Journal of Mathematical Modelling and Algorithms (2005) 4: 149–179
DOI: 10.1007/s10852-004-4080-3

Approximate k-Closest-Pairs in Large High-Dimensional Data Sets

FABRIZIO ANGIULLI and CLARA PIZZUTI
ICAR-CNR, Via Pietro Bucci 41C, Università della Calabria, 87036 Rende (CS), Italy.
e-mail:{angiulli, pizzuti}@icar.cnr.it

(Received: 24 June 2004; in final form: 15 September 2004)

Abstract. An approximate algorithm to efficiently solve the *k-Closest-Pairs* problem on large high-dimensional data sets is presented. The algorithm runs, for a suitable choice of the input parameters, in $\mathcal{O}(d^2nk)$ time, where d is the dimensionality and n is the number of points of the input data set, and requires linear space in the input size. It performs at most $d + 1$ iterations. At each iteration a shifted version of the data set is sequentially scanned according to the order induced on it by the Hilbert space filling curve and points whose contribution to the solution has already been analyzed are detected and eliminated. The pruning is lossless, in fact the remaining points along with the approximate solution found can be used for the computation of the exact solution. If the data set is entirely pruned, then the algorithm returns the exact solution. We prove that the pruning ability of the algorithm is related to the nearest neighbor distance distribution of the data set and show that there exists a class of data sets for which the method, augmented with a final step that applies an exact method to the reduced data set, calculates the exact solution with the same time requirements.

Although we are able to guarantee a $\mathcal{O}(d^{1+1/t})$ approximation to the solution, where $t \in \{1, 2, \ldots, \infty\}$ identifies the Minkowski (L_t) metric of interest, experimental results give the exact k closest pairs for all the large high-dimensional synthetic and real data sets considered and show that the pruning of the search space is effective. We present a thorough scaling analysis of the algorithm for in-memory and disk-resident data sets showing that the algorithm scales well in both cases.

Mathematics Subject Classification (2000): 68W25.

Key words: k-Closest-Pairs problem, Space Filling Curves, approximate algorithms.

1. Introduction

In recent years, there has been an increasing interest in finding efficient solutions to proximity problems in high-dimensional spaces. The *k-Closest-Pairs problem* is an example of this class of problems. Given a set $\mathcal{D}$ of n points in $\mathbb{R}^d$ and an integer $k, 1 \leqslant k \leqslant n(n-1)/2$, it consists in finding the k closest pairs of point of $\mathcal{D}$ under a given metric. Proximity problems have traditionally been studied in computational geometry but, in the last few years they received considerable attention in other fields such as statistics [20], pattern recognition [2, 15], spatial databases [12, 21, 40] and data mining [31]. When the dimension, d, of the search space is low or it is considered as a constant, very efficient solutions [5, 14, 25, 28, 32, 33] to

the k-Closest-Pairs problem have been found. However a thorough analysis reveals that the time requirements of these algorithms depends exponentially on d [29]. As a consequence, when the dimensionality, d, is a component of the input, the *brute force* approach, i.e. computing the distances of all the pairs of points and reporting the smallest ones, can outperform such methods, even for very small values of d. Thus, when d grows, the solution of the problem can be very time expensive. The applications related to the above mentioned fields could require a number of features of several hundreds. The brute force approach, on the other hand, takes time quadratic in the size of the data set, which is prohibitive for data sets with more than hundred thousands of points. The lack of efficient algorithms when the dimension is high is known as the "curse of dimensionality" [7]. An approach adopted to overcome this problem is based on being satisfied with an approximate, but faster, solution, considering that many of the aforementioned applications do not necessarily require exact solutions [4, 19, 22, 23].

In this paper we present an approximate algorithm to solve the *k-Closest-Pairs* problem. The method, named ASP, Approximate k-closest-pairs with SPace-filling curves, is based on the dimensionality reduction of the space $\mathbb{R}^d$ through a space filling curve, the Hilbert curve, and exploits the order induced on the space $\mathbb{R}^d$ by the Hilbert curve. It performs at most $d + 1$ sorts and scans of the data set by using shifted copies of the input data set.

Our approach makes use of a property that allows us to eliminate early those points whose contribution to the solution has already been analyzed. At each iteration, after the mapping of the d-dimensional points into one dimensional ones, the points are examined with respect to the order induced by the space filling curve. By exploring the $2m$ (m predecessors and m successors) nearest neighbors in the one-dimensional space of each point p, we are able to determine, among such $2m$ points, the true $l \leqslant 2m$ nearest neighbors of p in the d-dimensional space and to establish a lower bound to the distance from p to its $(l + 1)$-th nearest neighbor. Such a property allows us to detect the points that do not need to be considered any more and thus that can be discarded. The pruning of the search space is lossless, that is the remaining points along with the approximate solution found can be used for the computation of the exact solution.

The main contributions of this work can be summarized as follows:

- We define a general property related to space-filling curves, that we employ to solve the k-Closest-Pairs problem, but that can be exploited to solve other proximity problems.
- We give an approximate algorithm to efficiently solve the k-Closest-Pairs problem on large, high-dimensional data sets, that performs no more than $d + 1$ sorts and scans of the input data set in at most $\mathcal{O}(d^2nk)$ time, for a suitable choice of the input parameters, and requires $\mathcal{O}(dn + k)$ space (we note that dn is the size of the input data set).

- The algorithm is able to eliminate points from the data set after each scan. The pruning of the search space is *lossless*, thus the introduced approximate algorithm could constitute a fast *preprocessing step* for an exact algorithm.
- Although the algorithm is approximate, and guarantees an $\mathcal{O}(d^{1+1/t})$ approximation to the solution, if the data set is entirely pruned, then we are able to state that the returned solution is the exact one. Furthermore, we prove that the pruning ability of the algorithm is related to the nearest neighbor distance distribution of the data set and show that there exists a class of data sets for which it is possible to obtain the exact solution in $\mathcal{O}(d^2nk)$ time by augmenting the ASP algorithm with a final step that applies an exact method (like the brute force) to the pruned data set.
- The experimental results (a) give quickly the *exact* closest pairs for all the considered synthetic and real high-dimensional data sets and for different values of k, (b) confirm that the pruning of the search space is effective, that is the size of the data set at the end of the algorithm is significantly reduced, and (c) show that the algorithm outperforms brute force enumeration by some order of magnitude and that it guarantees a better approximation quality than related approaches.
- We present a thorough scaling analysis of the algorithm for in-memory and disk-resident synthetic data sets showing that the algorithm scales well with the data set size, both in memory and disk-resident, as well as with the dimension.

This work enhances that presented in [3] by giving detailed proofs of the stated properties of the algorithm ASP, by extending ASP for not-in-memory data sets, and by presenting an accurate scaling analysis for in-memory and disk-resident synthetic data sets.

The paper is organized as follows. The next section gives an overview of space filling curves. Section 3 gives the definitions and the properties necessary to introduce the method. Section 4 presents the algorithm. Section 5 extends the method when the data set does not fit in main memory. Section 6 gives an overview of existing and related approaches for the solution of the k-Closest-Pairs problem. In Section 7 experimental results on several data sets and a comparison with the algorithm proposed in [29] are reported.

2. Space Filling Curves

The concept of *space-filling curve* arose in the 19-th century and is accredited to Peano [35] who, in 1890, proved the existence of a continuous mapping from the interval $I = [0, 1]$ onto the square $Q = [0, 1]^2$, thus settling a question posed almost ten years before about the existence of a curve that passes through every point of a closed unit square. Curves of this type are called *Peano curves* or *space-filling curves*. More formally,

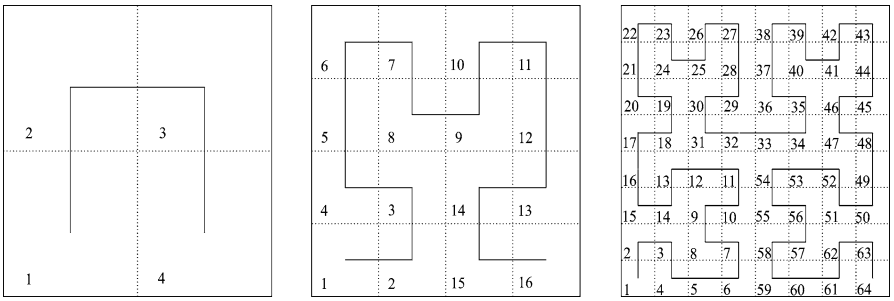

Figure 1. The Hilbert space-filling curve.

DEFINITION 2.1. Let E^d denote the d-dimensional Euclidean space. If a mapping $f : I \to E^d$, $d \geqslant 2$, is continuous and the image $f(I)$ of I under f has positive Jordan content (area for $d = 2$, volume for $d = 3$), then $f(I)$ is called a space-filling curve.

Peano discovered the first space-filling curve, but it was Hilbert in 1891 who defined a general procedure to generate an entire class of space-filling curves. Hilbert observed that if the interval I can be mapped continuously onto the square Q then, after partitioning I into four congruent subintervals and Q into four congruent subsquares, each subinterval can be mapped onto one of the sub-squares. Sub-squares are ordered such that each pair of consecutive sub-squares share a common edge. If this process is continued ad infinitum, I and Q are partitioned into 2^{2h} replicas for $h = 1, 2, 3, \ldots$. Figure 1 shows the first three steps of this process. Sub-squares are arranged so that the inclusion relationships and adjacency property are always preserved.

DEFINITION 2.2. Let D be $[0, 1]^d$ and $f_h : I \to D$, be a mapping such that every point $x \in I$ is uniquely determined by a sequence of nested closed intervals (generated by the above defined partitioning) the length of which shrinks to zero and to this sequence corresponds a unique sequence of nested closed hypercubes, the diagonal of which shrinks into a point, the image $f_h(x)$ of x, then $f_h(I)$ is called the *Hilbert curve* [35].

It has been proved that $f_h(I)$ is a space-filling curve. In practical applications the partitioning process is terminated after h steps to give an approximation of a space-filling curve of order h. For $h \geqslant 1$ and $d \geqslant 2$ let $\mathcal{H}_h^d$ denote the h-th order approximation of a d-dimensional Hilbert space-filling curve that maps 2^{hd} subintervals of length $1/2^{hd}$ into 2^{hd} sub-hypercubes whose centre-points are considered as points in a space of finite granularity. The Hilbert curve, thus, passes through every point in a d-dimensional space once and once only in a particular order. This establishes a mapping between values in the interval I and the coordinates of d-dimensional points. Let p be a d-dimensional point in D. The inverse image $f_h^{-1}(p)$ of p is called its *Hilbert value* and is denoted by $\mathcal{H}(p)$. Let $\mathcal{D}$ be a set

of n points in D. These points can be sorted according to the order in which the curve passes through them. We denote by $\mathcal{H}(\mathcal{D})$ the set $\{\mathcal{H}(p) \mid p \in \mathcal{D}\}$ sorted with respect to the order relation induced by the Hilbert curve. Given a point p the predecessor and the successor of p, denoted $\mathcal{H}_{pred}(p)$ and $\mathcal{H}_{succ}(p)$, in $\mathcal{H}(\mathcal{D})$ are thus the two closest points with respect to the ordering induced by the Hilbert curve. The m-th predecessor and successor of p are denoted by $\mathcal{H}_{pred}(p, m)$ and $\mathcal{H}_{succ}(p, m)$.

Space filling curves have been studied and used in several fields [1, 17, 18, 24, 30, 39]. A useful property of such a mapping is that if two points x_1 and x_2 from the unit interval I are close then the corresponding images $p = f_h(x_1)$ and $q = f_h(x_2)$ are close too in the hypercube, D, in particular, if

$$|x_1 - x_2| \leqslant 2^{-(h+1)d}$$

then

$$\max\{|p_i - q_i| : 1 \leqslant i \leqslant d\} \leqslant 2^{-h}.$$

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 rotation and/or translation (shift) of the hypercube D along the main diagonal with a fixed displacement have been proposed [29, 37, 39]. 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 particular, in [39] it is showed that if we consider the interval $[0, L]$, where L is a fixed constant and represents the degree of approximation desired, and a family of hypercubes D_l, $1 \leqslant l \leqslant L$, such that each hypercube D_l is obtainable by the translation of the hypercube D_{l-1} along the main diagonal with displacements 2^{-l} in each coordinate, then two d-dimensional points that coincide in all coordinates but one, and for this coordinate the difference does not exceed 2^{-p}, have at least two inverse images whose distance is less than 2^{-pd}, where p depends on L. A similar approach is described in [29], in this case, however, the number of shifts depends on the dimension d. Given a data set $\mathcal{D}$ and the d-dimensional vector

$$v^{(j)} = \left(\frac{j}{d+1}, \ldots, \frac{j}{d+1}\right) \in \mathbb{R}^d$$

each point $p \in \mathcal{D}$ can be translated $d + 1$ times along the main diagonal obtaining points $p^{(j)} = p + v^{(j)}$, for $j = 0, \ldots, d$. The shifted copies of points thus belong to $[0, 2]^d$ and, for each point, $d + 1$ Hilbert values in the interval $[0, 1]$ can be computed. Figure 2 shows the three shifted copies of a two-dimensional data set. Notice that the two close points p and q are far along the order induced by the Hilbert curve on the first copy of the data set. However, they are close along the

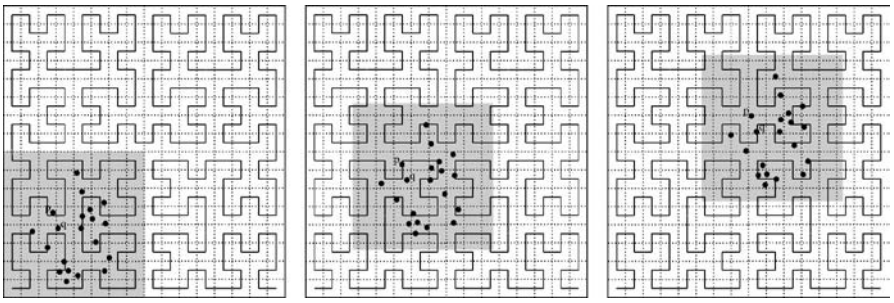

Figure 2. Shifts of a two-dimensional data set.

same order induced on the second copy. In this paper we make use of this family of shifts to overcome the loss of the nearness property.

In the next section we define a basic property exploited in the algorithm and give preliminary definitions necessary to introduce the method.

3. Preliminaries

In this section we give preliminary definitions and notations used in the paper, and we state the property, coming from space-filling curves, necessary to show the correctness of our algorithm. Although in the following we refer always to the Hilbert curve, the algorithm can be applied with other space-filling curves. We preferred the Hilbert curve because it has been shown to preserve locality better than other curves [30]. Without loss of generality, we assume that the given data set $\mathcal{D}$ has been normalized so that it is constituted by points in $D = [0, 1]^d$. The original and the shifted data points thus belong to $[0, 2)^d$, hence in the following we consider data sets on $[0, 2)^d$. Furthermore we fix the value h of the h-th order approximation of the Hilbert curve $\mathcal{H}_h^d$. Notice that the Hilbert value $\mathcal{H}(p)$ of a point p in $[0, 2)^d$ can be represented as a bit-string of length hd, i.e. as an integer in the interval $[0, 2^{hd})$ which, in turn, uniquely identifies a real in $[0, 1)$.

DEFINITION 3.1. Given a set $\mathcal{D}$ of n points in $\mathbb{R}^d$ and an integer k, $1 \leqslant k \leqslant n(n-1)/2$, the *k-Closest-Pairs* (CP_k) problem consists of finding the k closest pairs of $\mathcal{D}$ under the L_t metric.

For $t \geqslant 1$, the L_t distance $\mathrm{d}_t(p, q)$ between two points $p = (p_1, \ldots, p_d)$ and $q = (q_1, \ldots, q_d)$ is defined as

$$\mathrm{d}_t(p, q) = \left(\sum_{i=1}^{d} |p_i - q_i|^t \right)^{1/t}$$

for $1 \leqslant t < \infty$ and $\max_{1 \leqslant i \leqslant d} |p_i - q_i|$ for $t = \infty$.

In the following we denote by $CP_k(\mathcal{D})$ the output of the CP_k problem on an input data set $\mathcal{D}$.

DEFINITION 3.2. 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, $0 \leqslant i < d$, and s are in $\mathbb{N}$.

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

DEFINITION 3.3. Given a point p and other two points q_1 and q_2, we denote by $MaxReg(p, q_1, q_2)$ the side r of the greatest r-region containing p but neither q_1 nor q_2.

$MaxReg(p, q_1, q_2)$ can be calculated in time $\mathcal{O}(d)$ by working on the bit-string representation of $\mathcal{H}(p)$, $\mathcal{H}(q_1)$, and $\mathcal{H}(q_2)$.

DEFINITION 3.4. Let p be a point, and let r be the side of a r-region. We denote by $MinDist(p, r)$ the value

$$\min_{i=1,\ldots,d}\{\mathrm{mod}(p_i, r), r - \mathrm{mod}(p_i, r)\}$$

where $\mathrm{mod}(x, r) = x - \lfloor x/r \rfloor r$, and p_i denotes the value of p along the i-th coordinate.

Hence $MinDist(p, r)$ is the distance from the point p to the nearest face of the given r-region.

DEFINITION 3.5. Let p be a point in $\mathbb{R}^d$, and let r be a non-negative real. We define $\mathcal{B}(p, r)$ as the set of points whose distance from p is less than or equal to r, i.e. $\mathcal{B}(p, r) = \{q \in \mathbb{R}^d \mid \mathrm{d}_t(p, q) \leqslant r\}$.

DEFINITION 3.6. Let $\mathcal{D}$ be a data set, p a point in $\mathcal{D}$ and m be a positive integer. The *m-nearest-neighbor* of p in $\mathcal{D}$, written $\mathrm{NN}_m^{\mathcal{D}}(p)$, is a point q such that there exist $m-1$ points $q_1, \ldots, q_{m-1}$ with $\mathrm{d}_t(p, q_1) \leqslant \mathrm{d}_t(p, q_2) \leqslant \cdots \leqslant \mathrm{d}_t(p, q_{m-1}) \leqslant \mathrm{d}_t(p, q)$ and for each $x \in \mathcal{D}$, $x \neq p, q_i$, $i = 1, \ldots, m$, $\mathrm{d}_t(p, x) \geqslant \mathrm{d}_t(p, q_i)$.

DEFINITION 3.7. Given a data set, $\mathcal{D}$, a point p of $\mathcal{D}$, and a positive integer, m, the interval $\mathcal{I}_{\mathcal{H}}(p, m) \subseteq \mathcal{D}$ of the $2m$ points that precede and follow p in the Hilbert order is denoted by

$$\begin{aligned}\mathcal{I}_{\mathcal{H}}(p, m) = \{&q_m, q_{m-1}, \ldots, q_1, s_1, \ldots, s_{m-1}, s_m \mid \\ &q_m = \mathcal{H}_{pred}(p, m), \ldots, q_1 = \mathcal{H}_{pred}(p, 1), \\ &s_1 = \mathcal{H}_{succ}(p, 1), \ldots, s_m = \mathcal{H}_{succ}(p, m)\}.\end{aligned}$$

The following lemma allows us to determine, among such $2m$ points, the exact $l \leqslant 2m - 2$ nearest neighbors of p, i.e. $\mathrm{NN}_1^{\mathcal{D}}(p), \ldots, \mathrm{NN}_l^{\mathcal{D}}(p)$, and to establish a lower bound to the distance from p to the $(l+1)$-th nearest neighbor $\mathrm{NN}_{l+1}^{\mathcal{D}}(p)$.

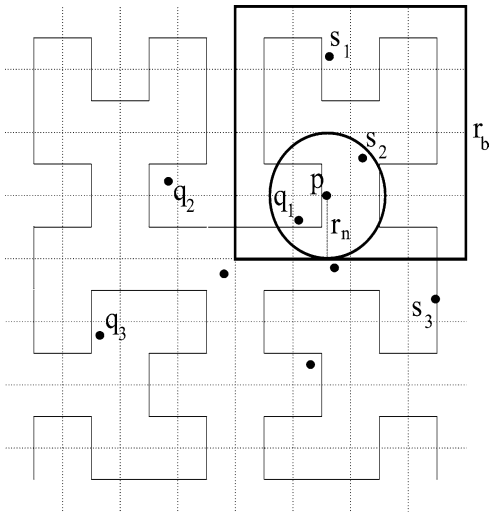

Figure 3. An example of application of Lemma 3.1 with $m = 3$.

LEMMA 3.1. *Given a data set $\mathcal{D}$, a point p of $\mathcal{D}$, a positive integer m and the interval $\mathcal{I}_{\mathcal{H}}(p, m)$, let $r_b = MaxReg(p, q_m, s_m)$, $r_n = MinDist(p, r_b)$, and $S_n = \mathcal{I}_{\mathcal{H}}(p, m) \cap \mathcal{B}(p, r_n)$. Then*

(1) *The points in S_n are the true first $|S_n|$ nearest-neighbors of p in $\mathcal{D}$;*
(2) $\mathrm{d}_t(p, \mathrm{NN}^{\mathcal{D}}_{|S_n|+1}(p)) > r_n$.

Proof. First, we note that, for each r-region, the intersection of the Hilbert space-filling curve, with the r-region results in a connected segment of the curve. Hence, to reach the points $\mathcal{H}_{pred}(p, m)$ and $\mathcal{H}_{succ}(p, m)$ from p following the Hilbert curve, we surely walk through the entire r-region of side r_b containing p.

As the distance from p to the nearest face of its r_b-region is r_n, then $\mathcal{B}(p, r_n)$ is entirely contained in that region. It follows that the points in S_n are all and the only points of $\mathcal{D}$ 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. □

Figure 3 shows an example of application of Lemma 3.1, and shows the r_b-region, the distance r_n, and $\mathcal{B}(p, r_n)$, for $m = 3$.

In the next section we describe the algorithm ASP.

4. Algorithm

In this section we give the description of the ASP algorithm. The method does at most $d + 1$ sorts and scans of the input data set. During each scan it works on

```
ASP (𝒟 = {p_1, ..., p_n}, k, m, h)
{
  Init(Q_CP, k);
  for (i = 1; i ⩽ n; i++) {
    F_i.point = p_i;
    F_i.lb = 0;
  }
  j := 0;
  while (F ≠ ∅ && j ⩽ d) {
    Hilbert(F, h, v^(j));
    Scan(F, v^(j), m|F|/n, Q_CP);
    Prune(F, Max(Q_CP));
    j := j + 1;
  }
  return 𝒫(Q_CP);
}
```

Figure 4. The algorithm ASP.

a shifted version of the data set. Exploiting the property defined in Lemma 3.1, at each iteration, it eliminates those points that do not need to be considered any more. This pruning of the search space allows us to obtain a fast and efficient approximate algorithm for the k-Closest-Pairs problem in high-dimensional spaces, as experimental results show in Section 7.

We recall that with the notation $v^{(j)}$ we denote a d-dimensional vector

$$v^{(j)} = \left(\frac{j}{d+1}, \ldots, \frac{j}{d+1} \right) \in \mathbb{R}^d.$$

Before starting with the description, we introduce the concept of *point feature* that contains, for each point of the input data set, its Hilbert value and a value that represents the radius of the maximum d-dimensional neighborhood explored.

DEFINITION 4.1. A *point feature* f is a triple $\langle point, hkey, lb \rangle$, where *point* is a point in $[0, 2)^d$, *hkey* is the Hilbert value associated to *point* in the h-th order approximation of the d-dimensional Hilbert space-filling curve mapping the hypercube $[0, 2)^d$ into the integer set $[0, 2^{hd})$, and *lb* is a distance representing the radius of the maximum d-dimensional neighborhood of *point* explored during the execution of the algorithm.

In the following the notation $f.point$, $f.hkey$, and $f.lb$ is used to the *point*, *hkey*, and *lb* value of the point feature f, respectively.

The algorithm is reported in Figure 4. Two data structures are employed, a priority queue, Q_{CP}, and a list of point features, F. Q_{CP} is a priority queue whose elements are triples of the form $\langle p, q, \delta \rangle$, where p and q are distinct points of the input data set, and δ is the associated distance $\mathrm{d}_t(p, q)$. Every triple contained in

```
Scan(F, v, M, Q_CP)
{
  for (i = 1; i ⩽ |F|; i++) {
    for (j = i + 1; j ⩽ i + M; j++) {
      δ = d_t(F_i.point, F_j.point);
      Update(Q_CP, F_i.point, F_j.point, δ);
    }
    p_l = F_{i−M}.point + v;
    p_r = F_{i+M}.point + v;
    r_b = MaxReg(F_i.point + v, pl, pr);
    r_n = MinDist(F_i.point + v, r_b);
    if (r_n > F_i.lb) F_i.lb = r_n;
  }
}
```

Figure 5. The procedure *Scan*.

Q_{CP} is one of the nearest pairs met during the execution of the algorithm. F is a list of point features, while F_i denotes the i-th element of the list F.

In the following we denote by $\mathcal{P}(Q_{CP})$ the set $\{\langle p, q\rangle \mid \langle p, q, \delta\rangle \in Q_{CP}\}$, and by $\mathcal{P}(F)$ the set $\{\langle p, q\rangle \mid \exists f, g \in F : f \neq g \wedge p = f.point \wedge q = g.point\}$.

The algorithm receives as input the data set $\mathcal{D} = \{p_1, \ldots, p_n\}$, the number k of closest pairs to find, the number m of neighbors that the algorithm is allowed to consider on the one-dimensional space (m predecessors and m successors), and the order approximation h of the Hilbert space-filling curve. The procedure *Init* initializes the priority queue Q_{CP} as an empty queue of size k, the number of closest pairs to find. The initialization phase builds also the list F associated to the input data set. The value of *lb* for each point feature of F is set to the value 0.

The main cycle, consists of at most $d+1$ steps. We explain the single operations performed during each step of this cycle.

The *Hilbert* procedure calculates the h-th order approximation of the value $\mathcal{H}(F_i.point + v^{(j)})$ of each point feature F_i of F, places this value in $F_i.hkey$, and sorts the point features in the list F using as order key the values $F_i.hkey$. Thus it performs the Hilbert mapping of a shifted version of the input dataset. It is straightforward to note that the shift operation does not alter the mutual distances between the points in F. As $v^{(0)}$ is the zero vector, at the first step ($j = 0$) no shift is performed. Thus during this step we work on the original data set.

The procedure *Scan* is reported in Figure 5. To update the priority queue, Q_{CP}, the procedure *Update* is employed. *Update*(Q, p, q, δ) modifies Q as follows: unless the triple $\langle p, q, \delta\rangle$ is already present in Q, if the size of Q is less than the maximum size allowed (we recall that, in the case of the queue Q_{CP}, this size is k, the number of closest-pairs we are searching for), then the triple $\langle p, q, \delta\rangle$ is inserted in the queue Q. Otherwise, if δ is less than the maximum distance associated to a triple in Q, then this triple is erased from Q and the triple $\langle p, q, \delta\rangle$

is inserted in Q. The procedure *Scan* performs a sequential scan of the list F. For each point feature F_i, the distances between the point $F_i.point$ and the points $F_{i+1}.point, \ldots, F_{i+M}.point$ are calculated. At the same time the priority queue, Q_{CP}, is updated through the procedure *Update*. After having examined the M point features consecutive to F_i, the size, r_n, of the greatest r-region containing $F_i.point$ but neither $F_{i-M}.point$ nor $F_{i+M}.point$ is calculated. Finally, if the value of r_n is greater than the value of $F_i.lb$ already determined, this value is set to r_n. Notice that the parameter M of *Scan* is set to $m|F|/n$, i.e. it is inversely proportional to the number of remaining point features in the list F. This allows the algorithm to further examine the neighborhood of the remaining points maintaining at the same time the number of distance computations performed in each iteration constant.

The procedure *Prune* deletes from F all the point features F_i such that $F_i.lb \geqslant Max(Q_{CP})$ where $Max(Q_{CP}) = \max\{\delta \mid \langle p, q, \delta\rangle \in Q_{CP}\}$. Hence, the list of features processed at the next step of the cycle is a subset (not necessarily proper) of the current list.

The main cycle stops when F is an empty list or after at most $d + 1$ steps. This terminates the description of the algorithm. the correctness of the algorithm. Now we state the main property of the algorithm.

THEOREM 4.1. *Let $\overline{F}$ and $\overline{Q}_{CP}$ be the current list of point features F and the current priority queue Q_{CP}, then, during the execution of the ASP algorithm, the following holds:*

$$CP_k(\mathcal{D}) \subseteq \mathcal{P}(\overline{Q}_{CP}) \cup \mathcal{P}(\overline{F}).$$

Proof. We point out that the value of each $F_i.lb$ (initially set to 0) is updated in the procedure *Scan* with the value r_n (see Figure 5), i.e. the distance from $F_i.point$ to the nearest face of the r-region containing itself but not its two furthest points $F_{i-M}.point$ and F_{i+M}.

Consider the first iteration ($j = 0$) of the algorithm, and the generic i-th main iteration of the procedure *Scan*. As the M points preceding $F_i.point$ are considered in the M preceding main iterations of *Scan* (i.e. the iterations $i - M, \ldots, i - 1$), then by Lemma 3.1 it follows that all the points in F placed at a distance not greater than r_n from $F_i.point$ were considered.

Once the scan of F is terminated, the procedure *Prune* deletes from F the point features f having $f.lb$ greater than $Max(Q_{CP})$. In Q_{CP} there are the top k closest pairs among those examined. Thus, if f is eliminated by *Prune*, then there does not exist a pair of points from $\mathcal{D}$ including $f.point$ that, at the same time, belongs to $CP_k(\mathcal{D})$ and was not examined.

Consider now a generic subsequent iteration ($j > 0$). Same considerations apply, but this time some points from the original data set could be missing. From the previous reasoning, all the pairs from $\mathcal{D}$ containing a point whose feature is not in F and potentially belonging to the solution set $CP_k(\mathcal{D})$, were examined in the previous iterations. Thus the property is guaranteed. □

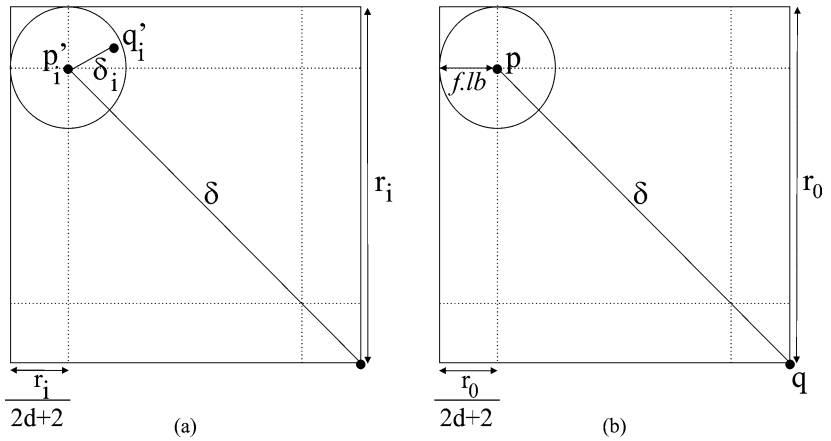

Figure 6. Approximation errors.

Theorem 4.1 proves the correctness of the algorithm. Furthermore, it establishes that the pruning operated by ASP 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 F^* and Q^*_{CP} we refer to the value of the list of point features F and of the priority queue Q_{CP} at the end of the ASP algorithm, respectively. Moreover, we denote by ϵ_d the value $2d^{\frac{1}{r}}(2d+1)$, and by δ_k the distance between the two points composing the k-th closest-pair of points in $\mathcal{D}$.

The following corollary gives the first important result regarding the algorithm, i.e. when the list F^* is empty than we can assert that the solution returned is the exact one.

COROLLARY 4.1. *$F^* = \emptyset$ implies that $CP_k(\mathcal{D}) = \mathcal{P}(Q^*_{CP})$.*

Proof. The result follows immediately from Theorem 4.1. □

Now we give some definitions and results useful to state the approximation error order for the value of δ_k guaranteed by the algorithm.

DEFINITION 4.2. A point p is *α-central* in an r-region iff for each $i = 1, \ldots, d$, we have $\alpha r \leqslant \mathrm{mod}(p_i, r) < (1-\alpha)r$, where $0 \leqslant \alpha < 0.5$.

The following result is from [8].

LEMMA 4.1. *Suppose d is even. Then, for any point $p \in \mathbb{R}^d$ and $r = 2^{-m}$ ($m \in \mathbb{N}$), 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 $\{p + v^{(0)}, \ldots, p + v^{(d)}\}$,

then, in at least one of these shifts, this point must become *sufficiently* central in an r-region, for each admissible value of r. This assures that for any point p and its closest q, in at least one of these shifts they will be in the same r-region.

As we use this family of shifts, we are able to state an upper bound for the approximation error of the algorithm similar to that defined in [29], that employed the same family of shifts.

THEOREM 4.2. $Max(Q^*_{CP}) \leqslant \epsilon_d \delta_k$.

Proof. Let $\{\langle p_1, q_1\rangle, \ldots, \langle p_k, q_k\rangle\}$ be the set $CP_k(\mathcal{D})$, and let $\delta_i = \mathrm{d}_t(p_i, q_i)$, for $i = 1, \ldots, k$. From Lemma 4.1 it follows that, for each $i = 1, \ldots, k$, there exists an r_i-region of side $\frac{r_i}{4d+4} \leqslant \delta_i < \frac{r_i}{2d+2}$ (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 $\mathcal{B}(p'_i, \delta_i)$ is entirely contained in that region, and both p'_i and q'_i belong to the r_i-region (see Figure 6(a)).

Notice that, as p'_i is $\frac{1}{2d+2}$-central in the r_i-region, this implies that the distance δ from p'_i and each point belonging to its r_i-region is at most $d^{\frac{1}{t}}(r_i - \frac{r_i}{2d+2})$, i.e. $\delta \leqslant d^{\frac{1}{t}} \frac{2d+1}{2d+2} r_i$.

Let $r_{\max} = \max\{r_1, \ldots, r_k\}$. If $Max(Q^*_{CP}) \leqslant d^{\frac{1}{t}} \frac{2d+1}{2d+2} r_{\max}$ then each triple $\langle p, q, \delta\rangle$ in Q^*_{CP} is such that $\delta \leqslant d^{\frac{1}{t}} \frac{2d+1}{2d+2}(4d+4)\delta_k \leqslant \epsilon_d \delta_k$.

Now we now show that $Max(Q^*_{CP}) \leqslant d^{\frac{1}{t}} \frac{2d+1}{2d+2} r_{\max}$. If every pair $\langle p_i, q_i\rangle$ is considered for insertion into Q_{CP} 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 Theorem 4.1 it follows that $\langle p_i, q_i\rangle \in \mathcal{P}(F^*)$, thus the point features associated to p_i and q_i certainly belong to 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 Hilbert curve in the j_i-th iteration. For the properties of the Hilbert curve, the M pairs examined in the j_i-th iteration belong to the same r_i-region containing both p'_i and q'_i, hence have distance less or equal than $d^{\frac{1}{t}} \frac{2d+1}{2d+2} r_i$.

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

Thus, the algorithm guarantees an $\mathcal{O}(d^{1+\frac{1}{t}})$ approximation to the solution. Now we define the worst case condition allowing a point to be pruned from the input data set by the ASP algorithm.

THEOREM 4.3. *The list F^* does not contain at least those points p of $\mathcal{D}$ such that $\delta > \epsilon_d^2 \delta_k$, where δ is the distance between p and its nearest neighbor in $\mathcal{D}$.*

Proof. Let p be a point of $\mathcal{D}$ and let δ be the distance between p and its nearest-neighbor q in $\mathcal{D}$. Let r_0 be the side of the greatest r-region satisfying the following inequality

$$d^{\frac{1}{t}} \frac{2d+1}{2d+2} r_0 < \delta.$$

Let $r_0 = 2^{-s_0}$ ($s_0 \in \mathbb{N}$), then s_0 is such that

$$\log_2 \left(\frac{d^{\frac{1}{t}}}{\delta} \frac{2d+1}{2d+2} \right) + 1 \geqslant s_0 > \log_2 \left(\frac{d^{\frac{1}{t}}}{\delta} \frac{2d+1}{2d+2} \right).$$

From Lemma 4.1, 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} (see Figure 6(b)). Thus, the in the worst case the value of $f.lb$ is $\frac{2^{-s_0}}{2d+2}$, where f is the feature such that $f.point = p$.

We know from Theorem 4.2 that the worst case approximation for the value of δ_k given by the ASP algorithm is $2d^{\frac{1}{t}}(2d+1)\delta_k$. Hence $f.lb > Max(Q_{CP})$ surely when

$$\frac{1}{2d+2} \left(2 \frac{d^{\frac{1}{t}}}{\delta} \frac{2d+1}{2d+2} \right)^{-1} > 2d^{\frac{1}{t}}(2d+1)\delta_k,$$

i.e. for $\delta > 4d^{\frac{2}{t}}(2d+1)^2\delta_k$. □

Thus, let Δ be the distribution of the nearest-neighbor distances of the points of $\mathcal{D}$, i.e. the distribution of the distances in the set $\{\mathrm{d}_t(p, \mathrm{NN}_1^{\mathcal{D}}(p)) \mid p \in \mathcal{D}\}$. Theorem 4.3 asserts that the ability of the ASP algorithm to prune points increases when the distribution Δ accumulates around and above the value $\epsilon_d^2 \delta_k$. Results in Section 7 give experimental evidence of the dependence of the pruning ability of the algorithm from the nearest neighbor distribution of the data set with respect to the value δ_k, but show also 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.

To conclude, we give time and space cost analysis of the algorithm. In the worst case the *Hilbert* procedure requires $\mathcal{O}(dn \log n)$ time, while the procedure *Prune* requires $\mathcal{O}(n)$ time. The procedure *Scan* requires in the worst case $\mathcal{O}(m(d + \log k))$ time to execute the inner cycle and $\mathcal{O}(d)$ time to calculate r_n, thus in total $\mathcal{O}(nm(d + \log k))$ time. Thus, in the worst case the algorithm runs in $\mathcal{O}(dn(d(\log n + m) + m \log k))$ time.

Without loss of generality, if we assume that m is $\mathcal{O}(k)$, $k \geqslant \log n$ and $d \geqslant \log k$, then the time complexity can be simplified in $\mathcal{O}(d^2 nk)$. As the algorithm enumerating all the possible pairs in the data set requires $\mathcal{O}(n^2(d + \log k))$ time,

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 one hundred dimensional data set composed by one million of points, using the ASP algorithm we expect time savings of at least two order of magnitude with respect to the brute force approach.

Notice 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.

From what is stated above, there exists a *class* of data sets for which it is still possible to obtain the exact solution in $\mathcal{O}(d^2nk)$ time by augmenting the ASP algorithm with a final step that applies an exact method (like the brute force) to the list F^*. This class is characterized by peculiar separation conditions, in particular a subset of its points have distance from their nearest neighbor less than $\epsilon_d^2\delta_k$ and the cardinality of this subset is of the order of $\mathcal{O}(\sqrt{dnk})$.

Let $N_k(\mathcal{D})$ be $|\{p \in \mathcal{D} \mid \mathrm{d}_t(p, \mathrm{NN}_1^{\mathcal{D}}(p)) \leqslant \epsilon_d^2\delta_k\}|$, and $\mathcal{E}_{\mathrm{ASP}}(k) = \{\mathcal{D} \mid N_k(\mathcal{D})$ is $\mathcal{O}(\sqrt{dnk})\}$.

THEOREM 4.4. *If the data set $\mathcal{D}$ belongs to $\mathcal{E}_{\mathrm{ASP}}(k)$ then the ASP algorithm, augmented with an exact post-processing step, calculates the exact solution of the k-Closest-Pairs Problem on $\mathcal{D}$ in time $\mathcal{O}(d^2nk)$.*

Proof. We already pointed out that because of Theorem 4.1, it is possible to calculate the exact solution $CP_k(\mathcal{D})$ starting from the queue Q_{CP}^* and the points in the list F^*. Thus, this can be done with no more than $\mathcal{O}(|F^*|^2(d + \log k))$ additional time, by using the brute force algorithm. If $\mathcal{D}$ belongs to $\mathcal{E}_{\mathrm{ASP}}(k)$, then from Theorem 4.3 and the definition of $\mathcal{E}_{\mathrm{ASP}}(k)$ it follows that $|F^*|$ is $\mathcal{O}(\sqrt{dnk})$. Hence, the total time necessary to calculate the exact solution $CP_k(\mathcal{D})$ is $\mathcal{O}(d^2nk + (\sqrt{dnk})^2d)$, i.e. $\mathcal{O}(d^2nk)$. □

Without loss of generality, we have defined $\mathcal{E}_{\mathrm{ASP}}(k)$ assuming that the parameters d, n and k satisfy the above introduced constraints. In practice, the class $\mathcal{E}_{\mathrm{ASP}}(k)$ is properly contained in a larger class $\mathcal{E}'_{\mathrm{ASP}}(k)$, composed by the data sets such that the ASP algorithm terminates with a list F^* of size $\mathcal{O}(\sqrt{dnk})$. In Section 7 we will see that this class contains data sets having less tightening separation conditions than those defining $\mathcal{E}_{\mathrm{ASP}}(k)$.

Finally, regarding the space cost analysis, the algorithm requires $\mathcal{O}(dn + k)$ space to store the list F and the queue Q_{CP}. As the size of the input data set is dn, then the algorithm requires space linear in the size of the input data set.

In the next section we describe the implementation of the disk-based version of the ASP algorithm.

5. Disk-Based Algorithm

In this section we describe the disk-based version of the algorithm ASP, designed to deal with data sets that cannot fit into main memory.

Let l and $\mathcal{F}$ be the number of bytes required to store, respectively, an integer and a floating point number.

DEFINITION 5.1. A *feature record* is a record composed of the following fields:

- *point*: an array of floating point numbers, representing a point in $[0, 2)^d$; this field is $\mathcal{F}d$ bytes long;
- *hkey*: is the Hilbert value associated to *point* in the h-th order approximation of the d-dimensional Hilbert space-filling curve mapping the hypercube $[0, 2)^d$ into the integer set $[0, 2^{hd})$; it is $\lceil \frac{hd}{8\mathit{l}} \rceil \mathit{l}$ bytes long;
- *lb*: is a floating point number representing the radius of a d-dimensional neighborhood of *point* entirely explored during the execution of the algorithm; it is $\mathcal{F}$ bytes long;
- *id*: is an integer whose value uniquely identifies the record; this field is l bytes long (assuming that $\log_2 n \leqslant 8\mathit{l}$).

Thus a feature record is $(\lceil \frac{hd}{8\mathit{l}} \rceil + 1)\mathit{l} + (d + 1)\mathcal{F}$ bytes long. A *feature file* is a sequential file composed by feature records.

The *Disk-based-ASP* algorithm is reported in Figure 7. The algorithm uses an in-memory data structure, the queue Q_{CP}, two feature files, FF_0 and FF_1, and a buffer to swap portions of the two files between main and secondary memory.

The input of the algorithm is composed by the file *InFile* containing the data set normalized in the hypercube $[0, 1)^d$, by the dimensionality d of the data set, the number k of closest pairs to find, the number m of neighbors that the algorithm is allowed to consider on the one-dimensional space, the approximation order h of the space-filling curve, and the size *BUF* of the buffer.

The procedure *FFCreate* reads the file *InFile* containing the data set and creates the corresponding feature file FF_0 in an obvious way.

The procedure *FFSort* performs an external sort of the file FF_0 producing the file FF_1 ordered with respect to the field *hkey*. In the current implementation of disk-based ASP, we used the *polyphase merge sort* with replacement selection to establish initial runs [27] to perform the external sort. This procedure requires the number *FIL* of auxiliary files allowed and the size *BUF* of the main memory buffer.

The procedure *FFScan*, reported in Figure 8, is the disk-based version of the procedure *Scan*. It allocates an array *buf* of feature records, used as a circular buffer to store a contiguous portion of the file *InFile*. The buffer must contain at least $2M + 1$ feature records, hence if the value of M becomes greater than *buflen* (recall that M is set to mN/n, thus it is initially equal to m and could increase during the execution of the algorithm) then it must be lowered to a suitable value. Basically, *FFScan* performs the same operations of *Scan*, but in addition it creates the feature file *OutFile* which will be processed during the next main iteration of the algorithm.

```
Disk-based-ASP(InFile, d, k, m, h, BUF)
{
  Init(Q_CP, k);
  n = FFCreate(InFile, FF_0);
  N = n;
  j = 0;
  while (N > 0 && j ⩽ d) {
    FFSort(FF_0, FF_1, FIL, BUF);
    Remove(FF_0);
    FFScan(FF_1, FF_0, BUF, mN/n, j);
    Remove(FF_1);
    N = FFPrune(FF_0, FF_1, Max(Q_CP));
    Remove(FF_0);
    Rename(FF_1, FF_0);
    j = j + 1;
  }
  return P(Q_CP);
}
```

Figure 7. The algorithm *Disk-based-ASP*.

Notice that the dimension of the buffer can be "doubled" by storing only the M feature records following the current record. Indeed, the M feature records preceding the current record are leaved into the buffer only to calculate the function *MaxReg*. But, $MaxReg(p, q_1, q_2)$, where p is the current point, q_1 is M-th predecessor of p, and q_2 is the M-th successor of p, is equal to $\min\{r_1, r_2\}$, where $r_1 = MaxReg(p, q_1, q_1)$, and $r_2 = MaxReg(p, q_2, q_2)$. Thus the real value r_1 could be stored in an augmented in-memory feature record (i.e. each in-memory feature record contains an additional field storing the floating point number r_1) and then used to calculate $\min\{r_1, MaxReg(p, q_2, q_2)\}$. As q_1 is not stored into the buffer when p is processed, it remains to establish how r_1 can be calculated: as p is the M-th predecessor of q_2, then the value r_1 associated to q_2 can be calculated as $MaxReg(q_2, p, p)$. This can be done contextually to the calculation of $MaxReg(p, q_2, q_2)$. So, when q_2 becomes the current point (M iterations later) the r_1 value associated to q_2 is correctly stored in its in-memory feature record. We do not report this version of *FFScan* because it is slightly more involved than those presented.

Finally, the procedure *FFPrune* deletes from the file FF_0 the feature records having value of *lb* greater than $Max(Q_{CP})$. Notice that *FFScan* operates an early pruning, as it stores in the file FF_0 only the feature records having *lb* less or equal than the current value of $Max(Q_{CP})$.

```
FFScan(InFile, OutFile, BUF, M, j)
#define succ(x) ((curr + x) % buflen)
#define pred(x) ((curr + M − x) % buflen)
{
  /* --- Allocates the buffer --- */
  buflen = BUF / sizeof(FeatureRecord);
  if (buflen < 2M + 1) M = floor((buflen − 1)/2);
  buf = calloc(buflen, sizeof(FeatureRecord));
  /* --- Fills the buffer --- */
  InFeatFile = fopen(InFile);
  maxrow = 0; // last record of InFile in the buffer
  for (i = 0; i < buflen && fread(buf[i], InFeatFile); i++) maxrow++;
  /* --- Starts the scan of the file InFile --- */
  OutFeatFile = fcreate(OutFile);
  curr = 0; // current position in the buffer
  currow = 0; // current record of InFile
  while (++currow ⩽ maxrow) {
    /* --- Compares the current point with the M successors --- */
    for (i = 1; i < min(M, maxrow − currow); i++) {
      other = succ(i);
      d = dist(buf[curr].point, buf[other].point, t);
      Update(Q_CP, buf[curr].id, buf[other].id, d);
    }
    /* --- Updates the field lb of the current record --- */
    pl = buf[pred(M)].point + v^(j);
    pr = buf[succ(M)].point + v^(j);
    rb = MaxReg(buf[curr].point + v^(j), pl, pr);
    rn = MinDist(buf[curr].point + v^(j), rb);
    if (rn > buf[curr].lb) buf[curr].lb = rn;
    /* --- Creates the feature file for the (j + 1)-th iteration --- */
    if (j < d && buf[curr].lb ⩽ Max(Q_CP)) {
      rec.hkey = Hilbert(buf[curr].point + v^(j+1), h);
      rec.point = buf[curr].point;
      rec.lb = buf[curr].lb;
      rec.id = buf[curr].id;
      fwrite(rec, OutFeatFile);
    }
    /* --- Loads into the buffer the next record of InFile --- */
    if (currow > m && fread(buf[pred(M)], InFeatFile)) maxrow++;
    curr = succ(1); // go to the next position in the buffer
  }
  fclose(OutFeatFile);
  fclose(InFeatFile);
  free(buf);
}
```

Figure 8. The procedure *FFScan*.

6. Related Work

Several exact algorithms regarding the k-Closest-Pairs Problem and its variations have been proposed. A comprehensive overview can be found in [32] and [38].

Bentley and Shamos [5] were the first who gave an optimal $O(n \log n)$ algorithm based on the divide and conquer paradigm. The search space was recursively partitioned and the problem was solved on these subspaces. To obtain the solution on the overall set, only those couples of points coming from two disjoint partitions whose distance is less than the minimum distance already found were considered.

This algorithm, as pointed out in [38], is complicated because it uses multidimensional divide and conquer. A more simple algorithm was proposed by Lenhof and Smid [28]. The algorithm requires $O(n \log n + k)$ time for any fixed dimension and is based on the construction of a grid such that (i) at least one cell contains at least two points and (ii) each cell contains at most 2^d points. If δ is the side length of this grid, property (i) implies that the distance between the two closest points in $\mathcal{D}$ is at most $d\delta$, thus each point of $\mathcal{D}$ must be compared with the points contained in its cell or in the $(2d+1)^d - 1$ neighboring cells. Property (ii) implies that each point is compared with a constant number of points. The algorithm then consists of two phases, an approximation phase in which the condition $20^d(k + 2n) \leqslant \frac{n(n-1)}{2}$ must be satisfied, and an enumeration phase that requires at least $\Omega(5^d\ k + 4^d n \log n)$. This implies that when $k = 1$ in the approximation phase d can not be greater that $\log_{20} n$. As k increases this limit value diminishes while the enumeration phase degenerates to the brute force $O(n^2)$ even for small values of d. A simplification of Lenhof and Smid's algorithm with the same running time has been introduced by Chan in [9].

Randomized algorithms to solve the closest pair problem in $O(n \log n)$ were proposed by Rabin [33], Dietzfelbinger et al. [14], and Khuller and Mathias [26]. The approach is similar to that of Lenhof and Smid [28], but the size δ of the cells is computed by using random sampling and the brute force approach on the sample.

Katoh and Iwano [25] 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 $d = 2$.

The on-line or dynamic extension to the closest pair problem has been studied in [6, 16, 36]. This version of the problem receives the points one after another and, as they arrive, the current closest pairs must be updated.

More recently algorithms for the problem of finding k closest pairs between two spatial data sets were introduced in [12, 11, 13]. 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. Each spatial data set is indexed using an R^*-tree, a variation of the R-trees hierarchical data structure that applies a sophisticated node split technique. 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 2D data, and studied the scalability of the methods when the data set size and the number of pairs required increases.

The algorithms were compared with those of Hjaltason and Samet [21] and showed to outperform them. Yang and Lin in [40] observed that the methods of Corral et al. work well when the data sets do not overlap and that the poor performances in case of overlapping are due to the inability of the join algorithm to prune nodes. Thus they proposed a new index structure, named the bichromatic Rdnn-Tree, that allows the pruning of the path search and algorithms with better performances than those of [12]. None of these algorithms were applied to data set containing more than four dimensions.

In [29], Lopez and Liao presented an approach to solve the k-Closest-Pairs Problem based on multiple shifted copies of the input data set ordered with respect to the Z-order (Peano space filling curve). Their method consists of two phases, an approximate phase, where the approximate solution is guaranteed within a small factor of the exact one, and an exact phase, in which the k closest pairs are obtained. The approximation phase uses a priority queue Q of size k to store a pair of unshifted points and the size of the smallest r-region that contains the (shifted) pair. For each point p_i, the k closest points q_j with respect to the Z-order are considered and the smallest r-region containing p_i and q_j is computed. If the size of this region is less than the size of the largest r-region contained in Q, the pair contained in this r-region is deleted while (p_i,q_j) is inserted in Q. This step is repeated $d+1$ times to consider for each point its $d+1$ shifts $v^{(0)}, v^{(1)}, \ldots, v^{(d)}$. The idea of using shifts to solve proximity problems, such as the exact Closest Pair and the Approximate Nearest Neighbor, has also been employed by Chan [10].

Though our approach is similar to the approximation phase of Lopez and Liao algorithm, the following main differences can be pointed out.

- Because of Lemma 3.1, our method at each step eliminates points of no more use, thus the number of points to be examined is sensibly reduced.
- Our method can stop without executing all the $d+1$ iterations and return the exact solution. Furthermore, it can be used as a preprocessing step for an exact algorithm, which takes advantage of the reduction of the data set size.
- We prove that there exists a class of data sets for which our method obtains the exact solution with the same time complexity by augmenting the algorithm with a final step that applies an exact method to the reduced data set. The characteristics of these data sets are intrinsic, that is they do not depend on the structure of the ASP algorithm.
- For each pair of points we consider their true distance instead of the size of the smallest r-region containing them.

- We consider the Hilbert space filling curve while they use the Peano curve. The Hilbert curve has been shown to achieve better results than the Peano curve [18].
- Experimental results shows that our method outperforms the Lopez and Liao method since, on the same data sets, we always obtain the exact solution.
- We realized a disk-based implementation of our method which is able to deal with data sets that do not fit into main memory.

In the next section we describe experimental results on large high-dimensional synthetic and real data sets.

7. Experimental Results

We implemented the algorithm using the C programming language[⋆] on a Pentium III 850 MHz based machine[⋆⋆,‡]. We used a 32 bit floating-point type to represent point coordinates and distances.

The data sets we used to test the algorithm are: *Cure* ($d = 2, n = 100{,}000$), *ColorMoments* (CM for short) ($d = 9, n = 68{,}040$), *CoocTexture* (COT for short) ($d = 16, n = 68{,}040$), *ColorHistogram* (CH for short) ($d = 32, n = 68{,}040$), and Landsat ($d = 60, n = 275{,}465$). Cure is the two-dimensional synthetic data set described in [34] (referred as Data set 1) and widely used to test clustering algorithms. The others data sets represent a collection of real images. The points of COT and CH are image features extracted from a Corel image collection[‡‡] while the points of Landsat are normalized feature vectors associated to tiles of a collection of large aerial photos.[¶]

We search the k closest pairs under the L_2 metric. We experimented also with the L_1 and L_∞ metrics and obtained analogous results, thus we just describe experiments under the Euclidean metric. Although we are able to guarantee an $\mathcal{O}(d^{\frac{3}{2}})$ approximation to the solution, the algorithm calculated the *exact solution* for all the experiments, even when F^* was not empty.

Table I reports the size, n, the dimension, d of each data set, the residual number, $|F^*|$, of points in the list of point features, F, at the end of the algorithm, the actual number, d^*, of steps necessary to the algorithm to find the solution, the *execution time*, [¶¶] T^*, of the ASP algorithm and the *execution time*, T, of the brute force approach when we are searching for the first $k = 100$ closest pairs ($m = k, h = 2$).

[⋆] We used Microsoft Visual C++ 6.0 as developing environment.

[⋆⋆] The operating system of the computer is Microsoft Windows XP Professional.

[‡] The machine has 256 MB of main memory and one EIDE hard disk.

[‡‡] See `http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeature.html` for more information.

[¶] See `http://vision.ece.ucsb.edu` for a detailed description.

[¶¶] Execution times were computed using the built-in function `clock()`, to obtain the overall number of clock cycles needed to perform the calculation, and the system constant `CLOCKS_PER_SEC`, which specifies the number of clock cycles per second.

Table I. Size of the F^* list and execution times of ASP and brute force method for $k = 100$

	n	d	$\lvert F^* \rvert$	d^*	T^* [sec]	T [sec]
Cure	100,000	2	0	2	40.28 ($h = 8$)	4,464.30
CM	68,040	9	0	3	24.63	2,890,83
COT	68,040	16	0	1	23.67	3,476.37
CH	68,040	32	0	3	38.91	5,369.59
Landsat	275,465	60	81,371	61	7,787.58	154,153.23

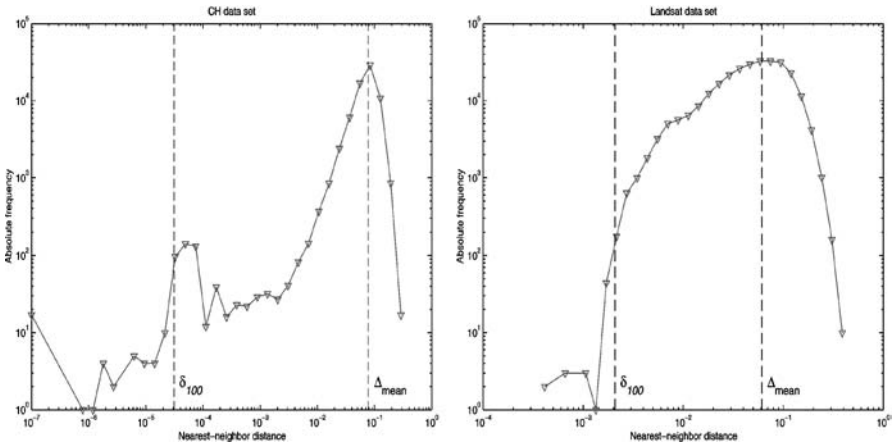

Figure 9. Nearest-neighbor distance distribution of CH and Landsat.

The table shows that ASP stops after at most 3 steps for the first four experiments obtaining the exact solution in few seconds. The results show the low execution times needed by ASP to obtain the solution with respect to the brute force approach (two order of magnitude less) and the effectiveness of the algorithm in reducing the size of the input data set. In practice the algorithm is able to prune points having distance δ to their nearest-neighbor significantly much less than $\epsilon_d^2 \delta_k$, the worst case stated by Theorem 4.3. The Landsat data set required all the 61 iterations (done in about two hours versus the about 42 hours of the brute force algorithm). In this latter case the algorithm was not able to produce an empty F^* list, but also in this case the list Q_{CP}^* contained the exact solution and the size of the input data set reduced significantly, in fact the size of F^* is about $1/3$ of the size of the data set.

Figure 9 depicts the distributions of the nearest-neighbor distances of the CH and Landsat data sets together with the value δ_{100} and the mean $\overline{\Delta}$ of each distribution. According to Theorem 4.3, for $k = 100$ the Landsat data set is more hard to manage than the CH data set by the ASP algorithm, as in the former case the ratio

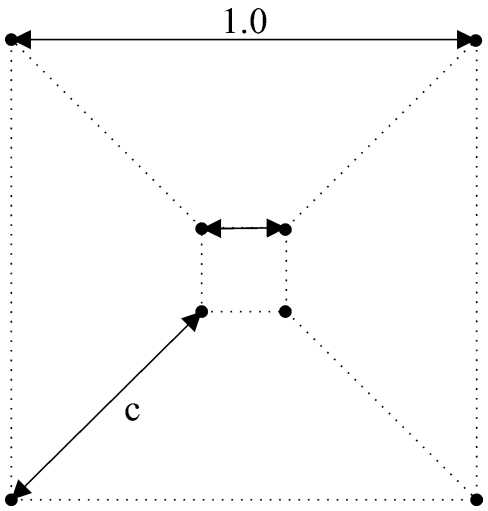

Figure 10. Synthetic data set.

$\overline{\Delta}/\delta_{100}$ is two order of magnitude less than the latter case (29.3 of Landsat versus 2474.3 of CH).

We now study how the algorithm scales with respect to the various parameters. To this end we built a family of synthetic data sets in which a subset of points is nearer each other with respect to the other points of the data set. Each data set of this family is characterized by the dimensionality, d, the total number of points, n, the number, s, of points that are closer to each other with respect to the remaining, $n - s$, and a parameter, c. Figure 10 shows the structure of these data sets. The s close points are placed in s distinct vertexes among the 2^d vertexes of the inner hypercube, having side length, δ, while the remaining $n - s$ nonclose points are placed in $n - s$ distinct vertexes among the 2^d vertexes of the outer hypercube, having side length 1.0. The L_2 minimum distance between a nonclose point and a close point is set to $c\delta$, this means that δ is equal to $\sqrt{d}/(\sqrt{d} + 2c)$. Thus the distribution Δ_c of the nearest-neighbor distances is such that there are s distances in the range $[\delta, \sqrt{d}\delta]$ and $n - s$ distances in the range $[c\delta, (c + \sqrt{d})\delta]$. We set $s = 1,000$ and $m = k$ in all the experiments, and considered the range $c \in \{10^3, 10^2, 10^1\}$. In all the experiments we found that the algorithm terminates with the exact solution and an empty F list after executing less than $d + 1$ iterations. We first describe the behavior of the memory-based version of ASP. Figure 11 shows the execution times of ASP for $n = 100,000$ points, $k = 100$, $h = 2$ and d ranging from 20 to 128. Figure 12, instead, shows the number of iterations performed in the aforementioned experiments by ASP before stopping the execution. Surprisingly, up to 128 dimensions, the algorithm executes a constant number of iterations for $c = 10^3$, less than 10 iterations for $c = 10^2$ and no more than $d/2$ iterations for $c = 10$. These curves show that in practice the algorithm is able to prune points having distance to their nearest-neighbor significantly less than $\epsilon_d^2 \delta_k$, the worst case

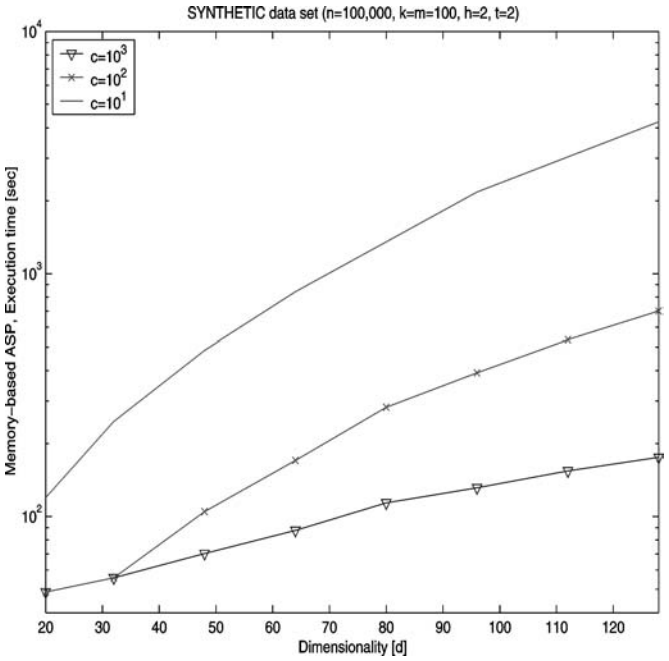

Figure 11. Execution time of ASP for different dimensions.

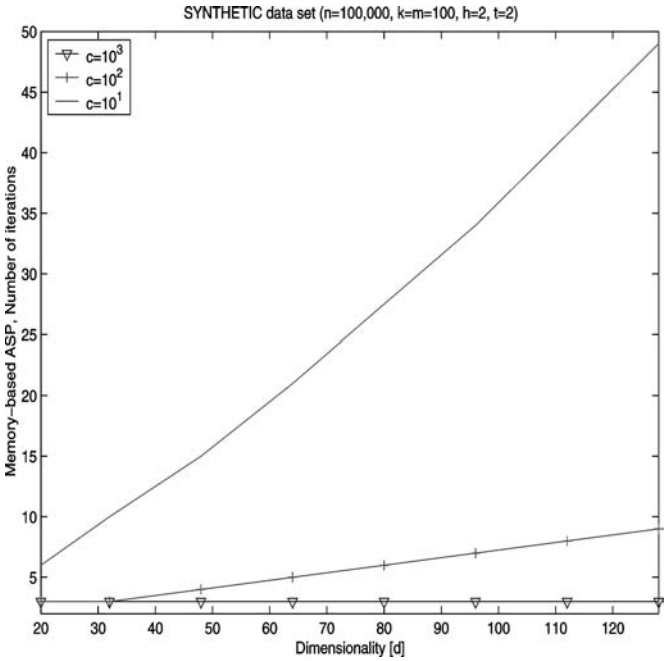

Figure 12. Number of iterations executed by ASP.

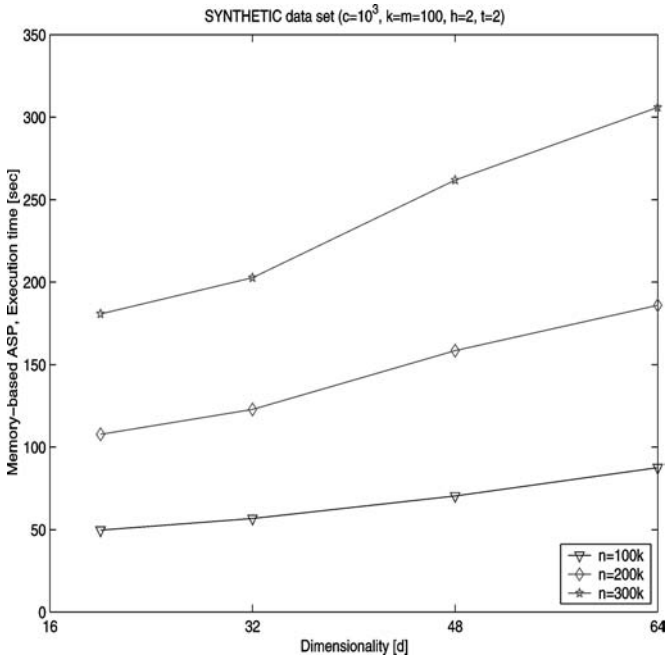

Figure 13. Execution times of memory-based ASP for different dimensions.

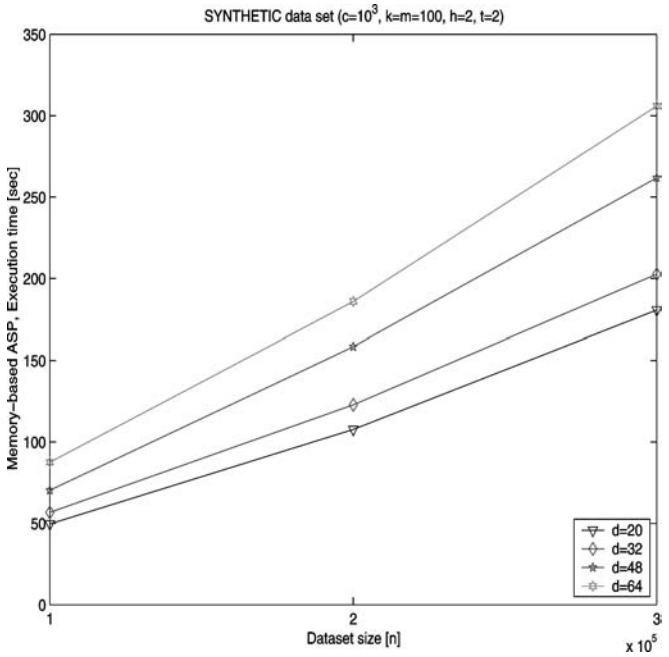

Figure 14. Execution times of memory-based ASP for different data set sizes.

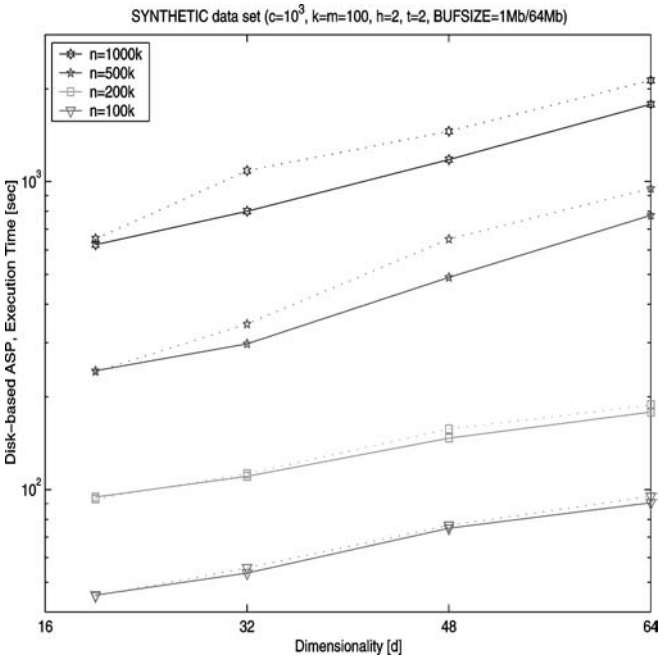

Figure 15. Execution times of disk-based ASP for different dimensions.

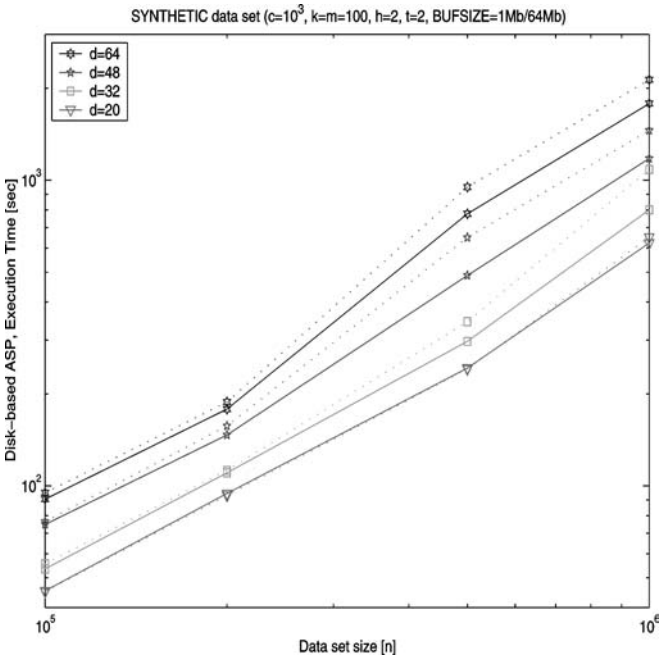

Figure 16. Execution times of disk-based ASP for different data set sizes.

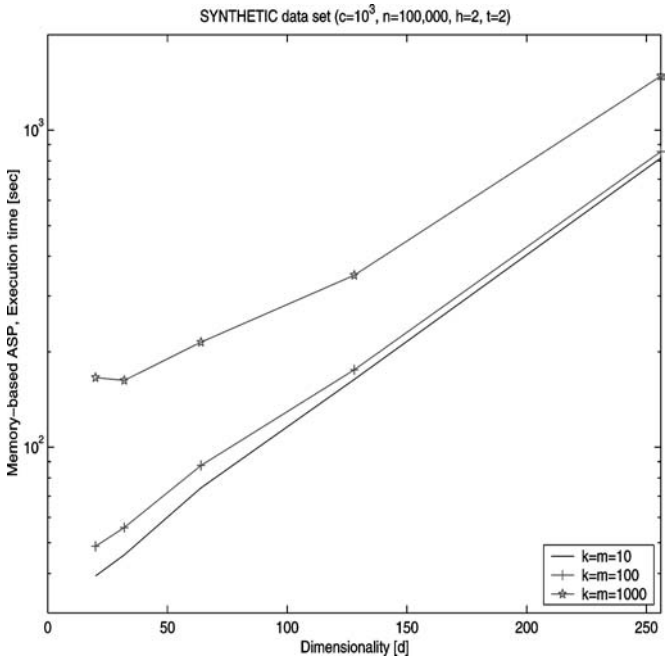

Figure 17. Execution times of memory-based ASP for different values of k.

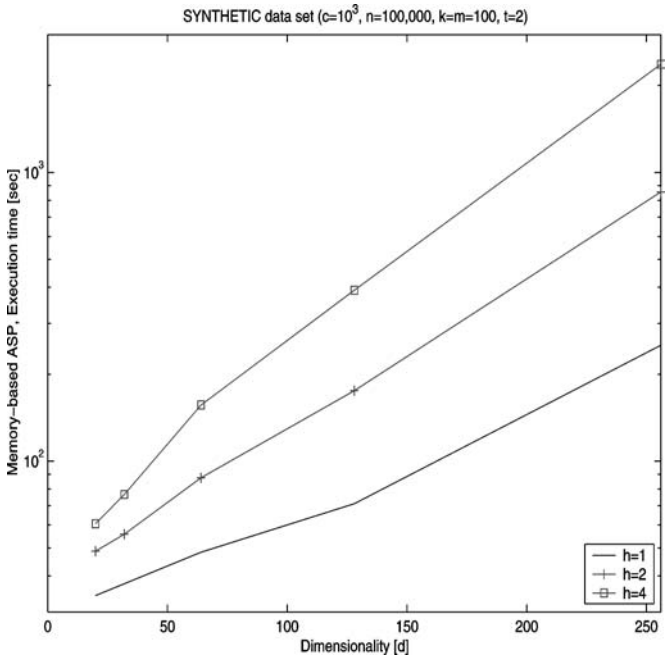

Figure 18. Execution times of memory-based ASP for different values of h.

Table II. Comparison between the approximation quality of LL and ASP

	$k = 1$		$k = 10$		$k = 20$		$k = 50$		$k = 100$	
	LL	ASP	LL	ASP	LL	ASP	LL	ASP	LL	ASP
COT	1.24	1.00	1.67	1.00	1.82	1.00	1.93	1.00	1.96	1.00
CH	1.00	1.00	1.00	1.00	1.32	1.00	1.15	1.00	1.68	1.00
Landsat	1.32	1.00	2.33	1.00	2.23	1.00	2.10	1.00	1.98	1.00

stated by Theorem 4.3. Indeed, we note that δ_k is such that $\delta \leqslant \delta_k \leqslant \sqrt{d}\delta$ and in the distribution Δ_c there are $n - s$ points having distance from their nearest-neighbor in the range $[c\delta, (c+\sqrt{d})\delta]$, thus significantly less than $\epsilon_d^2\delta_k$ (e.g., compare $\epsilon_{128}^2 = 33, 817, 088$ with $c = 10$). Figures 13 and 14 show how the memory-based version of ASP scales respectively with respect to the dimensionality d of the data set and the number n of points, while Figures 15 and 16 show the same information for the disk-based version of ASP. To study the effect of the buffer size on the execution time we varied the size from 1 MB (dashed curves) to 64 MB (solid curves). Time improvement associated to a larger buffer is mainly consequence of the better performance of the external sort algorithm. Figures 17 and 18 show how the algorithm ASP scales respectively with respect to the number k of closest pairs to find and the approximation order h of the Hilbert space-filling curve. All the figures points out the good performances of ASP as the values of the input parameters increases.

Finally we compared the quality of the approximation returned by our algorithm with those returned by algorithm [29] (LL for short in the following). We used the same data sets employed to test the LL algorithm: COT ($d = 16$, $n = 68,040$), CH ($d = 32$, $n = 68,040$), and Landsat ($d = 60$, $n = 275,465$).

The comparison between LL and ASP, under the L_2 metric, is reported in Table II. Each value reported in the table is a ratio of the form δ_k^*/δ_k, i.e. the ratio between the approximation δ_k^* to the k-th closest pair distance determined by the algorithm LL or ASP and the effective value δ_k of this distance. We run ASP with $h = 4$ and $m = k$ in all the experiments. The table shows that ASP guarantees a much more accurate approximation than the LL algorithm, since it always returned the exact closest pairs in all the experiments performed. LL found the exact solution only for the CH data set when $k = 1$ and $k = 10$.

8. Conclusions

In this paper an $\mathcal{O}(d^{1+1/t})$ approximate algorithm to solve efficiently the k-Closest-Pairs problem under the L_t metrics on large high-dimensional data sets in time $\mathcal{O}(d^2nk)$ and linear space has been presented. The algorithm exploits the order induced on the data by the Hilbert space-filling curve to eliminate points early from the input data set. The experimental results gave the exact closest pairs for all the

considered synthetic and real high-dimensional data sets and for different values of k, confirmed that the pruning of the search space is effective, and showed that the algorithm guarantees a better approximation quality than related approaches. We presented a thorough scaling analysis of the algorithm for in-memory and disk-resident synthetic data sets showing that the algorithm scales well with the data set size, both in memory and disk-resident, as well as with the dimension. The method is particularly suitable in all those applications in which the size n of the data set overcomes the product dk and the dimensionality d is large. In these cases it can be used as a preprocessing step for the brute force algorithm, actually the best exact algorithm for the k-Closest-Pairs problem in high dimensional spaces, by taking advantage of the reduction of the data set size operated by our algorithm.

Acknowledgements

Special thanks to Jonathan Lawder and Doung W. Moore who provided us their source codes implementing the multidimensional Hilbert mapping, and to Mario A. Lopez and Jelena Tesic who provided us some of the data sets we used to test the method.

References

1. Aluru, S. and Sevilgen, F. E.: Parallel domain decomposition and load balancing using space-filling curves, in *Proceedings of the International Conference on High Performace Computing*, 1997, pp. 230–235.
2. Andrews, H. C.: *Introduction to Mathematical Techniques in Pattern Recognition*, Wiley-Interscience, New York, 1972.
3. Angiulli, F. and Pizzuti, C.: Approximate k-closest-pairs with space filling curves, in Y. Kambayashi, W. Winiwarter and M. Arikawa (eds), *Proceedings of the Fourth International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2002)*, Aix-en-Provence, France, September 2002, Lecture Notes in Comput. Sci. 2454, Springer-Verlag, pp. 124–134.
4. Arya, S., Mount, D. M., Nethanyahu, N. S., Silverman, R. and Wu, A. Y.: An optimal algorithm for approximate nearest neighbour searching in fixed dimensions, *J. ACM* **45**(6) (1998), 891–923.
5. Bentley, J. L. and Shamos, M. I.: Divide-and-conquer in multidimensional space, in *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*, 1996, pp. 220–230.
6. Bespamyatnikh, S.: An optimal algorithm for closest pair maintenance (extended abstract), in *Proceedings of the 11th Annual ACM Symposium on Computational Geometry*, 1995, pp. 152–161.
7. Beyer, K., Goldstein, J., Ramakrishnan, R. and Shaft, U.: When is "nearest neighbor" meaningful? in *Proceedings of the Internatinal Conference on Database Theory*, 1999, pp. 217–235.
8. Chan, T.: Approximate nearest neighbor queries revisited, in *Proceedings of the 13th Annual ACM Symposium on Computational Geometry*, 1997, pp. 352–358.
9. Chan, T.: On enumerating and selecting distances, in *Proceedings of the 14th Annual ACM Symposium on Computational Geometry*, 1998, pp. 279–286.
10. Chan, T.: Closest-point problems simplified on the ram, in *Proceedings of the ACM Symposium on Discrete Algorithms (SODA'02)*, 2002.

11. Corral, A., Canadas, J. and Vassilakopoulos, M.: Approximate algorithms for distance-based queries in high-dimensional data spaces using R-trees, in *Proceedings East-European Conference on Advances in Databases and Information Systems (ADBIS'02)*, 2002, pp. 163–176.
12. Corral, A., Manolopoulos, Y., Theodoridis, Y. and Vassilakopoulos, M.: Closest pair queries in spatial databases, in *Proceedings of the ACM International Conference on Managment of Data (SIGMOD'00)*, 2000, pp. 189–200.
13. Corral, A., Manolopoulos, Y., Theodoridis, Y. and Vassilakopoulos, M.: Algorithms for processing k-closest-pair queries in spatial databases, *Data & Knowledge Engineering* **49** (2004), 67–104.
14. Dietzfelbinger, M., Hagerup, T., Katajainen, J. and Penttonen, M.: A reliable randomized algorithm for closest-pair problem, *J. Algorithms* **25**(1) (1997), 19–51.
15. Duda, R. O. and Hart, P. E.: *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.
16. Eppstein, D.: Fast hierarchical clustering and other applications of dynamic closest pairs, in *Proceedings of the ACM Symposium on Discrete Algorithms (SODA'98)*, 1998.
17. Faloutsos, C.: Multiattribute hashing using gray codes, in *Proceedings of the ACM International Conference on Managment of Data (SIGMOD'86)*, 1986, pp. 227–238.
18. Faloutsos, C. and Roseman, S.: Fractals for secondary key retrieval, in *Proceedings of the ACM International Conference on Principles of Database Systems (PODS'89)*, 1989, pp. 247–252.
19. Gionis, A., Indyk, P. and Motwani, R.: Similarity search in high dimensional via hashing, in *Proceedings of the 25th International Conference on Very Large Databases (VLDB'99)*, 1999.
20. Hartigan, J. A.: *Clustering Algorithms*, Wiley, New York, 1975.
21. Hjaltason, G. R. and Samet, H.: Incremental distance join algorithms for spatial databases, in *Proceedings of the ACM International Conference on Managment of Data (SIGMOD'98)*, 1998, pp. 237–248.
22. Indyk, P.: Sublinear time algorithm for metric space problems, in *ACM Symposium on Theory of Computing*, 1999, pp. 428–434.
23. Indyk, P.: High dimensional computational geometry, PhD thesis, Stanford University, September 2000.
24. Jagadish, H. V.: Linear clustering of objects with multiple atributes, in *Proceedings of the ACM International Conference on Managment of Data (SIGMOD'90)*, 1990, pp. 332–342.
25. Katoh, N. and Iwano, K.: Finding k furthest pairs and k closest/farthest bichromatic pairs for points in the plane, in *Proceedings of the 8th ACM Symposium on Computational Geometry*, 1992, pp. 320–329.
26. Khuller, S. and Matias, Y.: A simple randomized sieve algorithm for closest-pair problem, *Information and Computation* **118**(1) (1995), 34–37.
27. Knuth, D. E.: *The Art of Computer Programming*, Addison-Wesley, 1998.
28. Lenhof, H. P. and Smid, M.: Enumerating the k closest pairs optimally, in *Proceedings of the 33rd IEEE Symposium on Foundation of Computer Science (FOCS92)*, 1992, pp. 380–386.
29. Lopez, M. and Liao, S.: Finding k-closest-pairs efficiently for high-dimensional data, in *Proceedings of the 12th Canadian Conference on Computational Geometry (CCCG)*, 2000, pp. 197–204.
30. Moon, B., Jagadish, H. V., Faloutsos, C. and Saltz, J. H.: Analysis of the clustering properties of the Hilbet space-filling curve, *IEEE Trans. Knowledge Data Eng.* **13**(1) (2001), 124–141.
31. Nanopoulos, A., Theodoridis, Y. and Manolopoulos, Y.: C^2P: Clustering based on closest pairs, in *Proceedings of the 27th Very Large Database Conference (VLDB'01)*, 2001, pp. 331–340.
32. Preparata, F. P. and Shamos, M. I.: *Computational Geometry. An Introduction*, Springer-Verlag, New York, 1985.
33. Rabin, M. O.: Probabilistic algorithms, in J. F. Traub (ed.), *Algorithms and Complexity: New Directions and Recent Results*, Academic Press, 1976, pp. 21–39.

34. Shim, K., Guha, S. and Rastogi, R.: Cure: An efficient clustering algorithm for large databases, in *Proceedings of the ACM International Conference on Managment of Data (SIGMOD'86)*, 1998, pp. 73–84.
35. Sagan, H.: *Space Filling Curves*, Springer-Verlag, 1994.
36. Schwarz, C., Smid, M. and Snoeyink, J.: An optimal algorithm for the on-line closest-pair problem, in *Proceedings of the 8th ACM Symposium on Computational Geometry*, 1992, pp. 330–336.
37. Shepherd, J., Zhu, X. and Megiddo, N.: A fast indexing method for multidimensional nearest neighbor search, in *Proceedings of SPIE Vol. 3656, Storage and Retrieval for Image and Video Databases*, 1998, pp. 350–355.
38. Smid, M.: Closest-point problems in computational geometry, in J. Sack and J. Urrutia (eds), *Handbook of Computational Geometry*, Elsevier Science, Amsterdam, 1999, pp. 877–935.
39. Strongin, R. G. and Sergeyev, Y. D.: *Global Optimization with Non-Convex Costraints*, Kluwer Academic Publishers, 2000.
40. Yang, C. and Lin, K.: An index structure for improving closest pairs and related join queried in spatial databases, in *Proceedings of the International Database Engineering and Applications Symposium IDEAS'02*, 2002.