# Approximate $k$-Closest-Pairs with Space Filling Curves

Fabrizio Angiulli and Clara Pizzuti

ISI-CNR
c/o DEIS, Università della Calabria
87036 Rende (CS), Italy
{angiulli,pizzuti}@isi.cs.cnr.it

**Abstract.** An approximate algorithm to efficiently solve the *k-Closest-Pairs* problem in high-dimensional spaces is presented. The method is based on dimensionality reduction of the space $\mathbb{R}^d$ through the Hilbert space filling curve and performs at most $d+1$ scans of the data set. After each scan, those points whose contribution to the solution has already been analyzed, are eliminated from the data set. 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. Although we are able to guarantee an $\mathcal{O}(d^{1+\frac{1}{t}})$ approximation to the solution, where $t = 1, \ldots, \infty$ denotes the used $L_t$ metric, experimental results give the exact $k$-Closest-Pairs for all the data sets considered and show that the pruning of the search space is effective.

## 1 Introduction

The *k-Closest-Pairs problem* is a classical problem in computational geometry that received a lot of attention in other fields such as statistics [1], pattern recognition [7] and, more recently, spatial databases [4] and data mining [14]. In this problem we are given a set $\mathcal{D}$ of $n$ points in $\mathbb{R}^d$ and an integer $k$, $1 \leq k \leq \frac{n(n-1)}{2}$. The *k-Closest-Pairs* ($CP_k$) problem consists in finding the *k closest pairs of $\mathcal{D}$ under the $L_t$ metric*. The $L_t$ distance $d_t(p,q)$ between two points $p = (p_1, \ldots, p_d)$ and $q = (q_1, \ldots, q_d)$ is defined as $d_t(p,q) = (\sum_{i=1}^{d} |p_i - q_i|^t)^{1/t}$ for $1 \leq t < \infty$ and $\max_{1 \leq i \leq d} |p_i - q_i|$, for $t = \infty$. We denote by $CP_k(\mathcal{D})$ the output of the $CP_k$ problem on an input data set $\mathcal{D}$. Several algorithms have been given for this problem [2, 9, 10, 12]. A comprehensive overview on algorithms regarding closest-pair and related problems can be found in [18]. When the dimension $d$ is constant, such algorithms are very efficient. However a thorough analysis reveals that their time requirements grow exponentially with $d$. As a consequence, when the dimensionality $d$ is a parameter, the *brute force* approach, i.e. the simple enumeration of all the pairs of points, 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. Then, it makes sense to search for an approximate solution in a reasonably amount of time. The paper intends to give a contribution in this setting.

In [12], Lopez and Liao present 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.

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, analogously to [12], is based on dimensionality reduction of the space $\mathbb{R}^d$ through a space filling curve, in our case the Hilbert curve, and performs at most $d+1$ scans of the data set by using shifted copies of the input data set. Our approach however, differs substantially from that of [12] since it is based on a property that allows us to eliminate those points whose contribution to the solution has already been analyzed. At each iteration, after the transformation of the $d$-dimensional points into one dimensional points, the input points are examined with respect to the order induced by the Hilbert curve. By exploring the $2m$ ($m$ predecessors and $m$ successors) nearest approximate neighbors of each point $p$ on the line, we define a property that allows us to determine, among such $2m$ points, the true $l \leq 2m$ nearest neighbors of $p$ and to establish a lower bound to the distance from $p$ to the $(l+1)$-th nearest neighbor. Such a property is very useful since it permits to detect the points that do not need to be considered any more and thus that can be discarded. The main contributions of our 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 that performs no more than $d+1$ scans of the input data set in at most $\mathcal{O}(dn(d(\log n + m) + m\log k))$ time, which is able to eliminate points from the data set after each scan. The pruning of the search space is *lossless* thus our algorithm could constitute a *preprocessing step* for an exact algorithm. Furthermore we define the worst case condition allowing a point to be pruned from the input data set.
- Although the algorithm is approximate, and guarantees an $\mathcal{O}(d^{1+\frac{1}{t}})$ approximation to the solution, when a particular condition is satisfied, we are able to state that the returned solution is the exact one.
- The experimental results (a) give the *exact closest pairs* for all the considered high dimensional data sets and for different values of $k$, (b) confirm that the pruning of the search space is effective, that is that 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.

The paper is organized as follows. Next section gives an overview of space filling curves. Section 3 gives definitions and properties necessary to introduce the algorithm. Section 4 presents the method. In Section 5 experimental results on several data sets are reported.

## 2 Space filling curves

The concept of *space-filling curve* came out in the 19-th century and is accredited to Peano [16] 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.* 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 sub-squares, each subinterval can be mapped onto one of the sub-squares. Sub-squares are ordered such that each pair of consecutive sub-squares shares a common edge. If this process is continued ad infinitum, $I$ and $Q$ are partitioned into $2^{2h}$ replicas for $h = 1, 2, 3\dots$ Figure 1 (on the left) shows the second step of this process. Sub-squares are arranged so that the inclusion relationships and adjacency property are always preserved. In practical applications the partitioning process is terminated after $h$ steps to give an approximation of a space-filling curve of order $h$. For $h \geq 1$ and $d \geq 2$, 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 $D$ be the set $\{p \in \mathbb{R}^d : 0 \leq p_i \leq 1, 1 \leq i \leq d\}$ and $p$ a $d$-dimensional point in $D$. The inverse image of $p$ under this mapping is called its *Hilbert value* and is denoted by $\mathcal{H}(p)$. Let $\mathcal{D}$ be a set of 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 [5, 6, 8, 13, 19]. 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 [17, 12, 19]. 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 [12], the number of shifts depends on the dimension $d$. Given a data set $\mathcal{D}$ and the vector $v^{(j)} = (j/(d+1), \dots, j/(d+1)) \in \mathbb{R}^d$, each point $p \in \mathcal{D}$ can be translated

$d+1$ times along the main diagonal in the following way: $p^j = p + v^{(j)}$, for $j = 0, \ldots, d$. The shifted copies of points thus belong to $[0,2)^d$ and, for each $p$, $d+1$ Hilbert values in the interval $[0,2)$ can be computed. In this paper we make use of this family of shifts to overcome the loss of the nearness property.

## 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. Without loss of generality, we assume that the given data set $\mathcal{D}$ has been normalized so that it is constituted by points in $D$. 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 $\mathcal{H}_h^d$.

**Definition 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 $a_i$, $0 \le i < d$, and $s$ are in $\mathbb{N}$.

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

**Definition 2.** Given three points $p$, $q_1$, 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 $q_2$.

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

**Definition 3.** Let $p$ be a point, and let $r$ be the side of a $r$-region. We denote by $MinDist(p,r)$ the value $\min_{i=1}^{d}\{\min\{\mathrm{mod}(p_i,r), r - \mathrm{mod}(p_i,r)\}\}$ 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 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.

**Definition 4.** Let $p$ be a point in $\mathbb{R}^d$, and let $r$ be a non-negative real number. $\mathcal{B}(p,r)$ is defined as the set of points whose distance from $p$ is less than or equal to $r$, i.e. $\mathcal{B}(p,r) = \{q \in \mathbb{R}^d \mid \mathrm{d_t}(p,q) \le r\}$.

**Definition 5.** 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 $q$ of $\mathcal{D}$ such that there exist $m-1$ points $q_1, \ldots, q_{m-1}$ in $\mathcal{D}$ with $\mathrm{d_t}(p,q_1) \le \mathrm{d_t}(p,q_2) \le \ldots \le \mathrm{d_t}(p,q_{m-1}) \le \mathrm{d_t}(p,q)$, and for each $x \in \mathcal{D}, x \ne p, q_i, i = 1, \ldots, m-1$, $\mathrm{d_t}(p,x) \ge \mathrm{d_t}(p,q)$.

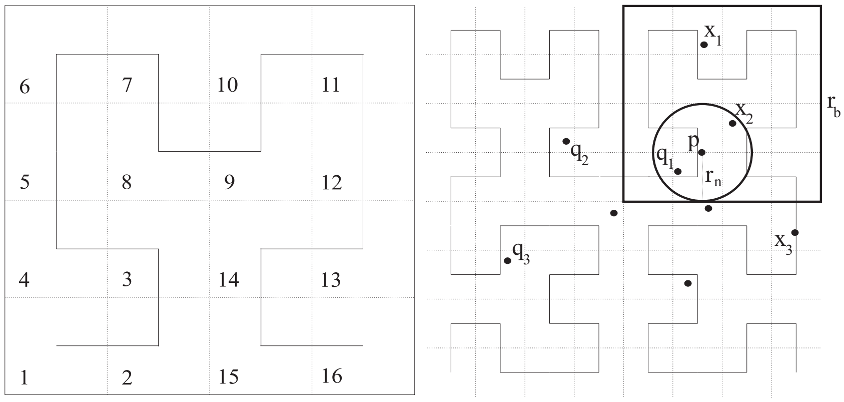


**Fig. 1.** Hilbert curve (left) and an example of application of Lemma 1 with $m = 3$ (right).

**Definition 6.** 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

$$\mathcal{I}_\mathcal{H}(p,m) = \{q_m, \ldots, q_1, x_1, \ldots, x_m \mid q_m = \mathcal{H}_{pred}(p,m), \ldots, q_1 = \mathcal{H}_{pred}(p,1),$$
$$x_1 = \mathcal{H}_{succ}(p,1), \ldots, x_m = \mathcal{H}_{succ}(p,m)\}$$

Notice that $\mathcal{I}_\mathcal{H}(p,m)$ are the $2m$ nearest neighbors of $p$ with respect to the Hilbert order and thus they constitute an approximation of the true closest neighbors. The following lemma allows us to determine, among such $2m$ points, the exact $l \leq 2m - 2$ 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 $\mathrm{NN}_{l+1}^\mathcal{D}(p)$.

**Lemma 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 $r_b = MaxReg(p, q_m, x_m)$, $r_n = MinDist(p, r_b)$, 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. $\mathrm{d}_t(p, \mathrm{NN}_{|S_n|+1}^\mathcal{D}(p)) > r_n$.

Figure 1 shows an example of application of Lemma 1, and shows the $r_b$-region, the distance $r_n$, and $\mathcal{B}(p, r_n)$, for $m = 3$ and $t = 2$.

## 4 Algorithm

In this section we give the description of the ASP algorithm. The method does $d+1$ scans of the input data set. During each scan it works on a shifted version of the data set. Exploiting the property defined in Lemma 1, at each iteration, it eliminates those points that do not need to be considered any more. This pruning of the search space allows to obtain a fast approximate algorithm for the $k$-Closest-Pairs problem in high dimensional spaces, as experimental results show in Section 5. 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 its maximum $d$-dimensional neighborhood explored.

**Definition 7.** A *point feature* $f$ is a triple $\langle point, h\_key, lb\rangle$, where *point* is a point in $[0,2)^d$, *h_key* 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 with the notation $f.point$, $f.h\_key$, and $f.lb$ we refer to the *point*, *h_key*, and *lb* value of the point feature $f$ respectively.

```
algorithm ASP ;
input:  D = {p_1, ..., p_n} : dataset;
k : integer; (* closest-pairs requested *)
m : integer; (* lookups allowed *)
begin
   INIT(Q_CP, k);
   for i := 1 to n do begin
      F_i.point := p_i; F_i.lb := 0;
   end;
   j := 0;
   while F ≠ ∅ and j ≤ d do begin
      HILBERT(F, v^(j));
      SCAN(F, v^(j), m, Q_CP);
      PRUNE(F, MAX(Q_CP));
      j := j + 1;
   end;
   return P(Q_CP);
end.
```

```
procedure SCAN(var F : features;
      v : point;
      m : integer;
      var Q_CP : queue);
begin
   for i := 1 to |F| do begin
      for j := i + 1 to min{i + m, |F|}
      do begin
         δ := d_t(F_i.point, F_j.point);
         UPDATE(Q_CP, F_i.point, F_j.point, δ);
      end;
      r_b := MaxReg(F_i.point + v,
      F_{i-m}.point + v, F_{i+m}.point + v);
      r_n := MinDist(F_i.point + v, r_b);
      if r_n > F_i.lb then
         F_i.lb := r_n;
   end;
end; { SCAN }
```

**Fig. 2.** The algorithm ASP and the procedure SCAN

The algorithm is reported in Figure 2. 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 $\delta$ is the associated distance $\mathrm{d_t}(p,q)$. Every triple contained in $Q_{CP}$ 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 $\mathcal{D}$, the number $k$ of closest pairs to find and the number $m$ of neighbors that the algorithm is allowed to consider on the line ($m$ predecessors and $m$ successors). The procedure INIT initializes the priority queue $Q_{CP}$ as an empty queue of size at most $k$, 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 value $\mathcal{H}(F_i.point + v^{(j)})$ of each point feature $F_i$ of $F$, places this value in $F_i.h\_key$, and sorts the point features in the list $F$ using as order key the values $F_i.h\_key$. Thus it performs the Hilbert mapping of a shifted version of the input data set. 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 2. To update the priority queue $Q_{CP}$ the procedure UPDATE is employed. UPDATE($Q$, $p$, $q$, $\delta$) 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 $\delta$ 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 $r_n$.

The procedure PRUNE deletes from $F$ all the point features $F_i$ such that $F_i.lb > \mathrm{MAX}(Q_{CP})$ where $\mathrm{MAX}(Q_{CP}) = \max\{\delta \mid \langle p,q,\delta\rangle \in Q_{CP}\}$. 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. Now we state the main property of the algorithm.

**Theorem 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:*

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

Theorem 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 $\epsilon_d$ the value $2d^{\frac{1}{t}}(2d+1)$, and by $\delta_k$ the distance between the two points composing the $k$-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 then we can assert that the solution returned is the exact one.

**Corollary 1.** *$F^* = \emptyset$ implies that $CP_k(\mathcal{D}) = \mathcal{P}(Q^*_{CP})$.*

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

**Definition 8.** A point $p$ is *$\alpha$-central* in an $r$-region iff for each $i = 1, \dots, d$, we have $\alpha r \leq \mathrm{mod}(p_i, r) < (1-\alpha)r$, where $0 \leq \alpha < 0.5$.

The following result is from [3].

**Lemma 2.** *Suppose $d$ is even. Then, for any point $p \in \mathbb{R}^d$ and $r = 2^{-s}$ ($s \in \mathbb{N}$), there exists $j \in \{0, \dots, d\}$ such that $p + v^{(j)}$ is $1/(2d+2)$-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 $\{p + v^{(0)}, \dots, p + v^{(d)}\}$, then, in at least one of these shifts, this point must become *sufficiently* central in an $r$-region, for each admissible value of $r$. 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 [12], that employed the same family of shifts.

**Theorem 2.** $\textsc{Max}(Q^*_{CP}) \leq \epsilon_d \delta_k$.

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

Thus, let $\Delta$ 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 3 asserts that the ability of the ASP algorithm to prune points increases when the distribution $\Delta$ accumulates around and above the value $\epsilon_d^2 \delta_k$.

Now we give time and space cost analysis of the algorithm. In the worst case the $\textsc{Hilbert}$ procedure requires $\mathcal{O}(dn \log n)$ time, while the procedure $\textsc{Prune}$ requires $\mathcal{O}(n)$ time. The procedure $\textsc{Scan}$ 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 $\mathcal{O}(n(d + m(d + \log k)))$ time. Thus, in the worst case the algorithm runs in $\mathcal{O}(dn(d(\log n + m) + m \log k))$ time. Since the algorithm enumerating all the possible pairs in the data set requires $\mathcal{O}(n^2(d+\log k))$ time, it must be the case that $k$ and $m$ are such that the following inequality holds

$$n \geq (d(d(\log n + m) + m \log k))/(d + \log k)$$

otherwise the algorithm could be outperformed by the brute force approach. 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 expected. Finally, regarding the space cost analysis, the algorithm requires $\mathcal{O}(n+k)$ space to store the list $F$ and the queue $Q_{CP}$.

## 5 Experimental results

We implemented the algorithm using the C programming language on a Pentium III 800MHz based machine having 256Mb of main memory. We used a 64 bit floating-point type to represent point coordinates and distances, and the 32th order approximation of the $d$-dimensional Hilbert curve to map the hypercube $[0,2)^d$ onto the set of integers $[0, 2^{32d})$. We set $m = k$ for all the experiments.

The data sets we used to test the algorithm are: *Cure* ($d = 2$, $n = 100000$), *ColorMoments* ($d = 9$, $n = 68040$), *CoocTexture* ($d = 16$, $n = 68040$), *ColorHistogram* ($d = 31$, $n = 68040$). Cure is the two dimensional very large synthetic data set described in [15] (referred as Data set 1) and widely used to test clustering algorithms. The other three data sets corresponds to a collection of real images. Each point of these data sets represents an image of the collection (see `http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.html` for more information). We consider only the first 31 dimensions of the 32 dimensional data set *ColorHistogram* as our actual implementation does not go beyond 31 dimensions.

We search the $k$ closest pairs under the $L_2$ metric. Although we are able to guarantee an $\mathcal{O}(d^{\frac{3}{2}})$ approximation to the solution, the algorithm calculates the *exact solution* for all the experiments, even when $F^*$ was not empty. Hence, we report only the evaluation of the *pruning effectiveness* and of the *running time* of the algorithm.

| *Dataset* \ $k$ | 1 | 10 | 100 | 1000 |
|---|---|---|---|---|
| *Cure* | 0 | 0 | 0 | 0 |
| *ColorMoments* | 0 | 0 | 0 | 35732 |
| *CoocTexture* | 0 | 0 | 0 | 25558 |
| *ColorHistogram* | 0 | 0 | 0 | 0 |

**Table 1.** The size of $F^*$

Table 1 reports the size of the list $F^*$ for different values of $k$. The results show the effectiveness of the algorithm in reducing the size of the input data set. In

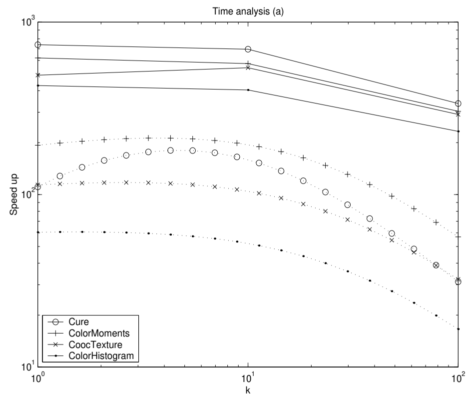


**Fig. 3.** Execution time

practice the algorithm is able to prune points having distance $\delta$ to their nearest-neighbor significantly more less than $\epsilon_d^2\delta_k$, the worst case stated by Theorem 3. For the experiments done we verified that the pruning effectiveness deteriorates when $\delta_k$ and the mode of the distribution $\Delta$ have the same order of magnitude (this is the case of $ColorMoments$ and $CoocTexture$ for $k = 1000$).

Figure 3 shows the *speed up* obtained in the experiments, i.e. the ratio $\frac{T}{T'}$ where $T'$ is the execution time of our algorithm and $T$ is the execution time of the brute force enumeration algorithm on the same input. We plot also the curves of the theoretical speed up associated to the data sets considered (represented as dotted curves), i.e. the speed up obtained in the worst case execution time of the algorithm. The experimental results show that in practice our algorithm outperforms brute force enumeration by some order of magnitude, and that it guarantees, in all the cases considered, a speed up greater than those predicted by the theory.

**Acknowledgments.** Special thanks to Jonathan Lawder who provided us his source code implementing the multidimensional Hilbert mapping.

## 6 Conclusions

In this paper an approximated algorithm, based on dimensionality reduction through space-filling curves, to solve the $k$-Closest- Pairs problem on high dimensional spaces has been presented. The method exploits a property of such curves to prune the search space. Experimental results give the exact closest pairs for all the considered data sets, confirm that the pruning is effective and show that the algorithm outperforms brute force enumeration by some order of magnitude. We intend to build a disk-based version of ASP to explore how the algorithm scales for data sets that do not fit into main memory. The introduced property can be fruitfully used for other classical proximity problems. Currently, we are investigating its application to different data mining tasks, such as the discovery of outliers.

## References


1. H. C. Andrews. *Introduction to Mathematical Techniques in Pattern Recognition.* Wiley-Interscience, New York, 1972.
2. J. L. Bentley and M. I. Shamos. Divide-and-conquer in multidimensional space. In *Proc. of the 8th ACM Symp. on Theory of Computing*, pages 220–230, 1996.
3. T. Chan. Approximate nearest neighbor queries revisited. In *Proc. of the 13th ACM Symp. on Computational Geometry*, pages 352–358, 1997.
4. A. Corral, Y. Manolopoulos, Y. Theodoridis, and M.Vassilakopoulos. Closest pair queries in spatial databases. In *Proc. ACM Int. Conf. on Managment of Data (SIGMOD'00)*, pages 189–2000, 2000.
5. C. Faloutsos. Multiattribute hashing using gray codes. In *Proceedings ACM Int. Conf. on Managment of Data (SIGMOD'86)*, pages 227–238, 1986.
6. C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. In *Proc. ACM Int. Conf. on Principles of Database Systems (PODS'89)*, pages 247–252, 1989.
7. J. A. Hartigan. *Clustering Algorithms.* Wiley, New York, 1975.
8. H.V. Jagadish. Linear clustering of objects with multiple atributes. In *Proc. ACM Int. Conf. on Managment of Data (SIGMOD'90)*, pages 332–342, 1990.
9. N. Katoh and K. Iwano. Finding $k$ furthest pairs and $k$ closest/farthest bichromatic pairs for points in the plane. In *Proc. of the 8th ACM Symp. on Computational Geometry*, pages 320–329, 1992.
10. H.P. Lenhof and M. Smid. Enumerating the $k$ closest pairs optimally. In *Proc. of the 33rd IEEE Symp. on Foundation of Computer Science (FOCS92)*, pages 380–386, 1992.
11. S. Liao, M. Lopez, and S. Leutenegger. High dimensional similarity search with space filling curves. In *Proc. of the 17th Int. Conf. on Data Engineering (ICDE)*, pages 615–622, 2001.
12. M. Lopez and S. Liao. Finding $k$-closest-pairs efficiently for high dimensional data. In *Proc. of the 12th Canadian Conf. on Computational Geometry (CCCG)*, pages 197–204, 2000.
13. B. Moon, H.V. Jagadish, C. Faloutsos, and J.H.Saltz. Analysis of the clustering properties of the Hilbert space-filling curve. *Technical Report 10, Department of Computer Science, University of Arizona, Tucson*, August 1999.
14. A. Nanopoulos, Y. Theodoridis, and Y. Manolopoulos. $C^2P$ : Clustering based on closest pairs. In *Proc. of the 27th Conf. on Very Large Database (VLDB'01)*, pages 331–340, 2001.
15. K. Shim S. Guha, R. Rastogi. Cure: An efficient clustering algorithm for large databases. In *Proc. ACM Int. Conf. on Managment of Data (SIGMOD'86)*, pages 73–84, 1998.
16. Hans Sagan. *Space Filling Curves.* Springer-Verlag, 1994.
17. J. Shepherd, X. Zhu, and N. Megiddo. A fast indexing method for multidimensional nearest neighbor search. In *Proc. of SPIE Vol. 3656, Storage and retrieval for image and video databases*, pages 350–355, 1998.
18. M. Smid. Closest-point problems in computational geometry. In *Tech. Report, Univ. Magdeburg, Germany*, pages 1–63, 1997.
19. Roman G. Strongin and Yaroslav D. Sergeyev. *Global Optimization with Non-Convex Costraints.* Kluwer Academic, 2000.