} \text{MinReg}(f[i].point + \sigma, f[j_{u' + 1}].point + \sigma + 1)) \), is greater than \( \text{CPQMax}(CPQ) \). We note that, in this case, none of the pairs neglected, i.e. none of the pairs \( \langle f[i].point, f[j_{u' + 1}].point \rangle, \langle f[i].point, f[j_{u' + 1}].point \rangle, \ldots, \langle f[i].point, f[j_m].point \rangle \), can belong to \( J_k(B, R) \), as the distances between these pairs of points are certainly greater than the distance of the \( k \)th closest pair of points from the two data sets. The function \( \text{GetSuccLev} \) is defined as follows: let \( \text{GetSuccLev}(i, j) \) be equal to \( f[i].lev \), if \( i + 1 = j \), and equal to \( \min(f[i].lev, f[i+1].succelev) \), otherwise. Then \( \text{GetSuccLev}(j_{u-1}, j_u), 1 \leq u \leq m \), is the value \( \text{MinReg}(f[j_{u-1}].point + \sigma, f[j_u].point + \sigma) \) (assume that \( j_0 = i \)). Consequently, the value
MinReg(f[i].points + σf[j], point + σ), 1 ≤ u ≤ m, can be obtained taking the minimum over GetSuccLev(0, j), ..., GetSuccLev(j - 1, j).

At the end of the inner cycle, the field f[i].lb is updated. Let f[i].point be a blue (red, respectively) point. We note that at the end of the inner cycle, j stores the index of the red (blue, respectively) point following the last red (blue, respectively) point compared with f[i].point. Furthermore, we note that f[last(1)[lasttop(1)]].point + σ(f[last(0)[lasttop(0)]].point + σ, respectively) is the (m + 1)th red (blue, respectively) point preceding f[i].point + σ. Hence, we can set x_{a+1} to this point and y_{b+1} to f[j].point + σ, and then apply Proposition 2.1 to obtain the radius \( r_n \). If the value \( r_n \) so obtained is greater than f[i].lb, then f[i].lb is set to \( r_n \). See Proposition 3.2 for the proof that this update preserves the property defined in Proposition 3.1.

Prune. The procedure Prune deletes from f all the features f[i] such that f[i].lb > CPQMax(CPQ) and updates consequently the values n, n_B and n_R.

The algorithm stops when n_B or n_R is equal to zero, or after at most d + 1 iterations. This terminates the description of the algorithm.

3.1. Disk based algorithm

Basically the disk based implementation of the Approx-SFC-CP-Join algorithm has the same structure of its main memory based counterpart. The main difference is that the array f is stored in a sequential file \( F \) on disk. Thus, the procedure Linearize performs an external sort of F [19]. The procedure Approx-Join maintains a contiguous portion of F in a main memory circular buffer B of size b. When a point feature f[j], subsequent to the current point feature f[i], is requested through the function GetSucc, if f[j] is not present in the buffer B, then an auxiliary buffer \( B_a \) of size \( b_a \ll b \) is employed to temporarily store a contiguous portion of F, subsequent to that stored in B, containing f[j]. Both the queue CPQ and the array last are stored in main memory. In particular, this time last stores the point feature, nor its index (or record number). Indeed, when the record \( \text{rec} \) of F storing the feature f[last(c)[lasttop(c)]} (c \in \{0, 1\}) must be used to update the field lb of the current point feature f[i], at the end of the main cycle of Approx-Join, rec could no longer be present in the buffer B. Finally, the procedure Prune deletes the records of F storing the point features f[i] such that f[i].lb > CPQMax(CPQ).

3.2. Algorithm properties

Now we state the main properties of the algorithm.

Proposition 3.2. During the execution of the algorithm Approx-SFC-CP-Join the following holds: \( J_k(B, R) \subseteq \text{Pairs}(CPQ) \cup \text{Pairs}(f) \).

Proof. We point out that the value of each f[i].lb is initially set to 0. Consider the first iteration (I = 0) of the algorithm. Let f[i].point be a blue (red, respectively) point. At the end of the \( i \)th cycle of Approx-Join, the value f[i].lb is updated with the radius \( r_n \) of the greatest d-dimensional neighborhood of f[i].point entirely contained in the greatest r-region containing p but not the (m + 1)th red (blue, respectively) point preceding p nor the jth red (blue, respectively) point following p. We note that the pairs \( \langle p, q \rangle \), where q is one of the j−1 red (blue, respectively) points following p, have
been processed in the current cycle of \textit{Approx-Join}. As regards the pairs $\langle p, q \rangle$, where $q$ is one of the $m$ red (blue, respectively) points preceding $p$, it could be that some of them have not been processed, since $q$ was not compared with all the $m$ blue (red, respectively) points following it. But, in this case $\text{dist}(p, q) > \text{CPQMax}(CPQ)$ and $\langle p, q \rangle$ does not belong to $J_k(B, R)$. Thus, this update preserves the property of Proposition 3.1.

Once the scan of $f$ is terminated, the procedure \textit{Prune} deletes from $f$ the point features $f[i]$ having $lb$ greater than $\text{CPQMax}(CPQ)$. In CPQ there are the top-$k$ closest pairs among those examined. Thus, if $f[i]$ is eliminated by \textit{Prune}, then there does not exist a pair of points from $B \times R$ including $f[i].point$ that, at the same time, belongs to $J_k(B, R)$ and was not examined.

Consider now a generic subsequent iteration ($l > 0$). Same considerations apply, but this time some points from the two input data sets could be missing. From the previous reasoning, all the pairs from $B \times R$ containing a point whose feature is not in $f$ and potentially belonging to the solution set $J_k(B, R)$, were examined in the previous iterations. Thus the property is guaranteed.  

The above property proves the correctness of the algorithm. Furthermore, it establishes that the pruning operated by the algorithm is lossless, that is the remaining points along with the approximate solution found can be used for the computation of the exact solution.

In the following with the notation $n^*_{B}, n^*_{R}, f^*, \text{CPQ}^*$ we refer, respectively, to the value of the variables $n_{B}, n_{R}, f$ and CPQ at the end of the algorithm. The following proposition states an important result regarding the algorithm, i.e. when the array $f^*$ is empty or contains only blue (red, respectively) points than we can assert that the solution returned is the exact one.

\textbf{Proposition 3.3.} $n^*_{B} = 0$ or $n^*_{R} = 0$ implies that $J_k(B, R) = \text{Pairs}(\text{CPQ}^*)$.

\textbf{Proof.} The result follows immediately from Proposition 3.2.  

We denote by $\epsilon_d$ the value $2d^2(2d + 1)$, and by $\delta_k$ the distance between the two points composing the $k$th closest pair of points in $B \times R$. As we use the family $0, \frac{1}{d+1}, \frac{1}{d+1}, \ldots, \frac{1}{d+1}$ of shifts defined in [8], we are able to state an upper bound for the approximation error of the algorithm similar to that stated in [23,2], that employed the same family of shifts. The following result is from [7].

\textbf{Lemma 3.1.} Suppose $d$ is even. Let $v^{(j)} = (1/d+1, \ldots, 1/d+1) \in \mathbb{R}^d$. Then, for any point $p \in \mathbb{R}^d$ and $r = 2^{-s}(s \in \mathbb{N})$, there exists $j \in \{0, \ldots, d\}$ such that $p + v^{(j)}$ is $(\frac{1}{2d+2})$-central in its $r$-region.

The previous lemma states that if we shift a point $p$ of $\mathbb{R}^d$ at most $d + 1$ times in a particular manner, i.e. if we consider the set of points \{p + v^{(0)}, \ldots, p + v^{(d)}\}, than, in at least one of these shifts, this point must become sufficiently central in an $r$-region, for each admissible value of $r$.

\textbf{Proposition 3.4.} $\text{CPQMax}(\text{CPQ}^*) \leq \epsilon_d \delta_k$.

\textbf{Proof.} Let \{\{p_1, q_1\}, \ldots, \{p_k, q_k\}\} be the set $J_k(B, R)$, and let $\delta_i = \text{dist}(p_i, q_i)$, for $i = 1, \ldots, k$. From Lemma 3.1 it follows that, for each $i = 1, \ldots, k$, there exists an $r_i$-region of side $\frac{1}{4d+4} \leq \delta_i < \frac{1}{2d+2}$ (this inequality defines the greatest $r_i$-region satisfying the following property) and an integer $j_i \in \{0, \ldots, d\}$ such that $p_i + v^{(j_i)}$ is $(\frac{1}{2d+2})$-central in the $r_i$-region. Let $p'_i = p_i + v^{(j_i)}$ and $q'_i = q_i + v^{(j_i)}$. As a consequence the $d$-dimensional neighborhood of $p'_i$ of radius $\delta_i$ is entirely contained in that region, and both $p'_i$ and $q'_i$ belong to the $r_i$-region.
Notice that, as $p'_i$ is $(\frac{1}{2d+2})$-central in the $r_r$-region, this implies that the distance $\delta$ from $p'_i$ and each point belonging to its $r_r$-region is at most $d^l (r_i - \frac{r_i}{2d+2})$ i.e. $\delta \leq d^l \frac{2d+1}{2d+2} r_i$.

Let $r_{\text{max}} = \max\{r_1, \ldots, r_k\}$. If \( \text{CPQMax}(\text{CPQ}) \leq d^l \frac{2d+1}{2d+2} r_{\text{max}} \) then each triple $\langle p, q, \delta \rangle$ in $\text{CPQ}^*$ is such that $\delta \leq d^l \frac{2d+1}{2d+2} (4d + 4) \delta_k \leq \epsilon_d \delta_k$. Now we show that $\text{CPQMax}(\text{CPQ}^*) \leq d^l \frac{2d+1}{2d+2} r_{\text{max}}$. If every pair $\langle p_i, q_i \rangle$ is considered for insertion into $\text{CPQ}$ then the result follows. Otherwise, there exists a pair $\langle p_i, q_i \rangle$ that was not examined in any iteration. In this case, from Proposition 3.2 it follows that $\langle p_i, q_i \rangle$ is in $\text{Pairs}(f^*)$, thus the point features associated with $p_i$ and $q_i$ certainly occurs in $f$ during the entire execution of the algorithm. Hence $q'_i$ is further than $m$ positions from $p'_i$ (w.l.o.g. assume that $q'_i$ comes after $p'_i$) along the order induced by the space filling curve in the $j$th iteration. For the properties of the space filling curves, the $m$ pairs examined in the $j$th iteration belong to the same $r_r$-region containing both $p'_i$ and $q'_i$, hence have distance less or equal than $d^l \frac{2d+1}{2d+2} r_i$.

As the algorithm determines at least $k$ pairs having such property (at least one for each closest pair), then the result follows. \( \square \)

Thus, the algorithm guarantees an $\mathcal{O}(d^{1+\frac{t}{2}})$ approximation to the solution, where $t = 1, 2, \ldots, \infty$ denotes the $L_t$ metrics used to compute distances.

Now we define the worst case condition allowing a point to be pruned from the input data set by the algorithm.

**Proposition 3.5.** The array $f^*$ does not contain at least those points $p$ of $B$ ($R$, respectively) such that $\delta > \epsilon_d^2 \delta_k$, where $\delta$ is the distance between $p$ and its nearest neighbor in $R$ ($B$, respectively).

**Proof.** Let $p$ be a blue (red, respectively) point and let $\delta$ be the distance between $p$ and its nearest red (blue, respectively) point $q$. Let $r_0$ be the side of the greatest $r$-region satisfying the inequality $d^l \frac{2d+1}{2d+2} r_0 < \delta$. Let $r_0 = 2^{-s_0} (s_0 \in \mathbb{N})$, then $s_0$ is such that $\log_2 \left( \frac{d^l}{\delta} \frac{2d+1}{2d+2} \right) + 1 \geq s_0 > \log_2 \left( \frac{d^l}{\delta} \frac{2d+1}{2d+2} \right)$. From Lemma 3.1 it follows that there exists $j \in \{0, \ldots, d\}$ such that $p + v^{(j)}$ is $(\frac{1}{2d+2})$-central in the $2^{-s_0}$-region. When the above condition occurs, the nearest-neighbor of $p$ is certainly out of its $r$-region of side $2^{-s_0}$. Thus, in the worst case the value of $f[i].lb$ is $2^{-s_0} d^l \frac{2d+1}{2d+2}$, where $f[i]$ is the feature s.t. $f[i].point = p$.

We know from Proposition 3.4 that the worst case approximation for the value of $\delta_k$ given by the Approx-SFC-CP-Join algorithm is $2d^l (2d + 1) \delta_k$. Hence $f[i].lb > \text{CPQMax}(\text{CPQ})$ surely when
\[
\frac{\delta(2d+2)}{2d^l (2d+1)} > 2d^l (2d + 1) \delta_k \text{ i.e. for } \delta > 4d^l (2d + 1)^2 \delta_k. \quad \square
\]

Thus, let $\delta$ be the distribution of the nearest-neighbor distances of the points of $B$ ($R$, respectively) w.r.t. the points of $R$ ($B$, respectively). Proposition 3.5 asserts that the ability of the algorithm to prune points increases when the distribution $\delta$ accumulates around and above the value $\epsilon_d^2 \delta_k$. Experimental results confirm that in practice the algorithm is able to prune points having distance to their nearest neighbor significantly less than the worst case value above stated.

### 3.3. Time and space cost analysis

Now we give time and space cost analysis of the algorithm. We start with the time complexity. Let $n = n_B + n_R$. The procedure Linearize requires $\mathcal{O}(dn \log n)$ time, while the procedure Prune
requires $O(n)$ time. The procedure Approx-Join requires $O(k(d + \log k))$ time to execute the inner cycle, i.e. $O(d)$ time to calculate the distance between two points and time $O(\log k)$ to update the queue $CPQ$, and $O(d)$ time to update the field $lb$, thus in total $O(nk(d + \log k))$ time. Thus, in the worst case the algorithm runs in $O(dn \log n + nk(d + \log k))$ time. W.l.o.g., if we assume that $k \geq \log n$ and $d \geq \log k$, then the time complexity can be simplified in $O((n_B + n_R)d^2k)$. We note that the naive nested-loop, or brute force, algorithm enumerating all the possible pairs in the data set requires $O(n_Bn_R(d + \log k))$ time to find the set $J_k(B,R)$. Thus, if we assume that $O(n_B) = O(n_R)$, then the algorithm is particularly suitable in all the applications in which $n$ overcomes the product $dk$. As an example, if we search for the top 100 closest pairs in a 100-dimensional data set composed by one million of points, using the Approx-SFC-Top-Join algorithm we expect to obtain the approximate solution, together with the reduced data sets, with time savings of at least two order of magnitude with respect to the brute force approach. Furthermore, we note that as the point features considered in the current iteration could be a proper subset of those considered in the preceding iteration, the effective execution time of the algorithm could be sensibly less than the worst case.

Now we consider the space complexity. A point feature requires $O(d)$ space. Indeed, the field point is an array of $d$ floating point numbers, the field key is a bit-array of size $hd$, where $h$ is the resolution of the space filling curve, resolution that can be considered fixed, while the other fields have all constant size. Thus, the algorithm needs $O(nd)$ space to store the array $f$. The space needed to store the queue $CPQ$ is $O(k)$, while the space needed to any other auxiliary data structure is upper bounded by $n$. Thus, the overall space complexity of the algorithm is $O((n_B + n_R)d)$, i.e. linear in the input size.

4. Experimental results

Next we describe the results of some experiments performed using the Approx-SFC-CP-Join algorithm. We tested the algorithm both on synthetic and real data sets. The synthetic data sets we used are composed by points uniformly distributed in the unit hypercube.

The real data sets are reported in Table 1. The data sets ColorMoments, CoocTexture and ColorHistogram contain image features extracted from a Corel image collection. 1 The CovType data set contains forest cover type for 30 × 30 meter cells obtained from US Forest Service (USFS). 2 The Landsat data set is a collection of 40 large aerial photos divided into 64 × 64 tiles. 3 In particular, from each of these collections we obtained a pair of data sets by randomly partitioning the collection into two equally sized sets, and then we searched for the top-$k$ closest pairs of these two data sets.

We note that both the synthetic and the real data sets employed are 100% overlapping. In all the experiments presented we used the Z-order space filling curve (with resolution $h = 8$) and the Euclidean distance.

---

1 See www.kdd.ics.uci.edu/databases/CorelFetures/CorelFeatures.html.
3 See www.vision.ece.ucsb.edu/datasets/index.html.
We point out that Approx-SFC-CP-Join behaves as an exact algorithm on low dimensional data, up to 6–7 dimensions, as in these spaces it always prunes all the input points. Fig. 4 shows the execution times obtained computing the top-$k$ pairs, with $k \in \{10, 100, 1,000\}$, of the uniform four-dimensional data set with $n_B$ (or $n_R$, respectively) varying from 10,000 to 1,000,000. In all these experiments the algorithm pruned all the data sets points and, thus, returned the exact solution. The figure also shows that the algorithm scales linearly w.r.t. the input size.

Increasing the dimensionality of the uniform data sets, the algorithm is no more able, in the ranges of $n$ and $k$ above considered, to prune all the points. Thus, it makes sense to study the approximation quality as a function of the dimension. The approximation error is the ratio $\frac{\sum_{i=1}^{k} \delta_i}{\sum_{i=1}^{k} \delta_i^{\star}}$, where $\delta_i$ and $\delta_i^{\star}$ denote, respectively, the true and the approximate, i.e. those returned by the algorithm, $i$th closest pair distance in $J_k(B, R)$, for $i = 1, \ldots, k$. Table 2 shows the approximation error obtained running the algorithm on two uniform data sets, containing 100,000 points each, and varying their dimensionality from 2 to 50. It is worth to note that the approximation error obtained for 50-dimensional data sets was about 1% when $k$ was set to 100. This confirms

---

Table 1
Real data sets used in the experiments

<table>
<thead>
<tr>
<th>Data set</th>
<th>$d$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColorMoments</td>
<td>9</td>
<td>68,040</td>
</tr>
<tr>
<td>CoocTexture</td>
<td>16</td>
<td>68,040</td>
</tr>
<tr>
<td>ColorHistogram</td>
<td>32</td>
<td>68,040</td>
</tr>
<tr>
<td>CovType</td>
<td>55</td>
<td>581,012</td>
</tr>
<tr>
<td>Landsat</td>
<td>60</td>
<td>275,465</td>
</tr>
</tbody>
</table>

Fig. 4. Execution times on four-dimensional uniform data sets.

We point out that Approx-SFC-CP-Join behaves as an exact algorithm on low dimensional data, up to 6–7 dimensions, as in these spaces it always prunes all the input points. Fig. 4 shows the execution times obtained computing the top-$k$ pairs, with $k \in \{10, 100, 1,000\}$, of the uniform four-dimensional data set with $n_B$ (or $n_R$, respectively) varying from 10,000 to 1,000,000. In all these experiments the algorithm pruned all the data sets points and, thus, returned the exact solution. The figure also shows that the algorithm scales linearly w.r.t. the input size.

Increasing the dimensionality of the uniform data sets, the algorithm is no more able, in the ranges of $n$ and $k$ above considered, to prune all the points. Thus, it makes sense to study the approximation quality as a function of the dimension. The approximation error is the ratio $\frac{\sum_{i=1}^{k} \delta_i}{\sum_{i=1}^{k} \delta_i^{\star}}$, where $\delta_i$ and $\delta_i^{\star}$ denote, respectively, the true and the approximate, i.e. those returned by the algorithm, $i$th closest pair distance in $J_k(B, R)$, for $i = 1, \ldots, k$. Table 2 shows the approximation error obtained running the algorithm on two uniform data sets, containing 100,000 points each, and varying their dimensionality from 2 to 50. It is worth to note that the approximation error obtained for 50-dimensional data sets was about 1% when $k$ was set to 100. This confirms

---

4 We ran the experiments on a Pentium III 800MHz based machine.
that in practice the approximation error guaranteed by the algorithm is much better than the worst case stated in the previous section.

Next, we present the experiments on real data sets. We point out that we obtained the exact solution in all the experiments executed on the real data sets, also when they were not entirely pruned. Fig. 5 shows the execution time on the real data sets considered, when $k$ is varied from 1 to 100. The figure points out the very good performances of the algorithm for all the data sets when $k$ increases.

Furthermore, we point out that the algorithm found the exact solution after few iterations. Table 3 shows the number of iterations performed by the Approx-SFC-CP-Join algorithm for $k = 100$ and the approximation error after the first two iterations. Thus, in these experiments, the algorithm computed the exact solution in at most two iterations, although it required more iterations to meet the stop conditions.

The pruning ratio $\frac{\sum_{i=0}^{k} n_i}{\sum_{i=0}^{k} \max(n_i,\bar{n}_i)}$ is the fraction of the data sets pruned by the algorithm. Table 4 shows the pruning ratio on the real data sets considered, when $k$ is varied from 1 to 100. The table points out the remarkable pruning ability of the algorithm. In fact for all the sets, except the Landsat one when $k$ equals 10% and 100%, the 100% of points were pruned.

Fig. 6 shows the number of points of the CovType data set (both blue and red) that at each iteration must still be considered for $k \in \{1,10,100,1000\}$. We note that already after the first
iteration the data set size sensibly reduces and only after three steps, for \( k = 1000 \) (or at most six for \( k = 1 \)), all the points have been pruned.

Finally, in Fig. 7, we show the behavior of the Approx-SFC-CP-Join algorithm on a spatial data set relative to roads and streams of California. The (2,249,727) blue points represents the roads while the (98,451) red points are relative to the streams of California. We searched for the top 10,000 closest pairs from the two data sets. Fig. 7 on the top depicts the two data sets

\[ \text{Table 3} \]
Approximation error after the first two iterations for \( k = 100 \) on real data sets

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Iterations performed</th>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColorMoments</td>
<td>3</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>CoocTexture</td>
<td>1</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>ColorHistogram</td>
<td>2</td>
<td>1.4057</td>
<td>1.0000</td>
</tr>
<tr>
<td>CovType</td>
<td>4</td>
<td>1.0055</td>
<td>1.0000</td>
</tr>
<tr>
<td>Landsat</td>
<td>61</td>
<td>1.0062</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

\[ \text{Table 4} \]
Pruning ratios on real data sets

<table>
<thead>
<tr>
<th>Data Set</th>
<th>( k = 1 )</th>
<th>( k = 10 )</th>
<th>( k = 100 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColorMoments</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>CoocTexture</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>ColorHistogram</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>CovType</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Landsat</td>
<td>1.00</td>
<td>0.84</td>
<td>0.85</td>
</tr>
</tbody>
</table>

Fig. 6. Non-pruned points per iteration of the CovType data set.

\[ \text{Fig. 7} \]

\[ \text{See www.dke.cti.gr/People/ytheod/research/datasets/spatial.html.} \]
to join, and Fig. 7 in the middle shows the blue and red points remained after the first iteration. The points representing the streets diminished from 2,249,727 to only 1246, and those representing the streams diminished from 98,451 to 678. The figure points out that the non-pruned points accumulate around the boundaries of some $r$-regions. At the end of the second iteration, all the points were pruned. Fig. 7 on the bottom reports the result of the join.

5. Conclusions

We presented an approximate algorithm to calculate the top-$k$ closest pairs join of two large high dimensional data sets. Experimental results show that in practice the algorithm guarantees the exact solution for low dimensional data and a solution within a small error to the exact one for high dimensional data.
References


Fabrizio Angiulli received the Diploma Universitario in Control Systems and Computer Engineering in February 1996 and the Laurea Degree in Computer Engineering in October 1999 from the University of Calabria. From January 1994 to December 1994 he had a research grant at the Institute of Systems Analysis and Information Technology of the Italian National Research Council (ISI-CNR). In January 2001 he joined the Institute of High Performance Computing and Networking of the Italian National Research Council (ICAR-CNR), as senior researcher. Since 1995 she is a lecturer at University of Calabria and University of Reggio Calabria. His research interests include artificial intelligence, data mining, and databases.

Clara Pizzuti received the Laurea degree in Mathematics from the University of Calabria, Italy. She worked in the research division of a software company on deductive databases and abduction, participating in Esprit projects on advanced logic based systems. In 1994 she joined the Institute of High Performance Computing and Networking of the Italian National Research Council (ICAR-CNR), as senior researcher. Since 1995 she is a lecturer in the department of Computer Science at the University of Calabria. Her research interests include genetic algorithms, genetic programming, knowledge discovery in databases and data mining.