Fast Outlier Detection in High Dimensional Spaces

Fabrizio Angiulli and Clara Pizzuti

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

Abstract. In this paper we propose a new definition of distance-based outlier that considers for each point the sum of the distances from its k nearest neighbors, called weight. Outliers are those points having the largest values of weight. In order to compute these weights, we find the k nearest neighbors of each point in a fast and efficient way by linearizing the search space through the Hilbert space filling curve. The algorithm consists of two phases, the first provides an approximated solution, within a small factor, after executing at most $d + 1$ scans of the data set with a low time complexity cost, where d is the number of dimensions of the data set. During each scan the number of points candidate to belong to the solution set is sensibly reduced. The second phase returns the exact solution by doing a single scan which examines further a little fraction of the data set. Experimental results show that the algorithm always finds the exact solution during the first phase after $\overline{d} \ll d + 1$ steps and it scales linearly both in the dimensionality and the size of the data set.

1 Introduction

Outlier detection is an outstanding data mining task referred to as *outlier mining* that has a lot of practical applications such as telecom or credit card frauds, medical analysis, pharmaceutical research, financial applications. Outlier mining can be defined as follows: "Given a set of N data points or objects, and n, the expected number of outliers, find the top n objects that are considerably dissimilar with respect to the remaining data" [9]. Many data mining algorithms consider outliers as noise that must be eliminated because it degrades their predictive accuracy. For example, in classification algorithms mislabelled instances are considered outliers and thus they are removed from the training set to improve the accuracy of the resulting classifier [6]. However, as pointed out in [9], "one person's noise could be another person's signal", thus outliers themselves can be of great interest. The approaches to outlier mining can be classified in supervised-learning based methods, where each example must be labelled as exceptional or not, and the unsupervised-learning based ones, where the label is not required. The latter approach is more general because in real situations we do not have such information. Unsupervised-learning based methods for outlier detection can be categorized in several approaches. The first is *statistical-based* and assumes that the given data set has a distribution model. Outliers are those points that satisfies a discordancy test, that is that are significantly larger (or smaller) in relation to the hypothesized distribution [4]. In [20] a Gaussian mixture model to represent the normal behaviors is used and each datum is given a score on the basis of changes in the model. High score indicates high possibility of being an outlier. This approach has been combined in [19] with a supervised-learning based approach to obtain general patterns for outliers. *Deviation-based* techniques identify outliers by inspecting the characteristics of objects and consider an object that deviates from these features an outlier [3,16]. A completely different approach that finds outliers by observing *low dimensional projections* of the search space is presented in [1]. Yu et al. [7] introduced *FindOut*, a method based on wavelet transform, that identifies outliers by removing clusters from the original data set. Wavelet transform has also been used in [18] to detect outliers in stochastic processes. Another category is the *density-based*, presented in [5] where a new notion of local outlier is introduced that measures the degree of an object to be an outlier with respect to the density of the local neighborhood. This degree is called *Local Outlier Factor LOF* and is assigned to each object. The computation of LOFs, however, is expensive and it must be done for each object. To reduce the computational load, Jin et al. in [10] proposed a new method to determine only the top-n local outliers that avoids the computation of LOFs for most objects if $n \ll N$, where N is the data set size. *Distance-based* outlier detection has been introduced by Knorr and Ng [12] to overcome the limitations of statistical methods. A *distance-based* outlier is defined as follows: *A point p in a data set is an outlier with respect to parameters k and δ if no more than k points in the data set are at a distance of δ or less from p.* This definition of outlier has a number of benefits but, as observed in [14], it depends on the two parameters k and δ and it does not provide a ranking of the outliers. Furthermore the two algorithms proposed are either quadratic in the data set size or exponential in the number of dimensions, thus their experiments cannot go beyond five dimensions. In the work [14] the definition of outlier is modified to address these drawbacks and it is based on the distance of the k-th nearest neighbor of a point p, denoted with $D^k(p)$. The new definition of outlier is the following: *Given a k and n, a point p is an outlier if no more than n-1 other points in the data set have a higher value for D^k than p.* This means that the top n points having the maximum D^k values are considered outliers. The experiments presented, up to 10 dimensions, show that their method scales well. This definition is interesting but does not take into account the local density of points. The authors note that "points with large values for $D^k(p)$ have more sparse neighborhoods and are thus typically stronger outliers than points belonging to dense clusters which will tend to have lower values of $D^k(p)$." However, consider Figure 1. If we set $k = 10$, $D^k(p_1) = D^k(p_2)$, but we can not state that p_1 and p_2 can be considered being outliers at the same way.

In this paper we propose a new definition of outlier that is distance-based but that considers for each point p the sum of the distances from its k nearest neighbors. This sum is called the *weight* of p, $\omega_k(p)$, and it is used to rank

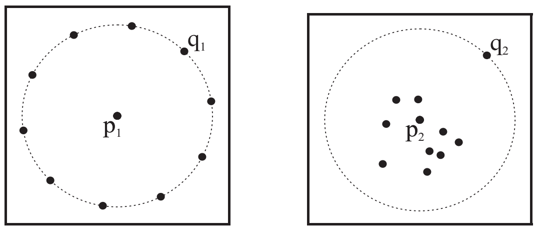

Fig. 1. Two points with same D^k values (k=10)

the points of the data set. Outliers are those points having the larger values of ω_k. In order to compute these weights, we find the k nearest neighbors of each point in a fast and efficient way by linearizing the search space. We fit the d-dimensional data set **DB** in the hypercube $D = [0,1]^d$, then we map D into the interval $I = [0,1]$ by using the *Hilbert space filling curve* and obtain the k nearest neighbors of each point by examining its predecessors and successors on I. The mapping assures that if two points are close in I, they are close in D too, although the reverse in not always true. To limit the loss of nearness, the data set is shifted $d+1$ times along the main diagonal of the hypercube $[0,2]^d$. The algorithm consists of two phases, the first provides an approximated solution, within a small factor, after executing at most $d+1$ scans of the data set with a low time complexity cost. During each scan a better lower bound for the weight of the k-th outlier of **DB** is obtained and the number of points candidate to belong to the solution set is sensibly reduced. The second returns the exact solution by doing a single scan which examines further a little fraction of the data set. However, as experimental results show, we always find the exact solution during the first phase after $\overline{d} \ll d+1$ steps.

It is worth to note that approaches based on wavelet transform apply this multi-resolution signal processing technique to transform the original space in a new one of the same dimension and find outliers in the transformed space at different levels of approximation. In our approach, however, space filling curves are used to map a multidimensional space in a one dimensional space to obtain the nearest neighbors of each point in a fast way, but the distance computation is done in the original space.

The paper is organized as follows. Section 2 gives definitions and properties necessary to introduce the algorithm and an overview of space filling curves. Section 3 presents the method. In Section 4, finally, experimental results on several data sets are reported.

2 Definitions and Notations

In this section we present the new definition of outlier and we introduce the notions that are necessary to describe our algorithm. The L_t *distance* between two points $p = (p_1, \ldots, p_d)$ and $q = (q_1, \ldots, q_d)$ is defined as $\mathrm{d}_t(p,q) = (\sum_{i=1}^{d} |p_i - q_i|^t)^{1/t}$ for $1 \leq t < \infty$, and $\max_{1 \leq i \leq d} |p_i - q_i|$ for $t = \infty$.

Let **DB** be a d-dimensional data set, k a parameter and let p be a point of **DB**. Then the *weight* of p in **DB** is defined as $\omega_k(p) = \sum_{i=1}^{k} \mathrm{d}_t(p, nn_i(p))$, where $nn_i(p)$ denotes the i-th nearest neighborhood of p in **DB**.

Given a data set **DB**, parameters k and n, a point $p \in \mathbf{DB}$ is the n-th outlier with respect to k, denoted as $outlier_k^n$, if there are exactly $n-1$ points q in **DB** such that $\omega_k(q) > \omega_k(p)$.

Given a data set **DB**, parameters k and n, we denote with Out_k^n the set of the top n outliers of **DB** with respect to k. Let Out^* be a set of n points of **DB** and ϵ a positive real number, we say that Out^* is an ϵ-*approximation* of Out_k^n if $\omega^* \epsilon \geq \omega^n$, where ω^* is $\min\{\omega_k(p) \mid p \in Out^*\}$ and ω^n is the weight of $outlier_k^n$.

Points in **DB** are thus ordered according to their weights $\omega_k(p)$, computed by using any L_t metrics. The n points Out_k^n having the maximum ω_k values are considered outliers. To compute the weights, the k nearest neighbors are obtained by using *space-filling curves*. The concept of *space-filling curve* came out in the 19-th century and it is accredited to Peano [15] who, in 1890, proved the existence of a continuous mapping from the interval $I = [0, 1]$ onto the square $Q = [0, 1]^2$. Hilbert in 1891 defined a general procedure to generate an entire class of space-filling curves. He 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 share a common edge. If this process is continued ad infinitum, I and Q are partitioned into 2^{2h} replicas for $h = 1, 2, 3 \ldots$ 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 **DB** 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 $\mathcal{H}(\mathbf{DB})$ the set $\{\mathcal{H}(p) \mid p \in \mathbf{DB}\}$ 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 $\mathcal{H}(\mathbf{DB})$ 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 [8,11,17]. 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 [13,17]. 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 [13], the number of shifts depends on the dimension d. Given a data set $\mathbf{DB}$ and the vector $v^{(j)} = (j/(d+1), \ldots, j/(d+1)) \in \mathbb{R}^d$, each point $p \in \mathbf{DB}$ 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.

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

Let p and q be two points. We denote by $\mathit{MinReg}(p,q)$ the side of smallest r-region containing both p and q. We denote by $\mathit{MaxReg}(p,q)$ the side of the greatest r-region containing p but not q.

Let p be a point, and let r be the side of an r-region. Then

$$\mathit{MinDist}(p,r) = \min_{i=1}^{d}\{\min\{p_i \bmod r, r - p_i \bmod r\}\}$$

$$\mathit{MaxDist}(p,r) = \begin{cases} \left(\sum_{i=1}^{d}(\max\{p_i \bmod r, r - p_i \bmod r\})^t\right)^{1/t} & \text{for } 1 \le t < \infty \\ \max_{i=1}^{d}\{\max\{p_i \bmod r, r - p_i \bmod r\}\} & \text{for } t = \infty \end{cases}$$

where $x \bmod r = x - \lfloor x/r \rfloor r$, and p_i denotes the value of p along the i-th coordinate, are respectively the perpendicular distance from p to the nearest face of the r-region of side r containing p, i.e. a lower bound for the distance between p and a point lying out of the above r-region, and the distance from p to the furthest vertex of the r-region of side r containing p, i.e. an upper bound for the distance between p and a point lying into the above r-region.

Let p be a point in $\mathbb{R}^d$, and let r be a non negative real. Then the d-dimensional *neighborhood* of p (under the L_t metric) of radius r, written $\mathcal{B}(p,r)$, is the set $\{q \in \mathbb{R}^d \mid \mathrm{d}_t(p,q) \le r\}$.

Let p, q_1, and q_2 be three points. Then

$$\mathit{BoxRadius}(p,q_1,q_2) = \mathit{MinDist}(p, \min\{\mathit{MaxReg}(p,q_1), \mathit{MaxReg}(p,q_2)\})$$

is the radius of the greatest neighborhood of p entirely contained in the greatest r-region containing p but neither q_1 nor q_2.

Lemma 1. *Given a data set* $\mathbf{DB}$*, a point p of* $\mathbf{DB}$*, two positive integers a and b, and the set of points*

$$I = \{\mathcal{H}_{pred}(p,a), \ldots, \mathcal{H}_{pred}(p,1), \mathcal{H}_{succ}(p,1), \ldots, \mathcal{H}_{succ}(p,b)\}$$

let r be $\mathit{BoxRadius}(p, \mathcal{H}_{pred}(p,a-1), \mathcal{H}_{succ}(p,b+1))$ and $S = I \cap \mathcal{B}(p,r)$. Then

1. *The points in S are the true first $|S|$ nearest-neighbors of p in* **DB***;*
2. $\mathrm{d}_t(p, nn_{|S|+1}(p)) > r$.

The above Lemma allows us to determine, among the $a+b$ points, nearest neighbors of p with respect to the Hilbert order (thus they constitute an approximation of the true closest neighbors), the exact $|S| \leq a+b$ nearest neighbors of p and to establish a lower bound to the distance from p to the $(|S|+1)$-th nearest neighbor. This result is used in the algorithm to estimate a lower bound to the weight of any point p.

3 Algorithm

In this section we give the description of the *HilOut* algorithm. The method consists of two phases, the first does at most $d+1$ scans of the input data set and guarantees a solution that is an $k\epsilon_d$-approximation of Out_k^n, where $\epsilon_d = 2d^{\frac{1}{t}}(2d+1)$ (for a proof of this statement we refer to [2]), with a low time complexity cost. The second phase does a single scan of the data set and computes the set Out_k^n. At each scan *HilOut* computes a lower bound and an upper bound to the weight ω_k of each point and it maintains the n greatest lower bound values in the heap WLB. The n-th value ω^* in WLB is a lower bound to the weight of the n-th outlier and it is used to detect those points that can be considered candidate outliers. The upper and lower bound of each point are computed by exploring a neighborhood of the point on the interval I. The neighborhood of each point is initially set to $2k$, then it is widened, proportionally to the number of remaining candidate outliers, to obtain a better estimate of the true k nearest neighbors. At each iteration, as experimental results show, the number of candidate outliers sensibly diminishes. This allows the algorithm to find the exact solution in few steps, in practice after $\overline{d}$ steps with $\overline{d} \ll d+1$.
Before starting with the description, we introduce the concept of *point feature*.
A *point feature* f is a 7-tuple $\langle point, hilbert, level, weight, weight_0, radius, count\rangle$ where *point* is a point in $[0,2)^d$, *hilbert* 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 $[0, 2^{hd})$, *level* is the order of the smallest r-region containing both *point* and its successor in **DB** (with respect to the Hilbert order), *weight* is an upper bound to the weight of *point* in **DB**, *radius* is the radius of a d-dimensional neighborhood of *point*, $weight_0$ is the sum of the distances between *point* and each point of **DB** lying in the d-dimensional neighborhood of radius *radius* of *point*, while *count* is the number of these points.
In the following with the notation $f.point$, $f.hilbert$, $f.level$, $f.weight$, $f.weight_0$, $f.radius$ and $f.count$ we refer to the *point*, *hilbert*, *level*, *type*, *weight*, $weight_0$, *radius*, and *count* value of the point feature f respectively. Let f be a point feature, we denote by $wlb(f)$ the value $f.weight_0 + (k - f.count) \times f.radius$. $wlb(f)$ is a lower bound to the weight of $f.point$ in **DB**. The algorithm, reported in Figure 2, receives as input a data set **DB** of N points in the hypercube $[0,1]^d$,

the number n of top outliers to find and the number k of neighbors to consider. The data structures employed are the two heaps of n point features OUT and WLB, the set TOP, and the list of point features PF. At the end of each iteration, the features stored in OUT are those with the n greatest values of the field *weight*, while the features f stored in WLB are those with the n greatest values of $wlb(f)$. TOP is a set of at most $2n$ point features which is set to the union of the features stored in OUT and WLB at the end of the previous iteration. PF is a list of point features. In the following, with the notation PF_i we mean the i-th element of the list PF.
First, the algorithm builds the list PF associated to the input data set, i.e. for each point p of **DB** a point feature f with $f.point = p$, $f.weight = \infty$, and the other fields set to 0, is inserted in PF, and initializes the set TOP and the global variables ω^*, N^*, and n^*. ω^* is a lower bound to the weight of the $outlier^n_k$ in **DB**. This value, initially set to 0, is then updated in the procedure *Scan*. N^* is the number of point features f of PF such that $f.weight \geq \omega^*$. The points whose point feature satisfies the above relation are called *candidate outliers* because the upper bound to their weight is greater than the current lower bound ω^*. This value is updated in the procedure *Hilbert*. n^* is the number of true outliers in the heap OUT. It is updated in the procedure *TrueOutliers* and it is equal to $|\{f \in OUT \mid wlb(f) = f.weight \wedge f.weight \geq \omega^*\}|$. The main cycle, consists of at most $d+1$ steps. We explain the single operations performed during each step of this cycle.
Hilbert. The *Hilbert* procedure calculates the value $\mathcal{H}(PF_i.point + v^{(j)})$ of each point feature PF_i of PF, places this value in $PF_i.hilbert$, and sorts the point features in the list PF using as order key the values $PF_i.hilbert$. After sorting, the procedure *Hilbert* updates the value of the field *level* of each point feature. In particular, the value $PF_i.level$ is set to the order of the smallest r-region containing both $PF_i.point$ and $PF_{i+1}.point$, i.e. to $MinReg(PF_i.point, PF_{i+1}.point)$, for each $i = 1, \ldots, N-1$. For example, consider figure 3 where seven points in the square $[0,1]^2$ are consecutively labelled with respect to the Hilbert order. Figure 3 (b) highlights the smallest r-region containing the two points 5 and 6 while Figure 3 (c) that containing the two points 2 and 3. The values of the levels associated with the points 5 and 2 are thus three and one because the order of corresponding r-regions are $-\log_2 2^{1-4} = 3$ and $-\log_2 2^{1-2} = 1$ respectively. On the contrary, the smallest r-region containing points 1 and 2 is all the square.
Scan. The procedure *Scan* is reported in Figure 2. This procedure performs a sequential scan of the list PF by considering only those features that have a weight upper bound not less than ω^*, the lower bound to the weight of $outlier^n_k$ of **DB**. These features are those candidate to be outliers, the others are simply skipped. If the value $PF_i.count$ is equal to k then $F_i.weight$ is the true weight of $PF_i.point$ in **DB**. Otherwise $PF_i.weight$ is an upper bound for the value $\omega_k(PF_i.point)$ and it could be improved. For this purpose the function *FastUpperBound* calculates a novel upper bound ω to the weight of $PF_i.point$, given by $k \times MaxDist(PF_i.point, 2^{-level})$, by examining k points among its successors and predecessors to find *level*, the order of the smallest r-region con-

```
$HilOut$ ($\mathbf{DB}$, $n$, $k$)
{
  $Initialize(PF, \mathbf{DB})$;

  /* First Phase */
  $TOP = \emptyset$;
  $N^* = N$; $n^* = 0$; $\omega^* = 0$;
  $j = 0$;
  while ($j \leq d$ && $n^* < n$) {
    $Initialize(OUT)$;
    $Initialize(WLB)$;
    $Hilbert(v^{(j)})$;
    $Scan(v^{(j)}, \frac{kN}{N^*})$;
    $TrueOutliers(OUT)$;
    $TOP = OUT \cup WLB$;
    $j = j + 1$;
  }

  /* Second Phase */
  if ($n^* < n$)
    $Scan(v^{(d)}, N)$;
  return $OUT$;
}
```

```
$Scan(v, k_0)$
{
  for ($i = 1$; $i \leq N$; $i$++) if ($PF_i.weight \geq \omega^*$) {
    if ($PF_i.count < k$) {
      $\omega = FastUpperBound(i)$;
      if ($\omega < \omega^*$) $PF_i.weight = \omega$ else {
        $maxc = \min(2k_0, N)$;
        if ($PF_i \in TOP$) $maxc = N$;
        $InnerScan(i, maxc, v, NN)$;
        if ($NN.radius > PF_i.radius$) {
          $PF_i.radius = NN.radius$;
          $PF_i.weight_0 = NN.weight_0$;
          $PF_i.count = NN.count$;
        }
        if ($NN.weight < PF_i.weight$)
          $PF_i.weight = NN.weight$;
      }
    }
    $Update(OUT, PF_i)$;
    $Update(WLB, wlb(PF_i))$;
    $\omega^* = Max(\omega^*, Min(WLB))$;
  }
}
```

Fig. 2. The algorithm *HilOut* and the procedure *Scan*

taining both $PF_i.point$ and other k neighbors. If ω is less than ω^*, no further elaboration is required. Otherwise the procedure *InnerScan* returns the data structure NN which has the fields $NN.weight$, $NN.weight_0$, $NN.radius$ and $NN.count$. If $NN.radius$ is greater than $PF_i.radius$ then a better lower bound for the weight of $PF_i.point$ is available, and the fields $radius$, $weight_0$, and $count$ of PF_i are updated. Same considerations hold for the value $PF_i.weight$. Finally, the heaps WLB and OUT process $wlb(PF_i)$ and PF_i respectively, and the value ω^* is updated.

InnerScan. This procedure takes into account the points whose Hilbert value lies in a one dimensional neighborhood of the integer value $PF_i.hilbert$. In particular, if PF_i belongs to TOP, then the size of the above neighborhood, stored in $maxc$ is at most N, otherwise this size is at most $2kN/N^*$, i.e. it is inversely proportional to the number N^* of candidate outliers. This procedure manages a data structure NN constituted by a heap of k real numbers and the fields $NN.weight$, $NN.weight_0$, $NN.count$, and $NN.radius$. At the end of *InnerScan*, NN contains the k smallest distances between the point $PF_i.point$ and the points of the above defined one dimensional neighborhood, $NN.radius$ is the radius of the d-dimensional neighborhood of $PF_i.point$ explored when considering these points, calculated as in Lemma 1, $NN.weight$ is the sum of the elements stored in the heap of NN, $NN.weight_0$ is the sum of the elements stored in the heap of NN which are less than or equal to $NN.radius$, while $NN.count$ is their number. Thus *InnerScan* returns a new upper bound and a new lower bound

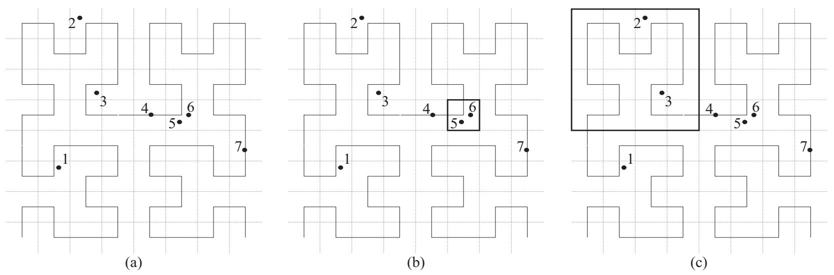

Fig. 3. The *level* field semantics

for the weight of $PF_i.point$. We note that the field *level* of each point feature is exploited by *InnerScan* to determine in a fast way if the exact k nearest neighbors of such point have already been encountered (see [2] for a detailed description). The main cycle of the algorithm stops when $n^* = n$, i.e. when the heap OUT is equal to the set of top n outliers, or after $d+1$ iterations. At the end of the first phase, the heap OUT contains a $k\epsilon_d$-approximation of Out_k^n. Finally, if $n^* < n$, that is if the number of true outliers found by the algorithm is not n, then a final scan computes the exact solution. This terminates the description of the algorithm.

As for the time complexity analysis, the time complexity of the first phase of the algorithm is $dN(d \log N + (n+k)(d + \log k))$. Let N^* be the number of candidate outliers at the end of the first phase. Then the time complexity of the second phase is $N^*(\log n + N^*(d + \log k))$.

4 Experimental Results and Conclusions

We implemented the algorithm using the C programming language on a Pentium III 850MHz based machine having 512Mb of main memory. We used a 64 bit floating-point type to represent the coordinates of the points and the 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 studied the behavior of the algorithm when the dimensionality d and the size N of the data set, the number n of top outliers we are searching for, the number k of nearest neighbors to consider and the metric L_t are varied. In particular, we considered $d \in \{2, 10, 20, 30\}$, $N \in \{10^3, 10^4, 10^5, 10^6\}$, $n, k \in \{1, 10, 100, 1000\}$, and the metrics L_1, L_2 and L_∞. We also studied how the number of candidate outliers decreases during the execution of the algorithm. To test our algorithm, we used three families of data sets called GAUSSIAN, CLUSTERS and DENSITIES. A data set of the GAUSSIAN family is composed by points generated from a normal distribution and scaled to fit into the unit hypercube. A data set of the CLUSTERS family is composed by 10 hyper-spherical clusters, formed by the same number of points generated from a normal distribution, having diameter 0.05 and equally spaced along the main diagonal of the unit hypercube. Each cluster is surrounded by 10 equally spaced outliers lying on a circumference of radius 0.1 and center in the cluster center. A data set of the DENSITIES family is composed by 2 gaussian clusters composed by the same number of points but having different standard deviations (0.25 and 0.75 respectively). The data sets of the same family differs only for their size N and for their dimensionality d. Figure 4 (a) shows the two dimensional GAUSSIAN data set together with its top 100 outliers (for $k = 100$) with $N = 10000$ points. In all the experiments considered, the algorithm terminates with the exact solution after executing a number of iterations much less than $d + 1$. Thus, we experimentally found that in practice the algorithms behaves as an exact algorithm without the need of the second phase. The algorithm exhibited the same behavior on all the considered data set families. For the lack of space, we report only the experiments relative to the GAUSSIAN data set (see [2] for a detailed description). Figures 4 (b) and (c) show the execution times obtained respectively varying the dimensionality d and the size N of the data set. The curves show that the algorithm scales linearly both with respect to the dimensionality and the size of the data set. Figures 4 (d) and (e) report the execution times obtained varying the number n of top outliers and the type k of outliers respectively. In the range of values considered the algorithm appears to be slightly superlinear. Figure 4 (f) illustrates the execution times corresponding to different values t of the metric L_t. Also in this case the algorithm scales linearly in almost all the experiments. The algorithm scales superlinearly only for L_∞ with the GAUSSIAN data set. This happens since, under the L_∞ metric, the points of the GAUSSIAN data set tend to have the same weight as the dimensionality increases. Finally, we studied how the number of candidate outliers decreases during the algorithm. Figure 4 (g) reports, in logarithmic scale, the number of candidate outliers at the beginning of each iteration for the the thirty dimensional GAUSSIAN data set and for various values of the data set size N. These curves show that, at each iteration, the algorithm is able to discharge from the set of the candidate outliers a considerable fraction of the whole data set. Moreover, the same curves show that the algorithm terminates, in all the considered cases, performing less than 31 iterations (5, 7, 10 and 13 iterations for N equal to 10^3, 10^4, 10^5 and 10^6 respectively). Figure 4 (h) reports, in logarithmic scale, the number of candidate outliers at the beginning of each iteration for various values of the dimensionality d of the GAUSSIAN data set with $N = 100000$. We note that, in the considered cases, if we fix the size of the data set and increase its dimensionality, then the ratio $\overline{d}/(d + 1)$, where $\overline{d}$ is the number of iterations needed by the algorithm to find the solution, sensibly decreases, thus showing the very good behavior of the method for high dimensional data sets.

To conclude, we presented a distance-based outlier detection algorithm to deal with high dimensional data sets that scales linearly with respect to both the dimensionality and the size of the data set. We presented experiments up to 1000000 of points in the 30-dimensional space. We are implementing a disk-based version of the algorithm to deal with data sets that cannot fit into main memory.

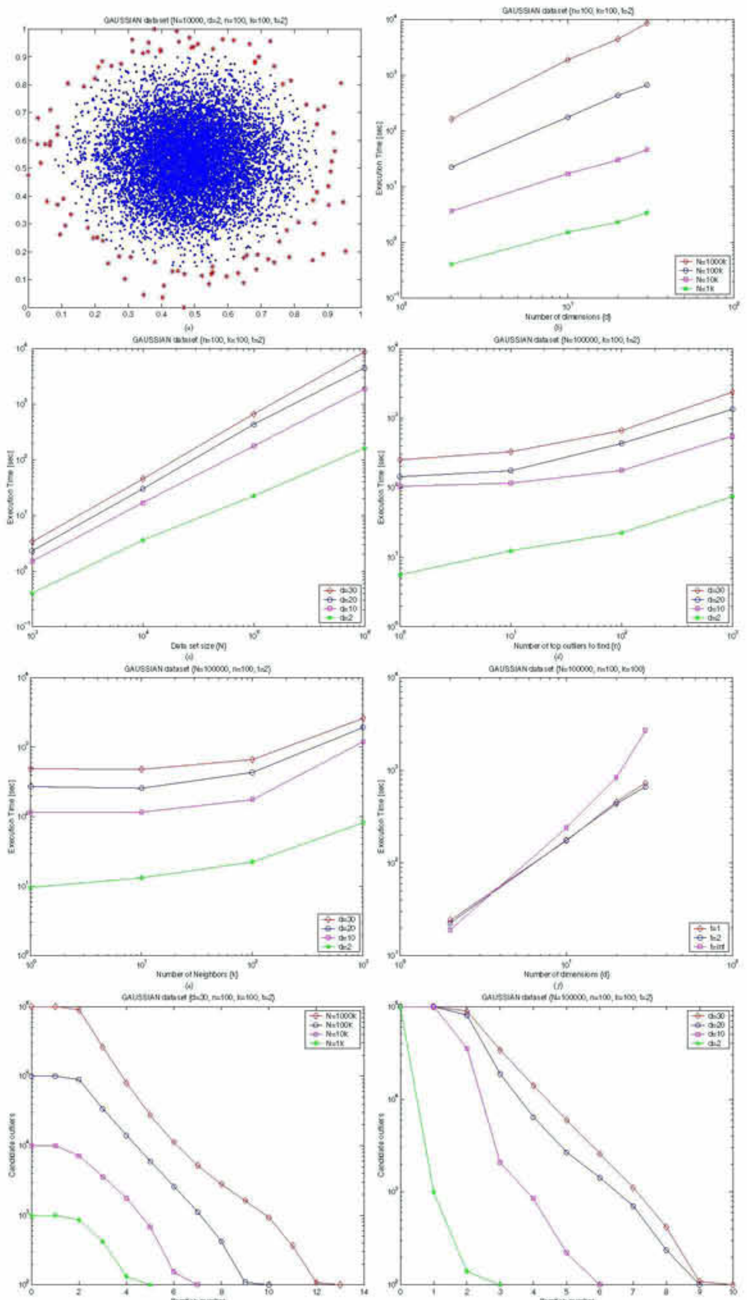

Fig. 4. Experimental results

References

1. C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In *Proc. ACM Int. Conference on Managment of Data (SIGMOD'01)*, 2001. 16
2. F. Angiulli and C. Pizzuti. Fast outlier detection in high dimensional spaces. In *Tech. Report, n. 25, ISI-CNR*, 2002. 20, 23, 24
3. A. Arning, C. Aggarwal, and P. Raghavan. A linear method for deviation detection in large databases. In *Proc. Int. Conf. on Knowledge Discovery and Data Mining*, pages 164–169, 1996. 16
4. V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley & Sons, 1994. 16
5. M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *Proc. ACM Int. Conf. on Managment of Data (SIGMOD'00)*, 2000. 16
6. C. E. Brodley and M. Friedl. Identifying and eliminating mislabeled training instances. In *Proc. National American Conf. on Artificial Intelligence (AAAI/IAAI 96)*, pages 799–805, 1996. 15
7. Yu D., Sheikholeslami S., and A. Zhang. Findout: Finding outliers in very large datasets. In *Tech. Report, 99-03, Univ. of New York, Buffalo*, pages 1–19, 1999. 16
8. 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. 18
9. J. Han and M. Kamber. *Data Mining, Concepts and Technique*. Morgan Kaufmann, San Francisco, 2001. 15
10. 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. 16
11. 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. 18
12. E. Knorr and R. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proc. Int. conf. on Very Large Databases (VLDB98)*, pages 392–403, 1998. 16
13. M. Lopez and S. Liao. Finding k-closest-pairs efficiently for high dimensional data. In *Proc. 12th Canadian Conf. on Computational Geometry (CCCG)*, pages 197–204, 2000. 19
14. S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Proc. ACM Int. Conf. on Managment of Data (SIGMOD'00)*, pages 427–438, 2000. 16
15. Hans Sagan. *Space Filling Curves*. Springer-Verlag, 1994. 18
16. S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of olap data cubes. In *Proc. Sixth Int. Conf on Extending Database Thecnology (EDBT)*, Valencia, Spain, March 1998. 16
17. J. Shepherd, X. Zhu, and N. Megiddo. A fast indexing method for multidimensional nearest neighbor search. In *Proc. SPIE Conf. on Storage and Retrieval for image and video databases VII*, pages 350–355, 1999. 18, 19
18. Z. R. Struzik and A. Siebes. Outliers detection and localisation with wavelet based multifractal formalism. In *Tech. Report, CWI,Amsterdam, INS-R0008*, 2000. 16
19. K. Yamanishi and J. Takeuchi. Discovering outlier filtering rules from unlabeled data. In *Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 389–394, 2001. 16

20. K. Yamanishi, J. Takeuchi, G.Williams, and P. Milne. On-line unsupervised learning outlier detection using finite mixtures with discounting learning algorithms. In *Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 250–254, 2000. 16