Towards Effective Subspace Clustering with an Evolutionary Algorithm

Ioannis A. Sarafis
School of Mathematical and Computer Sciences
Heriot-Watt University, EH14 4AS
Edinburgh, United Kingdom
I.Sarafis@hw.ac.uk

Phil W. Trinder
School of Mathematical and Computer Sciences
Heriot-Watt University, EH14 4AS
Edinburgh, United Kingdom
P.W.Trinder@hw.ac.uk

Ali M. S. Zalzala
School of Engineering and Physical Sciences
Heriot-Watt University, EH14 4AS
Edinburgh, United Kingdom
A.Zalzala@hw.ac.uk

Abstract- This paper proposes a new evolutionary algorithm for subspace clustering in very large and high dimensional databases. The design includes task-specific coding and genetic operators, along with a non-random initialization procedure. Reported experimental results show the algorithm scales almost linearly with the size and dimensionality of the database as well as the dimensionality of the hidden clusters. Our algorithm is able of discovering clusters of different densities embedded in low as well as in high dimensional subspaces of the original space. Finally, the discovered knowledge is presented in the form of non-overlapping clustering rules where only those features that are relevant to the clustering are reported. These two properties contributes to the high comprehensibility of the clustering output.

1 Introduction

Clustering is a common data analysis task that aims to partition a collection of objects into homogeneous groups, called clusters [10]. Objects assigned to the same cluster exhibit high similarity among themselves and are dissimilar to objects belonging to other clusters. Although, the idea of representing a group of similar object using its cluster descriptor loses fine details, it is very useful especially for large and high dimensional datasets to provide a simplification of the underlying distribution of objects. Many distance-based clustering algorithms that have been proposed over the last decade assume that the clusters exist in the full-dimensional feature space [8]. This means that all dimensions are given the same importance when computing the distance between points. However, the performance of distance-based algorithms degenerates rapidly with increasing dimensionality since in a very high dimensional space the distance of a point to its farthest and nearest neighbouring point is nearly the same [9]. The fact that different points may cluster better for different subsets of dimensions has been observed for the first time by the authors of the CLIQUE clustering algorithm [2].

Evolutionary Algorithms (EAs) are powerful optimization techniques based on the mechanics of natural selection and natural genetics [3]. The globalized and non-deterministic nature of an evolutionary search has inspired many researchers to use them for performing the actual search either for the centers of clusters in the context of k-means clustering algorithm [6] or for discovering sets of comprehensible and non-overlapping clustering rules [11].

In this paper we propose a novel evolutionary algorithm suitable for performing effective and efficient subspace clustering in very large and high dimensional datasets. The paper is organized as follows. In section 2 we introduce the problem of subspace clustering for high dimensional datasets and present the encoding scheme for the individuals. Section 3 discusses briefly the fitness function used in our system. Section 4 presents a task-specific genetic operator, namely the generalization operator, which contributes to high comprehensibility of the discovered clustering rules. In section 5 we propose a non-random initialization procedure that incorporates problem-specific knowledge to overcome the problem of poor initialization of the population members. In sections 6, 7 we present the special devised mutation and recombination operators, respectively. Section 8 discusses another task-specific genetic operator, namely the repair operator, which bias the search in discovering homogeneous clusters. The results of our experiments on synthetic datasets are reported in section 11. Finally, section 12 concludes the paper and briefly discusses potential improvements of the current work.

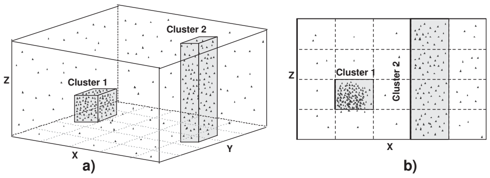

Figure 1: Cluster 2 is embedded in a subset of the original space

2 Individual Encoding

The basic idea of subspace clustering is to identify subspaces of the original high dimensional feature space that allow better clustering of the points than the original space [2]. Figure 1(a) shows an example three-dimensional data set containing two clusters, namely *cluster 1* and *cluster 2* that are defined in dimensions X×Y×Z and X×Y, respectively. Apparently, cluster 2 is bounded only in two dimensions X and Y (see projection of patterns in subspace XZ in figure 1(b)) while the third dimension Z is irrelevant to the clustering of these points. Our algorithm attempts a) to discover the subset of dimensions in which the clusters are embedded and b) to produce comprehensible and non-overlapping clustering rules describing the non-bounded dimensions.

[Sarafis et al. 2003] proposed a non-binary, rule-based representation, containing a variable number of non overlapping rules, where each rule consists of a fixed number of *d* conjunctions (intervals),

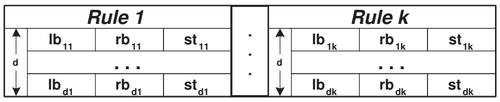

Figure 2: Variable-length genotype with fixed-length rules

one for each feature [11]. The boundaries of the intervals (left (*lb*) and right (*rb*)) are drawn from discrete domains, which reflect the automatic quantization of the feature space into a multidimensional grid. Note that the consequent part of the clustering rules does not need to be encoded into the chromosome since there are no overlapping rules. In order to effectively deal with subspace clustering the above encoding scheme is further enhanced by introducing an additional field for each feature in each rule, called *status* (*st*) (figure 2). st_{ij} is a flag indicating whether the i-th feature of j-th rule is relevant (active -"1") or irrelevant (inactive -"0") to the clustering of patterns that are covered by the j-th rule. An inactive status st_{ij}=0, corresponding to i-th feature is a strong indication that the patterns within that rule are spread out along the i-th dimension in a large extent and as a consequence the corresponding i-th condition does not need to be reported to an end-user because is not relevant to the clustering. Although the comprehensibility of the discovered knowledge in data mining is a subjective concept there are some ways based on objective criteria that can be used to measure it, such as number and length of rules (syntactical complexity) or level of abstraction associated with the features occurring in the discovered rules [5]. In our case, although the antecedent part of rules in the genotype of individuals has a fixed-length (d), inactive feature genes (conditions) are interpreted in a such a way that the phenotype of individuals (rules reported to an end-user) have variable-length. To decide when the *i*-th dimension of a rule R is irrelevant to the clustering we apply the following procedure, which is based on a comparison between the extent of spread of patterns within R and within the entire feature space regarding the *i*-th dimension. Generally speaking, if the patterns are projected onto *i*-th dimension then the outliers in that dimension can be statistically defined as the patterns lying more than (1.5*IQR$_i$) beyond the first Q1$_i$ or third Q3$_i$ quartile, where IQR$_i$ denotes the interquartile range in *i*-th dimension [7]. Moreover, lets assume that l_i and r_i denote the left and right bounds of the *i*-th domain D_i=[l_i, r_i], respectively. We suggest that the extent of spread of patterns in *i*-th dimension can be approximated using the length of the interval SP_i=[max(l_i, (Q1$_i$-1.5*IQR$_i$)), min(r_i, (Q3$_i$+1.5*IQR$_i$))]. Assuming that the function $I(R, SP_i, i)$ returns the amount of intersection between the *i*-th gene of a rule R and SP_i then whenever the ratio of $I(R, SP_i, i)$ to SP_i exceeds a user-defined threshold t_{status} the *i*-th feature can be effectively considered as inactive.

$$\frac{I(R,SP_i,i)}{SP_i} \geq t_{status} \tag{1}$$

3 Fitness Function

The fitness function used in our system is greedy with respect to the number of patterns that are covered by the rules of individuals and has been successfully employed by another EA-based clustering algorithm, called NOCEA [11]. In particular, NOCEA aims to maximize the *coverage C*, which is defined to be the percentage of *total patterns or points* n that are covered by the rules of the individual:

$$\text{C} = \max\left(\frac{1}{n}\sum_{i=1}^{k} n_i\right) \tag{2}$$

where, k denotes the number of rules and n_i the number of patterns in ith rule. In our clustering context, if the selectivity of a rule R (percentage of the points that are covered by R) is lower than a user-defined threshold t_{sparse}, then R is very sparse and consequently is discarded.

4 Generalization Operator

In supervised learning (classification) various generalization/specialization operators have been suggested aiming to remove/insert a condition from/into the antecedent part of a conjunctive rule, respectively [5]. In our case (unsupervised learning) the goal of the generalization operator is twofold: a) to replace whenever it is possible two adjacent rules with a single and more generic rule and b) to help in detecting inactive features. Two rules are adjacent if they have a common face, i.e there are d-1 dimensions where there is an intersection between the corresponding genes and there is one dimension where the left bound of one rule is adjacent to the right bound of the other one. For instance, the 2-dimensional rules R1 and R2 shown in figure 3 are adjacent because there is intersection between their corresponding genes in the vertical axis while in horizontal axis the right and left bound of R1 and R2, respectively, are touching.

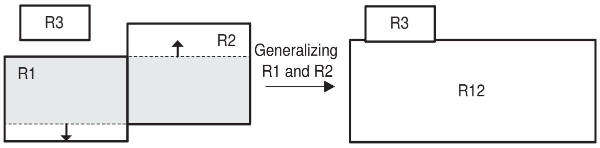

Figure 3: Generalizing two adjacent rules R1 and R2

The application of the generalization operator is subject to some constraints imposed a) by the extent (length) of intersection between the corresponding genes of the non touching features and b) by ratio of densities between the two adjacent rules. In particular, lets assume that the function $intersection(i, j, k)$ returns the extent of intersection between the touching rules R$_i$ and R$_j$ in k-th dimension. Additionally, let d_i denote the density of i-th rule, which is computed by dividing the number of patterns n_i that are covered by that rule with its d-dimensional volume V_i. The volume V_i of i-th rule is simply the product of the d sides, i.e $V_i = l_1 * l_2 * ... * l_d$, where $l_k = rb_{ki} - lb_{ki}$. The generalization operator can only be applied if the pair of adjacent rules R_i and

R_j satisfy the following three conditions:

$$\frac{\textit{intersection(i, j, k)}}{\mathrm{rb}_{ki}\text{-}\mathrm{lb}_{ki}} \geq \mathrm{t}_{intersection}, \qquad \text{k=1...d, } \mathrm{k} \neq \text{cf\_at} \quad (3)$$

$$\frac{\textit{intersection(i, j, k)}}{\mathrm{rb}_{kj}\text{-}\mathrm{lb}_{kj}} \geq \mathrm{t}_{intersection}, \qquad \text{k=1...d, } \mathrm{k} \neq \text{cf\_at} \quad (4)$$

$$\frac{\min(\mathrm{d}_i, \mathrm{d}_j)}{\max(\mathrm{d}_i, \mathrm{d}_j)} \geq \mathrm{t}_{density} \quad (5)$$

where $t_{intersection}$, $t_{density}$ are input parameter while cf_at denotes the dimension where the rules are touching.

Briefly speaking, conditions 3 and 4 ensures that both size and relative positioning of the two rules under generalization allow the replacement of these rules with a more generic one. Moreover, touching rules having similar densities (condition 5) is a strong indication that may cover patterns belonging to the same cluster and as a consequence the generalization is desirable to achieve comprehensibility.

The generalization operator first determines a d-dimensional region which is defined by the intersection of i-th and j-th rules in each non touching feature. In particular the left and right bounds of the incomplete generalized rule R_{ij} are as follows:

$$[\mathrm{lb}_k, \mathrm{rb}_k]= \begin{cases} [\min(\mathrm{lb}_{ki}, \mathrm{lb}_{kj}), \max(\mathrm{rb}_{ki}, \mathrm{rb}_{kj})], \text{ k=cf\_at} \\ [\max(\mathrm{lb}_{ki}, \mathrm{lb}_{kj}), \min(\mathrm{rb}_{ki}, \mathrm{rb}_{kj})], \text{ k}\neq\text{cf\_at} \end{cases} \quad (6)$$

Having determined the incomplete generalized rule R_{ij} (shadowed region in figure 3) the generalization procedure proceeds dimension-by-dimension with a random order. More precisely, R_{ij} is gradually expanded to all non touching features in a way that no overlapping with neighbouring rules occurs. The expansion to the left and to the right in k-th dimension is theoretically bounded by min(lb_{ki}, lb_{kj}) and max(rb_{ki}, rb_{kj}), respectively. However, in practice it is likely that the amount of expansion can be smaller if there exist(s) rule(s) which overlap with R_{ij}. For instance, considering the vertical axis in figure 3, it is obvious that the shadowed region is expandable to the left bound of rule R3 and not to the right bound of rule R2, because such a growing operation causes overlapping between the generalized rule and R3.

In general, the fewer the number of rules comprising the discovered knowledge the more comprehensible the output of a data mining task is [5]. Additionally, the shorter a rule, i.e the fewer conditions involved in the antecedent part, the more comprehensible it is [5]. Our proposed generalization operator ensures in some extent comprehensibility of the clustering output because: a) *reduces the number of reported rules by replacing pairs of adjacent rules with a single and more generic rule* and b) *it is likely that the generalized rule might be "large" enough in one or more dimensions triggering the deactivation of the corresponding genes*.

A drawback of the proposed generalization scheme is the fact that it is not always guaranteed that the selectivity of the generalized rule is greater or at least equal to the aggregated selectivity from the two adjacent rules. This is because neighbouring rules which may intersect with the generalized rule do not allow its expansion in full scale. This problem is exacerbated when the chosen value of the threshold $t_{intersection}$ is very small, which can be an obstacle to the EA in preserving rules having relatively large selectivity. On the other hand, choosing relatively large values for $t_{intersection}$ implies that the triggering of the generalization operator is not frequent and as a consequence the comprehensibility of the discovered knowledge might be poor. Finally, the generalization operator is applied to offsprings immediately after the crossover operation.

5 Population Initialization

5.1 Sparsity of High Dimensional Data Spaces

Usually, when an EA searches a solution space and *no special problem-specific knowledge* is exploited we generally seek to initialize its population so that all possible solutions are equally likely to appear. A major drawback of random initialization is that there is no guarantee that the randomly generated rules of individuals will cover an "adequate" large number of patterns so as not to be labelled as sparse rules. Recalling from section 3 that the function to be optimized is the maximization of the coverage it is very likely that a random initialization may generate individuals that either do not contain any non-sparse rules or have extremely low fitness. A possible solution to the above problem is to randomly initialize the population members with only *general* rule(s), i.e one that covers a very large portion of the feature space. However, even if we bias the initialization process to favour the random generation of general rules still there is no guarantee that the above problem is solved because of the tremendous increase in the size of the search space as the dimensionality of the dataset increases. For instance, assume we want to initialize each population member randomly with a single rule in a 50-dimensional feature space where each dimension is split only once along the median value. The resulting space consists of 2^{50} cells which can be translated as 2^{50} "different" choices in initializing a single rule. Now, each generated cell contains $(n/2^{50}) \to 0$ patterns because usually $n \ll 2^{50}$, recall from section 3 that n denotes the size of the dataset. Thus, it is impossible to generate a non sparse rule since the maximum number of patterns lying within a cell can not exceed the value one. Due to this "sparsity" of the feature space we must incorporate special problem-specific knowledge which will help us to overcome the problem of poor initialization.

5.2 Non-Random Initialization

We suggest a non-random initialization procedure which is based on a monotonicity lemma according to which *"...if a collection of points S is a cluster in a k-dimensional space, then S is also part of a cluster in any (k-1)-dimensional projections of this space"* [2]. In particular, the initialization phase is geared towards finding clusters existing in 2-dimensional projections of the original feature space. Due to the fact that the projection of patterns in a 2-dimensional subspace (say X×Y) does not consider the remaining dimensions, the discovery of a cluster in a subspace X×Y implies that there is *at least one* cluster in the original feature space existing in *at least two* dimensions (X and Y). In general, for each 2-dimensional subspace i×j the application of the initialization procedure described in figure 4 yields the generation of a set of non-overlapping 2-dimensional clustering rules denoted by S_{ij}. Having discovered a 2-dimensional cluster C in the subspace X×Y we can now generate a seed (d-dimensional rule) R by setting the value of genes X and Y based on the corresponding genes in C and initializing the remaining genes so as to cover the entire domain. It should be pointed out that members of S_{ij} can only be those rules where the condition 1 in section 2 is satisfied for both dimensions i and j. Additionally, subspaces where $S = \varnothing$ are omitted from further investigation.

As the dimensionality of the dataset increases there is an explosion in the number of candidate 2-dimensional subspaces that can be used to initialize an individual. Because the maximum number $((d * (d-1)/2)$ of candidate subspaces can be considerably higher comparing to the size of the population we need to prune the pool of candidates. To decide which 2-dimensional subspaces (and the corresponding rules) are the most interesting we compute for each subspace its coverage which is the percentage of total patterns that are covered by the rules in it. Obviously the initialization procedure must favour subspaces with large coverage because of the form of the fitness function used in our system (see section 3). For this reason we sort the subspaces in the descending order of their coverage. Finally, the i-th individual is initialized using the seeds (rules) that are contained in the i-th subspace. Note that if the number of individuals exceeds the size of the list containing the ordered subspaces, the pointer showing the next subspace to be chosen is reset to the first element of the list and the above procedure is repeated for all the remaining uninitialized individuals. Clearly, the above initialization procedure is non random and can generate one or more individuals with the same genetic material.

Seed Discovery Procedure in Two Dimensions

```
Input(i : feature in x-axis,   j : feature in y-axis)
Output(Sij : a vector of 2-dimensional rules existing in subspace (i×j))
{R, R': 2-dimensional rules}, {V, VR : vectors containing 2-dimensional rules}
{initialize(i, j) : returns a rule R covering the entire 2-dimensional subspace (i×j)}
{add(V, VR) : adds all the elements (rules) of VR to V}
{get(V, k) : returns the k-th rule from V (first element is stored in position 1)}
{remove(V, R) : removes rule R from V}
{size(V) : returns the number of rules in V}
{shuffle(V):returns (randomly) a rule from V that has not previously been examined}
{expand(R) : expands R in full scale without causing any overlapping}
{equals(R, R'):checks if R and R' have similar values for their corresponding genes}
{repair(R) : repairs R and returns a vector with the repaired rules}
{repairAll(V) : repairs all rules in V and returns a vector with the repaired rules}
{generalize(V) : returns a vector of rules after generalizing the rules in V}
{getClusters2D(V): returns a vector containing all the valid rules in V}
1.BEGIN
2.  Seedij := ∅;
3.  R := initialize(i, j);
4.  V := repair(R);
5.  DO{
6.     IF(size(V) = 0) THEN {Sij := ∅; terminate;}
7.     FLAG := true;
8.     FOR(k=1; k≤size(V); k++){
9.        R := shuffle(V);
10.       expand(R);
11.       VR := repair(R);
12.       IF(size(VR) = 0) THEN{remove(V, R); goto 5;}
13.       IF(size(VR) > 1)THEN {remove(V, R); add(V, VR); goto 5;}
14.       IF((size(VR) =1)∧¬equals(R,get(VR,1))) THEN {remove(V, R);
.                                                       add(V,VR); goto 5;}
15.       IF((size(VR) = 1) ∧ equals(R, get(VR, 1))) THEN {FLAG=false;}
16.    }
17.  }
18. WHILE(FLAG=true)
19.  V := generalize(V);
20.  V := repairAll(V);
21.  Sij := getClusters2D(V);
22.END.
```

Figure 4: Pseudo-code for seed discovery.

In the remaining part of this section, we detail how the proposed initialization scheme discovers clusters existing in 2-dimensional projections of the original feature space. We start by generating a 2-dimensional rule R which covers the entire (i×j) subspace. The next step is to apply the repair operator (see section 8) to R in order to get a set of rules that are homogeneous as far as the distribution of patterns lying inside these rules is concerned. Assume that the generated rules are stored in a vector V. Note that V consists of two types of rules a) *valid* rules that exists in the current subspace and b) *invalid* rules that do not exist in the current subspace. Obviously, only valid rules are interesting and can be used as seeds to initialize individuals. If the initial repairing generates an empty set of rules the seed discovery procedure terminates and the algorithm proceeds to the next candidate subspace. The next step is to grow each rule in V as much as possible in both axes in a way that no overlapping with other rules occurs. The expansion of a rule is followed by an immediate application of the repair operator in it. The order by which rules are processed is random. If the combined effects of expansion and repairing alter the genes of the rule that is currently being processed then we omit the rules that have not been expanded-repaired yet and the above procedure is reseted starting from an arbitrary chosen rule. Otherwise, if no modification in the current set of rules is observed after their expansion and repairing, the elements (rules) in V undergo generalization as described in section 4 to reveal (if possible) any invalid rule(s). Finally, after the generalization step the repair operator is applied for a last time and all the elements in V which correspond to valid rules are returned as seeds.

6 Mutation Operators

In a typical EA the mutation operator acts on a single individual at time by replacing the value of a gene with a randomly-generated value [3]. Usually the probability of altering a gene is relatively low. The mutation along with the recombination operator aim to increase the genetic diversity of the population (*exploration*), while the selection operator emphasizes good prior solutions by assigning many copies of them (*exploitation*).

In our system, the mutation operator has two functionalities, *a)* to perform local-fine tuning by increasing the size of existing rules and *b)* to discover previously unknown and potentially promising regions within the enormous d-dimensional search space. Hereafter, we will refer to the above two functionalities of mutation operator as *growing-mutation* and *seed-mutation*, respectively.

Growing-Mutation. The *growing-mutation* has been suggested in [11] and as its name implies aims to increase the size of existing rules in an attempt to cover as many patterns as possible but with a minimum number of rules. Briefly speaking, lets assume that the *i*th dimension of *j*th rule R undergoes growing-mutation. Additionally, lets m_i denotes the total number of bins in *i*th dimension (the first bin is always assigned the label "0"). We can compute the maximum value $rb_{ij(max)}$ that rb_{ij} can be extended to the right without causing overlapping as follows:

1. Sort in ascending order all the rules that their left bound is greater than the right bound of rule R in *i*th dimension.

2. If the sorted list (SL) is empty, set $rb_{ij(max)}$ to (m_i-1) and terminate. Otherwise, proceed to step 3.

3. Pick the next element RN of the list. If there is intersection in every dimension excluding the *i*th between R and RN, set $rb_{ij(max)}$ equal to the left bound of RN in *i*th dimension minus one and terminate the loop. Otherwise, repeat step 3. In case that there is no any element in SL which along with R satisfies the condition described in step 3, $rb_{ij(max)}$ is set to (m_i-1).

The derivation of the minimum value $lb_{ij(min)}$ that lb_{ij} can be extended to the left without occurring overlapping is the dual of computing $rb_{ij(max)}$. Obviously, the allowable range of modification due to growing mutation is solely determined by the relative position of rules within the d-dimensional feature space. Thus, lb_{ij} and rb_{ij} are randomly mutated within the intervals [$lb_{ij(min)}$, lb_{ij}) and (rb_{ij}, $rb_{ij(max)}$], respectively. An example of growing-mutation is shown in figure 5(a) where the allowable region of modification for the right boundary of rule R1 in the horizontal axis is bounded by the corresponding left boundary of rule R3. Note that the rule R2 does not impose any constraint because there is no intersection with R1 in the vertical axis.

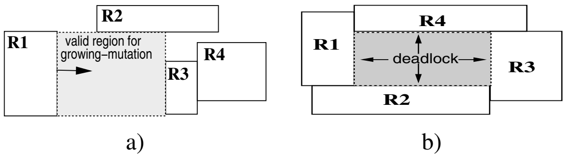

Figure 5: A successful growing mutation (a) and a failure to perform local fine-tuning using growing-mutation (b).

Seed-Mutation. In this paper we propose a special type of mutation operation, called *seed-mutation* that can significantly enhance the exploratory power of our evolutionary algorithm. Additionally, the seed-mutation is complementary to the growing-mutation in cases where the latter can not perform local fine tuning due to the strict requirement that no overlapping between rules is allowed. For instance, consider the case depicted in figure 5(b) where the evolutionary search has reached a deadlock in covering the central grey region. More precisely, due to the relative position of the rules none growing-mutation operation can ensure that the grey region will be covered.

Figure 6 depicts the working mechanism of the seed-mutation and demonstrates its usefulness in discovering isolated clusters. In particular, if the right bound of *j*th rule undergoes seed mutation in *i*-th dimension the operation proceeds as follows. Firstly, the algorithm determines a d-dimensional region, called *right bounding box* (RBB_{ij}) that corresponds to rb_{ij}. If there is at least one cell within the RBB which is not currently covered by other rules the algorithm can use this cell as seed in order to create a more generic rule. The left (*lbb*) and right (*rbb*) boundaries of RBB_{ij} are computed as follows:

$$[lbb_k, rbb_k]= \begin{cases} [(rb_{kj}+1), (m_k\text{-}1)], \text{ if } k=i \\ [lb_{kj}, rb_{kj}], \text{ if } k\neq i \end{cases} \qquad (7)$$

After having built the RBB the algorithm chooses an uncovered cell within it, arbitrarily . The dashed arrow in figure 6 shows this non-deterministic way of generating a seed (rule) within the RBB that corresponds to the horizontal axis of rule R1. The next step is to grow this seed in every dimension, both to the left and to the right, as much as possible without causing overlapping. The expansion is performed dimension-by-dimension in a random order. The rationale behind the full-scaled expansion of the seed is *a*) to increase the probability of discovering isolated clusters (i.e uncovered cluster C in figure 6) and *b*) to deal with clusters that are embedded in subspaces of the original feature space, i.e to ensure that the corresponding rule is sufficiently "large" along the non-active features. Note that if the RBB does not contain uncovered cell(s) the seed-mutation operation is aborted. Finally, a left-side seed-mutation is the dual of the procedure described above. As discussed earlier in this section one of the main challenges for the seed-mutation is to effectively replace the growing-mutation in cases where the later reaches a deadlock in covering particular regions of the search space. Obviously, our evolutionary algorithm can easily deal with the apparent deadlock shown figure 5(b) by employing a seed-mutation operation in the appropriate boundary of one of the neighbouring rules.

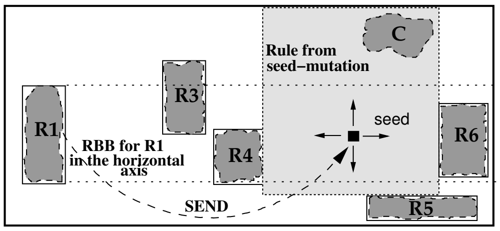

Figure 6: An example of seed-mutation.

7 Recombination Operator

In a typical binary-coded EA, during the recombination (or crossover) operation, two individuals (strings of bits) are picked from the mating pool at random and genetic material (portion of strings) is exchanged between them to create two new offspring [3]. In *one-point crossover* operation a cross site along the string is randomly chosen and all bits after that point are swapped. Another popular form of recombination is the *two-point crossover*, which is performed by choosing two cross sites randomly and swapping the corresponding segments from the two parents defined by the cross sites. Unlike one and two-point crossover which are segment-oriented, *uniform crossover* (*UC*) operates in bit-level by exchanging individual bits between the parents with a fixed and position-independent probability p_{uc}, which is usually set to 0.5.

There are two forms of bias associated with the crossover operator: *positional* and *distributional* [3]. Positional bias refers to the extent that the probability of passing a set of symbols (e.q bits) intact to the offsprings depends on their relative position in the genome. Distributional bias refers to the number of symbols that are swapped between the parents and the extent to which this number is distributed around some value(s). One and two point crossover have a strong positional bias because the probability of adjacent genes being transmitted intact is relatively higher than the probability of swapping distant genes intact. This strong positional bias is a drawback in the context of prediction-rule discovery with EA [5]. This is due to the fact that the crossover operator tends to disrupt combinations of features that are distant in the genotype and inversely to preserve intact combinations of attributes that are adjacent in the genotype. However, in the phenotype the features have no ordering. One-point crossover is free from distributional bias since the number of bits being transmitted is uniformly distributed in the range [1, L-1] (L: length of string). In contrast, UC has no positional bias because the probability of a gene being swapped is independent of the relative position of that gene in the chromosome. A major drawback of UC is its strong distributional bias since the expected number of symbols that are being transmitted is $p_{uc} * L$.

In this paper we introduce a novel crossover operator, called *imported rule crossover* (*IRC*), which operates on two parent solutions and creates two offspring. As the name suggests the IRC operator works by importing rules into offsprings rather than exchanging rules between the parents. The functionality of IRC operator is geared towards: *a*) *elimination of positional bias* *b*) *minimization of distributional bias* and *c*) *generation of non-lethal offsprings* (solutions containing no overlapping rules). Unlike the mutation operator, which alters the genetic material in a feature-gene level, IRC operates on different (higher) level, that is, the rule-level. In the rule-level, each rule can be viewed as a gene and consequently an individual consists of a vector of rule-genes. Note that this level of abstraction explicitly disregards any linkage among rule-genes in the chromosome. This is correct since there is no overlapping among rules and each rule (gene) is an independent entity. Additionally, the position of rule-genes on the parental chromosome is "unimportant". The strict requirement that no overlapping between rules is allowed has an important implication: Unlike traditional crossover schemes (one-point, UC etc.) where the genes (bits) can be transmitted to offsprings without any constraint, in our case an arbitrary swapping of rule-genes it is likely to cause the generation of lethal offsprings. Consequently, traditional bitwise crossover operators are not directly applicable in our context. Bearing in mind that the rule-genes are not correlated entities, the term *disruption* of the genetic material after a crossover operation, can be simply defined as the splitting of rules presented in the parental chromosomes to avoid overlapping in the offsprings.

In order to mitigate the disruptive effect of crossover but at the same time to encourage vigorous exploration we devised the IRC operator. The key ideas in the design of IRC is *a*) to preserve intact all the rules presented in the parents by directly copying them to the corresponding offsprings (mitigation of the disruptive effect of crossover) and *b*) to "import" in each offspring the entire rule(s) or part(s) of the rule(s) from the other parent in a way that no overlapping occurs (exploration). Figure 7 depicts an example of IRC operation where the colour of a rule indicates its origin. In particular, rules or part(s) of rules with light-grey colour are originated from parent A while dark-grey coloured rules are originated from parent B. IRC firstly clones the parents A and B to create the offspring A and B, respectively. The next step is to import in offspring A all the rules from parent B that do not overlap with rules in offspring A. For instance, rule B3 from parent B is directly copied to offspring A without any modification since such an operation does not violate the non-overlapping constraint. IRC then searches for rules in parent B that are fully covered by rules presented in the chromosome of parent A. These rules (if there exist) can be effectively omitted from further analysis because the d-dimensional region that represent in in the search space is already covered by some rule(s) in offspring A. For example, the rule B1 from parent B is fully contained in rule A1 of parent A and thus B1 is not imported to offspring A.

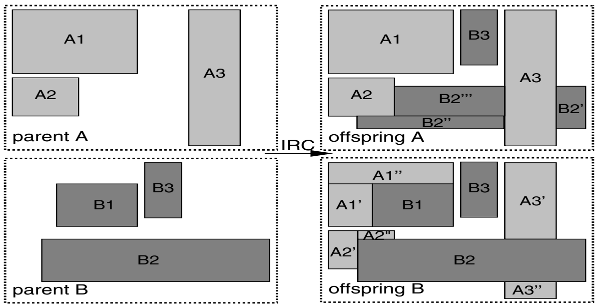

Figure 7: An example of IRC crossover operation.

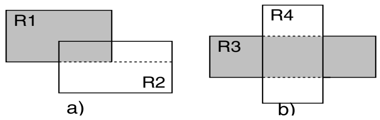

Figure 8: Examples of resolving overlapping

The last step during an IRC operation is to import all the remaining rules (if there exist any) from parent B to offspring A, which in turn requires the resolution of all the instances of overlapping. Assume that all rules from parent B that neither overlap with rules in offspring A nor covered fully by rules in offspring A are inserted into a vector V. The procedure described below is repeated until V has no more elements to proceed:

Step 1. Pick an element R_V from V at random.

Step 2. Pick a rule R_A from offspring A that intersects with R_V at random.

Step 3. Split R_V along d_{split} dimension in the interface bin with R_A. Remove R_V from V and add the rule(s) generated from breaking R_V into V. Remove all rules from V that do not overlap anymore with rules in offspring A and import these rules to offspring A. Go to step 1. Note that d_{split} is chosen at random. Furthermore, in d_{split} dimension the intersection between R_V and R_A must be different to the value of the corresponding feature gene of R_V to avoid unnecessary breaking of R_V along that dimension.

It is important to note that the number of possible ways to resolve overlapping among rules can grow significantly as the dimensionality of the dataset increases. However, since IRC is biased in transmitting parental rules intact to corresponding offspring, clearly only rules imported from the other parent must undergo (if necessary) splitting. Basically, we can distinguish two types of overlapping along a particular dimension, as shown in figure 8. Lets assume that the rule that is currently being imported into the offspring has no colour and the vertical axis has been chosen as d_{split}. Figure 8(a) demonstrates the case where there is a partial intersection between the rules R1 and R2 along d_{split} (vertical axis). In this case, the imported rule R2 is split in vertical axis along the dashed line resulting the generation of two rules. One of them does not overlap with R1 anymore and can be imported as it is in the corresponding offspring while the second rule will undergo additional splitting operation(s). In case where the imported rule R4 covers fully the rule R3 in d_{split} dimension the splitting is performed in two sites as shown in figure 8(b) resulting the generation of three rules. Finally, the above procedure of importing rules is similar in case of offspring B.

In summary, we propose a novel crossover operator that:

- guarantees the generation of non-lethal offsprings.
- mitigates in a certain extent the disruptive effect of the crossover operation by preserving intact all the rules presented in a parental chromosome in at least one of the two offspring chromosomes.
- allows useful exploration of the feature space because the genetic material of the parents is potentially enhanced with new rule-genes which are yielded by splitting parental rules.
- has no positional bias since the position of rule-genes is "unimportant" to the crossover operation.
- has no any specific distributional bias because the number of rules transmitted to offsprings is only dependent to the relative position of rules within the feature space. However, it should be pointed it out that during the early stages of the evolutionary search the number of rules that are imported is expected to be large but as the search progresses and the "similarity" of the genetic material of the individuals increases this number is expected to be considerably smaller.

8 Repair Operator

A closer look to the form of the the fitness function employed in our system can easily reveal that it is "blind" regarding the "quality" of the clustering output. The term quality refers to the ability of the clustering algorithm a) to separate adjacent and distinct clusters b) to capture with a certain extent of precision arbitrary-shaped clusters and c) to ensure that the d-dimensional region described by a rule is homogeneous, as much as possible. Clearly, the above quality characteristics can not be delivered through an evolutionary search which is guided only by the fitness function discussed in section 3. This is because that fitness function lacks any property that can bias the evolutionary search in discovering rules that satisfy the above quality characteristics, since it only attempts to maximize the coverage of the patterns.

In order to deliver in the clustering output the above-mentioned quality characteristics we devised a task-specific genetic operator, called *repair operator*. The design of the repair operator is motivated by the observation that clusters become separated because of the different extent of condensation of patterns between them. In order to find the borders between the clusters and to achieve precision in capturing their shape we developed the repair operator which works using parallel-axis projections of the patterns. Firstly we introduce some basic definitions in order to explain the functionality of the repair operator. The *selectivity* s_{ij} of *i*th bin in *j*th dimension of a rule R is the fraction of total patterns in R that lie inside that bin. The *Selectivity Control Chart* (*SCC*) in *j*th dimension is a histogram having an upper (*UCL*) and a lower (*LCL*) control limits. Additionally, measurements that correspond to the selectivity of bins in *j*th dimension are plotted on the chart.

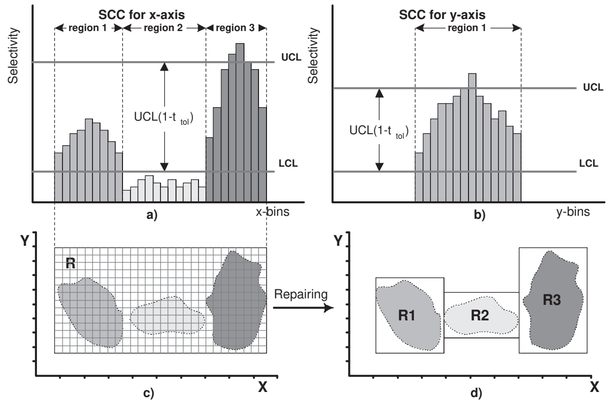

Figure 9: The working mechanism of the repair operator.

Often, the condensation of patterns in some region(s) of a cluster (e.g. central regions) can be relatively higher comparing to other region(s) (e.g. border regions) within the same cluster. Intuitively, UCL can be viewed as indicator of the condensation of patterns within the densely-populated region(s) of a cluster. In this paper we propose an estimator for UCL which is based on ordered statistics. Lets assume that the ordered list *OLS* (*Ordered List of Selectivity*) contains measurements, which correspond to the selectivity of bins in *j*th dimension in ascending order. Although it is possible to use the last element of OLS as value for UCL, this choice can create a misleading indicator regarding the condensation of patterns within the dense region(s) of a cluster, in case where the last element of OLS differs significantly from the others. A more realistic choice for UCL which is also more resistant to extreme values can be the upper decile or upper quantile of OLS. In the reported experiments the upper decile was used as an estimator of UCL. On the other hand the LCL can be viewed as a limit specifying the separative site(s) between adjacent clusters. Since the borders of a cluster are generally sparse comparing to the dense regions(s) of the same cluster, LCL must be dependent on UCL to determine the splitting site(s) dynamically, based on the condensation of patterns within the densely-populated region(s) of the cluster. More specifically,

$$\mathrm{LCL} = \mathrm{t}_{tol} * \mathrm{UCL}, \qquad \mathrm{t}_{tol} \in (0, 1) \tag{8}$$

where t_{tol} is a tolerance threshold that controls the sensitivity in detecting shifts in SCC histogram.

Consider the 2-dimensional rule R shown is figure 9(c) which covers three adjacent clusters. Note that the darker the colour of a cluster the more dense that cluster is. Additionally, figures 9(a) and 9(b) depict the SCC charts that correspond to the horizontal and vertical axis, respectively. Clearly, rule R does not deliver the quality characteristics discussed earlier since the region that describes is not homogeneous because there exist three distinct clusters as well as some very sparse regions. It can be observed from the x-axis SCC that there are three and well separated regions within R along x-axis, while in contrast in y-axis only a single region can be detected. In our approach the repair operator is applied to each rule recursively, until no more splitting site(s) is/are detectable. In particular, for each rule R the repair operator proceeds dimension-by-dimension with a random order. The algorithm constructs a SCC for the dimension that is currently being examined. If no point (bin) falls below LCL then R is kept intact and the algorithm proceeds to the next dimension. However, if there exists at least one measurement below LCL, the bins in the current dimension are grouped into segments. Two adjacent bins are grouped into the same segment as long as the selectivity in both bins is either below LCL or over LCL. Intuitively, the original rule R is replaced by a set of new rules, which are formed by combining each of the generated segments with the other feature genes of R. Note that each newly generated rule undergoes the same repairing procedure described above. The application of the repair operator in rule R of figure 9(c) yields the set of rules which are depicted in figure 9(d). Clearly, this set of rules fulfils the quality characteristics discussed earlier.

9 Time Complexity

Let T be the number of generations, N the number of individuals, and m the largest number of rules per individual. Assuming that the patterns are equi-distributed among the rules and the level of noise is zero then m=1/t_{sparse}. The computational expensive part of our algorithm is the repairing of individuals which is part of the evaluation step. Our system employs a linear search mechanism to find the patterns lying within a d-dimensional rule. Since the repairing of a rule requires to scan the entire dataset at least once we can assume that the time required to repair a rule is of order $O(nd)$. Consequently, the repairing of an individual which contains m rules can be performed in $O(mnd)$ or $O(nd/t_s)$ time. Thus, in total the time complexity of our algorithm is $O(\frac{NT}{t_s}nd)$. Theoretically, a significant speed-up can be achieved by replacing the current linear search mechanism with a faster hyper-rectangular query mechanism, i.e range or k-d trees [4]. The time needed to report the patterns that lie in a hyper-rectangular region using a range-tree is $O(n\log^{d-1}n+k)$, where k denotes the number of reported patterns. Thus, the use of a range-tree query mechanism can make our algorithm working with $O(\frac{NT}{t_s}(n\log^{d-1}n+t_sn))$ time complexity.

10 Setting Parameters

The mutation ratio was set to 1.0, while the probability of mutating the boundaries of rules is 0.05. Each boundary undergoes either growing or seed mutation at random. The ratio of the crossover was 0.65. The selection strategy that is used in our experiments is a k-fold tournament selection with tournament size k=4. We also used a elitist replacement strategy where the best 4 individuals of each generation were copied unaltered to the next generation. The only stopping criterion used in the reported experiments is the maximum number of generations 200, while the population size is 50. It should be pointed out that in most cases the evolutionary algorithm converged after 25-30 generations. The parameters $t_{intersection}$ and $t_{density}$ for the generalization operator were both set to 0.7. The tolerance threshold t_{tol} for the repair operator was 0.12. The threshold t_{sparse} for eliminating sparse rules was 0.5%. Finally, the parameter t_{status} which controls the deactivation of the feature genes were 0.9.

11 Experimental Results

In this section we empirically evaluate our algorithm using synthetic datasets. In particular, we report results of the experiments that were conducted to assess:

Efficiency: Determine how the running time scales with:

- Size of the dataset.
- Dimensionality of the dataset.
- Dimensionality of the clusters.

Accuracy: Test if our algorithm recovers correctly known clusters embedded in some subspace of the original feature space.

The simulation were performed on a Pentium III 800MHz machine with 256MB of RAM running the Linux operating system. The synthetic datasets were produced using a modified version of the generator described in [1]. The domain of each dimension was set to [0, 100]. In the original data generator all clusters have reasonably similar number of points assigned to them. In our case the number of points in each cluster is specified by the user. In particular, each dataset consists of 10 clusters and the non-outlier points were distributed among them as follows: 20%, 15%, 15%, 10%, 10%, 10%, 8%, 6%, 4% and 2%. Since different clusters often share subsets of correlated dimensions, when generating cluster i+1, approximately half of its bounded dimensions are chosen among the bounded dimensions of cluster i. Outliers were uniformly scattered in the entire feature space while the level of noise was set to $F_{outlier}$ = 10% of the total number of points. For all the non-bounded dimensions of a cluster the points are uniformly distributed over the entire domain. On the other hand in the bounded dimensions the values of the points are drawn either from a normal or from a uniform distribution at random. In both cases the points are spread in the bounded dimensions within a small range i.e 10, in a way that no overlapping between clusters occurs. Note that all clusters were embedded in the same number of dimensions. Therefore, the hyper-volume of clusters is roughly the same and because the points are not equi-distributed among clusters, the density between them varies considerably.

• **Dataset size:** Figure 10 depicts experimental results regarding the scalability of our algorithm with respect to the size of the dataset. Each dataset had 50 dimensions and there were 10 clusters embedded in 5-dimensional subspaces. The level of noise was chosen to be $F_{outlier}$ = 10% of the dataset size. It can be observed from figure 10 that the running time scales linearly with the size of the dataset. This fact verifies the complexity formula given in section 9. Similar scalability behaviour was reported in the experimental section of CLIQUE, which is the fundamental algorithm for massive subspace clustering [2]. Theoretically, a considerable scale up can be achieved by replacing the current linear search mechanism employed by our system with a faster hyper-rectangular query mechanism, i.e range or k-d trees [4].

• **Dimensionality of the dataset:** Figure 11 depicts the scalability of our algorithm in respect to the dimensionality of the dataset. Each dataset had 50,000 points and there were 10 clusters embedded in 5 dimensional subspaces. Unlike CLIQUE whose scalabil-

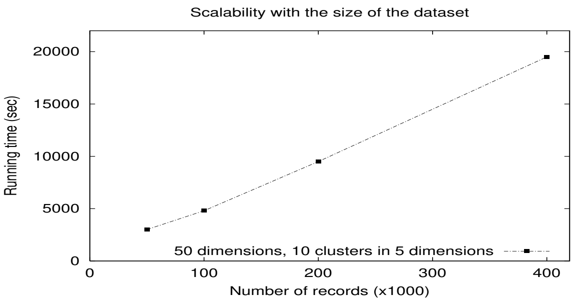

Figure 10: Scalability with the size of the dataset.

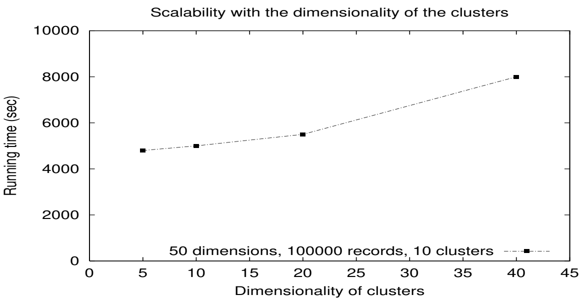

Figure 12: Scalability with the dimensionality of clusters.

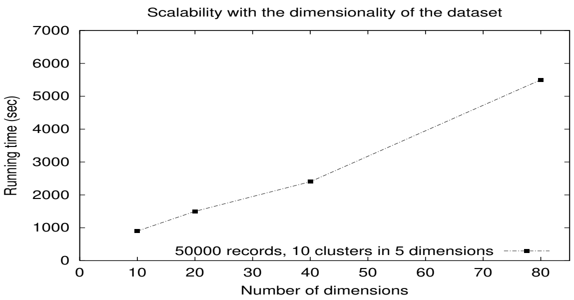

Figure 11: Scalability with the dimensionality of the dataset.

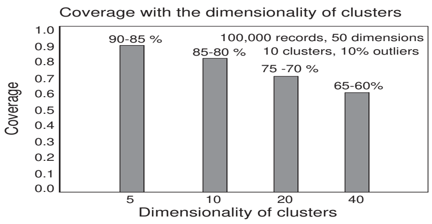

Figure 13: The dimensionality of clusters reduces accuracy.

ity is quadratic with the dimensionality of the dataset, in our case the curve exhibits a linear behaviour because of the linear nature of our query mechanism. The increase in running time with the dimensionality of the feature space reflects the additional number of dimensions that needed to be checked for repairing. Recall from section 8 that each dimension of a rule undergoes a repairing operation.

• **Dimensionality of clusters:** Figure 12 depicts the scalability of our algorithm with respect to the dimensionality of the clusters. The dataset had 100,000 points in a 50-dimensional feature space with noise $F_{outlier}$ = 10% of the total number of points. This experiment investigates how the running time scales as the dimensionality in which the clusters are embedded is increased from to 5 to 40. Apparently, the scalability curve exhibits nearly linear behaviour. The increase in the running time with the dimensionality of clusters reflects the high probability of splitting rules in possibly large number of bounded dimensions. In particular, as the dimensionality of the clusters increases, it is more likely that the evolutionary search (mutation, recombination and generalization) produces rules where splitting is necessary for a considerable large percentage of the bounded dimensions. Recall from section 8 that each time a rule is split along a dimension, all the newly generated rules must undergo repairing.

• **Accuracy:** In all the above experiments the original clusters were successfully detected and located within the enormous feature space. In some cases, especially when the dimensionality of the clusters was relatively high, few (1 or 2) extra clusters were reported. In essence, as the dimensionality of the clusters increases, more uncovered (by a rule) space becomes available and consequently some very large regions in the search space can have enough noisy points to be considered as homogeneous clusters instead of being eliminated as sparse clusters. However, the density of the points within these artifacts is considerably smaller comparing to the density within the real clusters. As expected, each uniform-type cluster was precisely captured by a singe hyper-rectangular rule with coverage that reached almost 95-100% of the total number of points assigned to that cluster. The fact that few points were missed is due to the quantization of each dimension into a number of bins, which in turn means that the boundaries of the rules are drawn from discrete rather than continuous domains. In case of a normal-type cluster the percentage of the cluster points that were missed by the algorithm in the boundaries of the distribution is higher. Typically, as the standard deviation of the normal distribution increases the number of missed points in the two tails of the distribution increases as well. Figure 13 depicts typical values of the fitness function (maximization of coverage) with respect to the dimensionality of clusters for various datasets consisting of 100.000 points in 50 dimensions with 10% outliers and 10 clusters. Not surprisingly, the performance of individuals decreases as the dimensionality of clusters increases since the points missed in each dimension are added up. This issue may be worthy of further research since a local tuning procedure can potentially improve the accuracy of the algorithm. Finally, it is important to mention that the non-random initialization scheme proposed in this paper yields individuals with coverage that in some cases reached 40% of the total number of points. The rest of the optimization was performed by the evolutionary algorithm itself.

12 Conclusions - Future work

In this paper we propose a novel evolutionary algorithm suitable for performing effective and efficient subspace clustering in very large and high dimensional databases. Each individual consists of a variable number of non-overlapping clustering rules, where each rule includes d intervals, one for each feature. The encoding scheme is non-binary as the values for the boundaries of the rules are drawn from discrete domains. Although the antecedent part of each rule in the genotype has fixed-length (d) only the bounded dimensions are reported to an end-user contributing to high comprehensibility of the clustering output. A task-specific genetic operator, namely the generalization operator, reduces the number of reported rules by replacing pairs of adjacent rules which satisfy some merging conditions with a single and more generic rule. We introduce a novel crossover operator which works by importing rules into individuals rather than exchanging rules between them. The functionality of the crossover operator is geared towards elimination of positional bias, minimization of distributional bias and generation of non-lethal offsprings. The growing-mutation performs local-tuning by increasing the size of existing rules at random, while the proposed seed-mutation enhances the exploratory power of our algorithm by creating randomly seeds(rules) within specific regions of the feature space. Our system employs an advanced repairing operator ensuring that only homogeneous rules are evolved. Finally, we propose a non-random initialization procedure which incorporates special problem-specific knowledge in order to overcome the problem of poor initialization.

Experimental results show that our algorithm scales almost linearly with the size and the dimensionality of the dataset as well as the dimensionality of the hidden clusters. Additionally, our algorithm was able to discover all the clusters which were embedded in lower dimensional subspaces, despite the fact that were no clusters existing in the original feature space. The algorithm missed only few points for uniform-type clusters but these points are statistically and practically insignificant. The percentage of missed points was higher for normal-type clusters.

In future work, we plan to incorporate a faster range query mechanism which will replace the linear query mechanism that is currently being employed. Additionally, ongoing research investigates the effectiveness of using various types of parallelism aiming to reduce the running time of our algorithm. Finally, the goal of accurately reporting the exact number of points for various data distributions can be another direction for future research.

Bibliography

[1] C.C. Aggarwal, C. Procopiuc, J. L. Wolf, P. S. Yu, and J.S. Park. Fast algorithms for projected clustering. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, pages 61–72, 1999.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, pages 94–105, 1998.

[3] T. Bäck, D. B. Fogel, and Z. Michalewicz (Eds.). *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, 2000.

[4] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry Algorithms and Applications*. Springer-Verlag, Berlin Heidelberg, 2000.

[5] A.A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Spinger-Verlag, 2002.

[6] L.O. Hall, I.B. Özyurt, and J. C. Bezdek. Clustering with a genetically optimized approach. *IEEE Trans. on Evolutionary Computation*, 3(2):103–112, 1999.

[7] L. C. Hamilton. *Modern Data Analyis: A First Course in Applied Statistics*. Brooks/Cole, 1990.

[8] J. Han and M. Kamer. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[9] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What is the nearest neighbor in high dimensional spaces? In *Proceedings of 26th International Conference on Very Large Data Bases*, pages 506–515, 2000.

[10] Jain, A.K., Dubes, R.C: . *Algorithms for Clustering Data*. Prentice-Hall, Englewood Cliffs, NJ, 1988.

[11] I.A. Sarafis, P.W. Trinder, and A.M.S Zalzala. Mining comprehensible clustering rules with an evolutionary algorithm. *To appear in Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003), Chicago-USA, LNCS- Springer-Verlag*, 2003.