Pattern Recognition 41 (2008) 1338–1349

A genetic approach for efficient outlier detection in projected space

Sanghamitra Bandyopadhyay, Santanu Santra*

Machine Intelligence Unit, Indian Statistical Institute, Kolkata 700108, India

Received 26 April 2006; received in revised form 27 September 2007; accepted 4 October 2007

Abstract

In this paper we present a genetic solution to the outlier detection problem. The essential idea behind this technique is to define outliers by examining those projections of the data, along which the data points have abnormal or inconsistent behavior (defined in terms of their sparsity values). We use a partitioning method to divide the data set into groups such that all the objects in a group can be considered to behave similarly. We then identify those groups that contain outliers. The algorithm assigns an 'outlier-ness' value that gives a relative measure of how strong an outlier group is. An evolutionary search computation technique is employed for determining those projections of the data over which the outliers can be identified. A new data structure, called the grid count tree (GCT), is used for efficient computation of the sparsity factor. GCT helps in quickly determining the number of points within any grid defined over the projected space and hence facilitates faster computation of the sparsity factor. A new crossover is also defined for this purpose. The proposed method is applicable for both numeric and categorical attributes. The search complexity of the GCT traversal algorithm is provided. Results are demonstrated for both artificial and real life data sets including four gene expression data sets.

Keywords: Deviation detection; Gene expression; Genetic algorithm; Grid count tree; Projected dimension; Outlier

1. Introduction

In a data set, one may find a small percentage of data objects, called outliers, which are considerably dissimilar with the rest of the data based on some measurement [1]. Outliers could be some noisy observations or may contain useful, though rarely occurring, information because they contains indication of some abnormal behavior of the system [26,27]. The identification of outliers lead to discovery of typical data objects that do not comply with the general data model. Outlier detection can be described as follows: Given a set of n points or objects, and k, the expected number of outliers, find the top k objects that are considerably dissimilar, exceptional, or inconsistent with respect to the remaining data [1,31]. The outlier detection problem can be viewed as two subproblems:

- Define what data can be considered as inconsistent in a given data set.
- Find an efficient method to mine the outliers so defined.

For general knowledge discovery tasks, it is imperative that the outliers are first identified and ignored, since they may interfere (negatively) in the knowledge extraction process. Thus outlier detection is an inseparably important aspect in knowledge discovery in data (KDD). The application areas of outlier detection include fraud detection, network intrusion detection, weather prediction, detection of criminal activity in e-commerce and in analysis of gene expression data.

The task of outlier detection has been tackled by researchers in a variety of ways. These can be broadly classified into three approaches: *statistical*, *deviation-based* and *distance-based*. The statistical approach requires an a priori knowledge of the probability distribution of the data. A discordancy test is then used to verify whether an object o is significantly larger (or smaller) in relation to the distribution [30]. Deviation-based approach identifies outliers by examining the main characteristics of the objects in a group. Objects that deviate significantly from the main characteristics of the data are considered to be outliers. The distance-based approach was proposed by Knorr and Ng [2] as "*An object O in a data set T is a DB(p, D)—outlier if at least fraction p of the object in T lies greater than distance D from O*". This definition of outlier has the advantage

* Corresponding author. Tel.: +91 94 3376 7822; fax: +91 33 2578 3357.
E-mail addresses: sanghami@isical.ac.in (S. Bandyopadhyay), santanu_t@isical.ac.in (S. Santra).

of being intuitive and simple as well as computationally feasible. Moreover, no previous knowledge about the data set is necessary. More details about them are available in Refs. [1–5].

Many data-mining algorithms in the literature find outliers as a side product of clustering [1,6,7]. For these, outliers are objects that do not belong to any cluster. Thus, the technique implicitly defines outliers as the background noise in which the clusters are embedded. Another class of techniques defines outliers as points, which are neither a part of a cluster nor a part of the background noise; rather they are specific points, which behave very differently from the normal once [32]. The outlier detection problem can also be modelled as a highly imbalanced classification problem, and measures like receiver operating characteristic curves may be used to estimate the performance of a learning algorithm [8,9].

Recently genetic algorithms (GAs), a class of evolutionary search and optimizing tools, have been applied to the problem of outlier detection in multidimensional space [10,11]. The method is based on taking projections of the data in some lower dimensional spaces, dividing these space into grids and computing the sparsity factor of the grid, such that the deviations become prominent. However, this process is found to be extremely computation intensive. The computation time is found to be significantly large even for moderately large data. In order to overcome this limitation, a novel structure has been proposed in this article for storing the data so that the computation time is reduced significantly. In contrast to the method in Ref. [10], which assumes the data to be numeric, here a modified genetic outlier detection method is proposed which can work on mixed data, containing both numerical and categorical features. The proposed method is compared to some other outlier detection methods like K–d tree based method [5] and distance based method [2], in addition to the earlier genetic scheme [10] for several artificial and real life data sets including four gene expression data sets.

The article is organized as follows. Section 2 describes the main issues involved in the proposed grid count tree based algorithm. This includes the definition of the GCT and how it is used for computing the fitness in genetic algorithm. The following section describes the proposed algorithm in detail. The experimental results are provided in Section 4. Section 5 concludes the article.

2. GCT-GOD: genetic algorithm for outlier detection using grid count tree

For high dimensional data, it is often found that a few data points deviate from the normal data in only a few dimensions. However, in these cases, an algorithm that tries to detect the outliers considering all the dimensions will be misled due to the averaging effect of the non-deviating attributes. In such situations, it may be advisable to first project the data in a lower dimensional space, and then try to detect outliers in this space. It may be noted in this context that the problem of determining the appropriate projection is exponential in the number of dimensions. In Ref. [10] a genetic algorithm is used to identify the appropriate low dimensional projection where the most sparse data can be found. In order to define such projection first a grid discretization of the data space is performed. Each attribute of the K-dimensional data is divided into $f = \frac{1}{\phi}$ equi-depth intervals. Thus each interval along each dimension contains a fraction $f = \frac{1}{\phi}$ of the points on an average. For an s-dimensional grid, $s < K$ (K is the number of dimensions), which is created by projecting the data on s different dimensions, the expected fraction of records would be equal to f^s. If the data are uniformly distributed, then the presence or absence of any point in the s-dimensional grid is a Bernoulli random variable with probability f^s [10]. Let N be the total number of points. Then expected number of points and the standard deviation of the points in an s-dimensional grid are given by $N \cdot f^s$ and $\sqrt{N \cdot f^s \cdot (1 - f^s)}$, respectively. Let N' be the number of points in an s-dimensional grid H. Then the sparsity coefficient S_c of grid H is defined as follows [10]:

$$S_c = \frac{(N' - N \cdot f^s)}{\sqrt{N \cdot f^s (1 - f^s)}}. \quad (1)$$

Negative sparsity coefficients of the grids indicate the presence of significantly lower data points than expected.

Although the method of Ref. [10], referred to as GOD (genetic algorithm for outlier detection) in this article, works well for very high dimensional space when the number of data points is low, it becomes computationally infeasible if the size of the data set becomes even moderately large. This is because every time the sparsity coefficient is computed (which is in fact computed a large number of times in GOD), the entire data set needs to be projected in a lower dimensional space and a grid count operator needs to be carried out. In this article a new structure for data representation, called grid count tree (GCT), is proposed for storing some information about the data points such that the genetic technique becomes computationally feasible for large data sets. The genetic scheme using GCT is referred to as GCT-GOD. A further modification is made so as to enable GCT-GOD to operate on categorical as well as numerical data. This is in contrast to Ref. [10], where only numerical data can be considered. A detailed description of the GCT is given below followed by an explanation of the GCT-GOD technique.

2.1. GCT: grid count tree

The data set is first divided into a number of equi-sized intervals along each dimension. In case an attribute is categorical having m possible values $v_1, v_2, \ldots, v_m$, that particular dimension is divided into m distinct values [29]. Note that in contrast to Ref. [10] where each dimension was divided into a fixed number (say, ϕ) of intervals, here this may vary depending on whether an attribute is categorical or not; and if so, then on the number of possible values it can take. The number of intervals along dimension i is defined as ϕ_i. The entire feature space can now be considered to be a set of non-overlapping hypergrids containing all the data points. A special representation, called the GCT, is used to store the definition of each grid (or, hypergrid) as well as the number of data points residing in it. In the first step, each point of the data set is represented as a string of

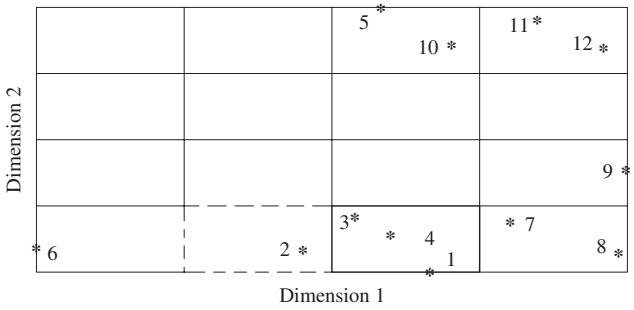

Fig. 1. An example of a data set and the grid structure imposed on it. The grids with bold and dotted boundaries map to nodes 'a' and 'b', respectively, in Fig. 2.

length K, where K is the number of dimensions. This corresponds to a multidimensional grid where the point resides. The way this mapping is done is as follows.

Let us consider a data set $X = \{\vec{x_1}, \vec{x_2}, \ldots, \vec{x_N}\}$ in a K-dimensional space. Let Max_i and Min_i represent the maximum and minimum extent of X along dimension i, respectively. If ϕ_i is the number of intervals along dimension i, then the length of each interval is

$$\Delta_i = \frac{\text{Max}_i - \text{Min}_i}{\phi_i}. \tag{2}$$

A point $\vec{x_j} = (x_{j1}, x_{j2}, \ldots, x_{jK})$ maps to a string $st_j = s_{j1}s_{j2}\ldots s_{jK}$, where

$$s_{ji} = \left\lceil \frac{x_{ji} - \text{Min}_i}{\Delta_i} \right\rceil. \tag{3}$$

Consider Fig. 1. This figure shows a sample data set having 12 points in a two-dimensional space. Here each dimension is divided into four intervals as shown. As can be observed, points labeled 1, 3 and 4 lie in the same grid (3,1) and can be represented by the string 31. The string representation of a hypergrid can contain '$*$'s which indicates all the intervals along the corresponding dimension (i.e., the data are effectively projected along the dimensions that do not have a '$*$'). For example, grid $(4, *)$ in Fig. 1, represented by string $4*$, contains points labeled 7, 8, 9, 11 and 12, i.e., those points which would like in the 4th interval of dimension 1, if all the points are projected along this dimension.

The GCT contains a parent node, $\wp$, corresponding to some dimension i, that consists of ϕ_i pointers, where ϕ_i is the number of intervals along that dimension. An important issue during the construction of the GCT is the choice of the dimension i as the parent node. In this article the dimension with the least number of intervals is selected. A tie is resolved randomly. Each pointer of the parent node $\wp$ points to a binary search tree (BST). One may recall that a BST is a binary tree where each node has a key and two child pointers. The key values of the left (or, right) child of any parent node in a BST is smaller (or greater) than the key value of the parent. In GCT, each node of the ϕ_i BSTs (corresponding to the ϕ_i pointers from $\wp$) contains the following:

- *String*—A string that defines a multidimensional hypergrid in the feature space. Note that depending on the context, it can be treated as a string or an integral value.
- *Count*—A counter to count the number of points that reside in that grid.
- *Left*—A left pointer.
- *Right*—A right pointer.

Consider Fig. 2 which is the GCT corresponding to the points shown in Fig. 1. $\wp$ is the root node whose entries correspond to the four intervals along dimension 2. Point '1' is the first point to be considered, and its grid location is (3,1) which corresponds to the string '31'. Therefore, a node (represented in Fig. 1 as 'a') containing '31' as the grid location, and 1 in the count field is created as the child of $\wp$ [12]. Thereafter point '2' is encountered whose grid location is (2,1). Since this corresponds to the string '21', and 21 is smaller than 31, it appears as the left child of 'a' (see node 'b' in Fig. 1). Subsequently, point '3' is encountered, and its grid location is again (3,1). Following the pointer from $\wp$ [12] (since the second dimension of the grid (3,1) is 1), a match with the grid location entry of node 'a' is found. Hence its count value is incremented by one. This is again incremented by one after encountering point '4'.The node 'c' which appears as the right child of 'a' corresponds to the points '7' and '8' having grid location (4,1). In this way the full GCT is constructed for all the eight points. Algorithm 1 shows how a string (st) is inserted into the GCT.

Algorithm 1. GCTbuild(st, GCT).

```
1:  Input: st, GCT /* GCT is a pointer pointing to the
    subtree ℘[st[v]], where v corresponds to the dimen-
    sion forming the root of the GCT, and st[v] is the vth
    position of the string st to be inserted. */
2:  Output: GCT
    /* For empty tree */
3:  if GCT is NULL then
4:    new = Allocate space for a GCT node
5:    new.String = st
6:    new.Count = 1
7:    new.Left = new.Right = NULL
8:    ℘[st[v]] = new
9:  end if
    /* The strings are treated as integral values here. */
10: if st > (GCT → String) then
11:   if (GCT → Right) is NULL then
12:     new = Allocate space for a GCT node
13:     new.String = st
14:     new.Count = 1
15:     new.Left = new.Right = NULL
16:     GCT → Right = new
17:   else
18:     Call GCTbuild(st, GCT → Right)
19:   end if
20: end if
```

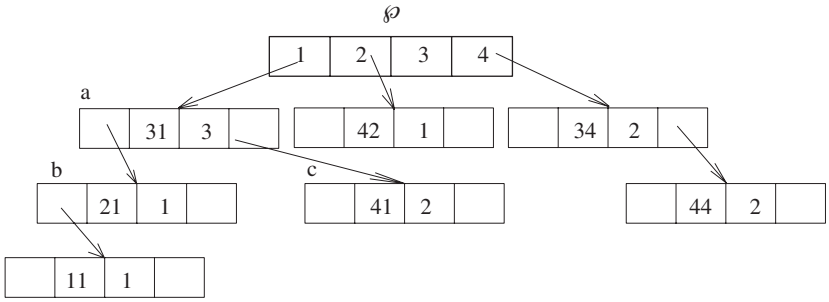

Fig. 2. GCT for the data in Fig. 1.

```
21:  if st < (GCT → String) then
22:    if (GCT → Left) is NULL then
23:      new = Allocate space for a GCT node
24:      new.String = st
25:      new.Count = 1
26:      new.Left = new.Right = NULL
27:      GCT → Left = new
28:    else
29:      Call GCTbuild(st, GCT → Left)
30:    end if
31:  end if
32:  if st = =(GCT → String) then
33:    GCT → Count = GCT → Count + 1
34:  end if
```

2.2. Lower dimensional projection and chromosome representation

As in Ref. [10], the data are first projected in a lower dimensional space. A chromosome is defined to encode a hypergrid in the projected space. The length of a chromosome is equal to the number of dimensions of the data set. For a K-dimensional data set the structure of the chromosome would be a string $[c_1c_2c_3\cdots c_K]$ where

$$c_i = j, \quad j \in \{`*', 1, 2, \ldots, \phi_i\}, \quad i = 1, 2, \ldots, K. \tag{4}$$

Here, a value of '$*$' in some c_i represents a don't care, indicating that the chromosome represents the whole range along the dimension i. Otherwise, some value j in c_i indicates that the chromosome represents the jth interval along dimension i. For a K-dimensional data set where the data are projected on an s-dimensional space, the chromosome should have $(K-s)$ '$*$'s and s numeric values. Consider the following chromosome $*3**2**5$. It represents a three-dimensional projection of an eight-dimensional data set. This chromosome corresponds to the grid composed of the 3rd, 2nd, 5th intervals along 2nd, 5th, and 8th dimension, while all other dimensional values are don't care's. In the example described in Fig. 1 one such chromosome may be $1*$ which indicates that the data are projected only along the first dimension, and the first interval is to be considered. From the figure, it is evident that this corresponds to the data point '6'.

2.3. Fitness computation

The fitness value of a chromosome is equal to the sparsity coefficient of the hypergrid represented by that chromosome. The sparsity coefficient provides a measure for detecting outliers based on the degree to which the characteristics of the data points in a hypergrid deviate from the normal data characteristics. If ϕ_i is the number of intervals along dimension i, then for a K-dimensional hypergrid with s projected dimensions (say, $pd_1, pd_2, \ldots, pd_s$), the expected number of points in it is equal to

$$\prod_{i=1}^{s} f_{pd_i},$$

where pd_i is the ith projected dimension and

$$f_{pd_i} = \frac{1}{\phi_{pd_i}}. \tag{5}$$

Therefore S_c (Eq. (1)) now becomes

$$S_c = \frac{(N' - N.\prod_{i=1}^{s} f_{pd_i})}{\sqrt{N.\prod_{i=1}^{s} f_{pd_i}(1 - \prod_{i=1}^{s} f_{pd_i})}}. \tag{6}$$

The fitness of a chromosome is defined as S_c. Note that the sparsity coefficient, S_c, which is a measure of the difference between expected uniform and actual sparsity of a hypergrid, will be very low for the hypergrids containing outliers. From Eq. (6) it is evident that the computation time depends primarily on the determination of N' since computation of $\prod_{i=1}^{s} f_{pd_i}$ is fixed. The computational complexity for fitness computation is hence mainly dependent on the complexity of the computation of N'.

2.3.1. N' computation from GCT

GCT is designed to optimize the computation of the sparsity coefficient S_c as given in Eq. (6). In general, to calculate S_c of a hypergrid all the data elements need to be compared with the string value of that hypergrid. This was the approach used in Ref. [10]. In the present article, a method is proposed where the GCT is used to perform this computation more efficiently.

Given a chromosome $\mathscr{C} = c_1c_2\ldots c_K$, ($c_j \in \{1, 2, \ldots, \phi_j, `*'\}$) that encodes a hypergrid in the projected space, the GCT

is traversed in an efficient way to find out the number of data points that lie in this hypergrid. The GCT traversal is referred to as the Match-Decision traversal (MDT).

MDT starts at the root with the initial value of $N' = 0$. If the root is the projection along dimension i, then the ith position of the chromosome is matched to the corresponding position of the root node. In case the ith position of the chromosome is a '$*$' (don't care), then this indicates that all the positions of the root node are matching, and all of them must be considered in the traversal. Following the link of the matched position from the root (which could be all the positions of the root in the worst case), the respective BSTs are now traversed. The nodes of the BSTs are matched to the string encoded in the chromosome. If the values in the defined positions in a chromosome (i.e., excluding the don't cares) match those in the corresponding positions of the *String* field, a match is declared. For example a chromosome $42*5$ would match a string 4275.

If a match is found then one of the following three cases will arise:

- *Match-Leaf* (*Decision*): The matching BST node is a leaf. In this case N' is incremented by the value in *Count* field of the leaf node, and the traversal in that BST terminates.
- *Match-NonLeaf-EndIntervals* (*Decision*): The matching BST node is not a leaf and the first '$*$' position of the chromosome (say j) has a value st_j in the corresponding *String* field of the BST node, where st_j is either 1 or ϕ_j. Here, N' is first incremented by the value in *Count* field of the match node. Thereafter, in case the value is 1 (or, ϕ_j), then the *Right* (or, *Left*) pointer has to be traversed fully from the matching node. Thus a decision regarding which link to follow in the next step of MDT can be taken.
- Example: For example let the chromosome be $[42*5]$ and the *String* field of the matching BST node be [4275] with $\phi_3 = 7$. As is evident, a match is found. Since $\phi_3 = 7$, the third dimensional value in the chromosome can be any number between 1 and 7. Since the *String* field contains 7 in the third position, all other matches with $42*5$ can only be smaller than 4275. Hence only the left subtree needs to be fully traversed. The converse is true if the *String* field of the matching BST node is [4215].
- *Match-NonLeaf-MidInterval* (*NoDecision*): The matching BST node is not a leaf and the first '$*$' position of the chromosome is j and the corresponding *String* field of the BST node has a value st_j where $1 < st_j < \phi_j$. In this case, N' is first incremented by the value in *Count* field of the match node. Thereafter, both the children of the matched BST node have to be fully traversed.
- Example: For the above example, let the *String* field of the matching BST node be [4225], while the chromosome be [42$*$5]. Then the other nodes that may match the chromosome reside in both the left and right subtrees of the BST node.

If a match is not found as per the above definition, then again one of the following three cases will arise:

- *MisMatch-Defined* (*Decision*): Let j be the position where the first mismatch occurs. Then, this situation arises when the jth position of the chromosome $\mathscr{C}$ contains a non-'$*$' value. (Note that *String* field will never contain '$*$'s.) In that case, if the jth value of $\mathscr{C}$ is less than that of *String*, then the *Left* pointer is to be followed. Otherwise, the *Right* pointer should be followed.
- Example: Let $\mathscr{C}$ be [42$*$5] and the *String* field of the matching BST node be [4374]. As is evident, this is a mismatch; the mismatch occurs at position 2. Moreover, since 42 is less than 43, so the *Left* pointer is to be traversed.
- *MisMatch-Star-EndInterval* (*Decision*): Let j be the position where the first '$*$' appears in $\mathscr{C}$. Here, let st_j be the value in the jth position of *String*, and st_j is either 1 or ϕ_j. If $st_j = 1$ then the *Right* pointer of the BST node should be followed. If $st_j = \phi_j$, then the *Left* pointer should be followed.
- Example: Let $\mathscr{C}$ be [42$*$5] and the *String* field of the matching BST node be [4274] where $\phi_3 = 7$. Evidently, all matches with $42*5$ can only be found in the left subtree of the BST node.
- *MisMatch-Star-MidInterval* (*NoDecision*): Let j be the position where the first '$*$' appears in $\mathscr{C}$. Here, let st_j be the value in the jth position of *String*, and st_j is neither 1 nor ϕ_j. In this case no decision regarding which pointer to follow can be taken; both the pointers have to be followed.
- Example: Let $\mathscr{C}$ be [42$*$5] and the *String* field of the matching BST node be [4234] where $\phi_3 = 7$. Evidently, no decision can be made regarding which link to follow. Consequently, both the links have to be checked.

2.3.2. Search complexity in GCT

During MDT, if the dimension of the root node of GCT is one of the projected dimensions as encoded in $\mathscr{C}$, then searching is limited to a single underlying BST. Otherwise search is extended to all ϕ_i BSTs, where i is the dimension corresponding to the parent node.

Considering a data set generated using a uniform distribution, each interval of the parent node corresponds to $\frac{N}{\phi_i}$ points on an average. If H_{max} is the maximum number of hypergrids that may be considered along each interval of the parent node (the corresponding BST is constructed from these H_{max} hypergrids), then

$$H_{max} = \min\left\{\frac{N}{\phi_i}, \prod_{j=1, j\neq i}^{K} \phi_j\right\}. \tag{7}$$

If only one BST is traversed for MDT, then the average search complexity is equal to that for searching in the BST with H_{max} nodes; this is equal to

$$O(\log(H_{max})) = O\left(\log\left(\min\left\{\frac{N}{\phi_i}, \prod_{j=1, j\neq i}^{K} \phi_j\right\}\right)\right). \tag{8}$$

In case the search is extended to all the ϕ_i BSTs, the average search complexity becomes

$$O(\phi_i \log(H_{max})) = O\left(\phi_i \log\left(\min\left\{\frac{N}{\phi_i}, \prod_{j=1, j\neq i}^{K} \phi_j\right\}\right)\right). \tag{9}$$

3. The main algorithm

The main process is now described in Algorithm 2. It consist of four phases, namely, initialization, selection, crossover and mutation. These phases are described below in detail.

Algorithm 2. GCT-GOD(K, s, *PopSize*, *NumberOfSol*).

Require: $(0 < s \leqslant K)$
1: Select v (the GCT root dimension), the intervals along each dimension and initialize $\wp$
2: **for** each data point x_j **do**
3: $st =$ Convert x_j to a string in the feature space
4: GCTbuild(st,$\wp[st[v]]$)
5: **end for**
6: $PreserveSet \Leftarrow \emptyset$/* *PreserveSet* Resultant chromosome set */
7: *Initialize*(P)
8: *Compute Fitness* (P)
9: **while** Terminating condition = False **do**
10: $P \Leftarrow Selection(P)$
11: $P \Leftarrow Crossover(P)$
12: $P \Leftarrow Mutation(P)$
13: *Compute Fitness* (P)
14: $PreserveSet \Leftarrow Preserve(PreserveSet, P)$ /*Best $\mathscr{C}$s with the smallest values of S_c among *PreserveSet* and P are preserved */
15: **end while**
16: $\vartheta \Leftarrow Outlier(PreserveSet, Number\,Of\,Sol)$ /* Return outlier */
17: **return** (ϑ)

3.1. Initialization

Random initialization is used in order to create chromosomes for the initial population. In order to create only feasible chromosomes, s different dimensions are selected randomly. Then for each selected dimension i the corresponding position is filled with a random value between 1 and ϕ_i. All the remaining dimensions of the chromosome are filled with $*$. The two most important tasks during initialization are the selection of ϕ_i, the number of intervals along dimension i and s, the number of projected dimensions. When the dimension is numeric, ϕ_i is kept high enough so that points that are far apart along that dimension are not placed in the same interval [12]. For non-numeric attributes, ϕ_i equals the number of possible values on that dimension. The value of s is so chosen that there exists at least one point in a hypergrid. In this case, from Eq. (5), the expected fraction should be greater than or equal to $\frac{1}{N}$. For the determination of such a value of s, says_1, a decreasing order set (O_K) of K ϕs are created. In other words

$$O_K = \{\phi_{a_1}, \phi_{a_2}, \ldots \phi_{a_K}\}, \quad \text{s.t. } \phi_{a_i} \geqslant \phi_{a_j} \text{ for } i < j \text{ and } i, j = \{1, 2, \ldots, K\},$$
$$\text{then } s_1 = q \text{ where } q \in \{1, 2, \ldots, K\} \text{ s.t. } \prod_{i=1}^{q} \phi_{a_i} \leqslant N \leqslant \prod_{i=1}^{q+1} \phi_{a_i}. \tag{10}$$

It is to be noted that for any value of $s \leqslant s_1$, the condition that there exits at least one point in a hypergrid is satisfied.

The other consideration while choosing s is to make the number of possible s-dimensional projections ($^K C_s$) large. Note that $^K C_s$ is maximum when $s = \frac{K}{2}$. The value of s is then chosen as $s = \min\{s_1, \frac{K}{2}\}$ in order to ensure that a hypergrid in the projected space contains at least one point.

Example: For a four-dimensional data set with 10,000 data points let the ϕ_i values be 10, 15, 10 and 20 along the four dimensions, respectively. Then the value of s_1 is 3 since $20 \times 15 \times 10 \leqslant 10,000$. So for this data set, s should be $\min\{3, \frac{4}{2}\}$, which is 2.

Algorithm 3. Chromosome Initialization (P).

1: $P = \emptyset$
2: **for** ($i = 1$ to *PopSize*) **do**
3: Generate chromosome $\mathscr{C}_i$ of length K with all '$*$'s.
4: Randomly select s positions $a_1, a_2, \ldots a_s$ /* $s =$ number of projected dimensions */
5: **for** ($j = a_1$ to a_s) **do**
6: $\mathscr{C}_i[j] \Leftarrow$ Random(1, ϕ_j)
7: **end for**
8: **if** $\mathscr{C}_i \notin P$ **then**
9: $P \Leftarrow \{\mathscr{C}_i\} \cup P$
10: **else**
11: Return to step 3
12: **end if**
13: **end for**

3.2. Compute fitness and selection

The fitness of the chromosome is computed in terms of S_c as defined in Eq. (6). Several alternatives are possible for implementing selection in a GA. The present algorithm uses rank selection method [13]. This selection procedure has been often found to be more stable than the straight forward fitness proportional methods which samples the set of solutions in proportion to their actual fitness value [10]. The advantage of this method is that it can prevent very fit individuals from gaining dominance early at the expense of less fit ones.

3.3. Existing and proposed crossover operators

This is a key method in evolutionary algorithms where information is exchanged between two solutions in order to yield potentially better solutions. In crossover, performed with a probability p_c, two chromosomes ($\mathscr{C}_1, \mathscr{C}_2$) exchange parts of their characteristics (genes) and generate two new chromosomes ($\mathscr{C}'_1, \mathscr{C}'_2$) that share characteristics of both the parent chromosomes. Two crossover operations were used in Ref. [10] namely the conventional single point crossover and a crossover that employed a local search to test for all possible feasible combinations of the two parents, and selected the one having the lowest sparsity value. The latter operator is extremely time consuming, though it is shown to provide better results than the first scheme [10], and is referred to as 'optimized crossover' in this article.

A third crossover technique is proposed in this article that is not as time consuming as the above mentioned optimized crossover [24] and ensures that only feasible strings result from the process as the optimized crossover also does. The new crossover technique is shown in Algorithm 4. This is explained here through an example. Consider the following two chromosomes for a six-dimensional data with three-dimensional projection.

Algorithm 4. Crossover(P).

Require $(P \neq \emptyset) \wedge (|P| \text{ is even})$
$P' \Leftarrow \emptyset$
for $(j = 1 \text{ to } |P|/2)$ **do**
  Select two chromosomes $(\mathscr{C}_j, \mathscr{C}'_j)$ from P
  $X \Leftarrow$ set of positions in which neither $\mathscr{C}_j$ nor $\mathscr{C}_{j'}$ is '$*$'
  $Y \Leftarrow \{(i, j)|$ for parents $\mathscr{C}_j$ and $\mathscr{C}'_j \wedge ((\mathscr{C}_j[i] = \text{`}*\text{'}) \wedge (\mathscr{C}'_j[i] \neq \text{`}*\text{'}) \vee (\mathscr{C}_j[i] \neq \text{`}*\text{'}) \wedge (\mathscr{C}'_j[i] = \text{`}*\text{'}))$
  Initialize string $\mathscr{C}$ with K '$*$'s
  for all $(i, i \in X)$ **do**
    $\mathscr{C}[i] \Leftarrow \text{random}(\mathscr{C}_1[i], \mathscr{C}_2[i])$.
  end for
  Extend $\mathscr{C}$ by selecting $\frac{|Y|}{2}$ positions randomly from set Y and assign the value from the appropriate chromosome.
  Generate a complementary string $\mathscr{C}'$ in which values are derived from a different parent than $\mathscr{C}$ is derived.
  Add solutions $\mathscr{C}$ and $\mathscr{C}'$ to set P'.
end for
Replace P with P'
return (P)

$Position$	1	2	3	4	5	6
$\mathscr{C}_1 \rightarrow$	3	$*$	$*$	5	$*$	2
$\mathscr{C}_2 \rightarrow$	2	5	$*$	3	$*$	$*$

There are two positions (1 and 4) with no '$*$'s. These are kept in set X. There are two positions (2 and 6) with one '$*$' only; these are kept in Y. Thus $X = \{1, 4\}$ and $Y = \{2, 6\}$. Note that $|Y| = 2(s - x')$, when $x' = |X|$. Initially $\mathscr{C}$, a child chromosome, is assigned values in the positions of set X randomly from one of the two parent chromosome. So initially $\mathscr{C}$ may be $2 * *5 * *$; where the first position is taken from second chromosome and fourth position is taken from the first chromosome. Then $\frac{|Y|}{2} = s - x'$ positions are randomly selected from set Y and the values in these positions from the appropriate chromosome are put in $\mathscr{C}$. Or, $\mathscr{C}$ may be either$2 * *5 * 2$ or $25 * 5 * *$ depending on whether the second or the first position is selected from Y, respectively.

3.4. Mutation

In GAs, mutation is used to increase the diversity of the population, and to recover lost information [13]. Here mutation is similar to the one used in Ref. [10], expect that whenever a '$*$' has to be set to defined value, it is set to a randomly chosen value in $[1, \phi_i]$.

Consider the chromosome $\mathscr{C} = 3 * *5 * 2$ with $\phi_2 = 10$. Let the second position be selected for mutation that contains a '$*$'. It is then changed to a random value in Refs. [12,15], say 4. It is then necessary to randomly select a position not containing a '$*$'. Let position 4 be selected. It is changed to '$*$'. If no other position is changed, then after mutation $\mathscr{C}$ becomes $34 * * * 2$.

3.5. Complexity of GCT-GOD

In GCT-GOD, the GCT is created only once from the data set. The complexity of converting each point of the data set to a string is $O(N * K)$, where N and K are the numbers of points and dimensions, respectively. The average complexity for constructing the tree is $O(N \log(N))$. The procedure *Initialize* is also carried out only once with complexity $O(|P| \times K \times s)$, where $|P|$ is the size of the population and s is the number of dimensions on which the data are projected. As mentioned earlier, the average GCT search complexity is $O(\log(\min\{\frac{N}{\phi_i}, \prod_{j=1, j\neq i}^{K} \phi_j\}))$. Hence for $|P|$ chromosomes, the complexity for fitness computation is $O(|P| \times \log(\min\{\frac{N}{\phi_i}, \prod_{j=1, j\neq i}^{K} \phi_j\}))$. The selection complexity, that includes the complexity of ranking of the chromosome and selection of $|P|$ chromosomes is $O(|P| \log(|P|))$. Crossover operator has the complexity of $O(|P| \times K)$. GCT-GOD performs bit wise mutation so the complexity of this operator is $O(|P| \times K)$. In *Preserve* procedure, the chromosomes are sorted in the current population (with complexity $O(|P| \log |P|)$) and the best chromosomes are selected by merging the *PreserveSet* chromosomes and current population chromosomes. The complexity for this procedure is therefore $O(|P| \log(|P|) + |PreserveSet|) \approx O(|P| \log(|P|))$ as $|PreserveSet| < |P|$. The outliers are determined by projecting the chromosomes in the *PreserveSet* of last generation. It is carried out only once. The complexity of this procedure is $O(|PreserveSet| \times \log(\min\{\frac{N}{\phi_i}, \prod_{j=1, j\neq i}^{K} \phi_j\}))$. So the overall complexity of GCT-GOD is

$$O\left(N \log N + gen \times \left\{|P| \times \log\left(\min\left\{\frac{N}{\phi_i}, \prod_{j=1, j\neq i}^{K} \phi_j\right\}\right)\right\}\right), \tag{11}$$

where *gen* is the number of iterations of the genetic procedure.

4. Experimental study

This section presents an experimental evaluation of GCT-GOD along with its comparison to other outlier detection method. The performance is tested on both real and artificial data sets. All the tests are performed on a Sun Solaris machine. For the experiments, we choose the crossover probability as 0.9 while mutation probability is fixed to 0.005. Experiments with the other values of the parameter were also conducted with similar results The result presented here is the average result of 10 runs. The population number and the number of

Table 1
Comparative sensitivity and precision score for *Dset*1, *Dset*2 and *Dset*3

Method	Data set	Sensitivity	Precision
GCT-GOD	*Dset*1	0.73	0.49
	*Dset*2	0.76	0.54
	*Dset*3	0.68	0.45
GOD [10]	*Dset*1	0.64	0.46
	*Dset*2	0.21	0.14
	*Dset*3	0.61	0.41
K–d tree [15]	*Dset*1	0.41	0.35
	*Dset*2	–	–
	*Dset*3	0.34	0.30
DB-outlier [16]	*Dset*1	0.22	0.15
	*Dset*2	–	–
	*Dset*3	0.16	0.09

generations are determined according to the size of the data set [25].

4.1. Data sets

For the experiments, six data sets have been used. These consist of three artificial and three real life data sets. Two 10-dimensional artificial data sets (*Dset*1 and *Dset*2) and one 50-dimensional artificial data set (*Dset*3) have been considered for the experiments. Both *Dset*1 and *Dset*2 contain 10,100 instances of which 10,000 instances belong to normal data and 100 instances are outliers. The outliers are manually injected within the data sets, and they deviate from the normal data in randomly selected two or three dimensions. *Dset*3 contains 50,100 instances with 100 outliers. As before, these 100 points deviate from the normal data in randomly selected five or six dimensions. *Dset*1 and *Dset*3 contain all numeric attributes whereas *Dset*2 contains six non-numeric and four numeric attributes. The data sets are available on request to the authors.

The three real-life data sets, Arrythmia, Adult and Mechanical, are obtained from the UCI machine learning repository (http://www.ics.uci.edu/∼mlearn/MLRepository.html). Arrythmia data set consists of 452 instances with 279 attributes of which 206 are numeric. Mechanical data set contains 9254 data points and 8 attributes with only one non-numeric attribute. Adult data set contains 8 non-numeric attributes out of total 14 attributes with 32,561 data points.

4.2. Results on artificial data sets

Table 1 shows the results of GCT-GOD and GOD [10] when they are executed for the same amount of time (800 s of *Dset*1 and *Dset*2, and 2000 s. for *Dset*3) for the artificial data sets *Dset*1, *Dset*2 and *Dset*3. For the purpose of comparison with other outlier detection methods, the *K-d* tree based approach [15] and distance-based (DB) outlier detection [16] are also considered. Their results are also included in the table. Note that the execution times of these algorithms are not restricted; rather they execute to termination. Three factors, namely sensitivity, precision and specificity [9] are used to measure the quality of the results. These are defined as

$$\text{sensitivity} = \frac{TP}{TP+FN}, \quad \text{precision} = \frac{TP}{TP+FP} \quad \text{and}$$
$$\text{specificity} = 1 - fp \text{ rate},$$

where *TP*, *FN* and *FP* indicate the true positive, false negative and false positive calculated from the class confusion matrix [9,17,18]. *TP* is the number of data points which are predicted as outliers and they are indeed outliers. *FP* is the number of data points which are incorrectly predicted as outliers when they are in fact not outliers, *fp* rate is the false positive rate which is the ratio of *FP* and total number of negative.

As can be seen from Table 1 both the *K–d* tree based and distance based method, DB-outlier, yield poor scores, with the latter performing the poorest. The DB-outlier method is computationally expensive and is also found to be unable to provide reasonable result. Both DB-outlier and the *K–d* tree based method attempt to detect the outliers in all the dimensions and therefore end up with low sensitivity and precision values since the outlier characteristics of the manually injected points are evident in only few dimensions.

GCT-GOD and GOD, as expected, perform significantly better than the other two methods. From Table 1, it can seen that the performance of GCT-GOD, both in terms of the sensitivity and precision values, are better than GOD for all the data sets. The reason for this is that the number of iterations executed by GCT-GOD is more than that of GOD though both are executed for the same time (800 s of *Dset*1 and *Dset*2, and 2000 s for *Dset*3); this is because of the fact that the GCT employed during fitness evaluation (i.e., in the computation of N') in GCT-GOD makes it computationally more efficient than GOD [10].

The results demonstrate that for *Dset*3 with 50,100 points and 50 dimensions, GCT-GOD performs the best yielding reasonable sensitivity and precision values. The performance of GOD is poorer, while that of DB outlier detection method is the worst. This is expected since for this 50-dimensional data where the outlier characteristics are evident in only a few dimensions, the average effect of the non-deviating attribute confuses the DB-outlier detection method considerably, resulting in its poor performance. Since both GOD and GCT-GOD are looking at lower dimensional projection of the 50-dimensional data, they fare significantly better.

Fig. 3 shows the variation of the average percentage of outliers detected using GCT-GOD and GOD with time. As can be found from the graph, this percentage is greater for GCT-GOD than for the method in Ref. [10] at all time instances indicating its efficiency and robustness.

It may be noted that the outlier detection problem can also be modelled as a highly imbalanced classification problem, and measures like receiver operating characteristic (ROC) curves may be used to estimate the performance of a learning algorithm [8]. An ROC curve is used to visualize the tradeoff between hit ratio and false alarm rate. Fig. 4 shows the ROC curves of the proposed method for *Dset*1, *Dest*2 and *Dset*3. The X-axis represents the specificity value where as the corresponding sensitivity value is plotted along the *Y* axis for different generations.

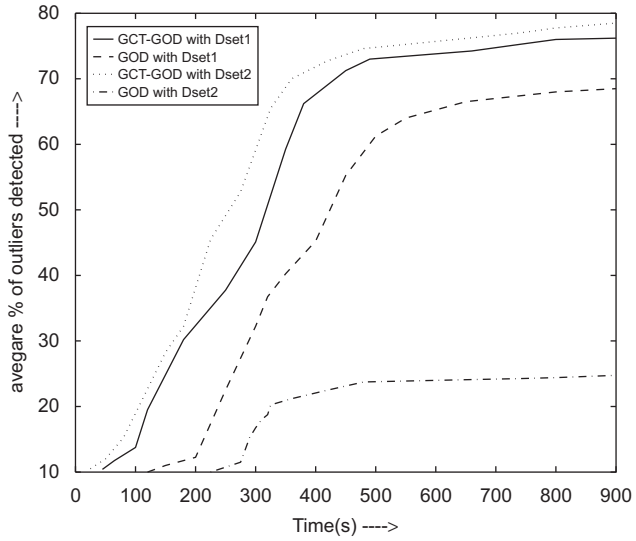

Fig. 3. Variation of the average percentage of outliers detected for *Dset*1 and *Dset*2 using GCT-GOD and GOD.

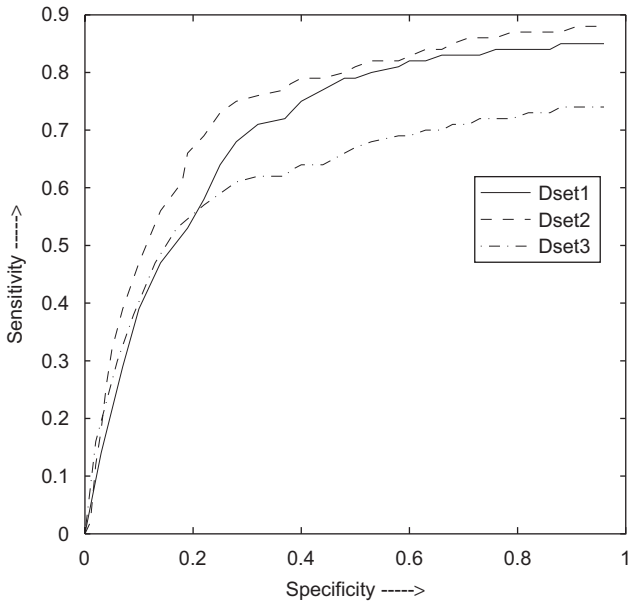

Fig. 4. ROC for three artificial datasets.

The area under the curve (AUC) calculated using the method in Ref. [9] is found to be *fair* (.71) for *Dset*3 where as this is *good* for *Dset*2 and *Dset*3 (.80 and .83, respectively).

4.3. Results on real-life data sets

The results on the real life data sets using both the proposed and optimized crossover are shown in Table 2 for GOD and GCT-GOD. The algorithms are executed for at most 1000 iterations, unless most of the individuals of a population become genetically similar [19,23]. As can be seen from Table 2, except for Arrhythmia data set, GCT-GOD produces better quality result much faster. For Arrhythmia, GCT-GOD produces better

Table 2
Comparative result for the real life data sets

Data set	Crossover	Time required		Average sparsity value	
		GOD	GCT-GOD	GOD	GCT-GOD
Mechanical	Optimized	845	231	−3.529	−3.541
	Proposed	263	52	−3.52	−3.536
Arrhythmia	Optimized	704	720	−1.21	−1.41
	Proposed	270	277	−1.1	−1.2
Adult	Optimized	2076	1667	−0.341	−8.890
	Proposed	–	542	–	−8.862

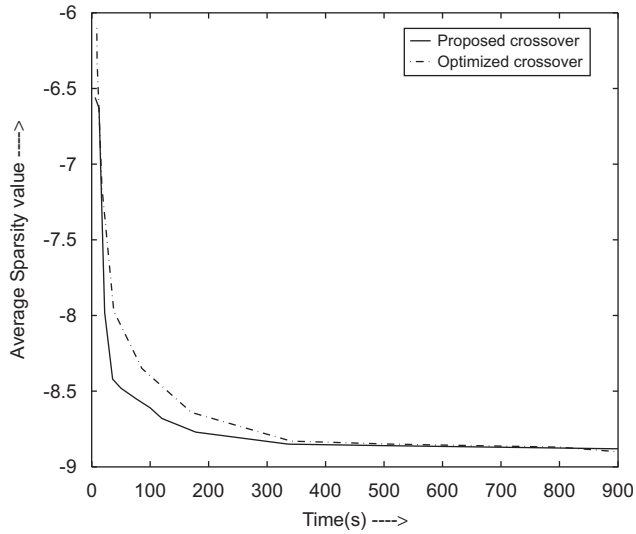

Fig. 5. Plot of the average sparsity value against time for two different crossover techniques in GCT-GOD on adult data set.

sparsity values by properly categorizing the non-numeric attributes, but more computation time is needed as the number of data points is too low and the number of attributes is relatively too high. In this case most of the time the string value corresponding to the parent node of GCT is '0'. So the search is extended to the entire grid count tree in general, thereby necessitating a higher computation time. As is evident from Table 2, the results for Adult and Mechanical data sets, which have a mixture of numeric and non-numeric attributes, GCT-GOD shows superior performance with respect to both time and the average sparsity values. In the Mechanical data set, though the sparsity values are almost equal because only one attribute is non-numeric, the time gain of GCT-GOD over GOD is quite significant for both optimized and proposed crossovers. For Adult data set, GOD is unable to provide reasonable result either with optimized or proposed crossover. This is because of the presence of non-numeric attributes and relatively large size of the data.

Fig. 5 shows the variation of average sparsity value provided by GCT-GOD with time for Adult data set using the proposed and optimized crossovers. As can be seen from the figure, in the initial stages the proposed crossover outperforms the optimized

Table 3
Comparative result for the real life data sets

Data set	Average time required (s)			
	GOD	GCT-GOD	Average sparsity	No. of outliers
CDT15	32	27	−2.18	20
Yeast	1560	291	−1.23	30
CD27+	3026	1083	−0.1416	50
CD27−	–	2131	−0.16541	50

one at any given time. This is because the optimized crossover technique uses a very expensive local search resulting in more time per iteration. For a fixed time interval, GCT-GOD executes more generations and produces better quality result. Only after a long time (i.e., toward the end of the process), the optimized crossover marginally outperforms the proposed one. This is also evident from Table 2 where it can be seen that the proposed crossover requires less time than the optimized one for both GOD and GCT-GOD while providing equivalent or marginally inferior result.

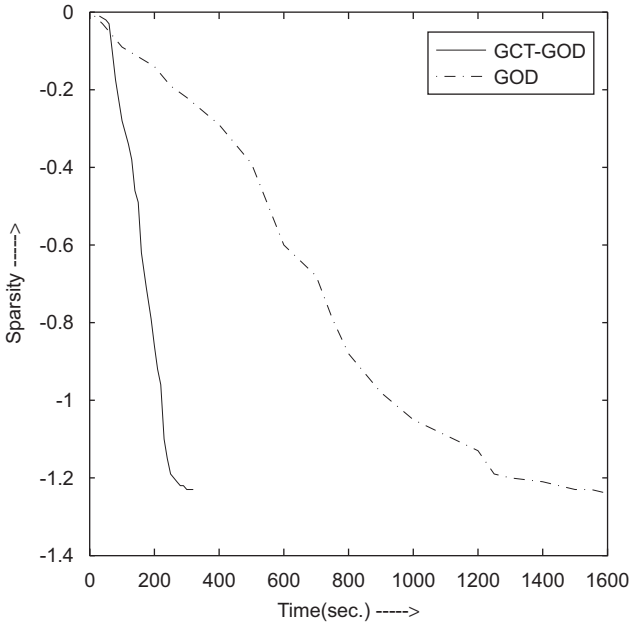

Fig. 6. Variation of the average sparsity values with time for two different crossover techniques in GCT-GOD on CDT15 data set.

4.4. Results on gene expression data

Microarray data sets contain the expression values of thousands of genes measured under different experimental condition/samples. Due to technological variation of sources not all observations for genes are correct. These observation, which are treated as deviations or outliers, should either be discarded or mapped to a value within in the normal range [20]. Alternatively, outlier genes may bear important information, e.g., implicating the genes in certain diseases, that require further analysis [21]. Therefore, outlier detection in gene expression data is an important problem. In this section, we compare the performance of GCT-GOD with GOD for the problem of outlier detection in gene expression data with respect to time requirements. A brief discussion is also included on the outliers detected by an earlier method [11] vis-á-vis the present one.

Four real life gene expression data sets have been used for comparing the performance of GOD with that of GCT-GOD using the proposed crossover. Among them, two lymphoma data (Blood B cell CD27) are obtained from the website http://llmpp.nih.gov/lymphoma. Blood B cell CD27+ and CD27− contain 9209 and 18,432 data points, respectively with 25 attributes. A gene expression data of *Yeast* (yeast.matrix) is obtained from the website http://arep.med.harvard.edu. *Yeast* expression data contains 2882 data points with 17 columns (originally it contained 2884 data points; among them row 57 and 1265 are discarded as they contain a large number of missing values). Another gene expression data cell cycle CDT15 is obtained from Stanford microarray data set. CDT15 contains 768 data points with 24 attributes.

The comparative results of GCT-GOD and the earlier GOD [10] are shown in Table 3 for the four data sets. As can be seen from the table, for the gene data CDT15, only a small time gain is obtained by GCT-GOD. This is because the size of the data set is rather small. For *Yeast* data, GCT-GOD produces much higher time gain. This is expected as the size of these data are quite large. The results for the two Blood B cell data again highlight the advantage of incorporating GCT in GOD for outlier detection when the data size is quite large. In particular for CD27− (which is much larger with 18,832 data points) GOD cannot provide any meaningful result. In contrast GCT-GOD is able to detect the outliers for this data set reasonably fast. This demonstrates the effectiveness of using the proposed GCT while computing the fitness especially when the size of the data set is large. It may be mentioned in this context that the sparsity values of GCT-GOD and GOD is the same since their computing process is the same, the difference being in terms of the time gained when using the proposed GCT. Fig. 6 shows variation of the average sparsity value with time as obtained by GOD and GCT-GOD for the *Yeast* data. As is evident, GCT-GOD attains a given sparsity much earlier as compared to GOD, indicating the effectiveness of incorporating the proposed GCT.

The problem of outlier detection in gene expression was tackled in Ref. [11] for the *Yeast* data, where outliers were detected in dimensions 1, 2, 3 and 14. For this data using GCT-GOD, outliers were also detected in dimensions 1, 2, 3 and 14. Additionally we have also found outliers in dimension 8, 10, 12, 13 and 17. In the case of CD27+ data we not only found outliers in dimension 1 and 3, as was obtained in Ref. [11], but also obtained outliers in dimensions 11, 17 and 22. For CD27−, the outliers were obtained when projected along dimensions 4, 13, 17, 22, 23 and 24, while for CDT15, the outliers were obtained in dimensions 2,4,7,8,15 and 22.

5. Discussion and conclusions

The search capability of genetic algorithm is utilized in this article to detect outliers in lower dimensional subspaces of a

given data set. The proposed algorithm ranks the outliers and declares the top n among them. The evolution mechanism, with a new crossover operator, is adopted to avoid combinatorial explosion that haunts a brute force algorithm. A grid count tree (GCT) data structure is proposed for efficient computation of the fitness value. With this data structure, the algorithm can efficiently handle large amounts of data with modest number of dimensions. Complexity analysis is also provided in this regard.

Results are demonstrated for artificial and real-life data sets which vary widely over the number of data points and dimensions. The artificial data sets are generated in such a way that the outliers are known. Hence the results are compared with respect to specificity and precision values for these data sets, where the proposed method is found to significantly outperform the other methods. Receiver operating characteristic (ROC) curves are also reported for three data sets. Results for real-life data sets shows the effectiveness of incorporating GCT in the genetic outlier detection method, which results in significantly faster execution of the algorithm. In a part of the investigation, the proposed method is used for detecting outliers in several gene expression data sets. The results once again highlight the effectiveness of using GCT.

Note that while computing the sparsity value, a simple partitioning scheme is used for limiting the time requirement of GCT-GOD. Several sophisticated techniques for detecting real data distribution on a subspace for high dimensional problem exists [6,7,14], which might be effective in improving the performance of the proposed method. However our purpose is to show that using GCT yields significant improvement in the timing requirement of the genetic scheme, e.g., GOD. It may be possible to integrate GCT with other techniques like subspace clustering to improve their efficiency. A comparative study of the performance of the proposed approach with those of methods like subspace clustering needs to be performed.

As a part of future research, in addition to computing the sparsity value, other methods of fitness evaluation need to be considered. An exhaustive comparison of the genetic scheme with other approaches like those based on different soft computing tools as well as with subspace clustering methods should be carried our. In this article, a tree structure is used to organize the hypergrids so as to make the computation of N' in projected subspaces easier. In this regard, use of some other more efficient data structure might be studied in future.

The main motivation of this paper is to improve on an earlier genetic approach, GOD, developed in Ref. [10], by proposing the GCT which significantly improves the time required for computing N' in Eq. (6). Thus following their approach, the data are assumed to be uniformly distributed. This assumption may not hold in practice. Moreover, discarding some dimensions, as followed in Ref. [10] without estimating the actual distribution of the data along these dimensions may not be effective. (In spite of this, the reason that this approach appears to work may be that it finally deals not with the exact sparsity values for the grids in different subspaces, but with those having much less than the expected sparsity value.) Therefore it is expected that incorporation of techniques for estimation of the actual data distribution, e.g., [22], will improve the accuracy of the method. Other techniques like principle component analysis, clustering, etc., can also be used in this regard. This constitutes an interesting area of future research.

Acknowledgment

The authors gratefully acknowledge the anonymous reviewers whose comments helped in improving the quality of the article.

References

[1] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann, Los, Altos, CA, 2001.
[2] E.M. Knorr, R.T. Ng, Algorithms for mining distance-based outliers in large datasets, in: Proceedings of VLDB Conference, 1998, pp. 392–403.
[3] A. Arning, R. Agrawal, P. Raghavan, A linear method for deviation detection in large databases, in: Proceedings of Knowledge Discovery and Data Mining, 1996, pp. 164–169.
[4] D. Hawkins, Identification of Outliers, Chapman & Hall, London, 1980.
[5] S. Ramaswamy, R. Rostogi, K. Shim, Efficient algorithms for mining outliers from large data sets, in: Proceedings of ACM SIGMOD Conference, 2000, pp. 427–438.
[6] L. Parsons, E. Haque, H. Liu, Subspace clustering for high dimensional data: a review, SIGKDD Explor. 6 (1) (2004) 90–105.
[7] R. Agrawal, J. Gehrke, D. Gunopulos, P. Raghavan, Automatic subspace clustering of high dimensional data for data mining applications, in: Proceeding ACM SIGMOD Conference, 1998, pp. 94–105.
[8] N.V. Chawla, N. Japkowicz, A. Kolcz, Editorial: special issue on learning from imbalanced data sets, SIGKDD Explor. 6 (1) (2004) 1–6.
[9] T. Fawcett, ROC Graphs: Notes and Practical Considerations for Researchers, Kluwer Academic Publishers, Dordrecht, 2004.
[10] C.C. Aggarwal, P.S. Yu, Outlier detection for high dimensional data, in: Proceedings of ACM SIGMOD Conference, 2001, pp. 37–47.
[11] C. Yan, G. Chen, Y. Shen, Outlier analysis for gene expression data, J. Comput. Sci. Technol. 19 (1) (2004) 13–21.
[12] F. Korn, T. Johnson, H. Jagadish, Range selectivity estimation for continuous attributes, in: Proceedings of International Conference on Scientific and Statistical Database Management, 1999, pp. 244–253.
[13] Z. Michalewicz, Genetic Algorithm + Data Structure = Evolution Program, Springer, Berlin, 1996.
[14] D. Jiang, C. Tang, A. Zhang, Cluster analysis for gene expression data: a survey, IEEE Trans. Knowl. Data Eng. 16 (11) (2004) 1370–1386.
[15] A. Chaudhary, A.S. Szalay, A.W. Moore, Very fast outlier detection in large multidimensional data sets, in: Proceedings of Data Mining and Knowledge Discovery, 2002.
[16] E.M. Knorr, R.T. Ng, Finding intensional knowledge of distance-based outliers, in: Proceedings of VLDB Conference, 1999, pp. 211–222.
[17] A.A. Freitas, Data Mining and Knowledge Discovery with Evolutionary Algorithms, Springer, Berlin, 2002.
[18] N. Ye, The Handbook of Data Mining, Lawrence Erlbaum, London, 2003.
[19] K. A. De Jong, Analysis of the behavior of a class of genetic adaptive system, Ph.D. Thesis, Department of Computer and Communication Sciences, University of Michigan, 1981.
[20] G. Parmigiani, E.S. Garett, R.A. Irizarry, S.L. Zeger, The Analysis of Gene Expression Data, Springer, Berlin, 2003.
[21] J. Wittes, H.P. Friedman, Searching for evidence of altered gene expression: a comment on statistical analysis of microarray data, J. Nat. Cancer Inst. 91 (5) (1999) 400–401.
[22] W. Feller, An Introduction to Probability and Its Application, Wiley, New York, 1968.
[23] A. Razzi, A convergence theorem for genetic algorithm, Metron 55 (1997) 69–83.
[24] C.C. Aggarwal, J.B. Orlin, R.P. Tai, Optimized crossover for the independent set problem, Oper. Res. 45 (2) (1997) 226–234.

[25] C. Gathercole, P. Ross, Small populations over many generations can beat large populations over few generations in genetic programming, in: Proceedings of GP-97, Stanford University, 1997, pp. 111–118.
[26] G. Kollios, D. Gunopulos, N. Koudas, S. Berchtold, Efficient biased sampling for approximate clustering and outlier detection in large data sets, IEEE Trans. Knowl. Data Eng. 15 (5) (2003) 1170–1187.
[27] M. M. Breunig, H. Kriegel, R.T. Ng, J. Sander, OPTICS-OF: identifying local outliers, Principle Data Min. Knowl. Discovery (1999) 262–270.
[29] S. Guha, R. Rastogi, K. Shim, A robust clustering algorithm for categorical attributes, in: Proceedings of International Conference on Data Engineering, 1999, pp. 512–521.
[30] V. Barnet, T. Lewis, Outliers in Statistical Data, Wiley, New York, 1994.
[31] W. Jin, A.K.H. Tung, J. Han, Mining top-n outlier in large databases, in: Proceedings of the ACM SIGKDD, 2001, pp. 293–298.
[32] Z. He, X. Xu, Shengchun Deng, Discovering cluster-based local outliers, Pattern Recognition Lett. 24 (9–10) (2003) 1641–1650.

About the Author—SANGHAMITRA BANDYOPADHYAY (SM'05) did her Bachelors in Physics and Computer Science in 1988 and 1991, respectively. Subsequently, she did her Masters in Computer Science from Indian Institute of Technology (IIT), Kharagpur in 1993 and Ph.D. in Computer Science from Indian Statistical Institute, Calcutta in 1998. Currently she is an Associate Professor, Indian Statistical Institute, Kolkata, India. Dr. Bandyopadhyay is the first recipient of Dr. Shanker Dayal Sharma Gold Medal and Institute Silver Medal for being adjudged the best all round post graduate performer in IIT, Kharagpur in 1994. She has worked in Los Alamos National Laboratory, Los Alamos, USA in 1997, University of New South Wales, Sydney, Australia in 1999, Department of Computer Science and Engineering, University of Texas at Arlington, USA in 2001, University of Maryland, Baltimore County, USA in 2004, Fraunhofer Institute AiS, St. Augustin, Germany in 2005 and Tsinghua University, China in 2006. She has delivered lectures at Imperial College, London, UK, Monash University, Australia, University of Aizu, Japan, University of Nice, France, University Kebangsaan Malaysia, Kuala Lumpur, Malaysia, and also made academic visits to many more Institutes/Universities around the world. She is a co-author of 3 books and more than 100 research publications. Dr. Bandyopadhyay received the Indian National Science Academy (INSA) and the Indian Science Congress Association (ISCA) Young Scientist Awards in 2000, as well as the Indian National Academy of Engineering (INAE) Young Engineers' Award in 2002. She has guest edited several journal special issues including IEEE Transactions on Systems, Man and Cybernetics, Part—B. Dr. Bandyopadhyay has been the Program Chair, Tutorial Chair, Associate Track Chair and a Member of the program committee of many international conferences. Her research interests include pattern recognition, data mining, evolutionary and soft computation, bioinformatics and parallel and distributed systems.

About the Author—SANTANU SANTRA received his Bachelor degree in Computer Science from Midnapore College, Vidyasagar University in 2002. He did his Masters in Computer Science from Vidyasagar University in 2004. He is the recipient of University Gold Medal in 2002. At present he is a Project Linked Personnel in Machine Intelligence Unit at Indian Statistical Institute, Kolkata. His research interests include pattern recognition, genetic algorithm, data mining, bioinformatics and drug design.